Escola Politècnica Superior
de Castelldefels
UNIVERSITAT POLITÈCNICA DE CATALUNYA

TRABAJO FINAL DE CARRERA

TITULO: Emulación de RFID activo mediante la plataforma MICAz

AUTOR: Tomás García Sotelo

DIRECTOR: Carlos Gómez Montenegro

FECHA: 24 de febrero de 2006

Titulo: Emulación de RFID activo mediante la plataforma MICAz

Autor: Tomás García Sotelo

Director: Carlos Gómez Montenegro

Fecha: 24 de febrero de 2006

Resumen

Este trabajo de final de carrera está basado en la tecnología inalámbrica RFID. Ésta tecnología, pretende desbancar a la añeja tecnología basada en códigos de barras.

En el siguiente informe, se estudia las ventajas y desventajas de esta innovadora tecnología, y se pretende informar al lector del amplio abanico de utilidades que establece esta tecnología en el mundo real.

Aparte de realizar un estudio de los dispositivos RFID, se han creado dos aplicaciones como prueba de concepto del uso de estos dispositivos en un entorno real. Las aplicaciones que se han implementado y validado son:

- La primera aplicación, Control de acceso, se caracteriza por ser una aplicación que emula un sistema RFID activo. Dicha aplicación tiene como función, la validación de los usuarios que acceden a un área restringida, mediante la transmisión de información de un dispositivo RFID adherido al usuario.

- La segunda aplicación, Autoescáner, es una evolución de la anterior y está orientada a la recogida de datos de diversos dispositivos (caja registradora de un supermercado o inventario). En este caso, el elemento controlador debe recibir datos de varios dispositivos a la vez con los problemas que esto supone.

Finalmente, se realiza el validado de las aplicaciones antes mentadas, y se comprueba mediante experimentos la utilidad y ventajas que la tecnología RFID puede tener en aplicaciones cotidianas.

Title: Emulación de RFID activo mediante la plataforma MICAz

Author: Tomás García Sotelo

Director: Carlos Gómez Montenegro

Date: Febrary, 24th 2006

Overview

This work of end of career is based on the wireless technology RFID. This one technology tries to go bust to the old technology based on codes of bars.

In the following report, there are studied the advantages and disadvantages of this innovative technology, and tries to inform the reader of the wide fan of utilities that establishes this technology in the real world.

Apart from realizing a study of the devices RFID, two applications have been created as test of concept of the use of these devices in a real environment. The applications that have been implemented and validated are:

- The first application, Control of access, is characterized for being an application that emulates a RFID active system. The above mentioned application has as function, the validation of the users who accede to a restricted area, through the transmission of information of a RFID device adhered to the user.

- The second application, Autoscanner, is an evolution of the previous one and is orientated to the collection of data of diverse devices (register cash of a supermarket or inventory). In this case, the control element must receive information of several devices simultaneously with the problems that this supposes.

Finally, the validation of the applications is realized before mentioned, and there are verified through experiments the utility and advantages that the RFID technology can have in daily applications.

AGRADECIMIENTOS

En este apartado quisiera agradecer a toda la gente que con su apoyo, dedicación y esfuerzo han ayudado a la realización de este trabajo.

Primero de todo, quisiera dar las gracias a mis padres por su apoyo y cariño mostrado toda la vida, sin su aportación hace unos años este trabajo habría sido del todo imposible.

En especial quisiera dar las gracias a mi hermano Juan García Sotelo, por su aportación a lo largo de toda la carrera tanto emocional como económica. Él ha sido un gran ejemplo para mí y gracias a él, hoy soy lo que soy en la vida.

Agradecer a mi director de trabajo Carlos Gómez Montenegro por su paciencia y esfuerzo mostrado, sin sus aportaciones este trabajo no hubiera sido el mismo.

He recibido una valiosa colaboración y ayuda desinteresada de Pere Salvatella, ingeniero superior de Telecomunicaciones y becario de Campus Nord, que resolvió las fatídicas dudas del presente trabajo.

Agradecer a mis amigos de la EPSC: Mario Depra, Omar Alcorcón, Dídac Madaleta, Fernando Sopas, Eloy Encias, Carlos Truchero, Alex Pajaro entre otros, por su ayuda y apoyo en lo bueno y en lo malo hasta que la muerte nos separe. De todo corazón decirles que muchas gracias por todo.

También agradecer a Jose Vera su colaboración y ayuda desinteresada en el in trepidante mundo de C# y sobretodo las aportaciones a la hora de administrar las bases de datos.

También agradecer a mi familia, amigos y novia, su comprensión, apoyo moral y cariño recibido.

Finalmente agradecer al tribunal la lectura de este trabajo de final de carrera.

ÍNDICE

INTRODUCCIÓN 1

CAPITULO 1. TECNOLOGÍA RFID 3

1.1. Definición 3

1.2. Historia 3
1.2.1. Componentes antecesores a la tecnología RFID 4

1.3. Sistemas RFID 5
1.3.1. Lector RFID 5
1.3.2. Etiquetas RFID 5

1.4. Aplicaciones basadas en la tecnología RFID 7
1.4.1. Aplicaciones actuales 7
1.4.2. Implantes humanos 8
1.4.3. Aplicaciones potenciales 9

1.5. Reglamentación y estandarización. 10

CAPITULO 2. ESTANDAR IEEE 802.15.4 (ZIGBEE) 11

2.1. Introducción 11

2.2. Tipos de topologías 12

2.3. Capa física del 802.15.4 14

2.4. Capa MAC del 802.15.4 15
2.4.1. Funcionamiento 16
2.4.2. Mecanismo de robustez 17
2.4.3. Implementación usada en nuestro trabajo 18

CAPITULO 3. ENTORNO DE DESARROLLO 19

3.1. Mote – Kit2400 19
3.1.1. Mote MICAz, MPR2400CA 20
3.1.2. Programador MIB510 21
3.1.3. Programador MIB600 23
3.1.4. Sensor MTS300CA 23
3.1.5. Sensor MTS310CA 24

3.2. TinyOS 24

3.3. Introducción al nesC 26

3.4. SerialForwarder 27

3.5. Introducción a C# 28

3.6. Introducción a Bases de datos ACCESS 29

CAPITULO 4. APLICACIONES DESARROLLADAS 31

4.1. Introducción 31

4.2. Aplicación "Control de acceso" 31
4.2.1. Descripción general de la aplicación 31
4.2.2. Aplicación mote MICAz parte usuario 33
4.2.3. Aplicación mote MICAz parte validador 35
4.2.4. Aplicación PC 37

4.3. Aplicación "Autoescáner" 39
4.3.1. Descripción general de la aplicación 39
4.3.2. Aplicación mote MICAz parte producto 40
4.3.3. Aplicación mote MICAz parte interrogador 41
4.3.4. Aplicación PC 42

CAPITULO 5. PRUEBAS Y RESULTADOS 44

5.1. Aplicación control de acceso 44
5.1.1. Escenario y configuración 44
5.1.2. Tiempo de validación 45

5.2. Aplicación Autoescaner 49
5.2.1. Escenario y configuración 49
5.2.2. Tiempo de recogida de información (configuración por defecto) 50
5.2.3. Probabilidad de perdida de información (ninguna retransmisión) 51
5.2.4. Tiempo de recogida de información (con retransmisiones) 52
5.2.5. Probabilidad de perdida de información (con retransmisiones) 53

CAPITULO 6. CONCLUSIONES Y LINEAS FUTURAS 56

6.1. Conclusiones 56

6.2. Líneas Futuras 57

IMPLICACIONES MEDIOMENTALES 58

BIBLIOGRAFIA 59

ANEXO A. TINYOS 61

1. Instalación TinyOs para Windows 61

2. Directorio TinyOS 62

3. Compilación de aplicaciones en TinyOS 64
3.1. Placa programadora puerto serie MIB510CA 65
3.2. Placa programadora Ethernet MIB600CA 65

ANEXO B. MANUAL DE SERIALFORWARDER 67

ANEXO C. CONFIGURACIÓN MIB600CA 69

ANEXO D: LIBRERÍA DE CONSTANTES REFERENTES A CHIP DE COMUNICACIONES CC2420 72

1. CC2420const.h 73

2. CC2420ControlM.nc 76

3. CC2420RadioM 82

ANEXO E. CÓDIGO CONTROL SEGURIDAD 91

1. Código nesC para motes MICAz 91
1.1. Archivo configuración nodo controlador 91
1.2. Archivo módulo nodo controlador 91
1.3. Archivo configuración nodo usuario 95
1.4. Archivo módulo nodo usuario 96

2. Código C# para aplicación PC 97
2.1. Programa principal 97
2.2. Libreria conexión 104
2.3. Biblioteca paquete de datos 105

ANEXO F. CÓDIGO AUTOESCANER 107

1. Código nesC para motes MICAz 107
1.1. Archivo configuración nodo validador 107
1.2. Archivo módulo nodo validador 107
1.3. Archivo configuración nodo producto 111
1.4. Archivo módulo nodo producto 111

2. Código C# para aplicación PC 113
2.1. Programa principal 113
2.2. Biblioteca paquete datos 121

ANEXO G. RESULTADOS OBTENIDOS 122
1. Resultados obtenidos sin incluir retransmisiones. 122
2. Resultados obtenidos con 1 retransmisión como máximo. 123
3. Resultados obtenidos con 2 retransmisión como máximo. 124
4. Resultados obtenidos con 3 retransmisión como máximo. 125
5. Resultados obtenidos con 4 retransmisión como máximo. 126
6. Resultados obtenidos con 5 retransmisión como máximo. 127
7. Resultados obtenidos con 6 retransmisión como máximo. 128
8. Resultados obtenidos con 7 retransmisión como máximo. 129
9. Resultados obtenidos con 8 retransmisión como máximo. 130
10. Resultados obtenidos con 9 retransmisión como máximo. 131

ANEXO H. ESTRUCTURAS DE TRAMAS DEL IEEE 802.15.4 132

ANEXO I. CARACTERIZACIÓN ESTADÍSTICA DE TIEMPO DE RECOGIDA DE DATOS. 135

INTRODUCCIÓN

Hoy en día, el mundo de la tecnología está avanzando a gran escala, llegando a decir que estamos en la "*era de la tecnología"*. Dicha evolución nos motiva especialmente en la investigación de estas nuevas tecnologías, como es la tecnología RFID que se estudia en este trabajo de final de carrera. Gracias a la evolución en prestaciones y tamaño que nos ofrece la electrónica actualmente, disponemos de sensores con comunicación inalámbrica, capacidad de procesamiento y autonomía propia, con los cuales se nos abre un abanico de oportunidades para diseñar y crear todo tipo de aplicaciones y sistemas capaces de facilitar el trabajo a los seres humanos y reducir los costes de éstos, como se verá posteriormente.

El objetivo de este trabajo es desarrollar aplicaciones de RFID mediante dispositivos sensores, de la casa XBOW, que implementan el novedoso protocolo IEEE 802.15.4, comercialmente llamado ZigBee. Dichos sensores utilizan un sistema operativo para sistemas empotrados de gran aceptación por parte de la industria y de libre distribución llamado TinyOS. Después de dicha evaluación se creará dos aplicaciones para observar el funcionamiento de los dispositivos inalámbricos en aplicaciones comerciales.
La primera aplicación que se trata es la de control de seguridad, donde intervendrán un dispositivo validador y un dispositivo adherido al usuario. El validador es el encargado de recoger un número de identificación. Éste será transmitido por el dispositivo que hipotéticamente lleve el usuario que quiera acceder a una determinada zona. El dispositivo validador enviará a la aplicación el número de identificación del usuario, y ésta contrastará la identificación con una base de datos para mostrar por pantalla los datos de dicho usuario.
La segunda aplicación consiste en la interface de un cajero de un supermercado, es decir, dispondremos de un dispositivo inalámbrico "interrogador" en la caja registradora, el cual interroga a los dispositivos adheridos a los productos cuando estos se acerquen a la caja registradora. El dispositivo interrogador enviará a la aplicación del PC los números de identificación de los productos para poder contabilizarlos. En esta segunda aplicación, cabe destacar que, varios sensores tendrán que transmitir a la vez con los problemas que conlleva debido a que el medio es compartido.

El trabajo está estructurado en seis capítulos. El capítulo inicial introducirá al lector en los fundamentos de la tecnología RFID, así como su historia, funcionamiento y sistemas desarrollados mediante esta tecnología. El segundo capitulo se hará un breve introducción al estándar IEEE 802.15.4 ya que es la tecnología radio que utiliza los sensores y es bastante novedosa. El capitulo tercero definirá el entorno de desarrollo utilizado para realizar el estudio y las aplicaciones. El cuarto apartado describirá las aplicaciones desarrolladas por el estudiante y su funcionamiento. En el quinto capítulo se podrán observar los experimentos y resultados obtenidos que validarán las aplicaciones desarrolladas. En el último capitulo se detallarán las conclusiones obtenidas de dicho trabajo como a su vez, las líneas futuras que se proponen a partir de él. Finalmente, se comentarán las implicaciones medioambientales del TFC.

CAPITULO 1. TECNOLOGÍA RFID

En este capitulo inicial describimos brevemente los aspectos mas destacables de esta nueva tecnología denominada RFID. También haremos una especial mención a las nuevas aplicaciones desarrolladas a través de esta tecnología.

1.1. Definición

RFID responde a las siglas *de **R**adio **F**requency **ID**entification*, en español Identificación por radiofrecuencia. Consiste en un método de almacenamiento y recuperación de datos remoto que usa etiquetas (*RFID tag* en inglés). Dichas etiquetas o tags son de tamaño tan reducido (como una pegatina) que puede ser adherido a una mercancía, producto, animal o persona (Fig. 1.1). Las etiquetas RFID incorporan antenas para permitir la comunicación entre el emisor y receptor mediante radiofrecuencia.

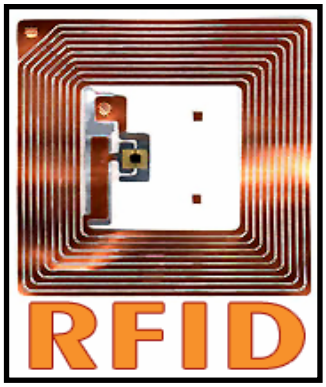


Fig. 1.1 Etiqueta RFID.

La tecnología RFID ha comenzado a desplazar al código de barras que aparece en las etiquetas que van pegadas a los productos de cualquier establecimiento, debido a las desventajas que presentan éstos respecto a RFID.
En la actualidad, diversas empresas (p.ej: *Wal-Mart)* ya han instalado esta tecnología en gran parte de sus productos, debido a que RFID les permite rastrear cualquier producto sin necesidad de movilizar a ningún empleado o hacer inventarios en tiempo récord, entre otras aplicaciones versátiles.

1.2. Historia

La historia de RFID se remonta a finales de los años 30 en la segunda guerra mundial, donde la tecnología RFID fue concebida por el gobierno de la Gran Bretaña para identificar de manera efectiva a sus propios aviones cuando estos

eran detectados por sus sistemas de radar. De esa manera, se redujo significativamente la posibilidad de confundirlos con la flota aérea enemiga (véase [1]).

A través de muchos años de innovación, la tecnología RFID ha llegado a un punto donde los costos y el tamaño de los dispositivos de RFID (etiquetas, lectores, antenas) hacen posible que pueda utilizarse en un sinnúmero de productos, desde contenedores de barcos hasta animales o seres humanos. La miniaturización y automatización de los procesos de fabricación de tecnología RFID se han traducido en una reducción en los costos de las etiquetas y han logrado que la tecnología esté prácticamente al alcance de cualquier organización. Actualmente, compañías innovadoras como Inkode de Estados Unidos están llevando RFID a nuevas fronteras tecnológicas.

1.2.1. Componentes antecesores a la tecnología RFID

El gran antecesor de la tecnología RFID es el código de barras. Esta es la tecnología más conocida y extendida para la identificación de productos desde hace muchos años. Dicho código es la representación de una determinada información mediante un conjunto de líneas paralelas verticales de diferente grosor y espaciado como se puede observar en la Fig. 1.2.

Fig. 1.2 Código de barras EAN13.

El código de barras más utilizado y estandarizado a nivel mundial es el EAN13, adoptado por más de 100 países y cerca de un millón de empresas en el 2003. El EAN13 está constituido por 13 dígitos. Este conjunto de dígitos está dividido en cuatro grupos, cada grupo representa un significado diferente (véase [2]).

Los códigos de barra se siguen utilizando en la actualidad, pero cada vez más la tecnología RFID les está recortando campo de acción debido a las limitaciones de los primeros, siendo las más destacables:

- Reducida capacidad de almacenamiento de los códigos de barra (un único numero) frente a la gran capacidad de los RFID debido a que la mayoría llevan incorporados un dispositivo de memoria.
- Los códigos de barras requieren de personal humano para poder escanear dicho código. En cambio, los RFID no es necesaria la presencia de un operario para poder ser escaneado.
- Para poder leer de un código de barras se necesita tener visión directa entre el aparato escaneador y el código. En cambio, en los RFID no es

necesario visión directa entre el lector y la etiqueta ya que su comunicación es por radiofrecuencia.
- Los códigos de barras son de fácil deterioro, pudiéndose llegar a erróneas lecturas por dicha causa. Las etiquetas RFID son más robustas al estar protegidas por materiales plásticos.

Como hemos visto, la tecnología RFID presenta un sinfín de ventajas respecto al código de barras, por ello cada vez mas se esta optando por esta tecnología. La única desventaja de las etiquetas RFID es el costo, pero éste está en continuo descenso, debido a que, cada vez mas, se esta optando por está opción.

1.3. Sistemas RFID

Un sistema RFID está formado por varios componentes: etiquetas, lectores de etiquetas y estaciones de programación de etiquetas. El propósito de un sistema RFID es permitir que se puedan transmitir datos mediante un dispositivo portátil, llamado etiqueta, que es leída por un lector RFID y procesada según las necesidades de una aplicación determinada. Los datos transmitidos por la etiqueta pueden proporcionar información sobre la identificación o localización, o datos específicos sobre el producto marcado con la etiqueta, como por ejemplo precio, color, fecha de compra, etc.
En los siguientes subapartados describimos brevemente los componentes que forman un sistema RFID.

1.3.1. Lector RFID

El lector RFID consiste en un dispositivo que emite señales de radio a una frecuencia predeterminada, con el fin de interrogar a la etiqueta RFID y obtener su número de identificación. Cuando el lector obtiene una respuesta de la etiqueta RFID, convierte la señal de radio frecuencia en un código numérico que puede ser transmitido a otros dispositivos o sistemas. Los lectores RFID pueden variar en tamaño, funcionalidad y costo, este último fluctuando desde unos cuantos cientos de euros hasta miles de euros, dependiendo de su nivel de complejidad y sofisticación.

1.3.2. Etiquetas RFID

La etiqueta de RFID (RFID tag en inglés) también conocida como *transponedor*, es un dispositivo que puede ser muy pequeño (40 micrómetros) y capaz de almacenar un número de identificación (ya sea único o genérico), el cual puede ser leído a través de una o múltiples antenas que captan radio-señales a una frecuencia predeterminada. Por medio del lector, las etiquetas son capaces de responder cuando son bombardeadas (*o en términos técnicos, interrogadas o leídas*).
El precio de las etiquetas RFID puede variar significativamente dependiendo de su clasificación, nivel de sofisticación, volumen de compra, etc. Se pueden

encontrar etiquetas RFID por unos cuantos centavos de euros y otras cuyo costo puede exceder los 100 euros por unidad.

Las etiquetas RFID pueden ser pasivas, semi-pasivas o activas:

Las etiquetas **RFID pasivas** no contienen fuente de alimentación propia, obtienen la corriente eléctrica induciéndola de la señal de radiofrecuencia del lector cuando está escaneando dicha etiqueta. Gracias a esta energía inducida las etiquetas son capaces de transmitir una respuesta al lector. Debido a las preocupaciones por la energía y el coste, la respuesta de una etiqueta pasiva RFID es necesariamente breve, normalmente un número de identificación de etiqueta. La falta de una fuente de alimentación propia hace que el dispositivo pueda ser bastante pequeño: existen productos disponibles de forma comercial que pueden ser insertados bajo la piel (0.4 milímetros × 0.4 milímetros). Las etiquetas pasivas, en la práctica tienen distancias de lectura que varían entre unos 10 milímetros hasta cerca de 6 metros dependiendo del tamaño de la antena de la etiqueta y de la potencia y frecuencia en la que opera el lector.

Las etiquetas **RFID semi-pasivas** son muy similares a las pasivas, salvo que incorporan además una pequeña batería. Esta batería permite a la etiqueta estar constantemente alimentada, eliminando la necesidad de diseñar una antena para recoger potencia de una señal entrante. Las etiquetas RFID semi-pasivas responden más rápidamente comparadas con las etiquetas pasivas.

Las etiquetas **RFID activas**, por otra parte, deben tener una fuente de energía, y pueden tener rangos mayores de cobertura y memorias más grandes que las etiquetas pasivas, así como la capacidad de poder almacenar información adicional enviada por el transmisor-receptor. Actualmente, las etiquetas activas más pequeñas tienen un tamaño aproximado de una moneda. Muchas etiquetas activas tienen rangos prácticos de diez metros, y una duración de batería de hasta varios años.

Como las etiquetas pasivas son mucho más baratas de fabricar y no necesitan batería, la gran mayoría de las etiquetas RFID existentes son del tipo pasivo. En fecha de 2004, las etiquetas tienen un precio desde 0,40€, en grandes pedidos. Los analistas de las compañías independientes de investigación como *Gartner and Forrester Research* convienen en que un nivel de precio de menos de 0,10€ (con un volumen de producción de 1.000 millones de unidades) requieren que transcurran unos 6 u 8 años, lo que limita los planes a corto plazo para una adopción extensa de las etiquetas RFID pasivas. Otros analistas creen que esos precios serían alcanzables dentro de 10-15 años.

A pesar de las ventajas en cuanto al coste de las etiquetas pasivas con respecto a las activas son significativas, otros factores incluyendo exactitud, funcionamiento en ciertos ambientes como cerca del agua o metal, y confiabilidad hacen que el uso de etiquetas activas sea muy común hoy en día.
Por otra parte las etiquetas se pueden clasificar según el rango radiofrecuencial que utilicen para recibir y transmitir datos, como vemos a continuación:

- Etiquetas de baja frecuencia que utilizan un rango de frecuencia comprendida entre 125KHz y 134,2 KHz.
- Etiquetas de alta frecuencia que transmiten a 13,56 MHz.
- Etiquetas UHF o frecuencia ultra elevada que utilizan un rango de frecuencia entre 868 a 956 MHz. Las etiquetas UHF no pueden ser utilizadas de forma global porque no existen regulaciones globales para su uso.
- Etiquetas de microondas, que son las que utilizamos en este trabajo, que transmiten en un rango de 2.45 GHz.

1.4. Aplicaciones basadas en la tecnología RFID

En este aparatado intentaremos introducir al lector en las nuevas aplicaciones existentes a nivel mundial que aplican la tecnología RFID.

1.4.1. Aplicaciones actuales

Las etiquetas RFID de baja frecuencia se utilizan comúnmente para la identificación de animales, seguimiento de productos, y como llave de automóviles con sistema antirrobo. En ocasiones, se insertan en pequeños chips en mascotas, para que puedan ser devueltas a su dueño en caso de pérdida. En los Estados Unidos se utilizan dos frecuencias para RFID: 125 KHz (el estándar original) y 134,5 KHz (el estándar internacional).
Las etiquetas RFID de alta frecuencia se utilizan en bibliotecas y seguimiento de libros, seguimiento de mercancías, control de acceso en edificios, seguimiento de equipaje en aerolíneas y seguimiento de artículos de ropa.
Las etiquetas RFID de UHF se utilizan comúnmente de forma comercial en seguimiento de envases, y seguimiento de camiones y remolques en envíos.
Las etiquetas RFID de microondas se utilizan en el control de acceso en vehículos de gama alta.
Algunas autopistas, como por ejemplo la *FasTrak* de California, el sistema *I-Pass* de Illinois y la *Philippines South Luzon Expressway E-Pass* utilizan etiquetas RFID para recaudación con peaje electrónico. Las tarjetas son leídas mientras los vehículos pasan; la información se utiliza para descontar el peaje de una cuenta prepago. El sistema ayuda a disminuir el tráfico causado por las cabinas de peaje.
Sensores como los sísmicos pueden ser leídos empleando transmisores-receptores RFID, simplificando enormemente la recolección de datos remotos.
En enero de 2003, Michelín anunció que había comenzado a probar transmisores-receptores RFID insertados en neumáticos. Después de un período de prueba estimado de 18 meses, el fabricante ofrecerá neumáticos con RFID a los fabricantes de automóviles. Su principal objetivo es el seguimiento de neumáticos en cumplimiento con la *United States Transportation, Recall, Enhancement, Accountability and Documentation Act* (TREAD Act).
Las tarjetas con chips RFID integrados se usan ampliamente como dinero electrónico, como por ejemplo la tarjeta *Octopus* en Hong-Kong y en los Países Bajos como forma de pago en transporte público y ventas menores.

Comenzando con el modelo de 2004, está disponible una "llave inteligente" como opción en el *Toyota Prius* y algunos modelos de Lexus. La llave emplea un circuito de RFID activo que permite que el automóvil reconozca la presencia de la llave a un metro del sensor. El conductor puede abrir las puertas y arrancar el automóvil mientras la llave sigue estando en la cartera o en el bolsillo.
En agosto de 2004, el Departamento de Rehabilitación y Corrección de Ohio (ODRH) aprobó un contrato de 415.000 dólares para ensayar la tecnología de seguimiento con *Alanco Technologies*. Los internos tienen unos transmisores del tamaño de un reloj de muñeca que pueden detectar si los presos han estado intentando quitárselas y enviar una alarma a los ordenadores de la prisión. Este proyecto no es el primero que trabaja en el desarrollo de chips de seguimiento en prisiones estadounidenses. Instalaciones en Michigan, California e Illinois emplean ya esta tecnología.

1.4.2. Implantes humanos

Los chips RFID implantables, diseñados originalmente para el etiquetado de animales se está utilizando y se está contemplando también para los seres humanos. *Applied Digital Solutions* propone su chip "*unique under-the-skin format*" (formato bajo-la-piel único) como solución a la usurpación de la identidad, al acceso seguro a un edificio, al acceso a un ordenador, al almacenamiento de expedientes médicos, a iniciativas de anti-secuestro y a una variedad de aplicaciones. Combinado con los sensores para supervisar diversas funciones del cuerpo, el dispositivo *Digital Angel* podría proporcionar supervisión de los pacientes. El *Baja Beach Club* en Barcelona utiliza un Verichip implantable para identificar a sus clientes VIP, que lo utilizan para pagar las bebidas (véase [3]). El departamento de policía de Ciudad de México ha implantado el Verichip a unos 170 de sus oficiales de policía, para permitir el acceso a las bases de datos de la policía y para poder seguirlos en caso de ser secuestrados.
Amal Graafstra, un empresario natural del estado de Washington, tenía un chip RFID implantado en su mano izquierda a principios de 2005. El chip medía 12 mm de largo por 2 milímetros de diámetro y tenía un radio de acción para su lectura de dos pulgadas (50 milímetros). La implantación fue realizada por un cirujano plástico, aunque el nombre del doctor no fue desvelado.

Fig. 1.3 Mano izquierda de Amal Graafstra después de la operación de implante de la etiqueta RFID

1.4.3. Aplicaciones potenciales

Hoy en día se está trabajando en nuevas aplicaciones basadas en la tecnología RFID, a continuación haremos una breve descripciones de las principales tendencias que se están adoptando a nivel mundial.

1.4.3.1. Identificación de pacientes

En julio de 2004, la *Food and Drug Administration* (Administración de Comida y Medicamentos) hizo pública la decisión de comenzar un proceso de estudio que determinará si los hospitales pueden utilizar sistemas RFID para identificar a pacientes y/o para permitir el acceso por parte del personal relevante del hospital a los expedientes médicos. El uso de RFID para prevenir mezclas entre esperma y óvulos en las clínicas de fecundación in vitro también está siendo considerado (véase [4]). Además, la FDA aprobó recientemente los primeros chips RFID de EE.UU. que se pueden implantar en seres humanos. Los chips RFID de 134,2kHz, de VeriChip Corp., una subsidiaria de Applied Digital Solutions Inc., pueden incorporar información médica personal y podrían salvar vidas y limitar lesiones causadas por errores en tratamientos médicos, según la compañía. La aprobación por parte de la FDA fue divulgada durante una conferencia telefónica con los inversionistas. También se ha propuesto su aplicación en el hogar, para permitir, por ejemplo, que un frigorífico pueda conocer las fechas de caducidad de los alimentos que contiene, pero ha habido pocos avances más allá de simples prototipos.

1.4.3.2. Tráfico y posicionamiento

Otra aplicación propuesta es el uso de RFID para **señales de tráfico inteligentes** en la carretera (**Road Beacon System** o RBS) (véase [5]). Se basa en el uso de etiquetas RFID enterrados bajo el pavimento (radiobalizas) que son leídos por una unidad que lleva el vehículo (OBU, de *onboard unit*) que filtra las diversas señales de tráfico y las traduce a mensajes de voz o da una proyección virtual usando un HUD (*Heads-Up Display*). Su principal ventaja comparada con los sistemas basados en satélite es que las radiobalizas no necesitan de mapeado digital ya que proporcionan el símbolo de la señal de tráfico y la información de su posición por sí mismas. Las radiobalizas RFID también son útiles para complementar sistemas de posicionamiento de satélite en lugares como los túneles o interiores, o en el guiado de personas ciegas.

Fig. 1.4 Balizas de tráfico basadas en la tecnología RFID

1.5. Reglamentación y estandarización.

No hay ninguna corporación pública global que gobierne las frecuencias usadas para RFID. En principio, cada país puede fijar sus propias reglas. Las principales corporaciones que gobiernan la asignación de las frecuencias para RFID son:

- EE.UU.: FCC (*Federal Communications Commission*)
- Canadá: DOC (Departamento de la Comunicación)
- Europa: ERO, CEPT, ETSI y administraciones nacionales. Obsérvese que las administraciones nacionales tienen que ratificar el uso de una frecuencia específica antes de que pueda ser utilizada en ese país.
- Japón: MPHPT (*Ministry of Public Management, Home Affairs, Post and Telecommunication*)
- China: Ministerio de la Industria de Información.
- Australia: Autoridad Australiana de la Comunicación (*Australian Communication Authority*)
- Nueva Zelanda: Ministerio de desarrollo económico de Nueva Zelanda (*New Zealand Ministry of Economic Development*

Las etiquetas RFID de baja frecuencia (LF: 125 - 134 KHz y 140 - 148.5 KHz) y de alta frecuencia (HF: 13.56 MHz) se pueden utilizar de forma global sin necesidad de licencia. La frecuencia ultraalta (UHF: 868 - 928 MHz) no puede ser utilizada de forma global, ya que no hay un único estándar global. En Norteamérica, la frecuencia ultraelevada se puede utilizar sin licencia para frecuencias entre 908 - 928 MHz, pero hay restricciones en la energía de transmisión. En Europa la frecuencia ultraelevada está bajo consideración para 865.6 - 867.6 MHz. Su uso es sin licencia sólo para el rango de 869.40 - 869.65 MHz, pero existen restricciones en la energía de transmisión. El estándar UHF norteamericano (908-928 MHz) no es aceptado en Francia ya que interfiere con sus bandas militares. En China y Japón no hay regulación para el uso de la frecuencia ultraelevada. Cada aplicación de frecuencia ultraelevada en estos países necesita de una licencia, que debe ser solicitada a las autoridades locales, y puede ser revocada. En Australia y Nueva Zelanda, el rango es de 918 - 926 MHz para uso sin licencia, pero hay restricciones en la energía de transmisión.

Existen regulaciones adicionales relacionadas con la salud y condiciones ambientales. Por ejemplo, en Europa, la regulación *Waste Electrical and Electronic Equipment* ("Equipos eléctricos y electrónicos inútiles"), no permite que se desechen las etiquetas RFID. Esto significa que las etiquetas RFID que estén en cajas de cartón deber de ser quitadas antes de deshacerse de ellas.

Algunos estándares que se han desarrollado en relación a la tecnología RFID son:

- ISO 10536
- ISO 14443
- ISO 15693
- ISO 18000
- EPCGlobal - éste es el estándar que tiene una mayor probabilidad de convertirse en la base de un estándar a nivel mundial.

CAPITULO 2. ESTANDAR IEEE 802.15.4 (ZigBee)

En este capítulo realizaremos una breve introducción al novedoso estándar IEEE 802.15.4 más conocido como ZigBee. Comentaremos las características más relevantes como los distintos modos de funcionamiento que dispone y haremos un brevemente inciso en el tipo de transceptor utilizado en nuestro trabajo.

2.1. Introducción

ZigBee es una alianza, sin ánimo de lucro, de más de 100 empresas, la mayoría de ellas fabricantes de semiconductores, con el objetivo de promover el desarrollo e implantación de una tecnología inalámbrica de bajo coste.
Destacan empresas como Invensys, Mitsubishi, Honeywell, Philips y Motorola que trabajan para crear un sistema estándar de comunicaciones, vía radio y bidireccional, para usarlo dentro de dispositivos de domótica, automatización de edificios (inmótica), control industrial, periféricos de PC, juguetería, sensores médicos. Los miembros de esta alianza justifican el desarrollo de este estándar para cubrir el vacío que se produce por debajo del Bluetooth como se explicara a continuación.

El 802.15.4 surgió debido a la escasez de estándares inalámbricos de baja tasa para redes de sensores. Los estándares disponibles en el mercado (wi-fi, wi-max, bluetooth) estaban orientados hacia aplicaciones con requerimientos de alto ancho de banda como puede ser redes domésticas, videoconferencia, etc.
El inconveniente que surgía al utilizar cualquiera de los estándares antes mencionados era un gran consumo de energía y un gran ancho de banda utilizado frente a la baja tasa y bajos requerimientos de energía necesaria para redes sensoras. A continuación se puede observar una pequeña comparativa entre ZigBee y otros estándares como son Bluetooth y wi-fi:

Tabla 2.1. Comparativa de estándares inalámbricos

Estándares	Ancho de banda	Consumo de potencia	Ventajas	Aplicaciones
Wi-fi	Hasta 54 Mbps	400 mA transmitiendo 20 mA en reposo	Gran ancho de banda	Navegar por Internet, redes de ordenadores, transferencia de ficheros
Bluetooth	1 Mbps	40 mA transmitiendo, 0.2mA en reposo	Interoperatividad, sustituto del cable	Wireless USB, móviles, informática doméstica
ZigBee	250 Kbps	30 mA transmitiendo, 3mA en reposo	Batería de larga duración, bajo coste	Control remoto, productos dependientes de la batería, sensores , juguetería

En un principio, cada fabricante de nodos sensores ha optado por utilizar soluciones propietarias dada la presión ejercida por el mercado, lo que trajo problemas de interoperabilidad entre los diversos fabricantes.

La industria entiende que hace falta un nuevo estándar que aúne autonomía, envío de datos de baja capacidad (Kbps) y un bajo coste, es con esto con lo que nace el estándar 802.15.4 comercialmente llamado ZigBee. Se ha convenido llamar a esta clase de redes LR-WPAN (Low Rate Wireless Personal Area Network), dado sus bajas tasas de transmisión y su corto alcance.

2.2. Tipos de topologías

Dentro del estándar 802.15.4 se pueden definir dos tipos de nodos según su funcionamiento y la topología utilizada en la red sensorial, a continuación definimos los dos tipos de nodos:

- *FFD (Full Function Device)* en español dispositivo de funciones completas, son dispositivos capaces de organizar y coordinar el acceso al medio de otros dispositivos de la misma red. Estos dispositivos se suelen utilizar en redes donde se necesita un nodo central, como puede ser en redes con topología en estrella, y suelen requerir un consumo de energía superior a otros nodos por lo que se suele conectar a la red eléctrica.

- *RFD (Reduced Function Device):* en español dispositivos con funcionalidad reducida, son dispositivos con un bajo consumo de energía y de un bajo coste y simplicidad, estos dispositivos se suelen utilizar en cualquier tipo de red.

Dependiendo de la aplicación que se esté desarrollando, se pueden configurar tres tipos de topología: en estrella (Star), peer to peer y cluster-tree, a continuación definiremos cada una:

En la topología en estrella la comunicación se establece entre los nodos (RFD o FFD) y el nodo central llamado PAN coordinator. Una vez se conectan los nodos en una red en estrella, se elige cual va a ser el nodo coordinador de dicha red, el nodo elegido proporciona un identificador de red que no puede ser igual al identificador de otra red dentro del radio de acción de este nodo coordinador (área de cobertura o huella). El nodo coordinador será el que autorice la transmisión a los demás nodos debido a que será este el controlador de la red.

Las aplicaciones más comunes que utilizan este tipo de topología son la conexión entre el ordenador personal y los periféricos, domótica o juguetes.

A continuación se puede observar la estructura que tendría una red con esta topología.

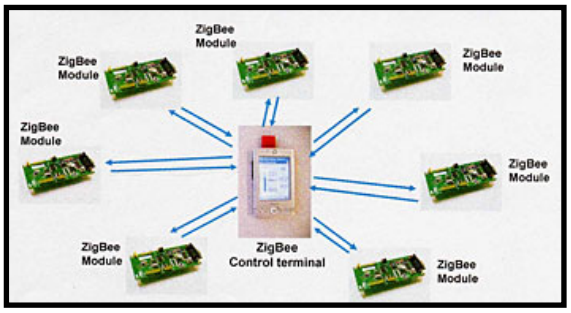


Fig. 2.1. Topología en estrella

En la topología Peer to peer o Mesh, también existe el papel del dispositivo PAN cordinator pero no tiene las mismas funciones revelantes, en contraste con la topología en estrella, cualquier dispositivo puede comunicarse con cualquier otro mientras ambos estén en la misma área de cobertura o utilizando otros nodos para llegar al destino (topología Mesh) debido a que tienen la misma prioridad a la hora de transmitir. Este tipo de topología es utilizada en redes ad hoc y se implementa en distintas aplicaciones como pueden ser control industrial, redes de sensores inalámbricas o aplicaciones de inventario. Esta topología permite múltiples saltos entre el nodo origen y destino con lo que conlleva la utilización de protocolos de enrutamiento en este tipo de topología (protocolos AODV, DYMO o DSR (véase [6])). A continuación podemos observar la estructura que tendría una red con esta topología.

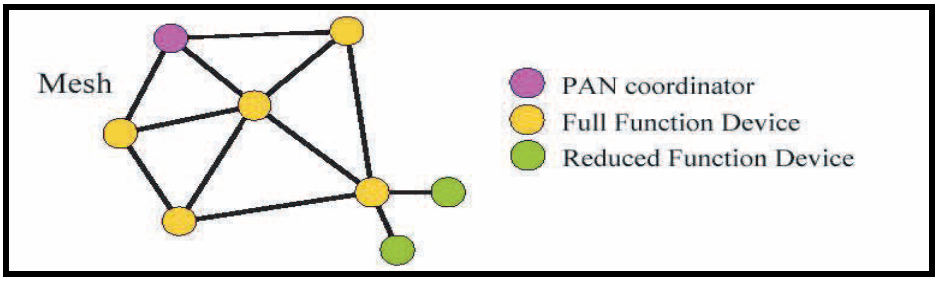


Fig. 2.2. Topología punto a punto o Mesh

La topología Cluster-tree es un caso especial de la topología peer to peer donde se conectan varios dispositivos FFD y RFD entre si, formando una jerarquía de árbol. En este tipo de topologías existen varios nodos coordinadores en una determinada zona y luego existe el papel del PAN cordinator que es el coordinador de toda la red que esta en un nivel superior como se puede observar en la siguiente figura:

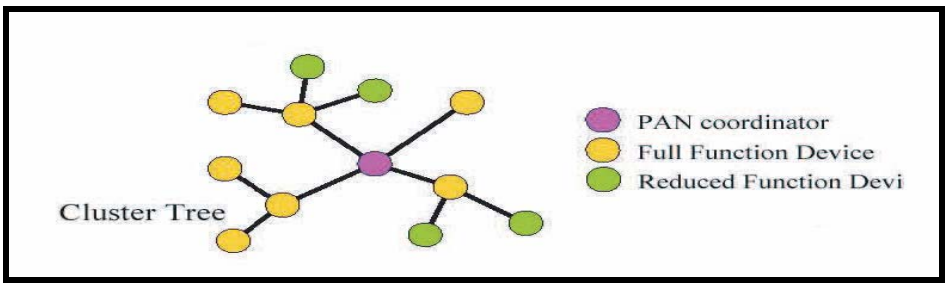


Fig. 2.3. Topología Cluster Tree

2.3. Capa física del 802.15.4

La capa física del 802.15.4 está separada en dos subcapas: PHY data service y PHY management que son las encargadas de transmitir y recibir mensajes a través del medio radio.

Algunas características globales de la capa física son el control del transceptor radio, calidad del enlace (LQI), selección de canal, detector de energía(ED), detección de portadora (CCA) para su uso en CSMA-CA a nivel MAC, etc…

El estándar define dos opciones de transmisión según la banda de frecuencia utilizada (868/915 MHz y 2450 MHz), ambos están basados en el DSSS[1] (direct sequence spread spectrum o espectro ensanchado). Como se puede observar en la tabla 2.2, tenemos distintas velocidades de transmisión dependiendo la frecuencia que utilicemos, con lo que conlleva que a mayor frecuencia mayor velocidad pero menor área de cobertura debido a la atenuación de la señal a frecuencias elevadas.

También se puede observar los distintos tipos de modulación que se utilizan dependiendo de la frecuencia utilizada (BPSK y O-QPSK) pero no vamos a entrar en estos temas ya que no es objetivo del trabajo (véase [7]).

Tabla 2.2. Parámetros establecidos en función de la banda utilizada.

PHY (MHz)	Frequency band (MHz)	Spreading parameters		Data parameters		
		Chip Rate(Kchip/s)	Modulacion	Kbps	Ksimbolos/s	2^n
868/915	868–868.6	300	BPSK	20	20	1
	902–928	600	BPSK	40	40	1
2450	2400–2483.5	2000	O-QPSK	250	62.5	4

En la banda de 2,4GHz, que es la más eficiente en cuanto al uso del ancho de banda, disponemos de un total de 16 bandas o canales de 2 MHz con una distancia entre canales de 5MHz para evitar interferencias, tal y como vemos en la Fig. 2.4.

[1] DSSS es una técnica de modulación que utiliza un código de pseudorruido para modular directamente una portadora, de tal forma que aumente el ancho de banda de la transmisión y reduzca la densidad de potencia espectral. La señal resultante tiene un espectro muy parecido al del ruido, de tal forma que a todos los radiorreceptores les parecerá ruido menos al que va dirigida la señal.

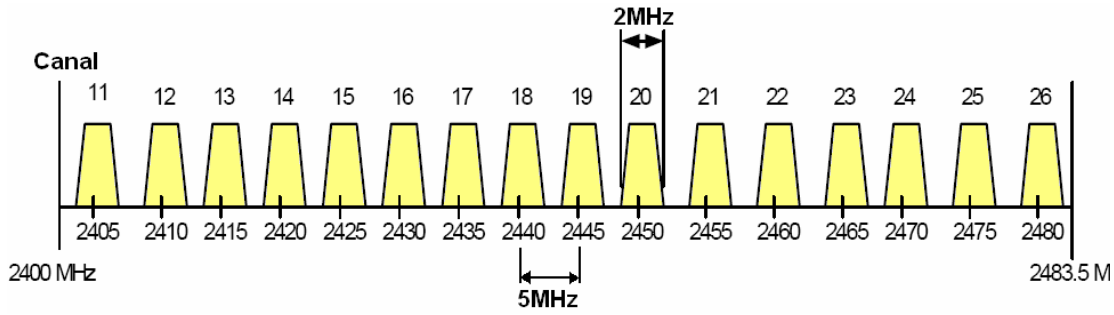


Fig. 2.4. Espectro radioeléctrico en la banda de 2.4GHz

La unidad de datos de nivel físico (PPDU) que es ilustrada en la Fig. 2.5. esta compuesto por tres partes bien definidas:

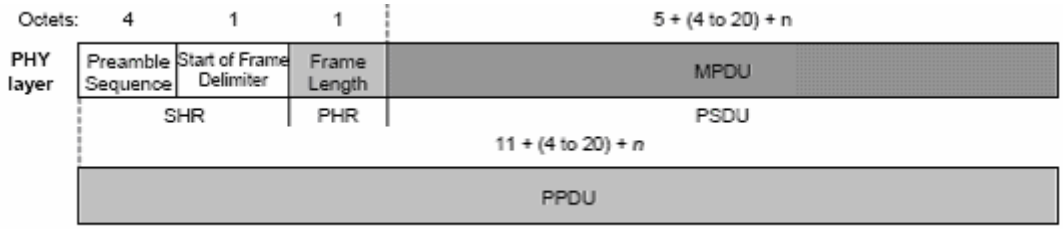


Fig. 2.5. Estructura paquete PPDU

- SHR, permite a un dispositivo receptor sincronizarse para poder leer bien la información contenida en la PPDU, también indica el final de trama ya que la trama puede tener una longitud variable.
- PHR indica la longitud de información ya que esta puede ser variable como hemos comentado anteriormente.
- PSDU es la información de la trama a nivel físico.

