UNIVERSIDAD POLITÉCNICA DE CARTAGENA

PROYECTO FIN DE CARRERA

Desarrollo y Estudio de Capacidades de Redes 802.15.4 Usando Dispositivos Micaz

AUTOR: Juan José García Heredia
DIRECTORES: Antonio Javier García Sánchez
Felipe García Sánchez

Autor	Juan José García Heredia
E-mail	garciaheredia@gmail.com
Director	Antonio Javier García Sánchez Felipe García Sanchez
E-mail	antoniojavier.garcia@upct.es felipe.garcia@upct.es

Titulo del PFC

Desarrollo Y Estudio De Capacidades De Redes 802.15.4 Usando Dispositivos Micaz

Resumen

Las Redes Inalámbricas de Área Personal (WPAN) están destinadas a la interconexión de dispositivos inalámbricos en entornos de oficinas, laboratorios y dentro de los hogares. A diferencia de las redes inalámbricas de área local (WLAN), una conexión que se hace a través de una WPAN suele involucrar a muy poca o ninguna infraestructura o conexiones directas hacia el mundo exterior. Este tipo de tecnología también procura hacer un uso eficiente de recursos, por lo que se han diseñado protocolos simples y óptimos para cada necesidad de comunicación y aplicación.

En este Proyecto Fin de Carrera, se va a tratar de hacer comprender al lector los fundamentos de las redes basadas en este protocolo, así como su funcionamiento. Una vez detallada la tecnología, se comentarán sus características fundamentales y la problemática que se plantea en este Proyecto Fin de Carrera.

De igual forma en este proyecto se tratará la temática de cómo implementar aplicaciones de cara a emplear esta tecnología para un determinado fin. El objetivo de este Proyecto Fin de Carrera es poder analizar las características de una red 802.15.4 y sus limitaciones, siendo necesario para ello, se han diseñado una serie de entornos de trabajo que permiten estudiar distintos aspectos de la tecnología 802.15.4.

Titulación	Ingeniero de Telecomunicación
Departamento	Tecnologías de la Información y Comunicaciones
Fecha de Presentación	23/09/08

Índice general

Índice de figuras

Índice de tablas

Capitulo 1 - Introducción

Las Redes Inalámbricas de Área Personal (WPAN) constituyen en la actualidad un importante desafío tecnológico dentro del escenario inalámbrico. La aparición de estas redes, surge ante la incipiente necesidad de acercar las redes al usuario y la utilización de estas redes para la automatización del entorno de una forma sencilla.

Las Redes Inalámbricas de Área Personal (WPAN) están destinadas a la interconexión de dispositivos inalámbricos en entornos de oficinas, laboratorios y dentro de los hogares. A diferencia de las redes inalámbricas de área local (WLAN), una conexión que se hace a través de una WPAN suele involucrar a muy poca o ninguna infraestructura o conexiones directas hacia el mundo exterior. Este tipo de tecnología también procura hacer un uso eficiente de recursos, por lo que se han diseñado protocolos simples y óptimos para cada necesidad de comunicación y aplicación.

Dentro de este grupo de protocolos simples, surge el estándar IEEE 802.15.4. Las características más importantes del estándar IEEE 802.15.4 son la flexibilidad de la red, bajo costo y bajo consumo de energía. Este estándar se puede utilizar para muchas aplicaciones domóticas e industriales.

En este Proyecto Fin de Carrera, se va a tratar de hacer comprender al lector los fundamentos de las redes basadas en este protocolo, así como su funcionamiento. Una vez detallada la tecnología, se comentarán sus características fundamentales y la problemática que se plantea en este Proyecto Fin de Carrera.

De igual forma en este proyecto se tratará la temática de cómo implementar aplicaciones de cara a emplear esta tecnología para un determinado fin. El objetivo de este Proyecto Fin de Carrera es poder analizar las características de una red 802.15.4 y sus limitaciones, siendo necesario para ello, se han diseñado una serie de entornos de trabajo que permiten estudiar distintos aspectos de la tecnología 802.15.4.

Las aplicaciones desarrolladas serán implementadas sobre hardware basado en la tecnología 802.15.4. Este hardware deberá ser adaptado a las capas PHY y MAC de esta tecnología. Las aplicaciones desarrolladas se implementarán sobre estas capas, permitiendo al usuario poder enviar la información directamente de nodo a nodo dentro de la red WPAN.

Por último, se analizarán diversos parámetros de QoS con el objeto deducir cuáles son las limitaciones de la tecnología a estudio dentro de los distintos escenarios diseñados, con las configuraciones existentes en la actualidad.

Capitulo 2 – Antecedentes

1.- Estándar 802.15.4

A lo largo de esta sección, se va a realizar una breve descripción del estándar 802.15.4.

1.1.- Introducción

Las características más importantes en este estándar son su flexibilidad de red, bajo coste y bajo consumo de energía. Este estándar se puede utilizar para muchas aplicaciones que requieren una tasa baja en la transmisión de datos.

La clave la de motivación para el uso de tecnología inalámbrica es la reducción en los gastos de instalación, ya que nunca es necesario cambiar el cableado. Las redes inalámbricas implican un gran intercambio de información con un mínimo de esfuerzo de instalación. Esta tendencia es impulsada por la gran capacidad de integrar componentes inalámbricos de una forma más barata y el éxito que tienen otros sistemas de comunicación inalámbrica como los móviles.

En el año 2000 dos grupos especialistas en estándares (ZigBee y el grupo 15 de trabajo IEEE 802) se unieron para dar a conocer la necesidad de un nuevo estándar para redes inalámbricas de bajo consumo y por lo tanto bajos costos en ambientes industriales y caseros. El resultado fue que en diciembre de ese año el comité para nuevos estándares IEEE (NesCom) designó oficialmente un nuevo grupo de trabajo para el desarrollo de un nuevo estándar de baja transmisión en redes inalámbricas para áreas personales (LR-WPAN), con lo que nació el estándar que ahora se conoce como el 802.15.4. Algunas características de alto nivel del 802.15.4 se resumen en la siguiente tabla.

Propiedad	**Rango**
Rango de transmisión de datos	868 MHz:20kb/s; 915 MHz:40kb/s; 2.4 GHz:250kb/s.
Alcance	20 – 30 m (indoor).
Latency	Por debajo de los 15 ms.
Canales	868/915 MHz: 11 canales. 2.4 GHz: 16 canales.
Bandas de frecuencia	Dos PHY: 868/915 MHz y 2.4 GHz.
Direccionamiento	Cortos de 8 bits o 64 bits IEEE
Canal de acceso	CSMA-CA y CSMA-CA ranurado
Temperatura	El rango de temperatura industrial: -40° a +85° C

Tabla 1 – Propiedades del IEEE 802.15.4

1.2.- Capas de red

En las redes tradicionales por cable, la capa de red es responsable de la topología de construcción y mantenimiento de la red, así como de nombrarla y de los servicios de enlace, es decir, de las tareas necesarias de direccionamiento y seguridad. Estos mismos servicios existen para redes inalámbricas para el hogar, sin embargo representan un reto mayor por la necesidad de ahorro de energía de estas redes. Las redes que se construyan dentro de esta capa del estándar IEEE 802.15.4 deben autoorganizarse y automantenerse para que de esta forma se reduzcan los costes totales para facilitar su uso.El estándar IEEE 802.15.4 soporta múltiples topologías para su conexión en red, entre ellas la topología tipo estrella y la topología peer-to-peer (Fig 1). La topología a escoger es una elección de diseño y va a estar dado por la aplicación a la que se desee orientar. Algunas aplicaciones como periféricos e interfaces de PC, requieren de conexiones de baja potencia de tipo estrella, mientras que otros como los perímetros de seguridad requieren de una mayor área de cobertura por lo que es necesario implementar una red peer-to-peer.

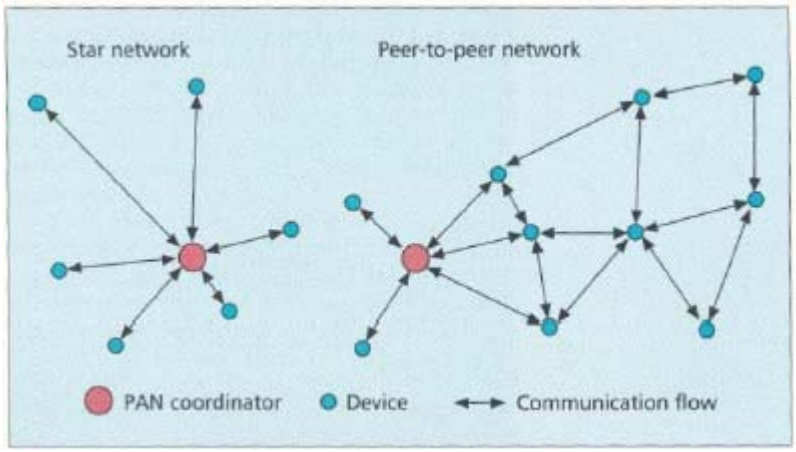

Fig. 1- Redes tipo estrella y peer-to-peer

1.3.- Capa de enlace de datos (data link layer, DLL).

El proyecto IEEE 802 divide al DLL en dos subcapas, la subcapa de enlace de acceso a medios (Medium Access Control, MAC) y la de control de enlaces lógicos (Logical link control, LLC). El LLC es común a todos estándares 802, mientras que la subcapa MAC depende del hardware y varía respecto a la implementación física de esta capa. La figura 2 ilustra la forma en que el estándar IEEE 802.15.4 se basa en la organización internacional para la estandarización (ISO) del modelo de referencia para la interconexión de sistemas abiertos (OSI).

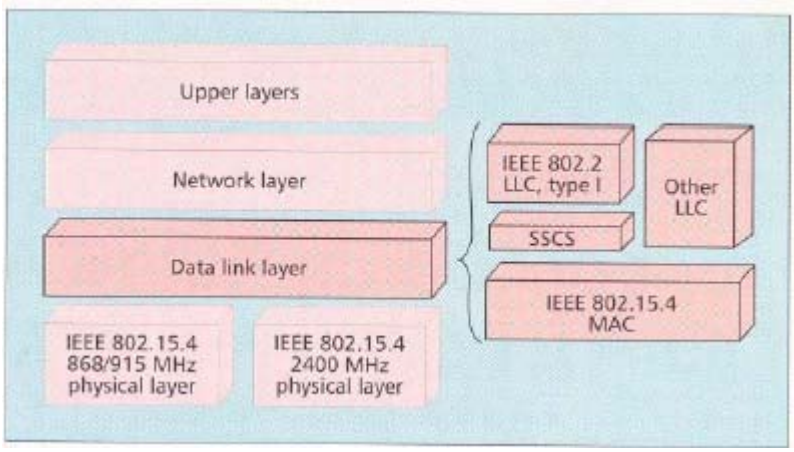

Fig. 2 – Relación del IEEE 802.15.4 con el sistema OSI

Las características del MAC IEEE 802.15.4 son: la asociación y la disociación, reconocimientos de entrega de trama, mecanismos de acceso al canal, validación de trama, garantía del manejo de las ranuras de tiempo, y manejo de guías. Las subcapas MAC proporcionan dos tipos de servicios hacia capas superiores que se acceden a través de dos puntos de acceso a servicios (SAPs). Los servicios de datos MAC se acceden por medio de la parte común de la subcapa (MCPS-SAP), y el manejo de servicios MAC se accede por medio de la capa MAC de manejo de identidades (MLME-SAP). Esos dos servicios proporcionan una interfase entre las subcapas de convergencia de servicios específicos (SSCS) u otro LLC y las capas físicas. El administrador de servicios MAC tiene 26 primitivas. haciéndolo muy versátil para las aplicaciones hacia las que fue orientado.

1.3.1- Formato general de tramas MAC

El formato general de las tramas MAC se diseño para ser muy flexible y que se ajustara a las necesidades de las diferentes aplicaciones con diversas topologías de red al mismo tiempo que se mantenía un protocolo simple. El formato general de una trama MAC se muestra en la figura 3. A la trama del MAC se le denomina unidad de datos de protocolos MAC (MPDU) y se compone del encabezado MAC (MHR), unidad de servicio de datos MAC (MSDU), pie de MAC (MFR). El primer campo del encabezado de trama es el campo de control. Este indica el tipo de trama MAC que se pretende trasmitir, especifica el formato y la dirección de campo y controla los mensajes reconocimiento. En pocas palabras, la trama de control especifica como es el resto de la trama de datos y que es lo que contiene.

El tamaño de las direcciones puede variar entre 0 y 20 bytes. Una trama de datos puede contener información de la fuente y del destinatario, mientras que la trama de enterado no contiene ninguna información de ninguna dirección. Por otro lado una trama de guía solo

tiene información de la dirección de la fuente. Esta flexibilidad en la estructura ayuda a incrementar la eficiencia del protocolo al mantener los paquetes lo más reducido que se puede.

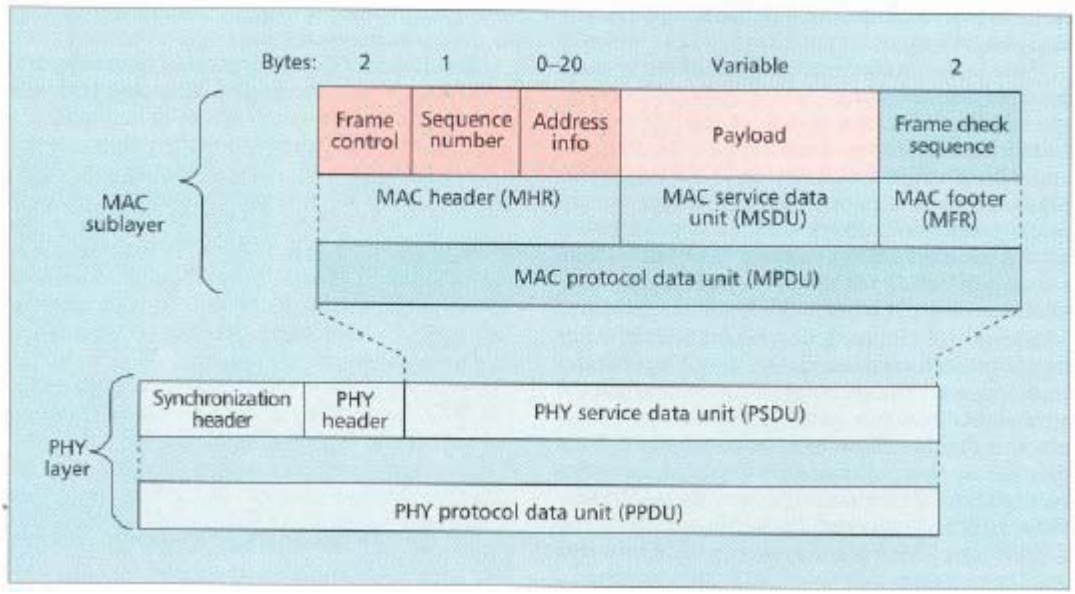

Fig.3– Forma general de la trama MAC

El campo llamado payload es variable en longitud; sin embargo, la trama completa de MAC no debe de exceder los 127 bytes de información. Los datos que lleva el payload depende del tipo de trama. El estándar IEEE 802.15.4 tiene cuatro diferentes tipos de tramas. Esas son la trama de guía, de datos, tramas de enterados y tramas de comandos MAC. Solo las tramas de datos y de guía contienen información provenientes de capas superiores; las tramas de mensajes de enterado y la de comandos MAC originados en el MAC son usados para comunicaciones MAC peer-to-peer.

Otro campo de la trama MAC es una secuencia de números denominada como trama de secuencia de chequeo (Frame Check Séquense). La secuencia de números en los encabezados enlaza a las tramas de acknowledgment con transmisiones anteriores. La transmisión se considera exitosa solo cuando la trama de reconocimiento contiene la misma secuencia de números que la secuencia anterior trasmitida. Las FCS ayudan a verificar la integridad de las tramas del MAC.

1.3.2- La estructura de las super-tramas

Algunas aplicaciones requieren anchos de banda dedicados a lograr estados latentes para un consumo de baja potencia. Para lograr dichos estados latentes el IEEE 802.15.4 se puede operar en un modo opcional llamado superestructuras (superframes).

En un superframe, un coordinador de red, denominado el coordinador PAN, transmite superframes de guía en intervalos definidos. Estos intervalos pueden ser tan cortos como unos 15 ms o tan largos como 245 s. El tiempo entre cada uno de ellos se divide en 16 ranuras de tiempo (slots) independientes a la duración de cada superframe. Un aparato o instrumento puede transmitir cuando desee durante una ranura de tiempo. Pero debe terminar su transmisión antes de la siguiente superframe.

Un superframe consiste en un periodo denominado CAP (Contention Access Period) seguido de otro perido denominado CFP (Contention Free Period) y un periodo inactivo en el cual el coordinador puede entrar en el modo de ahorro de energía y no necesita controlar su red. Un nodo puede transmitir datos en el periodo CAP empleando el algoritmo CSMA-CA, sin embargo el coordinador de PAN puede asignar intervalos o ranuras de tiempo a un solo dispositivo que requiera un determinado ancho de banda permanentemente o transmisiones latentes bajas. Estas ranuras de tiempo asignadas son llamadas ranuras de garantía (GTS) y juntas forman el periodo de contención libre (CPF) (Fig. 3). El tamaño del periodo de contención libre puede variar dependiendo de la demanda de los demás aparatos asociados a la red. Cuando el GTS se utiliza, todos los dispositivos deben de completar todas sus transacciones de contención de base antes de que el periodo de contención libre comience.

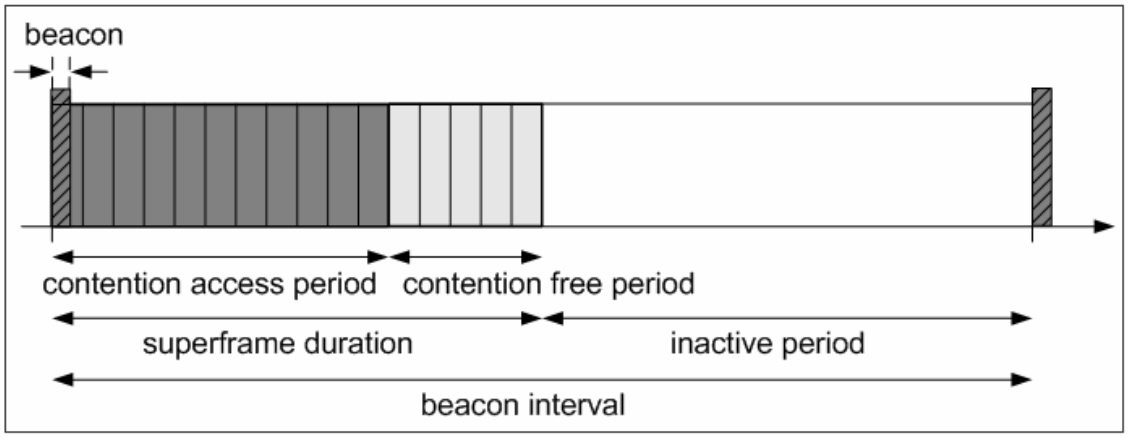

Fig. 4– Estructura de las supertramas

En una red beacon-enabled, se emplea el mecanismo de acceso al canal slotted CSMA/CA, por lo que los dispositivos deben esperar a las tramas de sincronismo para transmitir datos. Cualquier dispositivo, que desee transmitir durante el periodo de acceso de contención, espera a que empiece la siguiente ranura de tiempo y después determina si algún otro dispositivo se encuentra transmitiendo en la misma ranura de tiempo. Si algún otro dispositivo se encuentra transmitiendo en dicho slot, el dispositivo se repliega a un número aleatorio de slots o indica un fallo en la conexión después de varios intentos. Además en una red beacon-

enabled, las tramas de acknowledgment no utilizan CSMA. En la siguiente figura, se muestra el funcionamiento de este tipo de redes.

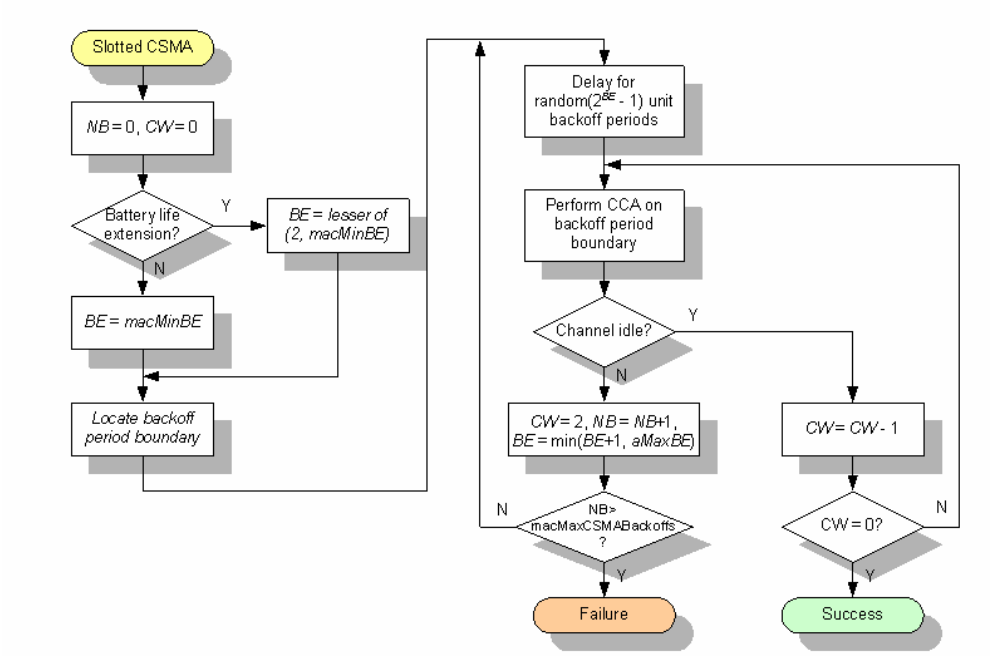

Fig. 5– Slotted CSMA/CA

En una red non-beacon-enabled se emplea el mecanismo de acceso al canal unslotted CSMA/CA, por lo que los dispositivos no deben esperar a las tramas de sincronismo para transmitir datos. En este tipo de redes, los dispositivos están continuamente activos y a la espera de recibir paquetes, lo que requiere requisitos de potencia mayores. En la siguiente figura, se muestra el funcionamiento de este tipo de redes.

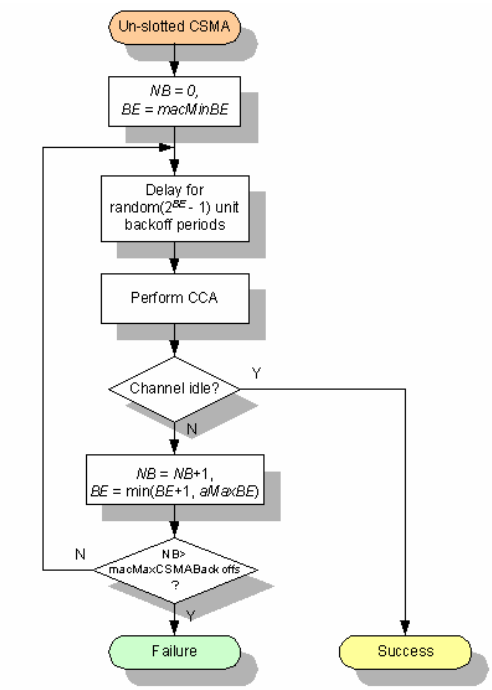

Fig. 6– UnSlotted CSMA/CA

1.3.3.- Mecanismo GTS (Guaranteed Time Slot)

GTS es un método de QoS en el sentido que permite que cada dispositivo tenga una duración específica en cada supertrama, para hacer lo que desee sin contención o latencia.

Algunas aplicaciones requieren banda ancha para conseguir latencia baja. Para acoplar estas bajas latencias, la IEEE 802.15.4 LR-WPAN puede operar en modo opcional de supertrama. El tiempo entre dos beacon es dividido entre ranuras de tiempo iguales independiente de la duración de la supertrama. Un dispositivo puede transmitir en cualquier tiempo durante una ranura, pero la mayoría completa esta transmisión antes de la siguiente supertrama de beacon.

El mecanismo de GTS, asigna un periodo de libre contención para que los dispositivos que lo necesiten, transmitan sus datos sin tener que luchar por el medio mediante el algoritmo de CSMA/CA. Los slots del periodo de libre contención son asignados por el coordinador, por lo que cada nodo solo podrá transmitir en los slots asignados. De esta forma, se pueden asegurar los requisitos de ancho de banda, y lo que es más importante, los requisitos de latencia.

1.3.4.- Escaneo pasivo del medio

Este mecanismo lo emplean los dispositivos para poder asociarse a un coordinador. Su funcionamiento es simple, y consiste en analizar todos los canales disponibles y comprobar si hay algún coordinador enviando beacons en dicho canal. A partir de dicho envío de beacons, se extraen parámetros de calidad de dicho coordinador y se almacenan. Una vez analizados todos los canales, se determina el coordinador que ofrece un mayor factor de calidad, y el dispositivo se asocia a dicho coordinador.

1.3.5.- Mecanismo de orfandad

Tras varios intentos fallidos de comunicación dentro de un cierto tiempo, las capas superiores de la pila de protocolos pueden concluir que el dispositivo ha quedado huérfano. Dicho puede tratar de reajustarse con el grupo al que pertenecía, en cuyo caso inicia el mecanismo de orfandad.

El dispositivo huérfano debe ser capaz de buscar el coordinador al que pertenecía en el canal adecuado. Para ello genera una notificación de orfandad y se queda durante un tiempo escuchado el canal. Si el coordinador recibe una notificación de orfandad de un dispositivo huérfano que anteriormente pertenecía a su grupo, este enviará un comando para la realineación con el coordinador, para que de esta forma el dispositivo que ha quedado huérfano pueda restablecer su condición de miembro de la agrupación. Los dispositivos que reciban dichas

notificaciones y comandos de orfandad, y no sean huérfanos, los ignorarán.

Si mediante el mecanismo de orfandad no se logra localizar el grupo adecuado y su coordinador, el dispositivo puede iniciar un escaneo pasivo o activo con el fin de encontrar otro grupo para unirse, o quizás empezar su propio grupo.

1.3.6- Otras características MAC

Una función importante del MAC es la confirmación de recepciones exitosas de frames de algún dispositivo. Las recepciones exitosas y las validaciones de datos o comandos MAC se confirman por medio de acknowledgment. Si el dispositivo de recepción no es capaz de recibir la información en ese momento por algún motivo, el receptor no manda ningún acknowledgment. El campo de control en el frame indica si se espera un acknowledgment o no. El frame que contiene al acknowledgment se manda de regreso inmediatamente después de que se hace una validación exitosa del frame de entrada. Los frames de guía (beacon frames) mandados por el coordinador del PAN y los frames de acknowledgments nunca son respondidos con algún acknowledgment.

El estándar IEEE 802.15.4 proporciona tres niveles de seguridad: sin seguridad (por ejemplo, aplicaciones de publicidad), control de acceso a listas (sin seguridad criptográfica) y seguridad con clave simétrica.

1.4.- Capa física.

El IEEE 802.15.4 ofrece dos opciones de PHY que combinan con el MAC para permitir un amplio rango de aplicaciones en red. Ambas PHYs se basan en métodos de secuencia directa de espectro extendido (DSSS). Se utiliza DSSS debido a los bajos costes de implementación digital en circuitos integrados, y ambas comparten la misma estructura básica de paquetes low-duty-cycle con operaciones de bajo consumo de energía. La principal diferencia entre ambas PHYs radica en la banda de frecuencias. La PHY de los 2.4 GHz, especifica operación en la banda industrial, médica y científica (ISM), que prácticamente está disponible a nivel mundial, mientras que la PHY de los 868/915 MHz especifica operaciones en la banda de 865 MHz en Europa y 915 MHz en la banda ISM en Estados Unidos. Mientras que la movilidad entre países no se anticipa para la mayoría de las aplicaciones de redes en las casas, la disponibilidad internacional de la banda de los 2.4 GHz ofrece ventajas en términos de mercados más amplios y costos de manufactura más bajos. Por otro lado las bandas de 868 MHz y 915 MHz ofrecen una alternativa a la cogestión creciente y demás interferencias (hornos de microondas, etc) asociadas a la banda de 2.4 GHz. Y mayores rangos por enlace debido a que existe menores pérdidas de propagación.

Existe una segunda distinción de las características de la PHY es el rango de transmisión. La PHY de 2.4 GHz permite un rango de transmisión de 250 kb/s, mientras que la PHY de los 868/915 MHz ofrece rangos de transmisión de 20 kb/s y 40 kb/s respectivamente. Este rango superior de transmisión en la PHY de los 2.4 GHz se atribuye principalmente a un mayor orden en la modulación, en la cual cada símbolo representa múltiples bits. Los diferentes rangos de transmisión se pueden explotar para lograr variedad de objetivos o aplicaciones. Por ejemplo la baja densidad de datos en la PHY de los 868/915 MHz se puede ocupar en lograr una mayor sensitividad y mayores áreas de cobertura, con lo que se reduce el número de nodos requeridos para cubrir un área geográfica, mientras que el rango superior de transmisión en la PHY de los 2.4 GHz se puede utilizar para conseguir throughput mayores y poca latencia. Se espera que en cada PHY se encuentren aplicaciones adecuadas a ellas y a sus rangos de transmisión.

1.5.- Canalización

En el IEEE 802.15.4 se definen 27 canales de frecuencia entre las tres bandas (ver figura 7 y tabla 2). La PHY de los 868/915 MHz soporta un solo canal entre los 868 y los 868.6 MHz , y diez canales entre los 902.0 y 928.0 MHz. Debido al soporte regional de esas dos bandas de frecuencias, es muy improbable que una sola red utilice los 11 canales. Sin embargo, las dos bandas se consideran lo suficientemente cercanas en frecuencia que se puede utilizar el mismo hardware para ambos y así reducir costos de manufactura. La PHY de los 2.4 GHz soporta 16 canales entre los 2.4 y los 2.4835 GHz con un amplio espacio entre canales (5 MHz) esto con el objetivo de facilitar los requerimientos de filtrado en la transmisión y en la recepción.

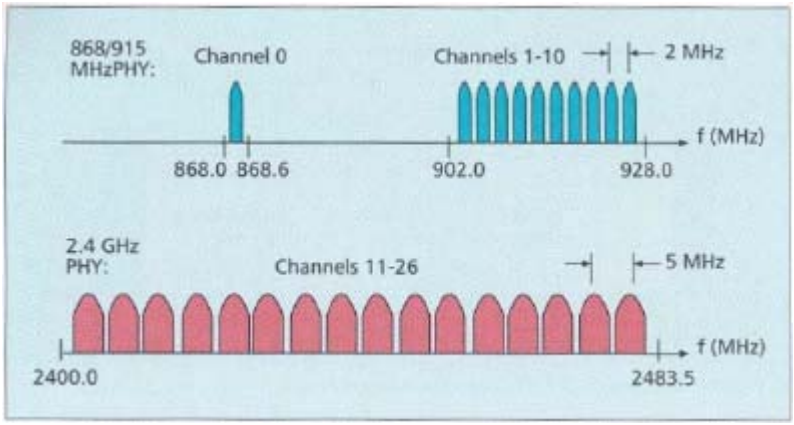

Fig. 7 Estructura de canales del IEEE 802.15.4

Channel number	Channel center frequency (MHz)
$k = 0$	868.3
$k = 1, 2, \ldots, 10$	$906 + 2(k - 1)$
$k = 11, 12, \ldots, 26$	$2405 + 5(k - 11)$

Tabla 2– Frecuencia de los canales IEEE 802.15.4

Dado que el hogar es propenso a tener múltiples redes inalámbricas trabajando en las mismas bandas de frecuencias, así como una interferencia no intencionada de las diferentes aplicaciones, la capacidad de relocalización dentro del espectro será un factor importante en el éxito de las redes inalámbricas. El estándar fue diseñado para implementar una selección dinámica de canales, a través de una selección específica de algoritmos la cual es responsabilidad de la capa de red. La capa MAC incluye funciones de búsqueda que sigue paso a paso a través de una lista de canales permitidos en busca de una señal de guía, mientras que la PHY contiene varias funciones de bajo nivel, tales como la detección de los niveles de energía recibidos, indicadores de calidad en el enlace así como de conmutación de canales, lo que permite asignación de canales y agilidad en la selección de frecuencias. Esas funciones son utilizadas por la red para establecer su canal inicial de operación y para cambiar canales en respuesta a una pausa muy prolongada.

1.6.- Estructura de paquetes de información.

Para mantener un una interfaz común y simple con el MAC, ambas capas PHY comparten una estructura simple del paquete (figura 8). Cada paquete, o unidad de datos del protocolo PHY (PPDU), contiene un encabezado de sincronización, un encabezado de PHY para indicar la longitud del paquete, y la carga de información, o la unidad de secuencia PHY (PSDU). El preámbulo de 32 bits esta diseñado para la adquisición de símbolos y para los tiempos de chip, y en algunos casos se utiliza para ajustes bruscos en la frecuencia. No se requiere una ecualización en el canal de la PHY debido a la combinación de áreas de cobertura pequeñas con rangos bajos de transmisión.

Dentro del encabezado del PHY, se utilizan 7 bits para especificar la longitud de la carga de datos (en bytes). La longitud de paquetes va de 0 a 127 bytes, a través del overhead de la capa MAC, paquetes con longitud cero no ocurren en la práctica.

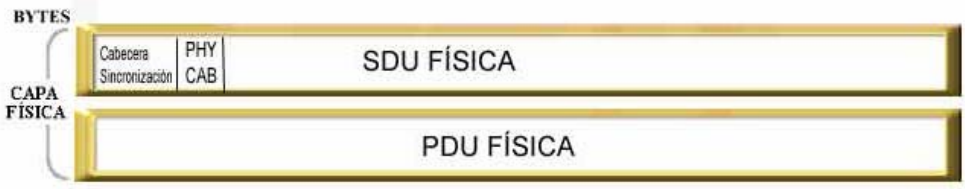

Fig. 8 – Paquete PHY

1.7.- Modulación

La PHY en los 868/915 MHz utiliza una aproximación simple DSSS en la cual cada bit transmitido se representa por un chip-15 de máxima longitud de secuencia (secuencia m). Los datos binarios son codificados al multiplicar cada secuencia m por +1 o -1, y la secuencia de chip que resulta se modula dentro de la portadora utilizando BPSK (binary phase shift keying). Antes de la modulación se utiliza una codificación de datos diferencial para permitir una recepción diferencial coherente de baja complejidad.

La PHY de los 2.4 GHz emplea una técnica de modulación semi-ortogonal basada en métodos de DSSS (con propiedades similares). Los datos binarios son agrupados en símbolos de 4 bits, y cada símbolo especifica una de las 16 secuencias de transmisión semi-ortogonales de código de seudo-ruido (PN). Las secuencias de PN son concadenadas para que sean datos de símbolos exitosos, y la secuencia agregada al chip es modulada en la portadora utilizando MSK (minimum shift keying). El uso de símbolos "casi ortogonales" simplifica la implementación a cambio de un desempeño ligeramente menor (< 0.5 dB). Los parámetros de modulación para ambas PHY se resumen en la tabla 3.

PHY (MHz)	***Banda***	**Parámetro de Datos.**			**Parámetros de riego.**	
		Velocidad de bits (kb/s)	***Velocidad de símbolos (kbaud)***	***Modulación***	***Velocidad de chip (Mchips/s)***	***Modulación***
868/915 (MHz)	*868.0-868.6 (MHz)*	20	20	BPSK	0.3	BPSK
	902.0-928.0 (MHz)	40	40	BPSK	0.6	BPSK
2.4 (GHz)	*2.4-4.4835 (GHz)*	250	62.5	16-semi ortogonal.	2.0	O-QPSK

Tabla 3 – Parámetros de modulación.

En términos de eficiencia (energía requerida por bit), la señalización ortogonal mejora su desempeño en 2 dB que BPSK diferencial. Sin embargo, en términos de sensitividad de recepción, la PHY 868/915 PHY tiene una ventaja de 6-8 dB debido a que tiene velocidades de transmisión más bajas. Por supuesto, que en ambos casos las pérdidas de implementación debido a la sincronización, forma del pulso, simplificaciones en el detector, y demás resultan en desviaciones en sus curvas óptimas de detección.

1.8.- Sensitividad y rango

Las especificaciones actuales de sensitividad del IEEE 802.15.4 especifican -85 dBm para la PHY de los 2.4 GHz y de -92 dBm para la PHY de los 868-915 MHz. Dichos valores incluyen suficiente margen para las tolerancias que se requieren debido a las imperfecciones en la manufactura de la misma manera que permite implementar aplicaciones de bajo costo. En cada caso, los mejores artículos deben de ser del orden de 10 dB mejores que las especificaciones.

Naturalmente el rango deseado estará en función de la sensitividad del receptor así como de la potencia del transmisor. El estándar especifica que cada dispositivo debe de ser capaz de transmitir al menos 1 mW, pero dependiendo de las necesidades de la aplicación, la potencia de transmisión puede ser mayor o menor, la potencia actual de transmisión puede ser menor o mayor (dentro de los límites de regulación establecidos). Los dispositivos típicos (1mW) se espera que cubran un rango de en tre 20-30 m, sin embargo, con una buena sensitividad y un incremento moderado en la potencia de transmisión, una red con topología tipo estrella puede proporcionar una cobertura total para una casa. Para aplicaciones que requieran mayor tiempo de latencia, la topología tipo mesh ofrecen una alternativa atractiva con coberturas caseras dado que cada dispositivo solo necesita suficiente energía para comunicarse con su vecino más cercano.

1.9.- Interferencia de y para otros dispositivos

Los dispositivos que operan en la banda de los 2.4 GHz pueden recibir interferencia causada por otros servicios que operan en dicha banda. Esta situación es aceptable en las aplicaciones que utilizan el estándar IEEE 802.15.4, las cuales requieren una baja calidad de servicio (QoS), no requiere comunicación asíncrona, y se espera que realice varios intentos para completar la transmisión de paquetes. Por el contrario, un requerimiento primario de las aplicaciones del IEEE 802.15.4 es una larga duración en baterías. Este objetivo se logra con poca energía de transmisión y muy pocas ciclos de servicio. Dado que los dispositivos IEEE 802.15.4 se la pasan dormidos el 99.9 por ciento del tiempo, y ocupan transmisiones de baja energía en el spread spectrum, deben estar trabajando en las vecindades de la banda de los 2.4 GHz.

2.- Tecnología

A lo largo de esta sección, se va a comentar la tecnología y el kit de desarrollo empleados en este Proyecto Fin de Carrera.

2.1.- MicaZ

Los motes MicaZ son dispositivos 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, 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.

Cada Mote MICAz incorpora un microprocesador ATMEL Mega128L de 7 MHz (Fig. X). Dicho procesador incorpora una EPROM 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 a una memoria flash externa de 512 kbytes de capacidad.

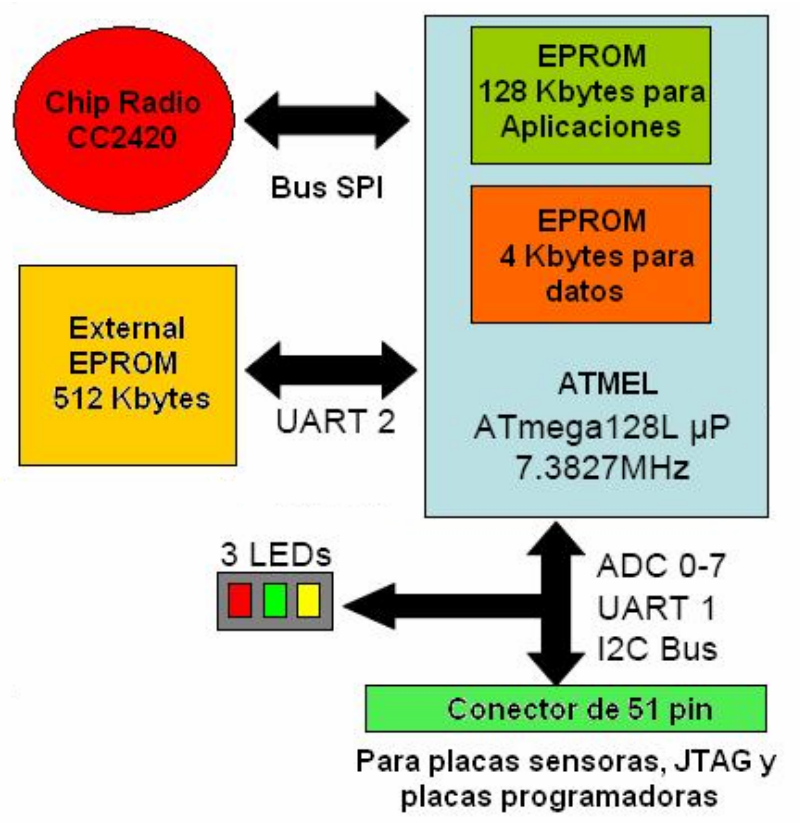

Fig. 9 - Diagrama de bloques de un mote MicaZ

2.2.- Crossbow

Crossbow ha estado en la vanguardia de la tecnología de sensores durante más de una década, y ha vendido cientos de miles de sensores a más de 4,000 clientes por todo el mundo. Hoy, Crossbow es el principal proveedor de tecnología de sensores inalámbricos y sensores MEMS de inercia para la navegación y el control.

