ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA DE TELECOMUNICACIÓN
UNIVERSIDAD POLITÉCNICA DE CARTAGENA

Proyecto Fin de Carrera

Estudio experimental del mecanismo de control de potencia a nivel MAC en redes de sensores inalámbricas tipo MICA

AUTOR: Juan Manuel Pérez Mañogil
DIRECTOR: Javier Vales Alonso
Marzo / 2009

No se ofusque con este terror tecnológico que ha construido. La posibilidad de destruir un planeta es algo insignificante comparado con el poder de la fuerza.

Darth Vader

Autor	Juan Manuel Pérez Mañogil
E-mail del Autor	kwyjibo@ono.com
Director(es)	Javier Vales Alonso
E-mail del Director	jvales@upct.es
Codirector(es)	-
Título del PFC	Estudio experimental del mecanismo de control de potencia a nivel MAC en redes de sensores inalámbricas tipo MICA
Descriptores	tinyos nesc tpc transmission power control s-mac scp-mac b-mac

Resumen

En los últimos años, las redes de sensores han empezado a cobrar importancia en el sector tecnológico, debido a sus cada vez mas atractivas características, tamaño, consumo, versatilidad.
Este nuevo campo de expansión ha sido explotado a conciencia con numerosas aplicaciones, pero no han de descuidarse los desarrollos que potencien las características de estos dispositivos.
En este proyecto, se ha explorado un campo característico de las redes de sensores, el bajo consumo, para analizar una posible optimización de los recursos en los dispositivos de redes de sensores.
Para ello, se propone un mecanismo de control de potencia, para minimizar el consumo en emisión, a la vez que se ha hecho una implementación de dicho mecanismo, con base a varios protocolos existentes, con los que se han tomado muestras para validar los objetivos de dicho mecanismo de control de potencia.

Titulación	Ingeniería de Telecomunicación
Intensificación	Planificación y Gestión de Telecomunicaciones
Departamento	Tecnologías de la Información y las Comunicaciones
Fecha de Presentación	Junio – 2009

Resumen

En los últimos años, las redes de sensores han empezado a cobrar importancia en el sector tecnológico, debido a sus cada vez mas atractivas características, tamaño, consumo, versatilidad.

Este nuevo campo de expansión ha sido explotado a conciencia con numerosas aplicaciones, pero no han de descuidarse los desarrollos que potencien las características de estos dispositivos.

En este proyecto, se ha explorado un campo característico de las redes de sensores, el bajo consumo, para analizar una posible optimización de los recursos en los dispositivos de redes de sensores.

Para ello, se propone un mecanismo de control de potencia, para minimizar el consumo en emisión, a la vez que se ha hecho una implementación de dicho mecanismo, con base a varios protocolos existentes, con los que se han tomado muestras para validar los objetivos de dicho mecanismo de control de potencia.

Agradecimientos

Son muchas las personas que deben recibir mis mas sinceros agradecimientos, así que espero no olvidarme de los mas importantes, y que los demás no se sientan ofendidos por no aparecer aquí.

Gracias a mi director, Javier Vales Alonso, no se cuanta paciencia le quedará para soportarme... Espero que la suficiente.

Gracias a los compañeros del laboratorio, primero Félix, Marco y Jose, luego Paco Vicente, y más tarde Francisco Ponce, Paco Parrado y Paco Pérez, también por aguantarme, y lo que les queda.

Gracias a la familia, mi padre, mi madre y mis hermanas, por aguantarme aún más.

Y gracias al resto de amigos, por aguantarme en los ratos ociosos. Gracias

Índice de materias

Índice de figuras

Capítulo 1

Introducción

1.1 Retrospectiva

El desarrollo tecnológico ligado a objetivos militares ha sido hervidero de multitud de tecnologías que con el transcurso de los años ha acabado desembocando en un uso civil. Bien conocido es el desarrollo de Internet, y por tanto, no debería sorprender que otro de estos desarrollos de origen militar sean las redes de sensores inalámbricas.

Durante la guerra fría, el gobierno de los Estados Unidos desplegó una serie de boyas submarinas equipadas con sensores acústicos que sirvieran para poder detectar la aproximación de los silenciosos submarinos soviéticos.

En la década de los ochenta, la *Defense Advanced Research Projects Agency* (DARPA) enfocó sus investigaciones en una red de sensores denominada *Distributed Sensor Networks* (DSN), gracias a la cual se desarrollaron sistemas operativos y lenguajes de programación orientados de forma específica a redes de sensores. Esto dio lugar posteriormente al sistema *Cooperative Engagement Capability* (CEC), consistente en un grupo de radares que trabajan de manera colaborativa, para poder obtener un mapa de mayor exactitud.

Estas redes, sin embargo, solo satisfacían objetivos militares, obviando en cierto grado lo que hoy se consideran características esenciales en este tipo de redes, a saber, la autonomía y el tamaño de los dispositivos. No es hasta finales de la década de los 90 cuando los

usos civiles de este tipo de redes se empieza a disparar, y con ello, se promueve el desarrollo de nuevos dispositivos, cada vez más pequeños y capaces.

1.2 Planteamiento

Ya en la década de los 90, el uso de tecnologías inalámbricas empieza a sufrir un espectacular desarrollo y explotación. Primero son tecnologías destinadas al mercado de voz, con GSM, y poco más tarde, las tecnologías para datos, con 802.11 primero, seguido de Bluetooth y otras muchas con aplicaciones aún más especificas.

Este auge de comunicaciones inalámbricas va, poco a poco, haciendo posibles nuevos progresos en aplicaciones que anteriormente eran prácticamente inviables, por diversos factores, y poniendo en mente de muchos desarrolladores aplicaciones que serán factibles en pocos años.

Las redes de sensores inalámbricas, aportan una serie de elementos característicos, que están en constante mejora, un tamaño reducido, una autonomía altísima e infinidad de elementos sensores disponibles.

Con estas características, por ejemplo, se hace posible tener monitorizada una gran extensión de terreno, sin perturbar apenas el mismo, dado su tamaño; pudiendo contar con la información en tiempo real, gracias a las capacidades de comunicación inalámbrica; con un mantenimiento mínimo, dada su alta autonomía; y lo que es más importante, sin tener que desplazar personal donde se despliegan, evitando posibles peligros, como por ejemplo, monitorizar zonas de alta actividad volcánica o sísmica, o reduciendo costes, como en una hipotética misión de exploración espacial.

Además, en las redes de sensores inalámbricas, hay inherente un concepto muy importante, que es a la vez característica fundamental. Esta red se concibe como un modelo distribuido, y por tanto debe de estar preparada para tolerar fallos. En cualquier momento uno de los nodos desplegados puede fallar, o quedar inutilizado por cualquier motivo, y el resto de la red debe acomodarse a este cambio, sin que la perdida suponga una catástrofe para toda la red.

Sin embargo, hay que tener otros factores muy en cuenta a la hora de diseñar este tipo de redes. En primer lugar, las restricciones de tamaño y autonomía hacen que los desarrollos en estas plataformas sean ciertamente limitados por estos aspectos, que implican

procesadores de (relativa) poca capacidad, y poca memoria. El desarrollador se verá obligado a tomar decisiones adecuadas al dispositivo.

Por otro lado, muchas de las aplicaciones más interesantes, imposibilitan que estos dispositivos cuenten con ningún tipo de mantenimiento (pensemos en el despliegue de una red en entornos de poca accesibilidad o de alta peligrosidad), lo que hace que el factor de autonomía sea importantísimo, ya que, la vida útil del dispositivo vendrá dada por la vida útil de las baterías que lo alimente.

1.3 Objetivos

Por tanto, en un intento de reforzar los posibles puntos débiles de las redes de sensores, el siguiente proyecto hará una propuesta de arquitectura que permita a casi cualquiera de estas redes inalámbricas disfrutar de un ahorro significativo de energía.

Esto, se llevará acabo mediante la implementación de un mecanismo de ajuste de potencia de emisión de radio, uno de los elementos, si no el más crítico, donde se consume mayor cantidad de energía.

Capítulo 2

Hardware

2.1 Introducción

Ha sido durante los últimos años que el mercado de sensores inalámbricos ha sufrido un aumento de oferta casi explosivo. De unos cuantos modelos casi experimentales, se ha pasado a decenas de empresas ofreciendo sus soluciones.

Para llevar acabo el proyecto que aquí se presenta se ha optado por usar los dispositivos de Crossbow[1], MICA2 y MICAz. La elección no es trivial, ya que se trata de unos dispositivos con mucha aceptación en la comunidad de desarrollo, lo que facilita en cierta medida la etapa de desarrollo.

2.2 Crossbow MICA2

Este dispositivo de 3ª generación ha sido diseñado por el fabricante especialmente para su uso en redes de sensores inalámbricas de bajo consumo.

Este modelo representa una mejora significativa sobre su antecesor, el modelo MICA. Sus principales características son:

- Transceptor multicanal en la banda de 868 y 916 Mhz.
- Soporta reprogramación inalámbrica.
- Amplia gama de placas sensoras y de adquisición de datos.
- Comunicación inalámbrica con otros nodos con capacidad de enrutamiento.

- Más de un año de duración de las baterías, usando los modos de ahorro de energía.

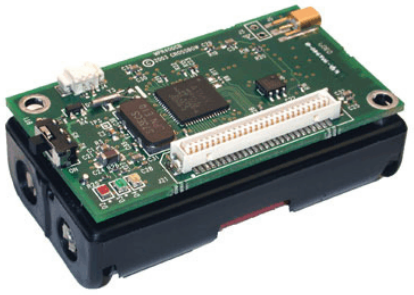

Figura 1 - MICA2, sin antena

Este dispositivo, está basado en la plataforma MPR400, que incorpora un microcontrolador Atmel ATmega128L de bajo consumo. Además puede ejecutar simultáneamente la pila de comunicaciones y el código de la aplicación instalada.

Dispone de 128KB de memoria flash para almacenar el programa, y otros 512KB adicionales de memoria flash para almacenar datos procedentes de las medidas tomadas. De manera similar, el MICA2 incluye un conversor analógico/digital, de 10 bits de 8 canales, para realizar las medidas.

También se dispone de un conector auxiliar, donde se pueden conectar las placas sensoras y de adquisición, o conectarlas a una de las placas programadoras.

La alimentación del dispositivo se puede realizar mediante 2 pilas de tipo AA de 1,5 Voltios (ya que viene incluido un porta-pilas junto al dispositivo), o bien usar el conector de alimentación externo instalado, con lo cual deberemos proporcionar entre 2,7 y 3,3 Voltios.

Finalmente, en la placa hay instalados 3 LED's, rojo, amarillo y verde, con el que poder realizar distintas señalizaciones de estado.

2.2.1 Chipcon CC1000

De las comunicaciones inalámbricas se encarga el microcontrolador Chipcon CC1000, pudiendo trabajar entre 300 y 1000 Mhz, pero que en el modelo particular que nos ocupa, trabaja, como se ha dicho antes, en la banda de 868 Mhz, o bien en la banda de 916 Mhz, proporcionando diversos canales (según la disponibilidad en cada

país).

Cuenta con una velocidad máxima de transmisión de 38,4 Kbps (76'8 Kbaudios), y una potencia de emisión de entre –20 a 5 dBm, lo cual puede llegar a proporcional un alcance en espacios abiertos de unos 150 metros, en condiciones óptimas.

Además, el CC1000 es capaz de realizar medidas de potencia (RSSI, *Received Signal Strength Indicator*) de la señal recibida a través de la antena (sea señal o ruido), con un rango de entre –105 y –50 dBm, con una precisión de ±6 dBm.

El CC1000 dispone de una patilla del encapsulado para dar salida a este valor, mediante un valor de voltaje inversamente proporcional al nivel de potencia de la señal.

Para hacer uso de este valor, al ser un dato analógico, ha de utilizarse el conversor analógico/digital del MICA2, y transformarlo al valor adecuado de potencia según la siguiente formula, la adecuada a la banda de transmisión usada en el proyecto:

$$P = -50V_{RSSI} - 45{,}5 \ (dBm)$$

Una gráfica típica entre la relación entre el voltaje en dicha patilla y la potencia de entrada es la siguiente:

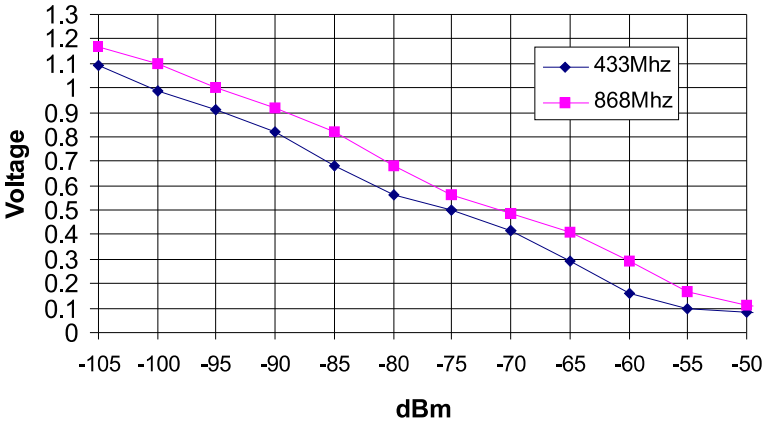

Figura 2 - Relación potencia frente a voltaje registrado

2.3 Crossbow MICAz

Este modelo, es la siguiente evolución en la gama MICA de Crossbow. Prácticamente hablamos de la misma plataforma que el modelo MICA2, pero en este caso, el microcontrolador encargado de la pila de comunicaciones es el Chipcon CC2420, que le otorga las

siguientes características:

- Banda de trabajo ISM (de 2,4 a 2,48 Ghz).
- Cumple el estándar de transceptor impuesto por la norma IEEE 802.15.4.
- 250 Kbps de velocidad máxima de transferencia.
- Capacidad para cifrado de datos integrada.

Figura 3 - MICAz, con antena

En los demás detalles, microcontrolador central, alimentación, etc..., nos encontramos con las mismas características. Es por tanto en el microcontrolador de radio, donde radican las diferencias.

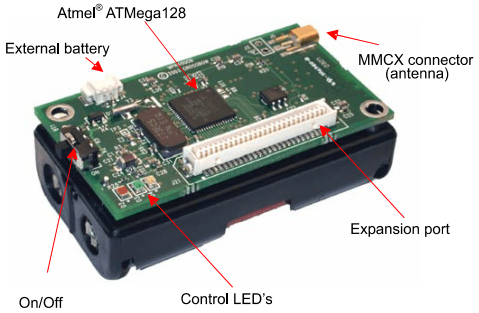

Figura 4 - Elementos principales en los dispositivos MICAx

2.3.1 Chipcon CC2420

El microcontrolador Chipcon CC2420 ha sido diseñado desde un principio teniendo en cuenta el estándar de IEEE 802.15.4, y por tanto cumple con los requisitos impuestos por éste.

Trabaja en la banda ISM 2,4 Ghz, y cumple con multitud de estándares internacionales, y permite transmitir hasta una velocidad

efectiva de 250 Kbps.

El integrado también es capaz de modificar la potencia de transmisión e igualmente, el CC2420 es capaz de realizar una medida de la potencia de la señal recibida, al igual que el CC1000, pero al contrario que éste último, este dato no se entrega mediante una señal analógica. Esta medida se hace por un periodo promediado de 128 μs, en un rango dinámico de 100 dB y con una precisión de ±6 dBm.

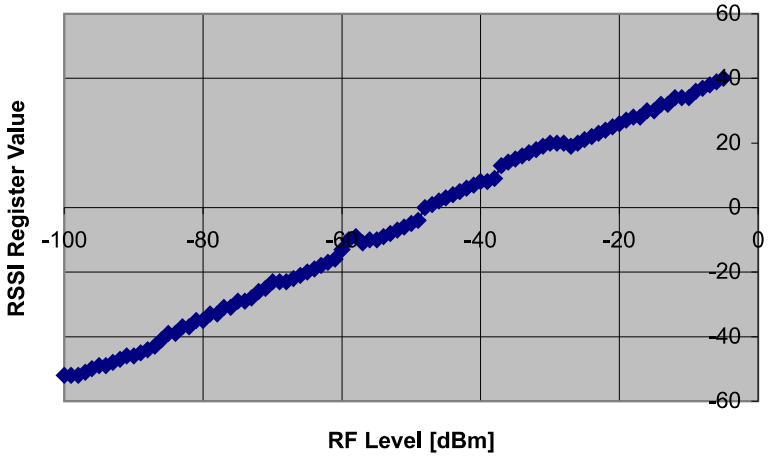

Figura 5 - Relación potencia frente al valor registrado

Como se observa en la figura, la lectura tiene un comportamiento muy lineal. El valor digital entregado al microcontrolador central, no es usable directamente, si no que hay que aplicarle un simple factor correctivo:

$$P = RSSI - Offset$$

Este valor puede ser hallado empíricamente, y se puede aproximar por −45 dB.

2.4 Resumen comparativo

La siguiente tabla compara las características de los componentes de radiofrecuencia de ambos dispositivos (que es lo que los diferencia):

	CC2420	CC1000
Rango	2400 – 2483.5 MHz	300 – 1000 Mhz
Velocidad	250 kbps	38,4 kbps
Consumo	RX: 19.7 mA TX: 17.4 mA	RX: 7.4 mA TX: 10.4 mA
Potencia	−24 a 0 dBm	−20 a 5 dBm
Sensibilidad	−94 dBm	−110 dBm
Control	Nivel de paquete	Nivel de bit

Como se verá con posterioridad, la diferencia en el control de lo que se transmite, determinará las capacidades de los protocolos que se implementen. También es destacable la diferencia de consumos, donde el integrado más reciente, el CC2420 consume significativamente más energía, dato que habrá que tener en cuenta.

2.5 Crossbow MIB510

Este dispositivo es el encargado de conectar los dispositivos a un PC, ya sea para programarlos, o bien para extraer datos de ellos.

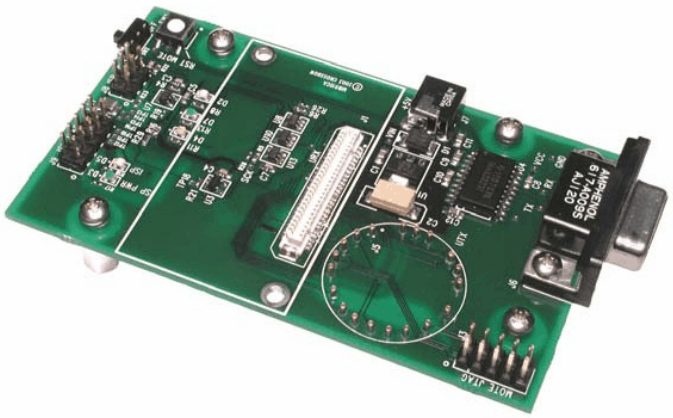

Figura 6 - MIB510

Este modelo, dispone de una conexión serie, un conector auxiliar en la parte superior, para conectar el dispositivo inalámbrico, y otro en la parte inferior, para posibilitar el uso de placas de expansión

conectadas al ordenador. Con este programador, también se pueden programar otro de los dispositivos de la gama MICA, los MICADOT, alimentados por una pila de botón, y más pequeños que los usados en este proyecto.

En la placa, se han replicado los LED's que estaban instalados en los dispositivos, ya que, al conectarlo a la placa programadora, estos quedan prácticamente ocultos. Se incluyen otros 2 LED's para indicar si está correctamente alimentado (LED verde) y si está siendo programado (LED rojo).