2.4. Capa MAC del 802.15.4

La capa MAC proporciona dos servicios: MAC data service y MAC management service, ambos servicios interactúan en la capa MAC y permiten la transmisión y recepción de unidades de datos, MPDU, a través del servicio de datos PHY. Las funciones mas revelantes de la capa MAC son:

- Generar beacons en el caso de ser un PAN coordinator y que el resto de nodos se sincronicen al ritmo de los beacons.
- Mecanismo de acceso al medio CSMA-CA.
- Asociación o desasociación a una PAN.
- Funciones de seguridad (encriptación AES).
- QoS mediante GTS (Granted Time Slot).
- Mecanismos de fiabilidad entre nodos (ACK's).

La ventaja de este nivel MAC respecto al de otros estándares es que tan solo se dispone de 21 primitivas de servicio o comandos, lo que redunda en un hardware más sencillo y más barato de fabricar.

2.4.1. Funcionamiento

Dentro de la capa MAC del protocolo 802.15.4 encontramos tres tipos de funcionamiento a la hora de transmitir datos: transmisión de datos utilizando beacons, transmisión de datos sin utilizar beacons y transmisión de datos utilizando beacons con un tiempo de acceso garantizado (GTS).
A continuación describiremos los tres tipos existentes:

La transmisión **con beacons** está orientada a redes donde existe el papel del coordinador (topología en estrella), el PAN coordinator se encarga de transmitir beacons cada cierto tiempo para que los dispositivos dentro de su red se puedan sincronizar. Como se puede observar en la Fig. 2.6, entre beacon y beacon se establece una supertrama compuesta por 15 slots (slots de backoff) llamados CAP (Contention Access Period), mediante los cuales los dispositivos de la red podrán transmitir de forma coordinada.

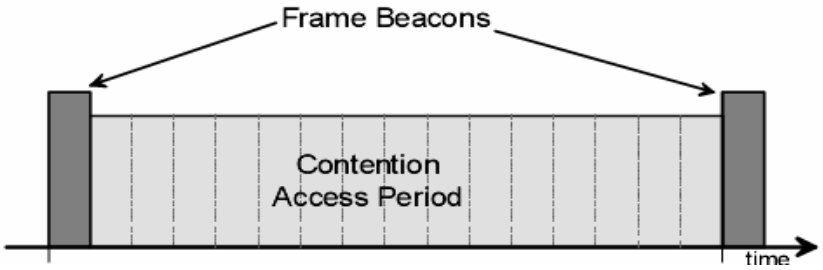


Fig. 2.5. Transmisión con beacons

En este caso, como mecanismo de acceso al medio utilizamos CSMA-CA ranurado en donde las ranuras de backoff están alineadas con el comienzo de un beacon. Cada vez que un dispositivo desea transmitir, primero tiene que alinearse con el siguiente slot de backoff y entonces tiene que esperar un número aleatorio de ranuras de backoff. Si el canal está libre, en el siguiente slot comenzaría a transmitir. Si el canal está ocupado, dejara pasar otro número aleatorio de ranuras de backoff. Los únicos paquetes que no están sometidos a CSMA-CA son los ack's y los beacons.

La transmisión **sin beacons** está orientada a redes peer to peer donde todos se comunican entre todos sin la intervención de un coordinador. El mecanismo de acceso al medio es el CSMA-CA no ranurado en lo que cada dispositivo transmite en el momento que es necesario sin esperar la pauta (beacons) de un PAN coordinator.
El mecanismo de funcionamiento sería el siguiente: cada vez que un dispositivo desea transmitir datos o comandos MAC tiene que esperarse un tiempo aleatorio, si encuentra el canal libre espera un tiempo de backoff, pasado este tiempo intenta transmitir. Si el canal estuviera ocupado después del periodo de backoff volvería a esperar otro tiempo aleatorio así sucesivamente hasta llegar a un tope de intentos definidos con la variable BE dentro del algoritmo CSMA-CA.
Esta es la configuración que nos encontramos por defecto durante el transcurso de las pruebas de nuestro trabajo de final de carrera, con la

diferencia que los motes solo realizaran dos tiempos de backoff a la hora de intentar transmitir.

La transmisión **con beacons y un tiempo de acceso garantizado** nos proporciona una latencia mínima para aquellos dispositivos que necesiten tener este parámetro garantizado, los GTS (Guaranted Time Slot) vendrán definidos en tiempo en la trama de beacon y se sitúan dentro del periodo de libre contención (Contention Free Period (CFP)). Este espacio es reservado para que en caso de haber mucho tráfico ciertos dispositivos tengan siempre prioridad para lograr mínima latencia.

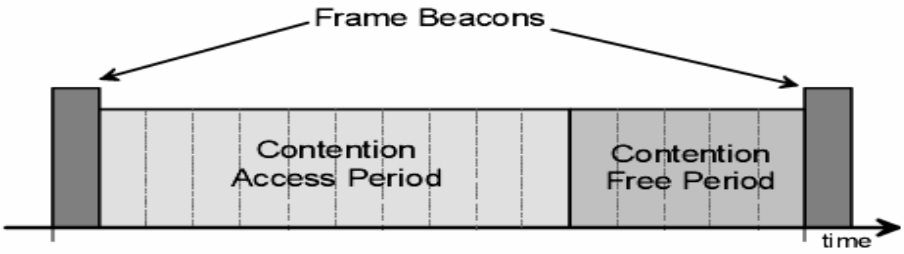


Fig. 2.6. Transmisión con beacon y zona exclusiva para GTS

Para observar la estructura de los distintos tipos de trama del estándar 802.15.4 véase anexo H.

2.4.2. Mecanismo de robustez

El estándar IEEE 802.15.4 se mueve en entornos hostiles y medios compartidos por lo que se han definido una serie de mecanismos para que así sea más robusto en de estos entornos:

- CSMA-CA: Sistema anteriormente comentado basado en la detección de portadora evitando colisiones. Su esquema de funcionamiento lo hace excelente compartiendo el medio.

- Paquetes con confirmación (ACK): Cuando enviamos paquetes, se nos devuelve un paquete ACK confirmando que el paquete de datos o cualquier otro ha sido recibido correctamente.

- Verificación de los datos (CRC): Mediante un polinomio generador de grado 16 obtenemos la redundancia y podemos comparar el CRC enviado con el calculado en destino y de esta manera verificar los datos.

- Restricciones de consumo: IEEE 802.15.4 esta pensado para aplicaciones que utilicen una batería o una unidad de energía agotable, ya que estas aplicaciones transmitirán información de forma muy esporádica por lo que la cantidad de energía que consume cuando escucha el canal es ultra baja.

- Seguridad: Implementa seguridad de clave simétrica mediante el estándar de encriptación AES, el manejo y gestión de la claves es derivado a capas superiores.

2.4.3. Implementación usada en nuestro trabajo

En nuestro trabajo, el transceptor radio que ha sido utilizado, es el modelo CC2420 perteneciente a la casa Chipcon [8], que implementa el estándar IEEE 802.15.4.

Este chipset cumple con todas las características mencionadas en los apartados anteriores, destacando que funciona por defecto en el modo IEEE 802.15.4 sin beacons y utiliza el mecanismo de acceso al medio CSMA-CA con la única modificación: sólo realiza dos intentos de transmisión, es decir, realiza un backoff para intentar transmitir y si ve el medio ocupado realiza otro backoff mas grande e intenta transmitir, si no lo consigue, desiste de la transmisión.

Para consultar alguno de sus parámetros por defecto y la forma de establecer esos cambios, véase anexo D.

CAPITULO 3. ENTORNO DE DESARROLLO

El capítulo numero 3 detalla el entorno de desarrollo que se utiliza para la confección de este Trabajo Final de Carrera, consistente en la programación de las dos aplicaciones de sensado. En especial se detalla la plataforma MICAz de XBow, ideal para desarrollar aplicaciones de sensado y la que más se adapta y/o parece al sistema RFID.

3.1. Mote – Kit2400

Los sensores utilizados en el proyecto, así como las placas programadoras y sensoras, vienen en un kit, el MOTE-KIT2400 (véase [9]). Los dispositivos de este kit se denominan Motes MICAz y son fabricados por la casa CrossBow (véase [10]).

Fig. 3.1 Imagen del MOTE - KIT2400, fabricado por CrossBow.

El MOTE-KIT2400 (Fig. 3.1) contiene todos los componentes necesarios para implementar una red sensora inalámbrica. El detalle del contenido del kit de CrossBow es:

- Ocho motes MICAz MPR2400CA.
- Dos placas programadoras: una placa programadora vía Ethernet, denominada MIB600, y otra vía puerto serie, la MIB510.
- Cuatro placas sensoras MTS310CA con termómetro, fotocélula, acelerómetro, sensor magnético, micrófono y zumbador.
- Tres placas sensoras del tipo MTS300CA con termómetro, fotocélula, micrófono y zumbador.
- Una placa sensora MDA300CA con higrómetro y termómetro.
- Software necesario para el desarrollo de aplicaciones e instalación de sistema operativo TinyOS.

Entre las aplicaciones que se pueden desarrollar con este kit encontramos la monitorización y seguridad de edificios residenciales e industriales, redes sensoras para vehículos de motor, para el campo de la agricultura entre otras muchas.

3.1.1. Mote MICAz, MPR2400CA

Los motes MPR2400CA (véase [11]) son sistemas inalámbricos para el desarrollo de aplicaciones de medición, control de acceso y demás aplicaciones de redes sensoras inalámbricas. En definitiva, es una plataforma especialmente diseñada para el desarrollo de todo tipo de aplicaciones para redes sensoras empotradas.

Los motes MICAz ofrecen una tasa de transferencia de datos vía radio, gracias a la interfaz 802.15.4 (ZigBee), de hasta 250 Kbps como hemos podido ver en el capítulo anterior, y comunicaciones inalámbricas con otros motes. También soportan funciones de enrutamiento como si de un router se tratara, ofreciendo la posibilidad de crear redes con topología estrella o con topología *mesh* o multisalto.

Se han utilizado estos motes como plataforma para desarrollar las dos aplicaciones de identificación. Las dos aplicaciones se servirán de la plataforma MICAz para implementar un sistema RFID que proporcione información acerca de un producto o de la persona según aplicación.

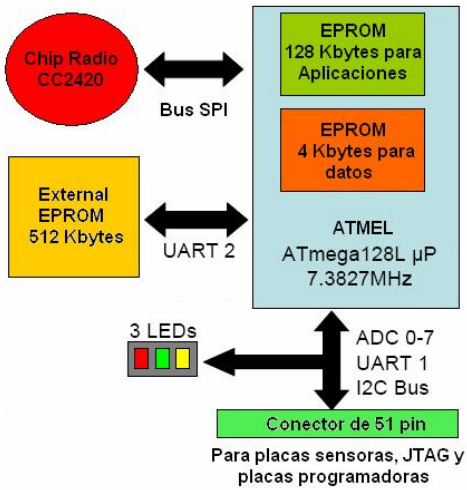


Fig. 3.2. Diagrama de bloques de un mote MPR2400CA

Cada Mote MICAz incorpora un microprocesador ATMEL Mega128L de 7 MHz (Fig. 3.2). Dicho procesador incorpora una EPROM[2] de 128 Kbytes de espacio para los programas y otra EPROM de 4 kbytes para datos. El procesador está conectado a través del puerto UART[3] 2 a una memoria flash externa de 512 kbytes de capacidad.

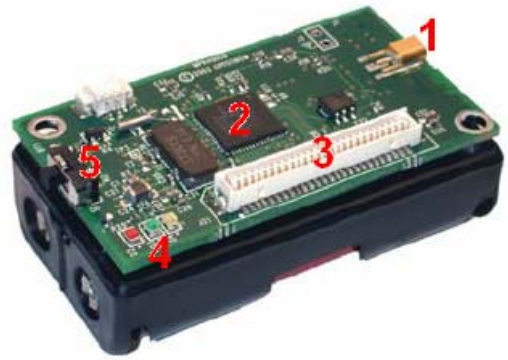
[2] EPROM es un chip de memoria ROM inventado por el ingeniero Dov Frohman que retiene los datos cuando la fuente de energía se apaga. En otras palabras, es no volátil.

[3] UART es un circuito integrado que utilizan ciertos sistemas digitales basados en microprocesador, para convertir los datos en paralelo, que manda la CPU, en serie y viciversa, con el fin de comunicarse con otro sistema externo.

Dicha memoria está al servicio de las aplicaciones que requieran guardar datos. Mediante el Bus SPI[4] el procesador se conecta con el chip radio CC2420 de la casa Chipcon, encargado de completar las cabeceras del paquete que se enviará vía radio. La interfaz radio ZigBee utiliza la banda de frecuencias compatible globalmente ISM[5] a 2'4385 GHz.

Además del procesador, cada MICAz incorpora una interfaz de usuario formada por tres diodos LEDs de colores rojo, verde y amarillo. También posee un conector de expansión de 51 pins para periféricos. Entre los periféricos compatibles encontramos las placas sensoras MTS y MDA, las placas programadoras MIB510 y MIB600 y otros dispositivos de entrada/salida.

En los motes se ejecuta el sistema operativo TinyOS (véase [12]). Los programas para este SO específico para dispositivos sensores deben de estar programados en lenguaje de programación nesC, una versión reducida de C y desarrollado específicamente para dispositivos empotrados (*Network Embedded System C*) que está basado en tareas y orientado a eventos.



1. Antena para comunicación radio, ZigBee (802.15.4)
2. Microprocesador ATMEL Mega128L
3. Conector de expansión de 51 pin macho, para periféricos como las placas MTS
4. Tres LEDs: rojo, verde y amarillo
5. Interruptor ON/OFF

Fig. 3.3 Mote MICAz, MPR2400CA:

El mote MICAz tiene unas dimensiones de 58x32x7 milímetros y pesa 58 gramos excluyendo las baterías. La batería de los motes se compone de dos pilas AA, concediendo un voltaje de 3 Voltios. El consumo de corriente del MPR2400CA en modo activo es de 8 mA, y en modo hibernación (modo *standby*) es menor a 15 µA. Las dimensiones del mote son considerables, pero existen los motes similares, los MICA2Dot, que tienen un tamaño no superior al de una moneda de dos euros.

3.1.2. Programador MIB510

La placa programadora MIB510 (véase [13]) utiliza un conector de expansión como vía de comunicación con los motes. La placa nos permite cargar código en los motes y hace la función de puerta de enlace entre los motes y el PC.

[4] El bus SPI es un bus diseñado para que sobre éste se coloque un dispositivo maestro y un dispositivo esclavo para una comunicación punto a punto.
[5] ISM (Industrial Scientific and Medical) son bandas reservadas internacionalmente para uso no comercial de Radio Frecuencia electromagnética en áreas industrial, científica y médica.

1. Interfaz puerto serie, conexión a PC
2. Interfaz Mote JTAG
3. Interfaz MICA2Dot
4. Interfaz MICAz y MICA2
5. Interfaz ISPJTAG
6. Interruptor ON/OFF para deshabilitar puerto serie
7. Botón Reset
8. Entrada alimentación 5 Voltios

Fig. 3.4 Placa programadora MIB510.

Esta estación base para los sensores inalámbricos MICAz tiene una interfaz RS-232 serie que permite hacer de puente entre los motes MPR2400CA y un PC. Las especificaciones de esta interfaz, a través del puerto serie, muestran una velocidad de transferencia de 57'6 kbaudios[6] para las comunicaciones con los motes y el PC, y una velocidad de programación de 115'2 kbaudios.

Fig. 3.5 MIB510 con un mote MICAz acoplado

El programador incorpora cinco LED's separados en dos grupos: el primer grupo esta compuesto por dos LED's, verde y rojo, que nos indica si la placa esta conectada a la corriente eléctrica (verde) y si se están cargando los datos en el mote acoplado desde el PC (rojo). El segundo grupo esta compuesto de tres LED's que forman una interface de usuario como en el caso de los motes. También incorpora un interruptor para deshabilitar la transmisión serie con los motes, y un botón para resetear la placa programadora y el mote acoplado.

La placa se alimenta de 5 Voltios y unos 50mA de corriente. El mote acoplado no necesita alimentación ya que se lo cede la misma MIB510.

[6] El baudio es la unidad informática que se utiliza para cuantificar el número de cambios de estado, o eventos de señalización, que se producen cada segundo durante la transferencia de datos.

3.1.3. Programador MIB600

El MOTE-KIT2400 también incluye otra placa programadora, la MIB600 (véase [14]). Realiza las mismas funciones que la placa programadora MIB510 pero a través del puerto Ethernet (véase anexo C para ver su configuración).

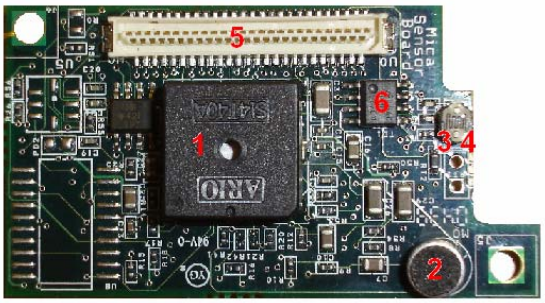
1. Interfaz puerto Ethernet
2. Interruptor ON/OFF para deshabilitar puerto Ethernet
3. Botón reset
4. Entrada fuente de alimentación de 5V

Fig. 3.6 Placa programadora MIB600.

Incorpora, como la MIB510, el interruptor para deshabilitar el puerto de comunicación, en este caso el puerto Ethernet. También tiene un botón reset, dos LEDs que cumplen las mismas funciones que el programador serie y el conector de 51 pin para acoplar los motes MICAz.

3.1.4. Sensor MTS300CA

Esta placa sensora de dimensiones 56x11x31 milímetros puede medir temperatura y nivel luminoso (véase [16]). Está dotada de un zumbador (Sirius PS14T40A) y de un micrófono (Panasonic WM-62A). El zumbador es capaz de producir tonos de 4 KHz y como se vera posteriormente será utilizado en la aplicación de control de acceso para avisar de un acceso no autorizado.



1. Zumbador de tonos de 4 KHz, Sirius PS14T40A
2. Micrófono Panasonic WM-62A
3. Fotocélula, Sensor Lumínico Clairex CL9P4L
4. Termómetro Panasonic ERT-J1VR103J
5. Conector hembra,51 PIN, para mote MICAz
6. Detector de tonos

Fig. 3.7 Placa sensora MTS300CA.

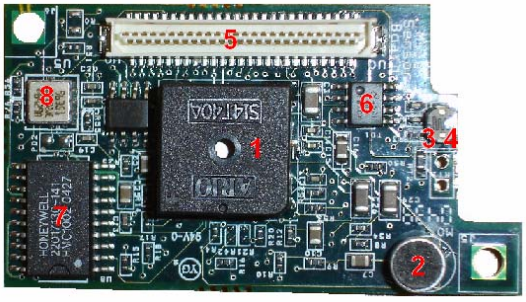
En nuestro trabajo de final de carrera se utilizarán estas placas sensoras para la indicación de acceso denegado en la aplicación de control de acceso. Dicha indicación se hará a través del zumbador.

3.1.5. Sensor MTS310CA

Esta placa sensora tiene las mismas dimensiones que la MTS300CA y puede medir temperatura, nivel lumínico, aceleración y campo magnético. También tiene un zumbador y micrófono idénticos al anterior modelo de placa, al igual que la fotocélula y el termómetro.

Esta placa, a diferencia de la placa MTS300CA, incorpora un acelerómetro de bajo coste modelo ADXL202JE y un sensor magnético modelo Honeywell HMC1002 para mediciones de intensidad del campo magnético (véase [16]).

En este trabajo no ha sido necesaria la utilización de estos sensores por lo que no se comentarán en detalle. Si se desea, se puede ver el trabajo fina de carrera de Carlos López en el cual se hace uso de dichos sensores (véase [17]).



1. Zumbador de tonos de 4 kHz, Sirius PS14T40A
2. Micrófono Panasonic WM-62A
3. Fotocélula, Sensor Lumínico Clairex CL9P4L
4. Termómetro Panasonic ERT-J1VR103J
5. Conector hembra,51 PIN, para mote MICAz
6. Detector de tonos
7. Sensor magnético Honeywell HMC1002, 2 Axis
8. Acelerómetro Analog ADXL202JE, 2 Axis

Fig. 3.8 Placa sensora MTS310CA:

3.2. TinyOS

TinyOS es un sistema operativo basado en componentes para sistemas empotrados inalámbricos y de libre distribución. Su principal característica es que utiliza muy poca memoria y maneja múltiples tareas y eventos.

La dinámica de los programas realizados bajo TinyOS está conducida por tareas las cuales pueden ser interrumpidas para atender eventos. Los eventos disponen de una prioridad mayor para ser atendidos.

El sistema operativo guarda en cola todas las tareas que tiene que realizar (Fig 3.10). En caso de que el nodo no tenga ninguna tarea, entrará en modo standby para ahorrar batería.

Muchos de los programas a los que llamamos mediante tareas tienen un evento indicando que su cometido ha sido realizado. Generalmente este evento puede desencadenar otra parte del código. Otros son inicializados y se activan cada vez que se realizan algo para lo que han sido programados (p.ej: La recepción de un paquete).

Muchas operaciones de bajo nivel son finalmente gestionadas por el sistema operativo y así nos permite centrarnos en la aplicación. Se puede entender que TinyOS es como una pirámide en la que a bajo nivel nos encontramos componentes de sistema y en capas superiores encontramos componentes de uso habitual que llaman de forma implícita y encapsulada a componentes del sistema.

Para entender TinyOS, debemos de tener presentes 3 clases de abstracción que son la esencia para su entendimiento y posterior programación:

- Los comandos son las llamadas hacia abajo (Fig 3.2). Al llamar a un comando, éste dentro de su componente llama a otros componentes de capas inferiores.

- Los eventos conllevan el efecto inverso, una llamada hacia arriba (Fig 3.2). Un componente de bajo nivel avisa a uno de alto nivel de que ha sucedido algo. Si sucede un evento, éste tiene mayor prioridad que cualquier tarea y pasaría a ejecutarse. En caso de haber una interrupción dentro de una interrupción lo más probable es que no sea ejecutada esta hasta que acabara la anterior, siendo esto algo crítico.

- Las tareas son porciones de código que se ejecutan de forma asíncrona siempre y cuando la CPU no tenga que ejecutar ningún evento. Estas se ejecutan por orden de llamada (FIFO) y tal y como antes hemos comentado en caso de que la cola este vacía el procesador entrara en standby hasta que un nuevo evento haga despertar al procesador.

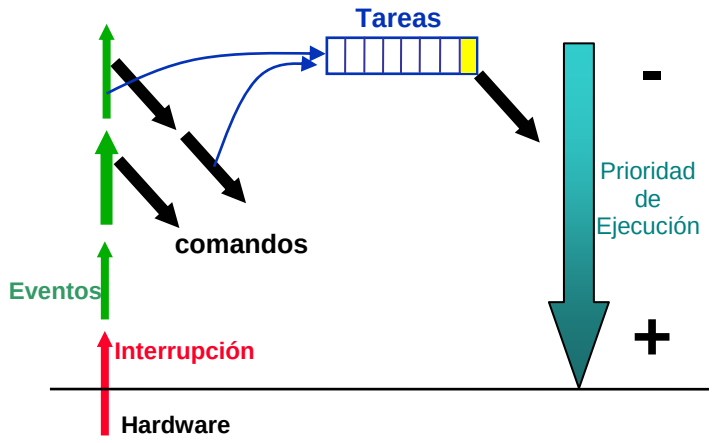


Fig. 3.10 Esquema funcional de TinyOS

Cuando programamos y compilamos una aplicación, esta va siempre unida de forma implícita al sistema operativo, de forma que no existe el sistema operativo (previamente precargado) y el programa compilado, sino el conjunto que es subido al sistema empotrado cada vez que necesitamos actualizar el programa y su tamaño en Bytes sigue siendo minúsculo.

Para programar mediante TinyOS utilizamos una extensión del lenguaje C llamada nesC, la cual nos permite el uso de interficies que van conectadas a componentes.

Para crear un programa en nesC, únicamente tenemos que escoger los componentes (las partes funcionales) previamente programadas y después "linkarlos" mediante otro fichero de configuración a las interficies que serán utilizadas como código en nuestra futura aplicación.

3.3. Introducción al nesC

Como hemos comentado antes, nesC (Network Embedded Systems C), es una extensión del lenguaje C pero no siempre fue así. Inicialmente y hasta la versión 0.6 de TinyOS, todo el sistema y la programación de aplicaciones fue desarrollado en C. Luego se rescribió todo el código en nesC para la versión 1.0 de TinyOS en adelante.

NesC es un lenguaje de programación que se utiliza para crear aplicaciones que serán ejecutadas en sensores con el sistema operativo TinyOS. Por tanto, dicho lenguaje, proporciona características necesarias para poder realizar aplicaciones de una forma más cómoda para el programador.

La adopción de esta extensión obtuvo algunas ventajas:

- La aparición de interficies.
- Detección de errores de conexión de componentes a interfices (wiring) en tiempo de compilación.
- Generación automática de documentación referida a funciones.
- Optimización del código utilizado mediante componentes.
- Una mayor limpieza con lo que conseguimos más sencillez.

En nesC cada aplicación está construida a base de componentes enlazados para formar un ejecutable. Los componentes pueden proporcionar o utilizar interficies de otros componentes. La forma de acceder al uso en otro contexto de estos componentes es mediante estas interficies que son bidireccionales es decir mediante eventos y mediante comandos. Con estas interficies podemos acceder a la funcionalidad de los componentes.

En nesC se definen dos clases de componentes: los módulos y las configuraciones. Los módulos contienen el código en sí del componente mediante interficies. Por otra parte, las configuraciones enlazan (wiring) las interficies de los componentes que vamos a necesitar en nuestro módulo. Este fichero es crucial porque si no está bien configurado, la aplicación no

compilará. Tanto módulos como configuraciones utilizan la misma extensión *.nc. La diferencia entre ambos está en la sintaxis que utilizan y es por ello que siguen un código de letras para diferenciar unos de otros. Por ejemplo, cuando trabajamos con un programa “miaplicacion”, miaplicacionM.nc sería el módulo y miaplicacion.nc seria el fichero configuración.

Para concluir, destacar que el código de la aplicación no se ejecutará en modo secuencial, ya que atiende a una programación basada en eventos. Esto conlleva que se programan las acciones que se desea realizar cuando se produzca un evento, que se ejecutará cada vez que se lleve a cabo dicho evento.

3.4. SerialForwarder

El programa SerialForwarder (véase [18]), tiene la funcionalidad de leer paquetes del puerto serie del ordenador, en el caso de utilizar la MIB510, o del puerto Ethernet, utilizando la MIB600, y replicarlos hacia el puerto servidor de las conexiones TCP. SerialForwarder no muestra los datos de los paquetes, solamente muestra los contadores de paquetes escritos y leídos por el, así como de clientes conectados.

Una vez ejecutado, SerialForwarder escucha peticiones de conexiones de clientes TCP en el puerto 9001 o 10002. Como se puede observar en la Fig. 3.11, el puerto TCP es el 9001 y la tarjeta es la MIB510 debido que en el campo “Mote Communications” se observa la línea “serial@COM4:57600” donde el 57600 es la velocidad de transferencia de los datos en Kbps y la interficie es la serie COM4. También se puede observar las contadores antes mencionados y los botones de encendido y apagado de la aplicación.

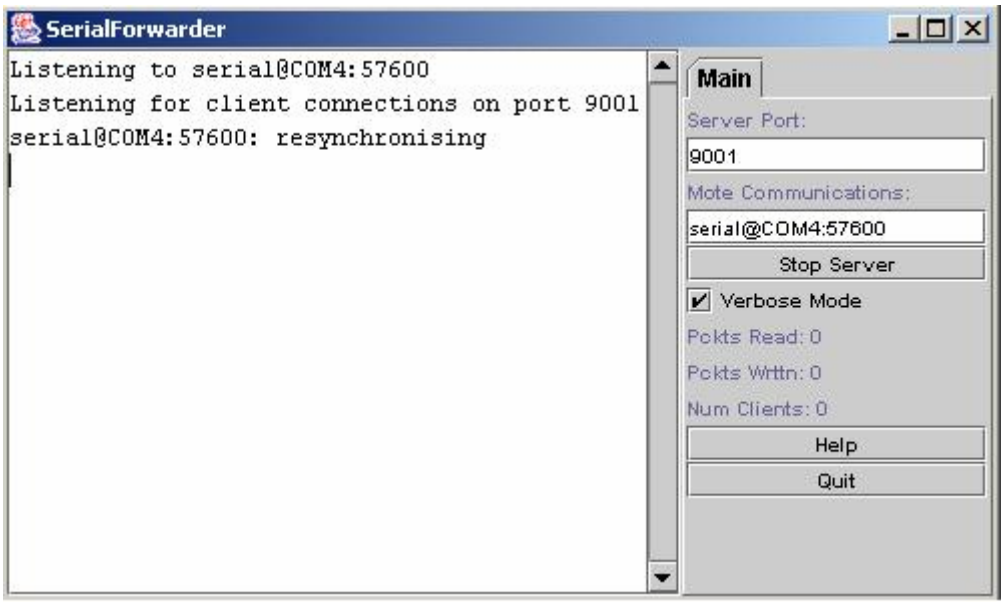


Fig. 3.11 Pantalla de la aplicación SerialForwarder.

Tanto la aplicación “Control de acceso” como “Autoescáner” ejecutan automáticamente SerialForwarder. Esta ejecución automática se lleva a cabo gracias a un archivo serial.bat[7], que se inicia desde la aplicación en C#. El archivo serial.bat lleva los parámetros necesarios para la configuración del programa SerialForwarder:

- El puerto 9001, por defecto de la placa MIB510
- Placa programadora utilizada: serial (para indicar la MIB510) o network (para seleccionar la MIB600)
- El puerto serie utilizado COMx, por ejemplo COM1
- Tasa de transferencia de los motes MICAz, 57’6 kbaudios.

3.5. Introducción a C#

El lenguaje de programación utilizado para el desarrollo de la parte del PC de las dos aplicaciones, véase anexo E y F, es C#. Dicho lenguaje ofrece facilidad en la creación del entorno gráfico de las dos aplicaciones, además de tener un excelente soporte técnico para resolver los problemas derivados de la programación.

El lenguaje C# se ha construido suponiendo que los sistemas de software modernos se construyen usando componentes. Por lo tanto, C# proporciona soporte a nivel de lenguaje para los constructores básicos de los componentes, como pueden ser propiedades, métodos y eventos. C# tiene mecanismos para permitir, al mismo tiempo, un orientación a componentes y a la vez dar un gran rendimiento.
C#, además de soportar el desarrollo de software basado en componentes, es un lenguaje completamente orientado a objetos que implementa casi todos los conceptos y abstracciones presentes en C++ y Java.
C# consigue aunar orientación a objetos y rendimiento. Esta aproximación tiene la ventaja de una completa orientación a objetos, pero tiene la desventaja de ser muy ineficiente. Por esta razón, para mejorar el rendimiento, otros lenguajes, como Java, separan el sistema de tipos; en tipos primitivos y todo el resto. Dando lugar un mejor rendimiento en los tipos primitivos, pero con una separación, a veces molesta, entre tipos primitivos y tipos definidos por el usuario. En C# se han aunado ambas aproximaciones presentando lo que se llama un sistema unificado de tipos, en el que todos los tipos, incluso los primitivos, derivan de un tipo objeto común, a la vez que permite el uso de optimizaciones para tipos primitivos.

[7] Línea escrita en archivo serial.bat para la ejecución del programa SerialForwarder ya configurado para los motes MICAz y la placa MIB510: “C:\Surge-View\serialforwarder -comm serial@COM1:57600”

3.6. Introducción a Bases de datos ACCESS

Microsoft Access es un sistema de gestión de bases de datos (DBMS[8]) para uso personal o de pequeñas organizaciones. Es un componente de la suite Microsoft Office aunque no se incluye en el paquete básico. Para bases de datos de gran calibre (en cuanto a volumen de datos o de usuarios) es recomendable usar otros sistemas de datos como Microsoft SQL Server, MySQL u Oracle. Su principal función es ser una potente base de datos, capaz de trabajar en si misma o bien con conexión hacia otros lenguajes de programación, tales como Visual Basic 6.0 o Visual Basic .NET. Pueden realizarse consultas directas a las tablas contenidas mediante instrucciones SQL. Internamente trae consigo el lenguaje Visual Basic for Application (VBA) el cual es similar en forma a VB6.
Permite el ingreso de datos de tipos: Numericos, Texto, Fecha, Sí/No, OLE, Moneda, Memo y Boolean. Pueden desarrollarse aplicaciones completas basadas en Microsoft Access, pues trae consigo las herramientas necesarias para el diseño y desarrollo de formularios para el ingreso y trabajo con datos e informes para visualizar e imprimir la información requerida.
Su funcionamiento se basa en un motor llamado Microsoft Jet, y permite el desarrollo de pequeñas aplicaciones autónomas formadas por formularios Windows y código VBA (Visual Basic para Aplicaciones). Una posibilidad adicional es la de crear ficheros con bases de datos que pueden ser consultados por otros programas. Entre las principales funcionalidades de Access se encuentran:

- Crear tablas de datos indexadas.
- Modificar tablas de datos.
- Relaciones entre tablas (creación de bases de datos relacionales).
- Creación de consultas y vistas.
- Consultas referencias cruzadas.
- Consultas de acción (INSERT, DELETE, UPDATE).
- Formularios.
- Informes.
- Llamadas a la API[9] de windows.
- Interacción con otras aplicaciones que usen VBA (resto de aplicaciones de Microsoft Office, Autocad, etc.).
- Macros.

Además, permite crear frontends o programa que muestra la interfaz de usuario de bases de datos más potentes ya que es un sistema capaz de acceder a tablas externas a través de ODBC[10] como si fueran tablas Access.
Es un software de gran difusión entre pequeñas empresas (PYMES) cuyas bases de datos no requieren de excesiva potencia, ya que se integra

[8] DBMS (Sistemas Gestores de Bases de Datos) es un tipo de software muy específico, dedicado a servir de interfaz entre las bases de datos y las aplicaciones que la utilizan.
[9] Una API (del inglés Application Programming Interface) es un conjunto de especificaciones de comunicación entre componentes software, es decir, un método de abstracción en la programación generalmente entre los niveles o capas inferiores y los superiores del software.
[10] ODBC (Open DataBase Connectivity) es un estándar de acceso a Bases de Datos desarrollado por Microsoft Corporation.

perfectamente con el resto de aplicaciones de Microsoft y permite crear pequeñas aplicaciones con unos pocos conocimientos de programación.
Entre sus mayores inconvenientes figuran que no es multiplataforma, pues sólo está disponible para sistemas operativos de Microsoft, y que no permite transacciones. Su uso es inadecuado para grandes proyectos de software que requieren tiempos de respuesta críticos o muchos accesos simultáneos a la base de datos.

En éste trabajo, hemos utilizado una base de datos access para guardar todos los datos referentes a los dispositivos usuarios / productos en cada aplicación. En el caso de la aplicación de seguridad (véase anexo E-2), la base de datos contendrá la información perteniente (nombre, apellidos, teléfono, etc.) de cada uno de los usuarios que accedan al recinto. En el caso de la aplicación autoescáner, la base de datos nos proporcionara toda la información referente a los productos (nombre, precio, etc.).

CAPITULO 4. APLICACIONES DESARROLLADAS

4.1. Introducción

En este apartado vamos a desarrollar las dos aplicaciones realizadas en este trabajo de final de carrera “control de acceso” y “autoescáner” mediante la plataforma MICAz descrita en el apartado 3.1. Las aplicaciones desarrolladas son útiles tanto para el control y vigilancia de un área restringida como para el control de cualquier tipo de mercancía, debido a que toda persona o producto será “interrogado” por el elemento controlador para ser informado de su presencia.

Estas aplicaciones se han desarrollado de forma dual: cada una de las dos aplicaciones consta de un código específico para los motes MICAz, programado en lenguaje nesC, y otro código para el PC, que conforma la interfaz gráfica y de centralización de datos de dichas aplicaciones. Este último programa ha sido desarrollado con el lenguaje C# de Microsoft, y nos permitirá controlar y gestionar la información adquirida por los motes.

A continuación pasamos a describir las funcionalidades de las dos aplicaciones desarrolladas; primero describiendo la aplicación en si, seguido de una descripción de los programas en nesC para los motes MICAz y, finalmente, comentando las características de los programas de gestión de datos e interfaz gráfica del PC central en C#.

4.2. Aplicación “Control de acceso”

4.2.1. Descripción general de la aplicación

Hoy en día, en el mundo donde nos movemos, se hace imprescindible el control de acceso a una determina área (p.ej: parlamento de los diputados, bolsa de valores, campos de fútbol, etc), es decir, supongámonos que no existiera un control de acceso a un campo de fútbol, esto podría desembocar en una aglomeración de personas deseosas de ver a sus ídolos, llegándose a producir avalanchas de seres humanos con la desastrosa consecuencia que eso conlleva, seis muertos y cuarenta y tres heridos como se pudo observar el pasado año en un partido en Irán (véase[19]).

Los controles de acceso que estamos acostumbrados a ver son más bien físicos, es decir, se hace imprescindible el uso de un aparato, llámese molinete o lector magnético, con el inconveniente que eso provoca debido a que todas las personas que deseen acceder deberán pasar por el determinado elemento llegándose a producir retrasos de acceso indeterminados.

Gracias a la tecnología RFID podemos realizar el control de acceso a cualquier instalación evitando los inconvenientes que presentan los antiguos sistemas de una manera más económica y eficaz.

El sistema realizado esta compuesto por un elemento validador, un elemento identificador y una interface gráfica (PC) como se puede ver en la Fig. 4.2.

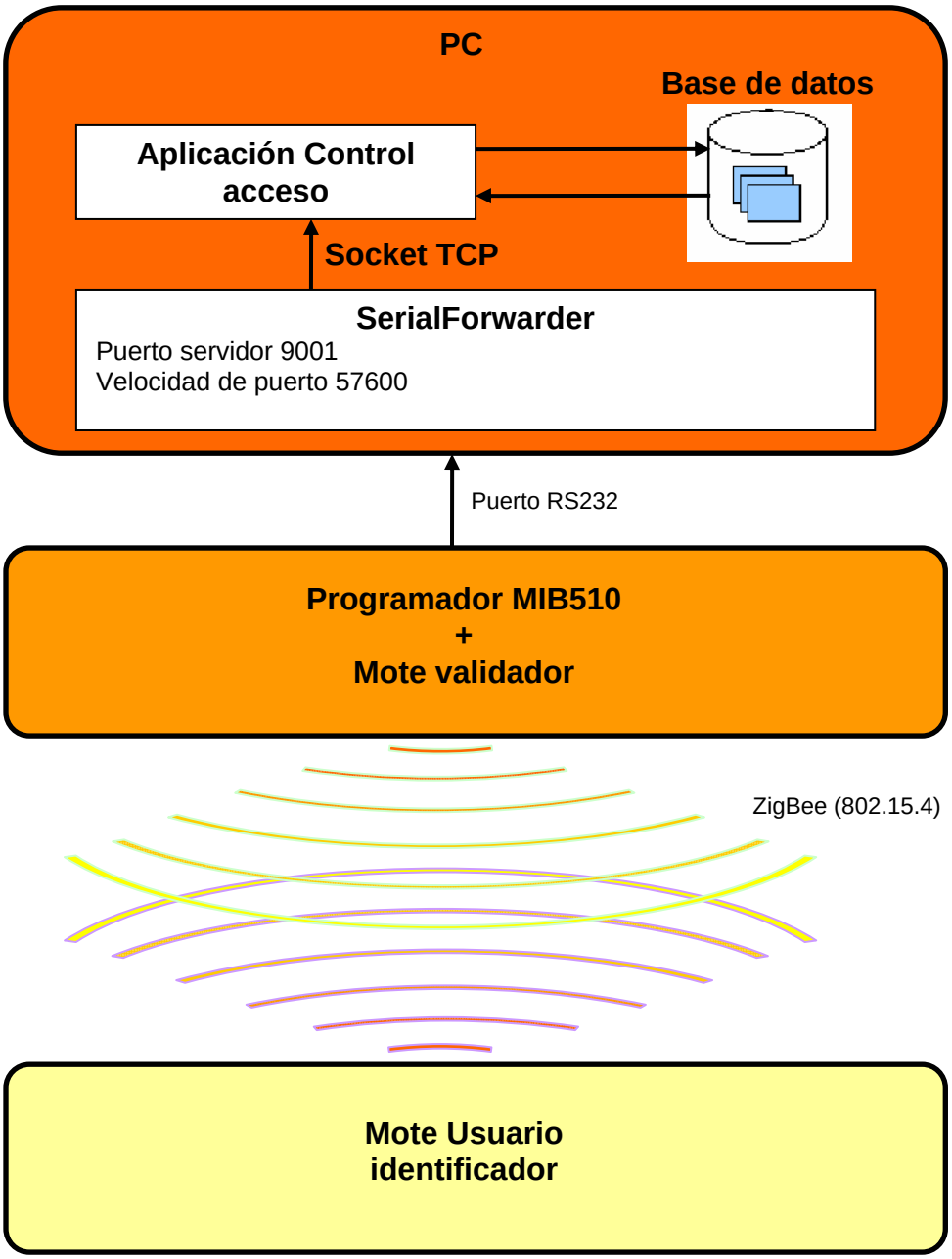


Fig.4.2 Esquema general de la aplicación de control de acceso

La aplicación Control de Acceso se encargara de visualizar por la interficie gráfica (pantalla) de un PC, toda la información necesaria del usuario que acceda a el área, de esta forma se tendrá un control exhaustivo del acceso a

dicha zona, en el apartado 4.2.4 se detalla con mas profundidad dicha aplicación.

El elemento validador, conectado al PC mediante la placa programadora MIB510 (véase apartado 3.1.2), será el encargado de dar permiso o denegarlo a los usuarios que quieran acceder al recinto controlado. Este se encargara de pedir la identificación al mote que llevara el usuario y verificar su acceso. En el apartado 4.2.3 se explica la función de este elemento.

La función del mote identificador del usuario es la de identificarse una vez reciba la petición del validador. En el siguiente apartado se ve con más detalle su funcionamiento.

4.2.2. Aplicación mote MICAz parte usuario

Como hemos comentado, cada usuario deberá llevar adherido un RFID de identificación. Este RFID será programado previamente para atorgarle un identificador exclusivo mediante nesC (véase apartado 3.3).
La función que debe realizar este RFID es exclusivamente la de responder a las peticiones de identificación de los elementos controladores y la de activar el zumbador en el caso de que se deniegue el acceso. A continuación se podrá observar el diagrama de funcionamiento de este dispositivo.