2.2.1.- WSN OEM Design Kit.

Los sensores inalámbricos Crossbow permiten la instalación de una red extremo a extremo de forma rápida y económica. En este

Proyecto Fin de Carrera se emplea la tecnología MicaZ de Crossbow, incluida en WSN OEM Design Kit.

El Crossbow OEM Design Kit permite el desarrollo rápido de sistemas de red de sensor inalámbricos.

El Crossbow OEM Design Kit es empleado en sistemas que usan la banda de frecuencia de 2.4 GHz. El equipo provee un programador de módulo, la pasarela de diseño de referencia preprogramada, la pasarela de sensor/adquisición de datos y una estación base con conectividad a Ethernet. En la tabla 4 se muestra el contenido completo del Kit.

Cantidad	***Dispositivo***
5	MICAz OEM Edition Modules
5	MICAz OEM Edition Reference Boards
3	MDA300 Sensor/Data Acquisition Boards
1	MDA100 Sensor/Data Acquisition Board
1	MIB600 Ethernet Gateway
1	Programming/Debugging Pod
1	OEM Module Programmer

Tabla 4 – Contenido del Crossbow OEM Design Kit

2.2.2.- MoteView

MoteView ha sido diseñado para hacer de interfaz entre el usuario y una red de sensores inalámbricos en funcionamiento. MoteView proporciona a los usuarios las herramientas para simplificar el despliegue y monitorización. También facilita la conexión a una base de datos, para poder analizar y hacer gráficas a partir de las lecturas de los sensores.

MoteView soporta todas las tarjetas sensoras y de adquisición de datos de Crossbow. MoteView trabaja con la serie MICA (plataforma de red de sensores inalámbricos), incluyendo las familias de motesMICA2, MICA2DOT, y MICAz. Además se pueden desarrollar y monitorizar plataformas de sensores integrados como el sistema de seguridad de detección de intrusos basado en motes MSP o el sistema de monitorización ambiental, basado en los motes MEP.

2.2.3.- MoteWorks

El desarrollo de las aplicaciones de sensor a medida se permite con la plataforma software de MoteWorks de Crossbow, que está disponible como opción con el kit. MoteWorks se optimiza específicamente para las redes con batería de baja potencia y proporciona ayuda para:

- *Dispositivos sensores*: Pila de protocolos de la red y sistema operativo, soporte de los estándares (802.15.4), programación al alcance de la mano y herramientas de desarrollo.

- *Gateways del servidor*: Software intermedio (middleware) para conectar redes de sensores inalámbricas con información de empresa y sistemas de gestión

- *Interfaz de usuario*: Aplicación del cliente para el análisis y la supervisión remota, gerencia y configuración de la red de sensores.

El software **MoteWorks** no necesita desarrollar la aplicación para la programación remota de la red de sensores, ya que permite configurar cada nodo y capturar sus datos. Además, proporciona un software adicional Sniffer que permite visualizar el estado radio-eléctrico de la red de sensores e incluye un soporte vía e-mail desde la página web de Crossbow. En la figura 10 se muestra la arquitectura de MoteWorks.

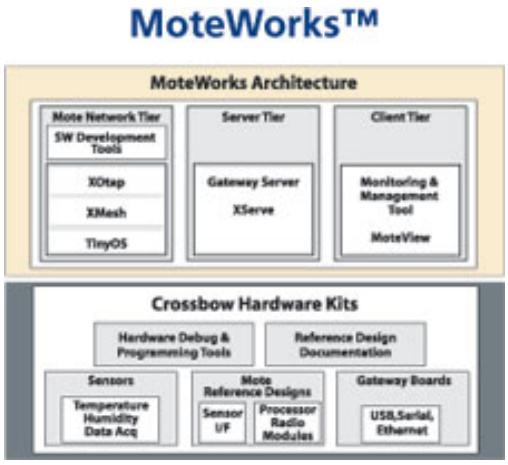

Fig. 10 –Arquitectura de MoteWorks

2.2.4.- MoteConfig

MoteConfig es un GUI basado en Windows empleado para la programación de Motes. Esta utilidad proporciona una interfaz para la configuración y la descarga de firmware de aplicaciones pre-compiladas XMesh / TinyOS en Motes. MoteConfig permite al usuario configurar el ID Mote, identificador de grupo, canal de RF y de potencia de RF. El usuario también tiene la posibilidad de la programación over-the-air-programming presente en todos los XMesh (basada en el firmware). Todas los firmwares de alta potencia y baja potencia XMesh están disponibles para cada dispositivo y sensor de plataforma fabricados por Crossbow.

3.- TinyOS (TINY MICROTHREADING OPERATING SYSTEM)

TinyOs es un Sistema operativo diseñado específicamente para sensores en red, con el objetivo de llenar el vacío existente entre las capacidades hardware y el sistema completo. Este sistema es capaz de manejar las capacidades limitadas del hardware

eficientemente y está programado mediante el lenguaje de programación NesC, el cual se explica en el apartado 3.1.

Está diseñado para escalar con las tendencias tecnológicas de la actualidad y soporta operaciones intensivas de concurrencia y garantiza atomicidad (es posible construir bloques de sentencias que se ejecutan todo o nada). Emplea un paradigma de comunicación basado en Active Message y no maneja memoria dinámica:

• Las zonas de memoria se localizan en memoria estática.

• No existe memoria dinámica. No hay punteros.

• Una única pila asignada la tarea que se ejecuta en un momento dado.

• No existe protección de memoria.

El modelo de ejecución es basado en eventos:

• Soporta altos niveles de concurrencia en poca cantidad de espacio.
• Un único contexto de ejecución se comparte entre tareas de procesamiento no relacionadas (se evita la sobrecarga de los cambios de contexto).
• Concurrencia, gracias a tareas o procesos de larga ejecución que ejecutan hasta completarse en background sin interferir con otros eventos del sistema. Pueden ser interrumpidos por eventos del sistema de bajo nivel (concepto de concurrencia).
• Proporciona mecanismos para crear exclusión mutua en secciones de código (concepto de atomicidad)

El Modelo de programación está basado en componentes (módulos):

• Cada módulo es diseñado para operar continuamente respondiendo a eventos de entrada (alarmas, timer del reloj, radio, etc.).
• Cuando un evento llega, trae con él el contexto de ejecución requerido.
• Las aplicaciones deben declarar implícitamente cuando han finalizado de usar la CPU.

A continuación se muestran las funciones que puede ejecutar Tinyos:

- Enrutamiento
- Conversor analógico-digital
- Identificación de un nodo
- Reserva de memoria
- Conversión serie-paralelo
- Pila de comunicación
- Radio
- Active Messages
- Logs del sistema
- Reloj del sistema
- Energía del sistema
- Resetear el sistema

- Generador de números aleatorios
- Leds del sistema
- Datos (Int a Leds, a Radio y viceversa)
- Gestión de errores CRC

3.1.- NesC

NesC es una variante del lenguaje de programación C, optimizado para las limitaciones de memoria de las redes de sensores. Existen varias herramientas que completan y facilitan su uso, escritas en su mayoría en Java y en Bash. Otras herramientas y librerías asociadas están escritas principalmente en C.

NesC es un lenguaje orientado a componentes y está especialmente diseñado para programar aplicaciones sobre redes de sensores, en particular en el sistema operativo TinyOS.

Un programa en NesC esta estructurado mediante componentes, el usuario crea su propio componente ayudado a su vez de otros componentes ya creados. Dos componentes podrán comunicarse entre mediante una interfaz, la cual definifirá una serie de métodos (commands y events) los cuales deberán ser implementados en cada componente. Así, un método podrá solicitar la ejecución de un command de otro componente; por otro lado, para enviar una notificación se utilizarán un event.

Todo componente esta dividido lógicamente en tres partes: Configuración, Interfaces y Módulos.

Como se ha dicho anteriormente, una aplicación escrita para tinyOS se compone de implementación de módulos y de configuraciones. En algunos casos, pueden hacerse aplicaciones solo con archivos de configuraciones. Un archivo de configuración consiste básicamente en un archivo formado por los módulos (o componentes) a utilizar en la aplicación y la relación que hay entre dichos módulos y las interfaces utilizadas por otros módulos.

Las interfaces se pueden definir como la parte "visible" de los módulos. No se puede acceder a ninguna función de un módulo si antes no ha sido definida en una interfaz. Una interfaz puede ser provista por varios módulos distintos. Por ejemplo, podemos tener una interfaz de comunicaciones que abstraiga los procedimientos de envío y recepción de mensajes y varios módulos que hagan uso de esa interfaz, uno para cada tipo de hardware subyacente (por ejemplo no se maneja igual el hardware de comunicaciones de una mica2 que el de una telos). Los métodos serán los mismos y se llamarán igual (la interfaz es la misma), pero la implementación interna puede ser totalmente distinta. Nosotros lo veremos como una caja negra.

Los módulos implementan bloques funcionales generalmente de un nivel de abstracción más bajo que el bloque que los referencia. En los módulos es donde se programarán las acciones que llevara acabo nuestro componente. Un

mismo módulo puede proveer varias interfaces y por lo tanto puede implementar distintas funcionalidades (Por ejemplo el módulo *AMSender* provee las interfaces *Packet*, *AMPacket*, *AMSend*, etc...). Lo lógico es que todos los conjuntos de funciones (interfaces) que implementa un módulo guarden algún tipo de relación. Los módulos se dividen en tres partes: Provides, son las interfaces que nuestro componente ofrece, Uses, las interfaces que usa nuestro componente e Implementation, donde verdaderamente se realizan las acciones que queremos que realice nuestro programa.

Capítulo 3 - Desarrollo de una implementación basada en 802.15.4

En éste capitulo se va a explicar como se ha realizado el desarrollo y puesta en marcha de una red de sensores (WSN) con tecnología 802.15.4 y módulos MicaZ/Imote2. Además se va a comentar también como se han realizado las herramientas de análisis de características de dichas redes así como los resultados obtenidos a través de dichas herramientas.

1.- Puesta en marcha a partir de MoteView

Este apartado explica como realizar la puesta en marcha de una red 802.15.4 empleando para ello MoteView. Como se explica en el apartado 2.2.2. del capítulo 2, MoteView es un conjunto de herramientas que permiten controlar y analizar fácilmente una red de sensores inalámbricos. Para ello primero hay que programar los dispositivos a través de la herramienta MoteConfig.

1.1.- Programación a través de MoteConfig

Como se dijo en el apartado 2.2.4. del capítulo 2, MoteConfig es un interfaz gráfico que proporciona Crossbow para poder programar fácilmente código precompilado XMesh/TinyOS en dispositivos inalámbricos 802.15.4. MoteConfig tiene el siguiente aspecto gráfico.

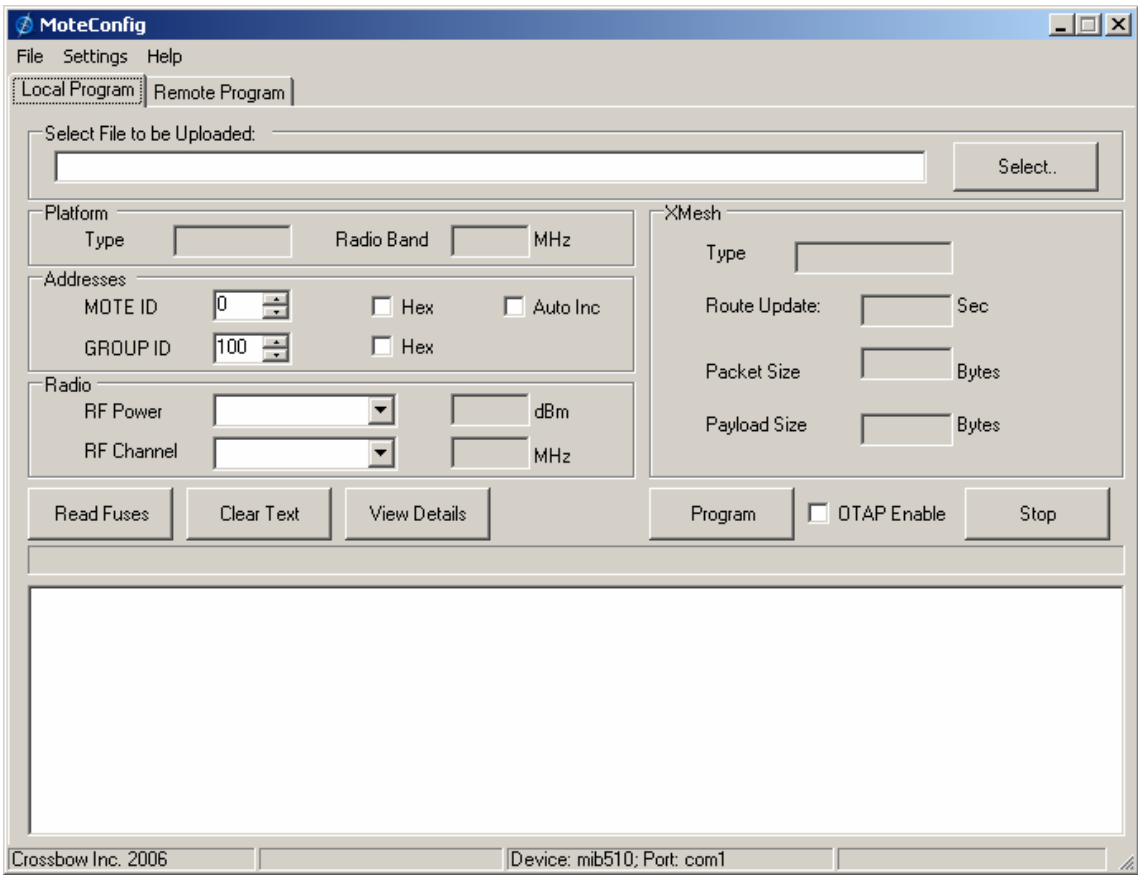

Fig 11 - Pantalla principal de MoteConfig

A lo largo de este proyecto, la programación de los dispositivos se va a realizar siempre a través del dispositivo MIB600, al cual se accede a través de una conexión Ethernet.

Lo primero que hay que hacer para programar es indicar en Settings el tipo de dispositivo empleado para la programación (opción Interface Board Settings). Se selecciona la opción MIB600 e indicamos la dirección IP del dispositivo. Por defecto tiene la dirección 10.1.1.12, aunque se puede modificar a través de Telnet. La computadora desde la que se realiza la programación debe estar configurada para pertenecer a la misma red que el dispositivo.

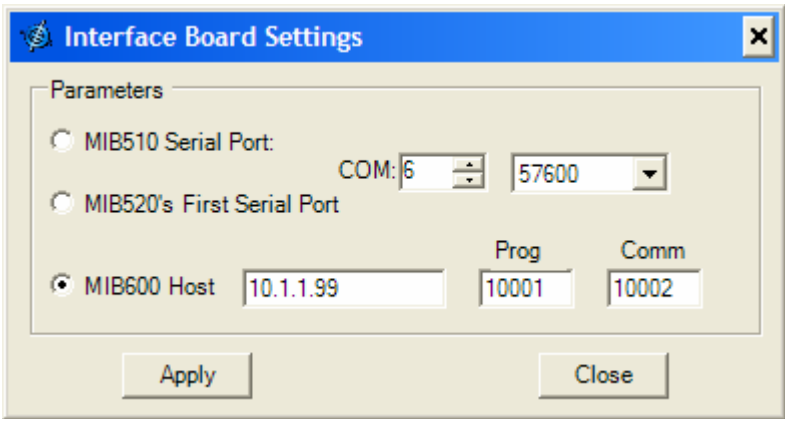

Fig. 12 – Pantalla de configuración de Interface Board Settings

Una vez configurado, se pulsa Apply y se puede proceder a la programación de los sensores inalámbricos. En esta parte de la redacción solo se va a explicar como programar los sensores con el código precompilado que incluye MoteView en sus librerías para poder empezar a utilizar los dispositivos inalámbricos y los diferentes sensores.

Si se presiona sobre el botón Select, aparece una ventana en donde se puede seleccionar el archivo que se desea programar. Dependiendo de la función del nodo y del tipo de sensor, se opta por un determinado código precompilado u otro.

- Si el nodo va a encargarse de capturar todo el tráfico generado por los sensores y después va a enviarlo a un ordenador, se selecciona el fichero XMeshBase con el perfil HP, pues se supone que este nodo va a estar conectado a la red eléctrica. Dicho fichero se encuentra en la carpeta *C:\Archivos de programa\Crossbow\MoteView\xmesh\micaz\XMeshBase*
- Si el nodo se encarga de enviar trafico a la red con la información de los sensores, se emplean los ficheros contenidos en las carpetas de *C:\Archivos de programa\Crossbow\MoteView\xmesh\micaz* dependiendo del tipo de sensor (MDA100, MDA300, MDA320,MTS300,…) que se utilice y del tipo de alimentación del sensor (LP si está alimentado por baterías y HP si está conectado a la red eléctrica).

Una vez elegido el código, se selecciona el canal de trabajo, la potencia, el identificador de grupo y el identificador de nodo y se pulsa el botón de Program. Si algo va mal durante la programación, aparece un mensaje de error debajo de la barra de progreso. Para que todo vaya bien, el nodo programado con XMeshBase debe tener el identificador de nodo 0.

Este código precompilado sirve para empezar a trabajar con los sensores proporcionados por Crossbow. A continuación se explica como analizar una red 802.15.4 con código XMesh/TinyOS precompilado con la herramienta MoteView.

1.2.- Análisis de una red utilizando MoteView 2.0

Gracias a la herramienta MoteView 2.0 se puede analizar la red de sensores y recolectar la información obtenida a través de dichos sensores. El aspecto gráfico de esta herramienta es el siguiente.

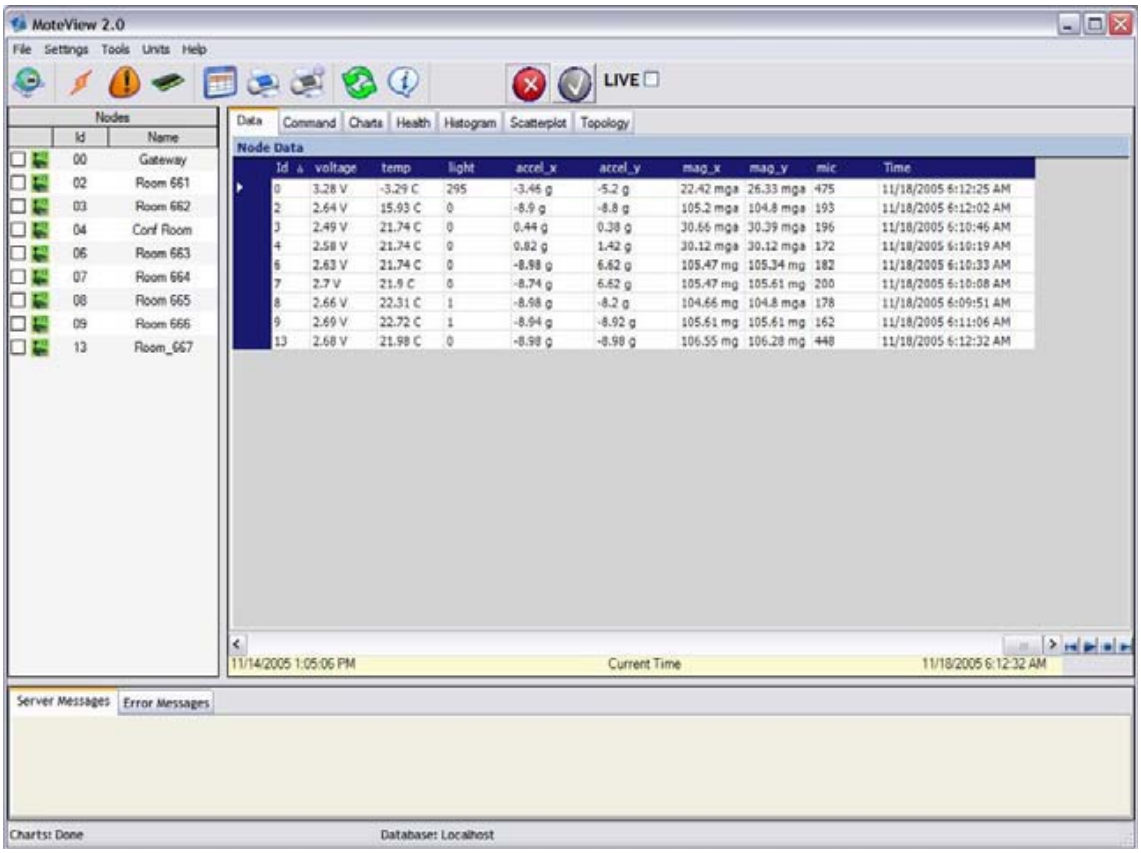

Fig. 13 – Entorno gráfico de MoteView 2.0

Para poder recibir la información de los sensores, se debe presionar sobre File y seleccionar la opción *Connect to WSN*. Como la recolección de datos de los sensores se va a hacer a través del nodo conectado a MIB600, se selecciona la opción de MIB600, se indica la dirección IP y el puerto por el que se van a recibir los datos (dirección IP 10.1.1.12 y puerto 10002), se pulsa *Next* y se

selecciona el tipo de sensor que se esta utilizando en los motes. Cuando se pulsa *Done*, el sensor empieza a recolectar datos de los nodos.

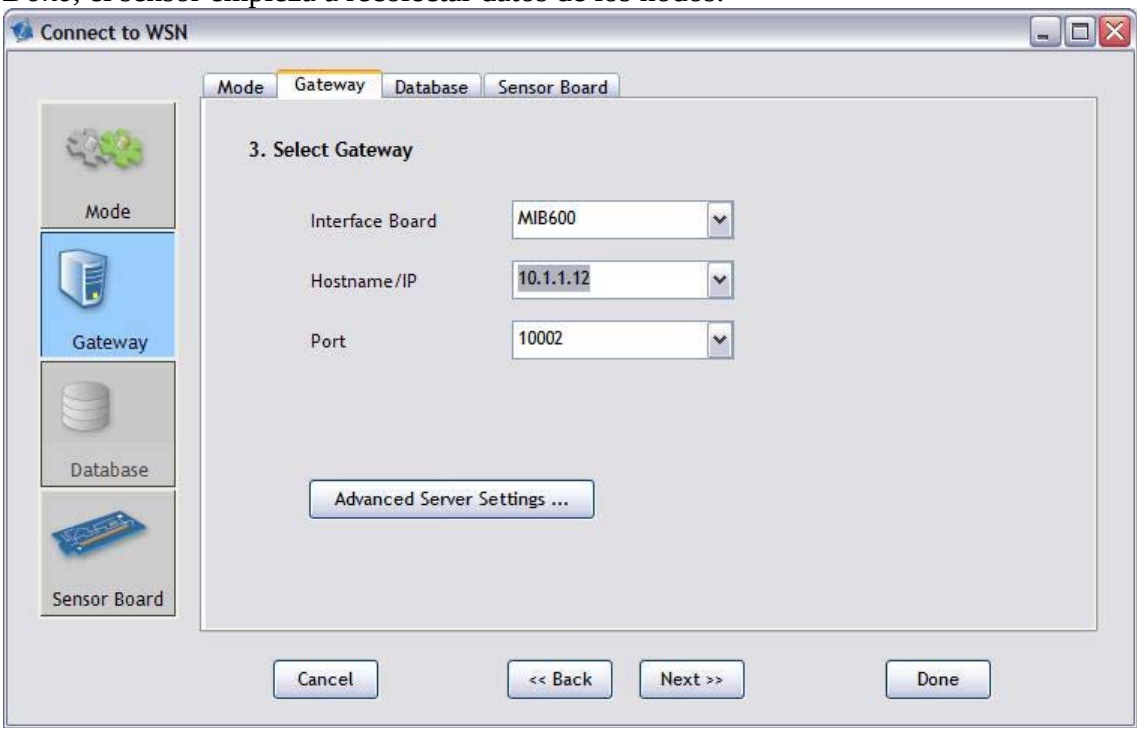

Fig 14 – Configuración de la conexión

Fig 15 – Selección del tipo de sensor

A través del nodo principal, el programa empieza a descubrir el resto de nodos. Se puede acceder a la información de dichos nodos a través de la pestaña de *Data*. En dicha pestaña aparece la información aportada por el sensor de cada nodo.

Para testar el correcto funcionamiento de los nodos a través de los LEDs y para modificar la tasa de transferencia de los datos o el identificador de nodo, se emplea la pestaña de *Command*.

La pestaña *Chart*, permite realizar gráficas de los datos capturados por los sensores.

Gracias a XMesh, los nodos recopilan información sobre paquetes perdidos, enviados, reenviados, rechazados, nivel de batería, datos de QoS y se la envian al nodo principal. En la pestaña de *Health* se muestra toda esta información.

En *Histogram*, se muestran histogramas con los datos recopilados por cada nodo.

Mediante *Topology* se puede ver como están interconectados todos los nodos, mostrando como algunos nodos enlutan paquetes de otros para conectarse con el servidor.

En este primer apartado se ha resumido brevemente como se realiza la puesta en marcha a partir de MoteView. A continuación se va a argumentar el motivo por el que se descartó el uso de esta herramienta y MoteWorks para el desarrollo de este proyecto fin de carrera.

1.3.- Motivaciones para rechazar el uso de MoteWorks y MoteView

Como se comentó en el apartado 2.2.3. del capitulo 2, MoteWorks es un conjunto de herramientas que permiten diseñar aplicaciones que trabajan sobre 802.15.4 y gracias a MoteView y MoteConfig, sería muy fácil programarlas, cargarlas en los motes y analizar la red.

De la interconectividad entre motes y el envío de información de Health y de los sensores, se encarga un conjunto de librerías, donde la mayor parte de las funciones están precompiladas y que permiten realizar toda esa funcionalidad.

El problema de programar con MoteWorks, es que solo sirve para crear aplicaciones por encima de la capa MAC de 802.15.4. Esto se debe a que dicha capa está precompilada. Evidentemente toda la funcionalidad de XMesh sería muy útil a la hora de realizar pruebas sobre 802.15.4, pero parámetros como la tasa de transferencia o el método de enrutamiento, se ven limitados a causa de que el código venga precompilado.

Otro problema es que las aplicaciones aportadas por Crossbow no utilizan una implementación completa de 802.15.4, siendo el motivo de este proyecto analizar redes 802.15.4. Crossbow no incluye las funcionalidades de GTS y de CSMA/CA ranurado. Incorporar estas funcionalidades es imposible ya que la capa MAC y la capa PHY están precompiladas también, y solo se puede acceder a algunas funciones de éstas.

Crossbow no implementa 802.15.4 completo porque el radiotransmisor no soporta dicha implementación, por lo que se opta por utilizar primitivas B-MAC (desarrollo MAC de la Universidad de Berkeley) y paquetes MAC compatibles.

Como el principal motivo de este proyecto es el análisis de redes 802.15.4 completas y poner al límite sus características como la tasa de transferencia o el retardo, se rechaza la posibilidad de avanzar empleando MoteWorks y MoteView, pese a que facilitarían mucho el análisis de la red. En el siguiente apartado se habla de la alternativa para poder desarrollar la idea inicial del proyecto fin de carrera.

2.- Open-ZigBee

Una vez visto que las herramientas proporcionadas por Crossbow no son válidas para el desarrollo del proyecto, es cuando se busca alguna alternativa. Dicha alternativa es Open-Zigbee.

Open-ZigBee ofrece de forma gratuita, el código fuente para el desarrollo de aplicaciones basadas en IEEE 802.15.4/ZigBee. Dichas herramientas están siendo desarrolladas en colaboración con IPP-Hurray y Art-Wise FrameWork. Este código esta implementado en nesC y permite poner al límite los distintos parámetros de los dispositivos MicaZ, como pueden ser la velocidad de transmisión o la potencia radiada, ya que no emplea código precompilado, si no que utiliza código abierto. En el próximo apartado se analizarán las principales funcionalidades implementadas por Open-ZigBee.

2.1.- Estudio de las funcionalidades de Open-ZigBee

En este apartado se van a enumerar las funcionalidades desarrolladas por Open-ZigBee en la versión actual de la implementación.

· Algoritmo CSMA/CA slotted
· Mecanismo GTS
· Mecanismo de transmisión indirecta
· Transmisión directa e indirecta de datos GTS
· Gestión del mecanismo de Beacons
· Construcción de la trama – Solo campos con direccionamiento corto y direccionamiento extendido en la peticiones de asociación
· Mecanismo de asociación y desasociación
· Gestión de MAC PIB
· Condiciones de recepción de trama
· Detección de energía y escaneo del canal pasivo

A continuación se explica como esta organizada la implementación en nesC de Open-ZigBee y cual es la utilidad de cada fichero de la implementación.

2.2.- Organización de la implementación

En este apartado se va a comentar la estructura de la implementación de Open-ZigBee, la cual se muestra en la figura 16.

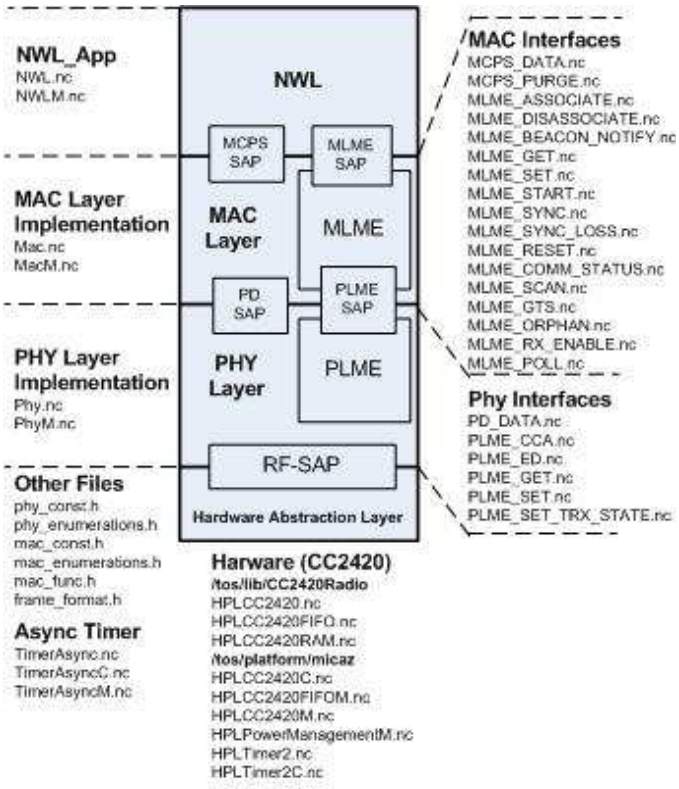

Fig 16 – Organización de la implementación de Open-ZigBee

Todos los archivos para la biblioteca deben estar en la carpeta de TinyOS contrib/hurray/tos. En Interfaces tenemos interfaces genéricas en la cual solo existe TimerAsync.nc que corresponde a la interfaz encargada de la sincronización del tiempo. En Interfaces/ieee802154/mac están las interfaces de conexión para MAC. Dichas interfaces son las siguientes:

-*MCPS_DATA.nc* – MAC parte común de la subcapa dato-servicio del punto de acceso.

-*MCPS_PURGE-nc* –MAC parte común de la subcapa de inicio de punto de acceso.

-*MLME_ASSOCIATE.nc* – Administrador de capa MAC asociado al ingreso del punto de acceso.

-*MLME_BEACON_NOTIFY.nc* – Administrador de capa MAC para las señales de notificación del servicio de punto de acceso.

-*MLME_COMM_STATUS.nc* – Entidad de la capa MAC encargada de administrar la comunicación de estado del servicio del punto de acceso.

-*MLME_DISASSOCIATE.nc* – Entidad de la capa MAC encargada de desasociar servicio de punto de acceso.

-*MLME_GET.nc* - Entidad de la capa MAC encargada de obtener servicio de punto de acceso.

-*MLME_GTS.nc* - Entidad de la capa MAC encargada de garantizar "Time Slot Service" de punto de acceso.

-*MLME_POLL.nc* - Entidad de la capa MAC encargada del servicio de encuesta en punto de acceso.

-*MLME_RESET.nc* – Administrador de capa MAC encargado del servicio de "RESET" con el Acces Point.

-*MLME_SCAN-nc* - Administrador de capa MAC encargado del servisio de "SCAN" con el Acces Point.

-*MLME_SET-nc* - Administrador de capa MAC encargado del servisio de "Set" con el Acces Point.

-*MLME_START-nc* - Administrador de capa MAC encargado del servisio de "Start" con el Acces Point.

-*MLME_SYNC-nc* - Administrador de capa MAC encargado del servicio de sincronización con el Acces Point.

-*MLME_SYNC_LOSS-nc* - Administrador de capa MAC encargado del servicio de sincronismo perdida con el Acces Point.

En interfaces/ieee802154/phy están las interfaces de conexión entre las capas MAC y física.Dichas interfaces son las siguientes:

-*PD_DATA.nc* – PHY "DATA-Service" punto de acceso.

-*PLME_CCA.nc* - administración de la capa física – servicio limpiador de canales punto de acceso.

-*PLME_ED.nc* - administración en capa física de detección de energía.

-*PLME_GET.nc* - administración en capa física de "GET-SERVICE".

-*PLME_SET.nc* - administración en capa física de "SET-SERVICE".

-*PMLE_SET_TRX_STATE.nc* - administración en capa física de colocar valores de inicio a transiver "State-Service" en el acceso.

En lib/mac se encuentran los archivos de la implementación de la capa MAC:

-*MAC.nc* - Los archivos de configuración del modulo MacM.

-*MACM.nc* - Implementación de la capa MAC.

-*mac_const.h* - Constantes de la capa MAC.

-*mac_enumerations.h*- Enumeraciones de la capa MAC.

En lib/phy se encuentran los archivos de implementación de la capa física:

-*Phy.nc* - Configuración del módulo de implementación de PhyM.nc.

-*PhyM.nc* - Implementación de la capa física.

-*phy_const* - Constantes de la capa física.

-*phy_enumerations.h* – enumeraciones de la capa física.

Finalmente en tos/system módulos generales del sistema:

-*TimerAsysnC.nc* - Implementación de la configuración del módulo TimerAsyncM.

-*TimerAsyncM.nc* - Implementación de tiempo asincrónico.

-*mac.func.h* - Principales funciones genéricas usadas por la implementación de la capa MAC.

-*frame_format.h* - Definición del formato del frame.

En el siguiente apartado se comentan las funcionalidades implementadas en cada capa y las relaciones existentes.

2.3 Implementación de las capas Física y MAC

A continuación se va a comentar como que se ha implementado cada una de las capas de Open-ZigBee.

2.3.1.- Implementación en la capa Fisica

La capa física IEEE 802.15.4 es responsable de implementar las siguientes funcionalidades:

-Activación y la desactivación del radiotransmisor
-La detección de energía con el canal presente.
-LQI (link quality indicator) para los paquetes recibidos.
-CCA (Clear channel Assement) para CSMA/CA.
-Selección de frecuencia del canal.
-Transmisión y recepción de datos.

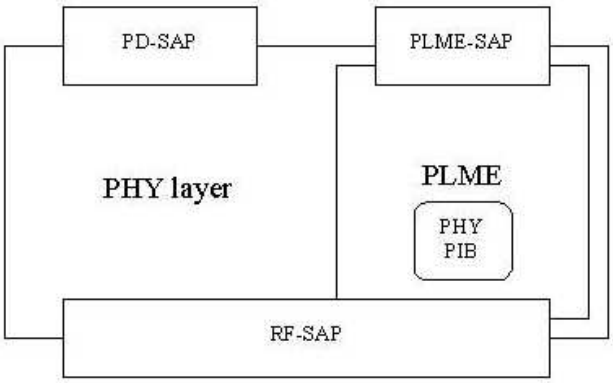

Fig 17 - Modelo de referencia de la capa física

RF-SAP es la interfaz con la capa física por medio del firmware de CC2420, provisto por TinyOS.

El PD-SAP es la interfaz que intercambia datos entre la capa MAC y la física.

PLME-SAP es la interfaz encargada para intercambiar información de administración entre las capas Mac y física.

2.3.2.- Implementación en la capa MAC

Las funciones que se implementan en la capa MAC de la IEEE 802.15.4 son las siguientes:

-Generación señales de red si el dispositivo es coordinador.
-Sincronización de las señales.
-Soporte de PAN, asociación y desasociación.
-Soporte de seguridad del dispositivo.
-Uso del mecanismo CSMA-CA para el acceso al canal.

-Manejo y administración de mecanismo GTS.
-Resolución de conflicto entre dos entidades MAC iguales.

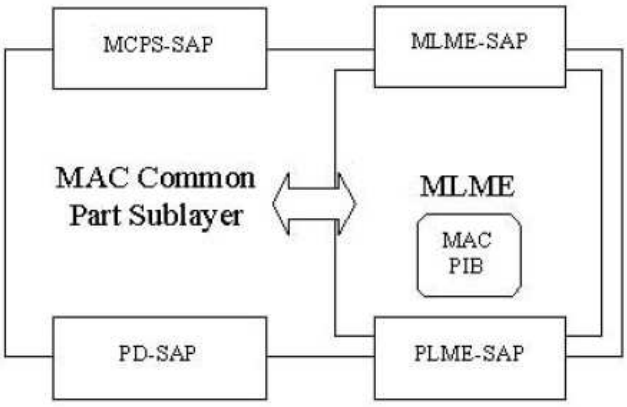

Fig 18. Modelo de referencia de la capa MAC.

El bloque MCPS-SAP se encarga de la MSDU (service data unit) para transferencia de señales entre la capa MAC y la capa superior.

El bloque MLME-SAP está encargado del intercambio de comandos de administración entre la capa MAC y la capa superior.

PD-SAP y PMLE-SAP son usadas para conectar la capa MAC con las funciones provistas por la capa física.

El bloque MAC PIB (MAC PAN Information Base) es administrado en la capa MAC y es una base de dato de la administración. Este bloque almacena la entre otras cosas la siguiente información:

- Duración de espera de "Acknowledgment" (comprobación del dato recibido).
- Permisos asociados.
- Respuesta automática de Datos.
- Opciones de extensión de vida de batería.
- Periodos de extensión de vida de batería.
- Carga de señal útil.
- Largo de carga de señal útil.
- Orden de señales.
- Tiempos de transmisión de señales.
- Número de secuencia de las señales.
- Coordinador de direcciones largas.
- Coordinador de direcciones pequeñas.
- Número de secuencia del dato.
- Opciones de permiso de e GTS.
- Máximos intentos para CSMA.
- Identificador PAN.
- Direcciones cortas.
- Orden de "Superframe" (super trama).
- Tiempo de transacción persistente.

Una vez visto como funciona Open-ZigBee, se van a explicar los conflictos encontrados inicialmente con dicha implementación para poder conseguir el objetivo final que es el poder realizar un estudio estadístico de las redes 802.15.4.

2.4. Open-ZigBee 1.15: Estudio de la implementación y problemática

Open-ZigBee 1.15 es una implementación de Open-ZigBee diseñada para trabajar sobre TinyOS 1.1.15. Para trabajar sobre esta implementación se hace uso de Cygwin configurado para trabajar con TinyOS. Cuando se comenzó el desarrollo del proyecto fin de carrera, esta era la implementación más reciente, aunque unos meses después salió Open-ZigBee 2.0 que trabaja sobre TinyOS 2.0.

El primer paso es conseguir la comunicación entre 2 dispositivos MicaZ. El campo *datos* se deja vacío aunque ocupa 32 bytes. El escenario inicial se muestra en la siguiente figura.