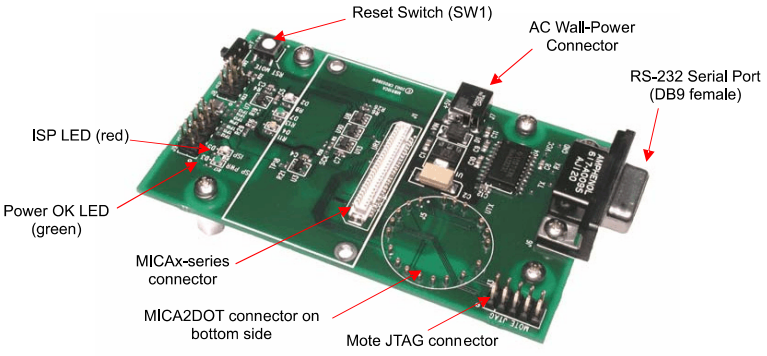

Figura 7 - Elementos principales de MIB510

Es importante resaltar que, para programar o enviar/recibir datos con esta placa, si no se conecta una fuente de alimentación externa, se tendrá que hacer uso de la propia alimentación del dispositivo que conectemos. Es por tanto muy recomendable usar una fuente de alimentación externa siempre que se use este programador.

2.6 Crossbow MIB520

El otro dispositivo usado para la programación de los dispositivos, se trata de un modelo posterior al anteriormente comentado, con una conexión USB en vez de la conexión serie.

Figura 8 - MIB520

Usando el puerto USB como fuente de alimentación, se quita la necesidad de tener que usar una fuente externa. Sin embargo, con este programador, no se dispone de un conector auxiliar para usar placas sensoras, y tampoco se pueden programar MICADOT. Así mismo, también incluye los dos grupos de LED's que tiene el MIB510.

Ya sea en el sistema operativo Microsoft Windows XP o alguna distribución de Linux, proporciona dos interfaces serie, uno para programar el dispositivo, y otro para realizar intercambio de datos con él. Con esto, se pretende mantener la compatibilidad con el MIB510, al mostrarse como un dispositivo serie más.

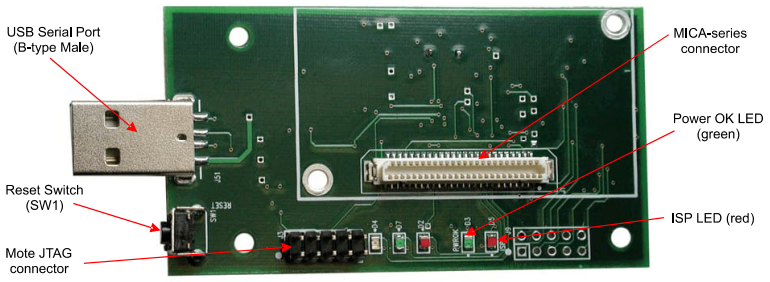

Figura 9 - Elementos principales de MIB520

Capítulo 3

Software

3.1 Introducción

En esta sección vamos a presentar las herramientas de software que han sido usadas en el desarrollo del proyecto, y sus principales características.

La elección de estas herramientas, nuevamente, no es trivial. La elección hecha en hardware invitaba a escoger de esta manera. Existen otros sistemas operativos disponibles para sensores inalámbricos, pero, TinyOS[2] está sobradamente establecido como la mejor elección, por capacidad, estabilidad, y software disponible.

3.2 TinyOS

TinyOS es un sistema operativo de código abierto especialmente diseñado para sistemas embebidos, escrito en el lenguaje de programación nesC[3] (que se trata en el siguiente punto), optimizado para entornos con poca memoria y que está compuesto por una serie de tareas y procesos cooperativos.

Los programas escritos para TinyOS se construyen a base de componentes, que se interconectan entre sí mediante interfaces. De esta manera, hace posible la reutilización de estos componentes de una manera sencilla, lo que acorta el tiempo dedicado a la programación.

Es un sistema operativo conducido por eventos, es decir, que funciona a partir de interrupciones en el sistema, los cuales harán

llamadas a funciones. Está diseñado para incorporar nuevas innovaciones rápidamente y para funcionar bajo las importantes restricciones de memoria que se dan en las redes de sensores.

Hay que resaltar, que al ser un sistema operativo embebido, éste no reside como una parte separada de la aplicación a la que sirve de interfaz. Es decir, a la hora de desarrollar una aplicación, el archivo binario resultante de la compilación incluye tanto la aplicación desarrollada como el sistema TinyOS.

El entorno de desarrollo de TinyOS soporta directamente la programación de diferentes microprocesadores y permite programar cada tipo con un único identificador para diferenciarlo, o lo que es lo mismo se puede compilar para diferentes plataformas cambiando una serie de parámetros en tiempo de compilación.

TinyOS fue inicialmente desarrollado por un consorcio liderado por la Universidad de California en Berkley, en cooperación con Intel Research, y desde entonces ha seguido creciendo bajo un consorcio internacional, la Alianza TinyOS (*TinyOS Alliance*).

3.3 nesC

El lenguaje de programación nesC es el lenguaje empleado para la creación de aplicaciones en los MICA2 y MICAz. Básicamente se trata de lenguaje C, con una serie de sentencias para el precompilador, scripts y palabras clave, para dar cabida a los nuevos aspectos que introduce este lenguaje.

nesC es un lenguaje cómodo para la programación en sensores, ya que está orientado a componentes. Con esto, se consigue una filosofía que permite abstraer al programador de detalles de bajo nivel presentes en el sistema operativo.

La idea subyacente en la orientación a componentes es que el propio sistema operativo, junto a los fabricantes de los nodos sensores, proporcione ciertos componentes ya implementados que ofrezcan al programador una gran cantidad de funciones y utilidades que le permitan centrarse únicamente en la funcionalidad que desea implementar en el dispositivo, sin necesidad de tener que preocuparse por otros aspectos secundarios.

Por otro lado, nesC combina ciertos aspectos de la orientación a objetos, en el sentido de que se basa en una programación orientada a interfaces y a eventos, de manera que posee un manejador de even-

tos propio de este tipo de lenguajes.

Todos los componentes que ofrece de forma intrínseca el sistema operativo se van a denominar, a partir de ahora, componentes primitivos y los componentes proporcionados por terceros se van a denominar componentes complejos.

Cuando se dice que es un lenguaje orientado a objetos, se quiere decir que todos los componentes, tanto primitivos como complejos, proporcionan unas interfaces y si un programador desea utilizar un componente, la forma de hacerlo es usando dichas interfaces, pero recordemos que son interfaces en el sentido de orientación a objetos, por lo que en un momento dado se podría cambiar un componente por otro siempre y cuando este otro proporcionase la misma interfaz y dicho cambio no afectase al código de la implementación de la aplicación.

Por lo tanto, vamos a tener dos partes diferenciadas, como mínimo, en cada componente, la parte de implementación que estará programada hacia componentes y la parte de configuración que permitirá decidir que componentes son los que se utilizan para proporcionar dichas interfaces a mi componente, lo que se denomina *wiring*.

Como la forma de programar no es del todo secuencial, se recurre a la orientación a eventos, de manera que se programan las acciones que se desean realizar cuando se produzca un evento. Cada interfaz puede tener métodos y eventos y todos los componentes que implementen una interfaz deberán implementar todos sus métodos y los que las usen sus eventos.

A modo de resumen, mencionamos los principales aspectos que el modelo de programación nesC ofrece y que deben ser entendidos para el entendimiento del diseño con TinyOS:

- nesC se compone de interfaces y componentes.
- Una interfaz puede ser usada o puede ser provista.
- Los componentes son módulos o configuraciones.
- Una aplicación se verá representada como un conjunto de componentes, agrupados y relacionados entre sí
- Las interfaces se utilizan para operaciones que describen la interacción bidireccional; el proveedor de la interfaz debe implementar comandos, mientras que el usuario de la interfaz debe implementar eventos.
- Existen dos tipos de componentes, módulos y configu-

raciones:

- Los módulos implementan las especificaciones de un componente.
- Las configuraciones se encargan de unir o conectar (el *wiring* mencionado anteriormente) diferentes componentes en función de sus interfaces.

3.3.1 Estructura de un componente

Un componente está formado por tres partes: `configuration`, `implementation` y `module`. Todos los componentes presentan estas partes aunque no necesariamente deben estar implementadas, es decir, puede que un componente sólo tenga una implementación y una configuración pero carezca de un módulo.

El estándar de TinyOS determina que las secciones `configuration` e `implementation` han de ir en un fichero que recibirá el nombre del componente con la extensión `.nc` y la sección `module` deberá ir en otro fichero que recibirá el nombre del componente concatenado con una `M`, teniendo también la extensión `.nc`.

Una buena costumbre que facilita la organización y claridad del código, consiste en crear un fichero de cabecera (con la típica extensión `.h`) que contenga todas las enumeraciones, registros o tipos de datos creados por el programador para la aplicación desarrollada.

La forma de ligar dicho fichero con los otros dos es utilizando al principio de los ficheros la directiva `includes header;` aunque como mención especial decir que si nos fijamos mejor en esta directiva se puede ver que no se incorpora la extensión `.h` en la misma (al igual que ocurre en C++).

De este modo, para un módulo ejemplo que podamos llamar Test, en el fichero `Test.nc`:

`configuration`: Solo contiene la declaración de las interfaces que va a proveer o usar.

```
configuration Ejemplo
{
    provides {
        interface InterfazProvistaEjemplo;
    }
    uses {
        interface InterfazUsadaEjemplo;
    }
}
```

`implementation` (`configuration`): Comúnmente, se le denomina *wiring* y no se corresponde a lo que, en una primera impresión se podría pensar que es la implementación. El código de nuestra aplicación puede utilizar (`uses`) interfaces y que éstas sean proporcionadas por otro componente. Aquí es donde se especifica qué componente implementará qué interfaces.

```
implementation
{
    components EjemploM;

    StdControl = EjemploM.StdControl;
}
```

Mientras que en el fichero `TestM.nc` tendremos:

`module`: Donde haremos una declaración de los modulos que se usan y que se proveen.

```
module EjemploM
{
    provides {
        interface InterfazProvistaEjemplo;
    }
    uses {
        interface InterfazUsadaEjemplo;
    }
}
```

`implementation` (`module`): Esta parte es la dedicada a realizar toda la implementación de código nesC, donde se crearan las funciones necesarias, y se implementarán aquellas que nos obliga las interfaces que usamos.

```
implementation
{
    command result_t StdControl.init()
    {
        ...
    }

    command result_t StdControl.start()
    {
        ...
    }

    command result_t StdControl.stop()
    {
        ...
    }
}
```

Las interfaces se incorporan haciendo uso de las siguientes palabras reservadas:

`provides`: especifica las interfaces que va a proporcionar nuestro componente. Es necesario que en el código que se encuentra dentro de `implementation` estén implementados (valga la redundancia) todos los métodos que especifique la interfaz.

`uses`: especifica las interfaces que utiliza nuestro componente. Estas interfaces estarán proporcionadas (`provides`) por otro componente que habrá que especificar en el fichero de conexiones (*wiring*). Cuando usemos una interfaz podremos llamar a sus métodos pero tendremos que implementar los eventos que puedan producirse.

3.3.2 Tipos de datos

Los tipos que pueden utilizarse son todos aquellos que proporciona C estándar (incluidos los tipos del estándar C99, por ejemplo `uint8_t`, `uint16_t`, `uint32_t`, `uint64_t`), más otros específicos de nesC (`bool`, `result_t`).

En nesC, es posible la utilización de memoria dinámica aunque debido a su complejidad, no es muy recomendable a no ser que sea absolutamente necesario. Para gestionar la memoria dinámica existe un componente especial denominado `MemAlloc`.

3.3.3 Tipos de funciones

Debido a que nesC es un dialecto del lenguaje de programación C, nesC ha heredado una serie de funciones clásicas que tienen la misma semántica que en C y con la misma manera de invocarlas. Pero existen otros tipos de funciones en nesC: `task`, `event` y `command`.

Las funciones `command` o comandos son funciones en el sentido más clásico, y que se ejecutan de forma síncrona (cuando se llaman, se ejecutan inmediatamente). La forma de llamar estas funciones es:

```
call interfaz.nombreFuncion;
```

Las funciones `task` o tareas se ejecutan concurrentemente en la aplicación, al igual que ocurre con los hilos o threads. Inmediatamente tras su invocación, continúa la ejecución del programa invocador.

```
post interfaz.nombreTarea;
```

Las funciones `event` o eventos son llamadas cuando se levanta una señal en el sistema. Están basadas en la filosofía de la programación orientada a eventos, de manera que cuando el componente recibe un evento, se realizará la invocación de dicha función. Cuando un componente quiere lanzar un evento en algún momento del código debe realizar una llamada con la siguiente sintaxis:

```
signal interfaz.nombreEvento;
```

Además en el caso de tratarse de interfaces con parámetros es posible utilizar la variante:

```
signal interfaz.nombreEvento[parámetro];
```

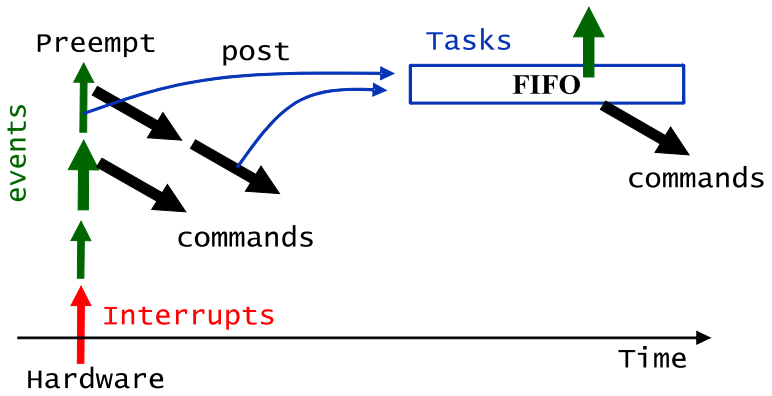

Figura 10 - Esquema de ejecución de procesos

Existen distintos tipos de funciones además de las nombradas, las más destacables son las funciones asíncronas; estas funciones no tienen por que realizarse de forma inmediata y, por tanto, cuando usan una variable global, se ha de indicar de forma explícita para poder realizar una exclusión mutua de la memoria y no perder ningún valor.

Este tipo de ejecución, por lo general, viene dado cuando la ejecución de uno de estos tipos de función viene determinada por el levantamiento de una señal hardware, por ejemplo, que la temperatura ya se encuentre preparada para ser leída. La forma de indicarlo es anteponiendo la palabra reservada `norace` delante de la declaración de la variable.

3.3.4 Otras funcionalidades

Debido a que se permite cierto tipo de programación recurrente mediante la invocación de tareas, el lenguaje de programación permite construir una agrupación de sentencias que aseguren que, mientras que se esté realizando dicha transacción, las variables implicadas no van a ser modificadas por otro hilo de ejecución. Esto se consigue mediante la palabra reservada `atomic`.

```
atomic {
    ...
}
```

Igualmente, debido a que ciertas interfaces parametrizadas no pueden ser llamadas dos veces con el mismo parámetro, existe una función definida que proporciona un valor único para una constante, es decir, al llamar a la función muchas veces con el mismo parámetro, nos aseguramos que cada vez va a devolver un valor distinto. La función que permite esto es:

```
uint_16 unique(string param);
```

3.3.5 Componentes primitivos de TinyOS

Los tipos de componentes que podemos encontrar en TinyOS pueden ser primitivos o compuestos (proporcionados por una librería o una aplicación). Los principales componentes primitivos son:

Componente `main`: representa el cuerpo main al estilo de C y necesita de la interfaz `StdControl` para su funcionamiento. Dicha interfaz será proporcionada (`provides`) por el componente que quiera convertirse en una aplicación.

Interfaz `StdControl`: es una interfaz especial que deben proporcionar todos los componentes que pretendan ser una aplicación y, por tanto, sean ejecutables en un sensor. Dicha interfaz obliga a implementar los siguientes métodos:

```
command result_t init();
```

Se ejecuta cuando el sensor arranca. Aquí es donde se inicializan las variables globales y se llama a los métodos `init()` de los componentes que se vayan a utilizar (no todos los componenentes usados (`uses`), sino los que sean necesarios).

```
command result_t start();
```

Se ejecuta después del método `init()` y cuando el dispositivo

pasa de off a on. Es aquí donde se arrancan los temporizadores, se llama a los métodos `start()` de los componentes utilizados (sólo los necesarios).

```
command result_t stop();
```

Se ejecuta cuando el sensor se apaga o se suspende (estado `idle`). En esta parte del código se realizan las llamadas a los métodos `stop()` de los componentes arrancados.

3.4 XubunTOS

XubunTOS[4] es una distribución de Linux, basada en Xubuntu, está a su vez basada en Ubuntu, pero contando con XFCE como entorno de ventanas, en vez del más pesado Gnome. Gracias a esto, y al ser una distribución Live CD (es decir, una distribución que se puede ejecutar desde el CD, sin necesidad de instalarla), se obtiene más espacio disponible en el CD.

Figura 11 - XubuntTOS, captura del escritorio

En esta distribución, como su nombre nos quiere indicar, se incluye, para poder empezar a usar desde el primer momento, todas las herramientas necesarias para programar con el entorno TinyOS. Incluye tanto las versiones 1.x como la 2.x de TinyOS, y una serie de scripts para poder cambiar entre estas dos versiones de manera sencilla; facilidades para actualizar las fuentes de ambas ramas de TinyOS fácilmente; enlaces simbólicos a los dispositivos de programa-

ción USB; y, en definitiva, toda la comodidad de poder trabajar en un entorno Linux.

Capítulo 4

Protocolos

4.1 Introducción

En la actualidad existe una gran cantidad de protocolos desarrollados para TinyOS, más para TinyOS 1.x que para la más reciente versión 2.x, y por esto en el desarrollo del proyecto se ha optado por el uso de la rama 1.x.

En el siguiente diagrama, extraído de la pagina *MAC Alphabet Soup*[5], se muestran los diversos protocolos creados específicamente para redes de sensores, ordenados cronológicamente y relacionados entre sí por el tipo que representan.

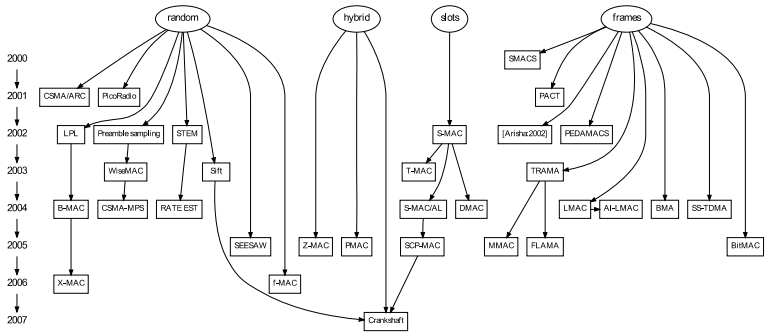

Figura 12 - Evolución de los algoritmos para WSN

En él, podemos encontrar los protocolos que vamos a pasar a describir a continuación, S–MAC, SCP–MAC, B–MAC, etc...

4.2 S–MAC

El protocolo S–MAC[6] se trata de un protocolo diseñado especialmente para redes de sensores, y poniendo especial énfasis en el cuidado del consumo, ya que éste es un punto crítico en todas las redes de sensores.