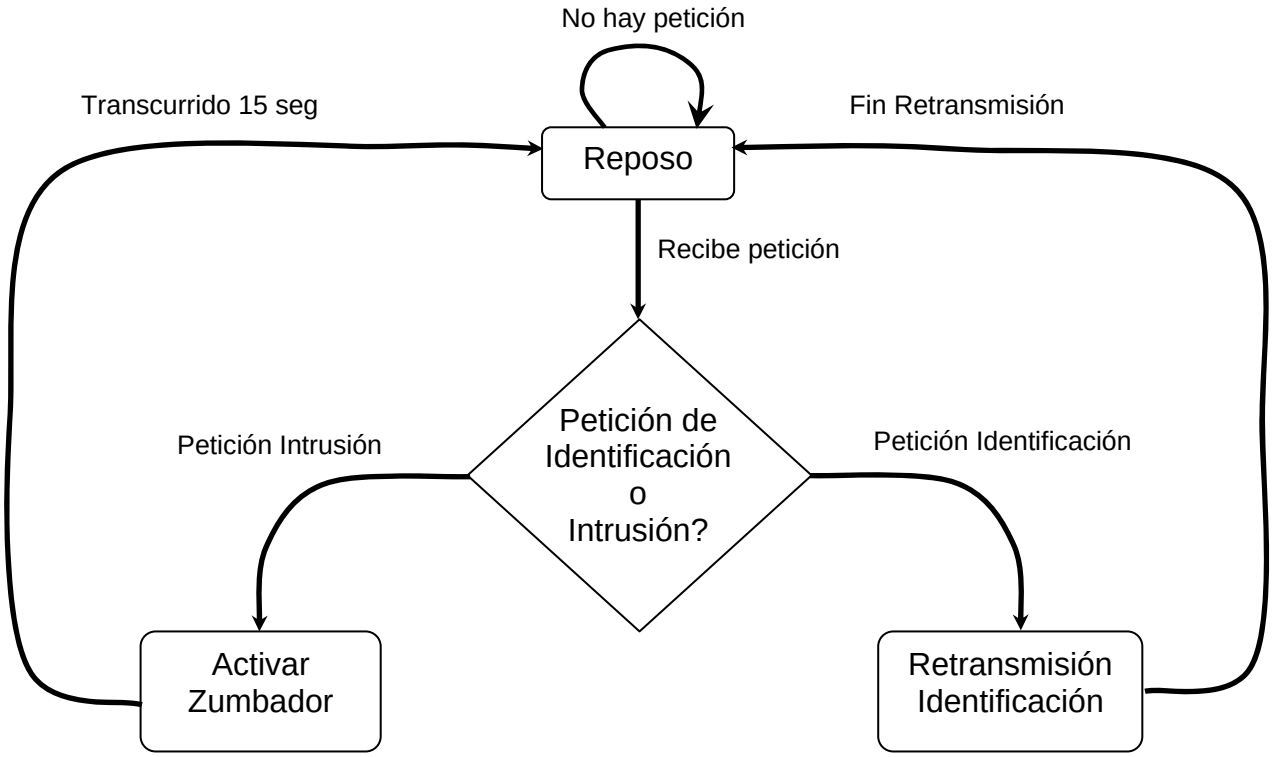


Fig.4.3 Diagrama del funcionamiento del RFID usuario

Como se puede ver en la Fig. 4.3, el mote usuario permanecerá en estado de reposo hasta recibir una petición, en este estado el consumo de energía es nulo, lo que conlleva que la batería tenga una vida útil elevada (2 años o más).
Al recibir una petición del elemento controlador, el dispositivo determinará qué tipo de petición le ha sido solicitada. Si se trata de una petición de identificación el dispositivo usuario retransmitirá su identificación y retornará al estado de reposo, de lo contrario, si recibe una petición de intrusión, el RFID activará el zumbador (véase apartado 3.1.4 y 3.1.5) de la placa sensora que tendrá conectada a él mediante un conector de 51 pins. Una vez activado el zumbador, éste emitirá una señal de 2.5Khz capaz de ser detectada por el oído humano durante 15 seg, una vez concluido este tiempo retornara al estado de reposo.

A continuación describiremos el formato de la trama de identificación a nivel de la aplicación MICAz:

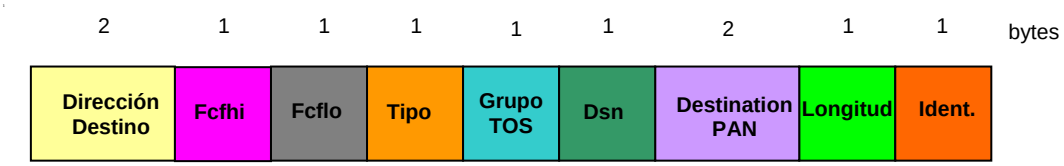


Fig.4.3 Formato de la trama de identificación

El formato del paquete que el RFID usuario transmitirá al controlador es el que se muestra en la Fig. 4.3 con una longitud total de 11 bytes. En el se observan distintos campos que pasamos a detallar a continuación:

- **Dirección destino→** Este campo, como su nombre bien indica, contiene la dirección del dispositivo destino (controlador). Este campo es rellenado con la dirección del dispositivo controlador que nos ha realizado una petición de identificación.

- **Fcfhi y Fcflo→** es un registro de 16 bits (parte hi y lo) donde cada grupo de bits indican una característica de la transmisión (utilización ack o el tipo de trama utilizada, etc (véase [20])).

- **Tipo→** destinado a identificar el tipo de mensaje, en caso de los paquetes trasmitidos por el dispositivo usuario, este siempre tendrá el valor 2 por defecto.

- **Grupo TOS→** define el grupo TOSBase, por si se quiere filtrar los paquetes según grupo o redes. Por defecto este campo adquiere el valor de 129.

- **Dsn→** Este campo es una secuencia de paquetes enviados, en cada nuevo paquete se incrementa en 1, este campo es rellenado por el sistema TinyOS.

- **Destinatio.PAN→** Indica la dirección de la red de área local en que se encuentre y es rellenado por el sistema TinyOS.

- **Longitud→** Indica la longitud del campo datos, en este caso 3 bytes, y es rellenado por el sistema operativo.

- **Ident→** Este campo indica la Identificación del usuario, la longitud de esta identificación es de 1 byte debido que para hacer las pruebas era mas que suficiente (2^8 = 256 Identificaciones diferentes) pero puede ser mayor hasta llegar a 102 bytes, también hemos utilizado este campo para identificar la dirección origen del dispositivo por lo que aparte de ser la identificación se tomara como la dirección del dispositivo.

En el anexo E se podrá observar con más detalle el código empleado para la configuración de los dispositivos de usuario.

4.2.3. Aplicación mote MICAz parte validador

El elemento validador está compuesto por un mote MPR2400CA acoplado a la placa programadora MIB510. Hemos optado por esta placa para la realización de las pruebas por su facilidad de uso ya que no necesita configuración previa como la placa programadora MIB600. Además esta placa al poco tiempo de empezar las pruebas empezó a fallar y dejo de funcionar correctamente.

El programa realizado en nesC para el validador está compuesto por dos partes, la parte de validación de identificación y la de comunicaciones. La parte de validación, como su nombre indica, es la encargada de comprobar si la identificación recibida esta autorizada. La parte de comunicaciones es la que se encarga de transmitir y recibir mensajes tanto por el medio radio como por el puerto serie. Esta parte de código está implementada, dentro del software facilitado por la casa XBOW, con el nombre “TOSBase”, su función es la de retransmitir los mensajes recibidos por el medio radio por el puerto serie y viceversa, así pues he modificado el código del programa TOSBase para que, aparte de conmutar los paquetes, haga la función de validación.

El controlador estará conectado a la corriente eléctrica a través de la placa MIB510 debido al alto consumo necesario por este elemento ya que continuamente está emitiendo peticiones de identificación a intervalos muy cortos (300 ms), como se puede observar en el diagrama de funcionamiento siguiente.

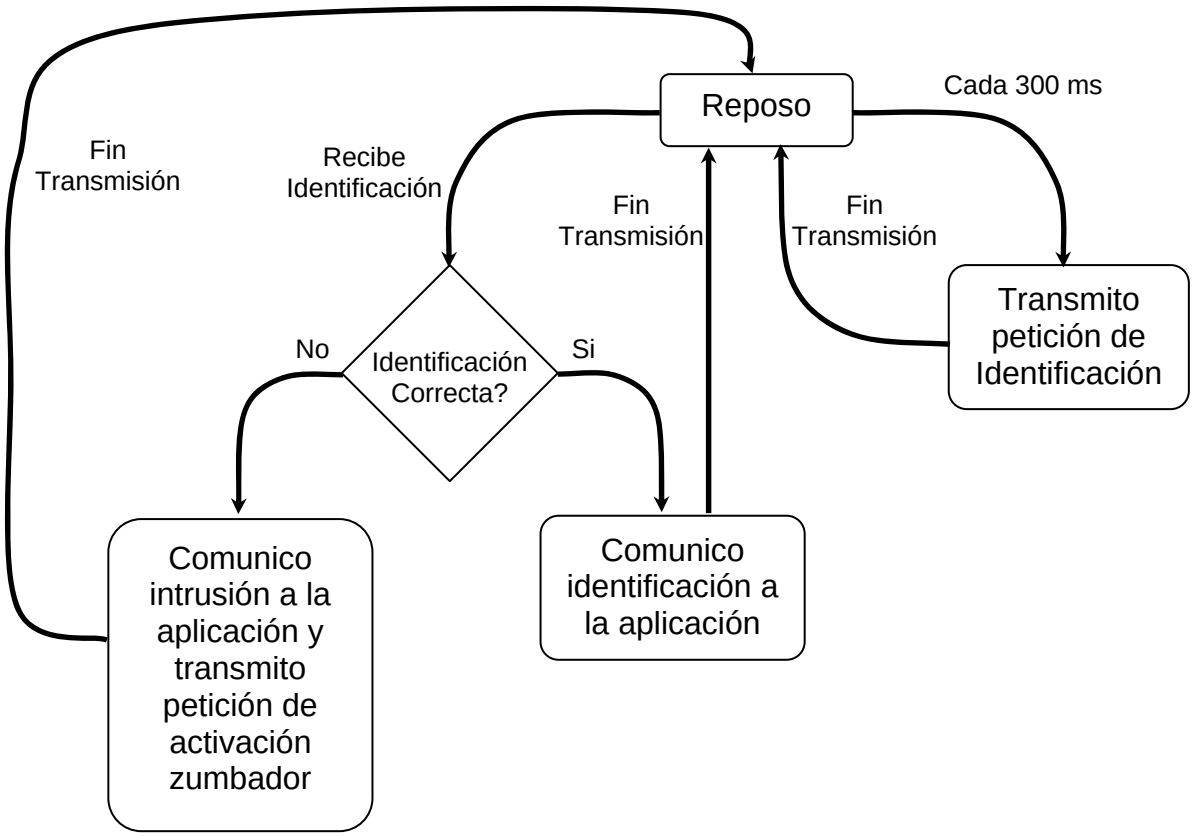


Fig.4.4 Diagrama del funcionamiento del RFID controlador

El RFID controlador realizará continuamente envíos de peticiones de tipo identificación (tipo 1), en modo broadcast[11] al medio con una cadencia de 300 ms. Hemos optado por esta cadencia debido a los resultados obtenidos al hacer las pruebas (capitulo 5) donde se puede ver que con 13 usuarios accediendo a la vez a una zona determinada el tiempo medio de recogida de los datos de todos los RFID es de 131.9 ms. Por lo tanto decidimos estipular una cadencia de 300 ms ya que la probabilidad de que más de 22 personas accedan a un recito controlado a la vez es baja.

Una vez el validador reciba una trama de Identificación la verificará para asegurarse de que el usuario tiene acceso al área controlada. Esta verificación se realizará comparando el identificador recibido con los identificadores validados ubicados en un vector dinámico. Si el usuario está acreditado, el validador enviará por el puerto serie de la placa programadora la identificación del usuario para informar a la aplicación de que dicho usuario ha accedido al recinto. Por lo contrario, si la identificación no está validada, el controlador enviará a través del puerto serie un identificador de intrusión en el recinto para notificar a la aplicación de la intrusión. A su vez, éste enviará un mensaje radio al RFID usuario de tipo intrusión (tipo 2) para que éste active su señal acústica y así poder detectar al intruso.

[11] **Broadcast** (o en castellano "difusiones"), se producen cuando una fuente envía datos a todos los dispositivos de una red, dirección destino 0xFF.

A continuación describiremos el formato de la trama de petición identificación e intrusión a nivel de la aplicación MICAz:

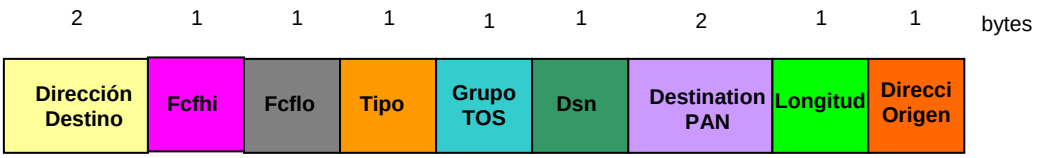


Fig.4.5 Formato de la trama de petición de identificación e intrusión

Como se puede ver en la Fig. 4.5, la trama es similar a la que transmite el RFID usuario, con la única variación de que el campo identificación es reemplazado por el campo dirección origen. Los campos realizan la misma función que en el caso de la trama usuario con la diferencia que en las tramas enviadas por el controlador el campo “tipo” tiene un significado particular, si el campo toma valor 0x01 el RFID usuario lo interpretara como una petición de identificación, en cambio si toma el valor 0x69 el RFID usuario lo interpretará como petición de intrusión y por lo tanto activara el zumbador acoplado a él.

En el anexo E se podrá observar con más detalle el código empleado para la configuración del dispositivo controlador descrito en este apartado.

4.2.4. Aplicación PC

La aplicación del PC esta realizada mediante programación en C# como ya hemos comentado anteriormente. La aplicación se encarga de mostrar gráficamente al personal encargado del acceso al recinto, (p.ej: vigilante de seguridad) los usuarios que están accediendo a dicho recinto detectados por el RFID controlador. La aplicación PC no interactúa con el RFID controlador, sino únicamente recibe información de éste y se encarga de visualizarla por pantalla. La interfície gráfica que el personal de seguridad dispone en su lugar de trabajo tiene la apariencia de la Fig. 4.6.

Fig.4.6 Apariencia de la interfície grafica de seguridad

Como se puede ver en la Fig. 4.6, la interficie gráfica está compuesta por una ventana central donde se muestra los datos de cada usuario, dos botones de control para arrancar / cerrar la aplicación y un calendario donde figura la fecha y hora del día correspondiente.
Una vez se arranca la aplicación de seguridad, se hace una conexión mediante un socket TCP (puerto 9001) con el programa java *serialforwarder* (véase apartado 3.4) para poder recibir los paquetes provinentes del controlador.
Cuando se ha establecido la conexión y accede un usuario al recinto, el RFID controlador nos envía la identificación de dicho usuario a la aplicación. Una vez la aplicación tiene el correspondiente identificador, se conecta con la base de datos para realizar la búsqueda de los datos del usuario. Esta búsqueda se realiza comparando el identificador que nos proporciona el RFID controlador con los identificadores que tenemos registrados en la base de datos. Cuando se encuentra el identificador la aplicación muestra los datos del usuario almacenados en la base de datos por pantalla.
En el caso de que un usuario no tenga permiso para acceder al recinto, el RFID controlador informará a la aplicación de dicha intrusión y esta a su vez mostrará por pantalla un aviso de acceso no permitido como se puede observar en la Fig. 4.7.

Fig.4.7 Apariencia de la interfície gráfica de seguridad en el caso de un acceso no permitido

La aplicación, deja constancia del acceso del usuario al área controlada mediante la escritura de los datos personales y la hora de acceso en un fichero de texto. Cada día se genera un fichero nuevo de nombre la fecha del día correspondiente, este fichero se genera una vez al día a la hora de arrancar la aplicación y es guardado dentro del PC.

En el anexo E se puede ver el código empleado para la realización de la interface de control de acceso.

4.3. Aplicación "Autoescáner"

4.3.1. Descripción general de la aplicación

La segunda aplicación que se ha desarrollado pretende facilitar el trabajo al personal de un supermercado a la hora de realizar el cobro de los productos que suministran. Esta aplicación tiene un funcionamiento muy similar a la aplicación de control de acceso, con la diferencia que en este caso el elemento controlador tiene que interrogar a varios RFID a la vez con los problemas que ello conlleva debido a que el medio de transmisión es compartido.
La aplicación Autoescáner tiene como finalidad facilitar el trabajo y reducir el tiempo de este a los usuarios de la tecnología RFID.
Este tipo de aplicaciones se puede aplicar en varios entornos de trabajo para realizar lecturas de productos o mercancías (p.ej: inventarios).

A continuación se puede ver el esquema general de la aplicación autoescaner:

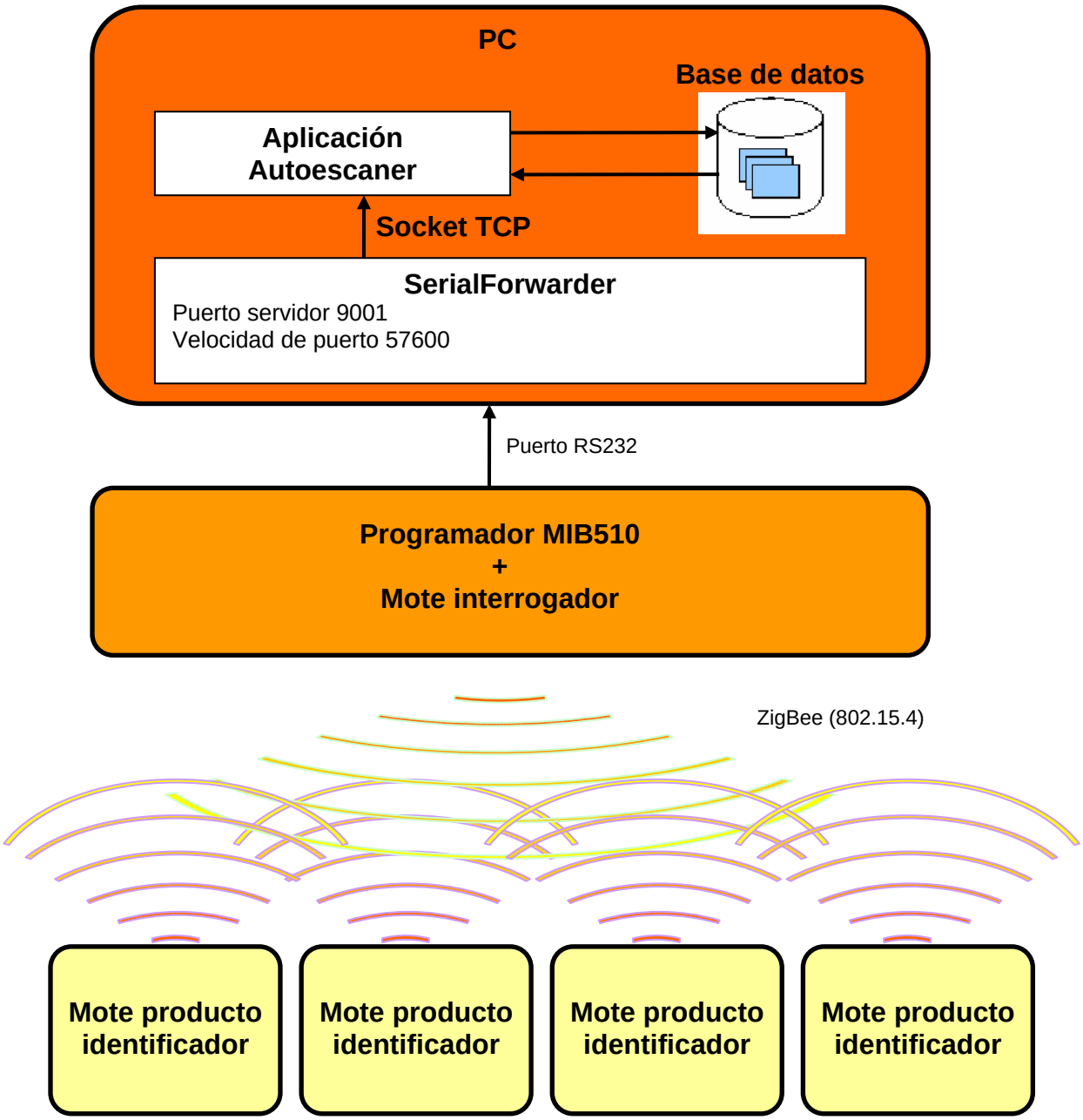


Fig.4.8 Esquema general de la aplicación autoescaner

La aplicación autoescáner se encarga de visualizar por pantalla todos los productos que son escaneandos por el mote interrogador para poder contabilizar u operar con dicha información. En los siguientes subapartados se detalla la funcionalidad de cada elemento que compone esta aplicación.

4.3.2. Aplicación mote MICAz parte producto

Como hemos comentado anteriormente, la función del mote identificador es la de identificarse cuando es consultado por el mote interrogador. Este RFID será

programado previamente para atorgarle un identificador exclusivo mediante nesC (véase apartado 3.3).
A continuación se podrá observar el diagrama de funcionamiento de este dispositivo.

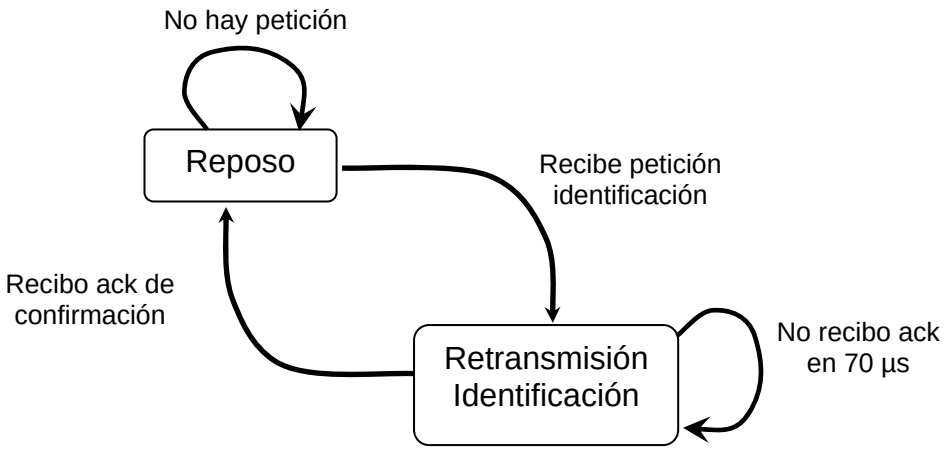


Fig.4.9 Diagrama funcional del RFID identificador

Como se puede observar, el RFID identificador permanece en reposo hasta que recibe una petición de identificación. Al recibir la petición transmite su identificación y espera a recibir el ack de confirmación del RFID interrogador. Si ésta no se produce, el RFID identificador vuelve a retransmitir su identificación así sucesivamente hasta que reciba el ack. En el capitulo 5, se verá que con nueve retransmisiones se recibe el 100% de información de 13 RFID transmitiendo casi simultáneamente resolviendo el problema del acceso al medio cuando varias fuentes requieren su acceso.
Según la configuración del chip de comunicaciones CC2420 de chipcon que incorpora los motes MICAz, éste espera un máximo de 70 µs para recibir el ack del paquete retransmitido, en el anexo D se puede observar la configuración estándar del chip de comunicaciones CC2420.
El formato del paquete que transmite el mote identificador es idéntico al que hemos visto para la aplicación de control de acceso, por lo que nos remitimos a la Fig. 4.3 para observar dicha estructura.
En el anexo F se puede observar el código implementado para la realización del mote identificador.

4.3.3. Aplicación mote MICAz parte interrogador

Como hemos comentado en el apartado 4.2.3, el elemento interrogador está compuesto por un mote MPR2400CA acoplado a la placa programadora MIB510 que es la que suministra la corriente eléctrica. En el siguiente diagrama se puede observar el funcionamiento del mote interrogador:

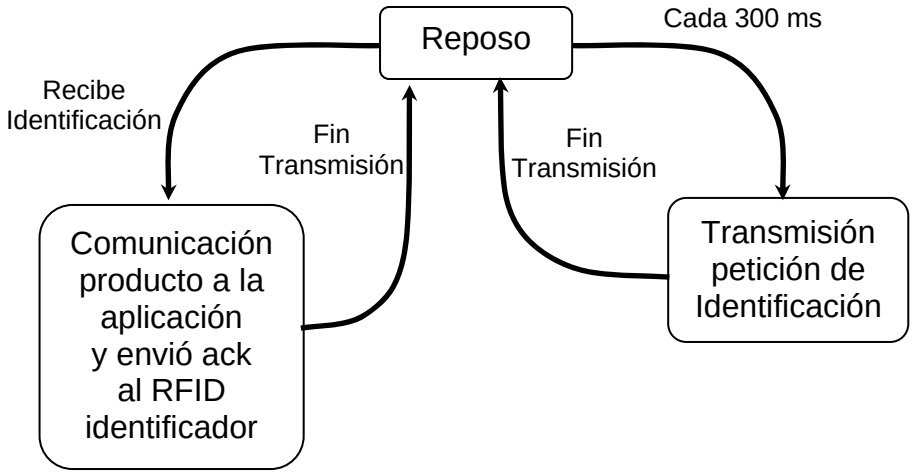


Fig.4.10 Diagrama funcional del RFID interrogador

El RFID interrogador realizará continuamente envíos de peticiones de identificación en modo broadcast al medio con una cadencia de 300 ms. Hemos optado por esta cadencia por lo comentado en el apartado 4.2.3.
Una vez el controlador reciba una trama de Identificación la comunicara a la aplicación y enviará un reconocimiento al mote que ha enviado la identificación para indicarle que ha recibido el paquete y retornará a su estado inicial.
El formato de la trama es idéntico al formato de la trama del controlador en la aplicación de control de acceso, con la diferencia que en este caso el campo tipo siempre tiene el mismo valor (0x01) ya que solo se transmite un tipo de trama (identificación), por lo que nos remetimos a la Fig. 4.5 para observar la estructura de dicha trama.

En el anexo F se podrá observar con más detalle el código empleado para la configuración del dispositivo interrogador descrito en este apartado.

4.3.4. Aplicación PC

La aplicación del PC está realizada mediante C# como ya hemos comentado anteriormente. La aplicación se encarga de mostrar gráficamente al personal encargado de los cobros, (p.ej: cajero) los productos adquiridos por los usuarios del establecimiento. La aplicación PC no interactúa con el RFID interrogador. Únicamente recibe información de éste y se encarga de gestionar y visualizar por pantalla los distintos productos. La interficie gráfica que el cajero dispone en su lugar de trabajo tiene la apariencia de la Fig 4.11.

Fig.4.11 Apariencia de la interface gráfica del vendedor

Como se observa en la Fig. 4.11, la interficie gráfica está compuesta por una ventana donde se muestran los productos que el usuario ha adquirido en el establecimiento, tres botones de control para arrancar, cerrar la aplicación y realizar una nueva captura de productos, y finalmente un recuadro donde se realiza la suma del coste de todos los productos seleccionados por el usuario.
Una vez se arranca la aplicación, se hace una conexión mediante un socket TCP (puerto 9001) con el programa java *serialforwarder* (véase apartado 3.4) para poder recibir los paquetes provenientes del interrogador.
Cuando se ha establecido la conexión y un usuario pasa por caja con los productos adquiridos, el RFID interrogador envía la identificación de los productos interrogados a la aplicación, una vez la aplicación tiene los correspondientes identificadores se conecta con la base de datos para realizar la búsqueda de los datos de estos, esta búsqueda se realiza comparando el identificador que nos proporciona el RFID interrogador con los identificadores que tenemos registrados en la base de datos, cuando se encuentra el identificador la aplicación muestra los datos de los productos almacenados en la base de datos por pantalla.

Hay que especificar que en la aplicación se podrían desarrollar diferentes funciones como; control de stocks, registro personal, etc, pero este no es el objetivo inicial del trabajo por lo que no se ha realizado una aplicación de este tipo.

En el anexo F se puede ver el código empleado para la realización de la interface de autoescaneo.

CAPITULO 5. PRUEBAS Y RESULTADOS

Este capítulo pretende validar las dos aplicaciones desarrolladas, así como explicar detalladamente las pruebas realizadas para comprobar su correcto funcionamiento.

Las pruebas realizadas darán al lector una idea bastante concreta de las posibilidades de la tecnología RFID, como puede ser tiempos de transmisiones o probabilidad de perdida de información.

Como se puede observar, primero describiremos las pruebas y presentaremos los resultados correspondientes para la aplicación de control de acceso seguidas de las pruebas de la aplicación de autoescaner.

5.1. Aplicación control de acceso

La aplicación control de acceso ha sido sometida a pruebas de medidas de tiempo en la comunicación entre el mote controlador y el mote usuario (RTT). En este caso no se han realizado pruebas de pérdida de información debido a que en una comunicación directa entre dos motes no hay pérdida de información ya que no hay colisiones en el medio.

5.1.1. Escenario y configuración

El escenario donde se han realizado las pruebas es el laboratorio 325 de la torre de profesores de la UPC en el campus del baix Llobregat en Casteldefels. Para validar la aplicación, hemos tomado como recinto de acceso controlado el laboratorio 325, por lo que hemos tenido que graduar la potencia de emisión de señal del chip CC2420 para que el alcance de la señal de emisión del controlador no sobrepasase la puerta de entrada de dicho laboratorio. Como se puede observar en la Fig. 5.1 la distancia entre la situación del mote controlador y la puerta de acceso a la sala es de 3,5 metros por lo que hemos configurado la potencia de transmisión para que la señal no sea detectable por ningún dispositivo que se encuentre a una distancia superior.

 # define CC2420_DEF_RFPOWER 0x04

Como se muestra en el anexo D, dentro del fichero de configuración del chip de comunicaciones CC2420, la variable *CC2420_DEF_RFPOWER* configura la potencia de transmisión del CC2420. Su rango de configuración está comprendido entre: 0x03 (3) donde su potencia es igual a -25 dBm con un consumo de corriente igual a 8.5mA hasta 0x1F (31) donde su potencia de transmisión es igual a 0 dBm con un consumo de 14.4 mA. Como se puede ver en el mencionado anexo, hemos tenido que graduar la potencia de transmisión a -2,5 dBm con un consumo de 8,9 mA para que la señal del controlador no sobrepasara la puerta del laboratorio.

Por otra parte, debido a que los motes trabajan sobre de la plataforma ZigBee en la banda de 2,4 GHz, se puede ocasionar problemas de interferencias con otros estándares inalámbricos que trabajen en la misma banda como puede ser Wireless. En el laboratorio 325 se dispone de un router inalámbrico Wireless, modelo WRT54G de la casa Cisco Systems, para dar acceso a Internet a los proyectistas de la sala, por lo que hemos tenido que seleccionar un canal de trabajo en los motes MICAz distinto al canal seleccionado por el router para evitar posibles interferencias entre los dos estándares.

#define CC2420_DEF_CHANNEL 26

Como se muestra en el anexo D, dentro del fichero de configuración del chip de comunicaciones CC2420 la variable *CC2420_DEF_CHANNEL* configura el canal en el cual se transmite la información. Las pruebas realizadas en el trabajo se han realizado en canal 26 entre 2479 y 2481 MHz (véase Fig. 2.4). Hemos escogido este canal debido a que en el laboratorio se captan más de un red Wireless que trabaja en la misma banda de frecuencia que ZigBee 2.4GHz, la red wireless más próxima a nuestro canal ZigBee era la del laboratorio, ésta utilizaba el canal 11 centrado a una frecuencia de 2462MHz y con un ancho de banda de 11 MHz por banda, por lo que esta señal no interferida en nuestro canal debido a que solamente llega hasta 2473 MHz y nos deja un margen de seguridad de 6 MHz (2479-2473 MHz).
Eligiendo el canal 26 nos aseguramos que nuestras tramas de información no se vean interferidas por señales procedentes de algún dispositivo trabajando bajo la plataforma Wireless.

5.1.2. Tiempo de validación

Después de configurar el escenario de trabajo nos dedicamos ha realizar las pruebas de tiempo de validación, en las que se medía el tiempo entre que el controlador enviaba la petición de Identificación a un dispositivo usuario y recibía esta identificación por parte del usuario.

Se ha considerado varios tiempos que en un principio eran desconocidos, como por ejemplo el tiempo de lectura y escritura en la memoria FIFO del transreceptor CC2420 o tiempo de configuración del CC2420 para la transmisión y recepción de paquetes. En cambio, hay tiempos que no hemos podido concretar, debido a que éstos dependen del hardware aplicado en los dispositivos, como puede ser el tiempo de encapsulación de las tramas, tiempo de propagación, etc. Sin embargo, éstos son tan pequeños (del orden de µs) que se ha optado por despreciarlos, ya que estos no repercuten sensiblemente en el resultado final.

El diagrama de tiempos asociado a la aplicación de control de acceso es el siguiente:

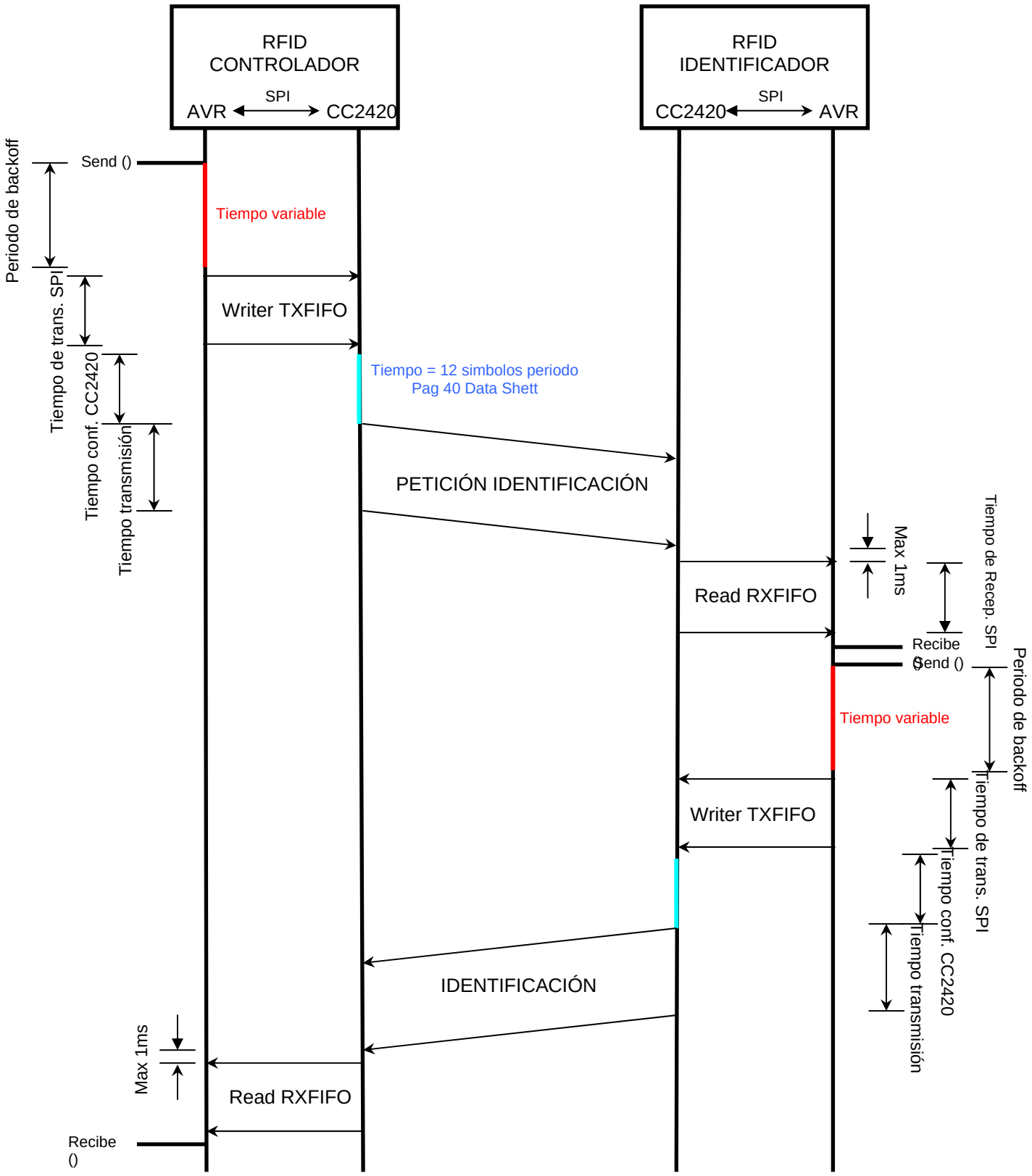


Fig.5.2 Cronograma temporal del sistema control de acceso.

A continuación describimos al detalle de cada tiempo que conforma la transmisión radio de la Fig. 5.2. Sus valores, o en su defecto, como calcularlo:

- **Periodo de backoff→** la capa de enlace ZigBee (IEE 802.15.4) establece como acceso al medio compartido el protocolo CSMA-CA (véase apartado 2.4.1). Este protocolo, antes de cada transmisión de un paquete, introduce un tiempo aleatorio de backoff para evitar colisiones. El tiempo de backoff es una variable aleatoria uniforme comprendida entre 0,32 ms a 5,12 ms (20 a 320 simbolos[12]).

- **Tiempo de Trans. SPI→** es el tiempo que emplea el microprocesador ATMEL para pasar la información por el bus SPI (ver Fig. 3.2) hacia el chip radio CC2420. Chip encargado de introducir algunos campos de cabecera y adaptar el paquete para la transmisión radio:

$$\text{Tiempo SPI} = (2'17\ \mu s / \text{byte}) \cdot L_{\text{Trama}} \qquad \textbf{(5.1)}$$

Donde,

L_{Trama} = Longitud de trama en bytes (incluye cabeceras a nivel MAC)

La longitud de trama de la petición e Identificación, a nivel de enlace, es de 20 bytes (16 cabecera + 1 datos), por lo que el tiempo de transferencia en el bus SPI es:

$$T_{\text{SPI}} = 2'17\ \mu s / \text{byte} \cdot (16)\ \text{bytes} = 0'03472\ \text{ms} \qquad \textbf{(5.2)}$$

- **Tiempo de config. CC2420→** Está definido como el tiempo de configuración de chip CC2420 para realizar la transmisión, este tiempo es constante y es igual a 12 símbolos (0.192 ms).

- **Tiempo de transmisión→** Es el tiempo que tarda el controlador en enviar toda la trama al medio.

$$T_{TX} = (L_{\text{Datos}} + L_{\text{Cabecera}}) / \text{Vel}_{\text{Canal Radio}} \qquad (\textbf{5.3})$$

Donde,

L_{Datos} = Número de bytes de los datos.
L_{Cabecera} = Bytes de cabecera introducidos por capas inferiores, 21 bytes.
$\text{Vel}_{\text{Canal Radio}}$ = Tasa de transferencia del canal radio, 250 Kbps.

$$T_{TX1} = (22\ \text{bytes} \cdot 8\ \text{bit/byte}) / 250\ \text{kbps} = 0{,}704\ \text{ms} \qquad \textbf{(5.4)}$$

[12] Símbolo es definido en el Data Shett del chip Chipcon como una unidad de tiempo con valor 16 µs.

- **Tiempo de propagación→** Es el tiempo que transcurre entre el envió de la trama al medio hasta que se recibe por el receptor, como hemos comentado anteriormente este es despreciable.

- **Otros tiempos→** Como hemos comentado, hay tiempos que no he podido especificar por su dificultar de computación, estos tiempos son minúsculos y se toman como despreciables.

5.1.2.1. Modelo Teórico

Como podemos ver, se puede calcular el margen de tiempos en el cual estará comprendido el tiempo de ida y vuelta, es decir, el tiempo en que el mote controlador emite la petición y este recibe la Identificación del usuario que accede al recinto controlado. Viendo el cronograma de la Fig. 5.2 podemos extraer una expresión matemática que nos caracterice el tiempo máximo y mínimo del proceso de identificación como podemos ver a continuación:

$$T_{TOTAL\ MIN} = (2 \times T_{(BO\ MIN)}) + (2 \times T_{TX}) + (4 \times T_{SPI}) + (2 \times T_{SET}) \quad \textbf{(5.5)}$$

$$T_{TOTAL\ MIN} = \mathbf{2{,}57088\ ms}$$

$$T_{TOTAL\ MÁX} = (2 \times T_{(BO\ MÁX)}) + (2 \times T_{TX}) + (4 \times T_{SPI}) + (2 \times T_{SET}) \quad \textbf{(5.6)}$$

$$T_{TOTAL\ MÁX} = \mathbf{12{,}17088\ ms}$$

Como vemos en las formulas 5.5 y 5.6, el tiempo total de transmisión en el sistema de control de acceso debe estar comprendido entre 2.57 ms y 12.17 ms, como vemos el tiempo más determinante es el tiempo de backoff ya que puede tomar un valor aleatorio como hemos comentado anteriormente.
Los tiempos estimados son ligeramente inferiores debido a que hemos despreciado tiempos como el de proceso, debido a que estos son muy pequeños y no afectaban determinantemente el resultado final.

5.1.2.2. Modelo práctico

Para llegar a cabo las pruebas, hemos configurado una aplicación que me recogía todos los tiempos (RTT's) para su posterior estudio, esta aplicación esta definida en el anexo E.
Las pruebas consisten en realizar 300 peticiones de identificación con sus correspondientes respuestas por el mote usuario, a continuación vemos los resultados obtenidos.

Tabla 5.1. Resultados obtenidos con un mote usuario.

Mínimo	**3,73915 ms**
Media	**8,90893 ms**
Máximo	**13,83681 ms**
Desviación Estándar	**2,1627 ms**
Prob. perdida	**0 %**

Como podemos observar, los resultados obtenidos son ligeramente superiores a los teóricos debido a los tiempos que no he podido contabilizar y he despreciado. Vemos que con un solo usuario accediendo al recinto, el tiempo medio de transferencia de datos es de 8,9 ms con una desviación estándar de 2,1 ms y un valor máximo de 13,83 ms.

Observando estos resultados, podemos deducir que el tiempo de transferencia de datos es muy pequeño, por lo que el comportamiento de la tecnología RFID en este tipo de sistema es muy satisfactorio.

Al ver estos resultados, decidimos realizar las mismas pruebas aumentando el numero de motes usuarios que accedían al recinto, en este momento observamos que a medida que aumentaba el numero de nodos usuarios, la probabilidad de perdida de información aumentaba considerablemente debido a la ocupación del medio de transmisión.
En el anexo D, se muestra como el chip de comunicaciones CC2420 esta configurado de manera que solo intentará transmitir la información dos veces, es decir, a la hora de transmitir una trama de información, éste realizara un backoff comprendido entre 0.32 y 5.12 ms. Si el medio esta ocupado, realizara un segundo backoff comprendido entre 0.32 y 20,16 ms, si el medio continua ocupado el CC2420 desistirá de transmitir la información.

Viendo estos resultados, decidimos introducir retransmisiones a nivel 3 para evitar problema de la ocupación del medio a la hora de tener varios nodos usuarios transmitiendo en el mismo instante. En la siguiente aplicación (autoescáner) se observa las consecuencias de estas modificaciones.

5.2. Aplicación Autoescaner

La aplicación autoescáner ha sido sometida a pruebas de medidas de tiempo y pérdidas de información en las comunicaciones entre el mote controlador y los motes usuarios. Al aplicar retransmisiones a nivel 3 en un sistema de varios nodos identificadores, como hemos comentado en la aplicación anterior y veremos a continuación, la probabilidad de pérdida disminuye considerable.

5.2.1. Escenario y configuración

La configuración de los dispositivos es la misma que anteriormente con la diferencia de que en este caso la potencia de transmisión es más reducida. En

esta aplicación nos interesa tener una distancia de transmisión de metro o metro y medio, con lo que hemos configurado la potencia a – 25 dBm con un consumo de corriente de 8,5 mA.
Los demás parámetros de configuración son idénticos a los definidos en el apartado 5.1.1.

5.2.2. Tiempo de recogida de información (configuración por defecto)

Después de configurar el escenario de trabajo nos dedicamos a realizar las pruebas de tiempo de validación sin ninguna retransmisión a nivel tres, en las que se mide el tiempo entre que el validador envía la petición de Identificación a varios dispositivos usuarios y recibía todas estas identificaciones por parte de los usuarios.

El esquema general que encontramos en la aplicación autoescaner sin retransmisiones es similar al de la Fig. 5.2 con la diferencia que en este caso hay más nodos usuarios, con lo que conlleva problemas de acceso al medio, colisiones, etc.

En este caso, al disponer de varios nodos usuarios, el tiempo de recogida de información dependerá del número de nodos y el tiempo de backoff de cada dispositivo. Por ello se nos hace complicado extraer una expresión matemática que caracterice el tiempo de transferencia de información.

Las pruebas realizadas son iguales a las descritas en el apartado 5.1.2. Se realiza 300 repeticiones de peticiones de identificación con un número de nodos usuarios que van de 1 a 13.

A continuación presentaremos los resultados obtenidos de las pruebas realizadas.