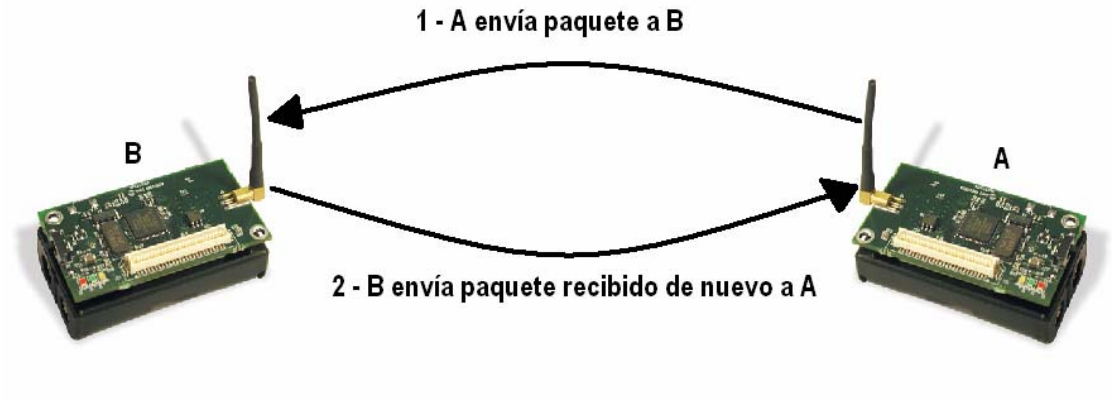

Fig 19 – Escenario para comunicación entre motes

Gracias a la iluminación de determinados Leds del dispositivo MicaZ, se determina si los motes están comunicándose entre si.

Este primer escenario funciona según lo esperado. Los motes se comunican sin problema y el sistema de Leds para validar la comunicación es un éxito. El siguiente paso es conseguir que uno de los dispositivos MicaZ se comunique con un PC, para que de esta forma se puedan calcular estadísticas. En concreto, estadísticas de retardo entre paquetes. Dicha comunicación se realiza empleando el dispositivo MIB600. Dicho escenario muestra en la siguiente figura.

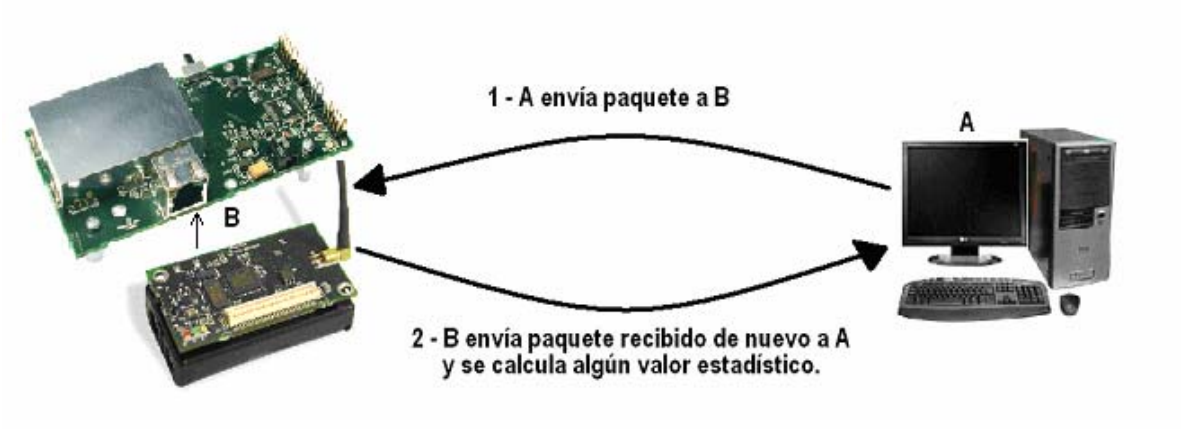

Fig 20 – Escenario para comunicación serie con PC.

En este escenario es donde encontramos la problemática de emplear TinyOS 1.15. En él, no se pueden enviar paquetes al PC de una forma sencilla, y además no hay demasiadas aplicaciones de interconexión con un PC en las que poder basarse. En cierto momento esto puede provocar el desanimo en el desarrollo de este proyecto, pero solo un mes después de aparecer este problema, surge Open-ZigBee 2.0, que es la solución que permite conseguir la finalidad de este proyecto fin de carrera.

2.5 Open-ZigBee 2.0

Esta es la solución final utilizada en este proyecto fin de carrera. Esta implementación, muy similar a la versión 1.15, pero trabajando sobre TinyOS 2.0, y permite el envío de datos a través de una conexión serie entre un dispositivo MicaZ y un PC. Su estudio va a ser mucho mas detallado y en dicha solución se implementarán todos los escenarios necesarios para la toma de estadísticas.

Para el desarrollo de esta solución, es necesario instalar en un PC el sistema operativo XubunTOS. Dicho sistema operativo no es más que una versión optimizada de Xubuntu para el desarrollo de aplicaciones basadas en TinyOS. Xubuntu es una distribución oficial basada en la distribución Linux Ubuntu, que utiliza el entorno de escritorio Xfce. Xubuntu está diseñado para usuarios con computadores que poseen recursos limitados de sistema, o para usuarios que buscan un entorno de escritorio altamente eficiente.

Todos los ficheros utilizados en el desarrollo de este Proyecto Fin de Carrera emplean como raíz el siguiente directorio:

```
\opt\tinyos-2.x-contrib\hurray2x\apps\
```

2.5.1. – Desarrollo de la comunicación entre Motes

Como se hace en el apartado 2.4, en primer lugar se va a explicar como se ha desarrollado la comunicación entre 2 dispositivos MicaZ. En este apartado, a diferencia del apartado 2.4, se va explicar también como se ha implementado el escenario. El sistema a desarrollar es el representado en la siguiente figura.

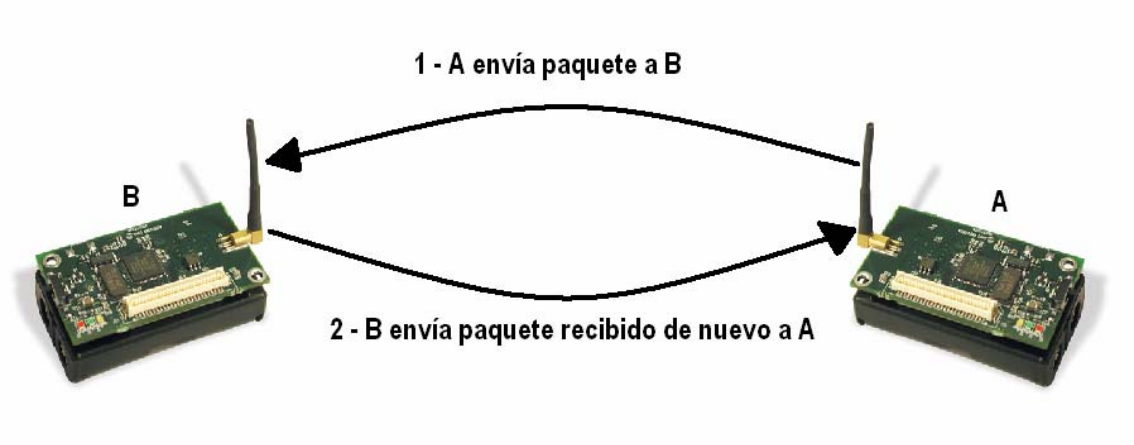

Fig 21 – Primer sistema implementado

Cuando los dispositivos envían o reciben paquetes, se encienden o apagan distintos Leds para saber que la comunicación ha sido exitosa.

En primer lugar hay que diferenciar cual va a ser el extremo que va a encargarse de la gestión de la comunicación, es decir, que extremo va a ser el PANCoordinator. Esta tarea se decide asignar al extremo B la función de PANCoordinator. El envío de paquetes entre Motes queda reflejado en la siguiente figura:

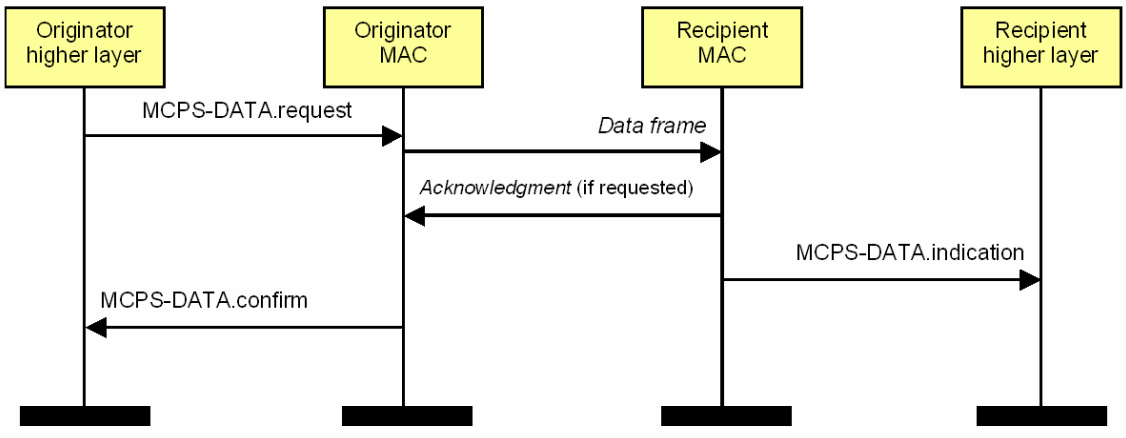

Fig. 22 – Envío de paquetes

2.5.1.1.- Implementación de PANCoordinator.

PANCoordinator se encarga de devolver los paquetes recibidos desde el extremo A de nuevo a dicho extremo. Los ficheros correspondientes a esta implementación se encuentran en el directorio `SimpleRoutingExample\PANCoordinator` del directorio raíz.

La implementación se desarrolla en el fichero SimpleRoutingExampleM.nc. Primero se cargan los interfaces necesarios para el uso de 802.15.4 y el control del dispositivo MicaZ

```
module SimpleRoutingExampleM {

//MICA CONTROL
uses interface Boot;
uses interface Leds;
uses interface Timer<TMilli> as Timer0;
uses interface Timer<TMilli> as Timer_Send;


//MAC interfaces
uses interface MLME_START;

uses interface MLME_GET;
uses interface MLME_SET;

uses interface MLME_BEACON_NOTIFY;
uses interface MLME_GTS;

uses interface MLME_ASSOCIATE;
uses interface MLME_DISASSOCIATE;

uses interface MLME_ORPHAN;

uses interface MLME_SYNC;
uses interface MLME_SYNC_LOSS;

uses interface MLME_RESET;

uses interface MLME_SCAN;

uses interface MCPS_DATA;

}
```

La segunda fase es la definición e inicialización de variables que van a ser utilizadas a lo largo de esta implementación.

```
implementation {

PANDescriptor pan_des;
```

```
uint32_t my_short_address=0x00000000;

uint32_t DestinationMote[2];
uint32_t SourceMoteAddr[2];



  event void Boot.booted() {

DestinationMote[0]=0x00000000;
DestinationMote[1]=0x00000002;

my_short_address = 0x0000;

//Genera el evento que va a iniciar al
PANCoordinator.
call Timer0.startOneShot(4000);


}
```

Ahora el PANCoordinator se establece como tal y comienza el proceso de envío de beacons.

```
event void Timer0.fired() {

uint8_t v_temp[2];


      //establece la variable de direccion corta
      v_temp[0] = (uint8_t)(my_short_address >> 8);
      v_temp[1] = (uint8_t)(my_short_address );

      call MLME_SET.request(MACSHORTADDRESS,v_temp);

      //establece la variable MAC PANID
      v_temp[0] = (uint8_t)(MAC_PANID >> 8);
      v_temp[1] = (uint8_t)(MAC_PANID );

      call MLME_SET.request(MACPANID,v_temp);

      //Comienza el envio de beacons
      call MLME_START.request(MAC_PANID,
LOGICAL_CHANNEL, BEACON_ORDER,
SUPERFRAME_ORDER,1,0,0,0,0);
      //Se modifica el LED0 para indicar que esta
      listo para recibir mensajes.
      call Leds.led0Toggle();

  }
```

Una vez echo esto, el dispositivo coordinador se queda a la espera de recibir paquetes y generar respuestas. Dicha recepción y reenvío se realiza de la siguiente forma:

```
//Esta función se le llama cuando se recibe un
paquete
```

```
event error_t MCPS_DATA.indication(uint16_t
SrcAddrMode, uint16_t SrcPANId, uint32_t SrcAddr[2],
uint16_t DstAddrMode, uint16_t DestPANId, uint32_t
DstAddr[2], uint16_t msduLength,uint8_t
msdu[100],uint16_t mpduLinkQuality, uint16_t
SecurityUse, uint16_t ACLEntry)
{

            //Se modifica el LED2 al recibir
            call Leds.led2Toggle();


            //Se envia a la dirección deseada. En
este escenario esta preestablecida
            DestinationMote[0]=0x00000000;
            DestinationMote[1]=0x00000002;



            //set_txoptions(ack, gts,
indirect_transmission, security)
            call MCPS_DATA.request(SHORT_ADDRESS,
MAC_PANID, SrcAddr, SHORT_ADDRESS, MAC_PANID,
DestinationMote, 1, msdu,1,set_txoptions(1,0,0,0));

            //Se modifica el LED1 al enviar
            call Leds.led1Toggle();




return SUCCESS;
}
```

Para el correcto funcionamiento del sistema, también deben definirse distintos eventos relacionados con 802.15.4. Dichos eventos no afectan al funcionamiento del escenario ya que son eventos vacíos, pero deben estar definidos ya que estan definidos en el interfaz de la capa MAC.

```
/****MLME-SCAN*****/
event error_t MLME_SCAN.confirm(uint8_t
status,uint8_t ScanType, uint32_t UnscannedChannels,
uint8_t ResultListSize, uint8_t EnergyDetectList[],
SCAN_PANDescriptor PANDescriptorList[])
{

return SUCCESS;
}

/****MLME-ORPHAN****/
event error_t MLME_ORPHAN.indication(uint32_t
OrphanAddress[1], uint8_t SecurityUse, uint8_t
ACLEntry)
{
```

```
return SUCCESS;
}

/****MLME-RESET****/
event error_t MLME_RESET.confirm(uint8_t status)
{
return SUCCESS;
}

/****MLME-SYNC-LOSS****/
event error_t MLME_SYNC_LOSS.indication(uint8_t
LossReason)
{

return SUCCESS;
}

/****MLME-GTS****/
event error_t MLME_GTS.confirm(uint8_t
GTSCharacteristics, uint8_t status)
{

return SUCCESS;
}

event error_t MLME_GTS.indication(uint16_t
DevAddress, uint8_t GTSCharacteristics, bool
SecurityUse, uint8_t ACLEntry)
{
return SUCCESS;
}
/****MLME-BEACON NOTIFY****/
event error_t MLME_BEACON_NOTIFY.indication(uint8_t
BSN,PANDescriptor pan_descriptor, uint8_t
PenAddrSpec, uint8_t AddrList, uint8_t sduLength,
uint8_t sdu[])
{

return SUCCESS;
}
/****MLME-START****/
event error_t MLME_START.confirm(uint8_t status)
{


return SUCCESS;
}

/****MLME-SET****/
event error_t MLME_SET.confirm(uint8_t status,uint8_t
PIBAttribute)
{


return SUCCESS;
}
/****MLME-GET****/
event error_t MLME_GET.confirm(uint8_t status,uint8_t
PIBAttribute, uint8_t PIBAttributeValue[])
{
```

```
return SUCCESS;
}


/****MLME-ASSOCIATE****/
event error_t MLME_ASSOCIATE.indication(uint32_t
DeviceAddress[], uint8_t CapabilityInformation, bool
SecurityUse, uint8_t ACLEntry)
{

return SUCCESS;
}

event error_t MLME_ASSOCIATE.confirm(uint16_t
AssocShortAddress, uint8_t status)
{

return SUCCESS;
}
/****MLME-DISASSOCIATE****/
event error_t MLME_DISASSOCIATE.indication(uint32_t
DeviceAddress[], uint8_t DisassociateReason, bool
SecurityUse, uint8_t ACLEntry)
{
return SUCCESS;
}

event error_t MLME_DISASSOCIATE.confirm(uint8_t
status)
{
return SUCCESS;
}


/****MCPS-DATA****/
event error_t MCPS_DATA.confirm(uint8_t msduHandle,
uint8_t status)
{

return SUCCESS;
      }
```

Una vez definido todo esto, se define un fichero de configuración (SimpleRoutingExample.nc) donde se indican todas las relaciones 802.15.4 y la implementación realizada:

```
#include <Timer.h>

#include "simpleroutingexample.h"
#include "phy_const.h"
#include "phy_enumerations.h"
#include "mac_const.h"
#include "mac_enumerations.h"
#include "mac_func.h"

 configuration SimpleRoutingExample {
}

implementation
```

```
{
  components MainC;
  components LedsC;
  components SimpleRoutingExampleM;

  SimpleRoutingExampleM.Boot -> MainC;

  components Mac;

  SimpleRoutingExampleM.Leds -> LedsC;

  components new TimerMilliC() as Timer0;
  SimpleRoutingExampleM.Timer0 -> Timer0;

  components new TimerMilliC() as Timer_Send;
  SimpleRoutingExampleM.Timer_Send ->Timer_Send;


  //MAC interfaces

  SimpleRoutingExampleM.MLME_START -> Mac.MLME_START;

  SimpleRoutingExampleM.MLME_GET ->Mac.MLME_GET;
  SimpleRoutingExampleM.MLME_SET ->Mac.MLME_SET;

  SimpleRoutingExampleM.MLME_BEACON_NOTIFY -
>Mac.MLME_BEACON_NOTIFY;
  SimpleRoutingExampleM.MLME_GTS -> Mac.MLME_GTS;

  SimpleRoutingExampleM.MLME_ASSOCIATE-
>Mac.MLME_ASSOCIATE;
  SimpleRoutingExampleM.MLME_DISASSOCIATE-
>Mac.MLME_DISASSOCIATE;

  SimpleRoutingExampleM.MLME_ORPHAN->Mac.MLME_ORPHAN;
  SimpleRoutingExampleM.MLME_SYNC->Mac.MLME_SYNC;
  SimpleRoutingExampleM.MLME_SYNC_LOSS-
>Mac.MLME_SYNC_LOSS;
  SimpleRoutingExampleM.MLME_RESET->Mac.MLME_RESET;

  SimpleRoutingExampleM.MLME_SCAN->Mac.MLME_SCAN;


  SimpleRoutingExampleM.MCPS_DATA->Mac.MCPS_DATA;


}
```

El fichero simpleroutingexample.h contiene información sobre la trama MAC, de los beacons y el identificador del PANCoordinator. Este fichero es común para los dos extremos.

```
#define BEACON_ORDER 6
#define SUPERFRAME_ORDER 4
#define LOGICAL_CHANNEL 0x15
#define MAC_PANID 0x1234
```

Una vez implementado todo esto, se compila el código y se carga en el dispositivo MicaZ empleando la siguiente sentencia en la consola de XubunTOS:

```
make micaz install.0 eprb,10.1.1.12
```

2.5.1.2.-Desarrollo del extremo EndDevice

A continuación se va a explicar como se desarrolla el extremo que se encarga de conectarse al PANCoodinator, enviarle paquetes y esperar su respuesta. Los ficheros correspondientes a esta implementación se encuentran en el directorio `SimpleRoutingExample\EndDevice` del directorio raíz.

La implementación se desarrolla en el fichero SimpleRoutingExampleM.nc. Igual que en el apartado anterior, primero se cargan los interfaces necesarios para el uso de 802.15.4 y el control del dispositivo MicaZ y se realiza la definición e inicialización de variables que van a ser utilizadas a lo largo de esta implementación

```
module SimpleRoutingExampleM {

uses interface Boot;
uses interface Leds;

uses interface Timer<TMilli> as Timer0;

uses interface Timer<TMilli> as Timer_Send;

//MAC interfaces
uses interface MLME_START;

uses interface MLME_GET;
uses interface MLME_SET;

uses interface MLME_BEACON_NOTIFY;
uses interface MLME_GTS;

uses interface MLME_ASSOCIATE;
uses interface MLME_DISASSOCIATE;

uses interface MLME_ORPHAN;

uses interface MLME_SYNC;
uses interface MLME_SYNC_LOSS;

uses interface MLME_RESET;

uses interface MLME_SCAN;

uses interface MCPS_DATA;

}
```

```
implementation {

PANDescriptor pan_des;

uint32_t my_short_address=0x00000000;

uint32_t DestinationMote[2];
uint32_t SourceMoteAddr[2];


event void Boot.booted() {

printfUART_init();

printfUART("i_am_pan: %i\n", TYPE_DEVICE);

DestinationMote[0]=0x00000000;
DestinationMote[1]=0x00000002;

call Timer0.startOneShot(4000);
}
```

Ahora el dispositivo EndDevice debe tratar de contactar con su PANCoordinator antes de comenzar a enviarle paquetes.

```
event void Timer0.fired() {

      uint8_t v_temp[2];


      my_short_address = TOS_NODE_ID;

      //Establece la dirección corta.
      v_temp[0] = (uint8_t)(my_short_address >> 8);
      v_temp[1] = (uint8_t)(my_short_address );

      call MLME_SET.request(MACSHORTADDRESS,v_temp);

      //Establece el MAC_PANID
      v_temp[0] = (uint8_t)(MAC_PANID >> 8);
      v_temp[1] = (uint8_t)(MAC_PANID );

      //Contacta con dicho PANCoodinator
      call MLME_SET.request(MACPANID,v_temp);

      //Se modifica el LED0 para indicar que ha
      contactado con el PANCoordinator
      call Leds.led0Toggle();

      //Genera un evento que se repite cada 1000ms y
      que se va a encargar del envío de paquetes al
      otro extremo.
      call Timer_Send.startPeriodic(1000);

}
```

El evento que gestiona el envío de paquetes es el siguiente:

```
event void Timer_Send.fired() {

      //Se crea un campo de datos vacio
      uint8_t msdu_payload[4];

      DestinationMote[0]=0x00000000;
      DestinationMote[1]=0x00000000;

      SourceMoteAddr[0]=0x00000000;
      SourceMoteAddr[1]=TOS_NODE_ID;


      //Se envian los datos
      //set_txoptions(ack, gts,
      indirect_transmission, security)
      call MCPS_DATA.request(SHORT_ADDRESS,
      MAC_PANID, SourceMoteAddr, SHORT_ADDRESS,
      MAC_PANID, DestinationMote, 2,
      msdu_payload,1,set_txoptions(1,0,0,0));
      }
      //Se cambia el estado del LED2 indicado que hay
      transmisión
      call Leds.led2Toggle();

      }
```

Cuando responde PANCoordinator, se cambia el estado del LED1 para indicar que se ha recibido un paquete.

```
event error_t MCPS_DATA.indication(uint16_t
SrcAddrMode, uint16_t SrcPANId, uint32_t SrcAddr[2],
uint16_t DstAddrMode, uint16_t DestPANId, uint32_t
DstAddr[2], uint16_t msduLength,uint8_t
msdu[100],uint16_t mpduLinkQuality, uint16_t
SecurityUse, uint16_t ACLEntry)
{

            call Leds.led1Toggle();

}
```

Al igual que para PANCoordinator, para el correcto funcionamiento del sistema, también deben definirse distintos eventos relacionados con 802.15.4. Dichos eventos no afectan al funcionamiento del escenario, pero deben estar definidos y son iguales a los del PANCoordinator.

El archivo de configuración así como el fichero simpleroutingexample.h están definidos de la misma manera que para el PANCoordinator.

Una vez implementado todo esto, se compila el código y se carga en el dispositivo MicaZ empleando la siguiente sentencia en la consola de XubunTOS:

```
make micaz install.2 eprb,10.1.1.12
```

Ya se dispone de un sistema de transmisión de datos entre dispositivos MicaZ. Falta conseguir que dichos datos sean enviados a un PC. Para ello, en el siguiente apartado, se explica como comunicar un dispositivo MicaZ con un PC.

2.5.2.- Comunicación PC –MicaZ

Ahora se va a tratar de comunicar el PC con el dispositivo MicaZ a través de un dispositivo MIB600 y su conexión Ethernet. El sistema a implementar sería el mostrado en la siguiente figura.

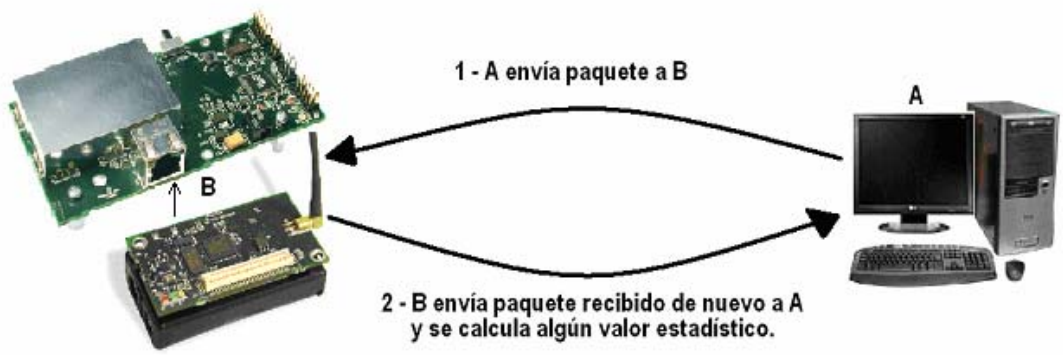

Fig 23 – Comunicación PC-MicaZ

En este escenario, el PC le manda un paquete al dispositivo MicaZ, este dispositivo nada mas recibirlo, lo reenvía al PC. El dispositivo MicaZ está programado utilizando nesC, mientras que el código que se encarga de enviarle paquetes desde el PC al dispositivo MicaZ está programado en Java. En la siguiente figura se muestra el funcionamiento de la implementación:

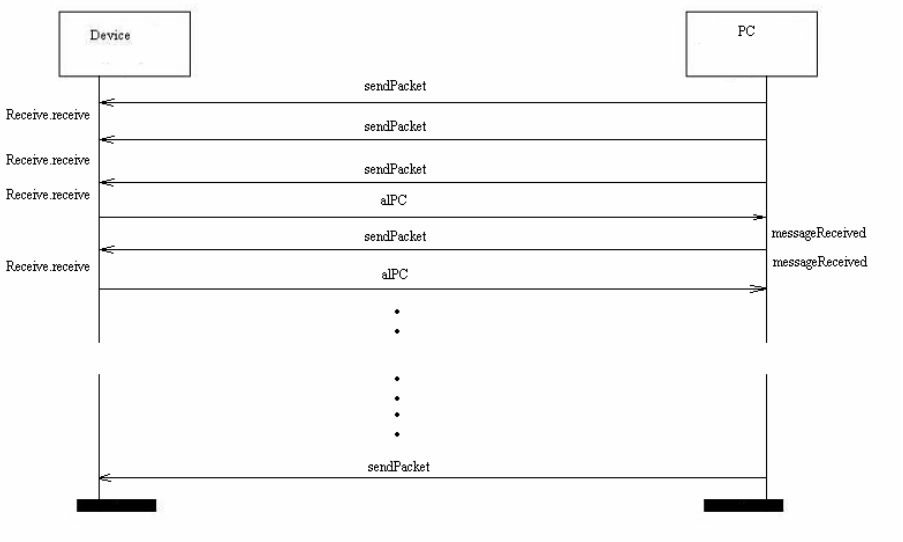

Fig. 24 – Envío de paquetes entre PC y MicaZ

2.5.2.1.- Implementación para el dispositivo MicaZ

Como se indica en el apartado anterior, el dispositivo MicaZ está implementado en nesC. Los ficheros correspondientes a esta implementación se encuentran en el directorio `SimpleRoutingExample\MicaZ` del directorio raíz.

La implementación se desarrolla en el fichero SimpleRoutingExampleM.nc. En primer lugar se cargan los interfaces necesarios de la comunicación serie y el control del dispositivo MicaZ y se realiza la definición e inicialización de variables que van a ser utilizadas a lo largo de esta implementación.

```
module SimpleRoutingExampleM {
uses interface Boot;
//Comunicacion serie
uses interface SplitControl as Control;
uses interface Receive;
      uses interface AMSend;
      uses interface Packet;

}
implementation {

//Serial
message_t packet;

      bool locked = FALSE;
      uint16_t counter = 0;
void alPC(uint8_t msduaux);


  event void Boot.booted() {
      call Control.start();

}
```

A continuación se crea la función que va a gestionar el envío de datos al PC.

```
void alPC(uint8_t aux) {
    if (locked) {
      return;
    }
    else {
     test_serial_msg_t* rcm =
(test_serial_msg_t*)call Packet.getPayload(&packet,
NULL);
      if (call Packet.maxPayloadLength() <
sizeof(test_serial_msg_t)) {
return;
      }

      rcm->counter = aux;
      if (call AMSend.send(AM_BROADCAST_ADDR,
&packet, sizeof(test_serial_msg_t))== SUCCESS) {
locked = TRUE;
      }
    }
}
```

La siguiente función se encarga de la recepción de datos desde el PC, y que se ejecute la función de envío de los datos recibidos al PC.

```
event message_t* Receive.receive(message_t* bufPtr,
                    void* payload, uint8_t len) {
    uint8_t carga;
if (len != sizeof(test_serial_msg_t)) {return
bufPtr;}
    else {
      test_serial_msg_t* rcm =
(test_serial_msg_t*)payload;
      carga=rcm->counter;

     call Leds.led0Toggle();
     alPC(carga);
     return bufPtr;
    }
  }
```

Por último, deben estar definidas algunas funciones relacionadas con el control de la comunicación.

```
event void AMSend.sendDone(message_t* bufPtr, error_t
error) {
    if (&packet == bufPtr) {
      locked = FALSE;
    }
  }

  event void Control.startDone(error_t err) {
    if (err == SUCCESS) {
      //call MilliTimer.startPeriodic(1000);
    }
  }
```

```
  event void Control.stopDone(error_t err) {}
```

El archivo de configuración SimpleRoutingExample.nc está definido de la siguiente manera:

```
#include <Timer.h>

//-------------------------------
#include "TestSerial.h"
//-------------------------------

configuration SimpleRoutingExample {
}

implementation
{
  components MainC;
  components SimpleRoutingExampleM;
  components SerialActiveMessageC as AM;
  components new TimerMilliC();

  SimpleRoutingExampleM.Boot -> MainC;

  components Mac;

  //Gestiona la comunicación Serie (Control,
recepción, envío y formato de paquetes)
  SimpleRoutingExampleM.Control -> AM;
  SimpleRoutingExampleM.Receive ->
AM.Receive[AM_TEST_SERIAL_MSG];
  SimpleRoutingExampleM.AMSend ->
AM.AMSend[AM_TEST_SERIAL_MSG];
  SimpleRoutingExampleM.Packet -> AM;
                       }
```

Finalmente hay que definir el fichero TestSerial.h de la siguiente forma:

```
#ifndef TEST_SERIAL_H
#define TEST_SERIAL_H

typedef nx_struct test_serial_msg {
  nx_uint16_t counter;
} test_serial_msg_t;

enum {
  AM_TEST_SERIAL_MSG = 9,
};

#endif
```

2.5.2.2.- Implementación en PC

Para que el PC mande paquetes al dispositivo y que reciba las respuestas, hay que desarrollar una aplicación en Java. Los

ficheros correspondientes a esta implementación se encuentran en el directorio `SimpleRoutingExample\PC` del directorio raíz.

La implementación se desarrolla en el fichero TestSerial.java. Para desarrollar dicha aplicación, lo primero que hay que hacer es incluir todas las librerías necesarias para trabajar con TinyOS y para almacenar las estadísticas en ficheros de texto.

```
//Almacenamiento en ficheros
import java.io.IOException;
import java.io.*;
//Control de TinyOS
import net.tinyos.message.*;
import net.tinyos.packet.*;
import net.tinyos.util.*;
```

El siguiente paso es la definición de variables que van a ser utilizadas a lo largo del desarrollo de la aplicación y su inicialización.

```
private MoteIF moteIF;
  private long retardo[];
  private int contador;
  private int counter;
  private long todos;
  private DataOutputStream dos;
  private TestSerialMsg payload;
  public TestSerial(MoteIF moteIF) {
    counter=0;
    contador=0;
    todos=0;
    payload = new TestSerialMsg();
    retardo=new long[256];
    this.moteIF = moteIF;
    this.moteIF.registerListener(new TestSerialMsg(),
this);
    try{
dos = new DataOutputStream(new
FileOutputStream("log/Retardo.log"));
     }catch(Exception e){
System.out.println(e.getMessage());
    }

  }
```

La función que se encarga del envío de los paquetes es sendPacket. SendPacket envÍa constantemente paquetes al dispositivo micaZ. Nada más terminar el envío de un paquete, se envía el siguiente. Esta función se define como se muestra a continuación.

```
public void sendPacket() {


    try {
```

```
            //System.out.println("Mandando mensaje
"+contador+" ...");
payload.set_counter(contador);
counter=counter+1;
moteIF.send(0, payload);
retardo[contador]=System.nanoTime();

//La variable contador indica el numero de secuencia
del paquete enviado, para poder identificarlo al
recibirlo//
if(contador==255){
      contador=0;

      }
      else{
      contador=contador+1;
      }
    }
    catch (IOException exception) {
      System.err.println("Exception thrown when
sending packets. Exiting.");
      System.err.println(exception);
    }

    sendPacket();


}
```

La función que gestiona la recepción de paquetes por parte del dispositivo MicaZ se define como messageReceived. Dicha función se encarga también de almacenar el tiempo de llegada de cada paquete en un fichero de texto, para así verificar su recepción (identificándolo con el número de secuencia) y determinar también el retardo de la recepción de paquetes.

```
public void messageReceived(int to, Message message)
{
     if(todos==0){
todos=System.nanoTime();
     }
     int paquete;
     String datos;
     TestSerialMsg msg = (TestSerialMsg)message;
     paquete=msg.get_counter();
     retardo[paquete]=System.nanoTime()-todos;
     datos="Recibido mensaje "+paquete+" con un
tiempo de " + retardo[paquete]/1000000+"\n";
     try{
      dos.writeBytes(datos);
     }catch(Exception e){
System.out.println(e.getMessage()+" error al
almacenar estadisticas");
     }

  }
```

Por último, se define *usage* para informar al usuario de los errores en los comandos necesarios para la inicialización del programa, y main, que se encarga de lanzar el programa al hacer una inicialización correcta de éste.

```
private static void usage() {
    System.err.println("usage: TestSerial [-comm
<source>]");
  }

  public static void main(String[] args) throws
Exception {
    String source = null;
    if (args.length == 2) {
      if (!args[0].equals("-comm")) {
usage();
System.exit(1);
      }
      source = args[1];
    }
    else if (args.length != 0) {
      usage();
      System.exit(1);
    }

    PhoenixSource phoenix;

    if (source == null) {
      phoenix =
BuildSource.makePhoenix(PrintStreamMessenger.err);
    }
    else {
      phoenix = BuildSource.makePhoenix(source,
PrintStreamMessenger.err);
    }

    MoteIF mif = new MoteIF(phoenix);
    TestSerial serial = new TestSerial(mif);
    serial.sendPacket();
  }
```

Es interesante la descripción de como funciona el mecanismo de orfandad y búsqueda pasiva de canales. En el siguiente apartado se explica el funcionamiento de la herramienta AssociationExample para dicho análisis.