Una primera implementación del protocolo fue llevada acabo en la *University of Southern California*, aunque la implementación más usada es la implementada por el profesor de la misma universidad, Wei Ye, y que se puede obtener desde su pagina dedicada al protocolo S–MAC o en el repositorio de código contribuido de TinyOS.

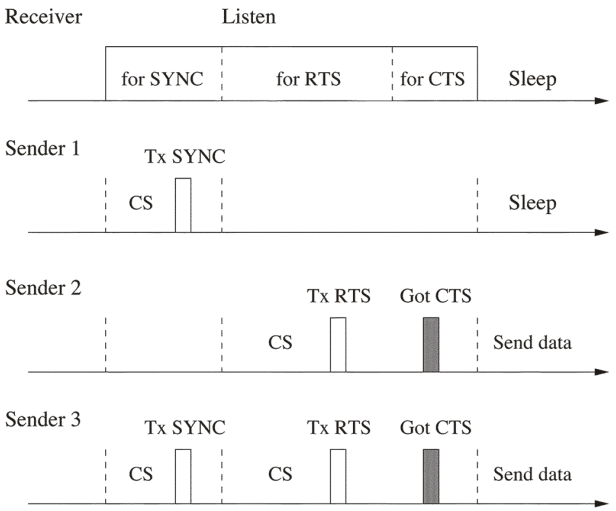

Figura 13 - Esquema de funcionamiento básico de S–MAC

La principal característica del protocolo, y a la vez, la base de su bajo consumo, es el hecho de que el transceptor del dispositivo no está escuchando en todo momento. Varios estudios han puesto de manifiesto que, un dispositivo mientras está escuchando el medio, consume casi tanto como cuando está recibiendo datos. Esto es lo que se denomina una estrategia de *Low Power Listening* (LPL).

Para evitar esta situación, esto es, que los dispositivos estén escuchando constantemente, S–MAC alterna periodos de escucha con periodos de inactividad total, siendo los periodos de escucha mucho más cortos que los periodos de inactividad.

Esta solución, sin embargo, también añade ciertas complicaciones. Es evidente entonces, que todos los dispositivos que se quieran comunicar directamente, deberán acertar a transmitir en el perio-

do en el que el nodo receptor esté escuchando.

Es por esto, que el propio protocolo incluye un procedimiento para sincronizar a los nodos, de manera que un grupo que quiera comunicarse directamente entre ellos, sigan todos la misma pauta.

Además, el protocolo hace uso de un mecanismo RTS/CTS con ACK, para evitar los problemas típicos de las comunicaciones inalámbricas.

Hay que destacar sobre este protocolo, que su implementación se ha hecho teniendo en mente el microcontrolador de radio Chipcon CC1000, por lo tanto, solo los modelos que montan éste pueden usar este protocolo. Ha habido intentos de transportarlos a otros microcontroladores, en particular el CC2420, sin mucho éxito por parte de particulares.

4.2.1 Componentes nesC de S–MAC

En este apartado vamos a dar una breve explicación de los componentes que forman parte del protocolo en nesC, para su mejor comprensión. Empecemos por mostrar un diagrama de los principales componentes:

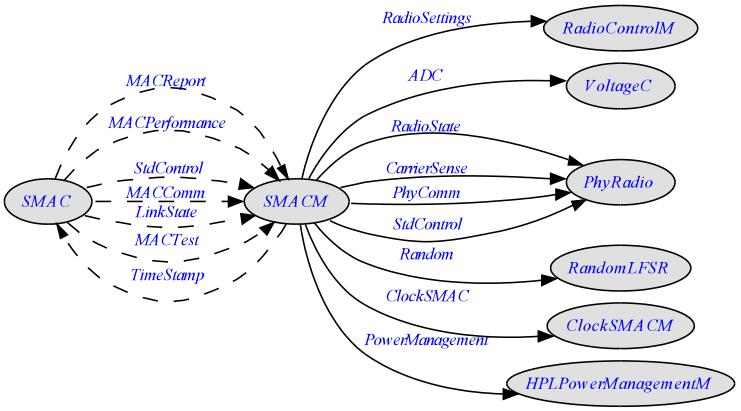

Figura 14 - Diagrama de componentes S–MAC

El componente configuración `SMAC` es al que se conecta (*wire*) la aplicación que se desarrolla. Ésta, provee varias interfaces, el principal, la interfaz `MACComm`, que es el que proporciona la funcionalidad para transmitir paquetes, con las siguientes funciones:

```
result_t broadcastMsg(void *msg, uint8_t length)
result_t unicastMsg(void *msg, uint8_t length, uint16_t toAddr,
uint8_t numFrags)
```

Igualmente, esta interfaz contiene varios eventos definidos que nos avisarán cuando se produzca una recepción de un paquete, o el resultado del intento de transmisión:

```
result_t broadcastDone(void *msg)
result_t unicastDone(void *msg, uint8_t txFragCount)
void rxMsgDone(void *msg)
```

El componente módulo SMACM es donde se encuentra el código principal que controla el comportamiento del protocolo, como por ejemplo el mecanismo RTS/CTS y otras funcionalidades.

El componente SMAC representa la parte del protocolo equivalente a la capa de transporte y red en la pila OSI, a grandes rasgos, y el componente PhyRadio representaría el nivel físico, y por lo tanto es el encargado de tratar la recepción y envío de mensajes a bajo nivel, y también de otros aspectos relativos al microcontrolador de radio, como por ejemplo, el de tomar muestras de la potencia de los mensajes recibidos, o la potencia de ruido en periodos de inactividad.

4.3 B–MAC

B–MAC[7], desarrollado en la *University of California*, por, entre otros, Joseph Polastre, representa una simplificación en muchos aspectos sobre S–MAC.

En primer lugar, el concepto de periodos alternados de actividad/inactividad se mantiene. Sin embargo, el aspecto de sincronización desaparece, aunque todos los nodos escuchan el medio con igual frecuencia, éstas no están sincronizadas.

Sin tener una sincronización entre los nodos, ha de recurrir a otra estrategia para llevar a cabo una comunicación con éxito. En este caso, se opta por el siguiente mecanismo: Cuando un nodo quiere realizar una transmisión, emite un preámbulo antes de la información a transmitir.

Este preámbulo, es de mayor longitud (en el tiempo) que el periodo entre escuchas de los nodos, de manera que, cualquier nodo tomará muestras del medio mientras este preámbulo es emitido, y al detectarlo, quedarán a la espera de recibir la información que se emitirá después de este preámbulo.

Al contrario que S–MAC, B–MAC no usa tramas de control previas a la comunicación (si no contamos al preámbulo como tal), solo hace uso de ACK's (que también puede ser desactivado).

Igual que en el caso anterior, las características de este protocolo han sido diseñadas con el microcontrolador CC1000 en mente, por lo que su implementación en otros microcontroladores, como el CC2420 no disfruta de las mismas funcionalidad. En particular, la implementación para CC2420, que es el protocolo estándar usado en TinyOS no cuenta con los periodos de actividad/inactividad mencionados, dado que el CC2420 está orientado a transmisión de paquetes, y no se tiene un control total sobre la longitud ni el contenido de lo que se transmite, haciendo casi imposible la emisión de preámbulos largos.

4.3.1 Componentes nesC de B–MAC

La conexión del modulo aplicación al protocolo B–MAC se hace a través del componente `GenericComm`:

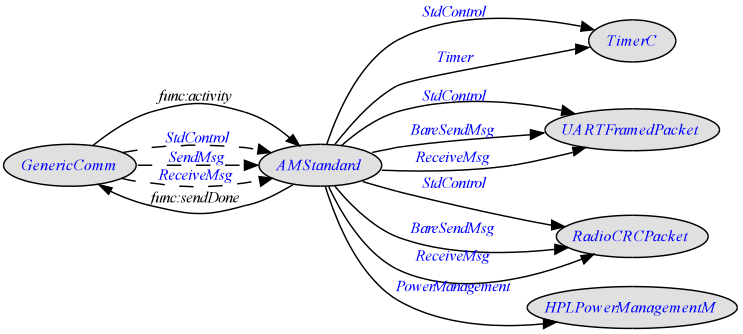

Figura 15 - Diagrama de componentes B–MAC

Este componente provee las interfaces `SendMsg` y `ReceiveMsg`, a parte de la interfaz `StdControl` (que siempre está presente). Estas dos interfaces son las encargadas de proporcionar las funciones apropiadas para el envío de mensajes.

La interfaz `SendMsg` provee una función para el envío de mensajes con sus parámetros asociados (dirección de destino, tamaño), y a su vez un evento que ha de ser implementado en el modulo aplicación, el cual se lanza después de intentar enviar un mensaje, y que informa del éxito o fracaso de éste:

```
result_t send(uint16_t address, uint8_t length, TOS_MsgPtr msg);
result_t sendDone(TOS_MsgPtr msg, result_t success);
```

La interfaz `ReceiveMsg` consta únicamente de un evento que se lanzara en el modulo aplicación, una vez se halla recibido un men-

saje:

```
TOS_MsgPtr receive(TOS_MsgPtr m);
```

El componente `GenericComm` se conecta al componente `AMStandard`, que es donde está realizada parte de la implementación del protocolo B–MAC. A este componente se conecta el componente `RadioCRCPacket`, que ya es dependiente del hardware, es decir, su implementación puede variar según el dispositivo que se use en cada caso.

4.4 SCP–MAC

El creador del protocolo S–MAC es el responsable de ésta nueva propuesta, Scheduled Channel Polling (SCP)[8], que toma elementos tanto de S–MAC como de B–MAC.

En primer lugar, se toma la idea usada en B–MAC, es decir, usar un preámbulo para advertir de una próxima transmisión. Sin embargo, también se mantiene una de las características de S–MAC, la sincronización entre nodos.

Con esta combinación se pretende minimizar el tiempo, tanto el que el emisor está emitiendo el preámbulo, como la de que el receptor esté esperando para recibir los datos.

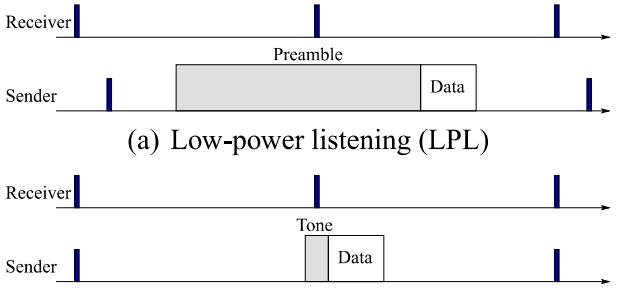

Figura 16 - Comparativa básica B–MAC (a) y SCP–MAC (b)

SCP–MAC en principio no hace uso del esquema RTS/CTS existente en S–MAC, pero éste puede ser activado en tiempo de compilación. Además, el protocolo está estructurado en varias capas, mas concretamente 4, que además son 4 componentes nesC, separando así las competencias de éstas:

- Capa física

- Capa CSMA básica
- Capa LPL
- Capa SCP

Este protocolo también ha sido creado tomando el microcontrolador CC1000 en mente, y por lo tanto, aunque los propios creadores de este protocolo han hecho una implementación para el CC2420, ésta no tiene las mismas funcionalidades que su versión para CC1000.

4.4.1 Componentes nesC de SCP–MAC

El componente que se conecta usualmente al módulo aplicación es el componente `Scp`, que es el que proporciona todas las funcionalidades descritas con anterioridad:

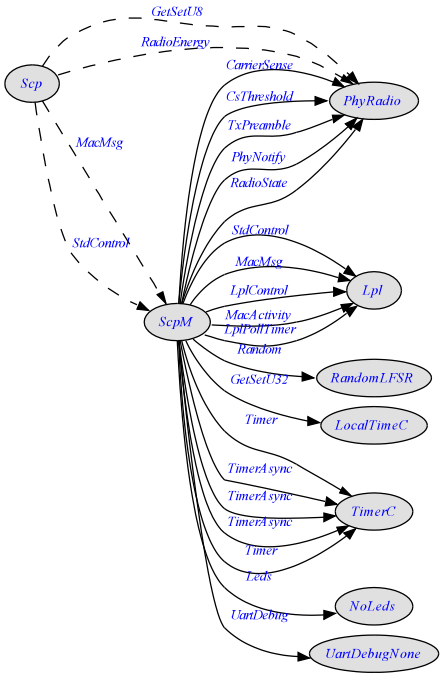

Figura 17 - Diagrama de componentes SCP–MAC

Este módulo, entre otras, provee la interfaz `MacMsg`, que es la encargada de dar la funcionalidad para enviar y recibir mensajes, mediante la función `send`, y el evento `receiveDone`:

```
result_t send (void *msg, uint8_t length, uint16_t toAddr);
result_t sendCancel (void *msg);
void sendDone (void *msg, result_t result);
void receiveDone (void *msg);
```

La implementación se encuentra, como es habitual, en el modulo `ScpM`, que a su vez se conecta, entre otros, al componente `Lpl`. Como se ha dicho anteriormente, el protocolo SCP–MAC está fuertemente estructurado en capas, lo que posibilita la desactivación de ciertas capas, para prescindir de ciertas funcionalidades. Es posible usar solo parte de esta pila, y hacer uso directamente del componente `Lpl`, `Csma` o `PhyRadio` para el envío y recepción de mensajes.

4.5 Otros protocolos

De entre todos los protocolos existentes actualmente, no son muchos los que tienen el código disponible para hacer pruebas con él. Aquí vamos a comentar de manera breve algunas propuestas interesantes, algunas de las cuales, o bien no tienen el código fuente disponible, o bien están diseñadas para plataformas no disponibles.

4.5.1 B–MAC+

Como su nombre parece indicar, B–MAC+[9] se trata de una mejora sobre B–MAC. Esta mejora está especialmente enfocada a hacer posible su uso en dispositivos que usen el microcontrolador CC2420.

Para solventar el problema de este chip, que no permite la transmisión de preámbulos de tamaño arbitrario, este protocolo propone dividir dicho preámbulo en varios. Cada uno de estas divisiones contendrá información especificando cuanto falta para que se realice la transmisión. En resumen, cada división representa una cuenta atrás.

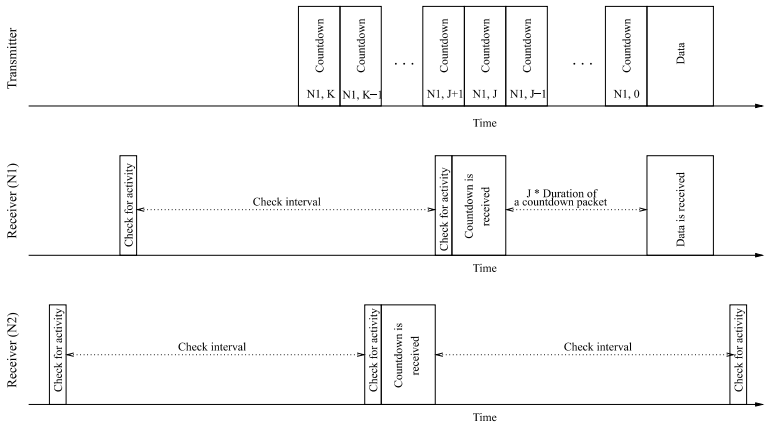

Figura 18 - Esquema de funcionamiento básico de B–MAC+

De este modo, el receptor, puede, según el numero de cuenta atrás que reciba, desconectar la radio, hasta poco antes de recibir el paquete.

4.5.2 X–MAC

Este protocolo[10], al igual que el B–MAC+, tiene como objetivo proporcionar un sistema de poder aprovechar el concepto de LPL a los microcontroladores como el CC2420, que eran problemáticos con los preámbulos largos.

En este caso, y de forma parecida a B–MAC+, un preámbulo largo se divide en secuencias más cortas. Sin embargo, en X–MAC, entre cada uno de estos preámbulos, hay el espacio suficiente para que un receptor envíe una trama de control, denominada *early* ACK (eACK), informando al emisor que está listo para recibir inmediatamente.

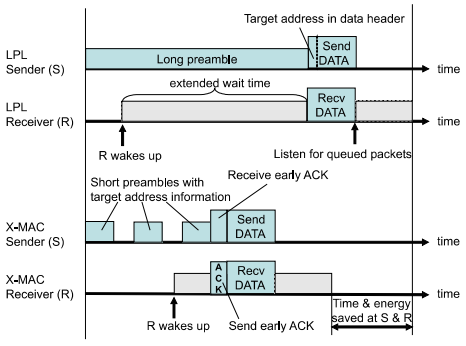

Figura 19 - Esquema comparativo B–MAC frente a X–MAC

Con esta estrategia, no solo se evita mantener al receptor escuchando, como ya se hacia en B–MAC y B–MAC+, si no que además se acorta el tiempo que el emisor va a estar emitiendo el preámbulo.

Sin embargo, y a modo de conclusión, estos dos protocolos, que presentan una cierta solución a un problema de implementación de B–MAC sobre algunos microcontroladores de radio (entre otros, el Chipcon CC2420), acaban usando técnicas que, planteadas de otro modo, no son mas que un control de flujo RTS/CTS, con pequeñas variantes.

Capítulo 5

Arquitectura

5.1 Introducción

Como se ha visto en el capítulo anterior, los protocolos explicados, alguno como B–MAC es de uso intensivo en la comunidad, por haber alcanzado un grado suficiente de confiabilidad; en su totalidad recurren a la misma estrategia a la hora de ahorrar energía en el uso de la radio.

Este mecanismo o, mejor dicho, concepto, denominado *Low Power Listening* (LPL), se basa en minimizar el tiempo que el dispositivo está escuchando el medio, a la espera de recibir datos. Desde esta premisa, cada protocolo la adapta de diversas formas, ya sea mediante la sincronización de los momentos de escucha, o la emisión de preámbulos.

Sin embargo, estos protocolos ignoran un aspecto importante de la transmisión que influye también en el consumo. Tan importante como el tiempo que se está usando el transceptor, es con cuanta potencia se emite. Es aquí donde entra el juego el concepto de *Transmission Power Control* (TPC).

Es evidente que, ante cualquier posible distribución de nodos, es más que probable que las distancias entre ellos no sean todas iguales, lo que hace posible que para los nodos más cercanos, la potencia de transmisión sea más baja que para aquellos que están más lejanos.

Pero antes de que esta transmisión se lleve acabo, un periodo

previo de negociación debe existir para que entre esos dos nodos se establezca una potencia de transmisión adecuada. Por esto, este mecanismo encaja perfectamente con protocolos que ya usan una negociación previa a la transmisión, como S–MAC, que usa un mecanismo RTS/CTS.

5.2 Planteamiento detallado

El sistema desarrollado en este proyecto para el control de potencia podemos dividirla en dos únicas fases:

- En la primera fase, se establece una comunicación inicial con la que se determina la potencia de emisión con la que ha de transmitirse.
- En la segunda fase, se lleva acabo la transmisión a la potencia deseada.