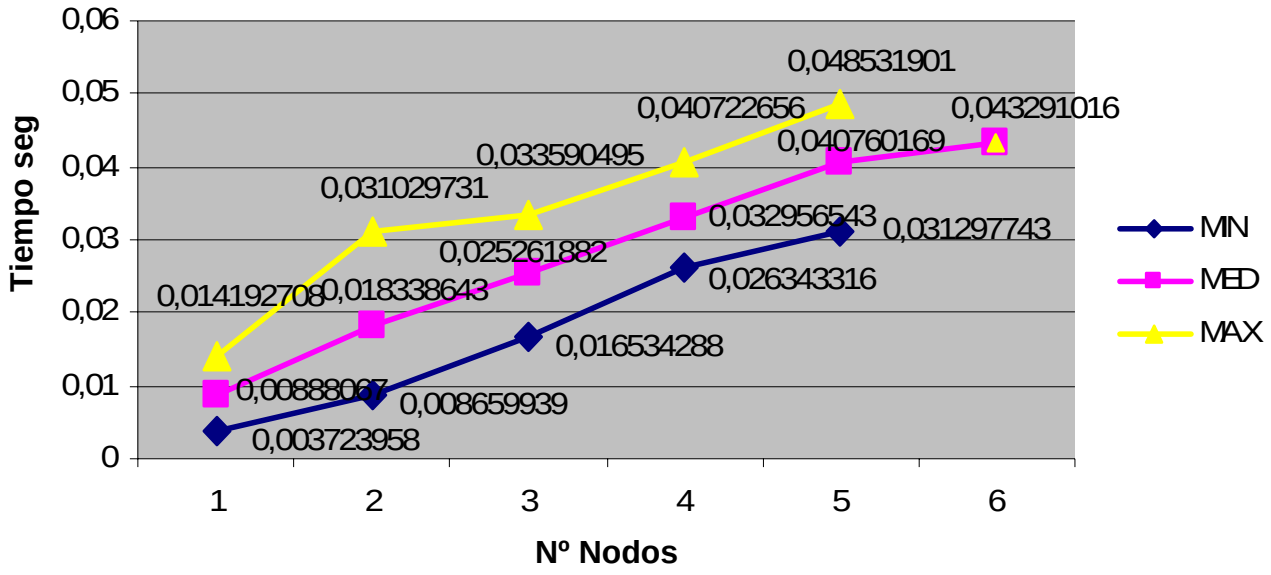


Fig.5.3 Tiempo mínimo, medio y máximo, configuración por defecto.

Como se observa en la Fig. 5.3, a medida que se van introduciendo nodos identificadores, el tiempo que emplea el nodo validador para recibir la información de todos los nodos identificadores es mayor. Si observamos la gráfica, deducimos que tiene un comportamiento lineal, es decir, a medida que introducimos más nodos el tiempo aumenta de una forma lineal.
En el nodo 6 las tres líneas se unirían en un solo punto debido que al realizar los pruebas con seis nodos solo conseguimos recoger la información de los 6 nodos en una de las 300 interacciones que hicimos, por lo tanto el tiempo medio, máximo y mínimo son iguales.
Igualmente observamos que al introducir 7 nodos identificadores, el nodo validador no consigue recoger la información de los siete nodos por lo que no podemos mostrar los tiempos de ésta.
Para el caso de un nodo identificador, lo más probable es que el tiempo medio de recogida de datos sea el valor central del rango posible de tiempos. En el anexo I se demuestra como hemos llegado a esa conclusión.

5.2.3. Probabilidad de perdida de información (ninguna retransmisión)

En la Fig. 5.4, vemos los resultados obtenidos al calcular la probabilidad de pérdida de información con los motes usando su configuración por defecto, como vemos hemos ido aumentado el numero de nodos hasta medir una probabilidad de pérdida total igual al 100%.

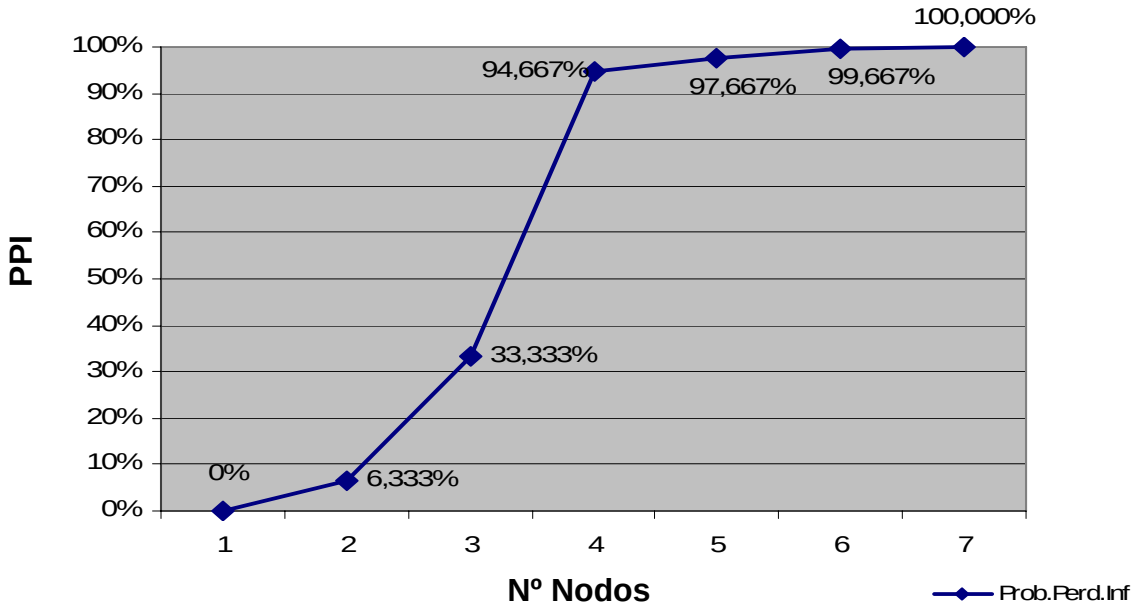


Fig.5.4 Probabilidad de pérdida de información sin Retx

En la Fig. 5.4 se muestra la probabilidad de pérdida de información. Como se puede ver, ésta aumenta considerablemente, llegando a ser del 100% con siete nodos identificadores. Esta pérdida de información es producida por estar el medio ocupado. Al analizar los resultados obtenidos optamos por seguir dos alternativas para conseguir disminuir la probabilidad de perdida de información sufrida en este momento.

La primera solución que aplicamos fue la de modificar el tiempo de backoff de los dispositivos identificadores. Esto lo realizamos aumentando, en el fichero de configuración del chip CC2420 (anexo D), el rango de posibles valores de tiempo de la función backoff, de esta manera los nodos disponen de más intervalos para transmitir y por lo tanto mas oportunidades de transmitir. Al realizar los cambios comprobamos, que por alguna circunstancia que desconocemos, los cambios realizados no surtían el efecto deseado, ya que los resultados que obteníamos no eran coherentes, por lo que optemos por seguir una segunda alternativa.

La segunda solución que aplicamos consiste en aplicar retransmisiones de paquetes a nivel 3. Introducimos el mecanismo de robustez de reconocimiento (ack) para que los dispositivos usuarios se asegurasen de que su trama ha sido procesada por el nodo validador. Al aplicar estos cambios comprobamos que estos surtían efecto puesto que la probabilidad de pérdida de información decrecía a medida que se iba aumentando el número de retransmisiones. En los siguientes apartados, se muestra los resultados obtenidos al aplicar el mecanismo de repeticiones.

5.2.4. Tiempo de recogida de información (con retransmisiones)

En la siguiente gráfica se puede observar los tiempos medios obtenidos al realizar las pruebas aplicando el mecanismo de retransmisiones.

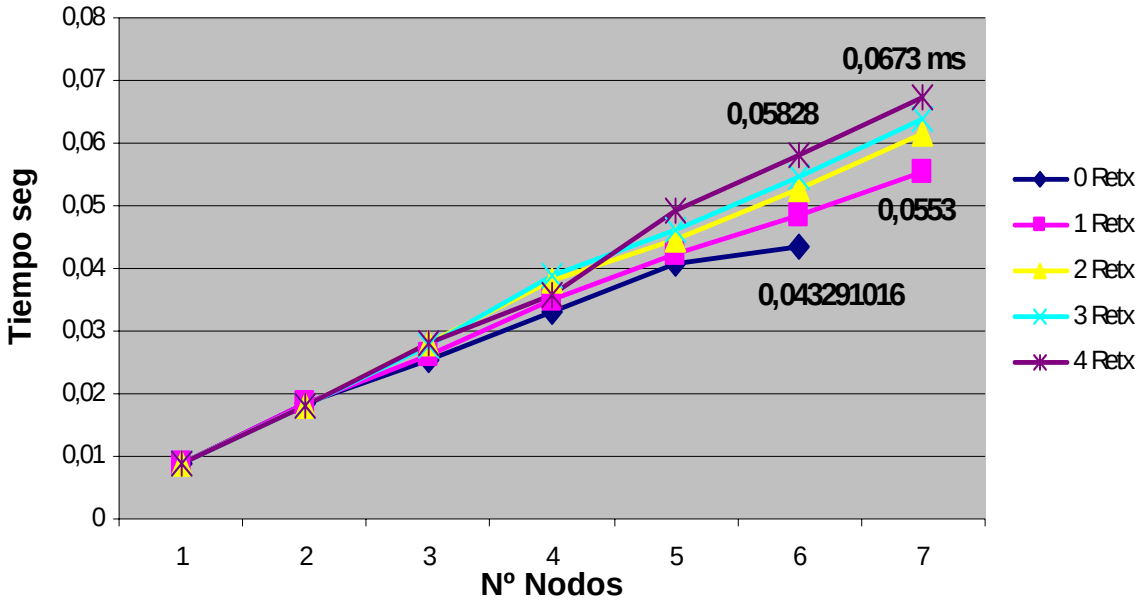


Fig.5.5 Tiempos medios según el nº de Retx (0-4)

De la Fig. 5.5 podemos deducir que el hecho de introducir un máximo de cuatro retransmisiones a nivel 3 no supone un retraso elevado a la hora de recopilar la información de todos los nodos, si comparamos, para un recopilación de 6 identificadores, se observa que la diferencia entre aplicar 4 retransmisiones y ninguna es, en media, de 14,99 ms o en el caso de 7 identificadores, la diferencia es de 12 ms entre introducir 4 retransmisiones o solamente una.

En cambio, como veremos en el apartado 5.2.5, al aplicar retransmisiones baja notablemente la probabilidad de pérdida de información.

Viendo estos resultados, decidimos introducir más nodos de identificación y aumentar el número máximo de retransmisiones. Para realizar esta segunda parte de las pruebas, se tuvo que pedir prestado otro kit de motes MICAz a los proyectistas de la UPC de Barcelona (Campus Nord).

En la siguiente figura se observa los resultados obtenidos al introducir estas variaciones.

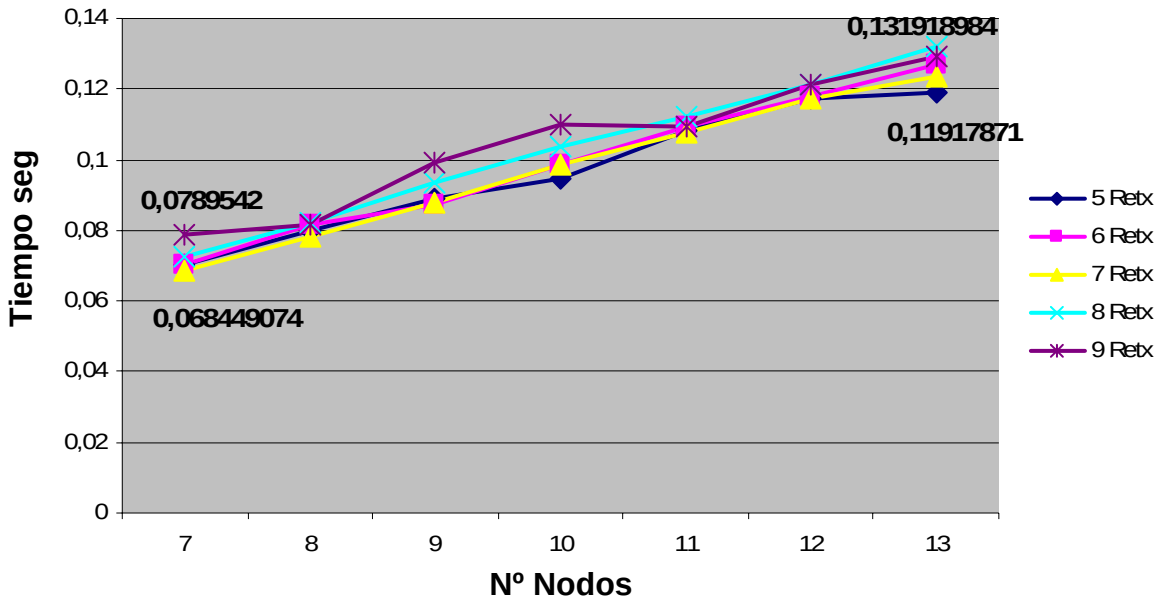


Fig.5.6 Tiempos medios según el nº de Retx (5-9)

Como en el caso anterior, observamos que la diferencia de tiempos entre aplicar un máximo de 9 retransmisiones y un máximo de 5 es, en media, de 10,5 ms para el caso de 7 nodos y de 12,8 ms para el caso de 13 nodos, por lo que este pequeño retraso es una desventaja despreciable delante de la ventaja del uso de retransmisiones a la hora de disminuir la probabilidad de perdida de información como vemos en el siguiente apartado.

5.2.5. Probabilidad de perdida de información (con retransmisiones)

Como hemos comentado en los aparatados anteriores, la introducción de retransmisiones a nivel implica un pequeño aumento del tiempo a la hora de recoger la información de todos los nodos identificadores pero supone una disminución contundente de la probabilidad de pérdida de información.
En los siguientes gráficos se muestra los resultados de probabilidad de perdida obtenidos de las pruebas realizadas.

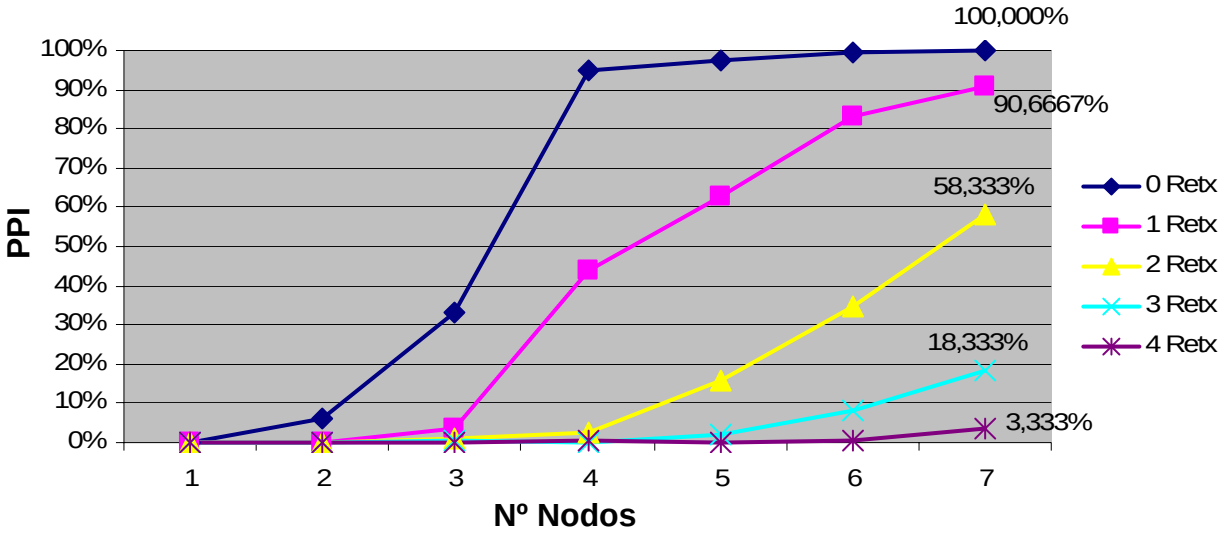


Fig.5.7 Probabilidad de pérdida de información con Retx (0-4)

En la Fig. 5.7 se observa la reducción sustancial de la probabilidad de pérdida de información con el aumento del número máximo de retransmisiones a nivel 3. Como dato referente, destacamos la reducción de pérdida de datos para el caso de siete nodos identificadores. Se puede observar que, sin ningún tipo de retransmisiones, la pérdida sufrida es del 100 %. En cambio, aplicando un máximo de 4 retransmisiones las perdidas se reducían sustancialmente hasta un 3,3 %.

Viendo los resultados obtenidos anteriormente, decidimos aumentar el número de nodos identificadores y de retransmisiones máxima para comprobar si el comportamiento del sistema con un número elevado de nodos es satisfactorio,

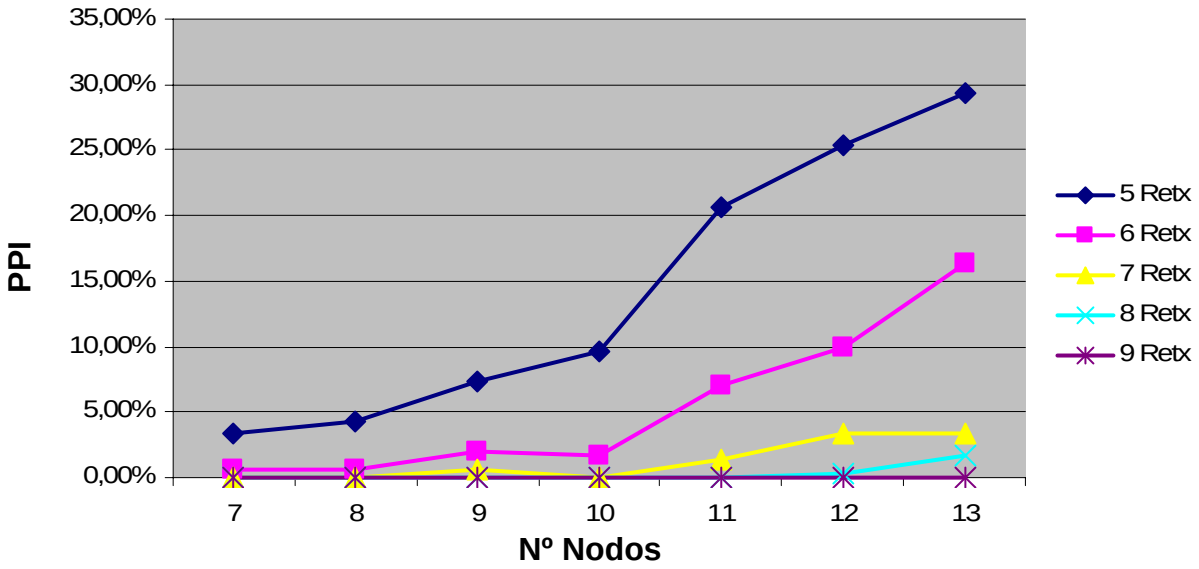


Fig.5.8 Probabilidad de pérdida de información con Retx (5-9)

Como se puede comprobar en la Fig. 5.8, se consiguió una transmisión entre el nodo validador y 13 nodos identificadores sin ninguna perdida de información durante 300 transmisiones aplicando un máximo de 9 retransmisiones.
Viendo los resultados obtenidos, podemos afirmar que la utilización de la tecnología RFID es ventajosa puesto que permite el escaneo de 13 identificaciones con una PPI despreciable y en un tiempo medio igual a 130 ms. Basta comparar con el tiempo que supondría la misma operación mediante código de barras para cuantificar las ventajas de la tecnología RFID. Además, el uso de RFID permite evitar la necesidad de presencia humana en la operación indicada.

CAPITULO 6. CONCLUSIONES Y LINEAS FUTURAS

6.1. Conclusiones

A lo largo de este documento hemos tratado de analizar las posibilidades de la tecnología RFID. Hoy en día, este es un tema de creciente interés, dadas las ventajas potenciales de RFID.

Lo que hemos pretendido, mediante la realización de este trabajo, es realizar una prueba de concepto. Para llevar acabo este propósito, se ha realizado un estudio de los dispositivos inalámbricos de la casa Crossbow. A su vez se han desarrollado dos aplicaciones (Control de acceso y Autoescáner) como prueba de concepto de la tecnología RFID.
La tecnología RFID, es una tecnología madura y fiable, aplicable a múltiples aplicaciones que actualmente presenta una clara tendencia a suplantar el código de barras. La tecnología RFID, se sirve de los servicios proporcionados por la plataforma ZigBee para establecer redes de corto alcance y baja tasa de transferencia ideales para este tipo de aplicaciones.

La aplicación *Control de Acceso,* aprovecha las ventajas ofrecidas por la tecnología RFID convirtiéndose en una herramienta útil en sistemas de seguridad.

La aplicación *Autoescáner* es una herramienta idónea y con multitud de usos para el control de los productos en cualquier empresa u organización. En este trabajo, nos hemos decantado por destinar este tipo de aplicación como caja registradora de productos ubicada en todo comercio.

Tras la realización de las distintas pruebas, se ha observado que el tiempo medio de recopilación de distintos identificadores por un nodo controlador sigue un comportamiento lineal. Este depende del número de nodos y el número de retransmisiones empleadas. De la misma forma, se ha comprobado que el tiempo medio en recopilar la información de 13 nodos identificándose al mismo tiempo no supera los 131 ms, lo cual hace que ésta tecnología sea idóneo para éste tipo de aplicaciones.

Como todo sistema de comunicaciones, la tecnología RFID esta expuesta a interferencias, obstáculos, etc, que pueden ocasionar perdida de información. Sin embargo, para la aplicación autoscáner, el principal problema es el acceso al medio por parte de múltiples dispositivos que intentan transmitir simultáneamente. Para subsanar éste problema he tenido que aplicar retransmisiones a nivel 3 y mecanismos de robustez como ack’s. Hemos comprobado que para recibir las IDs de 13 nodos transmitiendo a la vez, se han tenido que aplicar un máximo de 9 retransmisiones con un retardo adicional despreciable.

6.2. Líneas Futuras

A partir del trabajo realizado, se tendría que reforzar las aplicaciones implementadas debido a que éstas se han realizado como prueba de concepto de la tecnología RFID. Las aplicaciones podrían ser mejoradas con un sin fin de funcionalidades que toda aplicación comercial contiene (p.ej: gestión base de datos, registro de usuarios, etc.).

Quedan por estudiar ciertos aspectos, que mantienen o bien abren nuevas líneas de investigación. Una nueva línea de investigación es la implementación de capas superiores, es decir, estos dispositivos solo tienen implementado la capa física y la de enlace (ZigBee y MICAz), a partir de aquí hay un vacío sustancial. Este año, nuevos proyectistas realizarán un protocolo de direccionamiento, basado en IPv6[13], que se implementara por encima de estos dispositivos.

Otro campo por estudiar, sería la implementación de mecanismos de cifrado de datos. El protocolo ZigBee, ya utiliza un mecanismo de cifrado en la capa MAC, pero sería conveniente estudiar mecanismos de cifrado dedicados a estos dispositivos, debido a la importancia de la información que pueden llevar estos dispositivos (DNI como RFID o llave de casa como RFID).

[13] **IPv6** es la versión 6 del Protocolo de Internet (Internet Protocol), un estándar del nivel de red encargado de dirigir y encaminar los paquetes a través de una red.

IMPLICACIONES MEDIOMENTALES

El trabajo desarrollado no presenta impacto medioambiental alguno, sin ser pretenciosos, es totalmente inocuo ya que las aplicaciones creadas no generan residuo alguno. Se podría decir que es un trabajo limpio y respetuoso con el medioambiente.

No obstante, los dispositivos inalámbricos necesitan de una batería portátil para realizar las transmisiones radioeléctricas. Este tipo de baterías, en nuestro caso pilas de tipo AA, deben ser depositadas en el contenedor correspondiente para su reciclado o correcto almacenamiento, ya que estas pueden causar estragos en los ecosistemas donde se depositen. Incluso pueden llegar a afectar a los seres humanos.

Las pilas ofrecen una fuente de energía cómoda y portátil que forma parte integral de numerosos productos electrónicos modernos. No obstante, tan solo son capaces de almacenar una pequeña parte de la energía que se emplea en su fabricación. Suponen una carga para el medio ambiente tanto en su fabricación como en su eliminación. Algunos de los componentes de las pilas, como el cadmio, mercurio o zinc, pueden ser gravemente perjudiciales para el medio ambiente y ocasionar enfermedades a los seres humanos (dolores gastrointestinales) si no reciben el tratamiento adecuado.

Por el contrario, cabe destacar que ZigBee, IEEE 802.15.4, está diseñado para maximizar el tiempo de vida de las baterías. El mayor consumo que realizan estos dispositivos es a la hora de transmitir información, las redes RFID se caracterizan por sus escasas y breves transmisiones lo que conlleva una vida útil de batería muy elevada (2 años o más).

BIBLIOGRAFIA

[1] ¿Que es RFID?, Historia RFID, 2005 IDentika
http://www.identika.com.mx/LA/privacidad.htm

[2] Estructura de los códigos de barras según el estándar EAN
http://es.wikipedia.org/wiki/EAN

[3] Implantes en humanos de etiquetas RFID.
http://news.bbc.co.uk/2/hi/technology/3697940.stm

[4] RFID para esperma y óvulos.
http://www.newscientist.com/article.ns?id=dn7209

[5] Implantación de la tecnología RFID en Tráfico y posicionamiento.
http://www.roadbeacon.com/

[6] Didac Mediavilla Urra *"Comparativa de implementaciones de protocolos reactivos de encaminamiento en redes Ad-hoc"* trabajo de final de carrera, 24 febrero 2006.

[7] Tipos de modulaciones en el 802.15.4
http://es.wikipedia.org/wiki/DSSS

[8] RF Transmisores
http://www.chipcon.com/

[9] MOTE-KIT2400 Datasheet
http://www.xbow.com/Products/Product_pdf_files/Wireless_pdf/MICAz_Kit_Datasheet.pdf

[10] CrossBow
http://www.xbow.com

[11] Mote MICAz Datasheet
http://www.cmt-gmbh.de/MICAz.pdf

[12] TinyOS
http://www.cs.utah.edu/~coop/embedded/tinyos.html

[13] Serial Interface Board MIB510 Datasheet
http://www.cmt-gmbh.de/MIB510CA.pdf

[14] Ethernet Interface Board MIB600 Dataheet
http://www.cmt-gmbh.de/MIB600CA.pdf

[15] ZigBee/IEEE 802.15.4 Summary
http://www.eecs.berkeley.edu/~csinem/academic/publications/zigbee.pdf

[16] MTS/MDA Mote Sensor and DAQ Manual
http://www.xbow.com/Support/Support_pdf_files/MTS-MDA_Series_Users_Manual.pdf

[17] Carlos López Truchero *"Aplicaciones de sensado: emulación de RFID activo y medición en tiempo real"* trabajo final de carrera, 29 Enero 2006.

[18] Getting Started Guide, Crossbow
http://www.xbow.com/Support/Support_pdf_files/Getting_Started_Guide.pdf

[19] Desastre por error en el control de acceso a un estadio de fútbol
http://www.lukor.com/deportes/noticias/0503/28113622.htm

[20] Especificación de la estructura de trama del 802.15.4
http://www.chipcon.com/index.cfm?kat_id=1&action=faq&faq_id=340

ANEXO A. TINYOS

1. Instalación TinyOs para Windows

Requisitos para la instalación:

- El CD-ROM de CrossBow que contiene las herramientas de soporte de TinyOS
- Ordenador con SO Microsoft Windows (XP, 2000, NT), con 1 GB de espacio libre en la unidad de disco de destino de la instalación
- Tener los programas: WinZip, Adobe Acrobat y Programmers Notepad (disponible gratuitamente en http://www.pnotepad.org/)

La instalación de TinyOS para Windows es sencilla. En primer lugar tenemos que instalar el archivo ejecutable *tinyos-1.1.0-1is.exe,* el cual nos instalará la plataforma básica de desarrollo Cygwin, las librerías nesC de TinyOS y el compilador nesC.

El entorno de instalación de este ejecutable es muy intuitivo, guiado en todos sus pasos. Este archivo lo podemos encontrar en:

- El CD de CrossBow, dentro de la carpeta *TinyOS Install.*
- En el servidor de descargas de la plataforma Windows de www.tinyos.net que se encuentra en la siguiente página de internet http://www.tinyos.net/dist-1.1.0/tinyos/windows/

Después de la instalación de la plataforma TinyOS, la consola Cygwin y compilador de nesC, concentrados en el archivo *tinyos-1.1.0-1is.exe,* procederemos a la instalación del paquete rpm que nos actualizará a la última versión del compilador nesC. El nombre del paquete es el nesc-1.1.2a-1.cygwin.i386.rpm y lo podemos encontrar en el mismo servidor de descargas anterior (http://www.tinyos.net/dist-1.1.0/tinyos/windows/).

La instalación de este paquete se tiene que efectuar desde la consola Cygwin, dentro del directorio donde se encuentre el paquete rpm. La secuencia a escribir es la siguiente:

rpm -- force --ignoreos -Uvh nesc-1.1.2a-1.cygwin.i386.rpm

Finalmente instalaremos la actualización de TinyOS que deseemos, como requisito necesita tener instalado el anterior paquete. A enero de 2006 la actualización más reciente es la 1.1.15 de diciembre de 2005, y para descargarla sólo tenemos que ir al servidor de descargas de Windows ya indicado por duplicado anteriormente. El link a presionar es el tinyos-1.1.15Dec2005cvs-1.cygwin.noarch.rpm.

La instalación de este segundo paquete rpm se instalará de semejante forma al anterior paquete:

rpm -- force --ignoreos -Uvh tinyos-1.1.15Dec2005cvs-1.cygwin.noarch.rpm

La aplicación para los motes MICAz realizadas en este trabajo han sido realizadas con las versiones de TinyOS:

- tinyos-1.1.7July2004cvs-2.cygwin.noarch.rpm
- tinyos-1.1.10Jan2005cvs-1.cygwin.noarch.rpm

La carpeta xbow del CD de CrossBow dentro de la carpeta *TinyOS Updates* la copiaremos en la carpeta de nuestro disco C:\tinyos\cygwin\opt\tinyos-1.x\contrib. De esta forma podremos comenzar a trabajar desde el directorio C:\tinyos\cygwin\opt\tinyos-1.x\contrib\xbow\apps que hemos copiado, programando nuestras aplicaciones en nesC.

Con el comando **rpm –qa** en la consola Cygwin podemos comprobar la correcta instalación de TinyOS y los paquetes de actualización.

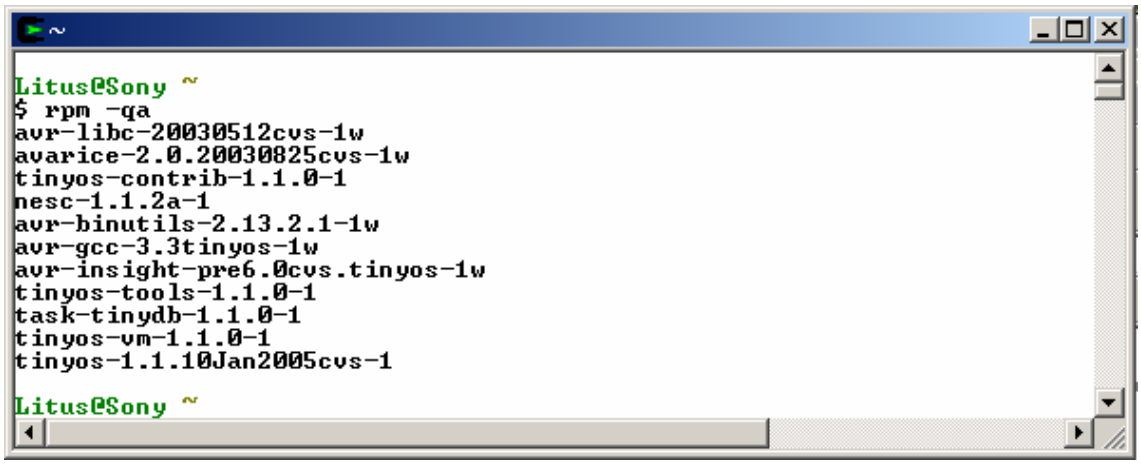


Fig. 1.1 Ejemplo del comando de comprobación “rpm -qa”. En el apreciamos los tres paquetes instalados:

1. tinyos-1.1.0-1is.exe
2. nesc-1.1.2a-1.cygwin.i386.rpm
3. tinyos-1.1.15Dec2005cvs-1.cygwin.noarch.rpm

2. Directorio TinyOS

Para trabajar con el directorio de TinyOS utilizaremos la consola Cygwin que con los comandos de linux nos permite explorar el directorio.

El directorio de carpetas de TinyOS está compuesto de numerosas carpetas y archivos. Este apartado pretende mostrar el contenido de las carpetas más importantes de TinyOS y su estructura.

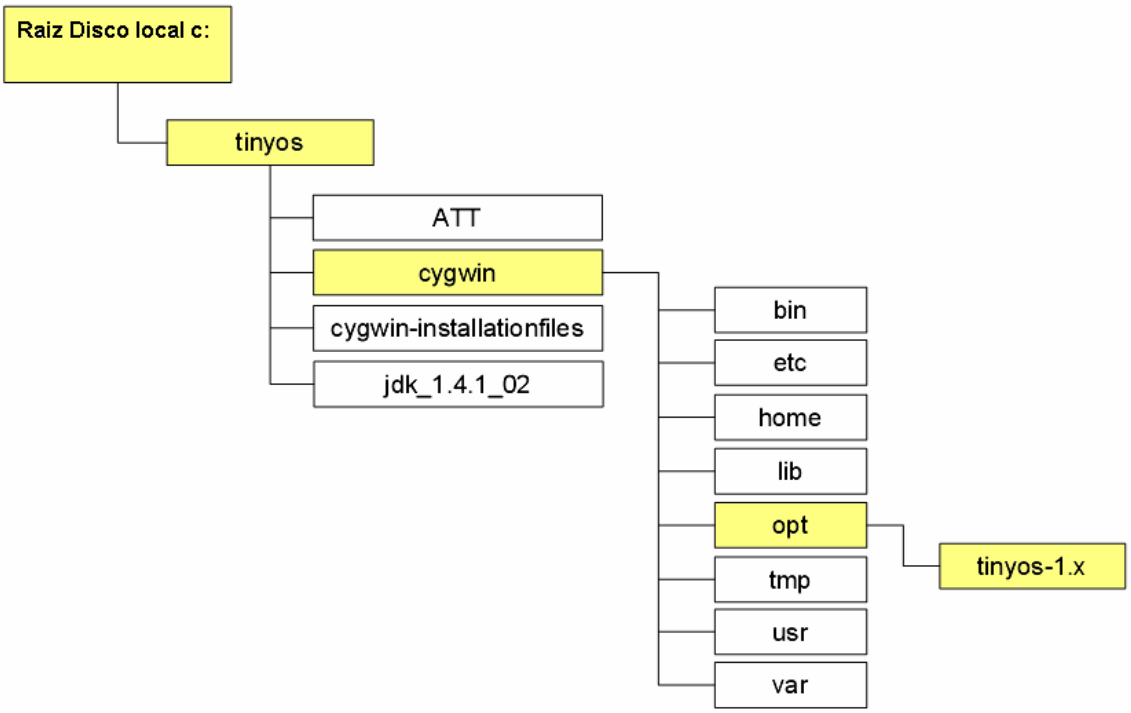


Fig. 2.1 Estructura principal desde la raíz del directorio de TinyOS

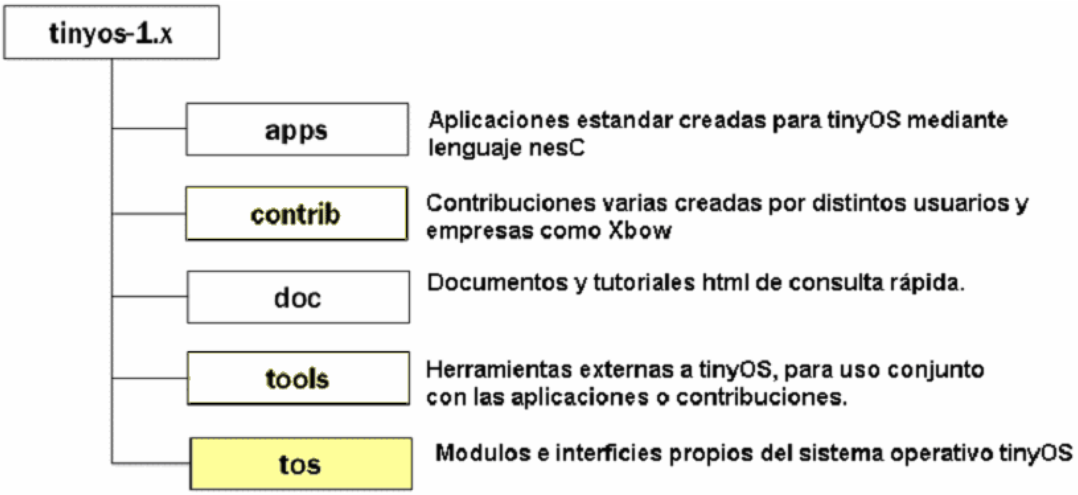


Fig. 2.2 Estructura y contenido del directorio tinyos-1.x

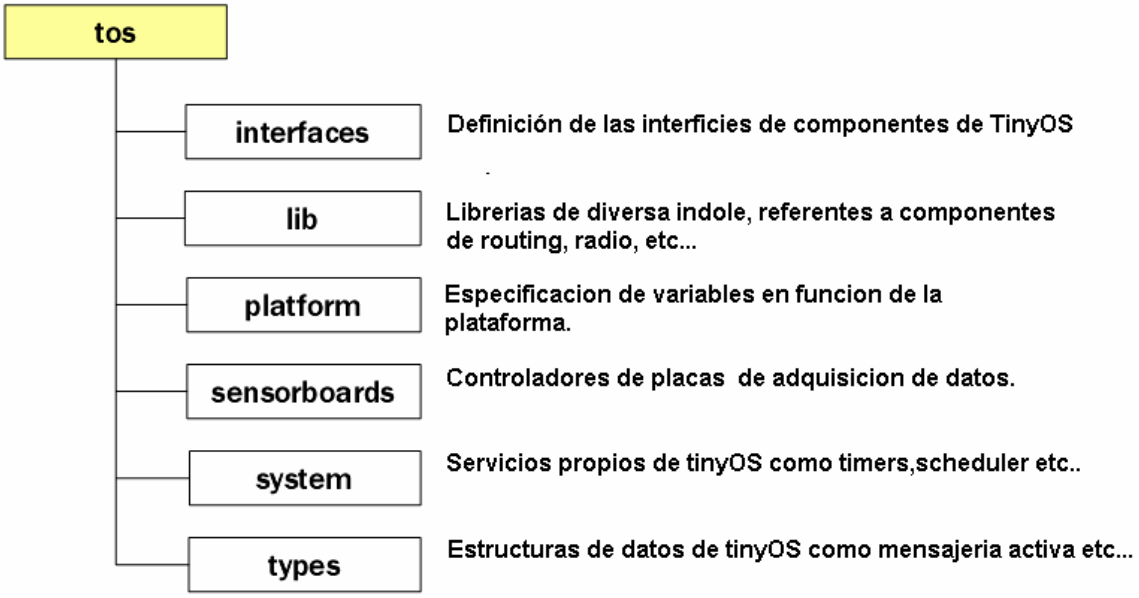


Fig. 2.3 Estructura y contenido del directorio tos dentro de la carpeta tinyos-1.x

3. Compilación de aplicaciones en TinyOS

Las nuevas versiones de TinyOS soportan las plataformas hardware de los motes MICAz, MICA2 y MICA2DOT. La sintaxis que se utiliza en la consola Cygwin para compilar las aplicaciones es la siguiente:

make <plataforma> install.<dirección dispositivo> <programador>,<puerto>
ó
make <plataforma> reinstall <programador>,<puerto>

La diferencia entre install o reinstall está detalla más abajo.

- Install.<n>: Compila la aplicación para la plataforma seleccionada, puede configurar el Identificador de nodo (<n>) del mote y lo programa.

- Reinstall.<n>: Puede también configurar el Identificador de nodo y instala el programa precompilado, no recompila. Esta opción es sustancialmente más rápida.

El nombre usado por <plataforma> se encuentra en la tabla 3.1.

Tabla 3.1 Lista de las plataformas hardware soportadas (<plataforma>)

Procesador / Plataforma Radio	<plataforma> usada
MICAZ (MPR2400 series)	micaz
MICA (MPR3x0 series)	mica
MICA2 (MPR4x0 series)	mica2
MICA2DOT (MPR5x0 series)	mica2dot

Esta sintaxis genérica se acompañará de los argumentos inherentes a la placa programadora que se use. Las placas compatibles son las siguientes:

- La placa programadora puerto paralelo MIB500
- La placa programadora puerto serie MIB510CA
- La placa programadora puerto USB MIB520CA
- La placa programadora Ethernet MIB600CA

La tabla 3.2 nos muestra el nombre usado para cada tipo de placa.

Tabla 3.2 Lista de placas programadoras (<programador>)

Placa MIB	**<programador> usado**
MIB600	eprb
MIB510	mib510
MIB520	mib520
MIB500	Por defecto, no necesario información adicional

En esta sección sólo se explicará la sintaxis de compilación de aplicaciones para las placas programadoras incluidas en el MOTE-KIT2400 de CrossBow:

3.1. Placa programadora puerto serie MIB510CA

Si utilizamos esta placa programadora tenemos que añadir la siguiente línea de comando:

mib510,com<x>

Donde <x> es el número del puerto serie conectado a la MIB510. Antes de ejecutar dicho comando cerciorase que vuestro sistema tiene habilitado el puerto.

Ejemplo:

Este ejemplo sirve para programar un mote MICAz desde una MIB510 conectada al puerto serie COM1 del PC.

make micaz install mib510,com1

3.2. Placa programadora Ethernet MIB600CA

En cambio, si se utiliza la placa programadora MIB600CA, es necesario asignarle una dirección IP. Cada dispositivo conectado tiene que tener una dirección IP única. Esta dirección es usada como referencia específica de la unidad. Cada conexión TCP y cada datagrama UDP es definido por una dirección IP y un número de puerto.

1. Instalar el programa de Lantronix (DeviceInstaller36.zip) de la capeta *Miscellaneous* del CD-ROM facilitado con el kit. Este programa también puede ser descargado desde la página http://www.lantronix.com/
2. Conectar la MIB600CA a la red con un cable Ethernet RJ-45 y conectarla a la corriente usando el transformador facilitado en el kit.
3. Clic en el botón inicio de la barra de tareas y seleccionar Inicio > Programas > Lantronix > Device Installer. Device Installer nos muestra una ventana.
4. Clic en el botón buscar (en inglés *search*) para ver la lista de dispositivos encontrados con la correspondiente dirección IP.
5. Mirar y encontrar la dirección física (MAC) de nuestra MIB600CA (p.e. 00-20-4A-63-47-31), una vez localizada la seleccionamos. Clic en *Assign IP* (Asignación IP) y seguimos las instrucciones. Asignamos una dirección IP dentro del rango de nuestra tarjeta de red.
6. Una vez asignada la dirección IP de la MIB600CA, el comando que tenemos que escribir en la consola Cygwin para programar el mote es:

make <platform> install eprb,<Dirección_IP>

Ejemplo:

Este ejemplo sirve para programar un mote MICAz desde una MIB560 con una dirección IP 192.168.100.123.

make micaz install eprb, 192.168.100.123

ANEXO B. MANUAL DE SERIALFORWARDER

SerialForwarder es un programa escrito en Java, y es usado para leer los paquetes de datos del puerto serie del PC y reenviarlos sobre la conexión del puerto servidor, cosa que permite que otros programas se puedan comunicar con la red de sensores. Actúa como una puerta de acceso (gateway). SerialForwarder no muestra los datos de los paquetes, pero tiene contadores de paquetes leídos y escritos en la conexión TCP.

Una vez ejecutado, SerialForwarder escucha conexiones de clientes en un puerto TCP determinado: por defecto el puerto 9001 para la placa programadora MIB510CA y el puerto 10002 para la placa MIB600CA. Y remite hacia los clientes todos los mensajes TinyOS del puerto serie o el puerto Ethernet, y viceversa.

Se puede ejecutar SerialForwarder de dos maneras:

1. **Ejecutando SerialForwarder.exe ubicado en el directorio de Windows *C:\Program Files\Surge-View***

Para la placa MIB510/MIB520 el puerto servidor debe de ser el 9001 (por defecto).