2.5.3.- Orfandad y búsqueda pasiva de canales

La herramienta expuesta en este apartado se emplea para hacer un análisis de las características de orfandad y búsqueda pasiva de canales. Como se describe en el apartado 1.3.4. del capítulo 2, cuando un dispositivo MicaZ se quiere conectar a un coordinador, puede realizar un escaneo de todos los canales para buscar el coordinador que le pueda

ofrecer las mejores condiciones de conexión, es decir, el coordinador que más nivel de señal le ofrezca. A este mecanismo se le conoce como escaneo pasivo de canales. Una vez determinado cual es el mejor coordinador, se conecta a éste y comienza a funcionar en dicha red. (Fig.24,Fig. 25 y Fig. 26)

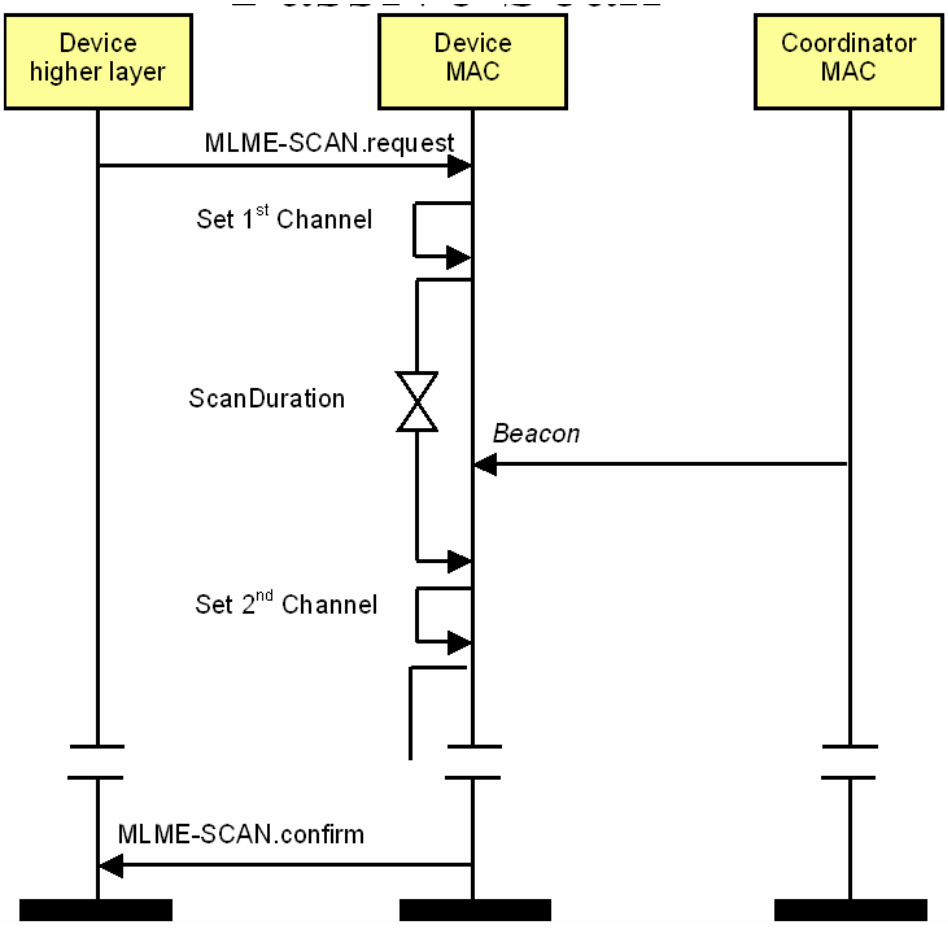

Fig. 25 – Escaneo pasivo de canales

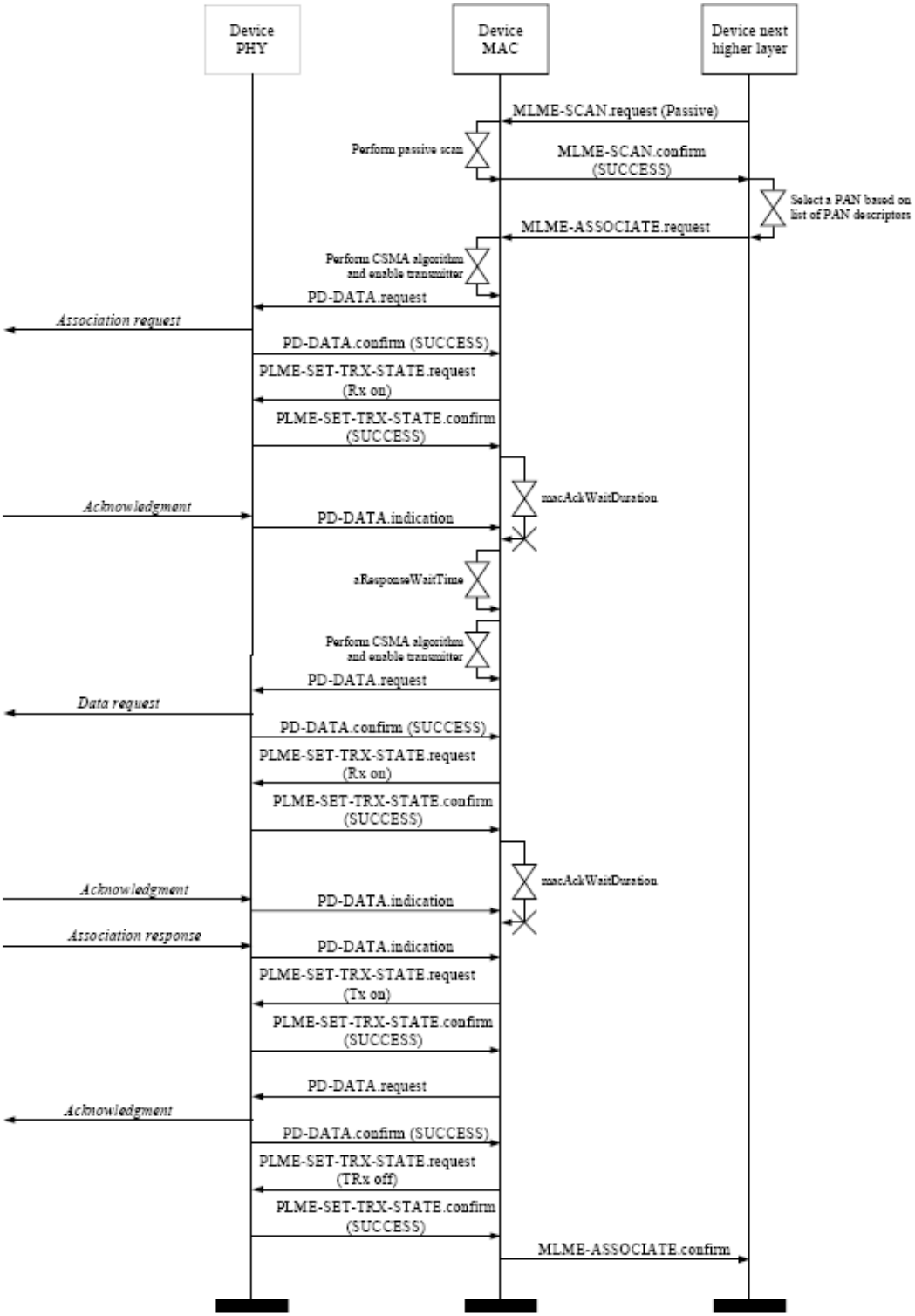

Fig. 26 – Mecanismo de asociación pasiva del dispositivo EndDevice

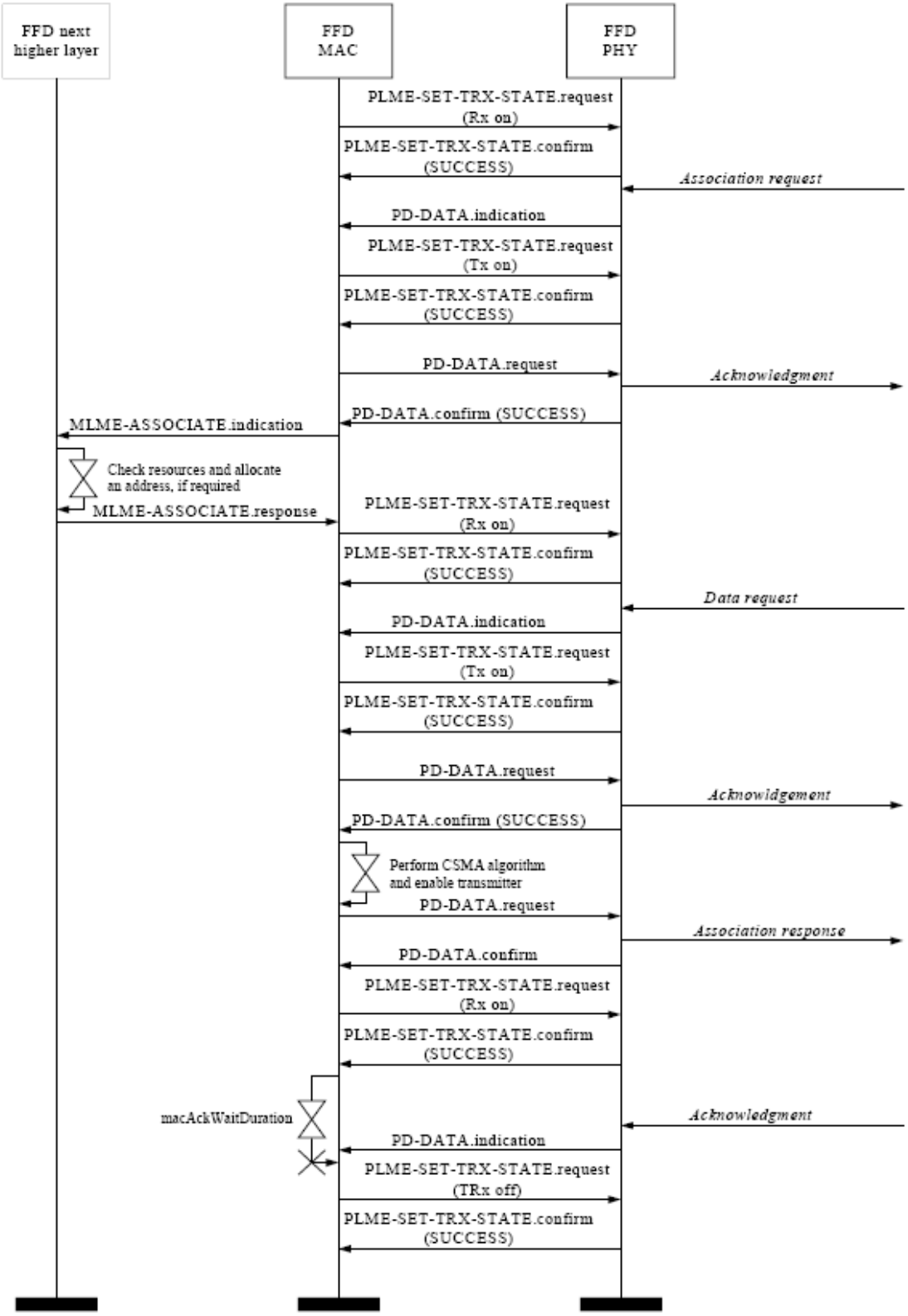

Fig. 27 – Mecanismo de asociación pasiva del dispositivo EndDevice

Si el dispositivo en algún momento pierde la comunicación con el coordinador, durante un tiempo va a intentar reconectarse a éste y en caso

de no poder, y pasado dicho tiempo, empleará el mecanismo de escaneo pasivo para buscar otro coordinador. Este mecanismo se conoce como mecanismo de orfandad y está explicado en el apartado 1.3.5. del capitulo 2. A continuación se muestra el funcionamiento de dicho mecanismo:

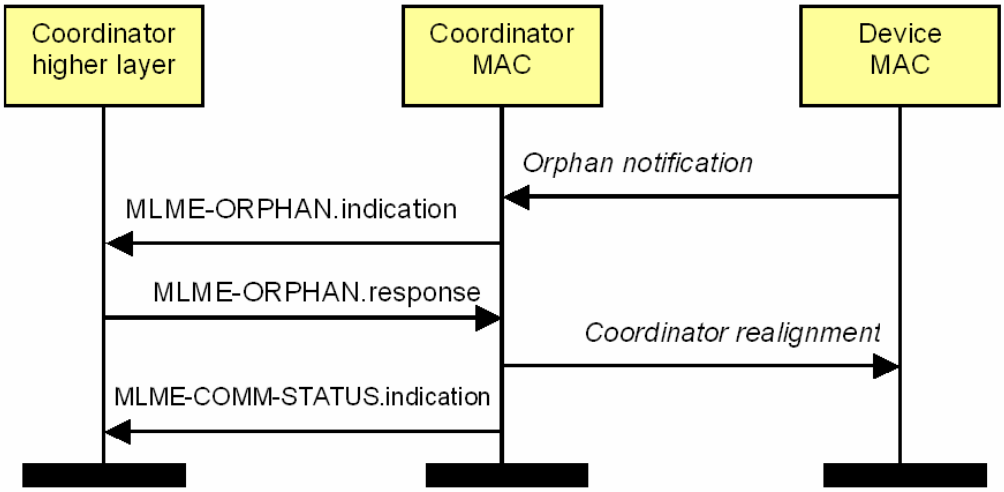

Fig. 28 – Secuencia de mensajes del mecanismo de orfandad

A continuación se va a describir como se ha desarrollado una herramienta para el análisis de dichos mecanismos. El sistema a implementar incluye dos dispositivos MicaZ en los cuales uno de los dispositivos hace las veces de coordinador, y el otro de extremo remoto que quiere conectarte o reconectarse a dicho coordinador mediante los mecanismos explicados. Una vez conectado al coordinador, éste se dedica a enviarle paquetes cada cierto tiempo. Dicha transmisión se para cuando se pierde la conexión con el coordinador, y se reactiva dicha transmisión si se reconecta o si se conecta a otro coordinador. En la figura siguiente se muestra el sistema a implementar.

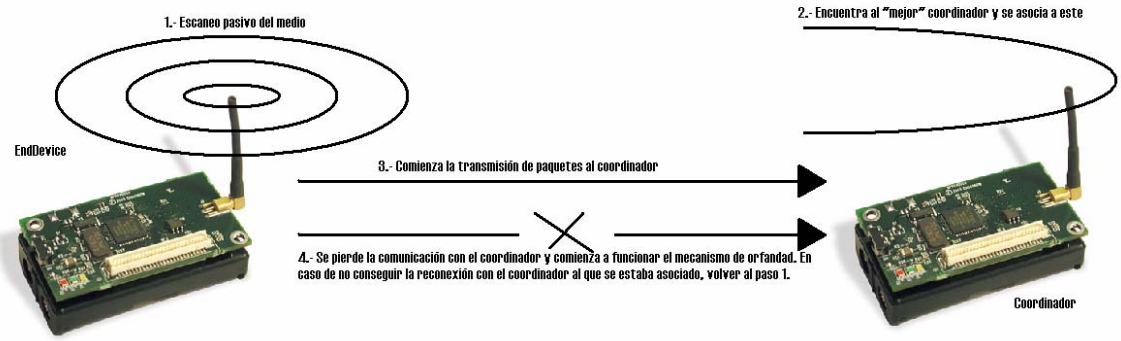

Fig 29 – Orfandad y escaneo pasivo

2.5.3.1.- Implementación del coordinador

A continuación se va a explicar como se desarrolla el coordinador. Los ficheros correspondientes a esta implementación se encuentran en el directorio `AssociationExample\Coordinator` del directorio raíz.

La implementación se desarrolla en el fichero AssociationExampleM.nc. En primer lugar se cargan los interfaces necesarios para el uso de 802.15.4 y el control del dispositivo MicaZ y se realiza la definición e inicialización de variables que van a ser utilizadas a lo largo de esta implementación

```
#include <Timer.h>
#include "printfUART.h"


module AssociationExampleM {

uses interface Boot;
uses interface Leds;

uses interface Timer<TMilli> as Timer0;

uses interface Timer<TMilli> as Timer_Send;

//Interfaces MAC

uses interface MLME_START;

uses interface MLME_GET;
uses interface MLME_SET;

uses interface MLME_BEACON_NOTIFY;
uses interface MLME_GTS;

uses interface MLME_ASSOCIATE;
uses interface MLME_DISASSOCIATE;

uses interface MLME_ORPHAN;

uses interface MLME_SYNC;
uses interface MLME_SYNC_LOSS;

uses interface MLME_RESET;

uses interface MLME_SCAN;


uses interface MCPS_DATA;

}
implementation {

//associated devices
uint16_t address_poll = 0x0003;
```

```
neighbour_table associated_devices[4];

uint16_t search_associated_devices(uint32_t ext1,
uint32_t ext2);

uint8_t number_associations =0;

PANDescriptor pan_des;

uint16_t my_short_address= 0xffff;

uint8_t received_beacon_count=0;
uint32_t coordinator_addr[2];

uint8_t go_associate =0;

event void Boot.booted() {

      //Se asigna la dirección corta del coordinador
y se genera un evento para que el coordinador se
establezca como tal
      my_short_address = 0x0000;
      call Timer0.startOneShot(3000);

}
```

Ahora el coordinador se establece como tal y comienza el proceso de envío de beacons.

```
event void Timer0.fired() {

      uint8_t v_temp[2];
      uint32_t c_addr[2];
      frame_counter=1;

      associated_devices[0].extended1=0x00000002;

      associated_devices[0].extended2=0x00000002;

      associated_devices[0].assigned_short=0x0004;

      //Se establece la dirección corta
      v_temp[0] = (uint8_t)(my_short_address >> 8);
      v_temp[1] = (uint8_t)(my_short_address );

      call MLME_SET.request(MACSHORTADDRESS,v_temp);

      //Se establece el identificador de PAN
      v_temp[0] = (uint8_t)(MAC_PANID >> 8);
      v_temp[1] = (uint8_t)(MAC_PANID );

      call MLME_SET.request(MACPANID,v_temp);

      //Comienza el envio de beacons
      call MLME_START.request(MAC_PANID,
      LOGICAL_CHANNEL, BEACON_ORDER,
      SUPERFRAME_ORDER,1,0,0,0,0);
}
```

Un dispositivo trata de asociarse al coordinador y éste genera el siguiente evento:

```
/****MLME-ASSOCIATE****/
event error_t MLME_ASSOCIATE.indication(uint32_t
DeviceAddress[], uint8_t CapabilityInformation, bool
SecurityUse, uint8_t ACLEntry)
{
//El dispositivo coordinador recibe la peticion de
asociación y le asigna una dirección corta al
dispositivo.
      address_poll ++;
      number_associations++;

      call
MLME_ASSOCIATE.response(DeviceAddress,address_poll,
CMD_RESP_ASSOCIATION_SUCCESSFUL, 0);
return SUCCESS;
}
```

El coordinador se queda ahora a la espera de recibir paquetes. Cuando recibe un paquete, éste simplemente lo procesa y no hace nada más. Esto se puede observar en la función que gestiona la recepción de paquetes, la cual es una función vacía.

```
/*****MCPS-DATA******/

event error_t MCPS_DATA.confirm(uint8_t msduHandle,
uint8_t status)
{

return SUCCESS;
}
event error_t MCPS_DATA.indication(uint16_t
SrcAddrMode, uint16_t SrcPANId, uint32_t SrcAddr[2],
uint16_t DstAddrMode, uint16_t DestPANId, uint32_t
DstAddr[2], uint16_t msduLength,uint8_t
msdu[100],uint16_t mpduLinkQuality, uint16_t
SecurityUse, uint16_t ACLEntry)
{

return SUCCESS;
}
```

Para el correcto funcionamiento del sistema, también deben definirse distintos eventos relacionados con 802.15.4. Dichos eventos no afectan al funcionamiento del escenario ya que son funciones vacías, pero deben estar definidos ya que estan definidos en el interfaz de la capa MAC.

```
/********MLME-SCAN*******/
event error_t MLME_SCAN.confirm(uint8_t
status,uint8_t ScanType, uint32_t UnscannedChannels,
uint8_t ResultListSize, uint8_t EnergyDetectList[],
SCAN_PANDescriptor PANDescriptorList[])
{
```

```
      return SUCCESS;
}

/*****MLME-ORPHAN*****/
event error_t MLME_ORPHAN.indication(uint32_t
OrphanAddress[1], uint8_t SecurityUse, uint8_t
ACLEntry)
{

return SUCCESS;
}


/*****MLME-RESET*****/
event error_t MLME_RESET.confirm(uint8_t status)
{

return SUCCESS;
}
/*****MLME-SYNC-LOSS*****/
event error_t MLME_SYNC_LOSS.indication(uint8_t
LossReason)
{
return SUCCESS;
}
/******MLME-GTS******/
event error_t MLME_GTS.confirm(uint8_t
GTSCharacteristics, uint8_t status)
{

return SUCCESS;
}

event error_t MLME_GTS.indication(uint16_t
DevAddress, uint8_t GTSCharacteristics, bool
SecurityUse, uint8_t ACLEntry)
{

return SUCCESS;
}

/*****MLME-BEACON NOTIFY******/
event error_t MLME_BEACON_NOTIFY.indication(uint8_t
BSN,PANDescriptor pan_descriptor, uint8_t
PenAddrSpec, uint8_t AddrList, uint8_t sduLength,
uint8_t sdu[])
{

return SUCCESS;
}
/******MLME-START******/
event error_t MLME_START.confirm(uint8_t status)
{
return SUCCESS;
}
/******MLME-SET******/
event error_t MLME_SET.confirm(uint8_t status,uint8_t
PIBAttribute)
{

return SUCCESS;
```

```
}

/*****MLME-GET****/
event error_t MLME_GET.confirm(uint8_t status,uint8_t
PIBAttribute, uint8_t PIBAttributeValue[])
{
return SUCCESS;
}

/*****MLME-ASSOCIATE******/
event error_t MLME_ASSOCIATE.confirm(uint16_t
AssocShortAddress, uint8_t status)
{

return SUCCESS;
}
/******MLME-DISASSOCIATE******/
event error_t MLME_DISASSOCIATE.indication(uint32_t
DeviceAddress[], uint8_t DisassociateReason, bool
SecurityUse, uint8_t ACLEntry)
{
return SUCCESS;
}

event error_t MLME_DISASSOCIATE.confirm(uint8_t
status)
{
return SUCCESS;
}
```

Una vez definido todo esto, se define un fichero de configuración (AssociationExample.nc) donde se indican todas las relaciones 802.15.4 y la implementación realizada:

```
#include <Timer.h>

#include "associationexample.h"
#include "phy_const.h"
#include "phy_enumerations.h"
#include "mac_const.h"
#include "mac_enumerations.h"
#include "mac_func.h"


configuration AssociationExample {
}
implementation {

  components MainC;
  components LedsC;
  components AssociationExampleM;

  AssociationExampleM.Boot -> MainC;

  components Mac;

  AssociationExampleM.Leds -> LedsC;
```

```
  components new TimerMilliC() as Timer0;
  AssociationExampleM.Timer0 -> Timer0;

  components new TimerMilliC() as Timer_Send;
  AssociationExampleM.Timer_Send ->Timer_Send;

  //MAC interfaces

  AssociationExampleM.MLME_START -> Mac.MLME_START;

  AssociationExampleM.MLME_GET ->Mac.MLME_GET;
  AssociationExampleM.MLME_SET ->Mac.MLME_SET;

  AssociationExampleM.MLME_BEACON_NOTIFY -
>Mac.MLME_BEACON_NOTIFY;
  AssociationExampleM.MLME_GTS -> Mac.MLME_GTS;

  AssociationExampleM.MLME_ASSOCIATE-
>Mac.MLME_ASSOCIATE;
  AssociationExampleM.MLME_DISASSOCIATE-
>Mac.MLME_DISASSOCIATE;

  AssociationExampleM.MLME_ORPHAN->Mac.MLME_ORPHAN;
  AssociationExampleM.MLME_SYNC->Mac.MLME_SYNC;
  AssociationExampleM.MLME_SYNC_LOSS-
>Mac.MLME_SYNC_LOSS;
  AssociationExampleM.MLME_RESET->Mac.MLME_RESET;

  AssociationExampleM.MLME_SCAN->Mac.MLME_SCAN;


  AssociationExampleM.MCPS_DATA->Mac.MCPS_DATA;


}
```

El fichero associationexample.h contiene información sobre la trama MAC, de los beacons y el identificador del PANCoordinator. Este fichero es común para los dos extremos.

```
enum {
COORDINATOR = 0x00,
ROUTER =0x01,
END_DEVICE = 0x02
};

#define BEACON_ORDER 6
#define SUPERFRAME_ORDER 4
#define LOGICAL_CHANNEL 0x15
#define TYPE_DEVICE COORDINATOR

//PAN VARIABLES
#define MAC_PANID 0x1234

typedef struct{

uint32_t extended1;
uint32_t extended2;
uint16_t assigned_short;
```

```
}neighbour_table;
```

Una vez implementado todo esto, se compila el código y se carga en el dispositivo MicaZ empleando la siguiente sentencia en la consola de XubunTOS:

```
make micaz install.2 eprb,10.1.1.12
```

Una vez implementado el coordinador, es necesario implementar el extremo remoto que va a conectarse a dicho coordinador. En el siguiente apartado se explica como implementarlo.

2.5.3.2.- Implementación del EndDevice

A continuación se va a explicar como se desarrolla el EndDevice. Los ficheros correspondientes a esta implementación se encuentran en el directorio `AssociationExample\EndDevice` del directorio raíz.

La implementación se desarrolla en el fichero AssociationExampleM.nc. Primero se cargan los interfaces necesarios para el uso de 802.15.4 y el control del dispositivo MicaZ y se realiza la definición e inicialización de variables que van a ser utilizadas a lo largo de esta implementación.

```
#include <Timer.h>
#include "printfUART.h"


module AssociationExampleM {

uses interface Boot;
uses interface Leds;

uses interface Timer<TMilli> as Timer0;

uses interface Timer<TMilli> as Timer_Send;

//Interfaces MAC

uses interface MLME_START;

uses interface MLME_GET;
uses interface MLME_SET;

uses interface MLME_BEACON_NOTIFY;
uses interface MLME_GTS;

uses interface MLME_ASSOCIATE;
uses interface MLME_DISASSOCIATE;

uses interface MLME_ORPHAN;
```

```
uses interface MLME_SYNC;
uses interface MLME_SYNC_LOSS;

uses interface MLME_RESET;

uses interface MLME_SCAN;


uses interface MCPS_DATA;

}
implementation {

//associated devices
uint16_t address_poll = 0x0003;

neighbour_table associated_devices[4];

uint16_t search_associated_devices(uint32_t ext1,
uint32_t ext2);

uint8_t number_associations =0;

PANDescriptor pan_des;

uint16_t my_short_address= 0xffff;

uint8_t received_beacon_count=0;
uint32_t coordinator_addr[2];

uint8_t go_associate =0;

event void Boot.booted() {

      //Se genera un evento para comenzar el escaneo
pasivo de canales.
      call Timer0.startOneShot(3000);

}
```

Ahora el extremo remoto comienza a la búsqueda de un coordinador al cual asociarse.

```
event void Timer0.fired() {

call MLME_SCAN.request(PASSIVE_SCAN,0xFFFFFFFF,7);

}
```

Cuando la capa MAC termina de escanear todos los canales, a través de la función MLME_SCAN.confirm se establece el canal que mejores calidad ofrece, y trata de sincronizarse con éste. En el caso de que se hubiese aplicado el mecanismo de orfandad, si se llamase a esta función desde la capa MAC, sería porque no se ha logrado reconectar al coordinador pasado un tiempo, lo que implicaría que a través de esta función, se va a comenzar a realizar un escaneo pasivo.

```
/*****MLME-SCAN*****/
```

```
event error_t MLME_SCAN.confirm(uint8_t
status,uint8_t ScanType, uint32_t UnscannedChannels,
uint8_t ResultListSize, uint8_t EnergyDetectList[],
SCAN_PANDescriptor PANDescriptorList[])
{

      int i;
      uint8_t max_lqi=0;
      best_pan_index=0;

//Si se estaba intentando reconectar mediante el
mecanismo de orfandad y no se consigue, pasar a
escaneo pasivo.

      if (ScanType == ORPHAN_SCAN)
      {

            call
MLME_SCAN.request(PASSIVE_SCAN,0xFFFFFFFF,7);
            return SUCCESS;
      }


//Se determina cual es el nodo con mejor nivel de
calidad a partir de la lista aportada por la capa
MAC.

      for (i=0; i<ResultListSize;i++)
      {
            if(max_lqi < PANDescriptorList[i].lqi)
            {
                  max_lqi =PANDescriptorList[i].lqi;
                  best_pan_index = i;
            }
      }


//Una vez establecido el mejor nodo, se establecen
los parámetros de dicho nodo.

      coordinator_addr[0] = 0x00000000;

      coordinator_addr[1] =
      PANDescriptorList[best_pan_index].CoordAddress;


      pan_des.CoordAddrMode = SHORT_ADDRESS;
      pan_des.CoordPANId =
      PANDescriptorList[best_pan_index].CoordPANId;
      pan_des.CoordAddress0=0x00000000;
      pan_des.CoordAddress1=0x00000000;
      pan_des.LogicalChannel=PANDescriptorList[best_p
      an_index].LogicalChannel;
      pan_des.SuperframeSpec =
      PANDescriptorList[best_pan_index].SuperframeSpe
      c;

      pan_des.GTSPermit=0x01;
      pan_des.LinkQuality=0x00;
      pan_des.TimeStamp=0x000000;
      pan_des.SecurityUse=0;
```

```
      pan_des.ACLEntry=0x00;
      pan_des.SecurityFailure=0x00;

//Se procede a tratar de sincronizarse con el
coordinador elegido y asociarse a este.
      received_beacon_count=0;
      go_associate=1;
//Se activan los eventos asíncronos para hacer
posible la sincronización con el coordinador a través
de la capa MAC.


call
MLME_SYNC.request(PANDescriptorList[best_pan_index].L
ogicalChannel,0);


      return SUCCESS;
}
```

Antes de asociarse con el coordinador, se deben recibir cinco tramas de beacon correctas desde el mejor coordinador. Una vez recibidas, comienza el proceso de asociación.

```
/*****MLME-BEACON NOTIFY*****/
event error_t MLME_BEACON_NOTIFY.indication(uint8_t
BSN,PANDescriptor pan_descriptor, uint8_t
PenAddrSpec, uint8_t AddrList, uint8_t sduLength,
uint8_t sdu[])
{

if (go_associate==1)
{
      received_beacon_count++;

      //printfUART("bn %i\n", received_beacon_count);

      if (received_beacon_count==5)
      {
                  uint32_t c_addr[2];
                  go_associate=0;
                  c_addr[0] = 0x00000000;
                  c_addr[1] = 0x00000000;
                  //Se asocia al coordinador
                  call
                  MLME_ASSOCIATE.request(pan_des.Logi
                  calChannel,SHORT_ADDRESS,pan_des.Co
                  ordPANId,c_addr,0x00,0x00);


      }

}

return SUCCESS;
}
```

Para asociarse al coordinador definitivamente, se emplea el siguiente evento.

```
/****MLME-ASSOCIATE****/
event error_t MLME_ASSOCIATE.indication(uint32_t
DeviceAddress[], uint8_t CapabilityInformation, bool
SecurityUse, uint8_t ACLEntry)
{
      return SUCCESS;
}

event error_t MLME_ASSOCIATE.confirm(uint16_t
AssocShortAddress, uint8_t status)
{

//El extremo remoto recibe la confirmación de
asociación y activa la temporización para el envío de
datos.
uint8_t v_temp[2];

      my_short_address = AssocShortAddress;

      v_temp[0] = (my_short_address >> 8);
      v_temp[1] = my_short_address;

      //Establece la direccion del coordinador
      call MLME_SET.request(MACSHORTADDRESS,v_temp);

      temp[0] = (uint8_t)(pan_des.CoordPANId >> 8);
      temp[1] = (uint8_t)(pan_des.CoordPANId );

      //Establece el canal del coordinador
      call MLME_SET.request(MACPANID,temp);

      //Comienza la transmision de datos
      call Timer_Send.startOneShot(3000);

      call Timer0.stop();

return SUCCESS;
}
```

Una vez asociado, se transmite cada cierto tiempo una trama de datos. Para dicho envío, se emplea el siguiente evento.

```
event void Timer_Send.fired() {

uint32_t SrcAddr[2];
uint32_t DstAddr[2];

uint8_t msdu_payload[4];



      if (my_short_address == 0x0000ffff)
            return;

      SrcAddr[0]=0x00000000;
      SrcAddr[1]=my_short_address;

      DstAddr[0]=0x00000000;
      DstAddr[1]=0x00000000;
```

```
      if(frame_counter==1){
            call MCPS_DATA.request(SHORT_ADDRESS,
MAC_PANID, SrcAddr, SHORT_ADDRESS, MAC_PANID,
DstAddr, 4, msdu_payload,1,set_txoptions(1,0,0,0));
            alPC(0x02);

      }
      call Timer_Send.startOneShot(3000);

}
```

Si en algún momento se pierde el contacto con el extremo remoto, la capa MAC llama al evento MLME-SYNC-LOSS.indication para que inicie el mecanismo de orfandad.

```
/*****MLME-SYNC-LOSS*****/
event error_t MLME_SYNC_LOSS.indication(uint8_t
LossReason)
{
      frame_counter=0;
      //Comienza el mecanismo de orfandad de la capa
      MAC.
      call
      MLME_SCAN.request(ORPHAN_SCAN,0xFFFFFFFF,7);


      return SUCCESS;
}
```

En el caso de que no lograse reconectarse, como se ha dicho anteriormente, se llamaría al evento MLME_SCAN.confirm y se procedería a realizar un escaneo pasivo. Si se lograse reconectar, la capa MAC a través de la función process_coordinator_realignment realinearía al extremo remoto con el coordinador. En los escenarios implementados en los siguientes capítulos, se puede observar como después de una realineación se reactiva el envío de paquetes también.

Para el correcto funcionamiento del sistema, también deben definirse distintos eventos relacionados con 802.15.4. Dichos eventos no afectan al funcionamiento del escenario ya que son funciones vacias, pero deben estar definidos ya que estan definidos en el interfaz de la capa MAC.

```
/*****MLME-RESET*****/
event error_t MLME_RESET.confirm(uint8_t status)
{

return SUCCESS;
}
```

```
/******MLME-GTS******/
event error_t MLME_GTS.confirm(uint8_t
GTSCharacteristics, uint8_t status)
{

return SUCCESS;
}

event error_t MLME_GTS.indication(uint16_t
DevAddress, uint8_t GTSCharacteristics, bool
SecurityUse, uint8_t ACLEntry)
{

return SUCCESS;
}


/******MLME-START******/
event error_t MLME_START.confirm(uint8_t status)
{
return SUCCESS;
}
/******MLME-SET******/
event error_t MLME_SET.confirm(uint8_t status,uint8_t
PIBAttribute)
{

return SUCCESS;
}

/*****MLME-GET****/
event error_t MLME_GET.confirm(uint8_t status,uint8_t
PIBAttribute, uint8_t PIBAttributeValue[])
{
return SUCCESS;
}


/******MLME-DISASSOCIATE******/
event error_t MLME_DISASSOCIATE.indication(uint32_t
DeviceAddress[], uint8_t DisassociateReason, bool
SecurityUse, uint8_t ACLEntry)
{
return SUCCESS;
}

event error_t MLME_DISASSOCIATE.confirm(uint8_t
status)
{
return SUCCESS;
}
```

Una vez definido todo esto, se define un fichero de configuración(AssociationExample.nc) donde se indican todas las relaciones 802.15.4 y la implementación realizada:

```
#include <Timer.h>

#include "associationexample.h"
#include "phy_const.h"
```

```
#include "phy_enumerations.h"
#include "mac_const.h"
#include "mac_enumerations.h"
#include "mac_func.h"


configuration AssociationExample {
}
implementation {

  components MainC;
  components LedsC;
  components AssociationExampleM;

  AssociationExampleM.Boot -> MainC;

  components Mac;

  AssociationExampleM.Leds -> LedsC;

  components new TimerMilliC() as Timer0;
  AssociationExampleM.Timer0 -> Timer0;

  components new TimerMilliC() as Timer_Send;
  AssociationExampleM.Timer_Send ->Timer_Send;

  //MAC interfaces

  AssociationExampleM.MLME_START -> Mac.MLME_START;

  AssociationExampleM.MLME_GET ->Mac.MLME_GET;
  AssociationExampleM.MLME_SET ->Mac.MLME_SET;

  AssociationExampleM.MLME_BEACON_NOTIFY -
>Mac.MLME_BEACON_NOTIFY;
  AssociationExampleM.MLME_GTS -> Mac.MLME_GTS;

  AssociationExampleM.MLME_ASSOCIATE-
>Mac.MLME_ASSOCIATE;
  AssociationExampleM.MLME_DISASSOCIATE-
>Mac.MLME_DISASSOCIATE;

  AssociationExampleM.MLME_ORPHAN->Mac.MLME_ORPHAN;
  AssociationExampleM.MLME_SYNC->Mac.MLME_SYNC;
  AssociationExampleM.MLME_SYNC_LOSS-
>Mac.MLME_SYNC_LOSS;
  AssociationExampleM.MLME_RESET->Mac.MLME_RESET;

  AssociationExampleM.MLME_SCAN->Mac.MLME_SCAN;


  AssociationExampleM.MCPS_DATA->Mac.MCPS_DATA;
```

El fichero associationexample.h contiene información sobre la trama MAC, de los beacons y el identificador del PANCoordinator. Este fichero es común para los dos extremos.

```
enum {
COORDINATOR = 0x00,
ROUTER =0x01,
```

```
END_DEVICE = 0x02
};

#define BEACON_ORDER 6
#define SUPERFRAME_ORDER 4
#define LOGICAL_CHANNEL 0x15
#define TYPE_DEVICE COORDINATOR

//PAN VARIABLES
#define MAC_PANID 0x1234

typedef struct{

uint32_t extended1;
uint32_t extended2;
uint16_t assigned_short;

}neighbour_table;
```

Una vez implementado todo esto, se compila el código y se carga en el dispositivo MicaZ empleando la siguiente sentencia en la consola de XubunTOS:

```
make micaz install.2 eprb,10.1.1.12
```

También es interesante el poder analizar las tramas GTS, para dicho fin, en el siguiente apartado se explica como implementar un ejemplo.

2.5.4.- Mecanismo GTS

Como se comentó en el apartado 1.3.3. del capitulo 2, algunas aplicaciones requieren anchos de banda dedicados a lograr estados latentes para un consumo de baja potencia. Para lograr dichos estados latentes el IEEE 802.15.4 se emplea el mecanismo de GTS, el cual fue explicado en el apartado indicado anteriormente. Dicho mecanismo viene representado en la siguiente figura:

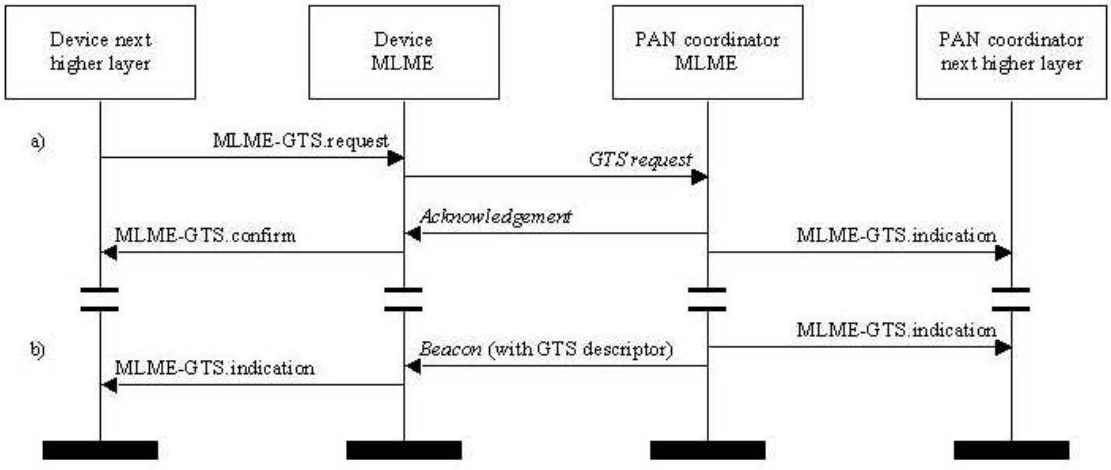

Fig.30 – Funcionamiento del mecanismo de GTS

En los próximos apartados, se explica como implementar aplicación para el análisis del mecanismo de GTS.

2.5.4.1.- Implementación del coordinador

A continuación se va a explicar como se desarrolla el coordinador. Los ficheros correspondientes a esta implementación se encuentran en el directorio `GTSManagementExample \Coordinator` del directorio raíz.

La implementación se desarrolla en el fichero GTSManagementExample.nc. Primero se cargan los interfaces necesarios para el uso de 802.15.4 y el control del dispositivo MicaZ y se realiza la definición e inicialización de variables que van a ser utilizadas a lo largo de esta implementación.

```
#include <Timer.h>
#include "printfUART.h"

module GTSManagementExampleM {

uses interface Boot;
uses interface Leds;

uses interface Timer<TMilli> as Timer0;

uses interface Timer<TMilli> as Timer_Send;

//MAC interfaces

uses interface MLME_START;

uses interface MLME_GET;
uses interface MLME_SET;

uses interface MLME_BEACON_NOTIFY;
uses interface MLME_GTS;

uses interface MLME_ASSOCIATE;
uses interface MLME_DISASSOCIATE;

uses interface MLME_ORPHAN;

uses interface MLME_SYNC;
uses interface MLME_SYNC_LOSS;

uses interface MLME_RESET;

uses interface MLME_SCAN;

uses interface MCPS_DATA;

}
implementation {
```

```
uint8_t beacon_present=0;
uint8_t on_sync=0;
uint8_t gts_allocated=0;

uint8_t gts_superframe_count=0;

PANDescriptor pan_des;

uint32_t my_short_address=0x00000000;
uint32_t my_pan_id=0x00000001;

  event void Boot.booted() {
      call Control.start();

      //Se asigna una dirección corta al coordinador
      my_short_address = 0x0000;
      call Timer0.startOneShot(5000);

  }
```

Ahora el coordinador se establece como tal y comienza el proceso de envío de beacons.

```
event void Timer0.fired() {

      uint8_t v_temp[2];

            //Establece la dirección corta
            v_temp[0] = (uint8_t)(my_short_address >>
            8);
            v_temp[1] = (uint8_t)(my_short_address );

            call
            MLME_SET.request(MACSHORTADDRESS,v_temp);

            //Establece el identificador PAN
            v_temp[0] = (uint8_t)(MAC_PANID >> 8);
            v_temp[1] = (uint8_t)(MAC_PANID );

            call MLME_SET.request(MACPANID,v_temp);

            //Comienza el envio de beacons
            call MLME_START.request(MAC_PANID,
            LOGICAL_CHANNEL, BEACON_ORDER,
            SUPERFRAME_ORDER,1,0,0,0,0);

      //Se inicia la transmisión de paquetes. Se
      desactiva la opción para un análisis de GTS en
      transmisión desde el extremo remoto.
      //call Timer_Send.startOneShot(1000);//Esto se
      utiliza en caso de que el coordinador también
      transmita datos. En principio se desactiva esta
      opción, aunque cabe la posibilidad de usarla.
}
```

El coordinador se queda ahora a la espera de recibir paquetes y enviando paquetes al medio (si se activa la opción de enviar paquetes).

```
event void Timer_Send.fired() {


      uint32_t SrcAddr[2];
      uint32_t DstAddr[2];
      uint8_t msdu_payload[4];

            SrcAddr[0]=0x00000000;
            SrcAddr[1]=TOS_NODE_ID;

            DstAddr[0]=0x00000000;
            DstAddr[1]=0x00000002;

            call MCPS_DATA.request(SHORT_ADDRESS,
            MAC_PANID, SrcAddr, SHORT_ADDRESS,
            MAC_PANID, DstAddr, 4,
            msdu_payload,1,set_txoptions(1,1,0,0));
            call Timer_Send.startOneShot(1000);
}
```

Cuando recibe un paquete, este simplemente lo procesa y no hace nada más. Esto se puede observar en la función que gestiona la recepción de paquetes, la cual es una función vacía.

```
/*****MCPS-DATA******/

event error_t MCPS_DATA.confirm(uint8_t msduHandle,
uint8_t status)
{

return SUCCESS;
}
event error_t MCPS_DATA.indication(uint16_t
SrcAddrMode, uint16_t SrcPANId, uint32_t SrcAddr[2],
uint16_t DstAddrMode, uint16_t DestPANId, uint32_t
DstAddr[2], uint16_t msduLength,uint8_t
msdu[100],uint16_t mpduLinkQuality, uint16_t
SecurityUse, uint16_t ACLEntry)
{

return SUCCESS;
}
```

Según el tipo de evento GTS producido en el sistema, se va a gestionar de una manera u otra a través de dos eventos:

```
/*****MLME-GTS*****/
//Esta confirmación se recibe tras suceder algún
evento de GTS. gts_allocated indica si la transmisión
GTS ha sido correcta. En este ejemplo, si no funciona
bien GTS, no se genera ningún evento a nivel de
aplicación, solo se modifica el valor del bit
gts_allocated.
event error_t MLME_GTS.confirm(uint8_t
GTSCharacteristics, uint8_t status)
{
      switch(status)
      {
            case MAC_SUCCESS:  gts_allocated=1;
```

```
                              break;

            case MAC_DENIED: gts_allocated=0;

                                          break;

            case MAC_NO_SHORT_ADDRESS:
gts_allocated=0;


      break;

            case MAC_CHANNEL_ACCESS_FAILURE:
gts_allocated=0;


      break;

            case MAC_NO_ACK: gts_allocated=0;


      break;

            case MAC_NO_DATA: gts_allocated=0;


      break;


            default: break;

      }

      return SUCCESS;
}


/*****MLME-BEACON NOTIFY*****/
//Esta indicación se produce cuando se reciben 30
supertramas.
event error_t MLME_BEACON_NOTIFY.indication(uint8_t
BSN,PANDescriptor pan_descriptor, uint8_t
PenAddrSpec, uint8_t AddrList, uint8_t sduLength,
uint8_t sdu[])
{
      gts_superframe_count++;
      if (gts_superframe_count==30)
      {
            call
            MLME_GTS.request(set_gts_characteristics(
            1, GTS_TX_ONLY,0),0x00);
      }
      return SUCCESS;
}
```

Para el correcto funcionamiento del sistema, también deben definirse distintos eventos relacionados con 802.15.4. Dichos eventos no afectan al funcionamiento del escenario ya que

son funciones vacías, pero deben estar definidos ya que estan definidos en el interfaz de la capa MAC.

```
/******MLME-SCAN******/
event error_t MLME_SCAN.confirm(uint8_t
status,uint8_t ScanType, uint32_t UnscannedChannels,
uint8_t ResultListSize, uint8_t EnergyDetectList[],
SCAN_PANDescriptor PANDescriptorList[])
{

return SUCCESS;
}

/******MLME-ORPHAN******/
event error_t MLME_ORPHAN.indication(uint32_t
OrphanAddress[1], uint8_t SecurityUse, uint8_t
ACLEntry)
{
return SUCCESS;
}

/******MLME-RESET******/
event error_t MLME_RESET.confirm(uint8_t status)
{



return SUCCESS;
}

/******MLME-SYNC-LOSS******/
event error_t MLME_SYNC_LOSS.indication(uint8_t
LossReason)
{

return SUCCESS;
}

/*****MLME-START*****/
    event error_t MLME_START.confirm(uint8_t status)
{


return SUCCESS;
}
/*****MLME-SET*****/

      event error_t MLME_SET.confirm(uint8_t
status,uint8_t PIBAttribute)
{


return SUCCESS;
}
/*****MLME-GET******/
event error_t MLME_GET.confirm(uint8_t status,uint8_t
PIBAttribute, uint8_t PIBAttributeValue[])
{


return SUCCESS;
```

```
}


/*****MLME-ASSOCIATE******/
event error_t MLME_ASSOCIATE.indication(uint32_t
DeviceAddress[], uint8_t CapabilityInformation, bool
SecurityUse, uint8_t ACLEntry)
{

return SUCCESS;
}

event error_t MLME_ASSOCIATE.confirm(uint16_t
AssocShortAddress, uint8_t status)
{
return SUCCESS;
}
/******MLME-DISASSOCIATE*****/
event error_t MLME_DISASSOCIATE.indication(uint32_t
DeviceAddress[], uint8_t DisassociateReason, bool
SecurityUse, uint8_t ACLEntry)
{
return SUCCESS;
}

event error_t MLME_DISASSOCIATE.confirm(uint8_t
status)
{
return SUCCESS;
}

/*****MCPS-DATA *****/
event error_t MCPS_DATA.confirm(uint8_t msduHandle,
uint8_t status)
{

return SUCCESS;
}
event error_t MCPS_DATA.indication(uint16_t
SrcAddrMode, uint16_t SrcPANId, uint32_t SrcAddr[2],
uint16_t DstAddrMode, uint16_t DestPANId, uint32_t
DstAddr[2], uint16_t msduLength,uint8_t
msdu[100],uint16_t mpduLinkQuality, uint16_t
SecurityUse, uint16_t ACLEntry)
{


return SUCCESS;
}
```

Una vez definido todo esto, se define un fichero de configuración donde se indican todas las relaciones 802.15.4 y la implementación realizada:

```
#include <Timer.h>


#include "gtsmanagementexample.h"
#include "phy_const.h"
#include "phy_enumerations.h"
#include "mac_const.h"
#include "mac_enumerations.h"
#include "mac_func.h"


configuration GTSManagementExample {
}
implementation {

  components MainC;
  components LedsC;
  components GTSManagementExampleM;

  GTSManagementExampleM.Boot -> MainC;

  components Mac;

  GTSManagementExampleM.Leds -> LedsC;

  components new TimerMilliC() as Timer0;
  GTSManagementExampleM.Timer0 -> Timer0;

  components new TimerMilliC() as Timer_Send;
  GTSManagementExampleM.Timer_Send ->Timer_Send;



  //MAC interfaces

  GTSManagementExampleM.MLME_START -> Mac.MLME_START;

  GTSManagementExampleM.MLME_GET ->Mac.MLME_GET;
  GTSManagementExampleM.MLME_SET ->Mac.MLME_SET;

  GTSManagementExampleM.MLME_BEACON_NOTIFY -
>Mac.MLME_BEACON_NOTIFY;
  GTSManagementExampleM.MLME_GTS -> Mac.MLME_GTS;

  GTSManagementExampleM.MLME_ASSOCIATE-
>Mac.MLME_ASSOCIATE;
  GTSManagementExampleM.MLME_DISASSOCIATE-
>Mac.MLME_DISASSOCIATE;

  GTSManagementExampleM.MLME_ORPHAN->Mac.MLME_ORPHAN;
  GTSManagementExampleM.MLME_SYNC->Mac.MLME_SYNC;
  GTSManagementExampleM.MLME_SYNC_LOSS-
>Mac.MLME_SYNC_LOSS;
  GTSManagementExampleM.MLME_RESET->Mac.MLME_RESET;

  GTSManagementExampleM.MLME_SCAN->Mac.MLME_SCAN;


  GTSManagementExampleM.MCPS_DATA->Mac.MCPS_DATA;
```

```
}
```

El fichero gtsmanagementexample.h contiene información sobre la trama MAC, de los beacons y el identificador del PANCoordinator. Este fichero es común para los dos extremos.

```
enum {
      COORDINATOR = 0x00,
      ROUTER =0x01,
      END_DEVICE = 0x02
      };

#define BEACON_ORDER 6
#define SUPERFRAME_ORDER 4
#define LOGICAL_CHANNEL 0x15


#define TYPE_DEVICE COORDINATOR

//PAN VARIABLES
#define MAC_PANID 0x1234
```

Una vez implementado todo esto, se compila el código y se carga en el dispositivo MicaZ empleando la siguiente sentencia en la consola de XubunTOS:

```
make micaz install.2 eprb,10.1.1.12
```

Una vez implementado el coordinador, es necesario implementar el extremo remoto que va a conectarse a dicho coordinador. En el siguiente apartado se explica como implementarlo.