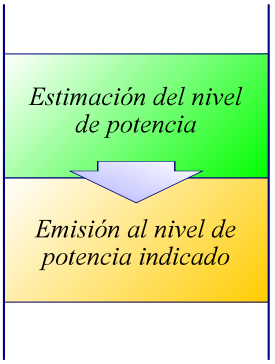

Figura 20 - Fases del mecanismo

5.2.1 Estimación de potencia

Para la estima de la potencia de emisión necesaria se ha de determinar cual es el valor de potencia que nos asegure una correcta recepción. Este valor se puede determinar a partir de los siguientes datos:

- Perdidas en el medio
- Nivel de ruido en el receptor
- Margen de recepción sobre el ruido

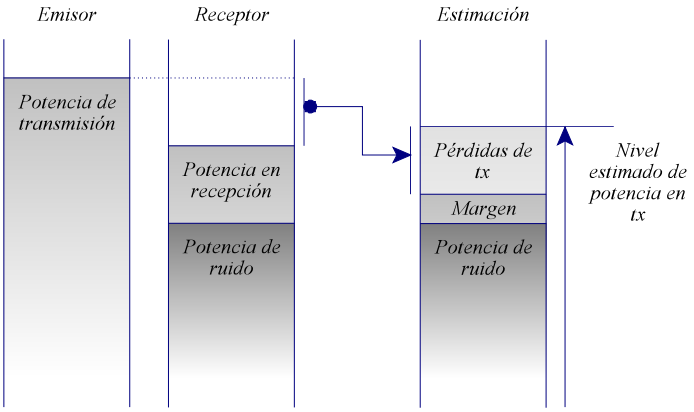

Figura 21 - Esquema seguido para el calculo de la potencia necesaria para transmitir

Las pérdidas en el medio son fácilmente calculables en el receptor, conociendo la potencia a la que se emitió una trama en concreto, y a que potencia se recibió. El primero de estos datos, es proporcionado por el emisor, que tiene conocimiento en todo momento de la potencia que usa para emitir. El segundo, la potencia recibida, se puede obtener del microcontrolador de radio.

Igualmente, el nivel de ruido se obtiene con ayuda del microcontrolador de radio.

El margen de recepción, es un valor que ha de establecerse, y que según las características de los dispositivos puede variar, pero siempre será aquel que nos asegure una recepción correcta en el nodo receptor.

5.2.2 Flujo de datos

El modelo de pérdidas, puede parecer un dato trivial, pero obtenerlo implica un problema dado que, el emisor, no puede saber de antemano el modelo entre el emisor y el receptor, y el emisor tampoco puede obtenerla por sí solo. Igualmente, el nivel de ruido ha de tomarse en el receptor.

Por lo tanto, antes de la transmisión de datos con el receptor, es necesario un intercambio donde se pueda establecer el nivel de potencia necesario.

Con este detalle presente, podemos establecer lo siguiente. El emisor conoce la potencia a la que está emitiendo en cada momento, mientras que el receptor es capaz de dar una magnitud de potencia de la señal recibida y del ruido. Cada una de las partes tiene una parte de la información.

En este punto se puede optar por dos opciones. O bien el emisor transmite al receptor la potencia a la que está emitiendo en esa primera comunicación, y entonces el receptor sería capaz de realizar los cálculos pertinentes, indicando al emisor después de haber determinado este dato a qué potencia seria necesario transmitir; o por el contrario, el receptor envía los datos de las perdidas y ruido de vuelta al emisor para que éste haga los cálculos.

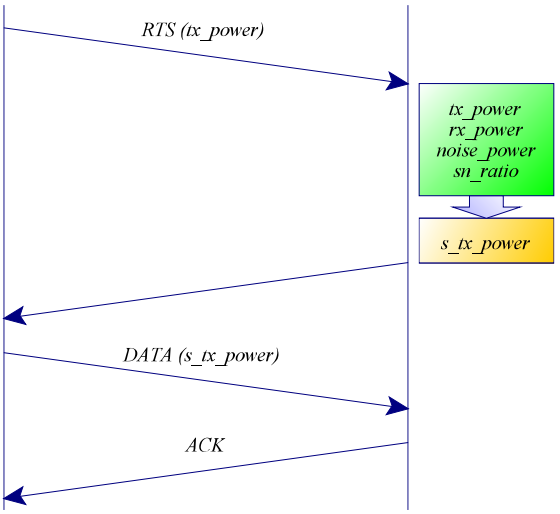

Figura 22 - Esquema del flujo de datos seguido para el calculo de la potencia

En cualquier caso, es necesario información de ida y vuelta para poder realizar la transmisión final, aunque en el primer caso, se requiere la transmisión de menos datos. Además, también puede ser interesante que sea el receptor el que realice los cálculos, porque esto nos puede llevar a la inclusión de nodos especiales con requisitos particulares, y el emisor se limitaría a acatar las órdenes del receptor en este aspecto.

5.2.3 Observaciones

Cabe destacar que, estas comunicaciones preliminares, a parte de realizarse a un nivel de potencia conocido, para que los cálculos

tengan efectos, también han de realizarse a una potencia máxima (o establecer un nivel alto), para que ésta primera comunicación tenga éxito en el primer intento, si es que el receptor está dentro del alcance del nodo en cuestión.

Como se habrá podido deducir de lo explicado hasta ahora, el mecanismo escogido, siempre hace una estimación para cada paquete que se va a enviar, es decir, no se guarda información de transmisiones anteriores. El motivo no es otro que, en primer lugar, evitar hacer un mal uso de la escasa memoria de estos dispositivos, y en segundo lugar, poder adaptarse instantáneamente a cambios en el medio o en la infraestructura de la red.

No obstante, podría ser interesante en ciertos escenarios, el de disponer de ciertos datos almacenados, aunque en la implementación que aquí se presenta, no se ha tenido en cuenta.

Capítulo 6

Implementación

6.1 Introducción

En este capítulo de va a proceder a explicar detalladamente los cambios y adiciones que se han realizado sobre los protocoles anteriormente comentados, para dotarlos de un sistema de control de potencia como el descrito en el capítulo de Arquitectura.

Se mostrará apoyandose en el código fuente que se ha programado, para intentar ser lo más claro posible, a la vez de enseñar el funcionamiento pormenorizado en cuanto a código se refiere.

6.2 S–MAC

Por las características de S–MAC comentadas en anteriores capítulos, a saber, usar mensajes de control previos a la comunicación, se presentaba como un protocolo ideal para empezar a implementar los mecanismos de control de potencia.

Además, el protocolo S–MAC también incorpora una serie de funcionalidades que simplificaban el trabajo a la hora de incorporar el control de potencia. Junto con cada paquete recibido, se guarda el valor de potencia al que se recibió. Igualmente, S–MAC va tomando muestras del medio, en los periodos donde no se reciben paquetes, para tener una estima del ruido.

Sin embargo, de estos dos valores, solo uno, el valor de potencia del paquete, es accesible en la implementación original. Por tanto, este aspecto tuvo que modificarse.

La información relativa al paquete, esta meta-información, se encuentra disponible en una estructura de datos asociada al paquete recibido. Cada paquete recibido, en una estructura `PhyPktBuf`, cuenta con el campo `info` del tipo `PhyPktInfo`:

```
// packet information to be recorded by physical layer
typedef struct
{
    uint16_t strength;
#ifdef TPC_ON
    uint16_t noise;
#endif
    uint32_t timestamp;   // can be used w/ external counter of fine
resolution
    uint32_t timeCoarse;  // S-MAC system time w/ resolution of 1 ms
} __attribute__((packed)) PhyPktInfo;
```

`PhyRadioMsg.h` - Estructura de meta-mensaje de nivel físico

Como se puede observar, esta estructura contaba con el campo `strength`, nosotros hemos añadido el campo `noise`, de manera que podamos acceder tanto al valor de potencia del paquete recibido, com al valor de ruido mas reciente. Con este cambio, resta hacer la modificación pertinente para rellenar el recién incorporado campo:

```
async event result_t SignalStrength.sampleReady(uint16_t value)
{
    ...
    if (typeRSSI == SIGNAL) { // RSSI of a receiving packet
        ((PhyPktBuf*)recvPtr)->info.strength = value;
#ifdef TPC_ON
        ((PhyPktBuf*)recvPtr)->info.noise = noiseLevel;
#endif
        ...
    }
    ...
}
```

`PhyRadioM.nc` - Inclusión del ruido en la estructura

A destacar también, que este valor de ruido no es fruto de una sola muestra, si no que el valor guardado corresponde a un promedio, en el que se le da más importancia a las muestras mas recientes.

Para poder reprogramar el microcontrolador de radio, es necesario conectar con el componente módulo `CC1000ControlM` a través del componente interfaz `CC1000Control`. En principio, esta conexión (*wiring*) podría hacerse directamente con esta interfaz, sin embargo, hay varias razones para no hacerlo así.

En primer lugar el componente interfaz `CC1000Control` es dependiente de la plataforma, es decir, que si el protocolo se adapta a otra plataforma, habría que cambiar esta conexión por otra.

La segunda razón, es que ya existe una conexión en el protoco-

lo S–MAC con este componente, desde el componente `RadioControlM`, por lo que seria redundante. Dicho componente también es dependiente de la plataforma, es decir, su implementación varia con ésta, pero, es un componente propio de `SMAC`, al contrario que `CC1000Control`.

Por esto se opta por la siguiente solución, que es la de crear una interfaz, que sea provista por `RadioControlM`, y que nos dé acceso a los métodos que nos permitan reprogramar la potencia de transmisión:

```
// Transmission Power Control
interface RadioSettings
{
    command result_t SetRFPower(uint8_t power);
    command uint8_t GetRFPower();
}
```

`RadioSettings.nc` - Interfaz de control de potencia

Creada esta interfaz, debemos modificar `RadioControlM`:

```
module RadioControlM
{
    provides {
        ...
        interface RadioSettings;
    }
    uses {
        ...
    }
}

implementation
{
    ...

    inline command result_t RadioSettings.SetRFPower(uint8_t power)
    {
        return call CC1000Control.SetRFPower(power);
    }

    inline command uint8_t RadioSettings.GetRFPower()
    {
        return call CC1000Control.GetRFPower();
    }
}
```

`RadioControlM.nc` - Implementación de la interfaz RadioSettings

El siguiente paso, modificar las estructuras de datos de los mensajes de control, para dotarlos de los campos necesarios para transmitir la potencia, y realizar los cálculos.

```
// control packet -- RTS, CTS, ACK
typedef struct
{
    PhyHeader phyHdr;  // include before my own stuff
    char type;  // type is the first byte following phyHdr
    uint16_t toAddr;
    uint16_t fromAddr;
    uint16_t duration;
#ifdef TPC_ON
    uint16_t strength;
    //uint16_t noise;
#endif
    int16_t crc;  // must be last two bytes, required by PhyRadio
} __attribute__((packed)) MACCtrlPkt;
```

SMACMsg.h - Estructura de mensajes

Se comparte la misma estructura de datos para los mensajes RTS, CTS y ACK. Se añadieron dos campos, `strength` y `noise`. Como se comentaba en anteriores capítulos, solo con un campo es suficiente, pero en la etapa de desarrollo, necesitábamos tener la mayor cantidad de información posible, para poder comprobar que todos los cálculos se realizaban con corrección, de ahí que se usara el campo `noise`.

Por otro lado, en la implementación final, se cambió el campo `strength` a uno del tipo `uint8_t`, puesto que al principio del desarrollo, en estos campos se enviaban los valores correspondientes en decibelios, mientras que en la fase final del desarrollo, se paso a transmitir el dígito de control que usa el microcontrolador CC1000, que es sólo un byte.

Ya por ultimo, resta implementar el mecanismo explicado en el capítulo de Arquitectura, que proporcionara el control de potencia. Este mecanismo se implementa junto con el control RTS/CTS. Este control está implementado en el componente modulo `SMACM`, y es desde este modulo desde donde vamos a hacer los cambios de potencia, por lo tanto, hemos de conectar este componente con el componente `RadioControlM` mediante la interfaz `RadioSettings`, tal y como habíamos comentado antes:

```
implementation
{
    ...
#ifdef TPC_ON
    components RadioControlM;
    components VoltageC;
#endif
    ...
#ifdef TPC_ON
    SMACM.RadioSettings->RadioControlM;
    SMACM.Voltage->VoltageC;
#endif
    ...
}
```

SMAC.nc - Conexión de nuevas interfaces

Como se puede ver en el código, a parte de conectar la interfaz `RadioSettings` con `RadioControlM`, también hemos añadido una conexión con el componente `VoltageC`. Este componente nos será de utilidad para calcular la potencia recibida, como se explicará más detalladamente a continuación.

En primer lugar, incluimos las interfaces que vamos a usar:

```
module SMACM
{
    provides {
        ...
    }
    uses {
        ...
#ifdef TPC_ON
        interface RadioSettings;
        interface ADC as Voltage;
#endif

    }
}
```

SMACM.nc - Declaración de las nuevas interfaces usadas

Como se comentaba antes, necesitamos el componente denominado `VoltageC` para obtener el nivel de voltaje de la alimentación del dispositivo. Este valor se actualizará cada vez que el dispositivo encienda el microcontrolador de radio (recordemos los intervalos de actividad e inactividad del protocolo S–MAC):

```
void wakeup()
{
    ...
#ifdef TPC_ON
    call Voltage.getData();
#endif
}
```

SMACM.nc - Muestreo periódico del nivel de baterías

E igualmente, hemos de definir la siguiente función, que es la

encargada de recibir el valor medido. El valor obtenido viene dado en milivoltios:

```
#ifdef TPC_ON
async event result_t Voltage.dataReady(uint16_t data) {
    atomic voltage = data;
    return SUCCESS;
}
#endif
```

SMACM.nc - Procesado del nivel de baterias

Este valor de voltaje es importante, dado que, las demás mediciones que hace el ADC usan como nivel de referencia el voltaje de la alimentación, de manera que para obtener una conversión lo más fiel posible debemos usar los valores más exactos posibles.

También es importante destacar, que para la medida de voltaje de la alimentación, se toma como nivel de voltaje de referencia el proporcionado por un componente del dispositivo, un circuito integrado LM4041 que proporciona un nivel de voltaje de 1,223 V. El cálculo del voltaje se realiza internamente de acuerdo a esta fórmula:

$$V_{batt} = V_{ref} \frac{ADC\_FS}{ADC\_Count}$$

Donde *ADC_FS* es una constante equivalente a 1024, *ADC_Count* es el valor sin procesar leído del ADC y V_{ref} el valor citado anteriormente.

Por otro lado, se definen un par de funciones, para poder convertir un valor dado de potencia de emisión, al valor correspondiente que ha de programarse en el registro del microcontrolador CC1000, y viceversa. Ha de tenerse en cuenta, que no es posible emitir a una potencia arbitraria dentro de un rango dado, si no que solo se puede emitir a unos valores dados. Por lo tanto, al hacer una conversión de dB a un valor de registro, siempre se hará de manera que éste sea suficientemente alto:

```
#ifdef TPC_ON
    signed char convertCC1000PowerTodBmPower(unsigned char pwr)
    {
          int i;

            for(i=0;i<22;i++)
        {
            if(pwr_reg[i] == pwr)
            {
                return power[i];
            }
        }
    }


    unsigned char convertdBmPowerToCC1000Power(signed char pwr)
    {
        int i;

        for(i=0;i<22;i++)
        {
            if(power[i] >= pwr)
            {
                return pwr_reg[i];
            }
        }

        return 0xFF;
    }
#endif
```

SMACM.nc - Funciones de conversión de potencia

Los valores vienen definidos en un archivo de cabecera adicional:

```
const int8_t power[22] = {
    -19,
    -17,
    ...
    3,
    4,
    5
};

const unsigned char pwr_reg[22] = {
    0x02,
    0x03,
    ...
    0xC0,
    0xF0,
    0xFF
};
```

tpc.h - Vectores de correspondencia de potencia

Con esto, podemos pasar a añadir los comandos necesarios para programar el control de potencia. En primer lugar, y de acuerdo al mecanismo explicado con anterioridad, programamos la emisión del paquete de control a máxima potencia (con el valor de registro 0xFF) y usamos el campo strength para informar de la potencia usada para enviar el paquete.

```
void sendRTS()
{
    // construct and send RTS packet
    ...
#ifdef TPC_ON
    call RadioSettings.SetRFPower(0xFF);
    rtsPkt.strength = call RadioSettings.GetRFPower();
#endif
    ...
}
```

SMACM.nc - Función para enviar paquete RTS

En el nodo receptor, al recibir el paquete RTS se procederá a hacer el cálculo oportuno para determinar a que potencia a de emitirse el paquete de datos.

```
void handleRTS(void* pkt)
{
    ...

    rts_strength = ((PhyPktBuf*)pkt)->info.strength;
    rts_noise = ((PhyPktBuf*)pkt)->info.noise;

    rx_power = -50.0 * ((voltage/1000.0)*(float)rts_strength/1024.0) -
45.5;

    noise_power = -50.0 * ((voltage/1000.0)*(float)rts_noise/1024.0) -
45.5;

    tx_power = convertCC1000PowerTodBmPower((unsigned
char)packet->strength);

    //tx_power = tx_power - ((rx_power - noise_power) - SN_RATIO);
    tx_power = tx_power - rx_power + noise_power + SN_RATIO;

    ctsPkt.strength = convertdBmPowerToCC1000Power((signed
char)tx_power);

    ...
}
```

SMACM.nc - Función encargada de administrar los paquetes RTS (incluye el calculo de la potencia de emisión)

En primer lugar aclaremos que los valores strength y noise de la estructura de datos info, son datos directos del ADC, y hay que convertirlos a voltaje con la siguiente fórmula:

$$V_{RSSI} = V_{batt} \frac{ADC\_Count}{ADC\_FS}$$

Fórmula con una estructura similar a la mostrada anteriormente. V_{batt} será en este caso el valor de voltaje de la alimentación (en la variable voltage, en milivoltios), *ADC_FS* seguira siendo 1024, y *ADC_Count* el valor proporcionado por la variable strength o noise.

Seguidamente, haciendo uso de la fórmula que nos permite calcular el valor de potencia dada la lectura del CC1000:

$$P = -50V_{RSSI} - 45{,}5 \ (dBm)$$

Calculamos las potencias en dBm del ruido y de la señal recibida. Igualmente, convertimos el valor de potencia proporcionado por el campo strength del paquete RTS recibido, que nos indica el valor de potencia al que fue emitido, a su valor equivalente en dBm.

Resumiendo, tenemos los siguientes valores:

- `tx_power`, o potencia al que fue emitido el paquete RTS.
- `rx_power`, o potencia al que se recibió el paquete RTS.
- `noise_power`, o potencia de ruido (promediada).
- `SN_RATIO`, o nivel de potencia por encima del nivel del ruido al que queremos emitir.

Con estos datos, podemos pasar a calcular la potencia necesaria de la siguiente manera:

$$P = tx_{power} - rx_{power} + noise_{power} + SN_{RATIO}$$

El resultado de la resta de los dos primeros términos nos da como resultado las perdidas del medio entre el emisor y el receptor, y que representa un margen que ha de superarse en la comunicación. Los otros dos términos, que ya se han comentado, representan otros dos margenes, el del ruido, y el de salvaguarda por encima del ruido. El resultado de la suma de todos estos términos, me da como resultado la potencia a la que se debe emitir el paquete de datos.

En el siguiente cuadro, se presenta el código optimizado para realizar este calculo, ya que muchos términos eran comunes y por tanto la expresión en si podía simplificarse:

```
void handleRTS(void* pkt)
{
    ...

    tx_power = convertCC1000PowerTodBmPower(
            (unsigned char)packet->strength
        )
        +
        (
            (4.8828125e-5)
            *
            (voltage)
            *
            (
                ((PhyPktBuf*)pkt)->info.strength
                -
                ((PhyPktBuf*)pkt)->info.noise
            )
        )
        + SN_RATIO;

    ctsPkt.strength = convertdBmPowerToCC1000Power(
            (signed char)tx_power
        );

    ...
}
```

SMACM.nc - Versión optimizada del cálculo de potencia

Nótese que el valor de potencia calculado, es convertido primero a valor de registro apropiado para el CC1000, y luego guardado en la variable apropiada de la estructura del paquete CTS (recordemos que S–MAC usa estructuras estáticas para los paquetes).