Clicamos en el botón *Stop Server* (entonces se convierte en *Start Server*). Seguidamente editamos el recuadro de *Mote Communications* de la siguiente manera:

serial@COM<#>:<plataforma>

Donde,

<#> = 1, 2, 3, 4, etc. es el puerto serie COM1, COM2, COM3, COM4, etc. Del PC que está conectado a la placa programadora

<plataforma> = Es la tasa en baudios de la plataforma utilizada

O, en caso de utilizar la placa MIB600 editaremos de la siguiente manera el campo *Mote Communications*:

network@<Dirección_IP_MIB600>:10002

Donde,

<Dirección_IP_MIB600> = Es la dirección IP de la placa MIB600

Finalmente, hacer clic en el botón *Start Server* (se convierte a *Stop Server*)

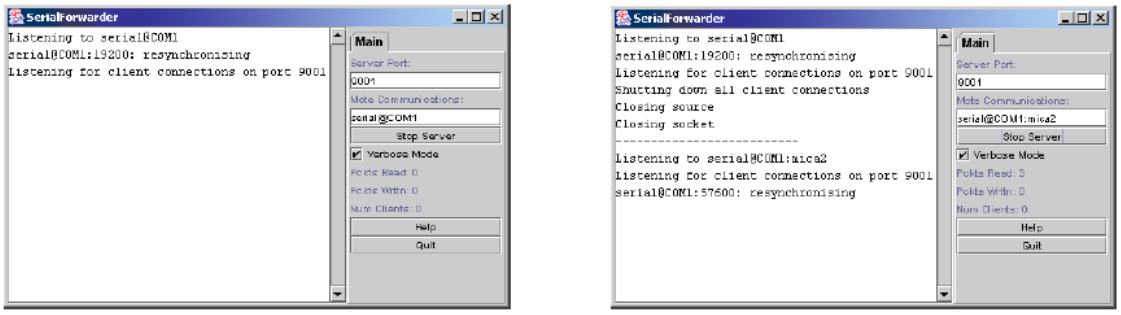


Fig. B.1 (Izquierda) La aplicación Java SerialForwarder al ejecutarse. (Derecha) Cuando clicamos en *Start Server* ya configurados el puerto serie, la tasa de transferencia y el puerto servidor. La última línea (la tercera) debe de contener la palabra "resynchronizing"

2. Con la consola de comandos MS-DOS

Primero debemos de ejecutar la consola de comandos de Windows clicando en el botón Inicio de la barra de tareas: **Inicio > Programas > Accesorios > Símbolo de Sistema**

Después nos situamos en el directorio C:\Program Files\Surge-View usando la siguiente sentencia:

cd ..\..\ Program Files\Surge-View

Por último, iniciamos SerialForwarder con la información –comm port:

SerialForwarder –comm serial@COM<#>:<plataforma>

Donde,

<#> = 1, 2, 3, 4, etc. es el puerto serie COM1, COM2, COM3, COM4, etc. Del PC que está conectado a la placa programadora

<plataforma> = Es la tasa en baudios de la plataforma utilizada

O, en caso de utilizar la placa MIB600 editaremos de la siguiente manera el campo *Mote Communications*:

SerialForwarder –comm network@<Dirección_IP_MIB600>:10002

Donde,

<Dirección_IP_MIB600> = Es la dirección IP de la placa MIB600

ANEXO C. CONFIGURACIÓN MIB600CA

En esta sección del anexo del trabajo se pretende mostrar los diferentes pasos que se deben de seguir para configurar correctamente la placa programadora MIB600CA.

En primer lugar procederemos a la instalación del programa creado por LanTronix, DeviceInstaller 3.6, que se encuentra en el directorio *Miscellaneous* del CD-ROM subministrado en el MOTE-KIT2400. O bien descargado de la página http://www.lantronix.com/

Después de la instalación ejecutamos el programa DeviceInstaller 3.6 y presionamos el botón de búsqueda (*search*). El programa detectará nuestra MIB600CA y procederemos a la asignación de una dirección IP (botón *Assign IP*) tal y como muestra la siguiente captura:

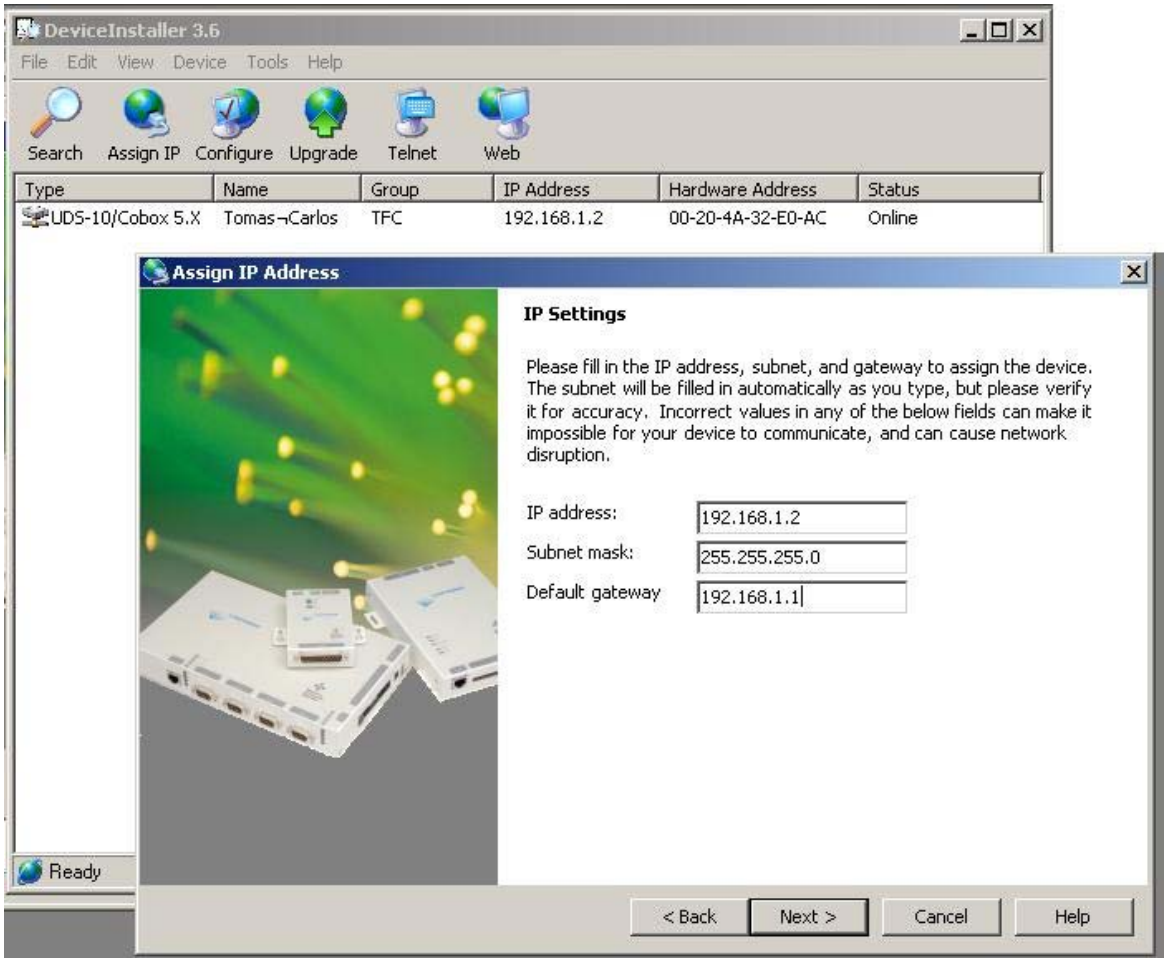


Fig. C.1 Captura donde se muestra la ventana de asignación de la dirección IP de la placa programadora MIB600

Después de la asignación de la dirección IP configuraremos la placa programadora para su correcto funcionamiento con la plataforma MICAz. Para ello nos dirigimos al menú *Options* dentro del botón *Device*:

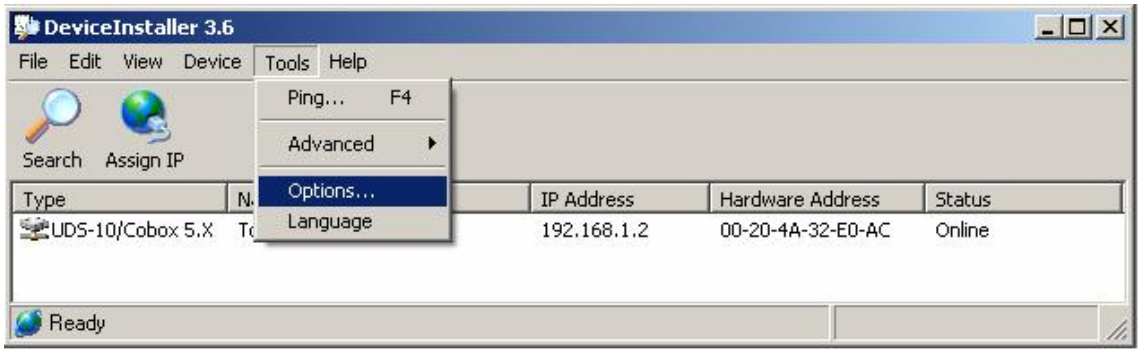


Fig. C.1 Captura donde se muestra la ventana de asignación de la dirección IP de la placa programadora MIB600

En el menú que se nos aparece, entramos en el submenú de los puertos llamado *Ports*:

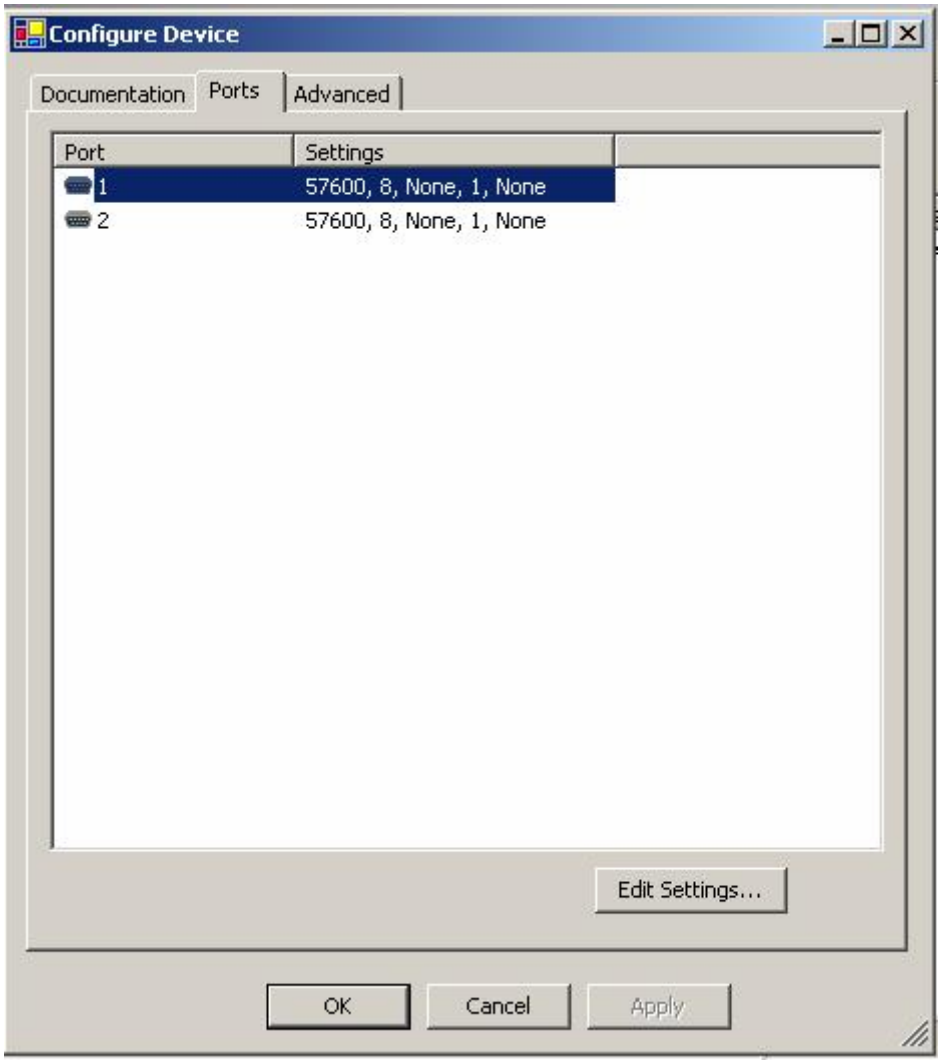


Fig. C.2 Captura de pantalla que muestra el menú *Ports* de la ventana *Device Properties*

Finalmente, procedemos a configurarlos de la siguiente manera:

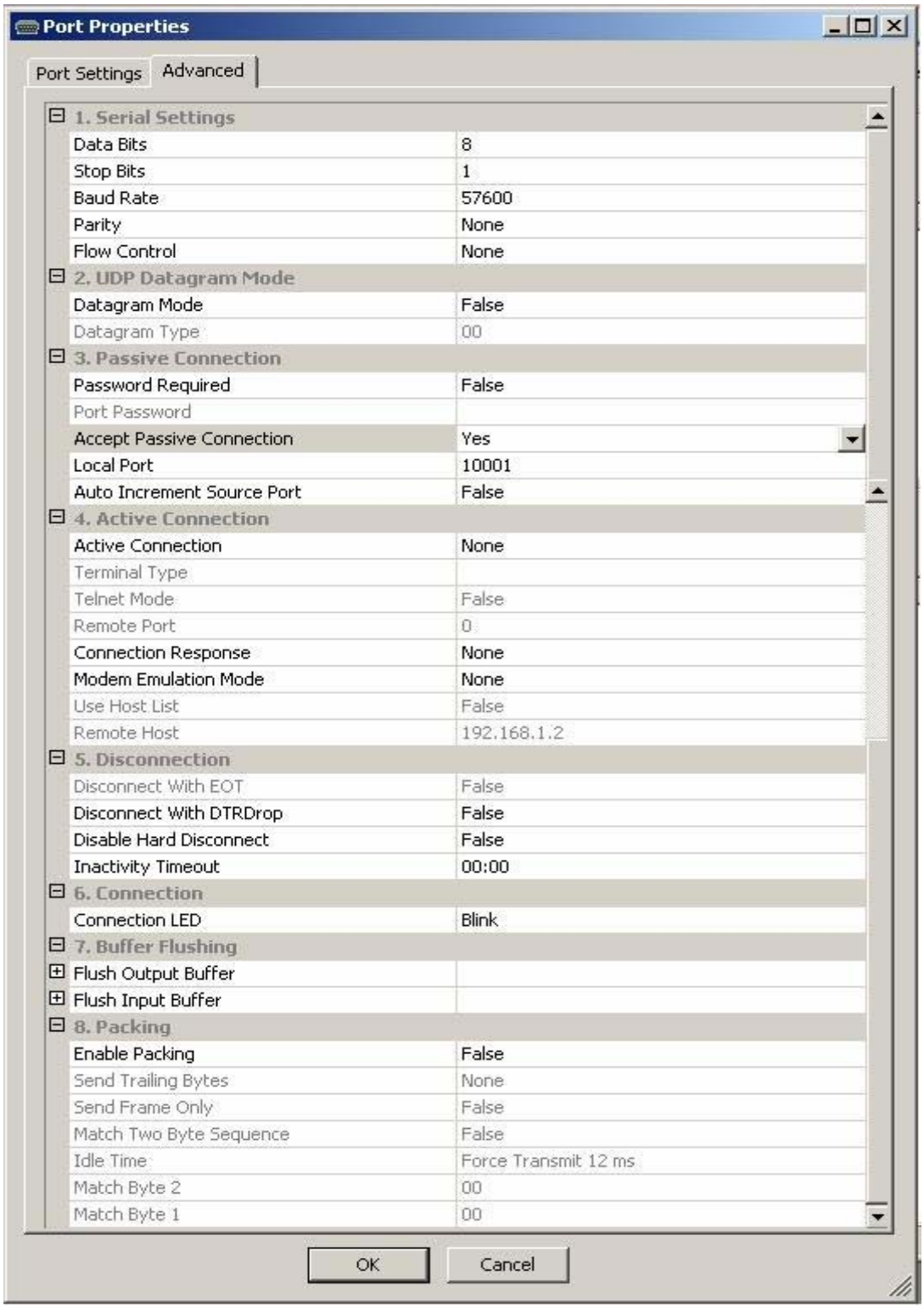


Fig. C.2 Captura de la ventana *Port Properties,* en la pestaña *Advanced*

Anexo D: Librería de constantes referentes a chip de comunicaciones CC2420

Para poder acceder a estos parámetros previamente debemos estudiar el datasheet [3] que nos habla de la parte radio. Posteriormente, buscamos en las librerías radio algún fichero relevante hasta encontrar el fichero llamado cc2420const.h en donde se definen los parámetros por defecto. Dentro nos encontramos con 2 clases de constantes, las que podemos modificar de forma directa y los que podemos modificar de una forma más indirecta, haciendo un pequeño análisis de las más relevantes.

De forma directa:

CC2420_DEF_RFPOWER: Nos permite variar la potencia de emisión de nuestra parte radio en 5 niveles distintos que oscilan entre 0dBm y -25dBm

CC2420_DEF_CHANNEL: Nos permite definir el canal por el que vamos a transmitir. Su valor va entre el canal 11 y el canal 26 es utilizado en combinación a CC2420_DEF_PRESET. Así el canal final se define de la siguiente forma:

Frecuencia(Mhz)=CC2420_DEF_PRESET + (CC2420_DEF_CHANNEL-11) $\times 5$

CC2420_ACK_DELAY: Tiempo en μs que puede tardar en confirmarse un paquete antes de declararlo inválido.

De forma indirecta:

También existen otra clase de variables que únicamente nos indican la posición en memoria y la posición del bit que modifica la configuración, éstos son una serie de registros que se manejan como tales.

Haremos una pequeña descripción de los que consideramos más importantes y su valor por defecto, para el resto será necesario consultar el datasheet que proporciona chipcon:

MDMCTRL1 y MDMCTRL0: son dos registros con diversas variables de configuración que nos permiten modificar los campos que se envían dentro de los paquetes a nivel de enlace, tales como el tamaño de preámbulo, tiempos entre paquetes, así como aspectos de compatibilidad y modulación. También nos permite determinar que nodo se comportará como coordinador PAN.

TXCTRL: es considerado importante ya que nos informa, entre otras cosas, de la potencia de transmisión que vamos a utilizar, de todas formas este parámetro hemos tenido ocasión de configurarlo de forma directa.

FSCTRL entre otros usos también nos permite configurar la frecuencia a la que transmitimos, en la forma directa esto se hace con las constantes CC2420_DEF_PRESET y CC2420_DEF_CHANNEL.

El resto de registros no tienen funciones que sean consideradas relevantes en nuestro proyecto, una extensión de las que hemos comentado y sus valores por defecto estan puestos en el anexo K y el datasheet del fabricante [3] .

Como bien decimos en la memoria existe un fichero de configuración y posicionamiento de registros llamado *CC2420Const.h* .

Para consultar la configuración que se utiliza por defecto se ha de consultar el fichero *CC2420ControlM.nc* que aquí es expuesto, como se podrá observar se utilizan mascaras para definir el valor de cada registro.

Existen mas ficheros referentes al chipset radio CC2420 pero no entraremos en ellos dada su extensión.

1. CC2420const.h

```
#ifndef _CC2420CONST_H
#define _CC2420CONST_H

// times for the CC2420 in microseconds
enum {
  CC2420_TIME_BIT = 4,
  CC2420_TIME_BYTE = CC2420_TIME_BIT << 3,
  CC2420_TIME_SYMBOL = 16
};

#ifndef CC2420_DEF_RFPOWER
#define CC2420_DEF_RFPOWER          0X01
#endif

#ifndef CC2420_DEF_CHANNEL
#define CC2420_DEF_CHANNEL        26  //channel select
#endif

enum {
  CC2420_MIN_CHANNEL =              11,
  CC2420_MAX_CHANNEL =              26
};

#define CC2420_DEF_PRESET           2405  //freq select

#define CC2420_DEF_FCF_LO          0x08
#define CC2420_DEF_FCF_HI          0x01  // without ACK
#define CC2420_DEF_FCF_HI_ACK      0x21  // with ACK
#define CC2420_DEF_FCF_TYPE_BEACON 0x00
#define CC2420_DEF_FCF_TYPE_DATA   0x01
#define CC2420_DEF_FCF_TYPE_ACK    0x02
#define CC2420_DEF_FCF_BIT_ACK     5
#define CC2420_DEF_BACKOFF         500
#define CC2420_SYMBOL_TIME         16 // 2^4
// 20 symbols make up a backoff period
// 10 jiffy's make up a backoff period
// due to timer overhead, 30.5us is close enough to 32us per 2 symbols
#define CC2420_SYMBOL_UNIT         10

// delay when waiting for the ack
#ifndef CC2420_ACK_DELAY
#define CC2420_ACK_DELAY           75
#endif

#ifndef CC2420_XOSC_TIMEOUT
#define CC2420_XOSC_TIMEOUT 200     //times to chk if CC2420 crystal is on
#endif

#define CC2420_SNOP            0x00
```

```
#define CC2420_SXOSCON         0x01
#define CC2420_STXCAL          0x02
#define CC2420_SRXON           0x03
#define CC2420_STXON           0x04
#define CC2420_STXONCCA        0x05
#define CC2420_SRFOFF          0x06
#define CC2420_SXOSCOFF        0x07
#define CC2420_SFLUSHRX        0x08
#define CC2420_SFLUSHTX        0x09
#define CC2420_SACK            0x0A
#define CC2420_SACKPEND        0x0B
#define CC2420_SRXDEC          0x0C
#define CC2420_STXENC          0x0D
#define CC2420_SAES            0x0E
#define CC2420_MAIN            0x10
#define CC2420_MDMCTRL0        0x11
#define CC2420_MDMCTRL1        0x12
#define CC2420_RSSI            0x13
#define CC2420_SYNCWORD        0x14
#define CC2420_TXCTRL          0x15
#define CC2420_RXCTRL0         0x16
#define CC2420_RXCTRL1         0x17
#define CC2420_FSCTRL          0x18
#define CC2420_SECCTRL0        0x19
#define CC2420_SECCTRL1        0x1A
#define CC2420_BATTMON         0x1B
#define CC2420_IOCFG0          0x1C
#define CC2420_IOCFG1          0x1D
#define CC2420_MANFIDL         0x1E
#define CC2420_MANFIDH         0x1F
#define CC2420_FSMTC           0x20
#define CC2420_MANAND          0x21
#define CC2420_MANOR           0x22
#define CC2420_AGCCTRL         0x23
#define CC2420_AGCTST0         0x24
#define CC2420_AGCTST1         0x25
#define CC2420_AGCTST2         0x26
#define CC2420_FSTST0          0x27
#define CC2420_FSTST1          0x28
#define CC2420_FSTST2          0x29
#define CC2420_FSTST3          0x2A
#define CC2420_RXBPFTST        0x2B
#define CC2420_FSMSTATE        0x2C
#define CC2420_ADCTST          0x2D
#define CC2420_DACTST          0x2E
#define CC2420_TOPTST          0x2F
#define CC2420_RESERVED        0x30
#define CC2420_TXFIFO          0x3E
#define CC2420_RXFIFO          0x3F

#define CC2420_RAM_SHORTADR    0x16A
#define CC2420_RAM_PANID       0x168
#define CC2420_RAM_IEEEADR     0x160
#define CC2420_RAM_CBCSTATE    0x150
#define CC2420_RAM_TXNONCE     0x140
#define CC2420_RAM_KEY1        0x130
#define CC2420_RAM_SABUF       0x120
#define CC2420_RAM_RXNONCE     0x110
#define CC2420_RAM_KEY0        0x100
#define CC2420_RAM_RXFIFO      0x080
#define CC2420_RAM_TXFIFO      0x000

// MDMCTRL0 Register Bit Positions
#define CC2420_MDMCTRL0_FRAME        13  // 0 : reject reserved frame types, 1 = accept
#define CC2420_MDMCTRL0_PANCRD       12  // 0 : not a PAN coordinator
#define CC2420_MDMCTRL0_ADRDECODE    11  // 1 : enable address decode
#define CC2420_MDMCTRL0_CCAHIST      8   // 3 bits (8,9,10) : CCA hysteris in db
#define CC2420_MDMCTRL0_CCAMODE      6   // 2 bits (6,7)    : CCA trigger modes
#define CC2420_MDMCTRL0_AUTOCRC      5   // 1 : generate/chk CRC
#define CC2420_MDMCTRL0_AUTOACK      4   // 1 : Ack valid packets
#define CC2420_MDMCTRL0_PREAMBL      0   // 4 bits (0..3): Preamble length

// MDMCTRL1 Register Bit Positions
#define CC2420_MDMCTRL1_CORRTHRESH   6   // 5 bits (6..10) : correlator threshold
#define CC2420_MDMCTRL1_DEMOD_MODE   5   // 0: lock freq after preamble match, 1:
continous udpate
#define CC2420_MDMCTRL1_MODU_MODE    4   // 0: IEEE 802.15.4
```

```
#define CC2420_MDMCTRL1_TX_MODE      2   // 2 bits (2,3) : 0: use buffered TXFIFO
#define CC2420_MDMCTRL1_RX_MODE      0   // 2 bits (0,1) : 0: use buffered RXFIFO

// RSSI Register Bit Positions
#define CC2420_RSSI_CCA_THRESH       8   // 8 bits (8..15) : 2's compl CCA threshold

// TXCTRL Register Bit Positions
#define CC2420_TXCTRL_BUFCUR         14  // 2 bits (14,15) : Tx mixer buffer bias
current
#define CC2420_TXCTRL_TURNARND       13  // wait time after STXON before xmit
#define CC2420_TXCTRL_VAR            11  // 2 bits (11,12) : Varactor array settings
#define CC2420_TXCTRL_XMITCUR        9   // 2 bits (9,10)  : Xmit mixer currents
#define CC2420_TXCTRL_PACUR          6   // 3 bits (6..8)  : PA current
#define CC2420_TXCTRL_PADIFF         5   // 1: Diff PA, 0: Single ended PA
#define CC2420_TXCTRL_PAPWR          0   // 5 bits (0..4): Output PA level

// Mask for the CC2420_TXCTRL_PAPWR register for RF power
#define CC2420_TXCTRL_PAPWR_MASK (0x1F << CC2420_TXCTRL_PAPWR)

// RXCTRL0 Register Bit Positions
#define CC2420_RXCTRL0_BUFCUR        12  // 2 bits (12,13) : Rx mixer buffer bias
current
#define CC2420_RXCTRL0_HILNAG        10  // 2 bits (10,11) : High gain, LNA current
#define CC2420_RXCTRL0_MLNAG          8  // 2 bits (8,9)   : Med gain, LNA current
#define CC2420_RXCTRL0_LOLNAG         6  // 2 bits (6,7)   : Lo gain, LNA current
#define CC2420_RXCTRL0_HICUR          4  // 2 bits (4,5)   : Main high LNA current
#define CC2420_RXCTRL0_MCUR           2  // 2 bits (2,3)   : Main med  LNA current
#define CC2420_RXCTRL0_LOCUR          0  // 2 bits (0,1)   : Main low LNA current

// RXCTRL1 Register Bit Positions
#define CC2420_RXCTRL1_LOCUR         13  // Ref bias current to Rx bandpass filter
#define CC2420_RXCTRL1_MIDCUR        12  // Ref bias current to Rx bandpass filter
#define CC2420_RXCTRL1_LOLOGAIN      11  // LAN low gain mode
#define CC2420_RXCTRL1_MEDLOGAIN     10  // LAN low gain mode
#define CC2420_RXCTRL1_HIHGM          9  // Rx mixers, hi gain mode
#define CC2420_RXCTRL1_MEDHGM         8  // Rx mixers, hi gain mode
#define CC2420_RXCTRL1_LNACAP         6  // 2 bits (6,7) Selects LAN varactor array
setting
#define CC2420_RXCTRL1_RMIXT          4  // 2 bits (4,5) Receiver mixer output current
#define CC2420_RXCTRL1_RMIXV          2  // 2 bits (2,3) VCM level, mixer feedback
#define CC2420_RXCTRL1_RMIXCUR        0  // 2 bits (0,1) Receiver mixer current

// FSCTRL Register Bit Positions
#define CC2420_FSCTRL_LOCK           14 // 2 bits (14,15) # of clocks for synch
#define CC2420_FSCTRL_CALDONE        13 // Read only, =1 if cal done since freq synth
turned on
#define CC2420_FSCTRL_CALRUNING      12 // Read only, =1 if cal in progress
#define CC2420_FSCTRL_LOCKLEN        11 // Synch window pulse width
#define CC2420_FSCTRL_LOCKSTAT       10 // Read only, = 1 if freq synthesizer is loced
#define CC2420_FSCTRL_FREQ            0 // 10 bits, set operating frequency

// SECCTRL0 Register Bit Positions
#define CC2420_SECCTRL0_PROTECT       9 // Protect enable Rx fifo
#define CC2420_SECCTRL0_CBCHEAD       8 // Define 1st byte of CBC-MAC
#define CC2420_SECCTRL0_SAKEYSEL      7 // Stand alone key select
#define CC2420_SECCTRL0_TXKEYSEL      6 // Tx key select
#define CC2420_SECCTRL0_RXKEYSEL      5 // Rx key select
#define CC2420_SECCTRL0_SECM          2 // 2 bits (2..4) # of bytes in CBC-MAX auth
field
#define CC2420_SECCTRL0_SECMODE       0 // Security mode

// SECCTRL1 Register Bit Positions
#define CC2420_SECCTRL1_TXL           8 // 7 bits (8..14) Tx in-line security
#define CC2420_SECCTRL1_RXL           0 // 7 bits (0..7)  Rx in-line security

// BATTMON  Register Bit Positions
#define CC2420_BATTMON_OK             6 // Read only, batter voltage OK
#define CC2420_BATTMON_EN             5 // Enable battery monitor
#define CC2420_BATTMON_VOLT           0 // 5 bits (0..4) Battery toggle voltage

// IOCFG0 Register Bit Positions
#define CC2420_IOCFG0_FIFOPOL        10 // Fifo signal polarity
#define CC2420_IOCFG0_FIFOPPOL        9 // FifoP signal polarity
#define CC2420_IOCFG0_SFD             8 // SFD signal polarity
#define CC2420_IOCFG0_CCAPOL          7 // CCA signal polarity
#define CC2420_IOCFG0_FIFOTHR         0 // 7 bits, (0..6) # of Rx bytes in fifo to trg
fifop
```

```
// IOCFG1 Register Bit Positions
#define CC2420_IOCFG1_HSSD            10 // 2 bits (10,11) HSSD module config
#define CC2420_IOCFG1_SFDMUX           5 // 5 bits (5..9)  SFD multiplexer pin settings
#define CC2420_IOCFG1_CCAMUX           0 // 5 bits (0..4)  CCA multiplexe pin settings

// Current Parameter Arrray Positions
enum{
 CP_MAIN = 0,
 CP_MDMCTRL0,
 CP_MDMCTRL1,
 CP_RSSI,
 CP_SYNCWORD,
 CP_TXCTRL,
 CP_RXCTRL0,
 CP_RXCTRL1,
 CP_FSCTRL,
 CP_SECCTRL0,
 CP_SECCTRL1,
 CP_BATTMON,
 CP_IOCFG0,
 CP_IOCFG1
} ;

// STATUS Bit Posititions
#define CC2420_XOSC16M_STABLE 6
#define CC2420_TX_UNDERFLOW   5
#define CC2420_ENC_BUSY              4
#define CC2420_TX_ACTIVE      3
#define CC2420_LOCK           2
#define CC2420_RSSI_VALID     1

#endif /* _CC2420CONST_H */
```

2. CC2420ControlM.nc

```
includes byteorder;

module CC2420ControlM {
  provides {
    interface SplitControl;
    interface CC2420Control;
  }
  uses {
    interface StdControl as HPLChipconControl;
    interface HPLCC2420 as HPLChipcon;
    interface HPLCC2420RAM as HPLChipconRAM;
    interface HPLCC2420Interrupt as CCA;
  }
}
implementation
{

  enum {
    IDLE_STATE = 0,
    INIT_STATE,
    INIT_STATE_DONE,
    START_STATE,
    START_STATE_DONE,
    STOP_STATE,
  };

  uint8_t state = 0;
  norace uint16_t gCurrentParameters[14];

   /**************************************************************************
   * SetRegs
   *   - Configure CC2420 registers with current values
   *   - Readback 1st register written to make sure electrical connection OK
   **************************************************************************/
  bool SetRegs(){
    uint16_t data;
```

```
    call HPLChipcon.write(CC2420_MAIN,gCurrentParameters[CP_MAIN]);
    call HPLChipcon.write(CC2420_MDMCTRL0, gCurrentParameters[CP_MDMCTRL0]);
    data = call HPLChipcon.read(CC2420_MDMCTRL0);
    if (data != gCurrentParameters[CP_MDMCTRL0]) return FALSE;

    call HPLChipcon.write(CC2420_MDMCTRL1, gCurrentParameters[CP_MDMCTRL1]);
    call HPLChipcon.write(CC2420_RSSI, gCurrentParameters[CP_RSSI]);
    call HPLChipcon.write(CC2420_SYNCWORD, gCurrentParameters[CP_SYNCWORD]);
    call HPLChipcon.write(CC2420_TXCTRL, gCurrentParameters[CP_TXCTRL]);
    call HPLChipcon.write(CC2420_RXCTRL0, gCurrentParameters[CP_RXCTRL0]);
    call HPLChipcon.write(CC2420_RXCTRL1, gCurrentParameters[CP_RXCTRL1]);
    call HPLChipcon.write(CC2420_FSCTRL, gCurrentParameters[CP_FSCTRL]);

    call HPLChipcon.write(CC2420_SECCTRL0, gCurrentParameters[CP_SECCTRL0]);
    call HPLChipcon.write(CC2420_SECCTRL1, gCurrentParameters[CP_SECCTRL1]);
    call HPLChipcon.write(CC2420_IOCFG0, gCurrentParameters[CP_IOCFG0]);
    call HPLChipcon.write(CC2420_IOCFG1, gCurrentParameters[CP_IOCFG1]);

    call HPLChipcon.cmd(CC2420_SFLUSHTX);    //flush Tx fifo
    call HPLChipcon.cmd(CC2420_SFLUSHRX);

    return TRUE;

  }

  task void taskInitDone() {
    signal SplitControl.initDone();
  }

  task void taskStopDone() {
    signal SplitControl.stopDone();
  }

  task void PostOscillatorOn() {
    //set freq, load regs
    SetRegs();
    call CC2420Control.setShortAddress(TOS_LOCAL_ADDRESS);
    call CC2420Control.TuneManual(((gCurrentParameters[CP_FSCTRL] << CC2420_FSCTRL_FREQ)
& 0x1FF) + 2048);
    atomic state = START_STATE_DONE;
    signal SplitControl.startDone();
  }

  /***************************************************************************
   * Init CC2420 radio:
   *
   ***************************************************************************/
  command result_t SplitControl.init() {

    uint8_t _state = FALSE;

    atomic {
      if (state == IDLE_STATE) {
    state = INIT_STATE;
    _state = TRUE;
      }
    }
    if (!_state)
      return FAIL;

    call HPLChipconControl.init();

    // Set default parameters
    gCurrentParameters[CP_MAIN] = 0xf800;
    gCurrentParameters[CP_MDMCTRL0] = ((0 << CC2420_MDMCTRL0_ADRDECODE) |
       (2 << CC2420_MDMCTRL0_CCAHIST) | (3 << CC2420_MDMCTRL0_CCAMODE)  |
       (1 << CC2420_MDMCTRL0_AUTOCRC) | (2 << CC2420_MDMCTRL0_PREAMBL));

    gCurrentParameters[CP_MDMCTRL1] = 20 << CC2420_MDMCTRL1_CORRTHRESH;

    gCurrentParameters[CP_RSSI] =     0xE080;
    gCurrentParameters[CP_SYNCWORD] = 0xA70F;
    gCurrentParameters[CP_TXCTRL] = ((1 << CC2420_TXCTRL_BUFCUR) |
       (1 << CC2420_TXCTRL_TURNARND) | (3 << CC2420_TXCTRL_PACUR) |
       (1 << CC2420_TXCTRL_PADIFF) | (CC2420_DEF_RFPOWER << CC2420_TXCTRL_PAPWR));

    gCurrentParameters[CP_RXCTRL0] = ((1 << CC2420_RXCTRL0_BUFCUR) |
```

```
      (2 << CC2420_RXCTRL0_MLNAG) | (3 << CC2420_RXCTRL0_LOLNAG) |
      (2 << CC2420_RXCTRL0_HICUR) | (1 << CC2420_RXCTRL0_MCUR) |
      (1 << CC2420_RXCTRL0_LOCUR));

    gCurrentParameters[CP_RXCTRL1]  = ((1 << CC2420_RXCTRL1_LOLOGAIN) |
      (1 << CC2420_RXCTRL1_HIHGM) |  (1 << CC2420_RXCTRL1_LNACAP) |
      (1 << CC2420_RXCTRL1_RMIXT) |  (1 << CC2420_RXCTRL1_RMIXV)  |
      (2 << CC2420_RXCTRL1_RMIXCUR));

    gCurrentParameters[CP_FSCTRL]   = ((1 << CC2420_FSCTRL_LOCK) |
      ((357+5*(CC2420_DEF_CHANNEL-11)) << CC2420_FSCTRL_FREQ));

    gCurrentParameters[CP_SECCTRL0] = ((1 << CC2420_SECCTRL0_CBCHEAD) |
      (1 << CC2420_SECCTRL0_SAKEYSEL)  | (1 << CC2420_SECCTRL0_TXKEYSEL) |
      (1 << CC2420_SECCTRL0_SECM));

    gCurrentParameters[CP_SECCTRL1] = 0;
    gCurrentParameters[CP_BATTMON]  = 0;

    // set fifop threshold to greater than size of tos msg,
    // fifop goes active at end of msg
    gCurrentParameters[CP_IOCFG0]   = (((127) << CC2420_IOCFG0_FIFOTHR) |
        (1 <<CC2420_IOCFG0_FIFOPPOL)) ;

    gCurrentParameters[CP_IOCFG1]   =  0;

    atomic state = INIT_STATE_DONE;
    post taskInitDone();
    return SUCCESS;
  }


  command result_t SplitControl.stop() {
    result_t ok;
    uint8_t _state = FALSE;

    atomic {
      if (state == START_STATE_DONE) {
    state = STOP_STATE;
    _state = TRUE;
      }
    }
    if (!_state)
      return FAIL;

    call HPLChipcon.cmd(CC2420_SXOSCOFF);
    ok = call CCA.disable();
    ok &= call HPLChipconControl.stop();

    TOSH_CLR_CC_RSTN_PIN();
    ok &= call CC2420Control.VREFOff();
    TOSH_SET_CC_RSTN_PIN();

    if (ok)
      post taskStopDone();

    atomic state = INIT_STATE_DONE;
    return ok;
  }

/****************************************************************************
 * Start CC2420 radio:
 * -Turn on 1.8V voltage regulator, wait for power-up, 0.6msec
 * -Release reset line
 * -Enable CC2420 crystal,          wait for stabilization, 0.9 msec
 *
 ****************************************************************************/

  command result_t SplitControl.start() {
    result_t status;
    uint8_t _state = FALSE;

    atomic {
      if (state == INIT_STATE_DONE) {
    state = START_STATE;
    _state = TRUE;
      }
```

```
    }
    if (!_state)
      return FAIL;

    call HPLChipconControl.start();
    //turn on power
    call CC2420Control.VREFOn();
    // toggle reset
    TOSH_CLR_CC_RSTN_PIN();
    TOSH_wait();
    TOSH_SET_CC_RSTN_PIN();
    TOSH_wait();
    // turn on crystal, takes about 860 usec,
    // chk CC2420 status reg for stablize
    status = call CC2420Control.OscillatorOn();

    return status;
  }

  /*****************************************************************************
   * TunePreset
   * -Set CC2420 channel
   * Valid channel values are 11 through 26.
   * The channels are calculated by:
   *  Freq = 2405 + 5(k-11) MHz for k = 11,12,...,26
   * chnl requested 802.15.4 channel
   * return Status of the tune operation
   *****************************************************************************/
  command result_t CC2420Control.TunePreset(uint8_t chnl) {
    int fsctrl;
    uint8_t status;

    fsctrl = 357 + 5*(chnl-11);
    gCurrentParameters[CP_FSCTRL] = (gCurrentParameters[CP_FSCTRL] & 0xfc00) | (fsctrl
<< CC2420_FSCTRL_FREQ);
    status = call HPLChipcon.write(CC2420_FSCTRL, gCurrentParameters[CP_FSCTRL]);
    // if the oscillator is started, recalibrate for the new frequency
    // if the oscillator is NOT on, we should not transition to RX mode
    if (status & (1 << CC2420_XOSC16M_STABLE))
      call HPLChipcon.cmd(CC2420_SRXON);
    return SUCCESS;
  }

  /*****************************************************************************
   * TuneManual
   * Tune the radio to a given frequency. Frequencies may be set in
   * 1 MHz steps between 2400 MHz and 2483 MHz
   *
   * Desiredfreq The desired frequency, in MHz.
   * Return Status of the tune operation
   *****************************************************************************/
  command result_t CC2420Control.TuneManual(uint16_t DesiredFreq) {
    int fsctrl;
    uint8_t status;

    fsctrl = DesiredFreq - 2048;
    gCurrentParameters[CP_FSCTRL] = (gCurrentParameters[CP_FSCTRL] & 0xfc00) | (fsctrl
<< CC2420_FSCTRL_FREQ);
    status = call HPLChipcon.write(CC2420_FSCTRL, gCurrentParameters[CP_FSCTRL]);
    // if the oscillator is started, recalibrate for the new frequency
    // if the oscillator is NOT on, we should not transition to RX mode
    if (status & (1 << CC2420_XOSC16M_STABLE))
      call HPLChipcon.cmd(CC2420_SRXON);
    return SUCCESS;
  }

  /*****************************************************************************
   * Get the current frequency of the radio
   */
  command uint16_t CC2420Control.GetFrequency() {
    return ((gCurrentParameters[CP_FSCTRL] & (0x1FF << CC2420_FSCTRL_FREQ))+2048);
  }

  /*****************************************************************************
   * Get the current channel of the radio
   */
  command uint8_t CC2420Control.GetPreset() {
```

```
    uint16_t _freq = (gCurrentParameters[CP_FSCTRL] & (0x1FF << CC2420_FSCTRL_FREQ));
    _freq = (_freq - 357)/5;
    _freq = _freq + 11;
    return _freq;
  }

  /***************************************************************************
   * TxMode
   * Shift the CC2420 Radio into transmit mode.
   * return SUCCESS if the radio was successfully switched to TX mode.
   ***************************************************************************/
  async command result_t CC2420Control.TxMode() {
    call HPLChipcon.cmd(CC2420_STXON);
    return SUCCESS;
  }

  /***************************************************************************
   * TxModeOnCCA
   * Shift the CC2420 Radio into transmit mode when the next clear channel
   * is detected.
   *
   * return SUCCESS if the transmit request has been accepted
   ***************************************************************************/
  async command result_t CC2420Control.TxModeOnCCA() {
   call HPLChipcon.cmd(CC2420_STXONCCA);
   return SUCCESS;
  }

  /***************************************************************************
   * RxMode
   * Shift the CC2420 Radio into receive mode
   ***************************************************************************/
  async command result_t CC2420Control.RxMode() {
    call HPLChipcon.cmd(CC2420_SRXON);
    return SUCCESS;
  }

  /***************************************************************************
   * SetRFPower
   * power = 31 => full power    (0dbm)
   *          3 => lowest power  (-25dbm)
   * return SUCCESS if the radio power was successfully set
   ***************************************************************************/
  command result_t CC2420Control.SetRFPower(uint8_t power) {
    gCurrentParameters[CP_TXCTRL] = (gCurrentParameters[CP_TXCTRL] &
(~CC2420_TXCTRL_PAPWR_MASK)) | (power << CC2420_TXCTRL_PAPWR);
    call HPLChipcon.write(CC2420_TXCTRL,gCurrentParameters[CP_TXCTRL]);
    return SUCCESS;
  }

  /***************************************************************************
   * GetRFPower
   * return power seeting
   ***************************************************************************/
  command uint8_t CC2420Control.GetRFPower() {
    return (gCurrentParameters[CP_TXCTRL] & CC2420_TXCTRL_PAPWR_MASK); //rfpower;
  }

  async command result_t CC2420Control.OscillatorOn() {
    uint16_t i;
    uint8_t status;

    i = 0;

    // uncomment to measure the startup time from
    // high to low to high transitions
    // output "1" on the CCA pin
#ifdef CC2420_MEASURE_OSCILLATOR_STARTUP
      call HPLChipcon.write(CC2420_IOCFG1, 31);
      // output oscillator stable on CCA pin
      // error in CC2420 datasheet 1.2: SFDMUX and CCAMUX incorrectly labelled
      TOSH_uwait(50);
#endif

    call HPLChipcon.write(CC2420_IOCFG1, 24);

    // have an event/interrupt triggered when it starts up
```

```
    call CCA.startWait(TRUE);

    // start the oscillator
    status = call HPLChipcon.cmd(CC2420_SXOSCON);   //turn-on crystal

    return SUCCESS;
  }

  async command result_t CC2420Control.OscillatorOff() {
    call HPLChipcon.cmd(CC2420_SXOSCOFF);   //turn-off crystal
    return SUCCESS;
  }

  async command result_t CC2420Control.VREFOn(){
    TOSH_SET_CC_VREN_PIN();                    //turn-on
    // TODO: JP: measure the actual time for VREF to stabilize
    TOSH_uwait(600);  // CC2420 spec: 600us max turn on time
    return SUCCESS;
  }

  async command result_t CC2420Control.VREFOff(){
    TOSH_CLR_CC_VREN_PIN();                    //turn-off
    return SUCCESS;
  }

  async command result_t CC2420Control.enableAutoAck() {
    gCurrentParameters[CP_MDMCTRL0] |= (1 << CC2420_MDMCTRL0_AUTOACK);
    return call HPLChipcon.write(CC2420_MDMCTRL0,gCurrentParameters[CP_MDMCTRL0]);
  }

  async command result_t CC2420Control.disableAutoAck() {
    gCurrentParameters[CP_MDMCTRL0] &= ~(1 << CC2420_MDMCTRL0_AUTOACK);
    return call HPLChipcon.write(CC2420_MDMCTRL0,gCurrentParameters[CP_MDMCTRL0]);
  }

  async command result_t CC2420Control.enableAddrDecode() {
    gCurrentParameters[CP_MDMCTRL0] |= (1 << CC2420_MDMCTRL0_ADRDECODE);
    return call HPLChipcon.write(CC2420_MDMCTRL0,gCurrentParameters[CP_MDMCTRL0]);
  }

  async command result_t CC2420Control.disableAddrDecode() {
    gCurrentParameters[CP_MDMCTRL0] &= ~(1 << CC2420_MDMCTRL0_ADRDECODE);
    return call HPLChipcon.write(CC2420_MDMCTRL0,gCurrentParameters[CP_MDMCTRL0]);
  }

  command result_t CC2420Control.setShortAddress(uint16_t addr) {
    addr = toLSB16(addr);
    return call HPLChipconRAM.write(CC2420_RAM_SHORTADR, 2, (uint8_t*)&addr);
  }

  async event result_t HPLChipconRAM.readDone(uint16_t addr, uint8_t length, uint8_t*
buffer) {
     return SUCCESS;
  }

  async event result_t HPLChipconRAM.writeDone(uint16_t addr, uint8_t length, uint8_t*
buffer) {
     return SUCCESS;
  }

   async event result_t CCA.fired() {
     // reset the CCA pin back to the CCA function
     call HPLChipcon.write(CC2420_IOCFG1, 0);
     post PostOscillatorOn();
     return FAIL;
   }

}
```