2.5.4.2.- Implementación del extremo remoto

A continuación se va a explicar como se desarrolla el extremo remoto. Los ficheros correspondientes a esta implementación se encuentran en el directorio `GTSManagementExample \Coordinator` del directorio raíz.

La implementación se desarrolla en el fichero GTSManagementExample.nc. Primero se cargan los interfaces necesarios para el uso de 802.15.4 y el control del dispositivo MicaZ y se realiza la definición e inicialización de variables que van a ser utilizadas a lo largo de esta implementación.

```
#include <Timer.h>
#include "printfUART.h"

module GTSManagementExampleM {

uses interface Boot;
```

```
uses interface Leds;

uses interface Timer<TMilli> as Timer0;

uses interface Timer<TMilli> as Timer_Send;

//MAC interfaces

uses interface MLME_START;

uses interface MLME_GET;
uses interface MLME_SET;

uses interface MLME_BEACON_NOTIFY;
uses interface MLME_GTS;

uses interface MLME_ASSOCIATE;
uses interface MLME_DISASSOCIATE;

uses interface MLME_ORPHAN;

uses interface MLME_SYNC;
uses interface MLME_SYNC_LOSS;

uses interface MLME_RESET;

uses interface MLME_SCAN;

uses interface MCPS_DATA;

}
implementation {

uint8_t beacon_present=0;
uint8_t on_sync=0;
uint8_t gts_allocated=0;

uint8_t gts_superframe_count=0;

PANDescriptor pan_des;

uint32_t my_short_address=0x00000000;
uint32_t my_pan_id=0x00000001;

  event void Boot.booted() {
      call Timer0.startOneShot(8000);

  }
```

Ahora el coordinador se establece como tal y comienza el proceso de envío de beacons.

```
event void Timer0.fired() {

      uint8_t v_temp[2];

      my_short_address = TOS_NODE_ID;
      v_temp[0] = (uint8_t)(my_short_address >> 8);
      v_temp[1] = (uint8_t)(my_short_address );
```

```
        call MLME_SET.request(MACSHORTADDRESS,v_temp);


        gts_superframe_count=0;


        printfUART("GTS req: %i\n", TYPE_DEVICE);


        //Localiza una transmision GTS – activa un slot
        GTS para la transmision al coordinador.

        call
        MLME_GTS.request(set_gts_characteristics(1,
        GTS_TX_ONLY,1),0x00);

        // Localiza una transmision GTS – activa un
        slot GTS para la recepción desde el
        coordinador. Se desactiva la opción para un
        análisis de GTS en transmisión.
        //call
        MLME_GTS.request(set_gts_characteristics(1,
        GTS_RX_ONLY,1),0x00);


        //Activa la transmision detos hacia el
        coordinador.
        call Timer_Send.startPeriodic(1000);

}
```

El extremo remoto se queda ahora a la espera de recibir paquetes (si se activa la opción) y enviando paquetes al medio.

```
event void Timer_Send.fired() {


        uint32_t SrcAddr[2];
        uint32_t DstAddr[2];
        uint8_t msdu_payload[4];

        SrcAddr[0]=0x00000000;
        SrcAddr[1]=TOS_NODE_ID;

        DstAddr[0]=0x00000000;
        DstAddr[1]=0x00000000;

        call MCPS_DATA.request(SHORT_ADDRESS,
        MAC_PANID, SrcAddr, SHORT_ADDRESS, MAC_PANID,
        DstAddr, 4,
        msdu_payload,1,set_txoptions(1,1,0,0));

}
```

Cuando recibe un paquete, éste simplemente lo procesa y no hace nada más. Esto se puede observar en la función que gestiona la recepción de paquetes, la cual es una función vacía.

```
/*****MCPS-DATA******/

event error_t MCPS_DATA.confirm(uint8_t msduHandle,
uint8_t status)
{

return SUCCESS;
}
event error_t MCPS_DATA.indication(uint16_t
SrcAddrMode, uint16_t SrcPANId, uint32_t SrcAddr[2],
uint16_t DstAddrMode, uint16_t DestPANId, uint32_t
DstAddr[2], uint16_t msduLength,uint8_t
msdu[100],uint16_t mpduLinkQuality, uint16_t
SecurityUse, uint16_t ACLEntry)
{

return SUCCESS;
}
```

Según el tipo de evento GTS producido en el sistema, se va a gestionar de una manera u otra a través de dos eventos:

```
/*****MLME-GTS*****/
//Esta confirmación se recibe tras suceder algún
evento de GTS.
event error_t MLME_GTS.confirm(uint8_t
GTSCharacteristics, uint8_t status)
{
      switch(status)
      {
            case MAC_SUCCESS:  gts_allocated=1;

                              break;

            case MAC_DENIED: gts_allocated=0;

                                    break;

            case MAC_NO_SHORT_ADDRESS:
gts_allocated=0;


      break;

            case MAC_CHANNEL_ACCESS_FAILURE:
gts_allocated=0;


      break;

            case MAC_NO_ACK: gts_allocated=0;


      break;

            case MAC_NO_DATA: gts_allocated=0;


      break;
```

```
            default: break;

      }

      return SUCCESS;
}


/*****MLME-BEACON NOTIFY*****/
//Esta indicación se produce cuando se reciben 30
supertramas.
event error_t MLME_BEACON_NOTIFY.indication(uint8_t
BSN,PANDescriptor pan_descriptor, uint8_t
PenAddrSpec, uint8_t AddrList, uint8_t sduLength,
uint8_t sdu[])
{
      gts_superframe_count++;
      if (gts_superframe_count==30)
      {
            call
            MLME_GTS.request(set_gts_characteristics(
            1, GTS_TX_ONLY,0),0x00);
      }
      return SUCCESS;
}
```

Para el correcto funcionamiento del sistema, también deben definirse distintos eventos relacionados con 802.15.4. Dichos eventos no afectan al funcionamiento del escenario ya que son funciones vacías, pero deben estar definidos ya que están definidos en el interfaz de la capa MAC.

```
/******MLME-SCAN******/
event error_t MLME_SCAN.confirm(uint8_t
status,uint8_t ScanType, uint32_t UnscannedChannels,
uint8_t ResultListSize, uint8_t EnergyDetectList[],
SCAN_PANDescriptor PANDescriptorList[])
{

return SUCCESS;
}

/******MLME-ORPHAN******/
event error_t MLME_ORPHAN.indication(uint32_t
OrphanAddress[1], uint8_t SecurityUse, uint8_t
ACLEntry)
{

return SUCCESS;
}

/******MLME-RESET******/
event error_t MLME_RESET.confirm(uint8_t status)
{



return SUCCESS;
}
```

```
/******MLME-SYNC-LOSS******/
event error_t MLME_SYNC_LOSS.indication(uint8_t
LossReason)
{

return SUCCESS;
}

/*****MLME-START*****/
    event error_t MLME_START.confirm(uint8_t status)
{


return SUCCESS;
}
/*****MLME-SET*****/

      event error_t MLME_SET.confirm(uint8_t
status,uint8_t PIBAttribute)
{


return SUCCESS;
}
/*****MLME-GET******/
event error_t MLME_GET.confirm(uint8_t status,uint8_t
PIBAttribute, uint8_t PIBAttributeValue[])
{


return SUCCESS;
}



/*****MLME-ASSOCIATE******/
event error_t MLME_ASSOCIATE.indication(uint32_t
DeviceAddress[], uint8_t CapabilityInformation, bool
SecurityUse, uint8_t ACLEntry)
{

return SUCCESS;
}

event error_t MLME_ASSOCIATE.confirm(uint16_t
AssocShortAddress, uint8_t status)
{

return SUCCESS;
}

/******MLME-DISASSOCIATE*****/
event error_t MLME_DISASSOCIATE.indication(uint32_t
DeviceAddress[], uint8_t DisassociateReason, bool
SecurityUse, uint8_t ACLEntry)
{
return SUCCESS;
}

event error_t MLME_DISASSOCIATE.confirm(uint8_t
status)
```

```
{
return SUCCESS;
}

/*****MCPS-DATA *****/
event error_t MCPS_DATA.confirm(uint8_t msduHandle,
uint8_t status)
{

return SUCCESS;
}
event error_t MCPS_DATA.indication(uint16_t
SrcAddrMode, uint16_t SrcPANId, uint32_t SrcAddr[2],
uint16_t DstAddrMode, uint16_t DestPANId, uint32_t
DstAddr[2], uint16_t msduLength,uint8_t
msdu[100],uint16_t mpduLinkQuality, uint16_t
SecurityUse, uint16_t ACLEntry)
{


return SUCCESS;
}
```

Una vez definido todo esto, se define un fichero de configuración(GTSManagementExample.nc) donde se indican todas las relaciones 802.15.4 y la implementación realizada:

```
#include <Timer.h>


#include "gtsmanagementexample.h"
#include "phy_const.h"
#include "phy_enumerations.h"
#include "mac_const.h"
#include "mac_enumerations.h"
#include "mac_func.h"


configuration GTSManagementExample {
}
implementation {

  components MainC;
  components LedsC;
  components GTSManagementExampleM;

  GTSManagementExampleM.Boot -> MainC;

  components Mac;

  GTSManagementExampleM.Leds -> LedsC;

  components new TimerMilliC() as Timer0;
  GTSManagementExampleM.Timer0 -> Timer0;

  components new TimerMilliC() as Timer_Send;
  GTSManagementExampleM.Timer_Send ->Timer_Send;
```

```
  //MAC interfaces

  GTSManagementExampleM.MLME_START -> Mac.MLME_START;

  GTSManagementExampleM.MLME_GET ->Mac.MLME_GET;
  GTSManagementExampleM.MLME_SET ->Mac.MLME_SET;

  GTSManagementExampleM.MLME_BEACON_NOTIFY -
>Mac.MLME_BEACON_NOTIFY;
  GTSManagementExampleM.MLME_GTS -> Mac.MLME_GTS;

  GTSManagementExampleM.MLME_ASSOCIATE-
>Mac.MLME_ASSOCIATE;
  GTSManagementExampleM.MLME_DISASSOCIATE-
>Mac.MLME_DISASSOCIATE;

  GTSManagementExampleM.MLME_ORPHAN->Mac.MLME_ORPHAN;
  GTSManagementExampleM.MLME_SYNC->Mac.MLME_SYNC;
  GTSManagementExampleM.MLME_SYNC_LOSS-
>Mac.MLME_SYNC_LOSS;
  GTSManagementExampleM.MLME_RESET->Mac.MLME_RESET;

  GTSManagementExampleM.MLME_SCAN->Mac.MLME_SCAN;


  GTSManagementExampleM.MCPS_DATA->Mac.MCPS_DATA;


}
```

El fichero gtsmanagementexample.h contiene información sobre la trama MAC, de los *beacons* y el identificador del *PANCoordinator*. Este fichero es común para los dos extremos.

```
enum {
      COORDINATOR = 0x00,
      ROUTER =0x01,
      END_DEVICE = 0x02
      };

#define BEACON_ORDER 6
#define SUPERFRAME_ORDER 4
#define LOGICAL_CHANNEL 0x15


#define TYPE_DEVICE END_DEVICE

//PAN VARIABLES
#define MAC_PANID 0x1234
```

Una vez implementado todo esto, se compila el código y se carga en el dispositivo MicaZ empleando la siguiente sentencia en la consola de XubunTOS:

```
      make micaz install.2 eprb,10.1.1.12
```

Capítulo 4 – Análisis de las características de una red 802.15.4

En este Capitulo, se tratan los distintos escenarios de trabajo desarrollados en este proyecto final de carrera, y sus resultados. Dichos escenarios, se desarrollan a partir de los escenarios mostrados en el apartado 2.5 del capitulo 3. No se va a explicar el código de la implementación de cada escenario. Solo se expone el escenario, sus resultados y las conclusiones. Para saber como ha sido implementado cada uno de los escenarios, hay que acudir a los anexos de la documentación de este proyecto fin de carrera.

Todos los escenarios han sido analizados con la herramienta CC2430DK de Texas Instruments. Dicha herramienta consiste en un Kit de desarrollo y análisis de redes 802.15.4, que entre otras particularidades permite trabajar como packet sniffer y capturar trafico de la red 802.15.4.

1.- Escenario de trabajo 1

1.1-Funcionamiento

El primer escenario de trabajo se emplea para analizar las características de transmisión de datos entre dispositivos MicaZ. Consta de un dispositivo MIB600 al que se conecta un dispositivo MicaZ local y un ordenador y otro dispositivo MicaZ remoto situado a 10 metros de distancia del dispostivo MicaZ local. Los dispositivos MicaZ se configuran para que los parámetros Beacon_Order y Superframe_Order tengan valor 3 (Beacon_Order=Superframe_Order=0.1228 segundos), de forma que el sistema va a transmitir paquetes lo mas rápido posible, lo que implica que se pueda obtener la velocidad de transmisión máxima. Para obtener dicha velocidad, se desactiva el uso de ACKs, lo que implica un aumento de la tasa de transmisión. El funcionamiento del sistema es el siguiente:

1. El dispositivo remoto envía continuamente paquetes 802.15.4 con destino el dispositivo MicaZ local. Nada más terminar de enviar un paquete, comienza la transmisión del siguiente. Los paquetes son identificados con un número de secuencia, el cual va introducido en el campo DATOS de la trama 802.15.4, y que actualiza después de cada transmisión el extremo remoto. El número de secuencia va desde 0 hasta 255.

2. El dispositivo MicaZ local recibe el paquete enviado desde el extremo remoto, lo desencapsula y envía al dispositivo MIB600 el contenido del campo DATOS.

3. El dispositivo MIB600 se comunica con el ordenador a través de una conexión Ethernet y le envía el paquete recibido.

4. El ordenador recibe el paquete, lo procesa y calcula las estadísticas deseadas.

En la siguiente figura se muestra el funcionamiento de este escenario.

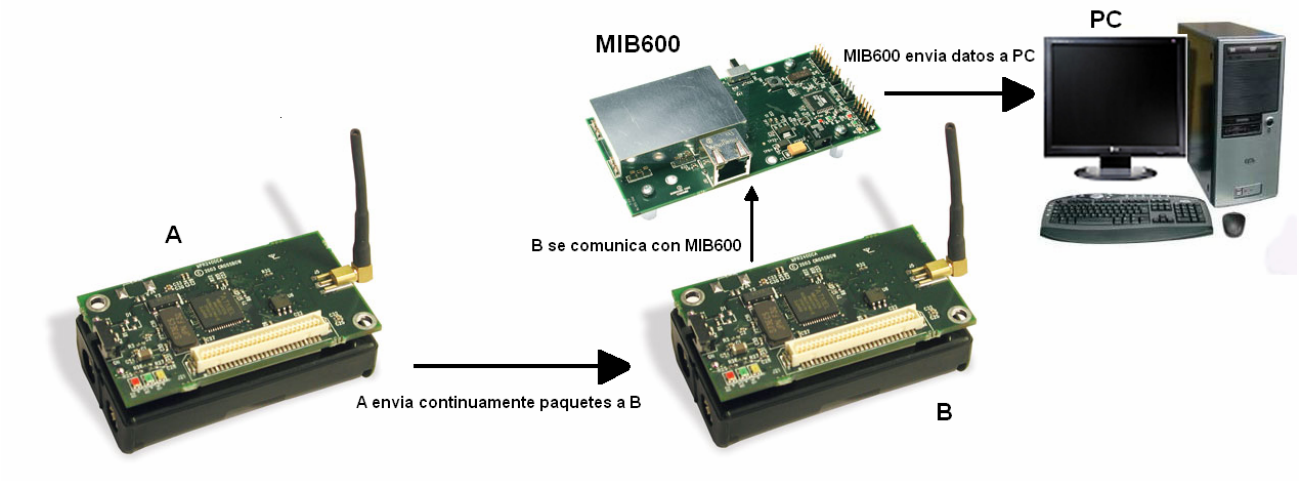

Fig. 31 – Funcionamiento del escenario 1

1.2.- Resultados

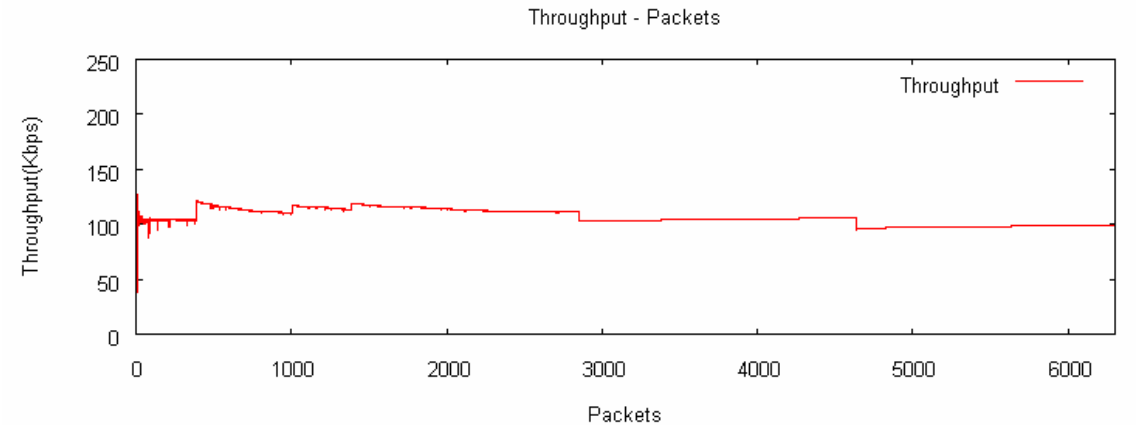

Fig. 32 – Throughput escenario1

Time (us)	Length	Frame control field								Sequence number	Dest. PAN	Dest. Address	Source PAN	Source Address	MAC payload	LQI	FCS
		Type	Sec	Pnd	Ack req	Intra PAN											
+1769	15	BCN	0	0	0	1				0x45	0x1234	0xFFFF	0x0000	Superframe specification: BO 03 SO 03 F.CAP 15 BLE 0 Coord 1 Assoc 1	GTS fields: Len 0 Permit 1	128	OK
=1070529																	
+3774	14	DATA	0	0	0	0				0x94	0x1234	0x0000	0x1234	0x0000	CA	112	OK
=1074303																	
+2210	14	DATA	0	0	0	0				0x95	0x1234	0x0000	0x1234	0x0000	CF	112	OK
=1076513																	
+2210	14	DATA	0	0	0	0				0x96	0x1234	0x0000	0x1234	0x0000	D4	112	OK
=1078723																	
+2357	14	DATA	0	0	0	0				0x97	0x1234	0x0000	0x1234	0x0000	DE	112	OK
=1081080																	
+2210	14	DATA	0	0	0	0				0x98	0x1234	0x0000	0x1234	0x0000	E3	120	OK
=1083290																	
+2212	14	DATA	0	0	0	0				0x99	0x1234	0x0000	0x1234	0x0000	E8	108	OK
=1085502																	
+2660	14	DATA	0	0	0	0				0x9A	0x1234	0x0000	0x1234	0x0000	EE	112	OK
=1088162																	
+2210	14	DATA	0	0	0	0				0x9B	0x1234	0x0000	0x1234	0x0000	F3	108	OK
=1090372																	
+2210	14	DATA	0	0	0	0				0x9C	0x1234	0x0000	0x1234	0x0000	F8	112	OK
=1092582																	
+2356	14	DATA	0	0	0	0				0x9D	0x1234	0x0000	0x1234	0x0000	FF	112	OK
=1094938																	
+2356	14	DATA	0	0	0	0				0x9E	0x1234	0x0000	0x1234	0x0000	04	112	OK
=1097294																	

Fig. 33 – Datos transmitidos escenario1

Throughput Medio (Kb/s)	*106,9195431*
Desviacion Típica	*2,430678897*

Tabla 5 - Estadísticas a partir de los datos obtenidos escenario 1

1.3.- Conclusiones

A la vista de los resultados, se puede afirmar, que el dispositivo MicaZ a máxima velocidad, funciona por debajo de los 250 Kbps cuando implementa el protocolo 802.15.4. Esto es debido a que dicho protocolo tiene un tiempo procesamiento, el cual influye negativamente en dicha tasa.

También cabe decir que la tasa de transferencia es constante con pequeñas variaciones a lo largo del tiempo. Dichas variaciones no son significativas y responden al propio procesamiento del protocolo 802.15.4 por parte de los dispositivos MicaZ.

Analizando las tramas obtenidas con el sniffer, se puede observar que entre beacon y beacon, se introducen en el canal una cantidad grande de datos.

Por último, es importante indicar que en recepción se pierden paquetes en el buffer del dispositivo MIB600, ya que no es capaz de procesar tan rápido el envío de paquetes al ordenador, por lo que se empiezan a descartar paquetes conforme se llena dicho buffer.

2.- Escenario de trabajo 2

2.1-Funcionamiento

El segundo escenario de trabajo se emplea para analizar las características de transmisión de datos entre dispositivos MicaZ. Consta de un dispositivo MIB600 al que se conecta un dispositivo MicaZ local y un ordenador y otro dispositivo MicaZ remoto situado a 10 metros de distancia del dispostivo MicaZ local. Los dispositivos MicaZ se configuran para que los parámetros Beacon_Order y Superframe_Order tengan valor 3 (Beacon_Order=Superframe_Order=0.12288 segundos), de forma que el sistema va a transmitir paquetes lo mas rápido posible, lo que implica que se pueda obtener la velocidad de transmisión máxima. Para obtener dicha velocidad, se desactiva el uso de ACKs, lo que implica un aumento de la tasa de transmisión. El funcionamiento del sistema es el siguiente:

1. En este caso el PC es el que envía continuamente al dispositivo local, paquetes con destino el dispositivo MicaZ remoto. Nada más terminar de enviar un paquete, comienza la transmisión del siguiente. Los paquetes son identificados con un número de secuencia. El número de secuencia va desde 0 hasta 255.

2. El dispositivo MIB600 local recibe el paquete enviado desde el ordenador a través de una conexión Ethernet y lo envía al dispositivo MicaZ local.

3. El dispositivo local recibe el paquete, lo encapsula en una trama 802.15.4 y lo envía al extremo remoto.

4. El extremo remoto recibe el paquete y lo reenvía al extremo local.

5. El extremo local recibe la respuesta y envía al dispositivo MIB600 el contenido del campo DATOS de la trama 802.15.4.

6. MIB600 responde al ordenador a través de una conexión Ethernet.

7. El ordenador recibe el paquete, lo procesa y calcula las estadísticas deseadas

En la siguiente figura se muestra el funcionamiento de este escenario.

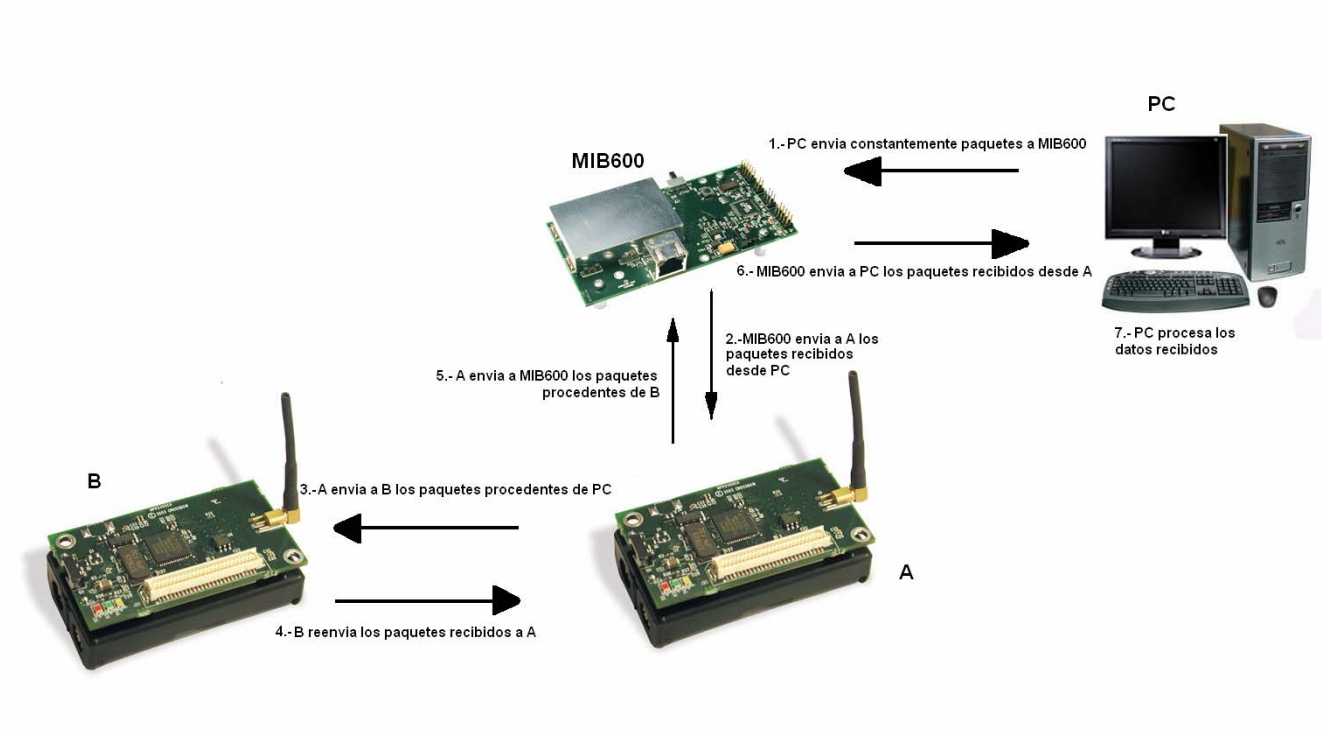

Fig. 34 – Funcionamiento del escenario 2

2.2.- Resultados

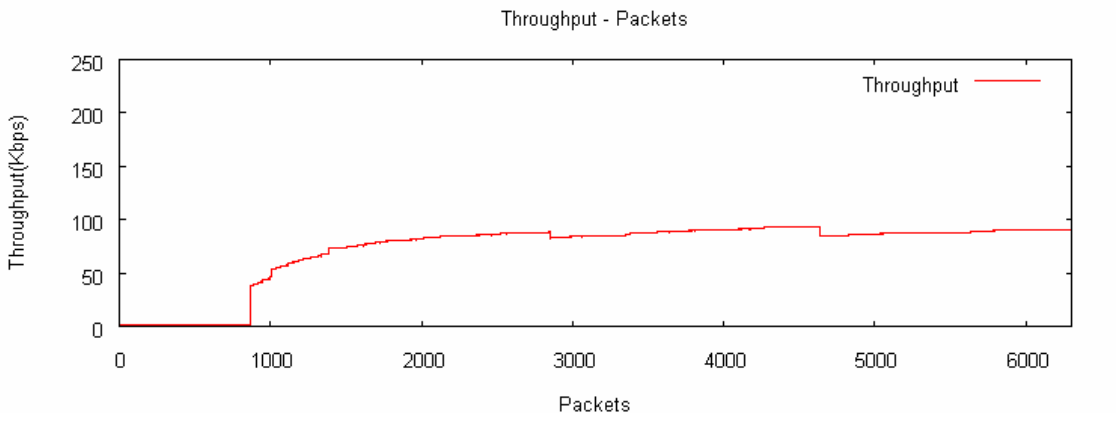

Fig. 35 – Throughput escenario 2

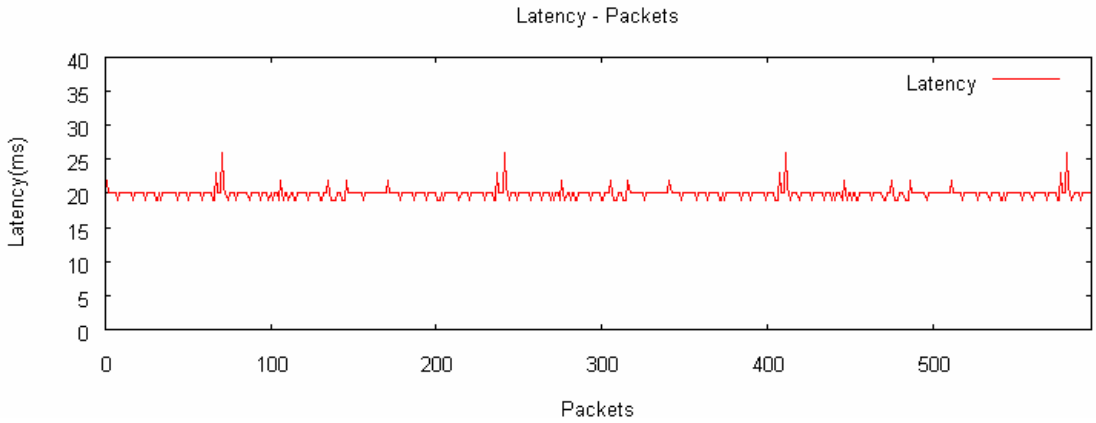

Fig. 36 - Retardo escenario 2

Time (us)	Length	Frame control field								Sequence number	Dest. PAN	Dest. Address	Source PAN	Source Address	Superframe specification							GTS fields		LQI	FCS
		Type	Sec	Pnd	Ack req		Intra PAN								BO	SO	F.CAP	BLE	Coord		Assoc	Len	Permit		
+18044 =444887	15	BCN	0	0	0		1			0x61	0x1234	0xFFFF	0x0000		03	03	15	0	1		1	0	1	44	OK
+21530 =466417	14	DATA	0	0	0		0			0xA3	0x1234	0x0002	0x1234	0x0000	MAC payload 56									44	OK
+40163 =506580	14	DATA	0	0	0		0			0xA4	0x1234	0x0002	0x1234	0x0000	MAC payload 57									44	OK
+39783 =546363	14	DATA	0	0	0		0			0xA5	0x1234	0x0002	0x1234	0x0000	MAC payload 58									40	OK
+40578 =586941	14	DATA	0	0	0		0			0xA6	0x1234	0x0002	0x1234	0x0000	MAC payload 59									44	OK
+8081 =595022	15	BCN	0	0	0		1			0x62	0x1234	0xFFFF	0x0000		03	03	15	0	1		1	0	1	40	OK
+31596 =626618	14	DATA	0	0	0		0			0xA7	0x1234	0x0002	0x1234	0x0000	MAC payload 5A									40	OK
+39816 =666434	14	DATA	0	0	0		0			0xA8	0x1234	0x0002	0x1234	0x0000	MAC payload 5B									40	OK
+40191 =706625	14	DATA	0	0	0		0			0xA9	0x1234	0x0002	0x1234	0x0000	MAC payload 5C									44	OK
+38129 =744754	15	BCN	0	0	0		1			0x63	0x1234	0xFFFF	0x0000		03	03	15	0	1		1	0	1	48	OK
+2285 =747039	14	DATA	0	0	0		0			0xAA	0x1234	0x0002	0x1234	0x0000	MAC payload 5D									48	OK
+39816 =786855	14	DATA	0	0	0		0			0xAB	0x1234	0x0002	0x1234	0x0000	MAC payload 5E									48	OK

Fig. 37 – Datos transmitidos escenario 2

Throughput Medio (Kb/s)	*72,7732009*
Throughput - Desviacion Típica	4,714746681
Retardo (ms)	19,98734177
Retardo - Desviacion Tipica(ms)	5,264727963

Tabla 6 - Estadísticas a partir de los datos obtenidos escenario 2

2.3.- Conclusiones

En este escenario, se obtiene una tasa de transmisión media inferior a la del escenario 1. Esto es debido a que la transferencia de datos la inicia desde el ordenador y tiene que pasar por el dispositivo MIB600, los cuales son el cuello de botella de la red. Esto se traduce en unas características de transmisión inferiores a las del escenario 1, donde no intervenía el ordenador nada más que para la recepción de datos. También se observa que la desviación típica aumenta respecto al escenario 1. Se puede afirmar entonces que la tasa transferencia de paquetes entre ordenador y MIB600 es menor que la tasa de transferencia de paquetes entre dispositivos MicaZ, lo cual afecta a las capacidades de transmisión a nivel de aplicación.

La gráfica muestran como en este escenario se tarda un tiempo superior al del escenario 1 en conseguir una velocidad constante.

En las tramas capturadas con el sniffer, se observa como en este escenario se introducen menos datos que en el anterior entre beacon y beacon debido a la disminución de la tasa de transmisión.

Cabe destacar que en este escenario no se pierden paquetes debido al buffer del dispositivo MIB600 y el ordenador, ya que al iniciar la transferencia de datos el ordenador, el flujo de datos lo regula éste, así que la tasa depende directamente de la capacidad de transmisión-recepción del bloque ordenador-MIB600.

3.- Escenario de trabajo 3

3.1-Funcionamiento

El tercer escenario de trabajo se emplea para analizar las características de transmisión de datos entre dispositivos MicaZ. Consta de un dispositivo MIB600 al que se conecta un dispositivo MicaZ local y un ordenador y otro dispositivo MicaZ remoto situado a 10 metros de distancia del dispostivo MicaZ local. Los dispositivos MicaZ se configuran para que los parámetros Beacon_Order tenga valor 6 y Superframe_Order tenga valor 4 (Beacon_Order=0.24576 segundos y Superframe_Order=0.16384 segundos), de forma que el sistema va a transmitir paquetes más despacio que en los dos primeros escenarios, ya que en este escenario lo importante es el cálculo del retardo entre paquetes. En este escenario se habilita el uso de ACKs. El funcionamiento del sistema es el siguiente:

1. En este caso el ordenador envía paquetes con destino el extremo remoto. No envía un nuevo paquete hasta que no recibe una respuesta del extremo remoto. Por si se pierde un paquete, pasado un tiempo, si no se recibe respuesta desde el extremo remoto, se procede a enviar el siguiente paquete, ya que si, no el ordenador se quedaría parado esperando la respuesta y no seguiría el proceso de envío. Los paquetes

son identificados con un número de secuencia. El número de secuencia va desde 0 hasta 255.

2. El dispositivo MIB600 local recibe el paquete enviado desde el ordenador a través de una conexión Ethernet y lo envía al dispositivo MicaZ local.

3. El dispositivo local recibe el paquete, lo encapsula en una trama 802.15.4 y lo envía al extremo remoto.

4. El extremo remoto recibe el paquete y lo reenvía al extremo local .

5. El extremo local recibe la respuesta y envía al dispositivo MIB600 el contenido del campo DATOS de la trama 802.15.4.

6. MIB600 responde al ordenador a través de una conexión Ethernet.

7. El ordenador recibe el paquete, lo procesa, calcula las estadísticas deseadas y envía el siguiente paquete (vuelta a paso 1).

En la siguiente figura se muestra el funcionamiento de este escenario.