Luego cuando se recibe el paquete CTS, el receptor del mismo extraerá el valor de potencia del paquete y lo guardará en una variable global:

```
void handleCTS(void* pkt)
{
    ...
    if (packet->toAddr == TOS_LOCAL_ADDRESS)
    {
        if (state == WAIT_CTS && packet->fromAddr == sendAddr)
        {
            ...

#ifdef TPC_ON
            tx_prg_power = packet->strength;
#endif
            ...
        }
        ...
    }
    ...
}
```

SMACM.nc - Función que administra los paquetes CTS (detalle de lectura de la potencia de transmisión deseada)

Para posteriormente ser usada cuando se emite el paquete de

datos, para programar el CC1000 a la potencia adecuada.

```
void sendDATA()
{
    ...
#ifdef TPC_ON
    call RadioSettings.SetRFPower(tx_prg_power);
    dataPkt->strength = call RadioSettings.GetRFPower();
#endif
}
```

SMACM.nc - Función para enviar paquetes de datos (detalle de programación de la potencia de transmisión deseada)

Como nota final, destacar que todos los cambios realizados, son desactivables mediante la macro TPC_ON, de manera que podemos, en tiempo de compilación, seleccionar una versión con control de potencia o no, definiendo o no está macro.

6.3 SCP–MAC

Como se había comentado en capítulos anteriores, el protocolo SCP–MAC ha sido desarrollado por los mismo autores del protocolo S–MAC, de este modo, existen multitud de similitudes, en cuanto a forma se refiere, en el modo de implementar ambos protocolos. Este detalle ha facilitado una mayor rapidez de implementación en este caso.

Igual que en S–MAC empezamos modificando las estructuras de datos, para dar cabida a un nuevo campo de datos, tanto en la estructura del paquete de datos:

```
typedef struct {
    PhyHeader phyHdr;
    uint8_t type;  // type is the first byte following phyHdr
    uint16_t fromAddr;
    uint16_t toAddr;
#ifdef TPC_ON
    uint8_t strength;
#endif
    //  uint8_t seqFragNo;
} __attribute__((packed)) CsmaHeader;
```

CsmaMsg.h - Modificación de la estructura cabecera de datos

Como en las estructuras de control RTS:

```
typedef struct {
    PhyHeader phyHdr;   // include before my own stuff
    uint8_t type;   // type is the first byte following phyHdr
    uint16_t fromAddr;
    uint16_t toAddr;
    uint16_t duration;
#ifdef TPC_ON
    uint8_t strength;
#endif
    int16_t crc;    // must be last two bytes, required by PhyRadio
} __attribute__((packed)) RTSPkt;
```

CsmaMsg.h - Modificación de la estructura RTS

Y CTS:

```
typedef struct {
    PhyHeader phyHdr;   // include before my own stuff
    uint8_t type;   // type is the first byte following phyHdr
    uint16_t toAddr;
    uint16_t duration;
#ifdef TPC_ON
    uint8_t strength;
#endif
    int16_t crc;    // must be last two bytes, required by PhyRadio
} __attribute__((packed)) CTSPkt;
```

CsmaMsg.h - Modificación de la estructura CTS

No ha sido necesario, como en el caso de S–MAC, realizar una modificación a la estructura de metadatos asociada a todo paquete, donde podíamos encontrar la potencia a la que se había recibido un paquete. Recordemos que en S–MAC, en esta estructura, teníamos un campo para la potencia, pero no teníamos uno para el ruido. En cambio, en SCP–MAC, ya existe el campo ruido, que nos proporciona el valor de ruido mas reciente.

Seguidamente, pasamos a introducir cambios en el módulo `Csma`, introduciendo las conexiones apropiadas para poder controlar la potencia de transmisión, como para poder monitorizar el nivel de voltaje de la batería (recordemos que este dato es importante para dar un valor preciso de la potencia recibida).

```
implementation
{
    components CsmaM, PhyRadio, RandomLFSR, TimerC,

    ...

#ifdef TPC_ON
    components VoltageC;
#endif

    ...

#ifdef TPC_ON
  CsmaM.RadioTxPower -> PhyRadio;
  CsmaM.Voltage->VoltageC;
#endif

    ...

}
```

Csma.nc - Introducción de nuevos componentes y conexionado

Concluimos añadiendo el código necesario para hacer uso de las nuevas interfaces que hemos creado:

```
module CsmaM
{
    provides {
        ...
    }
    uses {

    ...

#ifdef TPC_ON
    interface GetSetU8 as RadioTxPower;
    interface ADC as Voltage;
#endif

    ...

    }
}
```

CsmaM.nc - Nuevas interfaces incorporadas

Es importante destacar que, para la implementación del control de potencia en este protocolo, se ha tenido que tomar un procedimiento ligeramente diferente que en el caso de S–MAC. De este modo, se han añadido parámetros a las siguientes funciones, involucradas en el control de potencia:

```
#ifdef TPC_ON
    void sendCTS(unsigned char pwr);
#else
    void sendCTS();
#endif

#ifdef TPC_ON
    void sendUcast(uint16_t addPreamble, unsigned char tx_prg_power);
#else
    void sendUcast(uint16_t addPreamble);
#endif
```

CsmaM.nc - Modificación de la declaración de funciones vinculadas al TPC

También se vuelve a hacer uso de las funciones que ya se programaron para S–MAC, con las que podíamos convertir un valor de potencia en dB, a su correspondiente valor asociado en la tabla de registro del CC1000 (o viceversa):

```
#ifdef TPC_ON
    signed char convertCC1000PowerTodBmPower(unsigned char pwr);
    unsigned char convertdBmPowerToCC1000Power(signed char pwr);
#endif
```

CsmaM.nc - Incorporación de las funciones de conversion de potencia

Igualmente, debemos inicializar el componente monitor del voltaje de las baterías, del mismo modo que hicimos en S–MAC:

```
async event result_t RadioState.wakeupDone()
{
    // initialize CSMA and related components

    ...

#ifdef TPC_ON
    call Voltage.getData();
#endif
    ...

}
```

CsmaM.nc - Monitorización del nivel de baterias

E implementamos la función receptora del evento que se lanza cuando se ha realizado una medición:

```
#ifdef TPC_ON
  async event result_t Voltage.dataReady(uint16_t data) {

    atomic voltage = data;

    return SUCCESS;

  }
#endif
```

CsmaM.nc - Procesado del valor del nivel de voltaje

Pasamos ahora al control de flujo mismo, y empezamos relle-

nando el campo del paquete RTS, para indicar la potencia a la que se transmite dicho paquete:

```
void sendRTS()
{
    // construct and send RTS packet

    ...

#ifdef TPC_ON
    call RadioTxPower.set(0xFF);
    rtsPkt->strength = call RadioTxPower.get();
#endif

    ...

}
```

CsmaM.nc - Establecimiento del nivel de potencia usado en paquete RTS

Una vez recibido un paquete RTS, procedemos a realizar los cálculos oportunos para establecer el valor de potencia al que se debe emitir el paquete de datos.

```
void handleRTS(void* pkt)
{
    // internal handler for RTS

    ...

#ifdef TPC_ON
    tx_power = convertCC1000PowerTodBmPower((unsigned
char)packet->strength)
                    +
((4.8828125e-5)*(voltage)*(((PhyPktBuf*)pkt)->info.strength-((PhyPktBu
f*)pkt)->info.noise))
                    + SN_RATIO;

    sendCTS(convertdBmPowerToCC1000Power((signed char)tx_power));
#else
    sendCTS();
#endif

    ...

}
```

CsmaM.nc - Cálculo del valor de potencia deseado

Modificamos la función `sendCTS`, donde se puede observar como aquí el valor que se introduce en dicho campo `strength` es el que se le pasa a la función.

```
#ifdef TPC_ON
void sendCTS(unsigned char pwr)
#else
void sendCTS()
#endif
{
    // construct and send CTS

    ...

    #ifdef TPC_ON
    ctsPkt->strength = pwr;
    call RadioTxPower.set(0xFF);
    #endif

    ...
}
```

CsmaM.nc - Envio de paquete CTS con valor de potencia deseado

Y una vez recibido el paquete CTS, podemos realizar la transmisión a la potencia indicada:

```
void handleCTS(void* pkt)
{
    // internal handler for CTS

    ...

#ifdef TPC_ON
    sendUcast(0,packet->strength);      // use base preamble if RTS/CTS
is used
#else
    sendUcast(0);   // use base preamble if RTS/CTS is used
#endif

    ...

}
```

CsmaM.nc - Recepción de pauqete CTS y envio de paquete de datos a nivel de potencia deseado

Y por último modificamos la función con la que se envían los paquetes de datos, estableciendo la potencia de emisión al valor indicado.

```
#ifdef TPC_ON
void sendUcast(uint16_t addPreamble, unsigned char tx_prg_power)
#else
void sendUcast(uint16_t addPreamble)
#endif
{

    ...

#ifdef TPC_ON
    call RadioTxPower.set(tx_prg_power);
#endif

    ...

}
```

CsmaM.nc - Establecimiento del nivel de potencia deseado para la transmisión del paquete de datos

Al igual que en la implementación de S–MAC, mediante el uso de la macro `TPC_ON` es posible activar o desactivar el control de potencia.

Por otro lado, el protocolo SCP–MAC, en su comportamiento normal, solo usa las tramas de control RTS/CTS pasado un umbral de tamaño de paquete, es decir, que para paquetes pequeños, no se usa el control de flujo.

Sin embargo, para nuestros intereses, este control de flujo ha de existir siempre, y por lo tanto, hemos establecido un umbral de 1 byte para activar el uso de dichos paquetes. Esto se activa en un archivo de configuración (`config.h`), que establecen una serie de macros en tiempo de compilación.

6.4 B–MAC

Para el protocolo B–MAC se ha tomado una aproximación, a la hora de realizar la implementación, totalmente diferente. Como se comento en el capítulo en el que se daba una pequeña explicación sobre el funcionamiento de cada protocolo, B–MAC es un protocolo de acceso aleatorio al medio y la comunicación entre los dos extremos se realiza “directamente”, sin un periodo previo de establecimiento de la comunicación como el visto en S–MAC, sino que se usa un preámbulo para advertir de la inminente transmisión.

Por tanto, el protocolo B–MAC era, a priori, un mal candidato para nuestro mecanismo de control de potencia. Sin embargo, el protocolo B–MAC está implementado de tal manera que es posible realizar grandes cambios en su estructura interna fácilmente, lo que

unido a la naturaleza de división por componentes de nesC, posibilita crear una implementación de nuestro modelo para B–MAC.

A grandes rasgos, lo que se va a llevar a cabo es la creación de un nuevo componente, que colocaremos haciendo de intermediario entre dos de los ya existentes en el modelo B–MAC. Este componente intermedio será el responsable de crear los mensajes de control, y de realizar los cálculos de nivel de potencia. A nivel de aplicación, esto supone que no hay que realizar ningún cambio, ya que no interfieren en esta capa. De este modo, insertaremos nuestro recién creado componente, pasando de esta conexión:

Figura 23 - Conexionado inicial de componentes en B–MAC

A esta:

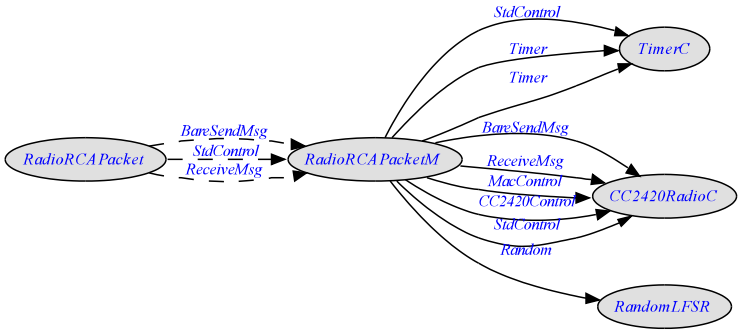

Figura 24 - Conexionado resultante de la implementación de los nuevos módulos

Como siempre, empezamos con las estructuras de datos, en este caso, partiendo desde cero, ya que B–MAC no cuenta con este tipo de estructuras.

```
typedef struct CtrlMsg
{
    uint8_t src;
    uint8_t potencia;
    uint16_t secuencia;
} CtrlMsg;
```

CtrlMsg.h - Estructura para los mensajes de control

Puesto que nuestra implementación hace uso del resto de componentes que tiene implementado B–MAC, vamos a hacer uso de la misma estructura de mensajes. Esto quiere decir, que nuestros mensajes de control irán encapsulados en dichas estructuras, que tienen la siguiente forma:

```
typedef struct TOS_Msg
{
    /* The following fields are transmitted/received on the radio. */
    uint16_t addr;
    uint8_t type;
    uint8_t group;
    uint8_t length;
    int8_t data[TOSH_DATA_LENGTH];
    uint16_t crc;

    /* The following fields are not actually transmitted or received
    * on the radio! They are used for internal accounting only.
    * The reason they are in this structure is that the AM interface
    * requires them to be part of the TOS_Msg that is passed to
    * send/receive operations.
    */
    uint16_t strength;
    uint8_t ack;
    uint16_t time;
    uint8_t sendSecurityMode;
    uint8_t receiveSecurityMode;
} TOS_Msg;
```

AM.h - Estructura de los mensajes B–MAC

Y por este motivo, se han definido unos valores particulares para poder definir los tipos de mensajes en el campo type:

```
enum
{
    RTS =   0x33,
    CTS =   0x44,
    ACK =   0x55
};
```

CtrlMsg.h - Identificador para los mensajes de control

A continuación, se modificó el componente RacioCRCPacket, para pasar a conectarse al nuevo componente creado:

```
configuration RadioCRCPacket
{
    provides {
        ...
    }
}

implementation
{
    //components CC2420RadioC as RadioCRCPacketM;
    components RadioRCAPacket as RadioCRCPacketM;

    Control = RadioCRCPacketM;
    Send = RadioCRCPacketM.Send;
    Receive = RadioCRCPacketM.Receive;

}
```

RadioRCAPacket.nc - Configuración de componentes y conexionado

Donde se puede ver, que se ha substituido la primera linea de componentes (ahora comentada) por una nueva, que substituye el componente CC2420RadioC por el componente RadioRCAPacket.

Resta únicamente la implementación de este nuevo componente. A grandes rasgos, este componente va a tener 4 funciones principales. Dos interfaces (Send y Receive) para servir a la capa superior, la que conecta con el componente RadioCRCPacket (a su vez conectado con el AMStandard, que serán las encargadas de recoger las peticiones de enviar datos, y de entregar los mensajes de datos recibidos de la capa superior, la de aplicación. Otras dos interfaces (RadioSend y RadioReceive), para establecer la comunicación con la capa inferior:

```
module RadioRCAPacketM
{
    provides
    {
        interface StdControl as Control;

        interface BareSendMsg as Send;
        interface ReceiveMsg as Receive;
    }

    uses
    {
        interface StdControl as RadioControl;

        interface BareSendMsg as RadioSend;
        interface ReceiveMsg as RadioReceive;

        ...
    }
}
```

RadioRCAPacketM.nc - Interfaces usados y provistos

Con la función Send.send damos servicio a las peticiones de envío de la capa superior:

```
command result_t Send.send(TOS_MsgPtr msg)
{
    if(pendingTX)
    {
        return FAIL;
    }
    else
    {
        pendingTX = TRUE;

        buffer = msg;

        call RTSTimer.start(TIMER_ONE_SHOT, RTS_WAIT);
        sendCtrlPkt(RTS,msg);

        return SUCCESS;
    }
}
```

`RadioRCAPacketM.nc` - Procesado de las peticiones de envío de la capa superior (aplicación)

Iniciando todo el proceso del mecanismo RTS/CTS, solo en el caso que no se esté ya intentando hacer un envío. Se lanza un temporizador de un solo disparo, es decir, se programa para que se lance una vez, y a la vez se envía el primer mensaje RTS. Este temporizador nos servirá para ir enviando una serie de mensajes RTS, hasta recibir respuesta, o hasta llegar a un máximo de envíos:

```
event result_t RTSTimer.fired()
{
    if(countRTS++ > RTS_RETRY_LIMIT)
    {
        countRTS = 0;
        signal Send.sendDone(buffer, FAIL);
        pendingTX = FALSE;
    }
    else
    {
        call RTSTimer.start(TIMER_ONE_SHOT, RTS_WAIT);
        sendCtrlPkt(RTS,buffer);
    }

    return SUCCESS;
}
```

`RadioRCAPacketM.nc` - Procesado del temporizador de mensajes RTS

Cuando el temporizador se lanza, se comprueba si se han alcanzado el numero máximo de mensajes RTS, en cuyo caso se notificará a la capa superior el fracaso en el intento del envío, a la vez que se reinicializan los contadores. En otro caso, se programa otra vez el disparador, a la vez que se vuelve a enviar un nuevo mensaje RTS.

Con el evento `RadioSend.sendDone`, que se lanza una vez que un mensaje se ha enviado, solo tenemos que informar a la capa de aplicación si el intento de envío no ha tenido éxito, mediante la señal

`Send.sendDone`, que provocara el lanzamiento de un evento en la capa aplicación.

```
event result_t RadioSend.sendDone(TOS_MsgPtr msg, result_t success)
{
    switch(msg->type)
    {
        case RTS:
        case CTS:
        case ACK:
            break;
        default:
            if(success == FAIL)
            {
                signal Send.sendDone(buffer, FAIL);
                pendingTX = FALSE;
            }
    }

    return SUCCESS;
}
```

`RadioRCAPacketM.nc` - Procesado de los intentos de envio de mensajes

El evento `RadioReceive.receive`, por otro lado, se lanzará por la capa inferior cada vez que reciba un paquete. En este caso, al tener que realizar una implementación del mecanismo RTS/CTS se ha replicado el comportamiento de los protocolos anteriores, a la vez que se combina con nuestro mecanismo de control de potencia.