3. CC2420RadioM

```
includes byteorder;

module CC2420RadioM {
  provides {
    interface StdControl;
    interface SplitControl;
    interface BareSendMsg as Send;
    interface ReceiveMsg as Receive;
    interface RadioCoordinator as RadioSendCoordinator;
    interface RadioCoordinator as RadioReceiveCoordinator;
    interface MacControl;
    interface MacBackoff;
  }
  uses {
    interface SplitControl as CC2420SplitControl;
    interface CC2420Control;
    interface HPLCC2420 as HPLChipcon;
    interface HPLCC2420FIFO as HPLChipconFIFO;
    interface HPLCC2420Interrupt as FIFOP;
    interface HPLCC2420Capture as SFD;
    interface StdControl as TimerControl;
    interface TimerJiffyAsync as BackoffTimerJiffy;
    interface Random;
    interface Leds;
  }
}

implementation {
  enum {
    DISABLED_STATE = 0,
    IDLE_STATE,
    TX_STATE,
    TX_WAIT,
    PRE_TX_STATE,
    POST_TX_STATE,
    POST_TX_ACK_STATE,
    RX_STATE,
    POWER_DOWN_STATE,
    WARMUP_STATE,

    TIMER_INITIAL = 0,
    TIMER_BACKOFF,
    TIMER_ACK
  };

#define MAX_SEND_TRIES 8

  norace uint8_t countRetry;
  uint8_t stateRadio;
  norace uint8_t stateTimer;
  norace uint8_t currentDSN;
  norace bool bAckEnable;
  bool bPacketReceiving;
  uint8_t txlength;
  norace TOS_MsgPtr txbufptr;  // pointer to transmit buffer
  norace TOS_MsgPtr rxbufptr;  // pointer to receive buffer
  TOS_Msg RxBuf;    // save received messages

  volatile uint16_t LocalAddr;

  ///*********************************************************
  //* local function definitions
  //*************************** ****************************/

   void sendFailed() {
     atomic stateRadio = IDLE_STATE;
     txbufptr->length = txbufptr->length - MSG_HEADER_SIZE - MSG_FOOTER_SIZE;
     signal Send.sendDone(txbufptr, FAIL);
   }

   void flushRXFIFO() {
     call FIFOP.disable();
```

```
    call HPLChipcon.read(CC2420_RXFIFO);          //flush Rx fifo
    call HPLChipcon.cmd(CC2420_SFLUSHRX);
    call HPLChipcon.cmd(CC2420_SFLUSHRX);
    atomic bPacketReceiving = FALSE;
    call FIFOP.startWait(FALSE);
  }

  inline result_t setInitialTimer( uint16_t jiffy ) {
    stateTimer = TIMER_INITIAL;
    return call BackoffTimerJiffy.setOneShot(jiffy);
  }

  inline result_t setBackoffTimer( uint16_t jiffy ) {
    stateTimer = TIMER_BACKOFF;
    return call BackoffTimerJiffy.setOneShot(jiffy);
  }

  inline result_t setAckTimer( uint16_t jiffy ) {
    stateTimer = TIMER_ACK;
    return call BackoffTimerJiffy.setOneShot(jiffy);
  }

 /****************************************************************************
  * PacketRcvd
  * - Radio packet rcvd, signal
  ****************************************************************************/
  task void PacketRcvd() {
    TOS_MsgPtr pBuf;

    atomic {
      pBuf = rxbufptr;
    }
    pBuf = signal Receive.receive((TOS_MsgPtr)pBuf);
    atomic {
      if (pBuf) rxbufptr = pBuf;
      rxbufptr->length = 0;
      bPacketReceiving = FALSE;
    }
  }


 task void PacketSent() {
   TOS_MsgPtr pBuf; //store buf on stack

   atomic {
     stateRadio = IDLE_STATE;
     pBuf = txbufptr;
     pBuf->length = pBuf->length - MSG_HEADER_SIZE - MSG_FOOTER_SIZE;
   }

   signal Send.sendDone(pBuf,SUCCESS);
 }

 //***************************** ***************************
 //* Exported interface functions for Std/SplitControl
 //* StdControl is deprecated, use SplitControl
 //*****************************************************/

 // This interface is depricated, please use SplitControl instead
 command result_t StdControl.init() {
   return call SplitControl.init();
 }

 // Split-phase initialization of the radio
 command result_t SplitControl.init() {

   atomic {
     stateRadio = DISABLED_STATE;
     currentDSN = 0;
     bAckEnable = FALSE;
     bPacketReceiving = FALSE;
     rxbufptr = &RxBuf;
     rxbufptr->length = 0;
   }

   call TimerControl.init();
   call Random.init();
```

```
  LocalAddr = TOS_LOCAL_ADDRESS;
  return call CC2420SplitControl.init();
}

event result_t CC2420SplitControl.initDone() {
  return signal SplitControl.initDone();
}

default event result_t SplitControl.initDone() {
  return SUCCESS;
}

// This interface is depricated, please use SplitControl instead
command result_t StdControl.stop() {
  return call SplitControl.stop();
}

// split phase stop of the radio stack
command result_t SplitControl.stop() {
  atomic stateRadio = DISABLED_STATE;

  call SFD.disable();
  call FIFOP.disable();
  call TimerControl.stop();
  return call CC2420SplitControl.stop();
}

event result_t CC2420SplitControl.stopDone() {
  return signal SplitControl.stopDone();
}

default event result_t SplitControl.stopDone() {
  return SUCCESS;
}

// This interface is depricated, please use SplitControl instead
command result_t StdControl.start() {
  return call SplitControl.start();
}

// split phase start of the radio stack (wait for oscillator to start)
command result_t SplitControl.start() {
  uint8_t chkstateRadio;

  atomic chkstateRadio = stateRadio;

  if (chkstateRadio == DISABLED_STATE) {
    atomic {
  stateRadio = WARMUP_STATE;
      countRetry = 0;
      rxbufptr->length = 0;
    }
    call TimerControl.start();
    return call CC2420SplitControl.start();
  }
  return FAIL;
}

event result_t CC2420SplitControl.startDone() {
  uint8_t chkstateRadio;

  atomic chkstateRadio = stateRadio;

  if (chkstateRadio == WARMUP_STATE) {
    call CC2420Control.RxMode();
    //enable interrupt when pkt rcvd
    call FIFOP.startWait(FALSE);
    // enable start of frame delimiter timer capture (timestamping)
    call SFD.enableCapture(TRUE);

    atomic stateRadio  = IDLE_STATE;
  }
  signal SplitControl.startDone();
  return SUCCESS;
}

default event result_t SplitControl.startDone() {
```

```
    return SUCCESS;
  }

  /************* END OF STDCONTROL/SPLITCONTROL INIT FUNCITONS **********/

  /**
   * Try to send a packet.  If unsuccessful, backoff again
   **/
  void sendPacket() {
    uint8_t status;

    call HPLChipcon.cmd(CC2420_STXONCCA);
    status = call HPLChipcon.cmd(CC2420_SNOP);
    if ((status >> CC2420_TX_ACTIVE) & 0x01) {
      // wait for the SFD to go high for the transmit SFD
      call SFD.enableCapture(TRUE);
    }
    else {
      // try again to send the packet
      atomic stateRadio = PRE_TX_STATE;
      if (!(setBackoffTimer(signal MacBackoff.congestionBackoff(txbufptr) *
CC2420_SYMBOL_UNIT))) {
        sendFailed();
      }
    }
  }

  /**
   * Captured an edge transition on the SFD pin
   * Useful for time synchronization as well as determining
   * when a packet has finished transmission
   */
  async event result_t SFD.captured(uint16_t time) {
    switch (stateRadio) {
    case TX_STATE:
      // wait for SFD to fall--indicates end of packet
      call SFD.enableCapture(FALSE);
      // if the pin already fell, disable the capture and let the next
      // state enable the cpature (bug fix from Phil Buonadonna)
      if (!TOSH_READ_CC_SFD_PIN()) {
    call SFD.disable();
      }
      else {
    stateRadio = TX_WAIT;
      }
      // fire TX SFD event
      txbufptr->time = time;
      signal RadioSendCoordinator.startSymbol(8,0,txbufptr);
      // if the pin hasn't fallen, break out and wait for the interrupt
      // if it fell, continue on the to the TX_WAIT state
      if (stateRadio == TX_WAIT) {
    break;
      }
    case TX_WAIT:
      // end of packet reached
      stateRadio = POST_TX_STATE;
      call SFD.disable();
      // revert to receive SFD capture
      call SFD.enableCapture(TRUE);
      // if acks are enabled and it is a unicast packet, wait for the ack
      if ((bAckEnable) && (txbufptr->addr != TOS_BCAST_ADDR)) {
        if (!(setAckTimer(CC2420_ACK_DELAY)))
          sendFailed();
      }
      // if no acks or broadcast, post packet send done event
      else {
        if (!post PacketSent())
          sendFailed();
      }
      break;
    default:
      // fire RX SFD handler
      rxbufptr->time = time;
      signal RadioReceiveCoordinator.startSymbol(8,0,rxbufptr);
    }
    return SUCCESS;
  }
```

```
  /**
   * Start sending the packet data to the TXFIFO of the CC2420
   */
  task void startSend() {
    // flush the tx fifo of stale data
    if (!(call HPLChipcon.cmd(CC2420_SFLUSHTX))) {
      sendFailed();
      return;
    }
    // write the txbuf data to the TXFIFO
    if (!(call HPLChipconFIFO.writeTXFIFO(txlength+1,(uint8_t*)txbufptr))) {
      sendFailed();
      return;
    }
  }

  /**
   * Check for a clear channel and try to send the packet if a clear
   * channel exists using the sendPacket() function
   */
  void tryToSend() {
     uint8_t currentstate;
     atomic currentstate = stateRadio;

     // and the CCA check is good
     if (currentstate == PRE_TX_STATE) {

       // if a FIFO overflow occurs or if the data length is invalid, flush
       // the RXFIFO to get back to a normal state.
       if ((!TOSH_READ_CC_FIFO_PIN() && !TOSH_READ_CC_FIFOP_PIN())) {
         flushRXFIFO();
       }

       if (TOSH_READ_RADIO_CCA_PIN()) {
         atomic stateRadio = TX_STATE;
         sendPacket();
       }
       else {
     // if we tried a bunch of times, the radio may be in a bad state
     // flushing the RXFIFO returns the radio to a non-overflow state
     // and it continue normal operation (and thus send our packet)
         if (countRetry-- <= 0) {
       flushRXFIFO();
       countRetry = MAX_SEND_TRIES;
       if (!post startSend())
         sendFailed();
           return;
         }
         if (!(setBackoffTimer(signal MacBackoff.congestionBackoff(txbufptr) *
CC2420_SYMBOL_UNIT))) {
           sendFailed();
         }
       }
     }
  }

  /**
   * Multiplexed timer to control initial backoff,
   * congestion backoff, and delay while waiting for an ACK
   */
  async event result_t BackoffTimerJiffy.fired() {
    uint8_t currentstate;
    atomic currentstate = stateRadio;

    switch (stateTimer) {
    case TIMER_INITIAL:
      if (!(post startSend())) {
        sendFailed();
      }
      break;
    case TIMER_BACKOFF:
      tryToSend();
      break;
    case TIMER_ACK:
      if (currentstate == POST_TX_STATE) {
        txbufptr->ack = 0;
```

```
        if (!post PacketSent())
      sendFailed();
      }
      break;
    }
    return SUCCESS;
  }

 /*************************************************************
   * Send
   * - Xmit a packet
   *    USE SFD FALLING FOR END OF XMIT !!!!!!!!!!!!!!!!! interrupt???
   * - If in power-down state start timer ? !!!!!!!!!!!!!!!!!!!!!!!!s
   * - If !TxBusy then
   *   a) Flush the tx fifo
   *   b) Write Txfifo address
   *
   *************************************************************/
  command result_t Send.send(TOS_MsgPtr pMsg) {
    uint8_t currentstate;
    atomic currentstate = stateRadio;

    if (currentstate == IDLE_STATE) {
      // put default FCF values in to get address checking to pass
      pMsg->fcflo = CC2420_DEF_FCF_LO;
      if (bAckEnable)
        pMsg->fcfhi = CC2420_DEF_FCF_HI_ACK;
      else
        pMsg->fcfhi = CC2420_DEF_FCF_HI;
      // destination PAN is broadcast
      pMsg->destpan = TOS_BCAST_ADDR;
      // adjust the destination address to be in the right byte order
      pMsg->addr = toLSB16(pMsg->addr);
      // adjust the data length to now include the full packet length
      pMsg->length = pMsg->length + MSG_HEADER_SIZE + MSG_FOOTER_SIZE;
      // keep the DSN increasing for ACK recognition
      pMsg->dsn = ++currentDSN;
      // reset the time field
      pMsg->time = 0;
      // FCS bytes generated by CC2420
      txlength = pMsg->length - MSG_FOOTER_SIZE;
      txbufptr = pMsg;
      countRetry = MAX_SEND_TRIES;

      if (setInitialTimer(signal MacBackoff.initialBackoff(txbufptr) *
CC2420_SYMBOL_UNIT)) {
        atomic stateRadio = PRE_TX_STATE;
        return SUCCESS;
      }
    }
    return FAIL;

  }

  /**
   * Delayed RXFIFO is used to read the receive FIFO of the CC2420
   * in task context after the uC receives an interrupt that a packet
   * is in the RXFIFO.  Task context is necessary since reading from
   * the FIFO may take a while and we'd like to get other interrupts
   * during that time, or notifications of additional packets received
   * and stored in the CC2420 RXFIFO.
   */
  void delayedRXFIFO();

  task void delayedRXFIFOtask() {
    delayedRXFIFO();
  }

  void delayedRXFIFO() {
    uint8_t len = MSG_DATA_SIZE;
    uint8_t _bPacketReceiving;

    if ((!TOSH_READ_CC_FIFO_PIN()) && (!TOSH_READ_CC_FIFOP_PIN())) {
        flushRXFIFO();
    return;
    }
```

```
    atomic {
      _bPacketReceiving = bPacketReceiving;

      if (_bPacketReceiving) {
    if (!post delayedRXFIFOtask())
      flushRXFIFO();
      } else {
    bPacketReceiving = TRUE;
      }
    }

    // JP NOTE: TODO: move readRXFIFO out of atomic context to permit
    // high frequency sampling applications and remove delays on
    // interrupts being processed.  There is a race condition
    // that has not yet been diagnosed when RXFIFO may be interrupted.
    if (!_bPacketReceiving) {
      if (!call HPLChipconFIFO.readRXFIFO(len,(uint8_t*)rxbufptr)) {
    atomic bPacketReceiving = FALSE;
    if (!post delayedRXFIFOtask()) {
      flushRXFIFO();
    }
    return;
      }
    }
    flushRXFIFO();
  }

  /****************************************************
   * FIFOP lo Interrupt: Rx data avail in CC2420 fifo
   * Radio must have been in Rx mode to get this interrupt
   * If FIFO pin =lo then fifo overflow=> flush fifo & exit
   *
   *
   * Things ToDo:
   *
   * -Disable FIFOP interrupt until PacketRcvd task complete
   * until send.done complete
   *
   * -Fix mixup: on return
   *  rxbufptr->rssi is CRC + Correlation value
   *  rxbufptr->strength is RSSI
   ****************************************************/
   async event result_t FIFOP.fired() {

     //     call Leds.yellowToggle();

     // if we're trying to send a message and a FIFOP interrupt occurs
     // and acks are enabled, we need to backoff longer so that we don't
     // interfere with the ACK
     if (bAckEnable && (stateRadio == PRE_TX_STATE)) {
       if (call BackoffTimerJiffy.isSet()) {
         call BackoffTimerJiffy.stop();
         call BackoffTimerJiffy.setOneShot((signal
MacBackoff.congestionBackoff(txbufptr) * CC2420_SYMBOL_UNIT) + CC2420_ACK_DELAY);
       }
     }

     /** Check for RXFIFO overflow **/
     if (!TOSH_READ_CC_FIFO_PIN()){
       flushRXFIFO();
       return SUCCESS;
     }

     atomic {
     if (post delayedRXFIFOtask()) {
       call FIFOP.disable();
     }
     else {
       flushRXFIFO();
     }
     }

     // return SUCCESS to keep FIFOP events occurring
     return SUCCESS;
  }

  /**
```

```
 * After the buffer is received from the RXFIFO,
 * process it, then post a task to signal it to the higher layers
 */
async event result_t HPLChipconFIFO.RXFIFODone(uint8_t length, uint8_t *data) {
  // JP NOTE: rare known bug in high contention:
  // radio stack will receive a valid packet, but for some reason the
  // length field will be longer than normal.  The packet data will
  // be valid up to the correct length, and then will contain garbage
  // after the correct length.  There is no currently known fix.
  uint8_t currentstate;
  atomic {
    currentstate = stateRadio;
  }

  // if a FIFO overflow occurs or if the data length is invalid, flush
  // the RXFIFO to get back to a normal state.
  if ((!TOSH_READ_CC_FIFO_PIN() && !TOSH_READ_CC_FIFOP_PIN())
      || (length == 0) || (length > MSG_DATA_SIZE)) {
    flushRXFIFO();
    atomic bPacketReceiving = FALSE;
    return SUCCESS;
  }

  rxbufptr = (TOS_MsgPtr)data;

  // check for an acknowledgement that passes the CRC check
  if (bAckEnable && (currentstate == POST_TX_STATE) &&
       ((rxbufptr->fcfhi & 0x07) == CC2420_DEF_FCF_TYPE_ACK) &&
       (rxbufptr->dsn == currentDSN) &&
       ((data[length-1] >> 7) == 1)) {
    atomic {
      txbufptr->ack = 1;
      txbufptr->strength = data[length-2];
      txbufptr->lqi = data[length-1] & 0x7F;
      currentstate = POST_TX_ACK_STATE;
      bPacketReceiving = FALSE;
    }
    if (!post PacketSent())
  sendFailed();
    return SUCCESS;
  }

  // check for invalid packets
  // an invalid packet is a non-data packet with the wrong
  // addressing mode (FCFLO byte)
  if (((rxbufptr->fcfhi & 0x07) != CC2420_DEF_FCF_TYPE_DATA) ||
       (rxbufptr->fcflo != CC2420_DEF_FCF_LO)) {
    flushRXFIFO();
    atomic bPacketReceiving = FALSE;
    return SUCCESS;
  }

  rxbufptr->length = rxbufptr->length - MSG_HEADER_SIZE - MSG_FOOTER_SIZE;

  if (rxbufptr->length > TOSH_DATA_LENGTH) {
    flushRXFIFO();
    atomic bPacketReceiving = FALSE;
    return SUCCESS;
  }

  // adjust destination to the right byte order
  rxbufptr->addr = fromLSB16(rxbufptr->addr);

  // if the length is shorter, we have to move the CRC bytes
  rxbufptr->crc = data[length-1] >> 7;
  // put in RSSI
  rxbufptr->strength = data[length-2];
  // put in LQI
  rxbufptr->lqi = data[length-1] & 0x7F;

  atomic {
    if (!post PacketRcvd()) {
  bPacketReceiving = FALSE;
    }
  }

  if ((!TOSH_READ_CC_FIFO_PIN()) && (!TOSH_READ_CC_FIFOP_PIN())) {
```

```
        flushRXFIFO();
    return SUCCESS;
    }

    if (!(TOSH_READ_CC_FIFOP_PIN())) {
      if (post delayedRXFIFOtask())
    return SUCCESS;
    }
    flushRXFIFO();
    //    call FIFOP.startWait(FALSE);

    return SUCCESS;
  }

  /**
   * Notification that the TXFIFO has been filled with the data from the packet
   * Next step is to try to send the packet
   */
  async event result_t HPLChipconFIFO.TXFIFODone(uint8_t length, uint8_t *data) {
     tryToSend();
     return SUCCESS;
  }

  /** Enable link layer hardware acknowledgements **/
  async command void MacControl.enableAck() {
    atomic bAckEnable = TRUE;
    call CC2420Control.enableAddrDecode();
    call CC2420Control.enableAutoAck();
  }

  /** Disable link layer hardware acknowledgements **/
  async command void MacControl.disableAck() {
    atomic bAckEnable = FALSE;
    call CC2420Control.disableAddrDecode();
    call CC2420Control.disableAutoAck();
  }

  /**
   * How many basic time periods to back off.
   * Each basic time period consists of 20 symbols (16uS per symbol)
   */
  default async event int16_t MacBackoff.initialBackoff(TOS_MsgPtr m) {
    return (call Random.rand() & 0xF) + 1;
  }
  /**
   * How many symbols to back off when there is congestion
   * (16uS per symbol * 20 symbols/block)
   */
  default async event int16_t MacBackoff.congestionBackoff(TOS_MsgPtr m) {
    return (call Random.rand() & 0x3F) + 1;
  }

// Default events for radio send/receive coordinators do nothing.
// Be very careful using these, you'll break the stack.
// The "byte()" event is never signalled because the CC2420 is a packet
// based radio.
default async event void RadioSendCoordinator.startSymbol(uint8_t bitsPerBlock, uint8_t
offset, TOS_MsgPtr msgBuff) { }
default async event void RadioSendCoordinator.byte(TOS_MsgPtr msg, uint8_t byteCount) {
}
default async event void RadioReceiveCoordinator.startSymbol(uint8_t bitsPerBlock,
uint8_t offset, TOS_MsgPtr msgBuff) { }
default async event void RadioReceiveCoordinator.byte(TOS_MsgPtr msg, uint8_t byteCount)
{ }

}
```

ANEXO E. CÓDIGO CONTROL SEGURIDAD

1. Código nesC para motes MICAz

La aplicación de control de seguridad, esta compuesta por dos códigos de nesC diferentes, un para el nodo controlador y otro para los nodos de los usuarios. A continuación se puede observar los codigos diseñados para esta aplicación.

1.1. Archivo configuración nodo controlador

```
includes AM;
configuration controlador {
}
implementation {
  components Main, controladorM, RadioCRCPacket as Comm, FramerM, UART, LedsC,
SysTimeC, CC2420RadioC, TimerC;

  Main.StdControl -> controladorM;

  controladorM.UARTControl -> FramerM;
  controladorM.UARTSend -> FramerM;
  controladorM.UARTReceive -> FramerM;
  controladorM.UARTTokenReceive -> FramerM;
  controladorM.SysTime -> SysTimeC;
  controladorM.RadioControl -> Comm;
  controladorM.RadioSend -> Comm;
  controladorM.RadioReceive -> Comm;

//controladorM.CC2420Radio -> CC2420RadioM;
  controladorM.MacControl -> CC2420RadioC.MacControl;
  controladorM.Leds -> LedsC;
  //controladorM.Timer -> TimerC.Timer[unique("Timer_1")];
  controladorM.Timer -> TimerC.Timer[unique("Timer")];
  FramerM.ByteControl -> UART;
  FramerM.ByteComm -> UART;
}
```

1.2. Archivo módulo nodo controlador

```
includes AM;
#ifndef TOSBASE_BLINK_ON_DROP
#define TOSBASE_BLINK_ON_DROP
#endif

module controladorM {
  provides{
    interface StdControl;

 }
  uses {
    interface StdControl as UARTControl;
    interface BareSendMsg as UARTSend;
    interface ReceiveMsg as UARTReceive;
    interface TokenReceiveMsg as UARTTokenReceive;

    interface StdControl as RadioControl;
    interface BareSendMsg as RadioSend;
    interface ReceiveMsg as RadioReceive;
    interface Timer;
   interface MacControl;
   //interface CC2420Radio;
    interface Leds;
    interface SysTime;
  }
}
```

```
implementation
{
  enum {
    UART_QUEUE_LEN = 12,
    RADIO_QUEUE_LEN = 12,
  };

  TOS_Msg    uartQueueBufs[UART_QUEUE_LEN];
  uint8_t    uartIn, uartOut;
  bool       uartBusy, uartCount;


  TOS_MsgPtr prueba;
  TOS_Msg Hello;
  TOS_Msg    radioQueueBufs[RADIO_QUEUE_LEN];
  uint8_t    radioIn, radioOut;
  bool       radioBusy, radioCount;


  typedef struct {

           int8_t valor;
  } datos;

  typedef datos *SHop_MsgPtr;


  task void UARTSendTask();
  task void RadioSendTask();

  void failBlink();
  void dropBlink();
  void processUartPacket(TOS_MsgPtr Msg, bool wantsAck, uint8_t Token);
  void processRadioPacket(TOS_MsgPtr Msg);

  command result_t StdControl.init() {
    result_t ok1, ok2, ok3;
    prueba = &Hello;
    contador = 0;
    uartIn = uartOut = uartCount = 0;
    uartBusy = FALSE;

    radioIn = radioOut = radioCount = 0;
    radioBusy = FALSE;



    ok1 = call UARTControl.init();
    ok2 = call RadioControl.init();
    ok3 = call Leds.init();

    dbg(DBG_BOOT, "TOSBase initialized\n");

    return rcombine3(ok1, ok2, ok3);
  }

  command result_t StdControl.start() {
    result_t ok1, ok2;

    call Timer.start(TIMER_REPEAT, 200);

    call MacControl.enableAck();

    ok1 = call UARTControl.start();
    ok2 = call RadioControl.start();
    return rcombine(ok1, ok2);
  }

  command result_t StdControl.stop() {
    result_t ok1, ok2;

    ok1 = call UARTControl.stop();
    ok2 = call RadioControl.stop();
    call Timer.stop();

    return rcombine(ok1, ok2);
```

```
 }

 event result_t Timer.fired(){

   SHop_MsgPtr dathello = (SHop_MsgPtr) prueba->data;
   prueba->addr=TOS_BCAST_ADDR;
   prueba->group =TOS_AM_GROUP;
   prueba->type =1;
   prueba->length=5;
   dathello->valor = 69;
   call Leds.yellowToggle();
   processUartPacket(prueba, FALSE, 0);


   return SUCCESS;
}

 event TOS_MsgPtr RadioReceive.receive(TOS_MsgPtr Msg) {

    SHop_MsgPtr ptrmsg = (SHop_MsgPtr) Msg->data;

    dbg(DBG_USR1, "TOSBase received radio packet.\n");

    if ((!Msg->crc) || (Msg->group != TOS_AM_GROUP)){
      return Msg;
    }



    if((ptrmsg>5)||(ptrmsg<25)
    {

           if (uartCount < UART_QUEUE_LEN ) {
   Msg->type=1 ;
           memcpy(&uartQueueBufs[uartIn], Msg, sizeof(TOS_Msg));
           uartCount++;
           if( ++uartIn >= UART_QUEUE_LEN ) uartIn = 0;
           if (!uartBusy) {
                  if (post UARTSendTask()) {
                         uartBusy = TRUE;
                  }
                  }
           } else {
           dropBlink();
           }

    }else{

   Msg->adrr=ptrmsg;
   ptrmsg=69;
   processUartPacket(Msg, FALSE, 0);
    }
    return Msg;
}

    task void UARTSendTask() {
    dbg (DBG_USR1, "TOSBase forwarding Radio packet to UART\n");

    if (uartCount == 0) {

      uartBusy = FALSE;

    } else {

      if (call UARTSend.send(&uartQueueBufs[uartOut]) == SUCCESS) {

      } else {
   failBlink();
   post UARTSendTask();
      }
    }
}

 event result_t UARTSend.sendDone(TOS_MsgPtr msg, result_t success) {

    if (!success) {
```

```
    failBlink();
  } else {
    uartCount--;
    if( ++uartOut >= UART_QUEUE_LEN ) uartOut = 0;
  }

  post UARTSendTask();

  return SUCCESS;
}

event TOS_MsgPtr UARTReceive.receive(TOS_MsgPtr Msg) {
  processUartPacket(Msg, FALSE, 0);
  return Msg;
}

event TOS_MsgPtr UARTTokenReceive.receive(TOS_MsgPtr Msg, uint8_t Token) {
  processUartPacket(Msg, TRUE, Token);
  return Msg;
}

void processRadioPacket(TOS_MsgPtr Msg) {
  dbg(DBG_USR1, "TOSBase received Radio token packet.\n");

  if (uartCount < UART_QUEUE_LEN ) {
    memcpy(&uartQueueBufs[uartIn], Msg, sizeof(TOS_Msg));
    uartCount++;
    if( ++uartIn >= UART_QUEUE_LEN ) uartIn = 0;
    if (!uartBusy) {
         if (post UARTSendTask()) {
                uartBusy = TRUE;
         }
    }
  } else {
    dropBlink();
  }

}

void processUartPacket(TOS_MsgPtr Msg, bool wantsAck, uint8_t Token) {
  bool reflectToken = FALSE;
  if(contador ==300)
  {
         call Timer.stop();
  }
  dbg(DBG_USR1, "TOSBase received UART token packet.\n");



  if (radioCount < RADIO_QUEUE_LEN) {
    reflectToken = TRUE;

    memcpy(&radioQueueBufs[radioIn], Msg, sizeof(TOS_Msg));

    radioCount++;


    if( ++radioIn >= RADIO_QUEUE_LEN ) radioIn = 0;

    if (!radioBusy) {
  tiempo1 = call SysTime.getTime32();
  if (post RadioSendTask()) {
    radioBusy = TRUE;
  }
    }
  } else {
    dropBlink();
  }

  if (wantsAck && reflectToken) {
    call UARTTokenReceive.ReflectToken(Token);
  }
}

task void RadioSendTask() {

  dbg(DBG_USR1, "TOSBase forwarding UART packet to Radio\n");
```

```
    if (radioCount == 0) {

      radioBusy = FALSE;

    } else {


      if (call RadioSend.send(&radioQueueBufs[radioOut]) == SUCCESS) {

      } else {
    failBlink();
    post RadioSendTask();
      }
    }
  }

  event result_t RadioSend.sendDone(TOS_MsgPtr msg, result_t success) {

    if (!success) {
      failBlink();
    } else {
      radioCount--;
      if( ++radioOut >= RADIO_QUEUE_LEN ) radioOut = 0;
    }

    post RadioSendTask();
    return SUCCESS;
  }

  void dropBlink() {
#ifdef TOSBASE_BLINK_ON_DROP
    call Leds.redToggle();
#endif
  }

  void failBlink() {
#ifdef TOSBASE_BLINK_ON_FAIL
    call Leds.redToggle();
#endif
  }
}
```

1.3. Archivo configuración nodo usuario

```
includes AM;

configuration usuario {

}

implementation
{
  components   Main,   usuarioM,   GenericComm   as   Comm,   LedsC,   CC2420RadioC,
CC2420ControlM, Sounder, TimerC;

  Main.StdControl -> usuarioM;

  usuarioM.RadioReceive -> Comm.ReceiveMsg[0x01];
  usuarioM.RadioSend -> Comm.SendMsg[0x01];
  usuarioM.RadioControl -> Comm;
  usuarioM.Sound -> Sounder;
  usuarioM.MacControl -> CC2420RadioC.MacControl;

  usuarioM.Timer -> TimerC.Timer[unique("Timer")];
  usuarioM.Leds -> LedsC;


}
```

1.4. Archivo módulo nodo usuario

```
includes AM;

module usuarioM {
  provides interface StdControl;
  uses {


            interface StdControl as RadioControl;
                   interface SendMsg as RadioSend;
                   interface ReceiveMsg as RadioReceive;
            interface StdControl as Sound;

            interface MacControl;

            interface Timer;
            interface Leds;
  }
}

implementation {

// declaration of the static variables of the module
    TOS_Msg msg;
    TOS_MsgPtr prueba;
    uint8_t Identificacion;
    int i;
    typedef struct {
            int8_t valor;
    } datos;

    typedef datos *Dato;


// implementation of StdControl interface

    command result_t StdControl.init()
    {
            call Leds.init();
            call Sound.init();
            call RadioControl.init();
            prueba = &msg;
            Identificacion = 17;
            //inicio=TRUE;
            i=0;

                    return SUCCESS;
    }

    command result_t StdControl.start()
    {
            call RadioControl.start();
            call MacControl.enableAck();
            return SUCCESS;
    }

    command result_t StdControl.stop()
    {
            call RadioControl.stop();
            call Sound.stop();
            call Timer.stop();
                    return SUCCESS;
    }




//Implementación de la funcion recibir

    event TOS_MsgPtr RadioReceive.receive(TOS_MsgPtr m)
    {

            Dato datrecib = (Dato) m->data;
```

```
                call Leds.yellowToggle();
                i=0;
        // Miro si es el mensaje del controlador y si es asi le envio mi identificacion

                if (m->data[0]==69)   {

                        prueba->data[0]=Identificacion;
                        prueba->addr = 0;
                        prueba-> type=1;
                        prueba->group=TOS_AM_GROUP;
                        prueba->length=1;
                        call Leds.greenToggle();

                        atomic
                        {
                                call RadioSend.send(0, 5, prueba);
                        }




                }

        //Si mi identificación es erronia el controlador me lo hara saber

                //if (inicio==1){

                if(datrecib->valor==16)
                {
                        call Sound.start();
                        for (i=0;i<3000;i++){
                        }

                        call Sound.stop();

                }
                //}
                //inicio=0;
                return m;
        }

        event result_t RadioSend.sendDone(TOS_MsgPtr sent, result_t success)
        {

                if((sent->ack == 0) && (i!=1))
                {
                        call RadioSend.send(0, 5, sent);
                        call Leds.redToggle();
                        i++;
                }
                return SUCCESS;
        }

    }
```

2. Código C# para aplicación PC

Este subapartado muestra el código de C# referente a la aplicación control de acceso para el PC.

2.1. Programa principal

```
using System;
using System.Drawing;
using System.Collections;
using System.ComponentModel;
using System.Windows.Forms;
using System.Data;
```

```
using System.Threading;
using System.Net;
using System.Net.Sockets;
using System.IO;
using System.Text;
using System.Timers;
using System.Runtime.Serialization;
using System.Runtime.Serialization.Formatters.Binary;
using System.Diagnostics;
using System.Drawing.Drawing2D;
using clastomi;
using llibreria_connexio;

using System.Runtime.InteropServices;
using System.Data.OleDb;


namespace Seguridad
{
	/// <summary>
	/// Descripción breve de Form1.
	/// </summary>
	///
	public class Form1 : System.Windows.Forms.Form
	{
		Thread t;
		Socket C;
		System.Timers.Timer tiempo = new System.Timers.Timer();

		static IPAddress local = Dns.Resolve(Dns.GetHostName()).AddressList[0];
		static byte[] buffer = new byte[7];
		public int cont;
		IPEndPoint localEndPoint = new IPEndPoint(local, 9001);
		static byte[] init = new byte[2];
		private System.ComponentModel.IContainer components;
		public event ElapsedEventHandler Elapsed;
		private System.Windows.Forms.Label label1;
		private System.Windows.Forms.Button button4;
		private System.Windows.Forms.Button button6;
		public System.Windows.Forms.DataGrid dataGrid1;
		public int cont1 = 0;
		DataSet tabla = new DataSet();
		DataSet tablita = new DataSet();
		DataTable dt = new DataTable();
		clastomi.clastomi.personas per = new clastomi.clastomi.personas();
		int  i, iden, iden2;
		private System.Windows.Forms.Timer timer1;
		private System.Windows.Forms.DateTimePicker dateTimePicker1;

		/// <summary>
		/// Variable del diseñador requerida.
		/// </summary>

		public Form1()
		{
			//
			// Necesario para admitir el Diseñador de Windows Forms
			//
			InitializeComponent();

			//
			// TODO: agregar código de constructor después de llamar a
InitializeComponent
			//
		}

		/// <summary>
		/// Limpiar los recursos que se estén utilizando.
		/// </summary>
		protected override void Dispose( bool disposing )
		{
			if( disposing )
			{
				if (components != null)
				{
					components.Dispose();
				}
```

```
                        }
                        base.Dispose( disposing );
                }

                #region Código generado por el Diseñador de Windows Forms
                /// <summary>
                /// Método necesario para admitir el Diseñador. No se puede modificar
                /// el contenido del método con el editor de código.
                /// </summary>
                private void InitializeComponent()
                {
                        this.components = new System.ComponentModel.Container();
                        System.Resources.ResourceManager resources = new
System.Resources.ResourceManager(typeof(Form1));
                        this.label1 = new System.Windows.Forms.Label();
                        this.button4 = new System.Windows.Forms.Button();
                        this.button6 = new System.Windows.Forms.Button();
                        this.dataGrid1 = new System.Windows.Forms.DataGrid();
                        this.timer1 = new System.Windows.Forms.Timer(this.components);
                        this.dateTimePicker1 = new System.Windows.Forms.DateTimePicker();

        ((System.ComponentModel.ISupportInitialize)(this.dataGrid1)).BeginInit();
                        this.SuspendLayout();
                        //
                        // label1
                        //
                        this.label1.BackColor = System.Drawing.Color.Transparent;
                        this.label1.Font = new System.Drawing.Font("Tahoma", 20.25F,
((System.Drawing.FontStyle)((System.Drawing.FontStyle.Bold |
System.Drawing.FontStyle.Underline))), System.Drawing.GraphicsUnit.Point,
((System.Byte)(0)));
                        this.label1.ForeColor = System.Drawing.Color.Black;
                        this.label1.Location = new System.Drawing.Point(248, 8);
                        this.label1.Name = "label1";
                        this.label1.Size = new System.Drawing.Size(408, 48);
                        this.label1.TabIndex = 0;
                        this.label1.Text = "CENTRAL DE SEGURIDAD";
                        this.label1.TextAlign =
System.Drawing.ContentAlignment.MiddleCenter;
                        //
                        // button4
                        //
                        this.button4.BackColor = System.Drawing.Color.Gray;
                        this.button4.Cursor = System.Windows.Forms.Cursors.Hand;
                        this.button4.FlatStyle = System.Windows.Forms.FlatStyle.Flat;
                        this.button4.Font = new System.Drawing.Font("Tahoma", 9.75F,
System.Drawing.FontStyle.Bold, System.Drawing.GraphicsUnit.Point, ((System.Byte)(0)));
                        this.button4.ForeColor =
System.Drawing.SystemColors.ActiveCaptionText;
                        this.button4.Location = new System.Drawing.Point(248, 490);
                        this.button4.Name = "button4";
                        this.button4.Size = new System.Drawing.Size(128, 40);
                        this.button4.TabIndex = 3;
                        this.button4.Text = "ENCENDER";
                        this.button4.Click += new System.EventHandler(this.button4_Click);
                        //
                        // button6
                        //
                        this.button6.BackColor = System.Drawing.Color.Gray;
                        this.button6.Cursor = System.Windows.Forms.Cursors.Hand;
                        this.button6.FlatStyle = System.Windows.Forms.FlatStyle.Flat;
                        this.button6.Font = new System.Drawing.Font("Tahoma", 9.75F,
System.Drawing.FontStyle.Bold, System.Drawing.GraphicsUnit.Point, ((System.Byte)(0)));
                        this.button6.ForeColor =
System.Drawing.SystemColors.ActiveCaptionText;
                        this.button6.Location = new System.Drawing.Point(616, 490);
                        this.button6.Name = "button6";
                        this.button6.Size = new System.Drawing.Size(128, 40);
                        this.button6.TabIndex = 5;
                        this.button6.Text = "SALIR";
                        this.button6.Click += new System.EventHandler(this.button6_Click);
                        //
                        // dataGrid1
                        //
                        this.dataGrid1.AllowDrop = true;
                        this.dataGrid1.AlternatingBackColor =
System.Drawing.SystemColors.Info;
```

```
                        this.dataGrid1.BackColor =
System.Drawing.SystemColors.ControlLightLight;
                        this.dataGrid1.BackgroundColor = System.Drawing.SystemColors.Info;
                        this.dataGrid1.CaptionBackColor =
System.Drawing.SystemColors.Control;
                        this.dataGrid1.CaptionFont = new System.Drawing.Font("Palatino
Linotype", 8.25F, System.Drawing.FontStyle.Bold, System.Drawing.GraphicsUnit.Point,
((System.Byte)(0)));
                        this.dataGrid1.DataMember = "";
                        this.dataGrid1.FlatMode = true;
                        this.dataGrid1.GridLineStyle =
System.Windows.Forms.DataGridLineStyle.None;
                        this.dataGrid1.HeaderForeColor =
System.Drawing.SystemColors.ControlText;
                        this.dataGrid1.ImeMode = System.Windows.Forms.ImeMode.Alpha;
                        this.dataGrid1.Location = new System.Drawing.Point(40, 64);
                        this.dataGrid1.Name = "dataGrid1";
                        this.dataGrid1.PreferredColumnWidth = 110;
                        this.dataGrid1.Size = new System.Drawing.Size(920, 392);
                        this.dataGrid1.TabIndex = 6;
                        //
                        // timer1
                        //
                        this.timer1.Tick += new System.EventHandler(this.timer1_Tick);
                        //
                        // dateTimePicker1
                        //
                        this.dateTimePicker1.CalendarMonthBackground =
System.Drawing.SystemColors.Info;
                        this.dateTimePicker1.CalendarTitleForeColor =
System.Drawing.SystemColors.Info;
                        this.dateTimePicker1.Location = new System.Drawing.Point(752, 24);
                        this.dateTimePicker1.Name = "dateTimePicker1";
                        this.dateTimePicker1.TabIndex = 8;
                        //
                        // Form1
                        //
                        this.AutoScaleBaseSize = new System.Drawing.Size(5, 13);
                        this.BackColor = System.Drawing.SystemColors.Control;
                        this.BackgroundImage =
((System.Drawing.Image)(resources.GetObject("$this.BackgroundImage")));
                        this.ClientSize = new System.Drawing.Size(1008, 589);
                        this.Controls.Add(this.dateTimePicker1);
                        this.Controls.Add(this.dataGrid1);
                        this.Controls.Add(this.button6);
                        this.Controls.Add(this.button4);
                        this.Controls.Add(this.label1);
                        this.Name = "Form1";
                        this.Text = "Form1";
                        this.Load += new System.EventHandler(this.Form1_Load);

       ((System.ComponentModel.ISupportInitialize)(this.dataGrid1)).EndInit();
                        this.ResumeLayout(false);

                }
                #endregion

                /// <summary>
                /// Punto de entrada principal de la aplicación.
                /// </summary>
                [STAThread]
                static void Main()
                {
                        Application.Run(new Form1());

                }

                private void Form1_Load(object sender, System.EventArgs e)
                {
                        timer1.Start();
                        iden2=0;
                        dt = tablita.Tables.Add("tablita"); //caixe es un dataset que esta
creado arriba del todo
                        dt.Columns.Add ("IdHuésped");
                        dt.Columns.Add ("Nombre");
                        dt.Columns.Add ("Apellidos");
                        dt.Columns.Add ("Dirección");
```

```
                dt.Columns.Add ("Ciudad");
                dt.Columns.Add ("Región");
                dt.Columns.Add ("TeléfonoMóvil");
                dt.Columns.Add ("Hora");
        }


        private void button6_Click(object sender, System.EventArgs e)
        {
                try{C.Close();}
                catch{}
                try{t.Abort();}
                catch{}
                try
                {
                        foreach(Process myProcess in myProcesses)
                        {
                                myProcess.CloseMainWindow();
                                myProcess.Close();
                        }
                }
                catch{}
                Application.Exit();
        }

        private void timer1_Tick(object sender, System.EventArgs e)
        {
                afegir();
        }
        public void afegir()
        {
        int p = 1;
        if ( cont1 == 1 )
        {
                DataRow row = dt.NewRow(); //DataTable dt creado arriba
                FileStream Seguridad = new FileStream ("USUARIOS REGISTRADOS.txt",
FileMode.Append,FileAccess.Write,FileShare.Read );
                StreamWriter F = new StreamWriter(Seguridad);
                OleDbDataAdapter canal;
                tabla = new DataSet();
                string taula="ususarios";
                try
                {

                        connexio c = new connexio();
                        canal=c.buscar(per); //función que nos devuelve lo que
queremos
                        canal.Fill(tabla,taula);

                        foreach ( DataTable table in tabla.Tables)
                        {
                                foreach ( DataRow myRow in table.Rows)
                                {
                                        row["IdHuésped"]=
Convert.ToString(myRow["IdHuésped"]);
                                        row["Nombre"]=
Convert.ToString(myRow["Nombre"]);
                                        row["Apellidos"]=
Convert.ToString(myRow["Apellidos"]);
                                        row["Dirección"]=
Convert.ToString(myRow["Dirección"]);
                                        row["Ciudad"]=
Convert.ToString(myRow["Ciudad"]);
                                        row["Región"]=
Convert.ToString(myRow["Región"]);
                                        //row["Ciudad"] =
Convert.ToSingle(myRow["Ciudad"]);
                                        row["TeléfonoMóvil"]=
Convert.ToString(myRow["TeléfonoMóvil"]);
                                        row["Hora"]= dateTimePicker1.Value;

                                }
                        }

                }
                catch(Exception error)
                {
```

```
                        MessageBox.Show(error.Message);
                }

                        dt.Rows.InsertAt(row,p);
                        DataView dv = new DataView(dt);
                        dataGrid1.DataSource = dv;


                foreach ( DataTable table in tabla.Tables)
                {

                        foreach ( DataRow myRow in table.Rows)
                        {
                                F.Write(row["IdHuésped"]=
Convert.ToString(myRow["IdHuésped"]));
                                F.Write("\n");
                                F.Write(row["Nombre"]=
Convert.ToString(myRow["Nombre"]));
                                F.Write("\n");
                                F.Write(row["Apellidos"]=
Convert.ToString(myRow["Apellidos"]));
                                F.Write("\n");
                                F.Write(row["Dirección"]=
Convert.ToString(myRow["Dirección"]));
                                F.Write("\n");
                                F.Write(row["Ciudad"]=
Convert.ToString(myRow["Ciudad"]));
                                F.Write("\n");
                                F.Write(row["Región"]=
Convert.ToString(myRow["Región"]));
                                F.Write("\n");
                                F.Write(row["TeléfonoMóvil"]=
Convert.ToString(myRow["TeléfonoMóvil"]));
                                F.Write("\n");
                                F.Write(row["Hora"]=
Convert.ToString(row["Hora"]));
                                F.Write("\n");
                                F.Write("\n");
                                F.Write("\n");

                        }
                }
                F.Flush();
                F.Close();
        }

                cont1 = 0;
        }
        public void captura()
        {
                iden =0;
                C = new Socket(AddressFamily.InterNetwork, SocketType.Stream,
ProtocolType.Tcp);
                try
                {

                        C.Connect(localEndPoint);
                        C.Receive(init);
                        foreach (byte a in init)
                        {
                                Console.Write(a);
                        }
                        Console.Write("\n");

                        C.Send(init);
                        for(i=0;i<7;i++) buffer[i]=0;
                        while (true)
                        {
                                C.Receive(buffer);
                                foreach (byte a in buffer)
                                {
                                        Console.Write(a+"-");
                                }
                                Console.Write("\n");

                                if(buffer[4]!=0)
                                {
```

```
                                        if(buffer[0]==6)
                                        {
                                                salida[iden]=buffer[6];
                                                per.idhuesped=buffer[6];
                                                iden ++;
                                        }
                                        else
                                        {
                                                salida[iden]=buffer[5];
                                                per.idhuesped=buffer[5];
                                                iden ++;

                                        }

                                        if (buffer[0]==10)
                                        {
                                                MessageBox.Show("AVISO DE INTRUSIÓN
NO PERMITIDA");

                                                iden--;
                                                cont1 = 0;
                                        }
                                        else
                                        {

                                                for (i=0;i<iden-1;i++)
                                                {
                                                        if (salida[i] ==
per.idhuesped)
                                                                cont1 = -1;
                                                }
                                                if (cont1 != -1)
                                                {
                                                        cont1 = 1;
                                                }
                                                else
                                                {
                                                        cont1 = 0;
                                                        iden--;
                                                }
                                        }

                                        for(i=0;i<7;i++) buffer[i]=0;

                                }

                        }

                }

                catch
                {


                        MessageBox.Show("Se ha cerrado el socket");
                }
        }


        private void button4_Click(object sender, System.EventArgs e)
        {
                if (iden2==0)
                {
                        iden2=1;
                        t = new Thread(new ThreadStart(captura));
                        t.Start();
                }
        }

        private void button1_Click(object sender, System.EventArgs e)
        {
                DataRow row = dt.NewRow();
                FileStream Seguridad = new FileStream ("USUARIOS REGISTRADOS.txt",
FileMode.Append,FileAccess.Write,FileShare.Read );
                StreamWriter F = new StreamWriter(Seguridad);


                foreach ( DataTable table in tabla.Tables)
```

```
                        {

                                foreach ( DataRow myRow in table.Rows)
                                {
                                        F.Write(row["IdHuésped"]=
Convert.ToString(myRow["IdHuésped"]));
                                        F.Write("      ");
                                        F.Write(row["Nombre"]=
Convert.ToString(myRow["Nombre"]));
                                        F.Write("      ");
                                        F.Write(row["Apellidos"]=
Convert.ToString(myRow["Apellidos"]));
                                        F.Write("      ");
                                        F.Write(row["Dirección"]=
Convert.ToString(myRow["Dirección"]));
                                        F.Write("      ");
                                        F.Write(row["Ciudad"]=
Convert.ToString(myRow["Ciudad"]));
                                        F.Write("      ");
                                        F.Write(row["Región"]=
Convert.ToString(myRow["Región"]));
                                        F.Write("      ");
                                        F.Write(row["TeléfonoMóvil"]=
Convert.ToString(myRow["TeléfonoMóvil"]));
                                        F.Write("      ");
                                        F.Write("      ");
                                }
                        }
                        F.Flush();
                        F.Close();

                }
        }
}
```