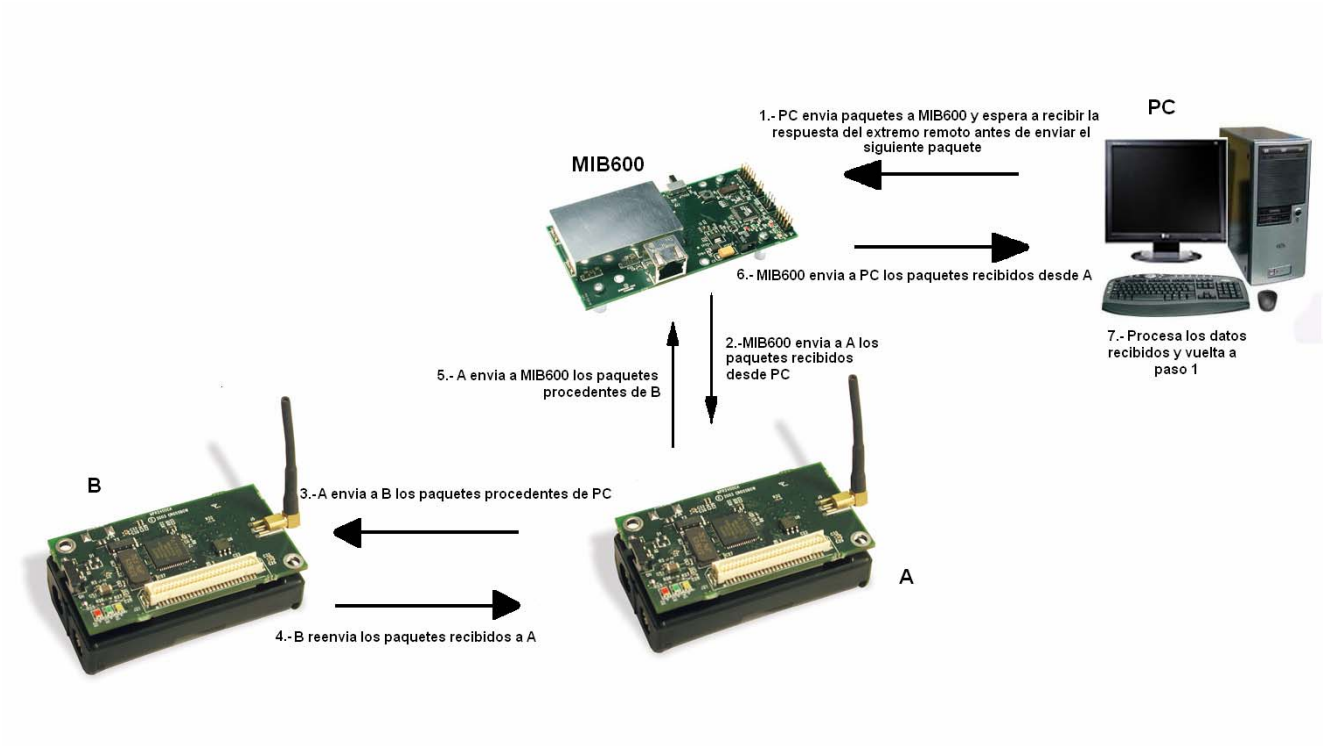

Fig. 38 – Funcionamiento del escenario 3

3.2.- Resultados

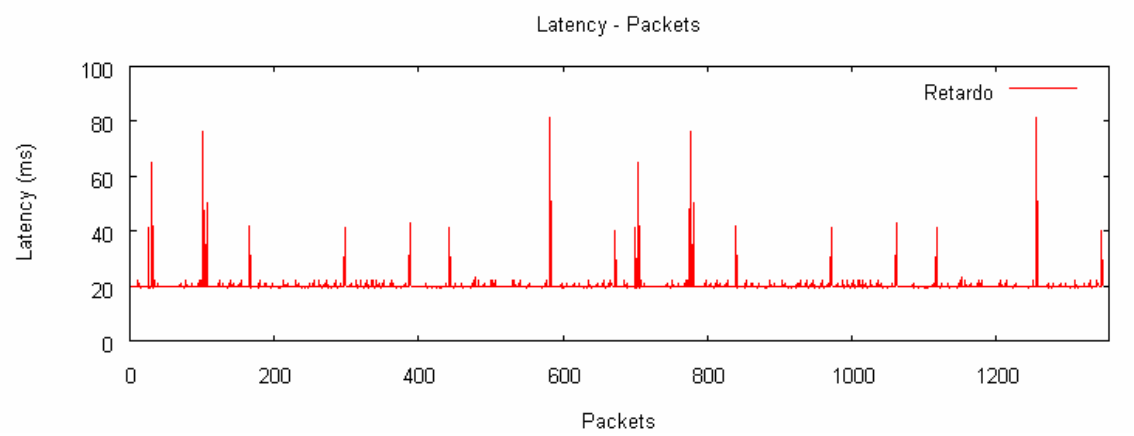

Fig. 39 – Retardo escenario 3

Time (us)	Length	Frame control field	Sequence number	Dest. PAN	Dest. Address	Source PAN	Source Address	MAC payload	LQI	FCS
+1822 =3665927	5	Type Sec Pnd Ack req Intra PAN: ACK 0 0 0 0	0x13						116	OK
+2185 =3668112	14	DATA 0 0 1 0	0xF4	0x1234	0x0000	0x1234	0x0000	21	116	OK
+1810 =3669922	5	ACK 0 0 0 0	0xF4						116	OK
+34408 =3704330	14	DATA 0 0 1 0	0x14	0x1234	0x0002	0x1234	0x0000	22	116	OK
+1814 =3706144	5	ACK 0 0 0 0	0x14						116	OK
+2166 =3708310	14	DATA 0 0 1 0	0xF5	0x1234	0x0000	0x1234	0x0000	22	116	OK
+1775 =3710085	5	ACK 0 0 0 0	0xF5						116	OK
+36707 =3746792	14	DATA 0 0 1 0	0x15	0x1234	0x0002	0x1234	0x0000	23	116	OK
+1794 =3748586	5	ACK 0 0 0 0	0x15						116	OK
+872541 =4621127	14	DATA 0 0 1 0	0xF6	0x1234	0x0000	0x1234	0x0000	23	116	OK
+1742 =4622869	5	ACK 0 0 0 0	0xF6						100	OK
+22550 =4645419	15	BCN 0 0 0 1	0xB3	0x1234	0xFFFF	0x0000	Superframe specification: BO SO F.CAP BLE Coord Assoc = 06 04 15 0 1 1	GTS fields: Len Permit = 0 1	100	OK

Fig. 40 – Datos transmitidos escenario 3

Throughput Medio (Kb/s)	1,022583077
Retardo Medio (ms)	20,76098418
Jitter (ms)	7,040922825

Tabla 7 - Estadísticas a partir de los datos obtenidos escenario 3

3.3.- Conclusiones

En este escenario se observa una tasa de transmisión muy inferior a la de los escenarios anteriores. Esto se debe a que por un lado se han modificado los parámetros Beacon_Order y Superframe_Order, y por otro, se espera a la recepción de un paquete antes de enviar el siguiente.

En los datos obtenidos, se puede observar como el retardo aumenta mucho en algunos puntos. Esto se debe a que se ha perdido algún paquete y pasado un determinado tiempo, se ha tenido que reiniciar desde nivel de aplicación la transmisión de datos. Estos retardos grandes son ignorados para la toma de estadísticas.

La captura de tramas del sniffer muestra tanto los datos transmitidos como el ACK de estos. Se puede observar también que al cambiar los parámetros de Beacon_Order y Superframe_Order, la cantidad de datos transmitidos entre beacon y beacon es superior a la del escenario 2 pese a tener un throughput inferior al de dicho escenario.

Se puede observar en las gráficas, que el retardo es más o menos constante con pequeñas variaciones provocadas por el propio procesamiento por parte del ordenador.

4.- Escenario de trabajo 4

4.1-Funcionamiento

El cuarto escenario de trabajo se emplea para analizar las características de transmisión de datos entre dispositivos MicaZ. Consta de un dispositivo MIB600 al que se conecta un dispositivo MicaZ local y un ordenador y otro dispositivo MicaZ remoto situado a 10 metros de distancia del dispostivo MicaZ local. Los dispositivos MicaZ se configuran para que los parámetros Beacon_Order tenga valor 6 y Superframe_Order tenga valor 4 (Beacon_Order=0.24576 segundos y Superframe_Order=0.16384 segundos), de forma que el sistema va a transmitir paquetes más despacio que en los dos primeros escenarios, ya que en este escenario lo importante es el cálculo del retardo entre paquetes. En este escenario se habilita el uso de ACKs. El funcionamiento del sistema es el siguiente:

1. En este caso el ordenador envía un paquete al dispositivo local, para indicar que está listo para comenzar a recibir paquetes. En este escenario, la única función del ordenador es recibir paquetes, calcular resultados estadísticos y reiniciar la comunicación pasado un tiempo, si no se recibe respuesta desde el extremo remoto para que en caso de perdida de algún paquete, el sistema no se quede parado. Para reiniciar la comunicación, se le envía un paquete al extremo local y al recibirlo, éste entiende que debe comenzar a enviar el siguiente paquete.

2. El dispositivo MIB600 local recibe el paquete enviado desde el ordenador a través de una conexión Ethernet y lo envía al dispositivo MicaZ local.

3. El dispositivo MicaZ local recibe el paquete, lo encapsula en una trama 802.15.4 y lo envía al extremo remoto. No envía un nuevo paquete hasta que no recibe una respuesta del extremo remoto. Los paquetes son identificados con un número de secuencia. El número de secuencia va desde 0 hasta 255.

4. El extremo remoto recibe el paquete y lo reenvía al extremo local .

5. El extremo local recibe la respuesta, envía al dispositivo MIB600 el contenido del campo DATOS de la trama 802.15.4 (paso 6) y comienza la transmisión del siguiente paquete (paso 3).

6. MIB600 responde al ordenador a través de la conexión Ethernet.

7. El ordenador recibe el paquete, lo procesa, calcula las estadísticas deseadas.

En la siguiente figura se muestra el funcionamiento de este escenario.

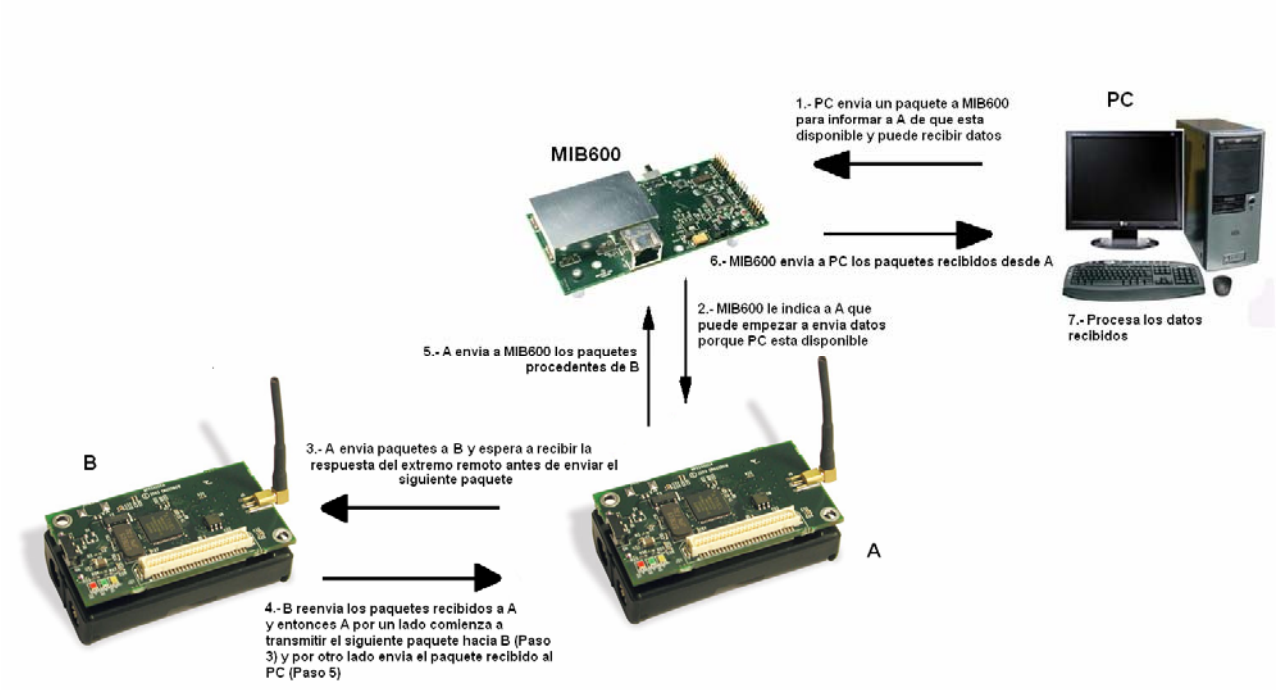

Fig. 41 – Funcionamiento del escenario 4

4.2.- Resultados

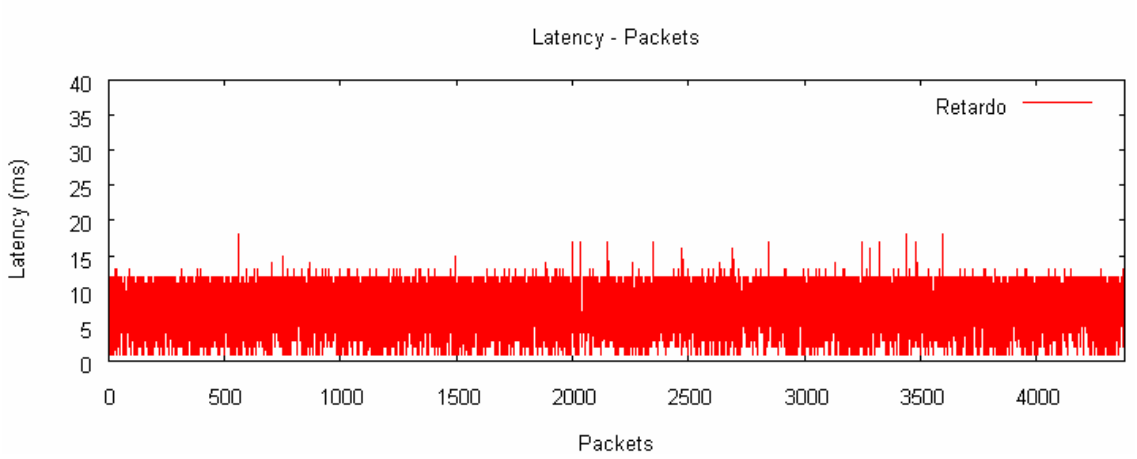

Fig. 42 – Retardo escenario 4

Time (us)	Length	Frame control field	Sequence number	Dest. PAN	Dest. Address	Source Address	Superframe specification	GTS fields	LQI	FCS
+1174463 =8303370	15	Type Sec Pnd Ack req Intra PAN: BCN 0 0 0 1	0xB1	0x1234	0xFFFF	0x0000	BO SO F.CAP BLE Coord Assoc: 06 04 15 0 1 1	Len Permit: 0 1	112	OK
+3082 =8306452	14	DATA 0 0 1 0	0x27	0x1234	0x0002	0x1234	0x0000	MAC payload: 34	112	OK
+2408 =8308860	5	ACK 0 0 0 0	0x27						116	OK
+2155 =8311015	14	DATA 0 0 1 0	0x09	0x1234	0x0000	0x1234	0x0000	MAC payload: 34	116	OK
+1785 =8312800	5	ACK 0 0 0 0	0x09						120	OK
+2614 =8315414	14	DATA 0 0 1 0	0x28	0x1234	0x0002	0x1234	0x0000	MAC payload: 35	112	OK
+1832 =8317246	5	ACK 0 0 0 0	0x28						116	OK
+2204 =8319450	14	DATA 0 0 1 0	0x0A	0x1234	0x0000	0x1234	0x0000	MAC payload: 35	116	OK
+1800 =8321250	5	ACK 0 0 0 0	0x0A						112	OK
+2600 =8323850	14	DATA 0 0 1 0	0x29	0x1234	0x0002	0x1234	0x0000	MAC payload: 36	112	OK
+1822 =8325672	5	ACK 0 0 0 0	0x29						116	OK
+2178 =8327850	14	DATA 0 0 1 0	0x0B	0x1234	0x0000	0x1234	0x0000	MAC payload: 36	116	OK

Fig. 43 – Datos transmitidos escenario 4

Throughput Medio (Kb/s)	5,646404269
Retardo Medio (ms)	4,536588235
Jitter (ms)	5,953634014

Tabla 8 - Estadísticas a partir de los datos obtenidos escenario 4

4.3.- Conclusiones

En este escenario se consigue una tasa de transferencia mayor y un retardo medio menor. Esto se debe a que el envío de paquetes lo gestiona el dispositivo MicaZ y no el ordenador, lo que se traduce en una mejora de dichos parámetros pese a tener un funcionamiento similar al del escenario 3.

También se consigue reducir la variación del retardo respecto al escenario anterior.

Se puede observar en las gráficas, que el retardo es más o menos constante con pequeñas variaciones provocadas por el propio procesamiento por parte del ordenador.

Se puede afirmar, por tanto, que el retardo introducido entre ordenador y MIB600 afecta al funcionamiento del sistema. Dicho retardo se reduce bastante si la comunicación la gestionase completamente el dispositivo MicaZ. El MicaZ introduce excesivo retardo en la comunicación con el PC.

La captura de tramas del sniffer muestra tanto los datos transmitidos como el ACK de éstos. Se puede observar también que al cambiar los parámetros de Beacon_Order y Superframe_Order, la cantidad de datos transmitidos entre beacon y beacon es superior a la del escenario 1 pese a tener un throughput inferior al de dicho escenario. También se comprueba que con valores de Beacon_Order y Superframe_Order iguales a los del escenario 3, si aumenta el throughput, la cantidad de datos transmitidos entre beacon y beacon es superior.

5.- Escenario de trabajo 5

5.1-Funcionamiento

El quinto escenario de trabajo se emplea para analizar las características de transmisión de datos entre dispositivos MicaZ. Consta de un dispositivo MIB600 al que se conecta un dispositivo MicaZ local y un ordenador, un dispositivo MicaZ remoto situado a 15 metros de distancia del dispostivo MicaZ local y un dispositivo MicaZ que hace de enlace entre el dispositivo local y el remoto situado entre ambos dispositivos. Los dispositivos MicaZ se configuran para que los parámetros Beacon_Order tenga valor 6 y Superframe_Order tenga valor 4 (Beacon_Order=0.24576 segundos y Superframe_Order=0.16384 segundos), de forma que el sistema va a transmitir paquetes más despacio que en los dos primeros escenarios, ya que en este escenario lo importante es el cálculo del retardo entre paquetes. En este escenario se habilita el uso de ACKs. El funcionamiento del sistema es el siguiente:

1. En este caso el ordenador envía paquetes con destino el extremo remoto. No envía un nuevo paquete hasta que no recibe una respuesta del extremo remoto. Por si se pierde un paquete, pasado un tiempo, si no se recibe respuesta desde el extremo remoto, se procede a enviar el siguiente paquete, ya que si no el ordenador se quedaría parado

esperando la respuesta y no seguiría el proceso de envío. Los paquetes son identificados con un número de secuencia. El número de secuencia va desde 0 hasta 255.

2. El dispositivo MIB600 local recibe el paquete enviado desde el ordenador a través de una conexión Ethernet y lo envía al dispositivo MicaZ local.

3. El dispositivo local recibe el paquete, lo encapsula en una trama 802.15.4 y lo envía al nodo de enlace.

4. En nodo de enlace recibe el paquete y lo envía al extremo remoto.

5. El extremo remoto recibe el paquete y lo reenvía al nodo de enlace para que lo mande hacia el extremo local.

6. En nodo de enlace recibe el paquete y lo envía al extremo local.

7. El extremo local recibe la respuesta y envía al dispositivo MIB600 el contenido del campo DATOS de la trama 802.15.4.

8. MIB600 responde al ordenador a través de una conexión Ethernet.

9. El ordenador recibe el paquete, lo procesa, calcula las estadísticas deseadas y envía el siguiente paquete (vuelta a paso 1).

En la siguiente figura se muestra el funcionamiento de este escenario.

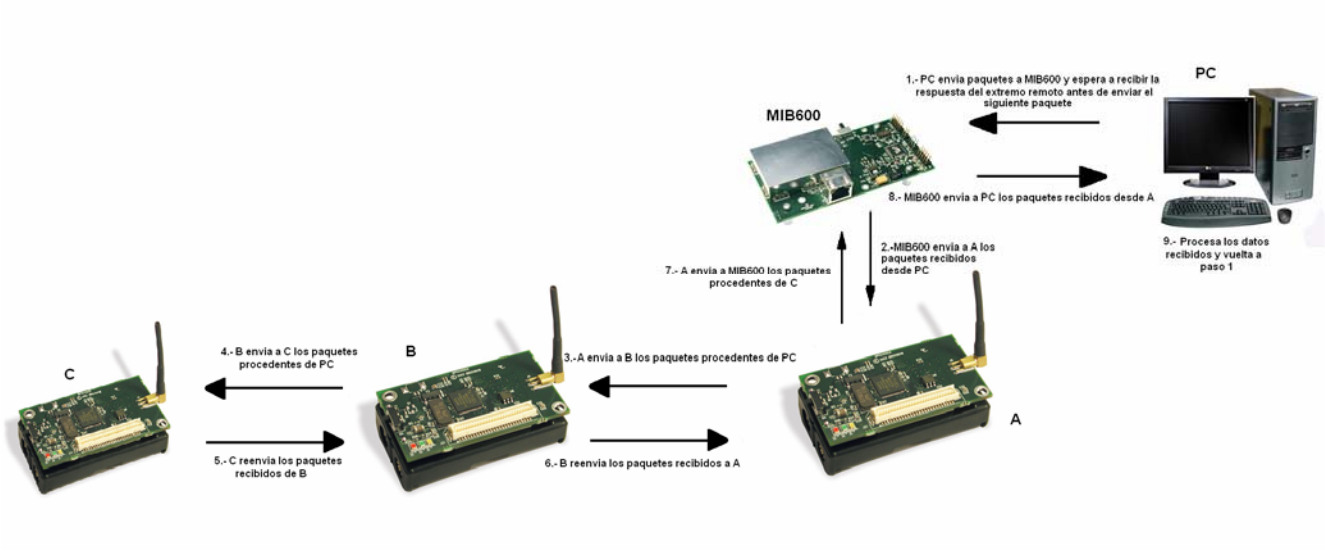

Fig. 44 – Funcionamiento del escenario 5

5.2.- Resultados

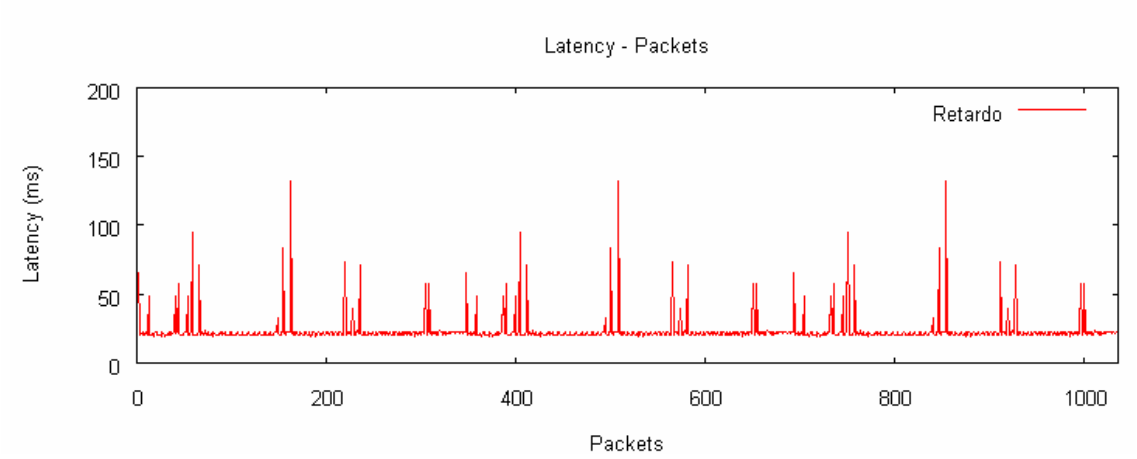

Fig. 45 – Retardo escenario 5

Time (us)	Length	Frame control field	Sequence number	Dest. PAN	Dest. Address	Source Address	Superframe specification	GTS fields	LQI	FCS
+26207 =133864524	15	Type Sec Pnd Ack req Intra PAN: BCN 0 0 0 1	0x19	0x1234	0xFFFF	0x0000	BO SO F.CAP BLE Coord Assoc: 06 04 15 0 1 1	Len Permit: 0 1	120	OK

Time (us)	Length	Frame control field	Sequence number	Dest. PAN	Dest. Address	Source PAN	Source Address	MAC payload	LQI	FCS
+2720 =133867244	14	Type Sec Pnd Ack req Intra PAN: DATA 0 0 1 0	0x0D	0x1234	0x0002	0x1234	0x0000	0C	128	OK
+2706 =133869950	5	Type Sec Pnd Ack req Intra PAN: ACK 0 0 0 0	0x0D						120	OK
+2225 =133872175	15	Type Sec Pnd Ack req Intra PAN: DATA 0 0 1 0	0xE3	0x1234	0x0000	0x1234	0x0002	0C 01	120	OK
+1868 =133874043	5	Type Sec Pnd Ack req Intra PAN: ACK 0 0 0 0	0xE3						120	OK
+2155 =133876198	14	Type Sec Pnd Ack req Intra PAN: DATA 0 0 1 0	0x0E	0x1234	0x0001	0x1234	0x0000	0C	128	OK
+1757 =133877955	5	Type Sec Pnd Ack req Intra PAN: ACK 0 0 0 0	0x0E						88	OK
+36618 =133914573	15	Type Sec Pnd Ack req Intra PAN: DATA 0 0 1 0	0xF4	0x1234	0x0000	0x1234	0x0001	0D 00	92	OK
+1868 =133916441	5	Type Sec Pnd Ack req Intra PAN: ACK 0 0 0 0	0xF4						124	OK
+2177 =133918618	14	Type Sec Pnd Ack req Intra PAN: DATA 0 0 1 0	0x0F	0x1234	0x0002	0x1234	0x0000	0D	116	OK
+1748 =133920366	5	Type Sec Pnd Ack req Intra PAN: ACK 0 0 0 0	0x0F						120	OK
+2213 =133922579	15	Type Sec Pnd Ack req Intra PAN: DATA 0 0 1 0	0xE4	0x1234	0x0000	0x1234	0x0002	0D 01	120	OK

Fig. 46 – Datos transmitidos escenario 5

Throughput Medio (Kb/s)	1,484902443
Retardo Medio (ms)	23,1468254
Jitter (ms)	7,966947173

Tabla 9 - Estadísticas a partir de los datos obtenidos escenario 5

5.3.- Conclusiones

Se puede observar que la tasa de transferencia es similar a la del escenario 3. Esto se debe a que esta tasa depende de la capacidad de recepción y transmisión del ordenador.

También se observa que el retardo medio aumenta. Esto se debe a que este escenario, los paquetes atraviesan un nodo más. Si el ordenador no introdujese tanto retardo, este retardo medio debería ser aproximadamente el doble del retardo del escenario 3, ya que se supone que al introducir un nodo intermedio, el paquete debe retransmitirse hasta que llegase al extremo remoto y esa retransmisión introduce un retardo igual al introducido para llegar al nodo intermedio. El problema está en que el ordenador es la mayor fuente de retardo del sistema, lo que provoca que el aumento de retardo por introducir un nodo intermedio no sea tan significativo respecto al que él mismo introduce.

Es importante observar que el Jitter es similar al del escenario 3, lo cual es lógico, puesto que la mayor parte de dicha variación la introduce el ordenador y el dispositivo MicaZ local.

Las gráficas de este escenario muestran un comportamiento similar a las del escenario 3, pero con un aumento del retardo. Esto se debe a la introducción de un tercer nodo.

En las tramas capturadas con el sniffer, se puede observar que aumenta el tráfico entre beacons respecto al escenario 3, debido a que se ha introducido un nodo más el cual también genera tráfico.

6.- Escenario de trabajo 6

6.1-Funcionamiento

El sexto escenario de trabajo se emplea para analizar las características de transmisión de datos entre dispositivos MicaZ. Consta de un dispositivo MIB600 al que se conecta un dispositivo MicaZ local y un ordenador, un dispositivo MicaZ remoto situado a 15 metros de distancia del dispostivo MicaZ local y un dispositivo MicaZ que hace de enlace entre el dispositivo local y el remoto situado entre ambos dispositivos. Los dispositivos MicaZ se configuran para que los parámetros Beacon_Order tenga valor 6 y Superframe_Order tenga valor 4 (Beacon_Order=0.24576 segundos y Superframe_Order=0.16384 segundos), de forma que el sistema va a transmitir paquetes más despacio que en los dos primeros escenarios, ya que en este escenario lo importante es el cálculo del retardo entre paquetes. En este escenario se habilita el uso de ACKs. El funcionamiento del sistema es el siguiente:

1. En este caso el ordenador envía un paquete al dispositivo local, para indicar que está listo para comenzar a recibir paquetes. En este escenario, la única función del ordenador es recibir paquetes, calcular resultados estadísticos y reiniciar la comunicación pasado un tiempo, si no se recibe respuesta desde el extremo remoto para que en caso de pérdida de algún

paquete, el sistema no se quede parado. Para reiniciar la comunicación, se le envía un paquete al extremo local y al recibirlo, éste entiende que debe comenzar a enviar el siguiente paquete.

2. El dispositivo MIB600 local recibe el paquete enviado desde el ordenador a través de una conexión Ethernet y lo envía al dispositivo MicaZ local.

3. El dispositivo MicaZ local recibe el paquete, lo encapsula en una trama 802.15.4 y lo envía al nodo de enlace. No envía un nuevo paquete hasta que no recibe una respuesta del extremo remoto. Los paquetes son identificados con un número de secuencia. El número de secuencia va desde 0 hasta 255.

4. El nodo de enlace recibe el paquete y lo envía al extremo remoto .

5. El extremo remoto recibe el paquete y lo reenvía al nodo de enlace para que lo mande hacia el extremo local.

6. El nodo de enlace recibe el paquete y lo reenvía al extremo local .

7. El extremo local recibe la respuesta, envía al dispositivo MIB600 el contenido del campo DATOS de la trama 802.15.4 (paso 8) y comienza la transmisión del siguiente paquete(paso 3).

8. MIB600 responde al ordenador a través de la conexión Ethernet.

9. El ordenador recibe el paquete, lo procesa, calcula las estadísticas deseadas.

En la siguiente figura se muestra el funcionamiento de este escenario.

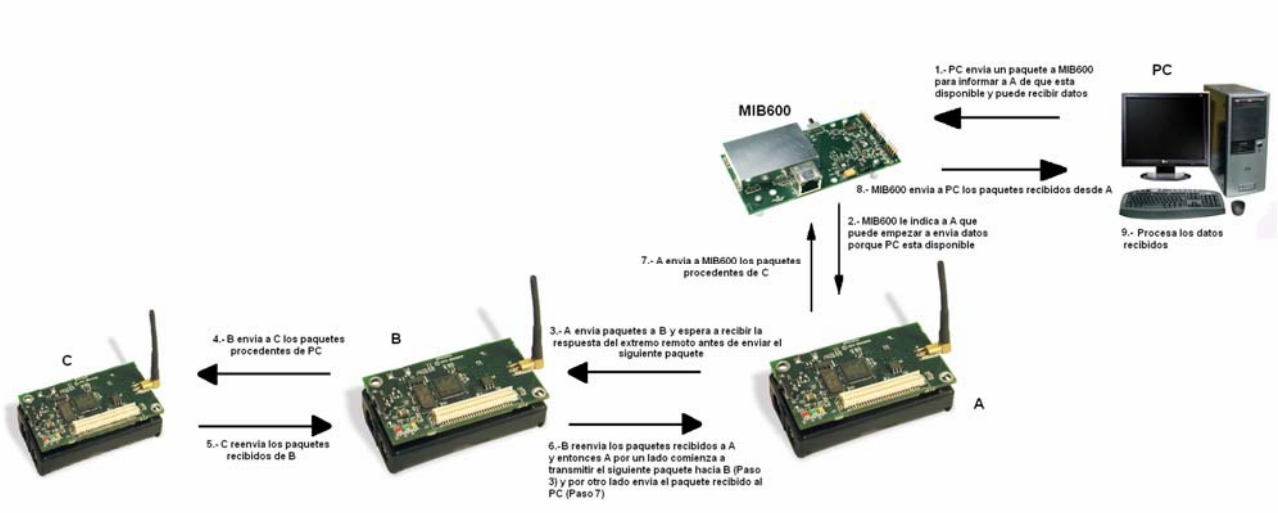

Fig. 47 – Funcionamiento del escenario 6

6.2.- Resultados

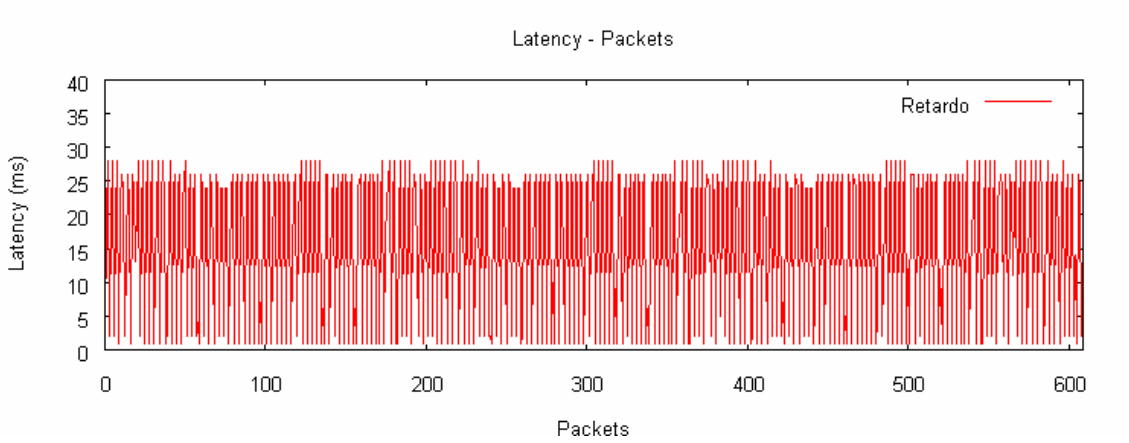

Fig. 48 – Retardo escenario 6

Time (us)	Length	Frame control field	Sequence number	Dest. PAN	Dest. Address	Source PAN	Source Address	Superframe specification	GTS fields	LQI	FCS
+12651 =135051862	15	Type Sec Pnd Ack req Intra PAN BCN 0 0 0 1	0x1A	0x1234	0xFFFF	0x0000		BO SO F.CAP BLE Coord Assoc 06 04 15 0 1 1	Len Permit 0 1	124	OK
+2720 =135054582	14	Type Sec Pnd Ack req Intra PAN DATA 0 0 1 0	0x15	0x1234	0x0002	0x1234	0x0000	MAC payload 0F		124	OK
+2686 =135057268	5	Type Sec Pnd Ack req Intra PAN ACK 0 0 0 0	0x15							116	OK
+2201 =135059469	15	Type Sec Pnd Ack req Intra PAN DATA 0 0 1 0	0xE7	0x1234	0x0000	0x1234	0x0002	MAC payload 0F 01		124	OK
+1867 =135061336	5	Type Sec Pnd Ack req Intra PAN ACK 0 0 0 0	0xE7							132	OK
+2164 =135063500	14	Type Sec Pnd Ack req Intra PAN DATA 0 0 1 0	0x16	0x1234	0x0001	0x1234	0x0000	MAC payload 0F		132	OK
+1758 =135065258	5	Type Sec Pnd Ack req Intra PAN ACK 0 0 0 0	0x16							76	OK
+16407 =135081665	15	Type Sec Pnd Ack req Intra PAN DATA 0 0 1 0	0xF9	0x1234	0x0000	0x1234	0x0001	MAC payload 11 00		76	OK
+1868 =135083533	5	Type Sec Pnd Ack req Intra PAN ACK 0 0 0 0	0xF9							132	OK
+2184 =135085717	14	Type Sec Pnd Ack req Intra PAN DATA 0 0 1 0	0x17	0x1234	0x0002	0x1234	0x0000	MAC payload 11		120	OK
+1748 =135087465	5	Type Sec Pnd Ack req Intra PAN ACK 0 0 0 0	0x17							124	OK
+2204 =135089669	15	Type Sec Pnd Ack req Intra PAN DATA 0 0 1 0	0xE8	0x1234	0x0000	0x1234	0x0002	MAC payload 11 01		116	OK

Fig. 49 – Datos transmitidos escenario 6

Throughput Medio (Kb/s)	4,719595609
Retardo Medio (ms)	8,150289017
Jitter (ms)	5,423923191

Tabla 10 - Estadísticas a partir de los datos obtenidos escenario 6

6.3.- Conclusiones

Se observa que la tasa de transferencia es similar a la del escenario 4 y superior a la del escenario 5. Esto se debe a que esta tasa depende de la capacidad de recepción y transmisión de los dispositivos MicaZ y no del ordenador. En dicha capacidad de recepción y transmisión influye el ancho de banda disponible por parte del dispositivo MicaZ, así como el ancho de banda disponible entre el ordenador y el dispositivo MIB600 en la entrega de paquetes al ordenador.

El retardo en este escenario es casi el doble que el del escenario 4, lo cual concuerda con la suposición expuesta en el escenario 5 de que el retardo introducido por los dispositivos MicaZ es inferior que el retardo introducido por el ordenador, y al introducir un nodo más, dicho retardo se duplica. Al no influir en el funcionamiento de este escenario el ordenador, sino que solo influyen los dispositivos MicaZ, dicho retardo se ajusta más al supuesto en el escenario 5.

Es importante observar que el Jitter es similar al del escenario 4, lo cual es lógico, puesto que la mayor parte de dicha variación la introduce el dispositivo MicaZ local.

Las gráficas de este escenario muestran un comportamiento similar a las del escenario 4, pero con un aumento del retardo. Esto se debe a la introducción de un tercer nodo.

En las tramas capturadas con el sniffer, se puede observar que aumenta el tráfico entre beacons respecto al escenario 4, debido a que se ha introducido un nodo más el cual también genera tráfico.

7.- Escenario de trabajo 7

7.1-Funcionamiento

El séptimo escenario de trabajo se emplea para analizar el mecanismo de asociación y reconexión entre dispositivos MicaZ. Consta de un dispositivo MIB600 al que se conecta un dispositivo MicaZ local (EndDevice), un ordenador y un dispositivo MicaZ remoto (Coordinador) situado a 10 metros del dispositivo MicaZ local. El dispositivo MicaZ local le indica constantemente al ordenador, el mecanismo que está ejecutando (escaneo pasivo, mecanismo de orfandad o envío de datos), y en qué fase de mecanismo está. A partir de estos datos, el ordenador calcula estadísticas. En los anexos aparece el código de la implementación del sistema y los mensajes que le envía al ordenador. En este apartado no aparece descrito el envío de mensajes al ordenador. El funcionamiento del sistema es el siguiente:

1. El extremo EndDevice realiza un escaneo pasivo del medio para buscar el coordinador que le ofrezca las mejores condiciones de comunicación.

2. Una vez encontrado el “mejor” coordinador (el coordinador que puede ofrecer las mejores condiciones de comunicación), se asocia a éste.

3. Cuando consigue asociarse, comienzan a transmitirse paquetes de datos al coordinador. El coordinador al recibir dichos paquetes, simplemente los elimina y no genera ninguna respuesta al EndDevice.

4. En cierto momento de la transmisión de datos, EndDevice pierde la comunicación con el coordinador (ya sea porque se ha alejado mucho el coordinador o porque se haya apagado). Ahora EndDevice trata de reconectarse al coordinador a través del mecanismo de orfandad. Si consigue reconectarse, simplemente seguirá enviándole paquetes de datos. En caso contrario, se volvería al paso 1.