Básicamente, la implementación de esta función podemos resumirlo en los siguientes casos:

- En el caso de un mensaje broadcast, lo pasamos a la capa superior.
- En caso de recibir un RTS, respondemos con un CTS.
- En caso de recibir un CTS, respondemos enviando el mensaje de datos.
- En caso de recibir un ACK, indicamos a la capa superior el éxito en la transmisión.
- En caso de tratarse de un mensaje de datos, se lo entregamos a la capa superior, y respondemos con ACK.

```
event TOS_MsgPtr RadioReceive.receive(TOS_MsgPtr packet)
{
    if(packet->addr == TOS_BCAST_ADDR)
    {
        return signal Receive.receive((TOS_MsgPtr)packet);
    }
    else if (packet->addr == TOS_LOCAL_ADDRESS)
    {
        switch(packet->type)
        {
            case RTS:
                sendCtrlPkt(CTS,packet);
                break;
            case CTS:
                call RTSTimer.stop();

                call CC2420Control.SetRFPower(((CtrlMsg
*)msg->data)->potencia);

                if(call RadioSend.send(buffer) == FAIL)
                {
                    signal Send.sendDone(buffer, FAIL);
                    pendingTX = FALSE;
                }
                break;
            case ACK:
                call ACKTimer.stop();
                signal Send.sendDone(buffer, SUCCESS);
                pendingTX = FALSE;
                break;
            default:
                sendCtrlPkt(ACK,packet);
                return signal Receive.receive((TOS_MsgPtr)packet);
        }
    }
}
```

`RadioRCAPacketM.nc` - Procesado de los mensajes de la capa inferior

Los dispositivos MICAz, los cuales usan, como se ha comentado, el microcontrolador Chipcon CC2420, disponen de menos variedad en la potencia de emisión. Mientras que en los MICA2 teníamos 22 potencias distintas, aquí solo contamos con 8. De esta manera, el fichero `tpc.h` se ha modificado para reflejar los valores apropiados:

```
const int8_t power[8] = {
    -25,
    -15,
    -10,
    -7,
    -5,
    -3,
    -1,
    0
};

const unsigned char pwr_reg[8] = {
    3,
    7,
    11,
    15,
    19,
    23,
    27,
    31
};
```

tpc.h - Vectores de correspondencia de potencia para CC2420

Igualmente, tendremos que hacer uso de unas funciones de funcionamiento similar a las implementadas en protocolos S–MAC y SCP–MAC, para convertir un valor de potencia, a un valor de registro válido para el CC2420, y que únicamente varían en los datos que manipulan.

```
signed char convertCC2420PowerTodBmPower(unsigned char pwr)
{
    int i;

    for(i=0;i<8;i++)
    {
        if(pwr_reg[i] == pwr)
        {
            return power[i];
        }
    }
}


unsigned char convertdBmPowerToCC2420Power(signed char pwr)
{
    int i;

    for(i=0;i<8;i++)
    {
        if(power[i] >= pwr)
        {
            return pwr_reg[i];
        }
    }

    return 0x31;
}
```

RadioRCAPacketM.nc - Funciones auxiliares de conversión

Por conveniencia, se ha programado una función auxiliar para el envío de los mensajes de control, para que así la implementación fuese lo más clara posible. Va a ser en esta función donde se lleve acabo el calculo de la potencia necesaria para poder indicarsela al emisor.

En este caso, vamos a optar por documentar el proceso llevado a cabo para los dispositivos MICAz, ya que en los protocolos anteriores ha quedado detallado el proceso, y éste seria prácticamente igual en el caso de MICAz, en cuanto al calculo de potencia se refiere.

```
void sendCtrlPkt(uint8_t type, TOS_MsgPtr msg)
{
    ctrlPkt.type = type; // Tipo de paquete
    ctrlPkt.group = TOS_AM_GROUP; // Grupo
    ctrlPkt.length = 4; // Tamaño de los datos

    call CC2420Control.SetRFPower(0x31);

    switch(type)
    {
        case RTS:
            ctrlPkt.addr = msg->addr; // Dir. destino
            secLocal++;
            ctrlMsg->secuencia = secLocal;
            ctrlMsg->potencia = 0x31;
            break;
        case CTS:
            ctrlPkt.addr = ((CtrlMsg *)msg->data)->src;
            ctrlMsg->secuencia = ((CtrlMsg *)msg->data)->secuencia;
            secExtern = ctrlMsg->secuencia;

            rx_power = msg->strength - 45;

            noise_power = msg->noise - 45;

            tx_power = convertCC2420PowerTodBmPower(((CtrlMsg
*)msg->data)->potencia);

            tx_power = tx_power - rx_power + noise_power + SN_RATIO;

            ctrlMsg->potencia = convertdBmPowerToCC2420Power((signed
char)tx_power);

            break;

        case ACK:
            ctrlPkt.addr = msg->addr;
            secExtern++;
            ctrlMsg->secuencia = secExtern;
            ctrlMsg->potencia = 0x31;
            call ACKTimer.start(TIMER_ONE_SHOT, RTS_WAIT*2.1);
            break;
    }

    call RadioSend.send(&ctrlPkt);
}
```

RadioRCAPacketM.nc - Función auxiliar para el envío de mensajes de control, y calculo de potencia

Recordar también que, el valor entregado por el CC2420, ya esta digitalizado, y para su uso, solo hay que aplicarle un factor co-

rrectivo, que puede ser aproximado por –45 dB.

Capítulo 7

Pruebas

7.1 Introducción

En este capítulo se presentan las mediciones experimentales que se han realizado para poder comprobar la bondad de la implementación realizada, y su capacidad para reducir el consumo.

En los siguientes puntos, se explicarán los detalles de las pruebas realizadas, así como del equipo utilizado y los resultados obtenidos. Cabe destacar que, finalmente, sólo se han podido realizar las pruebas en una única plataforma, MICA2, y con el protocolo S–MAC, debido a problemas y limitaciones de hardware, y también a limitaciones de tiempo.

7.2 Aplicación test

Para realizar las pruebas se ha diseñado una aplicación en nesC con el fin de instalarla en los dispositivos para realizar las mediciones, cuya principales premisas han sido la sencillez, y proporcionar un método de procurar que los dispositivos hagan el mismo número de transmisiones.

7.2.1 Arquitectura

Tal y como se ha indicado anteriormente, a la hora de diseñar una aplicación para llevar acabo las pruebas, se puso como premisa para éste, que fuese sencillo, de manera que cualquier error que pudiera surgir fuera lo más fácil y rápidamente posible encontrarlo y repararlo. Crear un experimento demasiado complicado puede vol-

verse en contra de los intereses del propio experimento en cualquier escenario.

En segundo lugar, también era importante conseguir que los dispositivos implicados en la prueba realizaran, dado un tiempo de observación lo suficientemente largo, una cantidad de transmisiones parecida, o idealmente, que realizaran el mismo numero de transmisiones, en las mismas condiciones de entorno, etc...

Para ello surgió la idea de realizar una aplicación, que podríamos asemejar al tenis de mesa, o, usando un ejemplo más tecnológico, la aplicación *ping*, esto es, de solicitud de eco/respuesta de eco. Es decir, en un escenario con únicamente dos dispositivos, uno, actuando como máster, envía mensajes al otro extremo, con una cierto intervalo entre solicitudes, mientras que el otro extremo, que actúa como esclavo o pasivo, se limita a responder de vuelta, en cuanto éste llega.

Además, uno de estos dispositivos tendrá activado el control de potencia, forzando al otro extremo a variar su potencia en función del nivel de ruido, mientras que el otro extremo siempre requerirá todos los mensajes con la misma potencia.

De este modo, los elementos queda de la siguiente manera:

- Dos dispositivos MICA2 (uno máster, otro esclavo)
- El máster envía mensajes con un cierto intervalo, ajustando el nivel de potencia acorde a los requisitos
- El esclavo responde al mensaje inmediatamente, reenviando el mensaje de vuelta, requiriendo potencia fija

7.2.2 Implementación

El módulo que hemos creado ha sido denominado `Test`, y está basado en la aplicación que acompaña al código fuente de S–MAC, `SMACTest`, pero la simplificación del código ha sido tal, que excepto por una estructura similar, apenas guardan relación.

Junto al módulo implementado se hace uso de un archivo de cabecera para establecer una configuración, mediante macros, que serán usadas por el programa de test:

```
#define TST_MIN_NODE_ID 1        // at least 2 nodes. node IDs must be
#define TST_MAX_NODE_ID 2        // consecutive from min to max
#define TST_MSG_INTERVAL 5000     // in ms
```

`config.h` - Macros de configuración para el programa de pruebas

O por el propio código de S–MAC:

```
#define PHY_MAX_PKT_LEN 80       // max: 250 (bytes), default: 100
#define SMAC_RTS_RETRY_LIMIT 1  // default value 7
#define SMAC_DATA_RETX_LIMIT 1  // default value 3

#define TPC_ON // Control de potencia
#define SN_RATIO 100 // Relación señal al ruido requerida
```

`config.h` - Macros de configuración para S–MAC

El conexionado de componentes del módulo queda de la siguiente manera, haciendo uso del menor número posible de componentes:

```
includes config;
includes testMsg;

configuration Test { }

implementation
{
    components Main, TestM, SMAC, LedsC;

    Main.StdControl -> TestM;
    TestM.MACControl -> SMAC;
    TestM.MACComm -> SMAC;
    TestM.MACTest -> SMAC;
    TestM.Leds -> LedsC;
}
```

`Test.nc` - Configuración del módulo `Test`

Mientras que el modulo de implementación TestM se detalla a continuación:

```
    ...

    command result_t StdControl.init()
    {

        if (TOS_LOCAL_ADDRESS == TST_MAX_NODE_ID)
        {
            unicastId = TST_MIN_NODE_ID;
            master = TRUE;
        }
        else
        {
            unicastId = TOS_LOCAL_ADDRESS + 1;
            master = FALSE;
        }

        timeCount = 20000;

        call MACControl.init();   // initialize MAC and lower layers
        call Leds.init();
        call Leds.yellowOn();
        return SUCCESS;
    }

    ...
```

`TestM.nc` - Función de inicialización

La función de inicialización, como es habitual, se encarga de

establecer unos valores iniciales de las variables. En este caso, se establece el papel de máster o esclavo, en función de la identificación que se haya asignado en tiempo de compilación a cada dispositivo, así como la dirección a la que se enviaran los mensajes.

Igualmente, también se establece un valor inicial en la variable `timeCount`, que se usa para establecer el intervalo entre mensajes. Sin embargo, este valor inicial es mayor de lo habitual para dejar tiempo a los dispositivos a realizar la secuencia inicial de sincronizado, que lleva algunos segundos.

Posteriormente, otro valor de intervalo se toma en el evento lanzado por el contador de la capa SMAC:

```
...

event void MACTest.clockFire()
{
    if (timeCount > 0)
    {
        timeCount--;
        if (timeCount == 0)
        {
            timeCount = TST_MSG_INTERVAL;
            if (master && !send_pending)
            {
                post sendMsg();
            }
        }
    }
}

...
```

`TestM.nc` - Función lanzada por el reloj de S–MAC

Como se puede ver, la función que se lanza en cada evento de reloj, es la encargada de llevar el contador que establece cuando se realiza el envío del mensaje.

Aquí la variable `timeCount` se inicializa al valor de macro `TST_MSG_INTERVAL`. Además se aprecia que, la condición para enviar ese mensaje, es que el dispositivo sea máster, y que no halla ninguna transmisión pendiente.

Por otro lado, es en la función `rxMsgDone`, lanzada al recibir un mensaje, donde el dispositivo que actúa como esclavo podrá lanzar sus mensajes de respuesta:

```
...

event void* MACComm.rxMsgDone(void* msg)
{
    call Leds.redToggle();
    if (!master && !send_pending)
    {
        post sendMsg();
    }
    return msg;
}

...
```

`TestM.nc` - Función para procesar los mensajes recibidos

7.3 Equipo de medición

Para la medición del consumo se ha hecho uso del osciloscopio PicoScope de la serie 5000, concretamente el modelo 5203, fabricado por la empresa Pico Technology[11].

Figura 25 - Picotech PicoScope 5203

Ancho de banda	250 Mhz
Canales	2
Resolución	8 bits
Precisión	±3%
Rango	de ±100 mV a ±20 V en 8 rangos
Muestreo	10^6 muestras/segundo

Por otro lado, se recurrió a un circuito estándar, para medir consumo, es decir, en el circuito de alimentación del dispositivo MICA2 insertamos una resistencia en serie. Esta resistencia se selecciona de muy bajo valor, 1 Ω nominal, para no interferir con el funcionamiento normal del dispositivo.

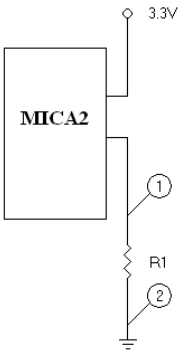

Figura 26 - Esquemático del circuito de medida

Posteriormente, se mide la caída de voltaje entre los nodos de la resistencia (en la imagen, nodos 1 y 2) que tienen una relación directa con la corriente que transcurre por ella, en función del valor de la propia resistencia.

7.4 Escenario de pruebas

El escenario previsto para las pruebas se ha intentado escoger, dentro de las limitaciones existentes, para intentar asemejarse lo máximo posible a una situación real.

Para ello, dos nodos han sido colocados a una distancia fija de 5 metros en el laboratorio de pruebas, y han sido programados con la

aplicación de test. Seguidamente, han sido puestos en marcha, y se les han tomado medidas a lo largo de 1 hora.

7.5 Resultados

En primer lugar, vamos a observar la captura individual de una secuencia de eco y respuesta, para identificar los distintos niveles de consumo asociados a los distintos paquetes y dispositivos MICA2.

El dispositivo máster, en rojo, es el que inicia la comunicación con un mensaje RTS (2), que es inmediatamente respondido por el dispositivo esclavo, en azul. Observese que el periodo de inactividad, tiene el nivel de voltaje más bajo, ya que en este modo (6) el transceptor esta apagado. Cuando el transceptor se conecta, el consumo es bastante evidente, como observa con el nivel marcado por (1).

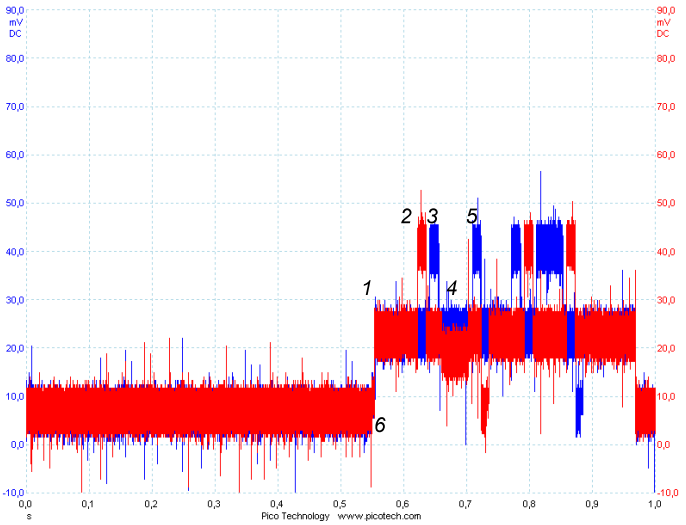

Figura 27 - Captura de una secuencia de eco

La secuencia RTS/CTS/DATA/ACK se sigue perfectamente, en los puntos (2), (3), (4) y (5). En esta captura, el nivel de emisión del dispositivo máster ha sido mínimo, y es curioso observar que en este nivel de potencia se consume menos que simplemente escuchando el medio.

Después de la secuencia emitida por el máster, viene la respuesta del esclavo, que como se puede observar se realiza íntegramente a potencia máxima.

Finalmente, los datos recogidos a lo largo de los 60 minutos que duraba la prueba, han sido procesados, para medir el consumo, mediante la siguiente fórmula:

$$P=\sum_{i=0}^{K}\frac{(V_i)^2}{R}*\Delta t_i$$

Donde *K*, era el numero total de muestras, *R*, en cada caso el valor de la resistencia usada, V_i el valor de voltaje medido, y Δt_i, el intervalo entre las muestras.

El Nodo A siempre transmitía a una potencia constante máxima, mientras que el Nodo B, variaba su potencia de emisión en función de las condiciones del medio.

	Nodo A	Nodo B
Resistencia	1,018 Ω	0,98 Ω
Potencia	0,330738381 W	0,310059665 W
Relación A/B	106,66927 %	

Se observa que, aunque leve, hay una reducción de potencia consumida, dado que el Nodo A presenta para el experimento casi el 7% más de potencia. Con estos datos es prudente decir que hay indicios de que el ahorro de energía es palpable, y que por lo tanto es una estrategia a seguir en las redes de sensores, que como se comenzó hablando en el capítulo introductorio, es un requisito para todo dispositivo que se quiera integrar en esta clase de redes.

Capítulo 8

Conclusiones y líneas futuras

En primer lugar, a tenor de las medidas llevadas a cabo, se puede ser optimista, en cuanto a que hay un ahorro significativo. Sin embargo, las pruebas no han sido todo lo exhaustivas que nos hubiese gustado. Por ello, queda pendiente el realizar pruebas con mas detalle, y variando más parámetros.

Por otro lado, el software usado, TinyOS 1.x, ya empieza a ser abandonado por los desarrolladores, en favor de la nueva rama de TinyOS, la versión 2.x. Algunos de los protocolos que se han comentado en esta memoria, no están disponibles en esta nueva versión, y de igual manera, están surgiendo protocolos programados solo para TinyOS 2.x. Por ello, también seria un cauce a seguir, el implementar este mecanismo en nuevas versiones y protocolos.

Anexo I

Bibliografía

[1] Crossbow Technology
http://www.xbow.com/

[2] TinyOS Community
http://www.tinyos.net/

[3] nesC, A Programming Language for Deeply Networked Systems
http://nescc.sourceforge.net/

[4] XubunTOS
http://toilers.mines.edu/Public/XubunTOS

[5] Prof. dr. K.G. Langendoen
The MAC Alphabet Soup served in Wireless Sensor Networks
http://www.st.ewi.tudelft.nl/~koen/MACsoup/

[6] W. Ye, J. Heidemann and D. Estrin
An Energy-efficient MAC Protocol for Wireless Sensor Networks
21st Conference of the IEEE Computer and Communications Societies (INFOCOM), 2002

[7] J. Polastre, J. Hill and D. Culler
Versatile low power media access for wireless sensor networks
SenSys04, 2004

[8] W. Ye, F. Silva and J. Heidemann
Ultra-Low Duty Cycle MAC with Scheduled Channel Polling
SenSys06, 2006

[9] Marco Avvenuti, Paolo Corsini, Paolo Masci and Alessio Vecchio
Increasing the efficiency of preamble sampling protocols for wireless sensor networks
Dipartimento di Ingegneria dell'Informazione, Università a di Pisa

[10] M. Buettner, G. Yee, E. Anderson and R. Han
X-MAC: A Short Preamble MAC Protocol For Duty-CycledWireless Networks
SenSys06, 2006

[11] Pico Technology
http://www.picotech.com

Anexo II

Análisis del mecanismo de control de potencia en el nivel MAC para redes de sensores

A continuación se anexa un capitulo del trabajo de investigación “Diseño y análisis de mecanismos de ahorro energético en el nivel de acceso al medio en redes de sensores inalámbricas” de Javier Vales Alonso, en la que se detallan algunos aspectos matemáticos sobre el modelo teórico explicado anteriormente.

Capítulo 2

Análisis del mecanismo de control de potencia en el nivel MAC para redes de sensores

Mediante el control de la potencia radiada (*Transmission Power Control*, TPC) idealmente se puede conseguir conformar la topología de una red de sensores, ya que el radio de cobertura es una función de la potencia radiada. El control dinámico de la topología de la red permitiría conseguir dos objetivos: asegurar la conectividad de la red o incrementar la utilización del canal, mediante la reutilización espacial del canal. Un beneficio adicional del control de potencia es el ahorro de energía, ya que el consumo de energía en transmisión es mayor a mayor potencia radiada. A pesar de que el ahorro energético es un requisito fundamental en redes de sensores, el control de potencia de transmisión no ha recibido demasiada atención en la literatura científica hasta el momento. De hecho, el mecanismo más extendido de ahorro de energía es la reducción del ciclo de trabajo, mediante la organización de periodos de actividad/sueño sincronizados. Al utilizar ciclos de trabajo muy reducidos, en los que el tiempo dedicado a transmisión es una fracción todavía menor, se considera habitualmente que el uso de control de potencia no producirá mejoras apreciables.