2.2. Libreria conexión

```
using System;
using System.Data.OleDb;
using System.Data;
using System.IO;
using System.Collections;
using clastomi;

namespace llibreria_connexio
{
        public class connexio
        {
                //public string myConnectionString = "Provider=Microsoft.Jet.OLEDB.4.0;"
                //             + "Data
Source=C:\\Inetpub\\wwwroot\\WebApplication1\\bd1.MDB";
                public string myConnectionString = "Provider=Microsoft.Jet.OLEDB.4.0;" +
"Data Source=J:\\Documents and Settings\\tomcar\\Escritorio\\BASEUSUARIOS.mdb";

                public OleDbConnection establiment()
                {
                        //Crear un objeto OleDbConnection
                        //y pasar la ConnectionString al constructor.

                        OleDbConnection myConnection = new
OleDbConnection(myConnectionString);

                        //Abrir la conexión.
                        myConnection.Open();

                        return myConnection;

                }

                //FUNCIO PER A TANCAR CONNEXIO I OBJECTE DE LECTURA
                public void tancar_connexio(OleDbConnection x, OleDbDataReader y)
                {
                        x.Close();
```

```
                        y.Close();


                }

                //FUNCIO PER A OBTINDRE CANAL DE LECTURA A PARTIR D'UN OLEDBCOMMAND
                public OleDbDataReader obtindre_lector(OleDbCommand myCommand)
                {
                        //Enviar el CommandText a la conexión y crear un OleDbDataReader.
                        //Nota: El OleDbDataReader es de sólo avance.

                        OleDbDataReader myReader = myCommand.ExecuteReader();

                        return myReader;

                }

                //ENVIEM PETICIO, DESPUES SE PASARA EL PARAMETRE RETORNAT A
"OBTINDRE_LECTOR"
                public OleDbCommand command(string mySelect, OleDbConnection
myConnection)
                {

                        //Crear un objeto OleDbCommand.
                        //Observe que pasamos la instrucción SQL y el objeto
OleDbConnection.
                        OleDbCommand myCommand = new OleDbCommand(mySelect,myConnection);

                        return myCommand;



                }

                public OleDbDataAdapter buscar(clastomi.clastomi.personas per)
                {
                          OleDbDataAdapter canal;
                          string cadena = null;

                          OleDbConnection enllaç;


                          if (per.idhuesped != -1)
                          {
                                  cadena= "SELECT IdHuésped, Nombre, Apellidos, Dirección,
Ciudad, Región, TeléfonoMóvil FROM ususarios WHERE IdHuésped= "+ per.idhuesped;
                          }



                          connexio nueva = new connexio();
                          enllaç = nueva.establiment();
                          canal = new OleDbDataAdapter(cadena, enllaç);
                          return canal;
                  }



        }

}
```

2.3. Biblioteca paquete de datos

```
using System.Data;
using System.Threading;
using System.Net;
using System.Net.Sockets;
using System.IO;
using System.Text;
```

```
using System.Runtime.Serialization;
using System.Runtime.Serialization.Formatters.Binary;
using System.Data.OleDb;


namespace clastomi
{

        [Serializable]
        public class clastomi
        {
                [Serializable]
                        public class personas
                        {
                                public int idhuesped ;
                                public string nombre;
                                public string apellidos;
                                public string direccion;
                                public string ciudad;
                                public string region;
                                public int movil;
                                public DateTime hora;


                        }


        }
}
```

ANEXO F. CÓDIGO AUTOESCANER

1. Código nesC para motes MICAz

La aplicación de autoescaner, esta compuesta por dos códigos de nesC diferentes, un para el nodo validador y otro para los nodos de los productos. A continuación se puede observar los codigos diseñados para esta aplicación.

1.1. Archivo configuración nodo validador

```
includes AM;
configuration validador {
}
implementation {
  components Main, validadorM, RadioCRCPacket as Comm, FramerM, UART, LedsC,
SysTimeC, CC2420RadioC, TimerC;

  Main.StdControl -> controladorM;

  validadorM.UARTControl -> FramerM;
  validadorM.UARTSend -> FramerM;
  validadorM.UARTReceive -> FramerM;
  validadorM.UARTTokenReceive -> FramerM;
  validadorM M.SysTime -> SysTimeC;
  validadorM.RadioControl -> Comm;
  validadorM.RadioSend -> Comm;
  validadorM.RadioReceive -> Comm;
  validadorM.MacControl -> CC2420RadioC.MacControl;
  validadorM.Leds -> LedsC;
  validadorM.Timer -> TimerC.Timer[unique("Timer")];
  FramerM.ByteControl -> UART;
  FramerM.ByteComm -> UART;
}
```

1.2. Archivo módulo nodo validador

```
includes AM;
#ifndef TOSBASE_BLINK_ON_DROP
#define TOSBASE_BLINK_ON_DROP
#endif

module validadorM {
  provides{
    interface StdControl;

 }
  uses {
    interface StdControl as UARTControl;
    interface BareSendMsg as UARTSend;
    interface ReceiveMsg as UARTReceive;
    interface TokenReceiveMsg as UARTTokenReceive;

    interface StdControl as RadioControl;
    interface BareSendMsg as RadioSend;
    interface ReceiveMsg as RadioReceive;
    interface Timer;
    interface MacControl;
    interface Leds;
    interface SysTime;
  }
}

implementation
{
  enum {
    UART_QUEUE_LEN = 12,
    RADIO_QUEUE_LEN = 12,
  };
```

```
TOS_Msg    uartQueueBufs[UART_QUEUE_LEN];
uint8_t    uartIn, uartOut;
bool       uartBusy, uartCount;


TOS_MsgPtr prueba;
TOS_Msg Hello;
TOS_Msg    radioQueueBufs[RADIO_QUEUE_LEN];
uint8_t    radioIn, radioOut;
bool       radioBusy, radioCount;


typedef struct {

         int8_t valor;
} datos;

typedef datos *SHop_MsgPtr;


task void UARTSendTask();
task void RadioSendTask();

void failBlink();
void dropBlink();
void processUartPacket(TOS_MsgPtr Msg, bool wantsAck, uint8_t Token);
void processRadioPacket(TOS_MsgPtr Msg);

command result_t StdControl.init() {
  result_t ok1, ok2, ok3;
  prueba = &Hello;
  contador = 0;
  uartIn = uartOut = uartCount = 0;
  uartBusy = FALSE;

  radioIn = radioOut = radioCount = 0;
  radioBusy = FALSE;
   ok1 = call UARTControl.init();
  ok2 = call RadioControl.init();
  ok3 = call Leds.init();

  dbg(DBG_BOOT, "TOSBase initialized\n");

  return rcombine3(ok1, ok2, ok3);
}

command result_t StdControl.start() {
  result_t ok1, ok2;

  call Timer.start(TIMER_REPEAT, 200);
  call MacControl.enableAck();

  ok1 = call UARTControl.start();
  ok2 = call RadioControl.start();
  return rcombine(ok1, ok2);
}

command result_t StdControl.stop() {
  result_t ok1, ok2;

  ok1 = call UARTControl.stop();
  ok2 = call RadioControl.stop();
  call Timer.stop();

  return rcombine(ok1, ok2);
}

event result_t Timer.fired(){

  SHop_MsgPtr dathello = (SHop_MsgPtr) prueba->data;
  prueba->addr=TOS_BCAST_ADDR;
  prueba->group =TOS_AM_GROUP;
  prueba->type =1;
  prueba->length=5;
  dathello->valor = 69;
  call Leds.yellowToggle();
  processUartPacket(prueba, FALSE, 0);
```

```
  return SUCCESS;
}

 event TOS_MsgPtr RadioReceive.receive(TOS_MsgPtr Msg) {

   SHop_MsgPtr ptrmsg = (SHop_MsgPtr) Msg->data;

   dbg(DBG_USR1, "TOSBase received radio packet.\n");

   if ((!Msg->crc) || (Msg->group != TOS_AM_GROUP)){
     return Msg;
   }

   if((ptrmsg>5)||(ptrmsg<25)
   {

          if (uartCount < UART_QUEUE_LEN ) {
          Msg->type=1 ;
          memcpy(&uartQueueBufs[uartIn], Msg, sizeof(TOS_Msg));
          uartCount++;
          if( ++uartIn >= UART_QUEUE_LEN ) uartIn = 0;
          if (!uartBusy) {
                 if (post UARTSendTask()) {
                        uartBusy = TRUE;
                 }
                 }
          } else {
          dropBlink();
          }

   }else{

  Msg->adrr=ptrmsg;
  ptrmsg=69;
  processUartPacket(Msg, FALSE, 0);
   }
   return Msg;
}

   task void UARTSendTask() {
   dbg (DBG_USR1, "TOSBase forwarding Radio packet to UART\n");

   if (uartCount == 0) {

     uartBusy = FALSE;

   } else {

     if (call UARTSend.send(&uartQueueBufs[uartOut]) == SUCCESS) {

     } else {
  failBlink();
  post UARTSendTask();
     }
   }
}

 event result_t UARTSend.sendDone(TOS_MsgPtr msg, result_t success) {

   if (!success) {
     failBlink();
   } else {
     uartCount--;
     if( ++uartOut >= UART_QUEUE_LEN ) uartOut = 0;
   }

   post UARTSendTask();

   return SUCCESS;
}

 event TOS_MsgPtr UARTReceive.receive(TOS_MsgPtr Msg) {
   processUartPacket(Msg, FALSE, 0);
   return Msg;
}

 event TOS_MsgPtr UARTTokenReceive.receive(TOS_MsgPtr Msg, uint8_t Token) {
```

```
  processUartPacket(Msg, TRUE, Token);
  return Msg;
}
void processRadioPacket(TOS_MsgPtr Msg) {
  dbg(DBG_USR1, "TOSBase received Radio token packet.\n");

  if (uartCount < UART_QUEUE_LEN ) {
    memcpy(&uartQueueBufs[uartIn], Msg, sizeof(TOS_Msg));
    uartCount++;
    if( ++uartIn >= UART_QUEUE_LEN ) uartIn = 0;
    if (!uartBusy) {
         if (post UARTSendTask()) {
                uartBusy = TRUE;
         }
    }
  } else {
    dropBlink();
  }

}

void processUartPacket(TOS_MsgPtr Msg, bool wantsAck, uint8_t Token) {
  bool reflectToken = FALSE;
  if(contador ==300)
  {
         call Timer.stop();
  }
  dbg(DBG_USR1, "TOSBase received UART token packet.\n");

  if (radioCount < RADIO_QUEUE_LEN) {
    reflectToken = TRUE;

    memcpy(&radioQueueBufs[radioIn], Msg, sizeof(TOS_Msg));

    radioCount++;

    if( ++radioIn >= RADIO_QUEUE_LEN ) radioIn = 0;

    if (!radioBusy) {
  tiempo1 = call SysTime.getTime32();
  if (post RadioSendTask()) {
    radioBusy = TRUE;
  }
    }
  } else {
    dropBlink();
  }

  if (wantsAck && reflectToken) {
    call UARTTokenReceive.ReflectToken(Token);
  }
}

task void RadioSendTask() {

  dbg(DBG_USR1, "TOSBase forwarding UART packet to Radio\n");

  if (radioCount == 0) {

    radioBusy = FALSE;

  } else {


    if (call RadioSend.send(&radioQueueBufs[radioOut]) == SUCCESS) {

    } else {
  failBlink();
  post RadioSendTask();
    }
  }
}

event result_t RadioSend.sendDone(TOS_MsgPtr msg, result_t success) {

  if (!success) {
    failBlink();
```

```
    } else {
      radioCount--;
      if( ++radioOut >= RADIO_QUEUE_LEN ) radioOut = 0;
    }

    post RadioSendTask();
    return SUCCESS;
  }

  void dropBlink() {
#ifdef TOSBASE_BLINK_ON_DROP
    call Leds.redToggle();
#endif
  }

  void failBlink() {
#ifdef TOSBASE_BLINK_ON_FAIL
    call Leds.redToggle();
#endif
  }
}
```

1.3. Archivo configuración nodo producto

```
includes AM;

configuration producto {

}

implementation
{
  components Main, productoM, GenericComm as Comm, LedsC, CC2420RadioC,
CC2420ControlM, Sounder, TimerC;

  Main.StdControl -> usuarioM;

  productoM.RadioReceive -> Comm.ReceiveMsg[0x01];
  productoM.RadioSend -> Comm.SendMsg[0x01];
  productoM.RadioControl -> Comm;
  productoM.Sound -> Sounder;
  productoM.MacControl -> CC2420RadioC.MacControl;

  productoM.Timer -> TimerC.Timer[unique("Timer")];
  productoM.Leds -> LedsC;


}
```

1.4. Archivo módulo nodo producto

```
includes AM;

module productoM {
  provides interface StdControl;
  uses {


            interface StdControl as RadioControl;
            interface SendMsg as RadioSend;
            interface ReceiveMsg as RadioReceive;
            interface StdControl as Sound;
            interface MacControl;
            interface Timer;
            interface Leds;
  }
}

implementation {
```

```
// declaration of the static variables of the module
    TOS_Msg msg;
    TOS_MsgPtr prueba;
    uint8_t Identificacion;
    int i;
    typedef struct {
           int8_t valor;
    } datos;

    typedef datos *Dato;


// implementation of StdControl interface

    command result_t StdControl.init()
    {
           call Leds.init();
           call Sound.init();
           call RadioControl.init();
           prueba = &msg;
           Identificacion = 17;
           //inicio=TRUE;
           i=0;

           return SUCCESS;
    }

    command result_t StdControl.start()
    {
           call RadioControl.start();
           call MacControl.enableAck();
           return SUCCESS;
    }

    command result_t StdControl.stop()
    {
           call RadioControl.stop();
           call Sound.stop();
           call Timer.stop();
                  return SUCCESS;
    }




//Implementación de la funcion recibir

    event TOS_MsgPtr RadioReceive.receive(TOS_MsgPtr m)
    {

           Dato datrecib = (Dato) m->data;
           call Leds.yellowToggle();
           i=0;
    // Miro si es el mensaje del controlador y si es asi le envio mi identificacion

           if (m->data[0]==69)   {

                  prueba->data[0]=Identificacion;
                  prueba->addr = 0;
                  prueba-> type=1;
                  prueba->group=TOS_AM_GROUP;
                  prueba->length=1;
                  call Leds.greenToggle();

                  atomic
                  {
                         call RadioSend.send(0, 5, prueba);
                  }




           }

    //Si mi identificación es erronia el controlador me lo hara saber
```

```
                //if (inicio==1){

                if(datrecib->valor==16)
                {
                        call Sound.start();
                        for (i=0;i<3000;i++){
                        }

                        call Sound.stop();

                }
                //}
                //inicio=0;
                return m;
        }

        event result_t RadioSend.sendDone(TOS_MsgPtr sent, result_t success)
        {

                if((sent->ack == 0) && (i!=1))
                {
                        call RadioSend.send(0, 5, sent);
                        call Leds.redToggle();
                        i++;
                }
                return SUCCESS;
        }

    }
```

2. Código C# para aplicación PC

Este subapartado muestra el código de C# referente a la aplicación autoescaner para el PC. En este caso no mostraremos la librería conexión ya que igual que en la aplicación anterior.

2.1. Programa principal

```
using System;
using System.Drawing;
using System.Collections;
using System.ComponentModel;
using System.Windows.Forms;
using System.Data;
using System.Threading;
using System.Net;
using System.Net.Sockets;
using System.IO;
using System.Text;
using System.Timers;
using System.Runtime.Serialization;
using System.Runtime.Serialization.Formatters.Binary;
using System.Diagnostics;
using System.Drawing.Drawing2D;
using biblioteca_super;
using llibreria_super;

using System.Runtime.InteropServices;
using System.Data.OleDb;

namespace SUPERMERCADO
{
        /// <summary>
        /// Descripción breve de Supermercado.
        /// </summary>
        public class Supermercado : System.Windows.Forms.Form
        {
```

```
		Thread t;
		Process[] myProcesses;
		bool prue= true;
		Socket C;
		System.Timers.Timer tiempo = new System.Timers.Timer();

		static IPAddress local = Dns.Resolve(Dns.GetHostName()).AddressList[0];
		static byte[] buffer = new byte[7];
		public int cont;
		static byte[] salida = new byte[128];
		static byte[] guardar = new byte[150];

		IPEndPoint localEndPoint = new IPEndPoint(local, 9001);
		static byte[] init = new byte[2];
		private System.ComponentModel.IContainer components;
		public event ElapsedEventHandler Elapsed;

		private System.Windows.Forms.Button button1;
		private System.Windows.Forms.Button button3;
		private System.Windows.Forms.DataGrid dataGrid1;

		public int cont1=0;
		/// <summary>
		/// Variable del diseñador requerida.
		/// </summary>
		/// public int cont1 = 0;
		//DataView dv = new DataView();
		DataSet tabla = new DataSet();
		DataSet tablita = new DataSet();
		DataTable dt = new DataTable();
		biblioteca_super.super.productos pro = new
biblioteca_super.super.productos();
		int  i, iden, iden2;
		float suma_total = 0;
		private System.Windows.Forms.Label preu_tot;
		private System.Windows.Forms.Label label1;
		private System.Windows.Forms.Label label2;
		private System.Windows.Forms.Button button2;
		private System.Windows.Forms.Label Total;
		private System.Windows.Forms.Timer timer1;

		public Supermercado()
		{
			//
			// Necesario para admitir el Diseñador de Windows Forms
			//
			InitializeComponent();

			//
			// TODO: agregar código de constructor después de llamar a
InitializeComponent
			//
		}

		/// <summary>
		/// Limpiar los recursos que se estén utilizando.
		/// </summary>
		protected override void Dispose( bool disposing )
		{
			if( disposing )
			{
				if(components != null)
				{
					components.Dispose();
				}
			}
			base.Dispose( disposing );
		}


		#region Código generado por el Diseñador de Windows Forms
		/// <summary>
		/// Método necesario para admitir el Diseñador. No se puede modificar
		/// el contenido del método con el editor de código.
		/// </summary>
		private void InitializeComponent()
		{
```

```
                        this.components = new System.ComponentModel.Container();
                        System.Resources.ResourceManager resources = new
System.Resources.ResourceManager(typeof(Supermercado));
                        this.button1 = new System.Windows.Forms.Button();
                        this.button3 = new System.Windows.Forms.Button();
                        this.dataGrid1 = new System.Windows.Forms.DataGrid();
                        this.timer1 = new System.Windows.Forms.Timer(this.components);
                        this.preu_tot = new System.Windows.Forms.Label();
                        this.label1 = new System.Windows.Forms.Label();
                        this.label2 = new System.Windows.Forms.Label();
                        this.button2 = new System.Windows.Forms.Button();
                        this.Total = new System.Windows.Forms.Label();

       ((System.ComponentModel.ISupportInitialize)(this.dataGrid1)).BeginInit();
                        this.SuspendLayout();
                        //
                        // button1
                        //
                        this.button1.BackColor = System.Drawing.Color.Transparent;
                        this.button1.Cursor = System.Windows.Forms.Cursors.Hand;
                        this.button1.Font = new System.Drawing.Font("Microsoft Sans
Serif", 12F, System.Drawing.FontStyle.Bold, System.Drawing.GraphicsUnit.Point,
((System.Byte)(0)));
                        this.button1.ForeColor = System.Drawing.Color.DarkOrange;
                        this.button1.Location = new System.Drawing.Point(64, 312);
                        this.button1.Name = "button1";
                        this.button1.Size = new System.Drawing.Size(96, 48);
                        this.button1.TabIndex = 0;
                        this.button1.Text = "Encender";
                        this.button1.Click += new System.EventHandler(this.button1_Click);
                        //
                        // button3
                        //
                        this.button3.BackColor = System.Drawing.Color.Transparent;
                        this.button3.Cursor = System.Windows.Forms.Cursors.Hand;
                        this.button3.Font = new System.Drawing.Font("Microsoft Sans
Serif", 12F, System.Drawing.FontStyle.Bold, System.Drawing.GraphicsUnit.Point,
((System.Byte)(0)));
                        this.button3.ForeColor = System.Drawing.Color.DarkOrange;
                        this.button3.Location = new System.Drawing.Point(448, 312);
                        this.button3.Name = "button3";
                        this.button3.RightToLeft = System.Windows.Forms.RightToLeft.Yes;
                        this.button3.Size = new System.Drawing.Size(96, 48);
                        this.button3.TabIndex = 0;
                        this.button3.Text = "Salir";
                        this.button3.Click += new System.EventHandler(this.button3_Click);
                        //
                        // dataGrid1
                        //
                        this.dataGrid1.BackgroundColor = System.Drawing.SystemColors.Info;
                        this.dataGrid1.CaptionBackColor =
System.Drawing.SystemColors.Desktop;
                        this.dataGrid1.CaptionForeColor =
System.Drawing.SystemColors.ActiveCaption;
                        this.dataGrid1.DataMember = "";
                        this.dataGrid1.ForeColor = System.Drawing.SystemColors.MenuText;
                        this.dataGrid1.GridLineColor =
System.Drawing.SystemColors.Desktop;
                        this.dataGrid1.HeaderBackColor =
System.Drawing.Color.FromArgb(((System.Byte)(255)), ((System.Byte)(192)),
((System.Byte)(128)));
                        this.dataGrid1.HeaderForeColor =
System.Drawing.SystemColors.ControlText;
                        this.dataGrid1.Location = new System.Drawing.Point(136, 72);
                        this.dataGrid1.Name = "dataGrid1";
                        this.dataGrid1.PreferredColumnWidth = 110;
                        this.dataGrid1.Size = new System.Drawing.Size(368, 144);
                        this.dataGrid1.TabIndex = 3;
                        //
                        // timer1
                        //
                        this.timer1.Tick += new System.EventHandler(this.timer1_Tick);
                        //
                        // preu_tot
                        //
                        this.preu_tot.BackColor = System.Drawing.SystemColors.Info;
```

```
                    this.preu_tot.Font = new System.Drawing.Font("Verdana", 26.25F,
System.Drawing.FontStyle.Bold, System.Drawing.GraphicsUnit.Point, ((System.Byte)(0)));
                    this.preu_tot.Location = new System.Drawing.Point(224, 248);
                    this.preu_tot.Name = "preu_tot";
                    this.preu_tot.Size = new System.Drawing.Size(144, 40);
                    this.preu_tot.TabIndex = 4;
                    //
                    // label1
                    //
                    this.label1.BackColor = System.Drawing.Color.Transparent;
                    this.label1.Font = new System.Drawing.Font("Times New Roman",
21.75F, System.Drawing.FontStyle.Bold, System.Drawing.GraphicsUnit.Point,
((System.Byte)(0)));
                    this.label1.Location = new System.Drawing.Point(408, 248);
                    this.label1.Name = "label1";
                    this.label1.Size = new System.Drawing.Size(128, 40);
                    this.label1.TabIndex = 6;
                    this.label1.Text = "€";
                    //
                    // label2
                    //
                    this.label2.BackColor = System.Drawing.Color.Transparent;
                    this.label2.Font = new System.Drawing.Font("Times New Roman",
21.75F, ((System.Drawing.FontStyle)((System.Drawing.FontStyle.Bold |
System.Drawing.FontStyle.Underline))), System.Drawing.GraphicsUnit.Point,
((System.Byte)(0)));
                    this.label2.Location = new System.Drawing.Point(136, 16);
                    this.label2.Name = "label2";
                    this.label2.Size = new System.Drawing.Size(376, 40);
                    this.label2.TabIndex = 7;
                    this.label2.Text = "INTERFACE VENDEDOR";
                    //
                    // button2
                    //
                    this.button2.BackColor = System.Drawing.Color.Transparent;
                    this.button2.Cursor = System.Windows.Forms.Cursors.Hand;
                    this.button2.Font = new System.Drawing.Font("Microsoft Sans
Serif", 12F, System.Drawing.FontStyle.Bold, System.Drawing.GraphicsUnit.Point,
((System.Byte)(0)));
                    this.button2.ForeColor = System.Drawing.Color.DarkOrange;
                    this.button2.Location = new System.Drawing.Point(208, 312);
                    this.button2.Name = "button2";
                    this.button2.Size = new System.Drawing.Size(192, 48);
                    this.button2.TabIndex = 8;
                    this.button2.Text = "NUEVO CLIENTE";
                    this.button2.Click += new
System.EventHandler(this.button2_Click_1);
                    //
                    // Total
                    //
                    this.Total.BackColor = System.Drawing.Color.Transparent;
                    this.Total.Font = new System.Drawing.Font("Times New Roman",
21.75F, ((System.Drawing.FontStyle)((System.Drawing.FontStyle.Bold |
System.Drawing.FontStyle.Underline))), System.Drawing.GraphicsUnit.Point,
((System.Byte)(0)));
                    this.Total.Location = new System.Drawing.Point(64, 248);
                    this.Total.Name = "Total";
                    this.Total.Size = new System.Drawing.Size(128, 40);
                    this.Total.TabIndex = 5;
                    this.Total.Text = "TOTAL:";
                    //
                    // Supermercado
                    //
                    this.AutoScaleBaseSize = new System.Drawing.Size(5, 13);
                    this.BackgroundImage =
((System.Drawing.Image)(resources.GetObject("$this.BackgroundImage")));
                    this.ClientSize = new System.Drawing.Size(616, 389);
                    this.Controls.Add(this.button2);
                    this.Controls.Add(this.label2);
                    this.Controls.Add(this.label1);
                    this.Controls.Add(this.Total);
                    this.Controls.Add(this.preu_tot);
                    this.Controls.Add(this.dataGrid1);
                    this.Controls.Add(this.button3);
                    this.Controls.Add(this.button1);
                    this.FormBorderStyle =
System.Windows.Forms.FormBorderStyle.Fixed3D;
```

```
                        this.Name = "Supermercado";
                        this.Text = "Supermercado";
                        this.Load += new System.EventHandler(this.Supermercado_Load);

        ((System.ComponentModel.ISupportInitialize)(this.dataGrid1)).EndInit();
                        this.ResumeLayout(false);

                }
                #endregion

                [STAThread]
                static void Main()
                {
                        Application.Run(new Supermercado());

                }

                private void Supermercado_Load(object sender, System.EventArgs e)
                {
                        timer1.Start();
                        iden2=0;
                        /*if(textBox1.Text.Length!=0)
                        {
                                valor =Convert.ToInt32(textBox1.Text);
                        }
                        else
                        {
                                valor=0;
                        }*/
                        dt = tablita.Tables.Add("tablita"); //caixe es un dataset que esta
creado arriba del todo

                        dt.Columns.Add ("Identificador");
                        dt.Columns.Add ("Nombre");
                        dt.Columns.Add ("Precio");
                }

                private void button3_Click(object sender, System.EventArgs e)
                {
                        try{C.Close();}
                        catch{}



                        try{t.Abort();}
                        catch{}
                        try
                        {
                                foreach(Process myProcess in myProcesses)
                                {
                                        myProcess.CloseMainWindow();
                                        myProcess.Close();
                                }
                        }
                        catch{}
                        Application.Exit();
                }

                private void button2_Click(object sender, System.EventArgs e)
                {
                        DataRow row = dt.NewRow();
                        FileStream SUPERMERCADO = new FileStream ("Productos.txt",
FileMode.Append,FileAccess.Write,FileShare.Read );
                        StreamWriter F = new StreamWriter(SUPERMERCADO);


                        foreach ( DataTable table in tabla.Tables)
                        {

                                foreach ( DataRow myRow in table.Rows)
                                {
                                        F.Write(row["Identificador"]=
Convert.ToString(myRow["Identificador"]));
                                        F.Write("      ");
                                        F.Write(row["Nombre"]=
Convert.ToString(myRow["Nombre"]));
                                        F.Write("      ");
```

```
                                F.Write(row["Precio"]=
Convert.ToString(myRow["Precio"]));
                                F.Write("      ");


                        }
                }
                F.Flush();
                F.Close();
        }

        private void timer1_Tick(object sender, System.EventArgs e)
        {
                afegir();

        }

        public void afegir()
        {
                int p = 1;
                if ( cont1 == 1 )
                {

                        DataRow row = dt.NewRow(); //DataTable dt creado arriba
                        //DataRow row2 = dt.NewRow();
                        //tomi.personas per = new biblioteca_tomi.tomi.personas ();
                        //botiga.producte_util pu= new
biblioteca_botiga.botiga.producte_util();
                        OleDbDataAdapter canal;

                        tabla = new DataSet();
                        string taula="productos";
                        //pu.codi_barres= textBox1.Text;



                        try
                        {

                                llibreria_super.super c = new
llibreria_super.super();
                                canal=c.buscar(pro); //función que nos devuelve lo
que queremos
                                canal.Fill(tabla,taula);

                                foreach ( DataTable table in tabla.Tables)
                                {
                                        foreach ( DataRow myRow in table.Rows)
                                        {
                                                row["Identificador"]=
Convert.ToString(myRow["Identificador"]);
                                                row["Nombre"]=
Convert.ToString(myRow["Nombre"]);
                                                row["Precio"]=
Convert.ToString(myRow["Precio"]);



                                        }
                                }

                        }
                        catch(Exception error)
                        {
                                MessageBox.Show(error.Message);
                        }



                        dt.Rows.InsertAt(row,p);
                        preu_tot.Text=suma().ToString();


                        DataView dv = new DataView(dt);
                        dataGrid1.DataSource = dv;
                        //dataGrid1.DataMember= taula;
```

```
                        //r.Abort();
                        //r.Suspend();
                        //timer1.Stop();
                }

                cont1 = 0;
        }

        public void captura()
        {
                iden =0;
                //tiempo.Elapsed+= new ElapsedEventHandler(afegir);
                //tiempo.Interval=100;
                //tiempo.Enabled=false;

                C = new Socket(AddressFamily.InterNetwork, SocketType.Stream,
ProtocolType.Tcp);
                try
                {

                        C.Connect(localEndPoint);
                        C.Receive(init);
                        foreach (byte a in init)
                        {
                                Console.Write(a);
                        }
                        Console.Write("\n");

                        C.Send(init);
                        for(i=0;i<7;i++) buffer[i]=0;
                        //per.id_huesped=0;
                        //r = new Thread(new ThreadStart(afegir));
                        while (true)
                        {
                                C.Receive(buffer);
                                //tiempo.Enabled=false;
                                foreach (byte a in buffer)
                                {
                                        Console.Write(a+"-");
                                }
                                Console.Write("\n");

                                if(buffer[4]!=0)
                                {

                                        if(buffer[0]==6)
                                        {
                                                salida[iden]=buffer[6];
                                                //textBox1.Text=buffer[6]+"";
                                                pro.Identificador=buffer[6];
                                                iden ++;

                                        }
                                        else
                                        {
                                                salida[iden]=buffer[5];
                                                //textBox1.Text=buffer[5]+"";
                                                pro.Identificador=buffer[5];
                                                iden ++;
                                        }
                                        for(i=0;i<7;i++) buffer[i]=0;

                                        for (i=0;i<iden-1;i++)
                                        {
                                                if (salida[i] == pro.Identificador)
                                                        cont1 = -1;
                                        }
                                        if (cont1 != -1)
                                        {
                                                cont1 = 1;

                                        }
                                        else
                                        {
                                                cont1 = 0;
                                                iden--;
                                        }
```

```
                                        //afegir();


                                }

                        }

                }

                catch ( Exception a )
                {

                        Console.WriteLine("Error de conexión");
                        MessageBox.Show("Error: " + a);
                }
        }

        private void button1_Click(object sender, System.EventArgs e)
        {
                if (iden2==0)
                {
                        iden2=1;
                        t = new Thread(new ThreadStart(captura));
                        t.Start();
                }
        }

        public float suma()
        {

                float aux;
                if(preu_tot.Text==Convert.ToString(0))
                {
                        suma_total = 0;
                }

                foreach(DataTable table in tabla.Tables)
                {

                        foreach(DataRow myRow in table.Rows)
                        {
                                try
                                {
                                        aux = Convert.ToSingle(myRow["Precio"]);
                                        if(preu_tot.Text.Length!=0)
                                        {
                                                suma_total =
Convert.ToSingle(preu_tot.Text) + aux ;
                                        }
                                        else
                                        {
                                                suma_total=aux;
                                        }
                                }
                                catch(Exception error)
                                {
                                        MessageBox.Show(error.Message);
                                }
                        }
                }
                return suma_total;
        }

        private void button2_Click_1(object sender, System.EventArgs e)
        {
                suma_total= 0;
        }
    }
}
```

2.2. Biblioteca paquete datos

```
using System;
using System.Collections;
using System.Data;
using System.Threading;
using System.Net;
using System.Net.Sockets;
using System.IO;
using System.Text;
using System.Runtime.Serialization;
using System.Runtime.Serialization.Formatters.Binary;
using System.Data.OleDb;


namespace biblioteca_super
{

        [Serializable]
        public class super
        {
                [Serializable]
                        public class productos
                {
                        public int Identificador;
                        public string nombre;
                        public float precio;



                }

        }
}
```

ANEXO G. RESULTADOS OBTENIDOS

En este anexo, mostraremos todos los resultados obtenido al realizar las pruebas. En el cápitulo 5 solo hemos mostrado los resultados más significantes, debido a las restricciones en la longuitud del texto he optado por crear un anexo con todos los resultados obtenidos.

Como se puede obsevar, se comentan los resultados a medida que incluiamos un número mayor de retransmisión.

1. Resultados obtenidos sin incluir retransmisiones.

Tabla G.1 Tabla de tiempos sin Retx.

	1	2	3	4	5	6
MIN	0,00372396	0,00865994	0,01653429	0,02634332	0,03129774	0,04329102
MED	0,00888067	0,01833864	0,02526188	0,03295654	0,04076017	0,04329102
MAX	0,01419271	0,03102973	0,03359049	0,04072266	0,0485319	0,04329102

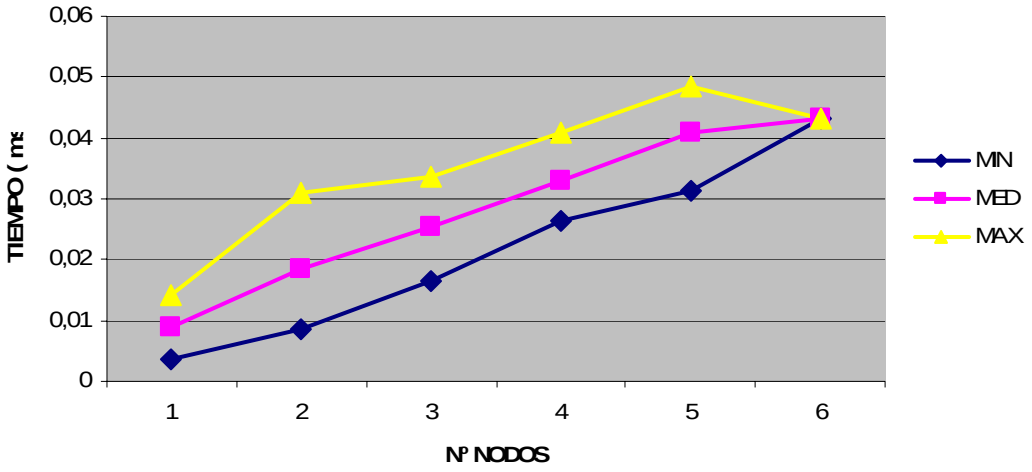


Fig G.1 Gráfico de tiempos sin Retx.

Tabla G.2 Tabla de probabilidad sin Retx.

	1	2	3	4	5	6	7
Prob.Perd.Inf	0%	6,333%	33,333%	94,667%	97,667%	99,667%	100,000%

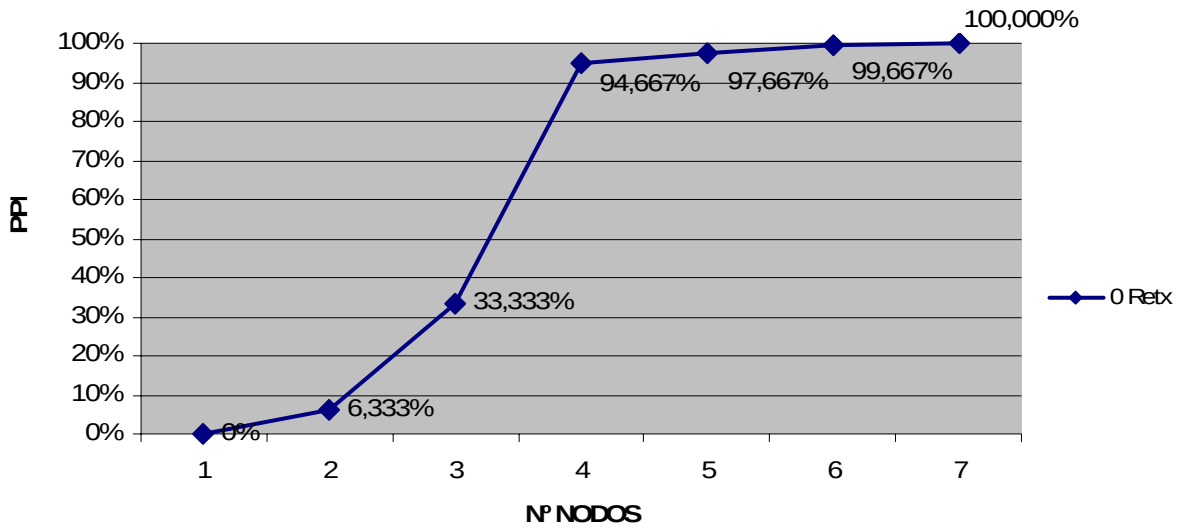


Fig G.2 Gráfico de probabilidad sin Retx.

2. Resultados obtenidos con 1 retransmisión como máximo.

Tabla G.3 Tabla de tiempos con 1 Retx.