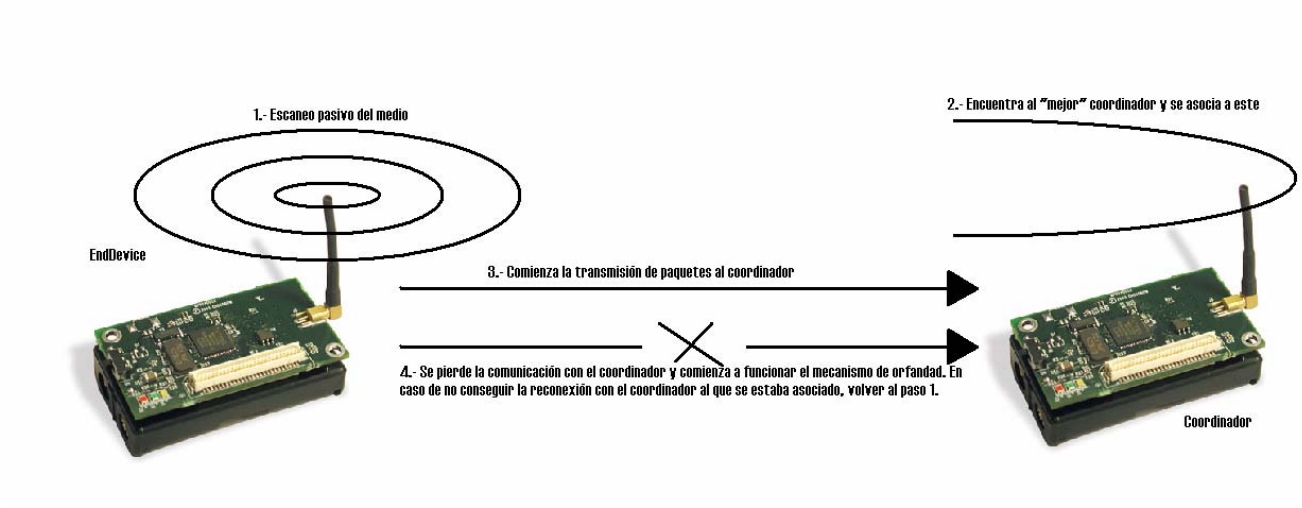

Fig. 50 – Funcionamiento del escenario 7

7.2.- Resultados

El extremo local produce los siguientes resultados al nivel de aplicación usando la aplicación TestSerial:

1.- Se inicia el escaneo pasivo del medio. Hay un único coordinador en el medio.

```
PASSIVE_SCAN
PASSIVE->T_ScanDuration.fired()1 - Tiempo:2071533000
PASSIVE->T_ScanDuration.fired()2 - Tiempo:4169810000
PASSIVE->T_ScanDuration.fired()3 - Tiempo:6270708000
PASSIVE->T_ScanDuration.fired()4 - Tiempo:8370318000
PASSIVE->T_ScanDuration.fired()5 - Tiempo:10470260000
PASSIVE->T_ScanDuration.fired()6 - Tiempo:12569385000
PASSIVE->T_ScanDuration.fired()7 - Tiempo:14670502000
PASSIVE->T_ScanDuration.fired()8 - Tiempo:16768988000
PASSIVE->T_ScanDuration.fired()9 - Tiempo:18870016000
PASSIVE->T_ScanDuration.fired()10 - Tiempo:20969540000
PASSIVE->T_ScanDuration.fired()11 - Tiempo:23069634000
PASSIVE->T_ScanDuration.fired()12 - Tiempo:25168892000
PASSIVE->T_ScanDuration.fired()13 - Tiempo:27286043000
PASSIVE->T_ScanDuration.fired()14 - Tiempo:29385948000
```

```
PASSIVE->T_ScanDuration.fired()15 - Tiempo:31487683000
PASSIVE->MLME_SCAN.confirm
MLME_ASSOCIATE.request
MLME_ASSOCIATE.confirm - Retardo:37991546000
```

2.- Comienza la transmisión de datos hacia el coordinador

```
MCPS_DATA.request
MCPS_DATA.request
MCPS_DATA.request
```

3.- Se pierde la comunicación con el coordinador y arranca el mecanismo de orfandad

```
MLME_SYNC_LOSS.indication->ORPHAN
ORPHAN->T_ScanDuration.fired() 1 - Tiempo:3927277000
ORPHAN->T_ScanDuration.fired() 2 - Tiempo:7815462000
ORPHAN->T_ScanDuration.fired() 3 - Tiempo:11742764000
ORPHAN->T_ScanDuration.fired() 4 - Tiempo:15646863000
ORPHAN->T_ScanDuration.fired() 5 - Tiempo:19556746000
ORPHAN->T_ScanDuration.fired() 6 - Tiempo:23461623000
ORPHAN->T_ScanDuration.fired() 7 - Tiempo:27365595000
ORPHAN->T_ScanDuration.fired() 8 - Tiempo:31253921000
ORPHAN->T_ScanDuration.fired() 9 - Tiempo:35181246000
ORPHAN->T_ScanDuration.fired() 10 - Tiempo:39085192000
ORPHAN->T_ScanDuration.fired() 11 - Tiempo:42972102000
process_coordinator_realignment->SENDDATA-
Retardo:43474398000
```

4.- En el caso de que mediante el mecanismo orfandad no consiguiese reconectarse, se inicia el mecanismo de escaneo pasivo para buscar un nuevo coordinador

```
MLME_SYNC_LOSS.indication->ORPHAN
ORPHAN->T_ScanDuration.fired() 1 - Tiempo:3904904000
ORPHAN->T_ScanDuration.fired() 2 - Tiempo:7792434000
ORPHAN->T_ScanDuration.fired() 3 - Tiempo:11719790000
ORPHAN->T_ScanDuration.fired() 4 - Tiempo:15623899000
ORPHAN->T_ScanDuration.fired() 5 - Tiempo:19511621000
ORPHAN->T_ScanDuration.fired() 6 - Tiempo:23438556000
ORPHAN->T_ScanDuration.fired() 7 - Tiempo:27343302000
ORPHAN->T_ScanDuration.fired() 8 - Tiempo:31230980000
ORPHAN->T_ScanDuration.fired() 9 - Tiempo:35158303000
ORPHAN->T_ScanDuration.fired() 10 - Tiempo:39062870000
ORPHAN->T_ScanDuration.fired() 11 - Tiempo:42950137000
ORPHAN->T_ScanDuration.fired() 12 - Tiempo:46877376000
ORPHAN->T_ScanDuration.fired() 13 - Tiempo:50781408000
ORPHAN->T_ScanDuration.fired() 14 - Tiempo:54669776000
ORPHAN->T_ScanDuration.fired() 15 - Tiempo:58595845000
ORPHAN->MLME_SCAN.confirm->PASSIVE
PASSIVE->T_ScanDuration.fired()1 - Tiempo:64601312000
PASSIVE->T_ScanDuration.fired()2 - Tiempo:66700609000
PASSIVE->T_ScanDuration.fired()3 - Tiempo:68800428000
PASSIVE->T_ScanDuration.fired()4 - Tiempo:70900355000
PASSIVE->T_ScanDuration.fired()5 - Tiempo:73000133000
PASSIVE->T_ScanDuration.fired()6 - Tiempo:75099904000
PASSIVE->T_ScanDuration.fired()7 - Tiempo:77199832000
PASSIVE->T_ScanDuration.fired()8 - Tiempo:79299684000
PASSIVE->T_ScanDuration.fired()9 - Tiempo:81399644000
PASSIVE->T_ScanDuration.fired()10 - Tiempo:83499536000
```

```
PASSIVE->T_ScanDuration.fired()11 - Tiempo:85599402000
PASSIVE->T_ScanDuration.fired()12 - Tiempo:87699416000
PASSIVE->T_ScanDuration.fired()13 - Tiempo:89799317000
PASSIVE->T_ScanDuration.fired()14 - Tiempo:91898703000
PASSIVE->T_ScanDuration.fired()15 - Tiempo:93976865000
PASSIVE->MLME_SCAN.confirm
MLME_ASSOCIATE.request
MLME_ASSOCIATE.confirm - Retardo:101033664000
```

A continuación se muestran las tramas capturadas a partir del sniffer.

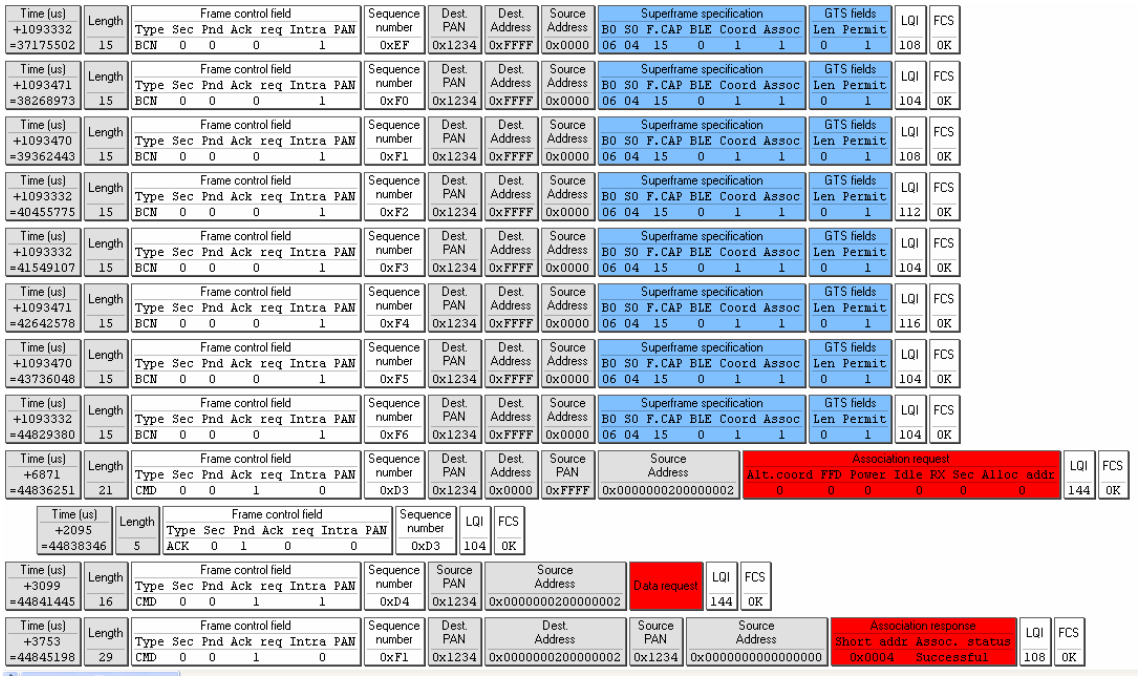

Fig. 51 – Finalización del escaneo pasivo y proceso de asociación escenario 7

Fig. 52 – Envío de datos al coordinador escenario 7

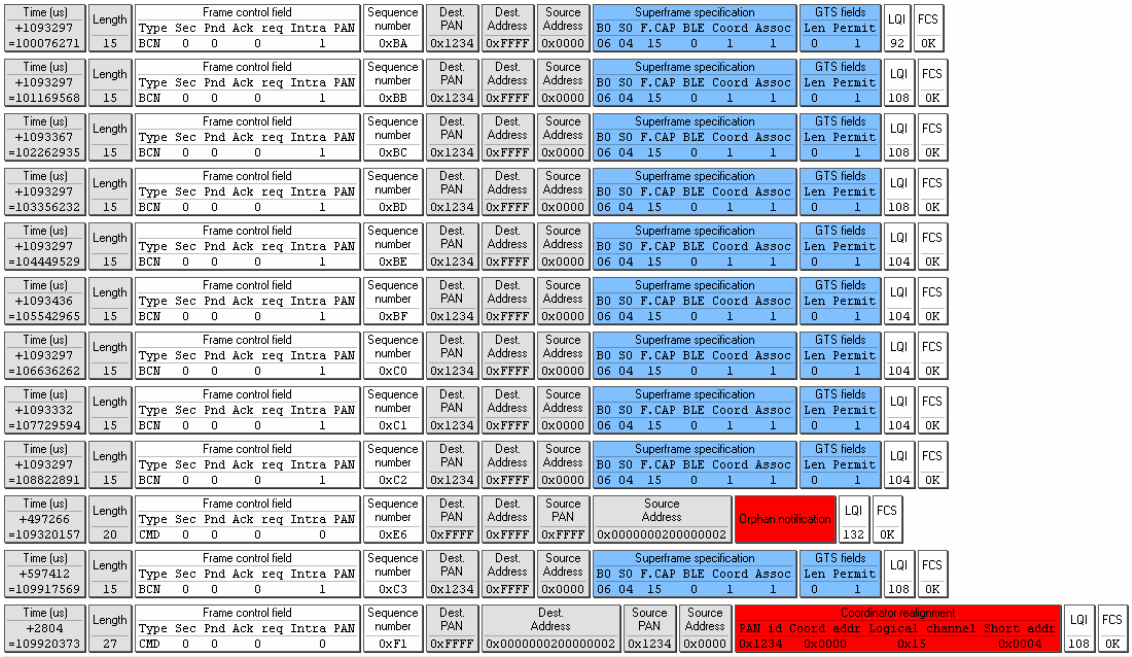

Fig. 53 – Finalización del mecanismo de orfandad y realineamiento con el coordinador escenario 7

Repitiendo este proceso varias veces, se obtienen los siguientes resultados:

Escaneo Pasivo al Inicio	Retardo (ms)
MLME_ASSOCIATE.confirm - Retardo Medio:	38234,98025
MLME_ASSOCIATE.confirm - Desviacion Tipica:	238,6607757
Tiempo de SCAN de canal en modo PASIVO	
PASSIVE->T_ScanDuration.fired() Medio:	2098,816333
PASSIVE->T_ScanDuration.fired() Desviacion:	7,261349222

Mecanismo de Orfandad	Retardo (ms)
process_coordinator_realignment->SENDDATA - Retardo Medio:	43836,204
process_coordinator_realignment->SENDDATA - Desviacion Tipica:	280,1602525
Tiempo de SCAN de canal en modo Orfandad	
ORPHAN->T_ScanDuration.fired() Medio:	3906,444764
ORPHAN->T_ScanDuration.fired() Desviacion:	16,11959317

Escaneo pasivo si no funciona la orfandad	
MLME_ASSOCIATE.confirm - Retardo Medio:	100965,5487
MLME_ASSOCIATE.confirm - Desviacion Tipica:	339,1523072

Tabla 11 - Estadísticas a partir de los datos obtenidos escenario 7

7.3.- Conclusiones

Se observa en las tramas analizadas por el sniffer y a nivel de aplicación el correcto funcionamiento del sistema. Se puede ver como en el proceso de asociación, ante una petición de asociación, se produce una respuesta de confirmación por parte del coordinador. Tras esto, comienza el envío de datos. También se puede ver como en el mecanismo de orfandad, cuando se trata de realinear el EndDevice con el coordinador, envía una notificación de orfandad, ante la cual se responde con una trama de Coordinator realignment. También se puede ver como en el mecanismo de orfandad, si no se consigue realinear con el anterior coordinador se inicia un escaneo pasivo del medio olvidándose de cual era el coordinador al que se estaba conectado.

A partir de los datos estadísticos obtenidos, se puede determinar que los procesos de búsqueda de canal y de reconexión son procesos lentos en los cuales la temporización es muy importante, por lo que las varianzas al final del proceso son mínimas en comparación a todo el tiempo necesario para la realización de dichos mecanismos. Pese a que se podría pensar que estas pequeñas variaciones temporales son causadas por la capacidad de procesamiento de MicaZ como ocurría en escenarios anteriores, realmente vienen propiciadas por el propio mecanismo de ejecución de eventos de TinyOs, los cuales dependen de la propia temporización de eventos de dicho sistema operativo.

También hay que destacar que el retardo producido por el mecanismo de orfandad depende del canal que nos encontremos, por lo que si se elige un canal distinto, dicho retardo variará.

8.- Escenario de trabajo 8

8.1-Funcionamiento

El octavo escenario de trabajo se emplea para analizar el mecanismo de asociación y reconexión entre dispositivos MicaZ y la capacidad de estos para cambiar de coordinador en el caso de desaparecer el anterior al que estaba conectado. Consta de un dispositivo MIB600 al que se conecta un dispositivo MicaZ local(EndDevice), un ordenador y dos dispositivos MicaZ remotos (Coordinador A y Coordinador B) situados a 10 metros del dispositivo MicaZ local. El dispositivo MicaZ local le indica constantemente al ordenador, el proceso que está ejecutando (escaneo pasivo, mecanismo de orfandad o envío de datos), y en qué fase de mecanismo está. A partir de estos datos, el ordenador calcula estadísticas. En los anexos aparece el código de la implementación del sistema y los mensajes que le envía al ordenador. En este apartado no aparece descrito el envío de mensajes al ordenador. El funcionamiento del sistema es el siguiente:

1. El extremo EndDevice realiza un escaneo pasivo del medio para buscar el coordinador que le ofrezca las mejores condiciones de comunicación.

2. Una vez encontrado el “mejor” coordinador (el coordinador que puede ofrecer las mejores condiciones de comunicación), se asocia a este.

3. Cuando consigue asociarse, comienzan a transmitirse paquetes de datos al coordinador. El coordinador al recibir dichos paquetes, simplemente los elimina y no genera ninguna respuesta al EndDevice.

4. En cierto momento de la transmisión de datos, EndDevice pierde la comunicación con el coordinador debido a que se apaga dicho coordinador. Ahora EndDevice trata de reconectarse al coordinador al que estaba conectado a través del mecanismo de orfandad.

5. No consigue reconectarse a dicho coordinador puesto que permanece apagado, por lo que se inicia otra vez un escaneo pasivo del medio para buscar el coordinador que le ofrezca las mejores condiciones de comunicación.

6. Logra entonces encontrar al otro coordinador y se asocia a éste.

7. Cuando consigue asociarse, comienzan a transmitirse paquetes de datos al coordinador. El coordinador al recibir dichos paquetes, simplemente los elimina y no genera ninguna respuesta al EndDevice.

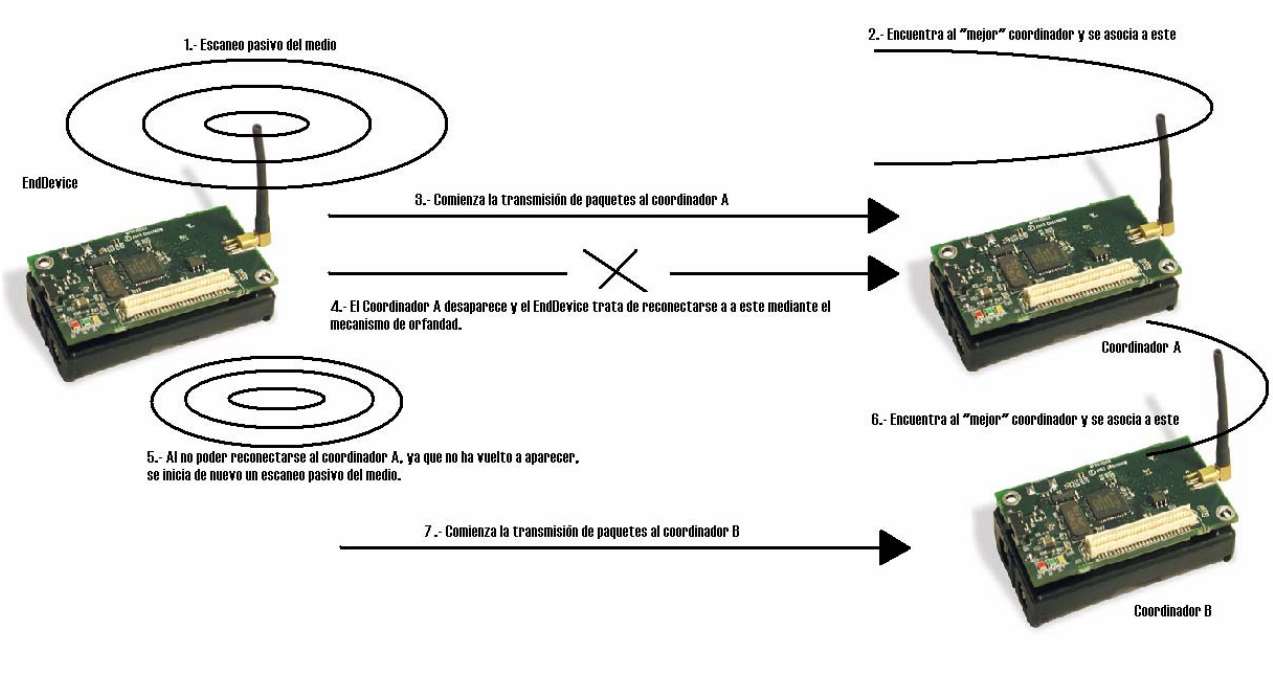

Fig. 54 – Funcionamiento del escenario 8

8.2.- Resultados

El extremo local produce los siguientes resultados al nivel de aplicación:

1.- Se inicia el escaneo pasivo del medio y se asocia al coordinador que mejores características de transmisión le ofrece. Hay dos coordinadores enviando beacons en distintos canales.

```
PASSIVE_SCAN
PASSIVE->T_ScanDuration.fired()1 - Tiempo:2096914000
PASSIVE->T_ScanDuration.fired()2 - Tiempo:4192807000
PASSIVE->T_ScanDuration.fired()3 - Tiempo:6292770000
PASSIVE->T_ScanDuration.fired()4 - Tiempo:8392773000
PASSIVE->T_ScanDuration.fired()5 - Tiempo:10468951000
PASSIVE->T_ScanDuration.fired()6 - Tiempo:12569566000
PASSIVE->T_ScanDuration.fired()7 - Tiempo:14668457000
PASSIVE->T_ScanDuration.fired()8 - Tiempo:16769075000
PASSIVE->T_ScanDuration.fired()9 - Tiempo:18868511000
PASSIVE->T_ScanDuration.fired()10 - Tiempo:20968517000
PASSIVE->T_ScanDuration.fired()11 - Tiempo:23067964000
PASSIVE->T_ScanDuration.fired()12 - Tiempo:25168683000
PASSIVE->T_ScanDuration.fired()13 - Tiempo:27268166000
PASSIVE->T_ScanDuration.fired()14 - Tiempo:29368082000
PASSIVE->T_ScanDuration.fired()15 - Tiempo:31468735000
PASSIVE->MLME_SCAN.confirm
MLME_ASSOCIATE.request
MLME_ASSOCIATE.confirm - Retardo:38500436000
```

2.- Comienza la transmisión de datos hacia el coordinador elegido.

```
MCPS_DATA.request
MCPS_DATA.request
MCPS_DATA.request
```

3.- Se pierde la comunicación con el coordinador y arranca el mecanismo de orfandad.

```
MCPS_DATA.request
MLME_SYNC_LOSS.indication->ORPHAN
ORPHAN->T_ScanDuration.fired() 1 - Tiempo:3927237000
ORPHAN->T_ScanDuration.fired() 2 - Tiempo:7831768000
ORPHAN->T_ScanDuration.fired() 3 - Tiempo:11718052000
ORPHAN->T_ScanDuration.fired() 4 - Tiempo:15646582000
ORPHAN->T_ScanDuration.fired() 5 - Tiempo:19550537000
ORPHAN->T_ScanDuration.fired() 6 - Tiempo:23437962000
ORPHAN->T_ScanDuration.fired() 7 - Tiempo:27342216000
ORPHAN->T_ScanDuration.fired() 8 - Tiempo:31269576000
ORPHAN->T_ScanDuration.fired() 9 - Tiempo:35157260000
ORPHAN->T_ScanDuration.fired() 10 - Tiempo:39061458000
ORPHAN->T_ScanDuration.fired() 11 - Tiempo:42988775000
ORPHAN->T_ScanDuration.fired() 12 - Tiempo:46876312000
ORPHAN->T_ScanDuration.fired() 13 - Tiempo:50780629000
ORPHAN->T_ScanDuration.fired() 14 - Tiempo:54707944000
ORPHAN->T_ScanDuration.fired() 15 - Tiempo:58595517000
```

4.- Al no poder reconectarse al coordinador al que estaba conectado, inicia un escaneo pasivo del medio. Al finalizar el escaneo pasivo del medio, se asocia a otro coordinador puesto que el anterior coordinador no ha vuelto a aparecer.

```
ORPHAN->MLME_SCAN.confirm->PASSIVE
PASSIVE->T_ScanDuration.fired()1 - Tiempo:64622024000
PASSIVE->T_ScanDuration.fired()2 - Tiempo:66722080000
PASSIVE->T_ScanDuration.fired()3 - Tiempo:68799634000
PASSIVE->T_ScanDuration.fired()4 - Tiempo:70899322000
PASSIVE->T_ScanDuration.fired()5 - Tiempo:72999229000
PASSIVE->T_ScanDuration.fired()6 - Tiempo:75098729000
PASSIVE->T_ScanDuration.fired()7 - Tiempo:77198729000
PASSIVE->T_ScanDuration.fired()8 - Tiempo:79298168000
PASSIVE->T_ScanDuration.fired()9 - Tiempo:81398890000
PASSIVE->T_ScanDuration.fired()10 - Tiempo:83498351000
PASSIVE->T_ScanDuration.fired()11 - Tiempo:85598239000
PASSIVE->T_ScanDuration.fired()12 - Tiempo:87698022000
PASSIVE->T_ScanDuration.fired()13 - Tiempo:89797488000
PASSIVE->T_ScanDuration.fired()14 - Tiempo:91898189000
PASSIVE->T_ScanDuration.fired()15 - Tiempo:93997683000
PASSIVE->MLME_SCAN.confirm
MLME_ASSOCIATE.request
MLME_ASSOCIATE.confirm - Retardo:100920764000
```

A continuación se muestran las tramas capturadas en el proceso de escaneo pasivo. Para la obtención de estas tramas, hay que indicarle al sniffer el canal que se quiere mostrar, ya que interesa mostrar los datos de los dos coordinadores y hay que tener en cuenta que cada coordinador trabaja en un canal distinto. Cuando pierde el primer coordinador y se reconecta al segundo, EndDevice cambia el canal de funcionamiento y se puede comprobar esto cambiando el canal de captura del sniffer.

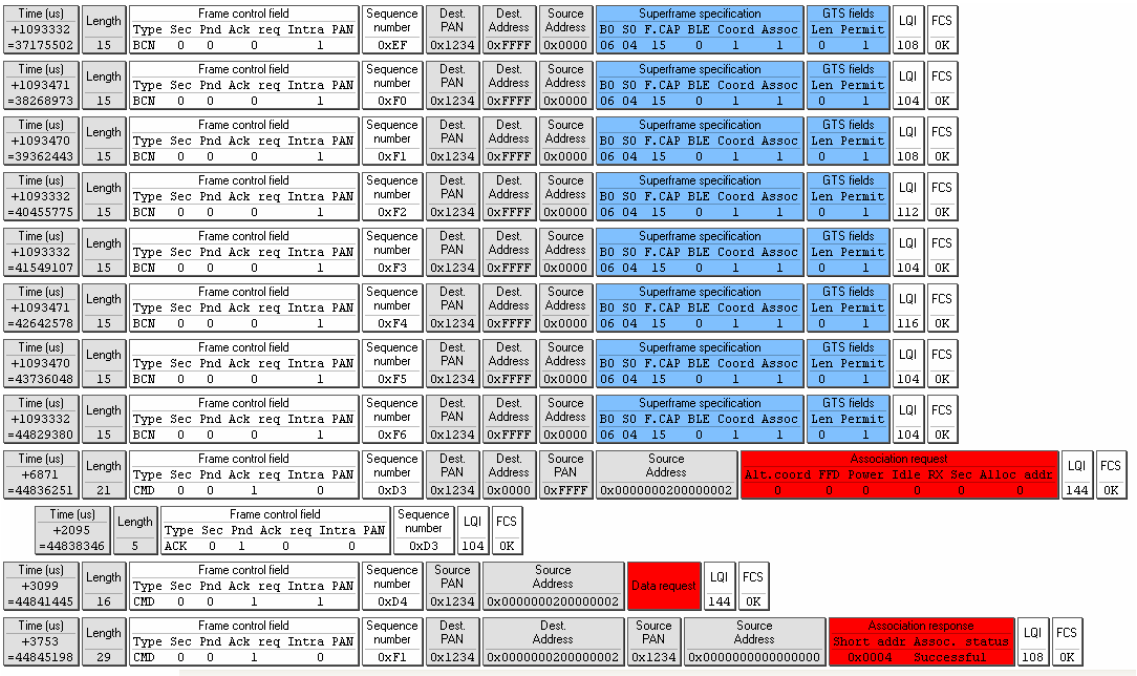

Fig. 55 – Finalización del escaneo pasivo y proceso de asociación escenario 8

Escaneo Pasivo	Retardo (ms)
MLME_ASSOCIATE.confirm - Retardo Medio:	101166,4248
MLME_ASSOCIATE.confirm - Desviacion Tipica:	204,5212481

Tabla 12 - Estadísticas a partir de los datos obtenidos escenario 8

8.3.- Conclusiones

Se observa en las tramas analizadas por el sniffer y a nivel de aplicación el correcto funcionamiento del sistema. Se puede ver como en el proceso de asociación, ante una petición de asociación, se produce una respuesta de confirmación por parte del coordinador. Tras esto, comienza el envío de datos. También se puede ver como tras fallar el mecanismo de orfandad, se inicia un escaneo pasivo del medio olvidándose de cual era el coordinador al que se estaba conectado, ya que tras desaparecer el primer coordinador, se reconecta al otro.

De los datos estadísticos obtenidos, se puede concluir que el tiempo empleado en un escaneo pasivo es independiente del canal al que se quiere conectar e independiente del canal al que estuviese anteriormente conectado, puesto que se obtienen resultados similares a los obtenidos en el escenario 7, donde la conexión se realiza al mismo canal al que estaba anteriormente conectado.

9.- Escenario de trabajo 9

9.1-Funcionamiento

El noveno escenario de trabajo se emplea para analizar el mecanismo de asociación y reconexión entre dispositivos MicaZ y su capacidad para conectarse al coordinador que mejores características de conexión ofrece. Consta de un dispositivo MIB600 al que se conecta un dispositivo MicaZ local (EndDevice), un ordenador y dos dispositivos MicaZ remotos (Coordinador A y Coordinador B) situados a 10 metros del dispositivo MicaZ local. El dispositivo MicaZ local le indica constantemente al ordenador el proceso que está ejecutando (escaneo pasivo, mecanismo de orfandad o envío de datos), y en qué fase de mecanismo está. A partir de estos datos, el ordenador calcula estadísticas. En los anexos aparece el código de la implementación del sistema y los mensajes que le envía al ordenador. En este apartado no aparece descrito el envío de mensajes al ordenador. El funcionamiento del sistema es el siguiente:

1. El extremo EndDevice realiza un escaneo pasivo del medio para buscar el coordinador que le ofrezca las mejores condiciones de comunicación.

2. Una vez encontrado el “mejor” coordinador (el coordinador que puede ofrecer las mejores condiciones de comunicación), se asocia a este.

3. Cuando consigue asociarse, comienzan a transmitirse paquetes de datos al coordinador. El coordinador al recibir dichos paquetes, simplemente los elimina y no genera ninguna respuesta al EndDevice.

4. En cierto momento de la transmisión de datos, EndDevice pierde la comunicación con el coordinador debido a que se apaga dicho coordinador. Ahora EndDevice trata de reconectarse al coordinador al que estaba conectado a través del mecanismo de orfandad.

5. No consigue reconectarse a dicho coordinador puesto que permanece apagado, por lo que se inicia otra vez un escaneo pasivo del medio para buscar el coordinador que le ofrezca las mejores condiciones de comunicación. Justo en el momento en el que va a empezar el escaneo pasivo, aparece otra vez el coordinador anterior pero a mucha mayor distancia que antes.

6. Mediante el escaneo pasivo, determina al otro coordinador como el “mejor” y se asocia a éste.

7. Cuando consigue asociarse, comienzan a transmitirse paquetes de datos al coordinador. El coordinador al recibir dichos paquetes, simplemente los elimina y no genera ninguna respuesta al EndDevice.

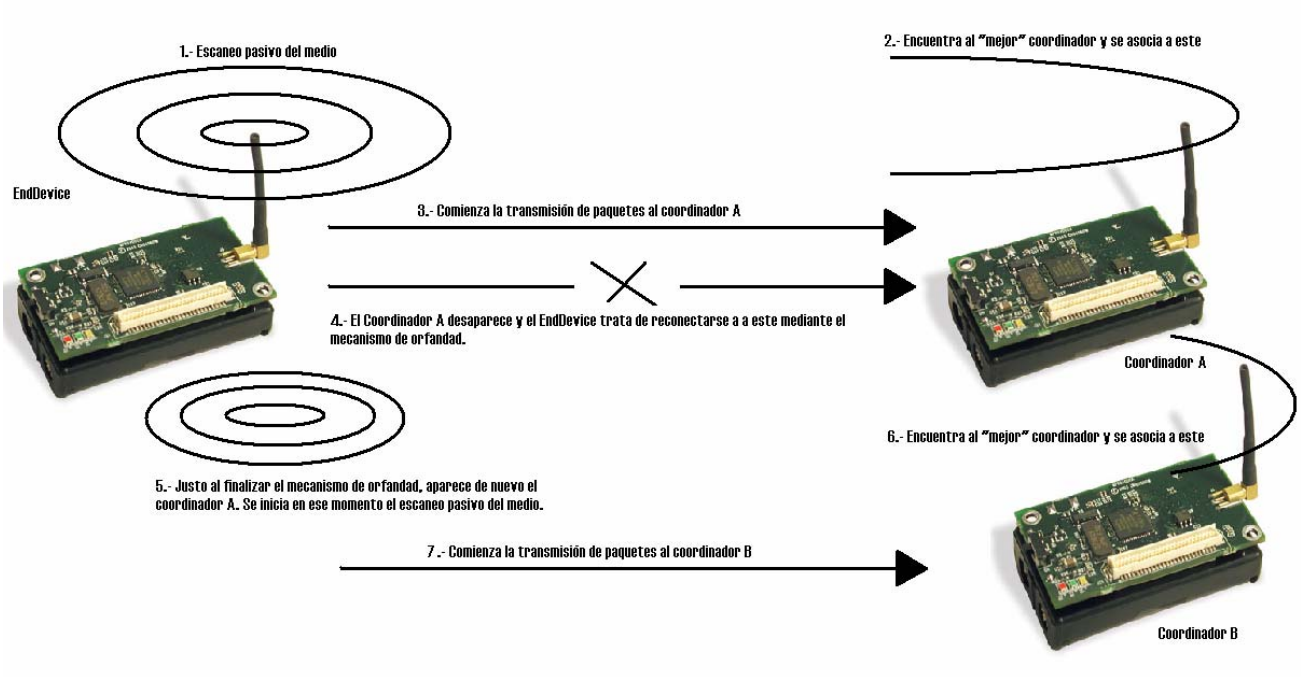

Fig. 56 – Funcionamiento del escenario 9

9.2.- Resultados

El extremo local produce los siguientes resultados al nivel de aplicación:

1.- Se inicia el escaneo pasivo del medio y se asocia al coordinador que mejores características de transmisión le ofrece. Hay dos coordinadores enviando beacons en distintos canales.

```
PASSIVE_SCAN
PASSIVE->T_ScanDuration.fired()1 - Tiempo:2093505000
PASSIVE->T_ScanDuration.fired()2 - Tiempo:4170434000
PASSIVE->T_ScanDuration.fired()3 - Tiempo:6269912000
PASSIVE->T_ScanDuration.fired()4 - Tiempo:8370036000
PASSIVE->T_ScanDuration.fired()5 - Tiempo:10469471000
PASSIVE->T_ScanDuration.fired()6 - Tiempo:12569353000
PASSIVE->T_ScanDuration.fired()7 - Tiempo:14668919000
PASSIVE->T_ScanDuration.fired()8 - Tiempo:16769483000
PASSIVE->T_ScanDuration.fired()9 - Tiempo:18869099000
PASSIVE->T_ScanDuration.fired()10 - Tiempo:20969031000
PASSIVE->T_ScanDuration.fired()11 - Tiempo:23068514000
PASSIVE->T_ScanDuration.fired()12 - Tiempo:25168323000
PASSIVE->T_ScanDuration.fired()13 - Tiempo:27268921000
PASSIVE->T_ScanDuration.fired()14 - Tiempo:29367844000
PASSIVE->T_ScanDuration.fired()15 - Tiempo:31486782000
PASSIVE->MLME_SCAN.confirm
MLME_ASSOCIATE.request
MLME_ASSOCIATE.confirm - Retardo:38445178000
```

2.- Comienza la transmisión de datos hacia el coordinador.

```
MCPS_DATA.request
MCPS_DATA.request
MCPS_DATA.request
```

3.- Se pierde la comunicación con el coordinador y arranca el mecanismo de orfandad. Vuelve a aparecer el coordinador y se conecta a éste. De esta forma se comprueba que está conectado a un coordinador que está en el canal 0x15. Para averiguar el canal se suma a 0x0a el valor que aparece en el escaneo anterior al process_coordinator_realignment.

```
MLME_SYNC_LOSS.indication->ORPHAN
ORPHAN->T_ScanDuration.fired() 1 - Tiempo:3927277000
ORPHAN->T_ScanDuration.fired() 2 - Tiempo:7815462000
ORPHAN->T_ScanDuration.fired() 3 - Tiempo:11742764000
ORPHAN->T_ScanDuration.fired() 4 - Tiempo:15646863000
ORPHAN->T_ScanDuration.fired() 5 - Tiempo:19556746000
ORPHAN->T_ScanDuration.fired() 6 - Tiempo:23461623000
ORPHAN->T_ScanDuration.fired() 7 - Tiempo:27365595000
ORPHAN->T_ScanDuration.fired() 8 - Tiempo:31253921000
ORPHAN->T_ScanDuration.fired() 9 - Tiempo:35181246000
ORPHAN->T_ScanDuration.fired() 10 - Tiempo:39085192000
ORPHAN->T_ScanDuration.fired() 11 - Tiempo:42972102000
process_coordinator_realignment->SENDDATA    -    Retardo:
43474398000
```

4.- Se vuelve a perder la comunicación con el coordinador, pero ahora no vuelve a aparecer, por lo que se inicia un escaneo pasivo del medio. Justo al iniciar dicho escaneo, aparece dicho coordinador de nuevo, pero más alejado del EndDevice que el resto de coordinadores, por lo que debe ofrecer peores características de transmisión, por lo que lo lógico es que se conecte a algún coordinador de los restantes.

```
MLME_SYNC_LOSS.indication->ORPHAN
ORPHAN->T_ScanDuration.fired() 1 - Tiempo:3887436000
ORPHAN->T_ScanDuration.fired() 2 - Tiempo:7814717000
ORPHAN->T_ScanDuration.fired() 3 - Tiempo:11719734000
ORPHAN->T_ScanDuration.fired() 4 - Tiempo:15605944000
ORPHAN->T_ScanDuration.fired() 5 - Tiempo:19533126000
ORPHAN->T_ScanDuration.fired() 6 - Tiempo:23438687000
ORPHAN->T_ScanDuration.fired() 7 - Tiempo:27325884000
ORPHAN->T_ScanDuration.fired() 8 - Tiempo:31253308000
ORPHAN->T_ScanDuration.fired() 9 - Tiempo:35158240000
ORPHAN->T_ScanDuration.fired() 10 - Tiempo:39044372000
ORPHAN->T_ScanDuration.fired() 11 - Tiempo:42972277000
ORPHAN->T_ScanDuration.fired() 12 - Tiempo:46877290000
ORPHAN->T_ScanDuration.fired() 13 - Tiempo:50763311000
ORPHAN->T_ScanDuration.fired() 14 - Tiempo:54690727000
ORPHAN->T_ScanDuration.fired() 15 - Tiempo:58596318000
ORPHAN->MLME_SCAN.confirm->PASSIVE
PASSIVE->T_ScanDuration.fired()1 - Tiempo:64583265000
PASSIVE->T_ScanDuration.fired()2 - Tiempo:66682209000
PASSIVE->T_ScanDuration.fired()3 - Tiempo:68782777000
PASSIVE->T_ScanDuration.fired()4 - Tiempo:70882420000
PASSIVE->T_ScanDuration.fired()5 - Tiempo:72982282000
PASSIVE->T_ScanDuration.fired()6 - Tiempo:75081754000
PASSIVE->T_ScanDuration.fired()7 - Tiempo:77182457000
PASSIVE->T_ScanDuration.fired()8 - Tiempo:79281246000
PASSIVE->T_ScanDuration.fired()9 - Tiempo:81381947000
PASSIVE->T_ScanDuration.fired()10 - Tiempo:83481429000
```

```
PASSIVE->T_ScanDuration.fired()11 - Tiempo:85581293000
PASSIVE->T_ScanDuration.fired()12 - Tiempo:87699349000
PASSIVE->T_ScanDuration.fired()13 - Tiempo:89799186000
PASSIVE->T_ScanDuration.fired()14 - Tiempo:91898269000
PASSIVE->T_ScanDuration.fired()15 - Tiempo:93998086000
PASSIVE->MLME_SCAN.confirm
MLME_ASSOCIATE.request
MLME_ASSOCIATE.confirm - Retardo: 100508040000
```

5.- Para comprobar que se ha cambiado a otro coordinador, se van desconectando coordinadores uno a uno para que se inicie el mecanismo de orfandad. Si al apagar un coordinador entra en funcionamiento dicho mecanismo, se determina que ese es el coordinador al que se asocia el dispositivo EndDevice. Si se enciende de nuevo y se reconecta, se puede averiguar en que canal está transmitiendo. En este caso, se determina que está en el canal 10 empleando el mismo método que en el paso 2. También se puede determinar el canal escaneando con el sniffer todos los canales y viendo cual es el que envía datos de `MCPS_DATA.request`.

```
ORPHAN->T_ScanDuration.fired() 1 - Tiempo:3927185000
ORPHAN->T_ScanDuration.fired() 2 - Tiempo:7831757000
ORPHAN->T_ScanDuration.fired() 3 - Tiempo:11719647000
ORPHAN->T_ScanDuration.fired() 4 - Tiempo:15623653000
ORPHAN->T_ScanDuration.fired() 5 - Tiempo:19551045000
ORPHAN->T_ScanDuration.fired() 6 - Tiempo:23438666000
process_coordinator_realignment->SENDDATA - Retardo:
23898038000
MCPS_DATA.request
MCPS_DATA.request
MCPS_DATA.request
```

A continuación se muestran las tramas capturadas a partir del sniffer.