En este capítulo comenzamos el estudio de mecanismos de ahorro de energía en redes de sensores abordando precisamente esta cuestión. Calcularemos el ahorro de energía que resulta de la aplicación de un mecanismo TPC en el nivel MAC. Este mecanismo es ideal en el sentido de que se asume que los nodos conocen la potencia mínima con la que pueden alcanzar cada uno de sus vecinos (manteniendo una determinada probabilidad de error de paquete). El procedimiento consiste en enviar los paquetes de datos a la potencia mínima requerida, mientras que los paquetes de control se envían a la potencia *nominal* (que suele ser la máxima habitualmente). De esta manera no se modifica la topología de la red respecto a la que se obtendría sin la aplicación del TPC. Esta última propiedad nos permite comparar de manera justa el consumo de energía en ambos casos.

La base del análisis es la evaluación del factor L, definido como la división de la energía consumida por un protocolo con y sin TPC. Para el cálculo de L se tienen en cuenta las propiedades fundamentales de una red de sensores. A saber, la red lleva un largo tiempo

en funcionamiento y en la que el número de nodos es muy grande. Además, la posición de los nodos es aleatoria y no se mueven. Bajo esta condiciones se realiza el cálculo del valor medio de L. Posteriormente, se particulariza para dos protocolos MAC representativos de redes redes de sensores, LMAC [VanH04a] y S-MAC [Ye04], con acceso TDMA y por contienda respectivamente. La conclusión es que los protocolos TDMA son preferibles a la hora de aplicar TPC.

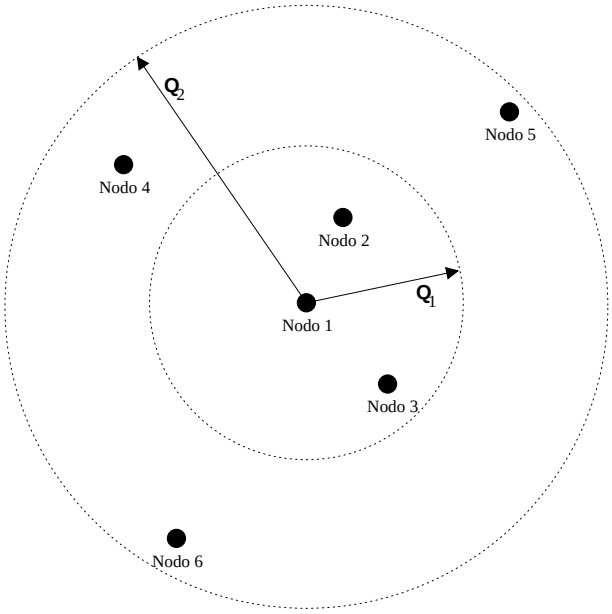

Figura 2.1: Transmisión con niveles discretos de potencia

2.1. Introducción

Como discutimos en el capítulo introductorio las principales fuentes de gasto de energía están relacionadas con el módulo de comunicaciones. El consumo de la interfaz radio depende del estado en el que se encuentre, que puede ser uno de los siguientes:

- Transmisión (tx). El *transceiver* está transmitiendo un paquete. El consumo de energía es proporcional (aunque no de manera lineal) a la potencia radiada. Ésta a su vez, puede ser seleccionada de un conjunto discreto de valores (por ejemplo, de -20 a -5 dBm en pasos de 1 dB en los Mica2 Motes, ver tabla 2.1). Sin embargo, en la práctica la potencia radiada se mantiene en un nivel *nominal* fijo (normalmente, la potencia máxima).

- Recepción o escucha (rx). La interfaz radio está recibiendo un paquete o simplemente escuchando el canal (detección de portadora). En ambos casos el consumo es idéntico, independientemente de si se recibe realmente uan transmisión o no, y de un nivel comparable al consumo en transmisión (por ejemplo, 35.4 mW para los *motes* Mica2).

- Sueño (sl). La radio está apagada. El consumo de energía es despreciable (varios órdenes de magnitud menor que en el resto de estados, por ejemplo, 3 μW para los nodos Mica2).

Desde la perspectiva de la capa MAC, por tanto, el consumo puede ser minimizado si se mantiene a los nodos durmiendo durante los periodos de inactividad, en lugar de estar en recepción. Esta estrategia requiere coordinación entre los nodos y supone un compromiso entre el tiempo de sueño y la utilización del canal, ya que no se puede recibir ni transmitir en este estado. El consumo podría también reducirse si se usara sólo la potencia

necesaria para cada transmisión de datos. La figura 2.1 ilustra esta idea. El nodo 1 alcanza a los nodos 2 y 3 usando una potencia Q_1, mientras que para llegar a los nodos 4, 5 y 6 se necesita una potencia Q_2. En este caso, la potencia Q_2 es la potencia nominal que nos garantiza conectividad total para esta red. Pero, si se usa esta potencia para cualquier transmisión, se malgasta energía cuando se envían paquetes a los nodos 2 y 3. Si seleccionamos la *mejor* potencia radiada para cada transmisión podemos reducir el consumo total de energía. Este tipo de estrategia es lo que se denomina Control de Potencia de Transmisión (*Transmission Power Control*, TPC). Actualmente, sólo la primera opción (coordinación de periodos de sueño) ha sido estudiada exhaustivamente como mecanismo de ahorro de energía. De hecho, es la base de múltiples propuestas de MAC específicos para redes de sensores [Ye04, VanH04a, VanH04b, Dam03].

En este capítulo evaluaremos el rendimiento de un mecanismo TPC muy simple en el nivel MAC, adecuado para la mayoría de las propuestas actuales. El procedimiento TPC en consideración utiliza la mínima potencia necesaria para transmitir los paquetes de datos (con una determinada probabilidad de error de paquete), y la potencia nominal (la máxima disponible) para la transmisión de paquetes de control. Por ejemplo, supongamos que en la red mostrada en la figura 2.1 se utiliza IEEE 802.11, con una secuencia RTS (*Request To Send*)/CTS (*Clear To Send*)/Datos/ACK (*Acknowledgment*). Si el nodo 1 tiene que enviar un paquete al nodo 2, comenzaría transmitiendo un paquete RTS a la potencia máxima Q_2. El nodo 2 respondería con un CTS también a la potencia máxima. A continuación los Datos se enviaría con potencia Q_1. Finalmente el ACK se enviaría con potencia Q_2.

Este enfoque no modifica la topología de la red respecto a la que se formaría si no se utilizara control de potencia, ya que el alcance de los paquetes de control no varía. Es decir, si pudiéramos reproducir exactamente las mismas condiciones en dos redes con y sin TPC cada nodo transmitiría exactamente los mismos paquetes de control, detectaría el canal ocupado en los mismos instantes y no se modificaría el comportamiento de las capas superiores. De esta forma podemos evaluar el ahorro de energía debido exclusivamente al TPC. De hecho, un mecanismo de este tipo es fácilmente implementable en los equipos sensores actuales y directamente aplicable a muchas de las propuestas disponibles. Hay que remarcar que este procedimiento requiere que los nodos conozcan exactamente la potencia necesaria para alcanzar a cada uno de sus vecinos, lo que necesitaría con seguridad de un procedimiento adicional, que probablemente disminuiría el ahorro de energía. Puesto que estamos interesados en el ahorro máximo posible obtenido con un TPC ideal, supondremos que esta información ya es conocida.

Para evaluar este mecanismo se calcula la relación entre la energía consumida sin TPC y con TPC después de que la red esté funcionando durante un periodo de tiempo muy extenso. Denominamos L a este ratio. Como se muestra en las siguientes secciones, dada una determinada distribución de los nodos, L depende de dos factores: uno caracteriza la distribución geométrica de los nodos y el otro está principalmente determinado por el tipo de protocolo MAC y el tráfico de la red. Ambos parámetros se evalúan bajo las asunciones más comúnmente aceptadas en WSN. Para finalizar, este resultado teórico se particulariza para un conjunto de cargas de tráfico y densidad de nodos con dos protocolos MAC: uno de tipo TDMA (LMAC) [VanH04a] y otro de contienda (S-MAC) [Ye04].

En el resto del capítulo se utilizará la siguiente notación:

- Las probabilidades se expresan como Pr[Evento].
- Las variables aleatorias (va) se expresan como $\boldsymbol{x}$.
- Los valores medios se expresan como $\overline{\boldsymbol{x}}$ o $E\{x\}$.
- Los procesos estocásticos se expresan como $x(t)$.
- Las potencias de transmisión se expresan con la letra Q.
- Los consumos de potencia se expresan con la letra P.

2.2. Trabajos relacionados

El objetivo de los mecanismos de control de potencia es la selección del nivel de potencia de transmisión *óptimo*. Un nivel óptimo que depende, de hecho, del ámbito y propósito final al que sirve la utilización del TPC. Para redes MANET, los mecanismos de TPC se diseñan habitualmente con la intención de incrementar la utilización del medio inalámbrico, gracias a la reutilización espacial del canal, o para asegurar la conectividad de la red. Se consideran dos tipos de estrategias:

1. TPC en el nivel de red. En este nivel el control de potencia se utiliza principalmente para elegir el mejor subconjunto de vecinos de entre todos los presentes, es decir, para controlar la topología de la red. COMPOW [Kaw03] y PSP [Yu04] son dos protocolos representativos. En ambos casos, el resultado es que un nodo selecciona *un único* nivel de potencia con el que realiza todas sus transmisiones.

2. TPC en el nivel MAC. En este nivel, la potencia se elige *para cada paquete* con tal de conseguir reutilización del canal o reducir la probabilidad de colisión del paquete. Hay numerosas propuestas en este sentido [Jun05, Muq03, Mon01]. Las dos últimas están basadas en dispositivos multicanal ya que usan un canal adicional para señalizar futuras transmisiones y calcular el mejor nivel de potencia de transmisión. La primera necesita equipos radio especializados capaces de variar muy rápidamente la potencia de transmisión.

Estas soluciones efectivamente reducen las colisiones y mejoran la utilización, aspectos muy importantes en MANET. En redes de sensores, por el contrario, prima el ahorro energético. Además, los protocolos para WSN deben tener en cuenta los recursos *hardware* limitados de los sensores, lo que excluye en la práctica el uso de alguna de las propuestas referidas anteriormente.

Recientemente se pueden encontrar también algoritmos diseñados específicamente para WSN. En [Kub03] se describe un conjunto de algoritmos TPC distribuidos, que seleccionan un único nivel de potencia para cada nodo. Estas propuestas se comparan entre sí, utilizando como métricas la conectividad de la red y el tiempo de vida. Sin embargo, no hay una evaluación de las mejoras respecto a no usar TPC. Lo cual nos parece un aspecto importante, puesto que la complejidad adicional necesaria para implementar un TPC difícilmente se justifica si realmente no se obtienen beneficios apreciables en ahorro de energía.

En cuanto a estudios analíticos relacionados, en [Tak84] los autores calculan el radio de transmisión óptimo que maximiza el progreso de un paquete en saltos en una determinada dirección (hacia el destino). La energía consumida no se considera. En este trabajo se asume también que la posición de los terminales es aleatoria. Más recientemente, en [Gom04] se realiza una comparación analítica entre las ventajas de un TPC de radio variable respecto a los TPC de radio único para MANET. De nuevo, la evaluación se centra en el impacto del TPC en la conectividad de la red, la utilización y los protocolos de enrutamiento. Finalmente, en [Che03] sí que se utiliza un modelo de consumo de energía, que depende del tamaño de paquete, el protocolo MAC y característica del canal radio, para obtener una potencia de transmisión óptima en términos de consumo de energía extremo a extremo, es decir, consumo en los nodos que intervienen en los saltos de un paquete hasta el destino.

Con nuestro enfoque podemos calcular la mejora que un TPC puede proporcionar respecto a la transmisión con un nivel *nominal*, que no tiene que ser la máxima potencia de transmisión disponible. Esto permite combinar el mecanismo genérico usado en esta evaluación con otras propuestas, como las centradas en el nivel de red. La topología de la red con el mecanismo referido viene determinada por el alcance de los paquetes de control, que se transmiten con la potencia nominal. En nuestro análisis se ha fijado la potencia nominal a la máxima disponible. Sin embargo, nada impide que la potencia nominal se elija de forma que se obtenga una propiedad deseada para la red, como conectividad total, al tiempo que el TPC se utiliza para ahorrar energía en cada transmisión.

2.3. Análisis del ahorro

En esta sección se calcula una expresión para el ahorro energético que resultaría de la aplicación del mecanismo TPC genérico apuntado anteriormente.

Sea n el número de nodos en la red. Sean $T^i_{tx}(t)$, $T^i_{rx}(t)$ y $T^i_{sl}(t)$ procesos estocásticos que representan el tiempo acumulado que, durante el intervalo [0,t), un nodo i, para i=1..n, está en cada estado de los definidos en la sección 2.1, es decir, transmisión (tx), recepción (rx) y sueño (sl). Entonces, podemos definir el tiempo total acumulado en la red en cada estado $T_{tx}(t)$, $T_{rx}(t)$ y $T_{sl}(t)$ si sumamos las contribuciones de cada nodo:

$$\begin{aligned} T_{tx}(t) &= \sum_{i=1}^{n} T^i_{tx}(t) \\ T_{rx}(t) &= \sum_{i=1}^{n} T^i_{rx}(t) \\ T_{sl}(t) &= \sum_{i=1}^{n} T^i_{sl}(t) \end{aligned} \tag{2.1}$$

De manera similar, sean P_{tx}, P_{rx} y P_{sl} el consumo de potencia asociado a cada estado. Entonces, la energía total consumida en la red hasta un instante arbitrario t, $E(t)$ viene dada por la ecuación (2.2).

$$E(t) = P_{tx}T_{tx}(t) + P_{rx}T_{rx}(t) + P_{sl}T_{sl}(t) \tag{2.2}$$

Deberíamos en este punto definir una forma de medir la mejora en la eficiencia energética debida a la aplicación del mecanismo TPC a un protocolo MAC en particular. Por ejemplo, mediante un parámetro como el "tiempo de vida" de la red. Sin embargo, no está claro cuándo una red no puede continuar su funcionamiento correcto, ya que dependerá de su aplicación. En su lugar, podemos calcular una métrica de la eficiencia del mecanismo TPC basada en la relación asintótica para un tiempo muy largo t entre el consumo de energía de la red cuando se usa TPC y cuando no se usa. Llamamos L a esta tasa que definimos mediante la ecuación (2.3).

$$L = \lim_{t\to\infty} \frac{E(t)|_{\text{no-TPC}}}{E(t)|_{\text{TPC}}} \tag{2.3}$$

Esta ecuación se puede desarrollar más si tenemos en cuenta la asunción de que los paquetes de control se envían siempre a la máxima potencia (nominal). En este caso, ni la topología ni $T^i_{tx}(t)$, $T^i_{rx}(t)$ y $T^i_{sl}(t)$ cambian cuando se utiliza TPC.

Supongamos que la red se compone de sensores homogéneos con p niveles de transmisión de potencia posibles (Q_j, $j = 1, \ldots, p$). Sea P_{tx_i} el consumo de potencia asociado a cada nivel de potencia de transmisión Q_i. Sea Q_p la potencia máxima (nominal) y, por tanto, P_{tx_p} su consumo asociado.

Entonces, nótese que podemos descomponer el tiempo de transmisión total de cada nodo en dos contribuciones: datos y señalización de control. Definamos los siguientes procesos estocásticos $T^i_{data}(t)$, $T^i_{sign}(t)$ para $i = 1, \ldots, n$ como el tiempo de transmisión acumulado dedicado a datos y señalización respectivamente para cada nodo. Como en la ecuación (2.1) sean $T_{data}(t) = \sum_{i=1}^{n} T^i_{data}(t)$, $T_{sign}(t) = \sum_{i=1}^{n} T^i_{sign}(t)$. Además, definimos como $T^i_{data_j}(t)$ el tiempo hasta el instante t que un nodo i ha pasado transmitiendo datos con un nivel j para $i = 1, \ldots, n$ y $j = 1, \ldots, p$. Finalmente, $T_{data_j}(t) = \sum_{i=1}^{n} T^i_{data_j}(t)$ para $j = 1, \ldots, p$.

Entonces, si despreciamos el consumo durante el periodo de sueño ya que es varios órdenes de magnitud menor, tenemos que la energía consumida con y sin TPC viene dada por las ecuaciones (2.4) y (2.5) respectivamente.

$$E(t)|_{\text{TPC}} = \sum_{j=1}^{p} T_{data_j}(t) P_{tx_j} + T_{sign}(t) P_{tx_p} + T_{rx}(t) P_{rx} \tag{2.4}$$

$$E(t)|_{\text{no-TPC}} = T_{data}(t) P_{tx_p} + T_{sign}(t) P_{tx_p} + T_{rx}(t) P_{rx} \tag{2.5}$$

Por tanto, de las ecuaciones (2.3), (2.4) y (2.5):

$$L = \lim_{t\to\infty} \frac{T_{data}(t) P_{tx_p} + T_{sign}(t) P_{tx_p} + T_{rx}(t) P_{rx}}{\sum_{j=1}^{p} T_{data_j}(t) P_{tx_j} + T_{sign}(t) P_{tx_p} + T_{rx}(t) P_{rx}} \tag{2.6}$$

L puede ser reescrito dividiendo tanto el numerador como el denominador por $P_{tx_p} T_{data}(t)$; obtenemos:

$$L = \lim_{t\to\infty} \frac{1 + \frac{T_{sign}(t)}{T_{data}(t)} + \frac{P_{rx} T_{rx}(t)}{P_{tx_p} T_{data}(t)}}{\sum_{j=1}^{p} \frac{P_{tx_j} T_{data_j}(t)}{P_{tx_p} T_{data}(t)} + \frac{T_{sign}(t)}{T_{data}(t)} + \frac{P_{rx} T_{rx}(t)}{P_{tx_p} T_{data}(t)}} \tag{2.7}$$

Definamos ahora la variable ξ como,

$$\xi = \lim_{t\to\infty}[\frac{T_{sign}(t)}{T_{data}(t)} + \frac{P_{rx}}{P_{tx_p}}\frac{T_{rx}(t)}{T_{data}(t)}] \tag{2.8}$$

Ahora, sea $\boldsymbol{Q}$ la variable aleatoria "potencia de transmisión" que asigna a cada nivel de potencia de transmisión la probabilidad Pr[$\boldsymbol{Q} = Q_j$] correspondiente. Es decir, esta variable representa la probabilidad de que una transmisión de datos ocurra en el i-ésimo cuanto de transmisión. Nótese que:

$$\text{Pr}[\boldsymbol{Q} = Q_j] = \lim_{t\to\infty}\frac{T_{data_j}(t)}{T_{data}(t)} \tag{2.9}$$

Por tanto:

$$L = \frac{1+\xi}{\sum_{j=1}^{p}\frac{P_{tx_j}}{P_{tx_p}}\text{Pr}[Q{=}Q_j] + \xi} \tag{2.10}$$

Si adicionalmente llamamos $s = \sum_{j=1}^{p}\frac{P_{tx_j}}{P_{tx_p}}\text{Pr}[Q{=}Q_j]$ llegamos a la expresión final:

$$L = \frac{1+\xi}{s+\xi} \tag{2.11}$$

Esta expresión indica que, dada una red, la relación entre ambos consumos de energía converge después de un tiempo largo de funcionamiento a un valor que es una función de dos variables: s y ξ.