	1	2	3	4	5	6	7
MIN	0,00409071	0,00767144	0,01320421	0,01964301	0,03044705	0,03551541	0,04212565
MED	0,00885836	0,01853032	0,02622671	0,03491366	0,04240048	0,04838359	0,05534447
MAX	0,01417209	0,02872938	0,04531793	0,0488444	0,05554796	0,06024848	0,06920139

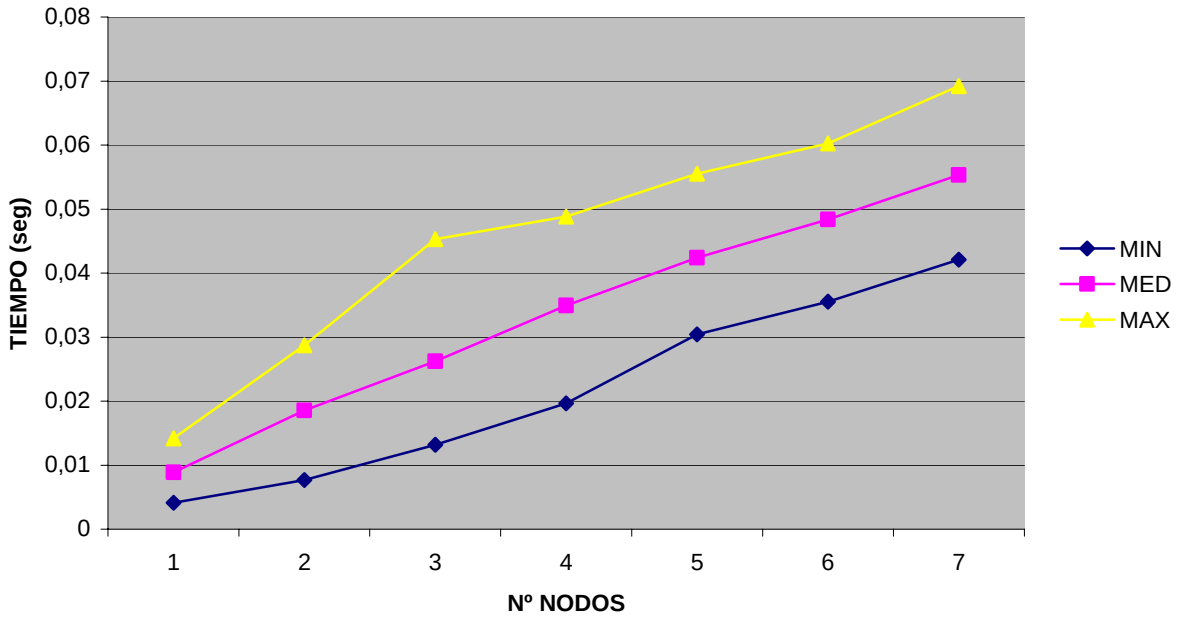


Fig G.3 Gráfico de tiempos con 1 Retx.

Tabla G.4 Tabla de probabilidad con 1 Retx.

	1	2	3	4	5	6	7
Prob.Perd.Inf	0,0000%	0,0000%	3,3333%	43,6667%	63,0000%	83,0000%	90,6667%

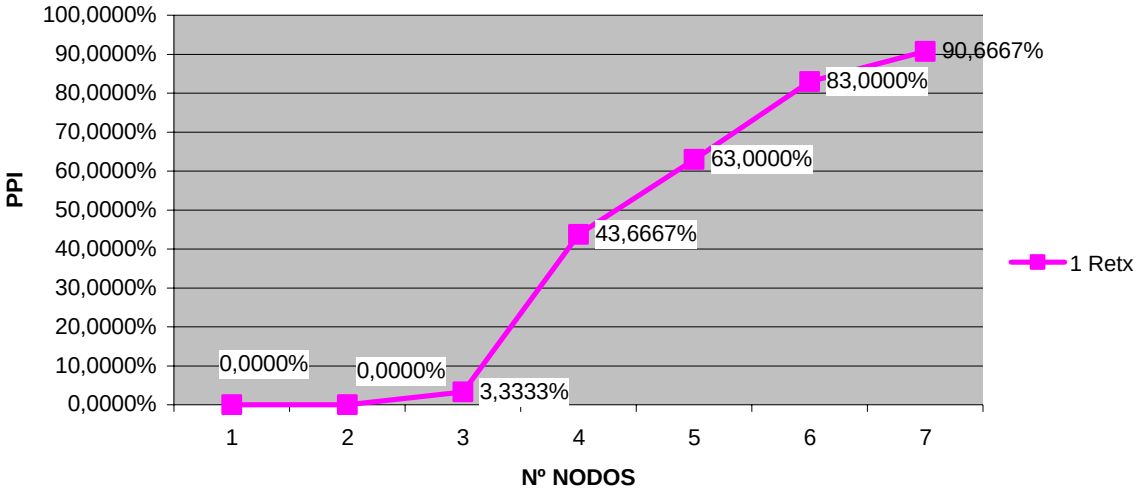


Fig G.4 Gráfico de probabilidad con 1 Retx.

3. Resultados obtenidos con 2 retransmisión como máximo.

Tabla G.5 Tabla de tiempos con 2 Retx.

	1	2	3	4	5	6	7
MIN	0,00373481	0,00837131	0,01406359	0,01936089	0,02828016	0,0343533	0,04086046
MED	0,00887697	0,01817223	0,02814116	0,03806786	0,04478132	0,05285256	0,061708
MAX	0,0138151	0,02661024	0,04553277	0,05456272	0,06010634	0,07198893	0,08687826

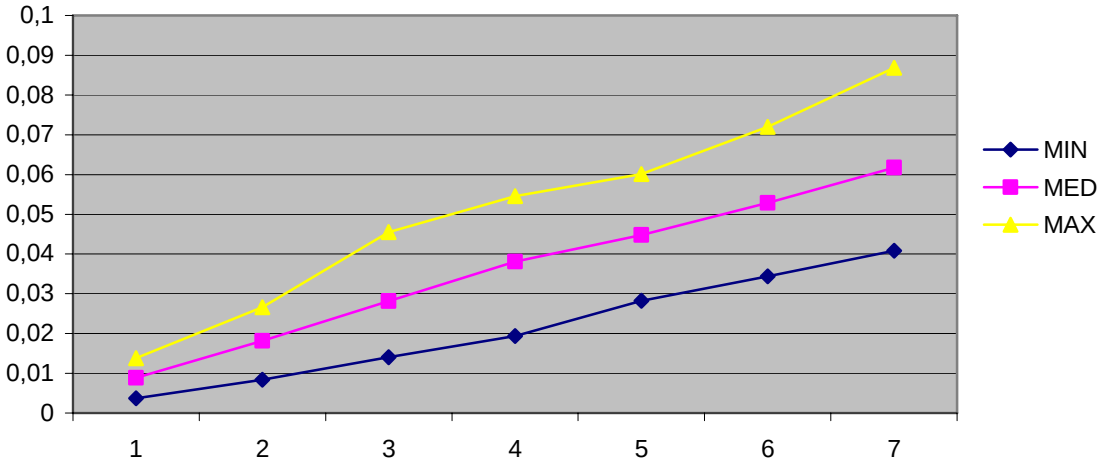


Fig G.5 Gráfico de tiempos con 2 Retx.

Tabla G.6 Tabla de probabilidad con 2 Retx.

	1	2	3	4	5	6	7
2 Retx	0,000%	0,000%	1,000%	2,667%	15,667%	34,667%	58,333%

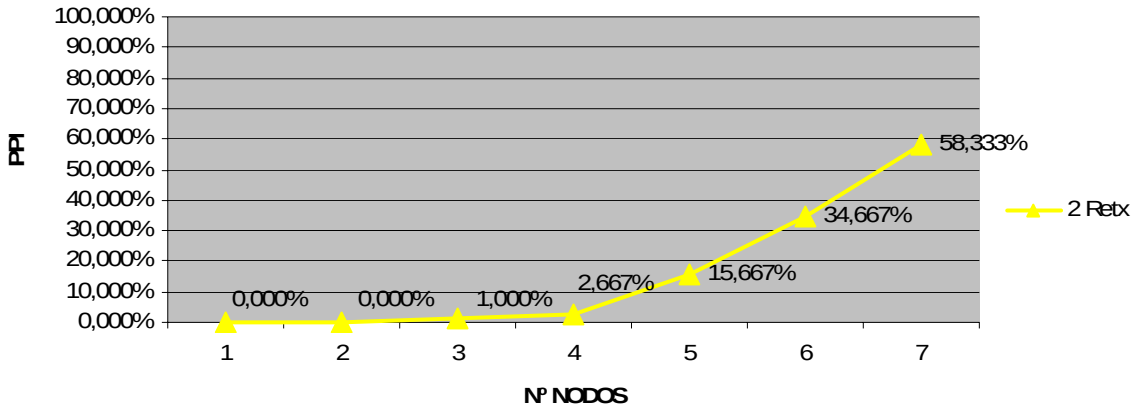


Fig G.6 Gráfico de probabilidad con 2 Retx.

4. Resultados obtenidos con 3 retransmisión como máximo.

Tabla G.7 Tabla de tiempos con 3 Retx.

	1	2	3	4	5	6	7
MIN	0,00373155	0,00656467	0,0145714	0,0185753	0,03077365	0,03657986	0,04614583
MED	0,00888616	0,01822471	0,02775505	0,03890538	0,04599291	0,05464085	0,06381857
MAX	0,01415473	0,02460395	0,04767036	0,06211697	0,07231554	0,07679362	0,08991102

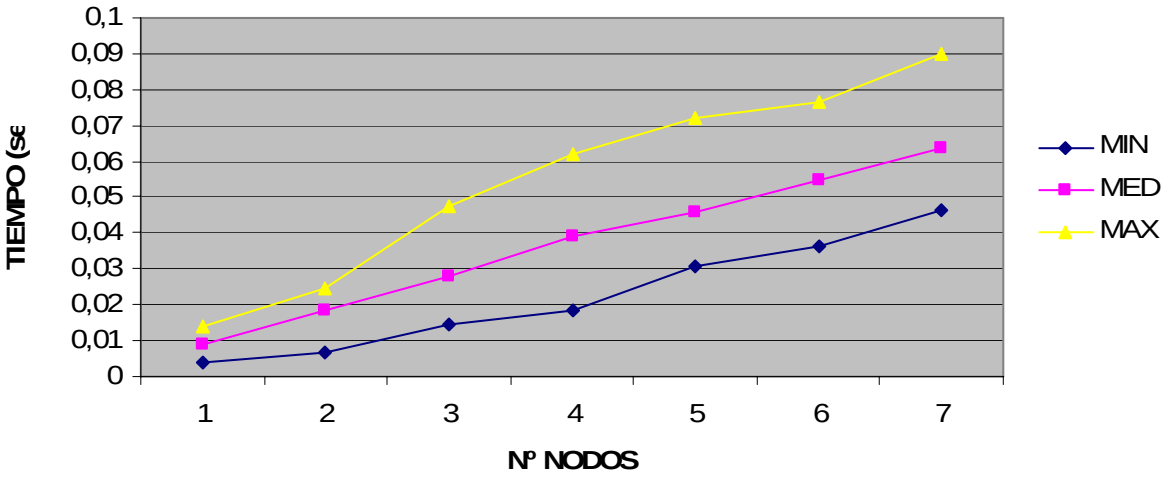


Fig G.7 Gráfico de tiempos con 3 Retx.

Tabla G.8 Tabla de probabilidad con 3 Retx.

	1	2	3	4	5	6	7
3 Retx	0,000%	0,000%	0,667%	0,000%	2,000%	8,000%	18,333%

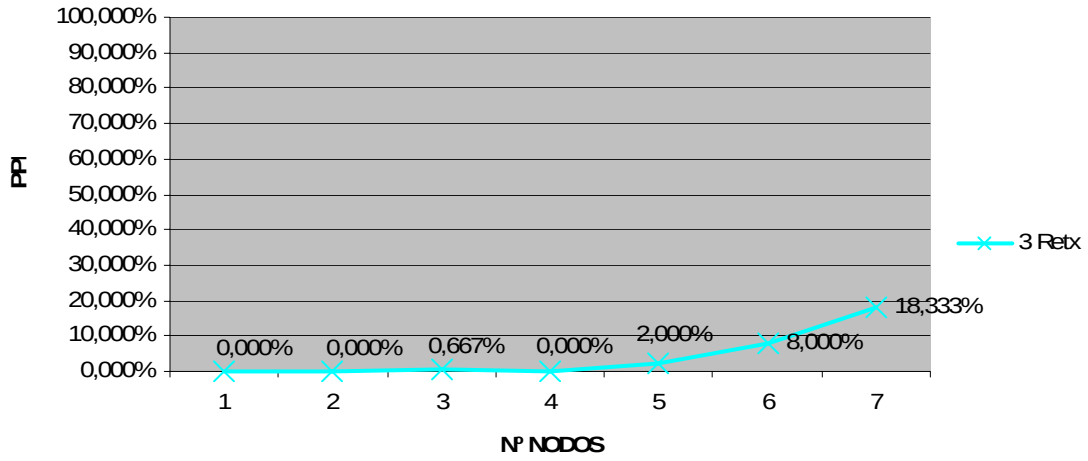


Fig G.8 Gráfico de probabilidad con 3 Retx.

5. Resultados obtenidos con 4 retransmisión como máximo.

Tabla G.9 Tabla de tiempos con 4 Retx.

	1	2	3	4	5	6	7
MIN	0,00374566	0,00660156	0,01723958	0,02171224	0,03157552	0,038227	0,0412207
MED	0,00887756	0,01806381	0,02788685	0,03593224	0,04931349	0,05826428	0,06821699
MAX	0,01378147	0,02397135	0,04645399	0,05535048	0,07044271	0,08778646	0,09536241

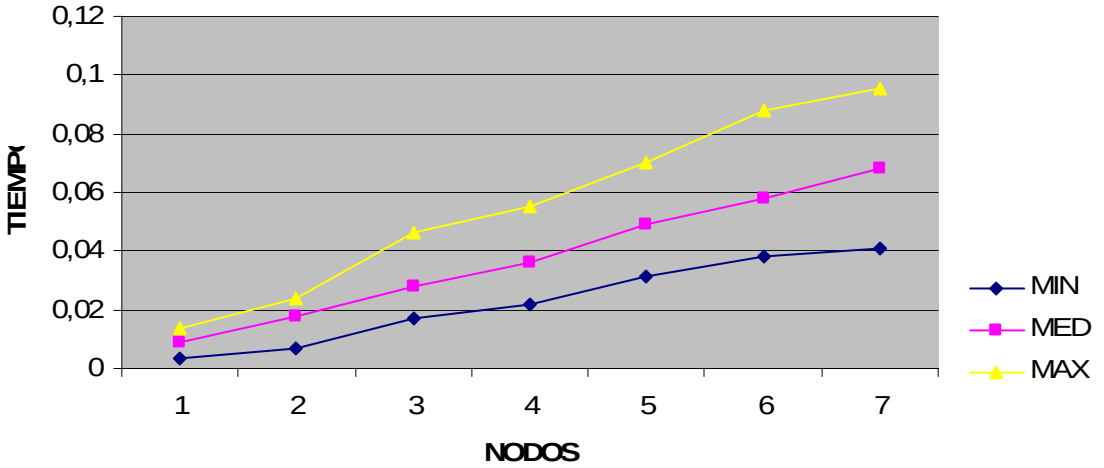


Fig G.9 Gráfico de tiempos con 4 Retx.

Tabla G.10 Tabla de probabilidad con 4 Retx.

	1	2	3	4	5	6	7
4 Retx	0,00%	0,00%	0,00%	0,33%	0,00%	0,667%	3,333%

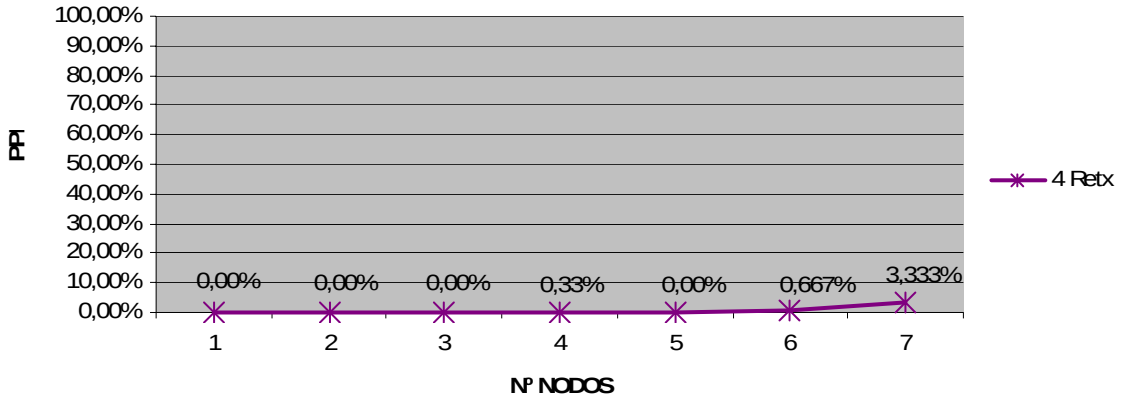


Fig G.10 Gráfico de probabilidad con 4 Retx.

6. Resultados obtenidos con 5 retransmisión como máximo.

Tabla G.11 Tabla de tiempos con 5 Retx.

	7	8	9	10	11	12	13
MIN	0,04027018	0,05253038	0,06001302	0,06423828	0,08109375	0,08390191	0,08837565
MED	0,07046643	0,08019523	0,08898609	0,09442273	0,10813011	0,11716365	0,11917871
MAX	0,10025825	0,12150499	0,12446832	0,12313477	0,14766602	0,15514214	0,15610135

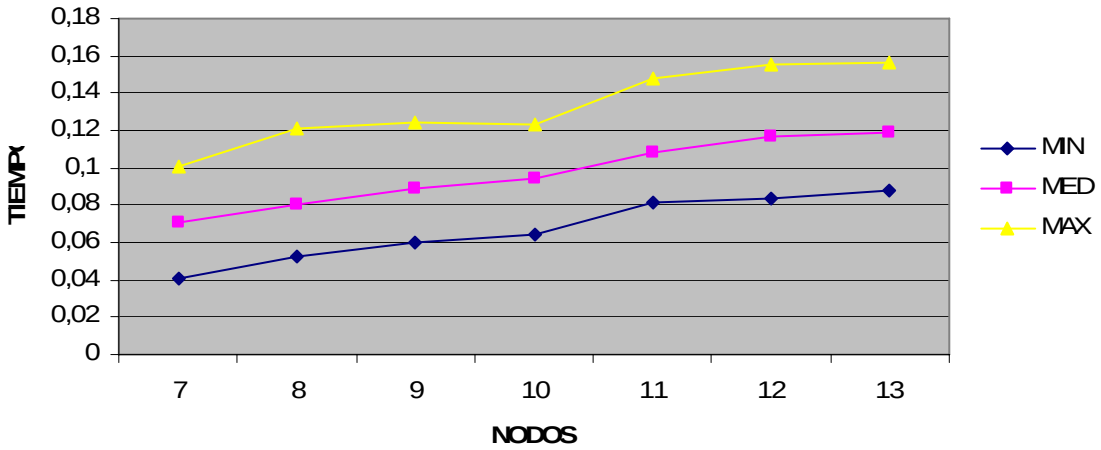


Fig G.11 Gráfico de tiempos con 5 Retx.

Tabla G.12 Tabla de probabilidad con 5 Retx.

	7	8	9	10	11	12	13
5 Retx	3,33%	4,33%	7,33%	9,67%	20,67%	25,33%	29,33%

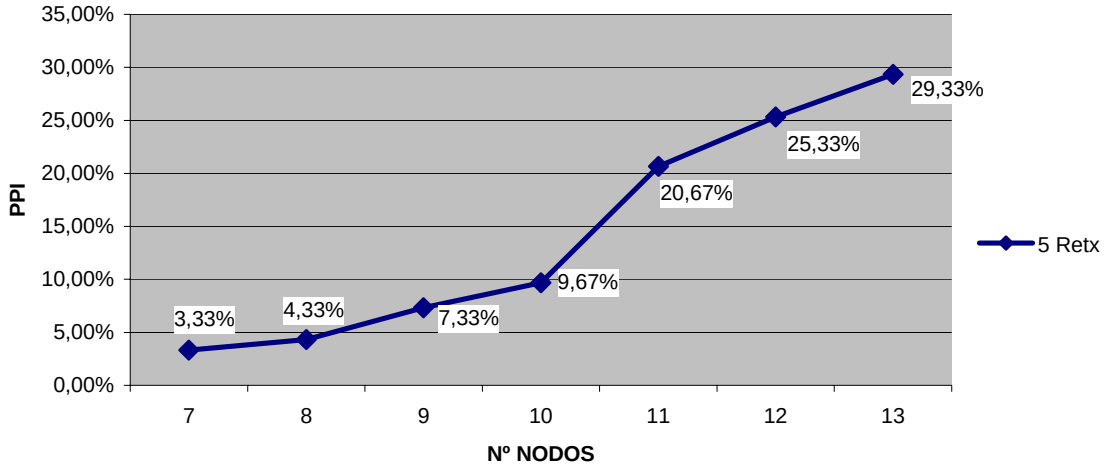


Fig G.12 Gráfico de probabilidad con 5 Retx.

7. Resultados obtenidos con 6 retransmisión como máximo.

Tabla G.13 Tabla de tiempos con 6 Retx.

	7	8	9	10	11	12	13
MIN	0,04319336	0,05106879	0,059949	0,06375326	0,07827799	0,07910265	0,0923036
MED	0,0702841	0,08176349	0,08709156	0,09851095	0,10965775	0,11814669	0,12712902
MAX	0,10374023	0,11687283	0,12310547	0,12994141	0,14477431	0,15019965	0,16008681

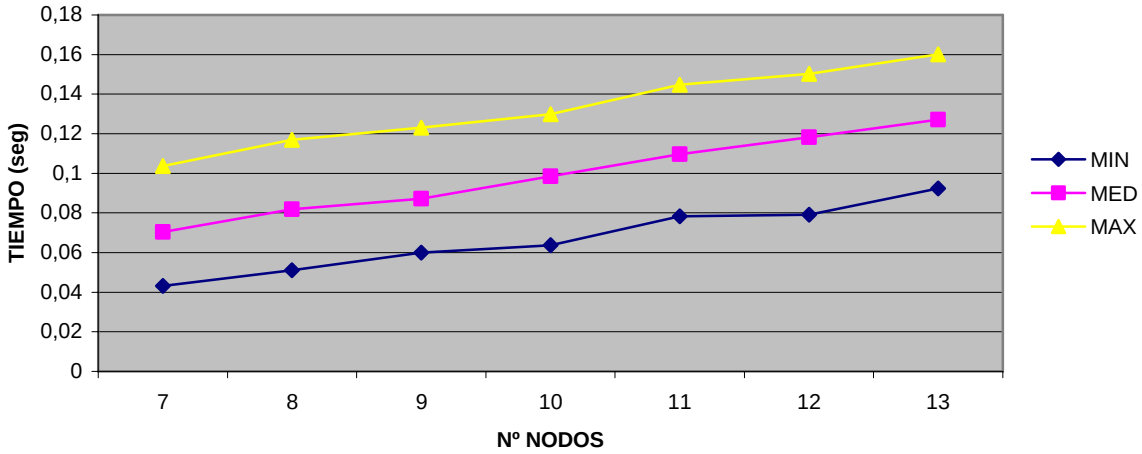


Fig G.13 Gráfico de tiempos con 6 Retx.

Tabla G.14 Tabla de probabilidad con 6 Retx.

	7	8	9	10	11	12	13
6 Retx	0,667%	0,667%	2,000%	1,667%	7,000%	10,000%	16,333%

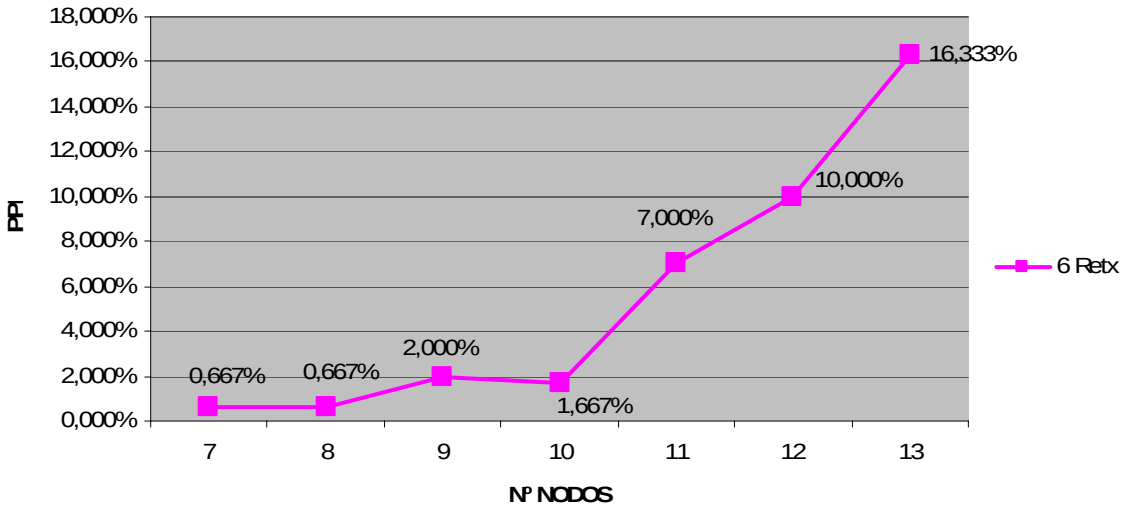


Fig G.14 Gráfico de probabilidad con 6 Retx.

8. Resultados obtenidos con 7 retransmisión como máximo.

Tabla G.15 Tabla de tiempos con 7 Retx.

	7	8	9	10	11	12	13
MIN	0,04461046	0,05021267	0,05840712	0,06762478	0,07588216	0,07769531	0,08516927
MED	0,06844907	0,07799576	0,0877795	0,09838603	0,10794001	0,11722645	0,12347435
MAX	0,10618815	0,12391819	0,12115451	0,13942925	0,15869358	0,15040039	0,16451497

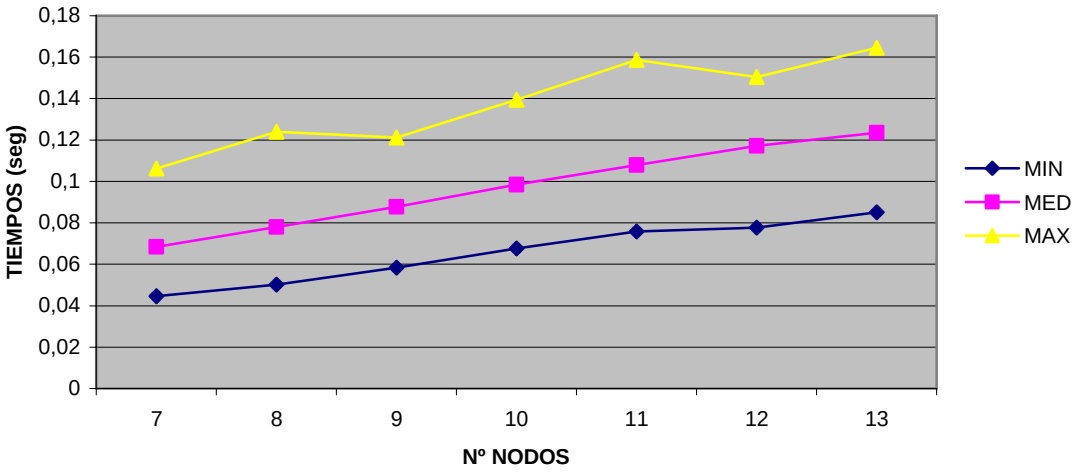


Fig G.15 Gráfico de tiempos con 7 Retx.

Tabla G.16 Tabla de probabilidad con 7 Retx.

	7	8	9	10	11	12	13
7 Retx	0,00%	0,00%	0,67%	0,00%	1,33%	3,33%	3,33%

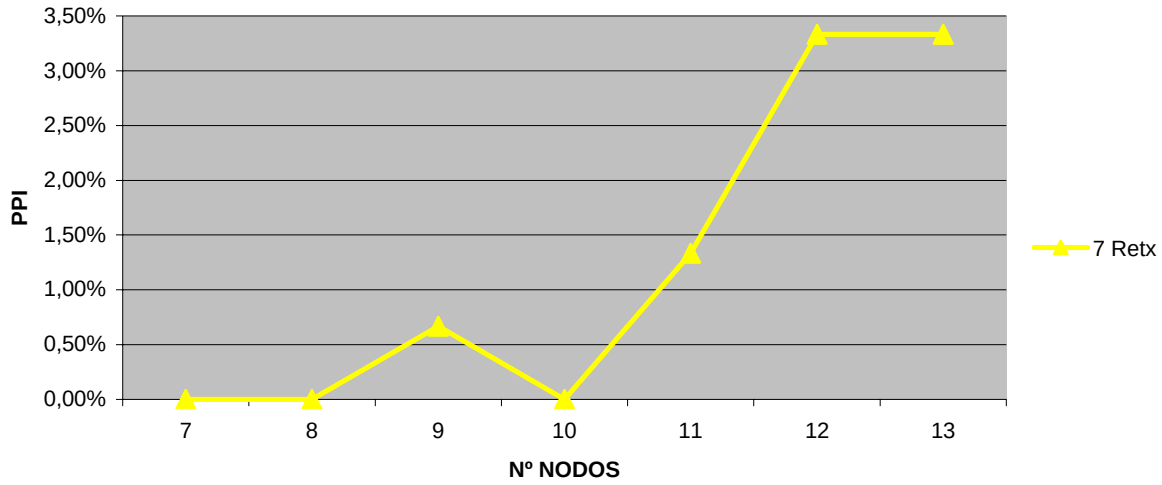


Fig G.16 Gráfico de probabilidad con 7 Retx.

9. Resultados obtenidos con 8 retransmisión como máximo.

Tabla G.17 Tabla de tiempos con 8 Retx.

	7	8	9	10	11	12	13
MIN	0,04535373	0,05511068	0,05597439	0,0734375	0,07474501	0,09123155	0,09512587
MED	0,07240292	0,0823174	0,09375574	0,10382288	0,11245195	0,12145028	0,13191898
MAX	0,10052734	0,11665256	0,12833008	0,1367296	0,16253906	0,16062066	0,16964844

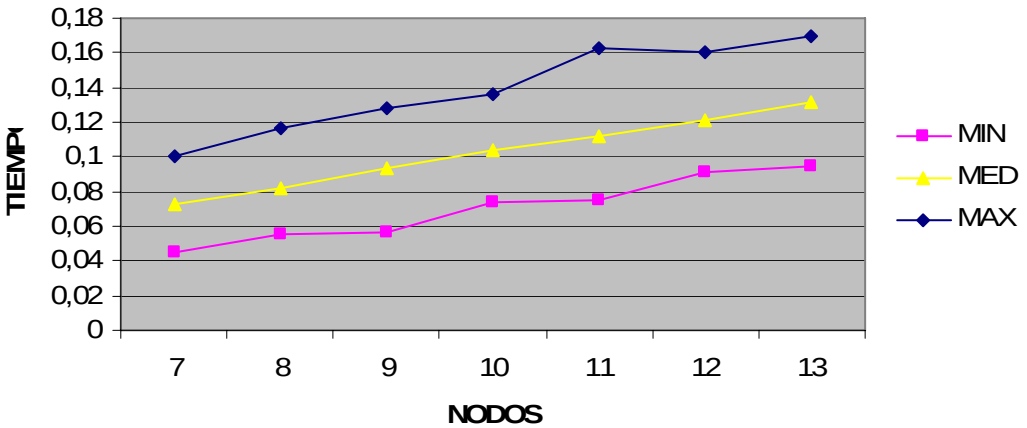
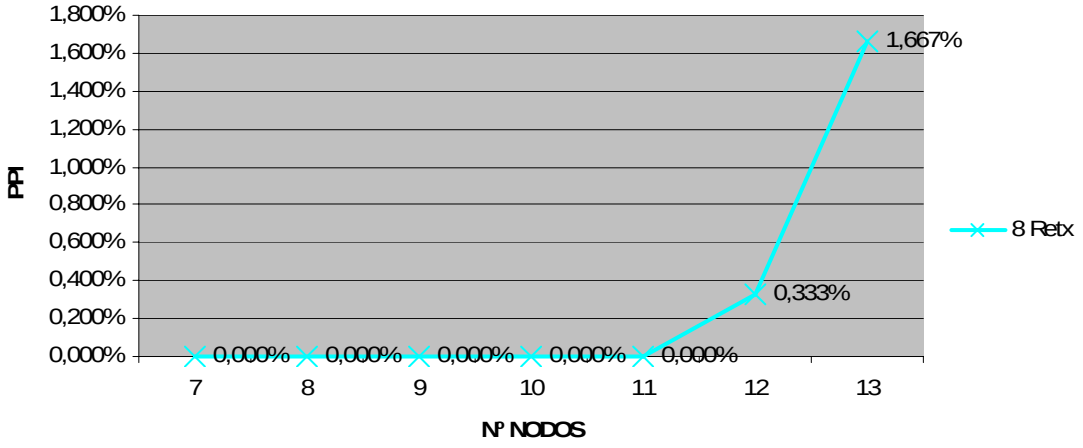


Fig G.17 Gráfico de tiempos con 8 Retx.

Tabla G.18 Tabla de probabilidad con 8 Retx.

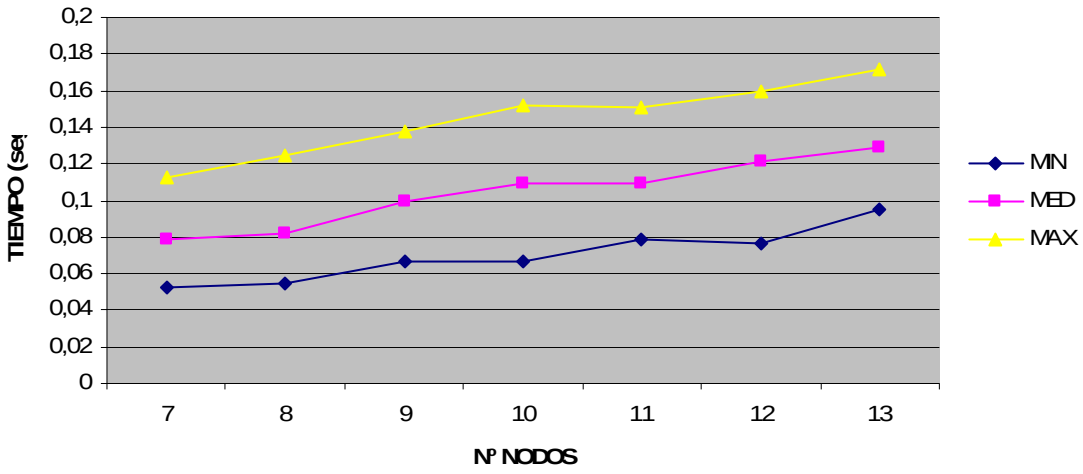
	7	8	9	10	11	12	13
8 Retx	0,000%	0,000%	0,000%	0,000%	0,000%	0,333%	1,667%



10. Resultados obtenidos con 9 retransmisión como máximo.

Tabla G.19 Tabla de tiempos con 9 Retx.

	7	8	9	10	11	12	13
MIN	0,05219256	0,05518338	0,06653971	0,06711263	0,07860677	0,07696181	0,09498481
MED	0,0789542	0,08144992	0,09892688	0,10973038	0,10931555	0,12107926	0,12909666
MAX	0,1125487	0,12503147	0,1380816	0,15147895	0,15115994	0,15955295	0,17153429



ANEXO H. ESTRUCTURAS DE TRAMAS DEL IEEE 802.15.4

En este anexo se realizará una descripción breve sobre el tipo de tramas que contiene el estándar 802.15.4 y su significado.

El estandar 802.15.4 define 4 tipos de trama: trama de beacon, trama de datos, trama ack's y Trama de comandos MAC.

- Trama beacon.

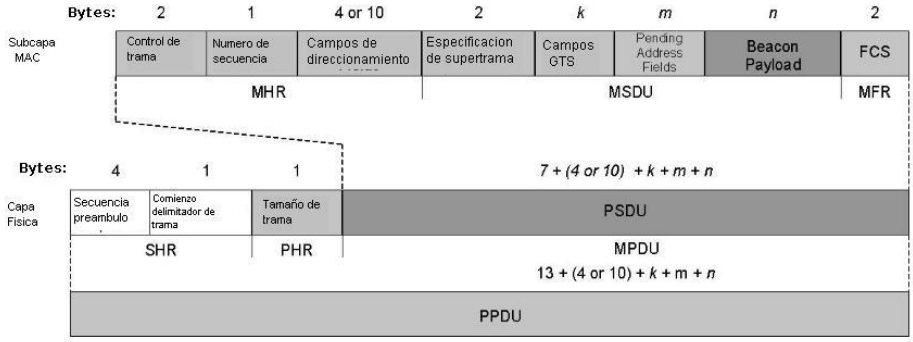


Fig. H.1 Trama beacon

Especificación de supertrama: Este campo especifica diversos parámetros referentes a la supertrama, tales como tiempo de supertrama, quien transmite el beacon o si permite la asociación de nuevos nodos. Nos da información para que el resto de nodos sepan cuando pueden transmitir.

Campos GTS: Aquí se especifica que nodo quiere latencia mínima y slot que ocupará dentro de la trama CFP.

Pending Address Fields: A modo informativo nos dice desde que direcciones se va a transmitir o recibir tramas.

Beacon payload: Este campo se utiliza para llevar información de capas superiores o información encriptada en caso de usar el algoritmo de encriptación AES[14] establecido en el estándar.

[14] AES es un esquema de cifrado por bloque adoptado como un estándar de encriptación por el gobierno de los Estados Unidos, y se espera que sea usado en el mundo entero.

- Trama de datos.

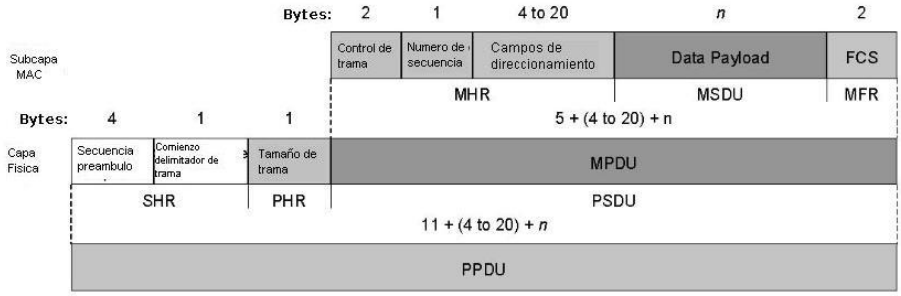


Fig. H.2 Trama de Datos

Data payload: Únicamente contendrá la información indicada por capas superiores.

- Trama de ack's.

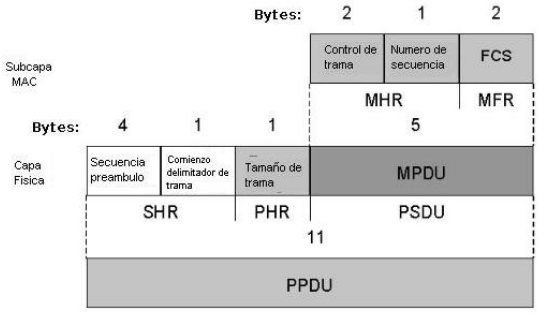


Fig. H.3 Trama de ack's

La trama de ack's no llevan ningún campo específico, únicamente quien envía una trama de datos o una trama de comandos MAC espera recibir, desde la dirección emisora, una trama ACK para confirmar la llegada.

- Trama de comando MAC.

La trama de comandos MAC nos permite llevar a cabo todas las funcionalidades primitivas que define el estándar 802.15.4 tales como asociación, desasociación, resincronización con PAN's coordinators, solicitudes de latencia con QoS (GTS), etc...

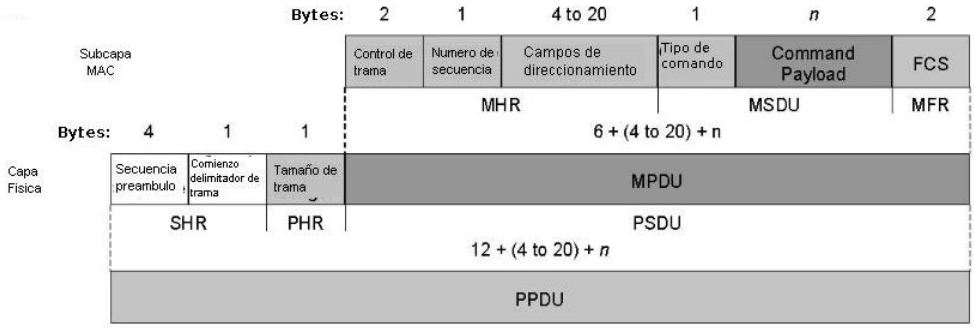


Fig. H.4 Trama de comandos MAC

Tipo de comando: Indica que tipo de comando solicita entre un total de 9 distintos.

Command payload: Su longitud es variable en función del comando utilizado.

2.4.3.1. Campos comunes en todas las tramas

Como podemos ver, cada trama tiene una función distinta muy localizada, excepto la trama de comandos MAC que puede llevar informaciones muy diversas sobre primitivas de servicio.

Los campos comunes en la cabecera MAC (MHR) son:

Control de trama: Nos permite escoger entre las 4 clases de tramas descritas.

Numero de secuencia: Este campo irá en función del paquete de origen y se incrementará para cada clase de trama que sea enviada.

Campos de direccionamiento: En función de si usamos direcciones cortas (16bit) o largas (48bits) de origen y de destino, en las direcciones largas los 20 primeros bits identifican el fabricante del dispositivo y los 28 restantes identifican el dispositivo, las direcciones cortas las asigna el PAN coordinator una vez ha recibido al dispositivo con la dirección larga y se realiza este cambio para tener mas espacio en la trama para transmitir datos utiles.

ANEXO I. CARACTERIZACIÓN ESTADÍSTICA DE TIEMPO DE RECOGIDA DE DATOS.

Para el caso de solo un nodo identificador, hemos realizado los calculos estadísticos del tiempo medio de recogida de dotos. Teniendo la formula original:

$$T_{TOTAL} = (2 \times T_{(backoff)}) + (2 \times T_{TX}) + (4 \times T_{SPI}) + (2 \times T_{SET}) \quad \textbf{(I.1)}$$

Por lo que podemos decir que el tiempo de transmisión, Spi y Set son fijos y los tomamos como una constante K. El unico valor variable son los tiempos de backoff.

$$T_{TOTAL} = T_{(BO\ A)} + T_{(BO\ B)} + K \quad \textbf{(I.2)}$$

$$T_{(BO\ A)}\ ,\ T_{(BO\ B)} = v.a.\ uniforme[0.32,5,32]ms \quad \textbf{(I.3)}$$

Al tener dos variables aleatoria uniforme, se puede saber la probabildad de que salga un determinado valor convolucionando las dos variables aleactorias cono se muestra a continuación:

$$T' = T_{total} - K = T_{(BO\ A)} + T_{(BO\ B)} \quad \textbf{(I.4)}$$

$$f_{T'}(t) = f_{(TBO\ A)}(t) * f_{(TBO\ B)}(t) \quad \textbf{(I.5)}$$

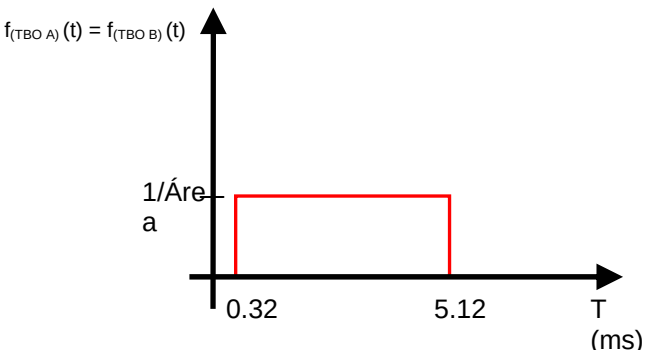


Fig I.1 Funciones de densidad

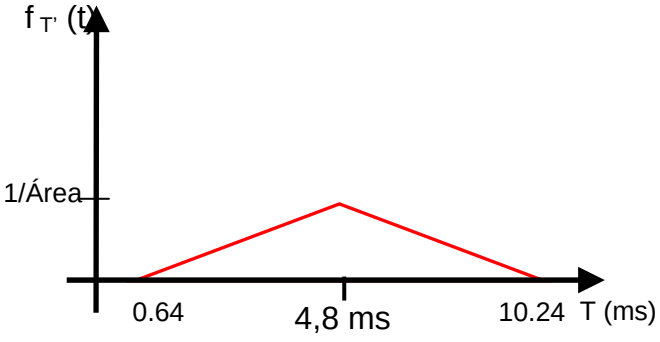


Fig I.2 Funciones Convolucionadas

Como se puede observar, el valor más probable que salga es de 4,8 ms, debido a que es la zona más probable como se observa en la figura I.2.
Si le sumamos la constante K, esta función de densidad se desplazaría K hacia la derecha, aproximándose al valor práctico.