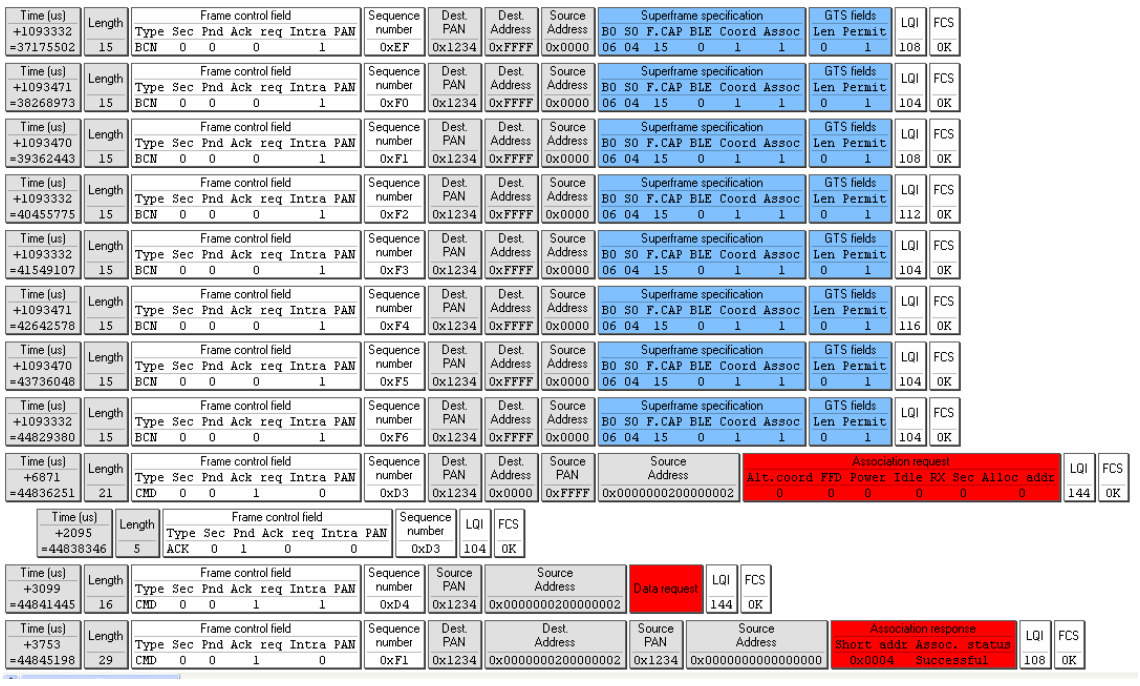

Fig. 57 – Finalización del escaneo pasivo y proceso de asociación escenario 9

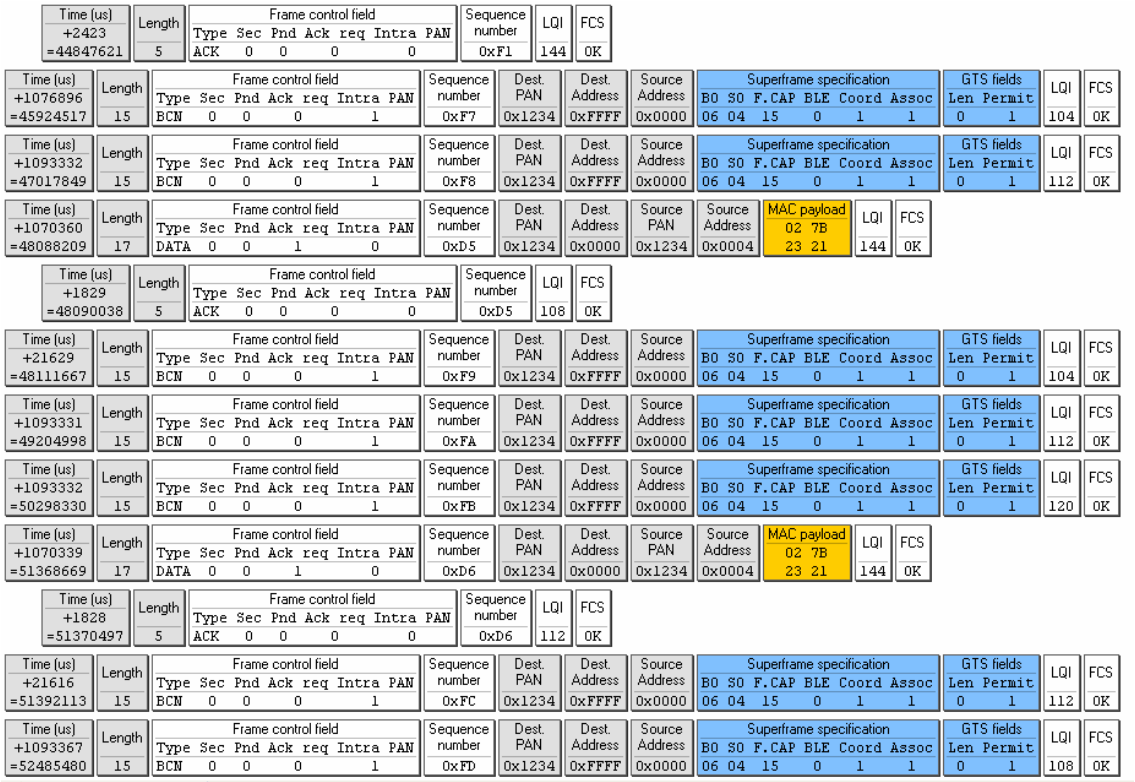

Fig. 58 – Envío de datos al coordinador escenario 9

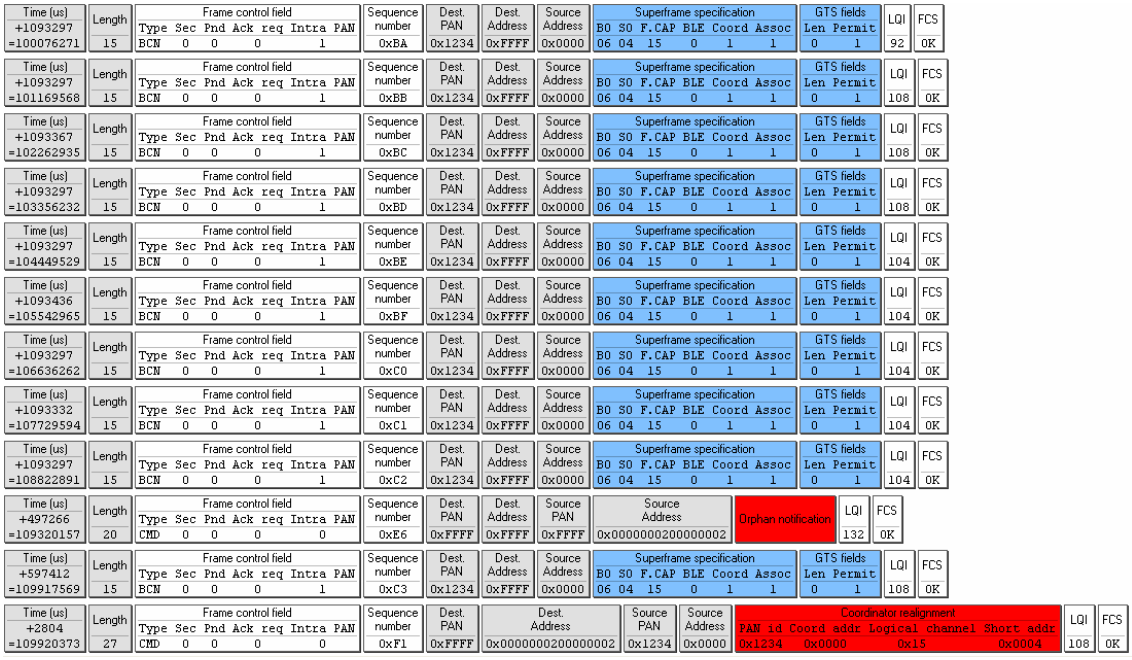

Fig. 59 – Finalización del mecanismo de orfandad y realineamiento con el coordinador escenario 9

A continuación se muestra el retardo producido por el escaneo pasivo y el mecanismo de orfandad en este escenario.

Escaneo Pasivo	Retardo (ms)
MLME_ASSOCIATE.confirm	100508,04

Escaneo Orfandad para canal 0x15	Retardo (ms)
process_coordinator_realignment->SENDDATA	43474,398

Escaneo Orfandad para canal 0x10	Retardo (ms)
process_coordinator_realignment->SENDDATA	23898,038

Tabla 13 - Retardo de escaneo pasivo y mecanismo de orfandad escenario 9

9.3.- Conclusiones

Se observa en las tramas analizadas por el sniffer y a nivel de aplicación el correcto funcionamiento del sistema. Se puede ver como en el proceso de asociación, ante una petición de asociación, se produce una respuesta de confirmación por parte del coordinador. Tras esto, comienza el envío de datos. También se puede observar como en el mecanismo de orfandad, cuando se trata de realinear el EndDevice con el coordinador, envía una notificación de orfandad, ante la cual se responde con una trama de Coordinator realignment. Por último, en el mecanismo de orfandad, si no se consigue realinear con el anterior coordinador se inicia un escaneo pasivo del medio olvidándose de cual era el coordinador al que se estaba conectado.

En este escenario EndDevice selecciona siempre el coordinador con las mejores características de transmisión. Esto se comprueba al observar que si se pierde la conexión con un coordinador y se aleja, acaba asociándose a otro coordinador más cercano.

10.- Escenario de trabajo 10

10.1-Funcionamiento

El décimo escenario de trabajo se emplea para analizar el mecanismo de asociación y reconexión entre dispositivos MicaZ en los límites de cobertura del coordinador. Consta de un dispositivo MIB600 al que se conecta un dispositivo MicaZ local (EndDevice), un ordenador y dos dispositivos MicaZ remotos (Coordinador A y Coordinador B). Los coordinadores están muy alejados de EndDevice (8 metros con obstáculos). El dispositivo MicaZ local le indica constantemente al ordenador el proceso que está ejecutando (escaneo pasivo, mecanismo de orfandad o envío de datos), y en qué fase de mecanismo está. A partir de estos datos, el ordenador calcula estadísticas. En los anexos aparece el código de la implementación del sistema y los mensajes que le envía al ordenador. En este apartado no aparece descrito el envío de mensajes al ordenador. El funcionamiento del sistema es el siguiente:

1. El extremo EndDevice realiza un escaneo pasivo del medio para buscar el coordinador que le ofrezca las mejores condiciones de comunicación.

Ambos coordinadores se encuentran muy alejados del extremo EndDevice, por lo que la calidad de la comunicación será peor.

2. Una vez encontrado el “mejor” coordinador (el coordinador que puede ofrecer las mejores condiciones de comunicación), se asocia a éste.

3. Cuando consigue asociarse, comienzan a transmitirse paquetes de datos al coordinador. El coordinador al recibir dichos paquetes, simplemente los elimina y no genera ninguna respuesta al EndDevice.

4. En cierto momento de la transmisión de datos, EndDevice pierde la comunicación con el coordinador debido a que se apaga dicho coordinador. Ahora EndDevice trata de reconectarse al coordinador al que estaba conectado a través del mecanismo de orfandad.

5. No consigue reconectarse a dicho coordinador puesto que permanece apagado, por lo que se inicia otra vez un escaneo pasivo del medio para buscar el coordinador que le ofrezca las mejores condiciones de comunicación. Justo en el momento en el que va a empezar el escaneo pasivo, aparece otra vez el coordinador anterior pero a mucha mayor distancia que antes.

6. Mediante el escaneo pasivo, determina al otro coordinador como el “mejor” y se asocia a éste.

7. Cuando consigue asociarse, comienzan a transmitirse paquetes de datos al coordinador. El coordinador al recibir dichos paquetes, simplemente los elimina y no genera ninguna respuesta al EndDevice.

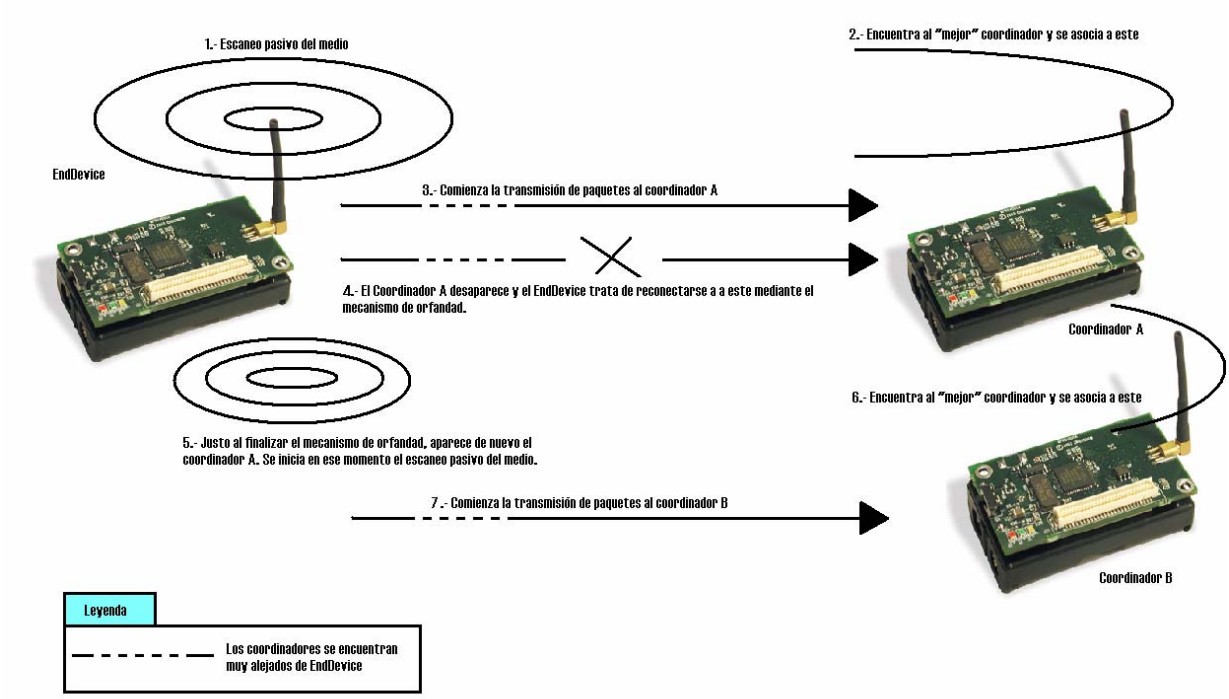

Fig. 60 – Funcionamiento del escenario 10

10.2.- Resultados

El extremo local produce los siguientes resultados al nivel de aplicación:

1.- Se inicia el escaneo pasivo del medio y se asocia al coordinador que mejores características de transmisión le ofrece. Hay dos coordinadores enviando beacons en distintos canales.

```
PASSIVE_SCAN
PASSIVE->T_ScanDuration.fired()1 - Tiempo:2110360000
PASSIVE->T_ScanDuration.fired()2 - Tiempo:4209950000
PASSIVE->T_ScanDuration.fired()3 - Tiempo:6309928000
PASSIVE->T_ScanDuration.fired()4 - Tiempo:8409801000
PASSIVE->T_ScanDuration.fired()5 - Tiempo:10509673000
PASSIVE->T_ScanDuration.fired()6 - Tiempo:12609625000
PASSIVE->T_ScanDuration.fired()7 - Tiempo:14709492000
PASSIVE->T_ScanDuration.fired()8 - Tiempo:16808816000
PASSIVE->T_ScanDuration.fired()9 - Tiempo:18908701000
PASSIVE->T_ScanDuration.fired()10 - Tiempo:21008623000
PASSIVE->T_ScanDuration.fired()11 - Tiempo:23108704000
PASSIVE->T_ScanDuration.fired()12 - Tiempo:25208573000
PASSIVE->T_ScanDuration.fired()13 - Tiempo:27308644000
PASSIVE->T_ScanDuration.fired()14 - Tiempo:29407859000
PASSIVE->T_ScanDuration.fired()15 - Tiempo:31485780000
PASSIVE->MLME_SCAN.confirm
MLME_ASSOCIATE.request
MLME_ASSOCIATE.confirm - Retardo:38002761000
```

2.- Comienza la transmisión de datos hacia el coordinador.

```
MCPS_DATA.request
MCPS_DATA.request
MCPS_DATA.request
```

3.- Se pierde la comunicación con el coordinador y arranca el mecanismo de orfandad.

```
MLME_SYNC_LOSS.indication->ORPHAN
ORPHAN->T_ScanDuration.fired() 1 - Tiempo:3887523000
ORPHAN->T_ScanDuration.fired() 2 - Tiempo:7814724000
ORPHAN->T_ScanDuration.fired() 3 - Tiempo:11719455000
ORPHAN->T_ScanDuration.fired() 4 - Tiempo:15606689000
ORPHAN->T_ScanDuration.fired() 5 - Tiempo:19532930000
ORPHAN->T_ScanDuration.fired() 6 - Tiempo:23437940000
ORPHAN->T_ScanDuration.fired() 7 - Tiempo:27326136000
ORPHAN->T_ScanDuration.fired() 8 - Tiempo:31253455000
ORPHAN->T_ScanDuration.fired() 9 - Tiempo:35157312000
ORPHAN->T_ScanDuration.fired() 10 - Tiempo:39044875000
ORPHAN->T_ScanDuration.fired() 11 - Tiempo:42972185000
ORPHAN->T_ScanDuration.fired() 12 - Tiempo:46877290000
ORPHAN->T_ScanDuration.fired() 13 - Tiempo:50763547000
ORPHAN->T_ScanDuration.fired() 14 - Tiempo:54690957000
ORPHAN->T_ScanDuration.fired() 15 - Tiempo:58595817000
```

4.- Se inicia un escaneo pasivo del medio al no poder reconectarse mediante el mecanismo de orfandad. Justo al iniciar dicho escaneo, aparece dicho coordinador de nuevo, pero más alejado del EndDevice que el resto de coordinadores, por lo que debe ofrecer peores características de transmisión, por lo que lo lógico es que se conecte a algún coordinador de los restantes.

```
ORPHAN->MLME_SCAN.confirm->PASSIVE
PASSIVE->T_ScanDuration.fired()1 - Tiempo:64583200000
PASSIVE->T_ScanDuration.fired()2 - Tiempo:66682612000
PASSIVE->T_ScanDuration.fired()3 - Tiempo:68782733000
PASSIVE->T_ScanDuration.fired()4 - Tiempo:70882090000
PASSIVE->T_ScanDuration.fired()5 - Tiempo:72983010000
PASSIVE->T_ScanDuration.fired()6 - Tiempo:75081481000
PASSIVE->T_ScanDuration.fired()7 - Tiempo:77181638000
PASSIVE->T_ScanDuration.fired()8 - Tiempo:79282540000
PASSIVE->T_ScanDuration.fired()9 - Tiempo:81381159000
PASSIVE->T_ScanDuration.fired()10 - Tiempo:83482060000
PASSIVE->T_ScanDuration.fired()11 - Tiempo:85598459000
PASSIVE->T_ScanDuration.fired()12 - Tiempo:87698367000
PASSIVE->T_ScanDuration.fired()13 - Tiempo:89798193000
PASSIVE->T_ScanDuration.fired()14 - Tiempo:91898162000
PASSIVE->T_ScanDuration.fired()15 - Tiempo:93997991000
PASSIVE->MLME_SCAN.confirm
MLME_ASSOCIATE.request
MLME_ASSOCIATE.confirm - Retardo:   101398163000
```

5.- Para comprobar que se ha cambiado a otro coordinador, se van desconectando coordinadores uno a uno para que se inicie el mecanismo de orfandad. Si al apagar un coordinador entra en funcionamiento dicho mecanismo, se determina que ese es el coordinador al que se asocia el dispositivo EndDevice. Si se enciende de nuevo y se reconecta, se puede averiguar en que canal está transmitiendo. En este caso, se determina que

está en el canal 10 empleando el mismo método que en el paso 2. También se puede determinar el canal escaneando con el sniffer todos los canales y viendo cual es el que envía datos de `MCPS_DATA.request`.

```
MLME_SYNC_LOSS.indication->ORPHAN
ORPHAN->T_ScanDuration.fired() 1 - Tiempo:3887074000
ORPHAN->T_ScanDuration.fired() 2 - Tiempo:7791901000
ORPHAN->T_ScanDuration.fired() 3 - Tiempo:11719211000
ORPHAN->T_ScanDuration.fired() 4 - Tiempo:15605378000
ORPHAN->T_ScanDuration.fired() 5 - Tiempo:19532717000
ORPHAN->T_ScanDuration.fired() 6 - Tiempo:23437430000
process_coordinator_realignment->SENDDATA - Retardo:
23990994000
```

A continuación se muestran las tramas capturadas a partir del sniffer.

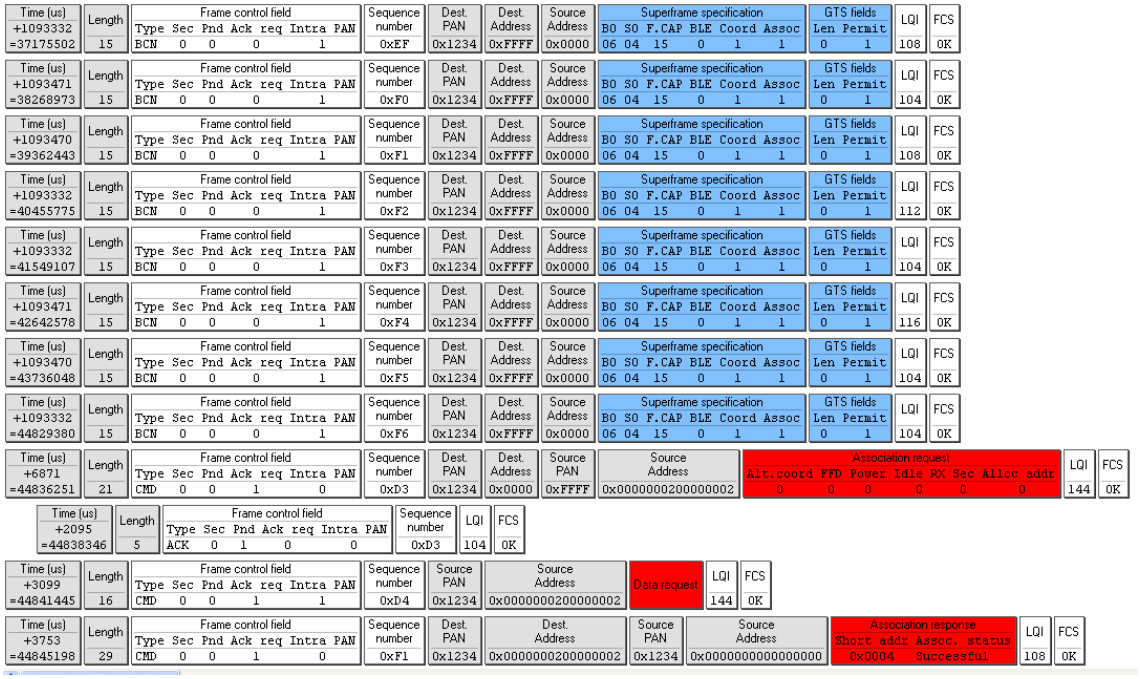

Fig. 61 – Finalización del escaneo pasivo y proceso de asociación escenario 10

Fig. 62 – Envío de datos al coordinador escenario 10

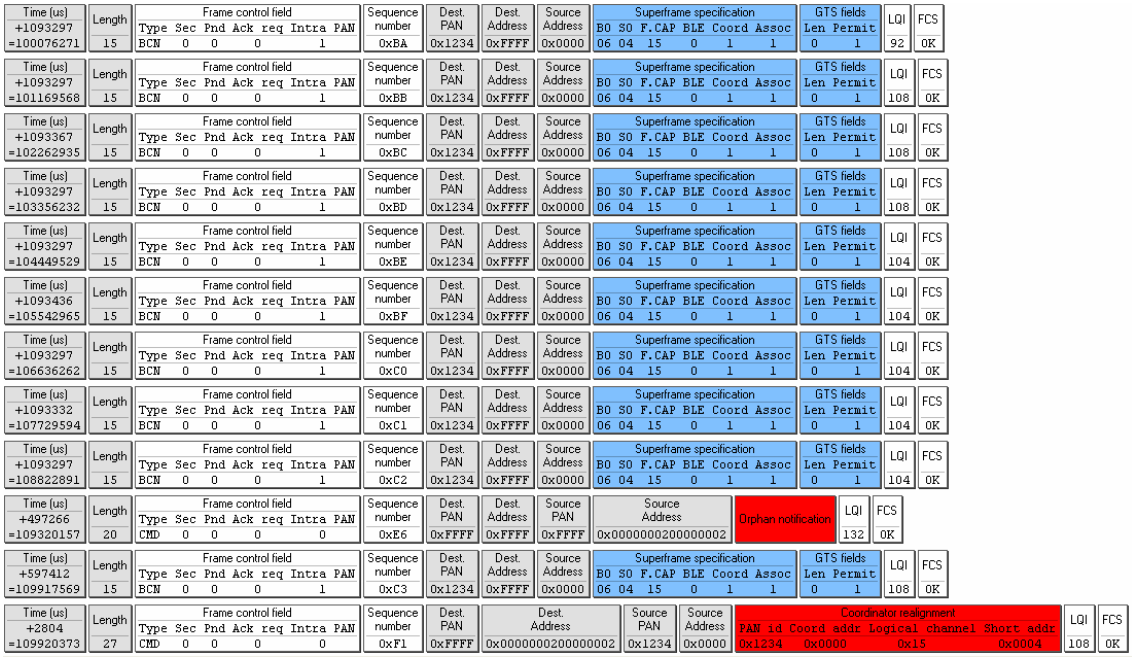

Fig. 63 – Finalización del mecanismo de orfandad y realineamiento con el coordinador escenario 10

A continuación se muestra el retardo producido por el escaneo pasivo y el mecanismo de orfandad en este escenario.

Escaneo Pasivo	Retardo (ms)
MLME_ASSOCIATE.confirm	101398,163

Escaneo Orfandad para canal 0x10	Retardo (ms)
process_coordinator_realignment->SENDDATA	23990,994

Tabla 14 - Retardo de escaneo pasivo y mecanismo de orfandad escenario 10

10.3.- Conclusiones

Se observa en las tramas analizadas por el sniffer y a nivel de aplicación el correcto funcionamiento del sistema. Se puede observar como en el proceso de asociación, ante una petición de asociación, se produce una respuesta de confirmación por parte del coordinador. Tras esto, comienza el envío de datos. También se puede observar como en el mecanismo de orfandad, cuando se trata de realinear el EndDevice con el coordinador, envía una notificación de orfandad, ante la cual se responde con una trama de Coordinator realignment. Por último, en el mecanismo de orfandad, si no se consigue realinear con el anterior coordinador se inicia un escaneo pasivo del medio olvidándose de cual era el coordinador al que se estaba conectado.

El colocar los coordinadores en los límites de cobertura de la tecnología MicaZ, no implica una variación significativa en los parámetros de retardo del mecanismo de orfandad y el escaneo pasivo de canales. Esto es debido principalmente a las limitadas distancias en las que se desarrolla la transmisión

11.- Escenario de trabajo 11

11.1.- Funcionamiento

Este escenario se emplea para el análisis del mecanismo de GTS y consta de dos dispositivos MicaZ, uno hace las veces de coordinador y el otro las veces de EndDevice situado a 10 metros del coordinador. Mediante un ordenador conectado al extremo EndDevice, se pueden comprobar los mensajes del mecanismo de GTS que se están generando en dicho extremo. El funcionamiento del escenario es el siguiente:

1. El extremo EndDevice se asocia al coordinador.

2. El extremo EndDevice le indica al coordinador que va a transmitirle datos empleando para ello el mecanismo de GTS. En este escenario, el parámetro BEACON ORDER es mayor que el parámetro SUPERFRAME ORDER.

3. Una vez indicados los parámetros al extremo coordinador, comienza la transmisión de datos.

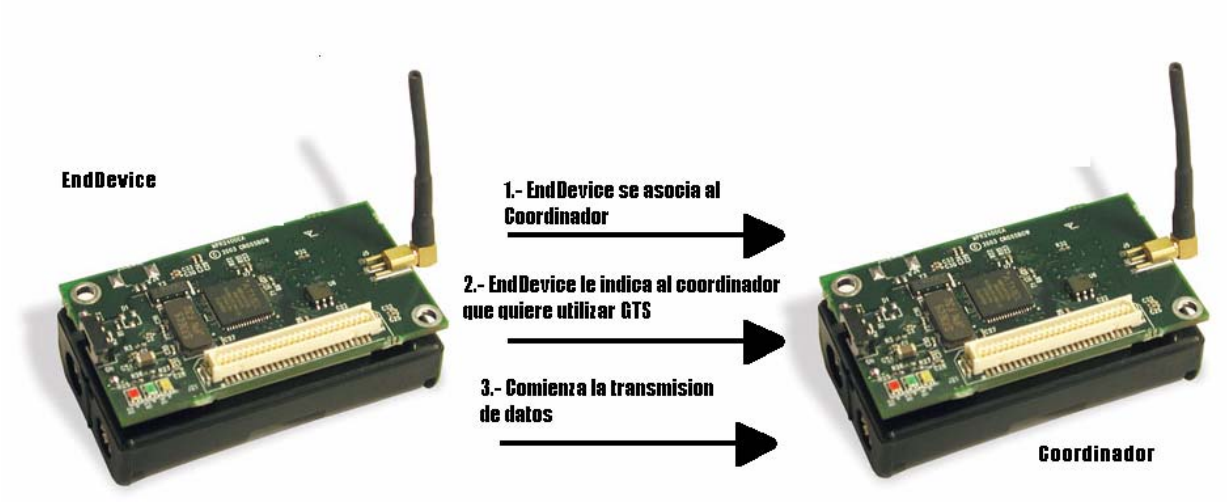

Fig. 64 – Funcionamiento del escenario 11

11.2.- Resultados

A continuación se muestran las tramas capturadas con el sniffer.

Time (us)	Length	Frame control field								Sequence number	Dest. PAN	Dest. Address	Source Address	Superframe specification						GTS fields		LQI	FCS
		Type	Sec	Pnd	Ack req	Intra PAN								BO	SO	F.CAP	BLE	Coord	Assoc	Len	Permit		
+1093332	15	BCN	0	0	0	1				0xBD	0x1234	0xFFFF	0x0000	06	04	15	0	1	1	0	1	76	OK
=24054696																							
+1093332	15	BCN	0	0	0	1				0xBE	0x1234	0xFFFF	0x0000	06	04	15	0	1	1	0	1	84	OK
=25148028																							
+1093402	15	BCN	0	0	0	1				0xBF	0x1234	0xFFFF	0x0000	06	04	15	0	1	1	0	1	128	OK
=26241430																							
+1093332	15	BCN	0	0	0	1				0xC0	0x1234	0xFFFF	0x0000	06	04	15	0	1	1	0	1	96	OK
=27334762																							
+1094061	15	BCN	0	0	0	1				0xC1	0x1234	0xFFFF	0x0000	06	04	15	0	1	1	0	1	104	OK
=28428823																							
+1093297	15	BCN	0	0	0	1				0xC2	0x1234	0xFFFF	0x0000	06	04	15	0	1	1	0	1	92	OK
=29522120																							
+1093333	15	BCN	0	0	0	1				0xC3	0x1234	0xFFFF	0x0000	06	04	15	0	1	1	0	1	120	OK
=30615453																							
+95341	11	CMD	0	0	1	1				0xD3	0x1234	0x0002	GTS request									68	OK
=30710794													Length	Direction	Type								
													01	TX only	Alloc								
+1551	5	ACK	0	0	0	0				0xD3												124	OK
=30712345																							
+997034	19	BCN	0	0	0	1				0xC4	0x1234	0xFFFF	0x0000	06	04	14	0	1	1	1	1	120	OK
=31709379																				0b00000000	0x0002/15/1		
+251643	15	DATA	0	0	1	0				0xD4	0x1234	0x0000	0x1234	0x0002	MAC payload							72	OK
=31961022															27								
															63								
+1752	5	ACK	0	0	0	0				0xD4												120	OK
=31962774																							

Fig. 65 – Mecanismo GTS

Retardo Medio de proceso de inicialización del Mecanismo de GTS(ms)	**0,174833333**
Desviacion Tipica	**0,03056652**

Tabla 15 - Estadísticas a partir de los datos obtenidos escenario 11

11.3.- Conclusiones

Se observa que el funcionamiento de este escenario es el correcto. El coordinador envía beacons normales, y tras una petición de GTS por parte de EndDevice, se adapta para garantizar la transmisión de datos por parte de EndDevice. Esta adaptación se observa en el campo GTS field. A partir de ese momento, los datos de EndDevice se envían empleando el mecanismo de GTS.

De las estadísticas obtenidas, se puede observar que el proceso de inicialización del mecanismo de GTS es rápido y tiene una varianza pequeña.

Conclusiones y lineas futuras

Durante el desarrollo de este Proyecto Fin de Carrera se han obtenido las siguientes conclusiones funcionales, útiles para futuros desarrollos:

- La implementación de aplicaciones basadas en TinyOS es un proceso laborioso debido al uso del nuevo lenguaje de programación nesC y a la gran cantidad de librerias proporcionadas por el equipo de desarrollo de TinyOS.
- El uso de XubunTOS facilita más aun el desarrollo de aplicaciones basadas en TinyOS.

Basados en los resultados obtenidos en el análisis de las redes 802.15.4, se concluyen de los siguientes puntos:

- Si bien las redes 802.15.4 permite transmitir datos a una velocidad de 250Kbps, el propio procesamiento hardware del protocolo limita la capacidad efectiva, lo que impide el completo aprovechamiento de dicha tecnología.
- Los retardos de transmisión se ven afectados en gran medida por los tamaños de búferes de transmisión y recepción de los equipos que se conecten a los dispositivos MicaZ.
- Los retardos se ven afectados en menor medida cuando la transferencia de datos la gestiona MicaZ.
- El mecanismo de asociación pasiva y el de orfandad permiten a los dispositivos basados en 802.15.4 la autogestión de sus redes sin que tenga que intervenir ningún usuario en caso de pérdida del nodo coordinador.
- El mecanismo de GTS es útil de cara al uso de aplicaciones con requisitos baja latencia y mayor ancho de banda.

En cuanto a lo personal, este proyecto me ha hecho descubrir una temática que desconocía totalmente y me ha permitido trabajar a un nivel más bajo en la capa OSI, lo que me ha permitido comprender mejor el funcionamiento de un protocolo a un nivel tan bajo. Además la compresión de 802.15.4 puede llegar a ser útil en el transcurso de mi vida laboral.

En cuanto posibles lineas futuras de este Proyecto Fin de Carrera:

- Este trabajo debe permitir el desarrollo de aplicaciones optimizadas en esta tecnología. De igual forma, en una segunda fase se puede plantear la posibilidad de optimizar la tecnología para determinadas aplicacaciones, como las que se plantean a contnuación.
- Se puede desarrollar una implementación que permita la videovigilancia en zonas donde el instalar una red convencional sería un proceso complicado.
- Se puede desarrollar un dispositivo de control de individuos en determinadas zonas acoplando dicha tecnología a dichos individuos. Por ejemplo sería útil en el caso de control de niños en colegios o guarderias.
- Sería una posibilidad la transmisión de audio entre dispositivos 802.15.4. Esto permitiría por ejemplo su uso en trenes, donde en la reproducción de las películas, el audio se transmite directamente a un receptor con unos cascos para escuchar la película.

- Se podrían desarrollar aplicaciones basadas en sensores para el control de procesos industriales en los que en su momento no se pudo desarrollar una red para su control.
- El desarrollo de una capa de red para el enrutamiento de la información entre los distintos dispositivos.

Bibliografía

- Tipos de modulaciones en el 802.15.4 - http://es.wikipedia.org/wiki/DSSS
- RF Transmisores- http://www.chipcon.com/
- MOTE-KIT Datasheet - http://www.xbow.com/Products/Product_pdf_files/Wireless_pdf/MICAz_Kit_Data
- sheet.pdf
- CrossBow - http://www.xbow.com
- Mote MICAz Datasheet - http://www.cmt-gmbh.de/MICAz.pdf
- TinyOS - http://www.cs.utah.edu/~coop/embedded/tinyos.html
- Serial Interface Board MIB510 Datasheet - http://www.cmt-gmbh.de/MIB510CA.pdf
- Ethernet Interface Board MIB600 Datasheet - http://www.cmt-gmbh.de/MIB600CA.pdf
- ZigBee/IEEE 802.15.4 Summary - http://www.eecs.berkeley.edu/~csinem/academic/publications/zigbee.pdf
- 60 Emulación de RFID activo mediante la plataforma MICAz
- MTS/MDA Mote Sensor and DAQ Manual - http://www.xbow.com/Support/Support_pdf_files/MTSMDA_
- Series_Users_Manual.pdf
- Carlos López Truchero *"Aplicaciones de sensado: emulación de RFID*
- *activo y medición en tiempo real"* trabajo final de carrera, 29 Enero 2006.
- Getting Started Guide, Crossbow - http://www.xbow.com/Support/Support_pdf_files/Getting_Started_Guide.pdf
- Especificación de la estructura de trama del 802.15.4 - http://www.chipcon.com/index.cfm?kat_id=1&action=faq&faq_id=3
- An IEEE 802.15.4 protocol implementation(in nesC/TinyOS): Reference Guide v1.2 - André CUNHA, Mário ALVES, Anis Koubâa
- IEEE 802.15.4 Standard-2003, "Part 15.4: Wireless Medium Access Control (MAC) and Physical Layer (PHY) Specifications for Low-Rate Wireless Personal Area Networks (LR-WPANs)", IEEE-SA Standards Board, 2003.
- D. Gay, P. Levis, R. Behren, M. Welsh, E. Brewer, D. Culler, "The nesC Language:A Holistic Approach to Networked Embedded Systems", in PLDI'03.
- Texas Instruments Incorporated, "Chipcon Packet Sniffer for IEEE 802.15.4",www.chipcon.com, 2006
- http://www.it.uc3m.es/~jgr/publicaciones/Tesis_Jaime.pdf
- IEEE 802.15 WPAN™ Task Group 4 (TG4). Abril/2006. Última consulta: Abril/2006. Disponible en: http://www.ieee802.org/15/pub/TG4.html
- MAYNÉ, Jordi. IEEE 802.15.4 y Zigbee. SILICA. Octubre/2004. Última consulta: Marzo/2006. Disponible en: http://www.bairesrobotics.com.ar/data/IEEE_ZIGBEE_SILICA.pdf

Agradecimientos

En primer lugar, me gustaría agradecer a mis directores de proyecto, Antonio y Felipe, el haberme dado la posibilidad de desarrollar este Proyecto Fin de Carrera y el Proyecto Fin de Carrera de la carrera de Telemática. También es de agradecer el hecho de que hubiesen sido ellos quienes me buscaron para desarrollarlo, ya que eso me dio confianza, ya que sentí que mi forma de trabajar en el proyecto que hice en la carrera técnica les agradó.

También quiero agradecerles a mis compañeros de carrera y especialmente a Angulo, Angel, Perico, Dayer, Alex, Antiñico, Edu, Jorgito y todos aquellos que me acompañaron a lo largo de toda mi formación como ingeniero, estos años universitarios donde he logrado madurar a su lado.

Agradezco también a mi familia el haberme apoyado todos estos años y el haber sufrido conmigo todos mis fracasos y haber disfrutado de mis exitos.

Muchas gracias a Lorena, ya que me ha dado fuerzas para poder acabar este proyecto que me parecía interminable.

Me acuerdo tambien de mis amigos, que tanto me han hecho por mi. Quiero agradecer a Jose, Selma, Sosa, Pedro Pablo, Ruben, Angel, Alex y a todos los que me conocen su amistad y esos momentos que me hacían olvidarme de los examenes cuando más estresado iba.

También hay que agradecer a Primary Net y Navantia el haberme dado mi primer trabajo como ingeniero y el haberme permitido acabar este proyecto en horas de trabajo.

Gracias también a Yván, Alicia, Jose, Fran, Iván y en general todos los compañeros bazaneros por soportarme estos meses dándoles la brasa con mis avances del proyecto.