Por una parte, el coeficiente s caracteriza la distribución geométrica de la red ya que depende de la distancia relativa entre vecinos, la cual determina el nivel de potencia necesario. Además, los valores de s están siempre dentro del intervalo (0,1]. Valores bajos indican que los nodos están cercanos, y los ahorros son por tanto significativos (valores grandes de L). Valores altos de s indican que es improbable que el cuanto de transmisión seleccionado pueda ser reducido. De hecho, si $s = 1$ significa que no es posible ningún tipo de ahorro. En este caso extremo el uso del TPC carece de interés.

Por otro lado, el coeficiente ξ mide el balance existente entre el tiempo dedicado a la transmisión de los datos y el tiempo empleado para señalización y recepción. Nótese que $\partial L/\partial\xi = (s-1)/(s+\xi)^2 < 0$ para todo $s \in (0,1)$ Por lo tanto, L decrece a medida que ξ aumenta. Los protocolos con un buen balance posee un bajo valor de ξ, proporcionando entonces mayores L, es decir, mayores ahorros.

Por último, debemos indicar que ambos coeficientes están influenciados por las características del tráfico, y que si la posición de los nodos varía o el número de sensores cambia estas variables (ξ y s) también cambiarán, y con ellas el factor L. Es decir, L es una función del número de nodos y de sus posiciones. Por tanto, si la posición de los nodos es aleatoria, la tasa L es también aleatoria. En las próximas secciones calcularemos la media de L, $\overline{\boldsymbol{L}}$. Veremos que para valores grandes de n, es decir el caso de las redes de sensores, $\overline{\boldsymbol{L}}$ viene dado por la expresión:

$$\overline{\boldsymbol{L}} = \frac{1 + \overline{\boldsymbol{\xi}}}{\overline{\boldsymbol{s}} + \overline{\boldsymbol{\xi}}} \tag{2.12}$$

Para poder seguir avanzando en el estudio debemos concretar nuestro modelo de red de sensores. En las próximas secciones calcularemos $\overline{\boldsymbol{s}}$ y $\overline{\boldsymbol{\xi}}$ (y, por tanto, $\overline{\boldsymbol{L}}$) con el modelo de WSN que se detalla en la sección 2.4.

2.4. Modelo de red de sensores

Para calcular la función general L obtenida en la sección anterior, necesitamos definir un modelo representativo de red de sensores. Este modelo viene especificado por:

- Un modelo de propagación adecuado.
- La distribución de los nodos.
- Un modelo de tráfico.
- Esquema de acceso múltiple

2.4.1. Modelo de propagación

El medio de transmisión puede considerarse un canal de banda estrecha invariante en el tiempo, que puede ser modelado empleando una aproximación de *path-loss* [Rap02]. En ese caso, a fin de no superar una probabilidad de transmisión errónea objetivo ($\widehat{p_e}$), la potencia de transmisión mínima necesaria ($\widehat{P_{tx}}$) is:

$$\widehat{P_{tx}} = d^{\alpha}\Omega(\widehat{p_e}) \tag{2.13}$$

siendo α el coeficiente de propagación para el modelo de path-loss y Ω una función que depende de $\widehat{p_e}$ y del entorno de comunicación. El apéndice A describe detalladamente como se ha obtenido dicha expresión.

Por ejemplo, para los nodos Mica2, un tamaño de paquete de 100 bytes, y una probabilidad de error objetivo máxima $\widehat{p_e} = 10^{-3}$, $\alpha = 3{,}95$ (este valor de α ha sido obtenido experimentalmente en nuestros despliegues de redes de sensores, ver [Ege05a]), y una tasa binaria de 30 Kbps, obtenemos $\Omega \approx$ -97.5 dB*. Para los valores discretos de la potencia de transmisión de los nodos Mica2, las distancias asociadas usando los parámetros anteriores se muestran en la tabla 2.1.

Alcance máximo. Adicionalmente, los receptores tienen una sensibilidad ($P_S \approx -102$ dBm para los Mica2) que establece la máxima distancia entre nodos (d_S). Por ejemplo, con los parámetros anteriores, la distancia máxima asociada d_S es de 89.92 m.

*Este valor ha sido calculado para una codificación Manchester, que dobla la tasa en baudios.

Potencia de transmisión	Consumo	Rango de transmisión
-20 dBm (0.0100 mW)	25.8 mW	19.30m
-19 dBm (0.0126 mW)	26.4 mW	20.46 m
-18 dBm (0.0158 mW)	27.0 mW	21.69 m
-17 dBm (0.0200 mW)	27.0 mW	22.99 m
-16 dBm (0.0251 mW)	27.3 mW	24.38 m
-15 dBm (0.0316 mW)	27.9 mW	25.84 m
-14 dBm (0.0398 mW)	27.9 mW	27.39 m
-13 dBm (0.0501 mW)	28.5 mW	29.03 m
-12 dBm (0.0631 mW)	29.1 mW	30.78 m
-11 dBm (0.0794 mW)	29.7 mW	32.62 m
-10 dBm (0.1000 mW)	30.3 mW	34.58 m
-9 dBm (0.1259 mW)	31.2 mW	36.66 m
-8 dBm (0.1585 mW)	31.8 mW	38.86 m
-7 dBm (0.1995 mW)	32.4 mW	41.19 m
-6 dBm (0.2512 mW)	33.3 mW	43.67 m
-5 dBm 0.3162 mW)	41.4 mW	46.29 m
-4 dBm (0.3981 mW)	43.5 mW	49.07 m
-3 dBm (0.5012 mW)	43.5 mW	52.01 m
-2 dBm (0.6310 mW)	45.3 mW	55.13 m
-1 dBm (0.7943 mW)	47.4 mW	58.44 m
+0 dBm (1.0000 mW)	50.4 mW	61.95 m
+1 dBm (1.2589 mW)	51.6 mW	65.67 m
+2 dBm (1.5849 mW)	55.5 mW	69.61 m
+3 dBm (1.9953 mW)	57.6 mW	73.79 m
+4 dBm (2.5119 mW)	63.9 mW	78.22 m
+5 dBm (3.1623 mW)	76.2 mW	82.92 m

Tabla 2.1: Consumos y rangos de cobertura con un modelo de path-loss de parámetro α = 3.95 para los sensores Mica2

2.4.2. Distribución de los nodos

En un caso general de despliegue de red de sensores la posición los nodos no puede ser controlada y es desconocida *a priori*. Podría ser el caso, por ejemplo, de lanzar los nodos desde un avión. Consideraremos un patrón de posición razonable, en el que los nodos se concentran alrededor de un *punto de interés*. Como ejemplo, supongamos una zona de desastre sobre la que se han lanzado los nodos. Una forma sencilla de modelar esta situación es seleccionar la posición de los nodos mediante una distribución normal bidimensional alrededor de las coordenadas del foco de interés. El parámetro de la desviación típica σ controla la dispersión. Un ejemplo de este tipo de distribución se muestra en la figura 2.2, para $n = 250$ nodos y $\sigma = 100$.

Supongamos, sin pérdida de generalidad, que el punto de interés está situado en el centro del plano real. Entonces, la posición del i-ésimo nodo es ($\boldsymbol{X_i}$, $\boldsymbol{Y_i}$), siendo $\boldsymbol{X_i}, \boldsymbol{Y_i}$ variables normales $N(0, \sigma)$ independientes e idénticamente distribuidas. La distancia cuadrática entre dos nodos i y j es $\boldsymbol{d^2} = \boldsymbol{\Delta X}^2 + \boldsymbol{\Delta Y}^2$, siendo $\boldsymbol{\Delta X} = \boldsymbol{X_i} - \boldsymbol{X_j}$, $\boldsymbol{\Delta Y} = \boldsymbol{Y_i} - \boldsymbol{Y_j}$. La función de densidad de probabilidad de $\boldsymbol{d^2}$ es (ver apéndice B):

$$f_{d^2}(x) = \frac{1}{4\sigma^2} \exp(-\frac{x}{4\sigma^2}) \tag{2.14}$$

2.4.3. Patrón de tráfico

En una red de sensores típica se supone que los nodos transportan flujos de tráfico desde los sensores hasta los sumideros. Éstos últimos son nodos especiales que reciben y

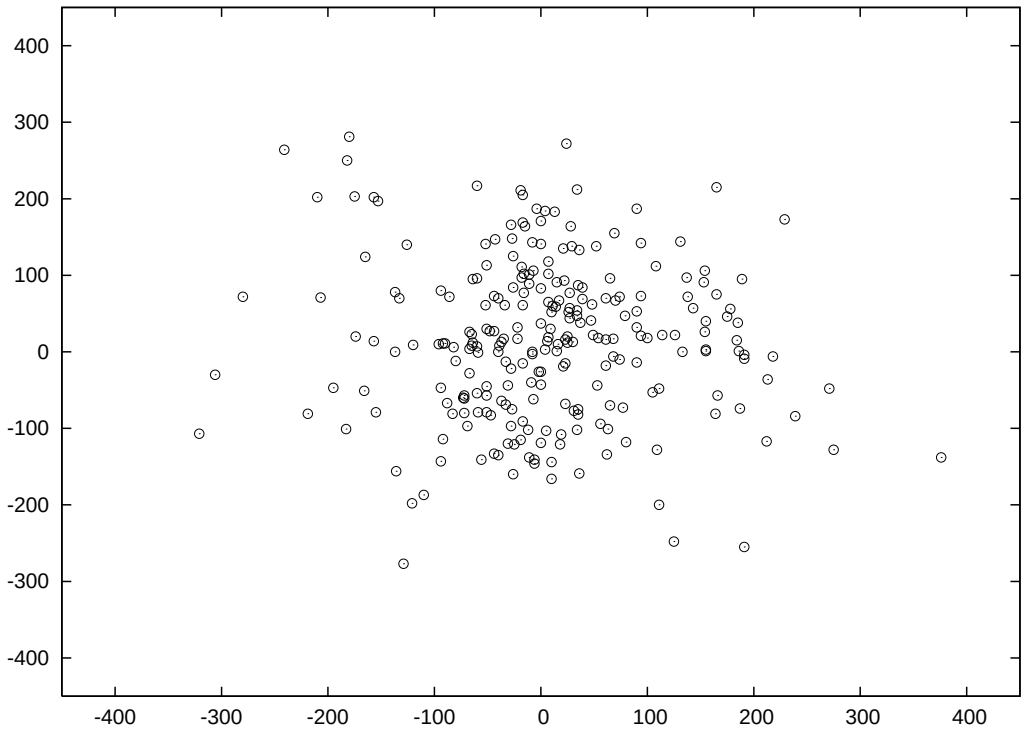

Figura 2.2: 250 nodos posicionados según una distribución bidimensional Normal de parámetro σ = 100

procesan los datos de todos los sensores en la red. Por tanto, los nodos próximos al sumidero transmiten más tráfico, ya que reenvían la información que les va llegando de otros nodos. La carga media en la red se suele considerar baja y el tamaño de paquete pequeño (alrededor de 100 bytes). Utilizando todas estas premisas, y con el objetivo de simplificar el análisis, aportaremos las siguientes consideraciones a nuestro modelo analítico:

- El número de nodos es muy grande.

- Los nodos sumidero se sitúan cerca del centro del plano. Por tanto, el tráfico fluye desde los extremos al centro.

- La utilización de los enlaces es uniforme en todos los enlaces de la red e igual a ρ. El uso de un modelo de propagación determinista permite definir el concepto de enlace sin ambigüedad. Un nodo posee un enlace con cada uno de sus vecinos en alcance. Puesto que un enlace es simétrico, cada uno de los nodos que forman el enlace aportará la mitad de la carga del mismo. Esta simplificación del modelo de tráfico concuerda con la idea de que existe más tráfico en los nodos centrales que en los extremos, ya que los nodos centrales tendrán más enlaces (debido a que se han posicionado con una distribución normal). Además, esta asunción es clave para poder determinar analíticamente la tasa L.

2.4.4. Esquema de acceso múltiple

Consideraremos un modelo genérico en el cual el tiempo está dividido en ranuras temporales, compuestas de un periodo activo para transmisión/recepción y un periodo pasivo para dormir. Ésta es la aproximación más común en redes de sensores, y es utilizada en una gran variedad de protocolos tanto deterministas (TDMA) [VanH04a, VanH04b] como basados en contienda (variantes de CSMA) [Ye04, Dam03]. En nuestro análisis asumiremos que no se producen colisiones. Tal asunción puede justificarse claramente en el caso de TDMA, en los protocolos de contienda la asunción de baja carga discutida en el apartado anterior nos permite considerar una colisión como un evento poco probable. De hecho, si tal condición no pudiese aplicarse a una propuesta tipo CSMA (y las colisiones ocurren de modo frecuente) el valor obtenido para L será una cota superior de la eficiencia alcanzable por el algoritmo.

En nuestro modelo las longitudes están expresadas en bits, y cada ranura consiste en (ver figura 2.3):

- Preámbulo (B_p bits). El preámbulo se utiliza normalmente para la recuperación de las derivas del reloj, descubrimiento de nuevos vecinos y funciones de mantenimiento. Todos los nodos en alcance deben escuchar los B_p bits del preámbulo para recibir los preámbulos transmitidos por otros nodos. El preámbulo se envía cíclicamente, es decir, cada nodo selecciona su ranura y retransmite su preámbulo cada C ranuras. Sin embargo, es posible que la transmisión del preámbulo ocupe menos de B_p bits (por ejemplo, si se usa contienda, debe existir tiempo adicional para acomodar la ventana de contienda). Por tanto, distinguiremos como B'_p la longitud del preámbulo que se transmite realmente. Siempre se envía a la potencia nominal.

- Periodo de escucha/notificación (B_l bits). Su propósito es el de notificar a los nodos el comienzo de una nueva transmisión (por ejemplo, la secuencia RTS/CTS se incluiría dentro de este periodo). Todos los nodos se mantienen a la escucha durante este periodo. Si no reciben ninguna notificación o no son el destino de la futura transmisión van a dormir hasta la siguiente ranura. Como en el caso del preámbulo, denotaremos B'_l al tamaño real en bits de los paquetes transmitidos. Si un nodo tiene que transmitir datos, informa de esta transmisión durante esta fase usando la potencia nominal.

- Periodo de intercambio de mensajes (B bits de datos más B_a bits auxiliares). En este periodo, los paquetes de datos e información auxiliar se intercambian (por ejemplo la secuencia Datos/ACK). Si se emplea el TPC, los mensajes de Datos se envían con la mínima potencia requerida para alcanzar la probabilidad de error objetivo en el receptor. Los paquetes auxiliares se envían siempre a la potencia nominal.

- Periodo de sueño. Después del intercambio de paquetes los nodos van a dormir hasta la próxima ranura. En esta fase el consumo de energía es despreciable.

Nótese que para cada etapa, en los protocolos de tipo TDMA sólo un nodo transmite a la vez en la misma región espacial de la red (obviamente, las ranuras temporales pueden reutilizarse espacialmente sin existencia de colisiones). En el caso de los protocolos

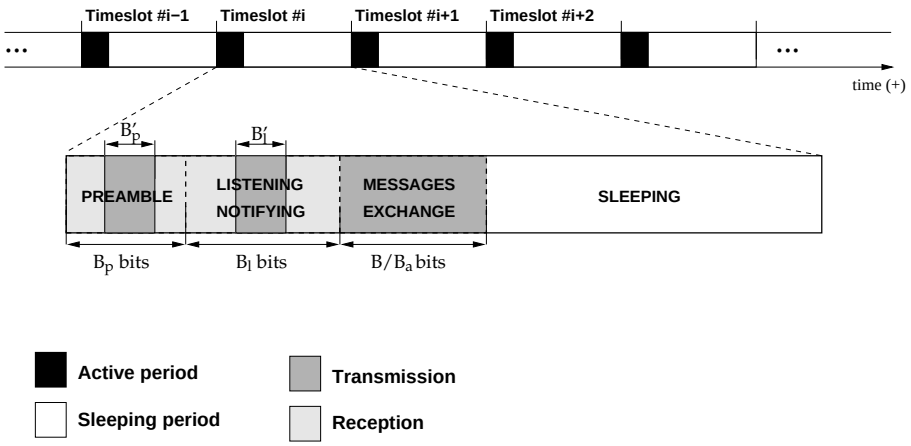

Figura 2.3: Composición de una ranura genérica

basados en CSMA todos los nodos con información para ser transmitida compiten para acceder al canal.

2.5. Evaluación de $\overline{\xi}$

Sea $c(t)$ el número de ranuras transcurridas hasta el instante t. Nótese que $c(t)$ es el mismo para todos los nodos en la red. Llamemos K al número de enlaces que existen en la red. Sea $m(t)$ un proceso estocástico que representa el número de paquetes enviados en cada uno de los K enlaces hasta el instante t. Entonces , $m(t) = c(t)\rho$. Además, sea $v^i(n)$ una variable aleatoria que representa el número de vecinos del nodo i-ésimo, para $i = 1, \ldots, n$. Obviamente, esta variable depende de n. Definamos $v(n) = \sum_{i=1}^{n} v^i(n)$ como la suma de todos los vecinos. Nótese que $v(n) = 2K$, ya que cada nodo vecino corresponde a un enlace en la red, que se cuenta dos veces en la suma. Entonces, se cumple la siguiente relación:

$$c(t) = 2\frac{Km(t)}{v(n)\rho} \tag{2.15}$$

Además, ya es posible en este punto el cálculo de la cantidad de tiempo que los nodos están en cada estado. Sea ν el tiempo de transmisión de un bit del *transceiver*. Hasta el instante t se han enviado $m(t)$ paquetes en cada enlace y han transcurrido $c(t)$ ranuras en cada nodo. Entonces, las contribuciones de cada tiempo son las siguientes:
Transmisión de datos: B bits por cada paquete enviado. $Km(t)$ paquetes en total.

$$T_{data}(t) = B\nu Km(t) \tag{2.16}$$

Transmisión de señalización: B'_p bits cada C ranuras por nodo ($c(t)$ en total). $B'_l + B_a$ bits por cada paquete enviado.

$$T_{sign}(t) = \frac{B'_p\nu}{C}nc(t) + (B'_l + B_a)\nu Km(t) \tag{2.17}$$

Recepción: B_p bits durante $C-1$ de cada C ranuras (en la ranura restante el nodo envía su propio preámbulo, y escucha sólo durante $B_p - B'_p$ bits). B_l bits por cada ranura en la que el propio nodo no transmite (si el nodo transmite en la ranura entonces permanece a la escucha sólo $B_l - B'_l$ bits). $B + B_a$ bits por cada paquete recibido.

$$\begin{aligned} T_{rx}(t) &= \nu\frac{B_pC - B'_p}{C}nc(t) + B_l\nu nc(t) + \\ &- B'_l\nu Km(t) + (B + B_a)\nu Km(t) \\ &= nc(t)\nu(\frac{B_pC - B'_p}{C} + B_l) + Km(t)\nu(B + B_a - B'_l) \end{aligned} \tag{2.18}$$

Utilizando estas ecuaciones y la ecuación (2.15), nótese que ν desaparece ya que está en el numerador y denominador, tenemos que :

$$\begin{aligned} \frac{T_{sign}(t)}{T_{data}(t)} &\approx \frac{B'_p\frac{n}{C}c(t) + (B'_l + B_a)Km(t)}{BKm(t)} = \\ &= \frac{2B'_p}{BC\rho}\frac{n}{v(n)} + \frac{B'_l + B_a}{B} \end{aligned} \tag{2.19}$$

y,

$$\begin{aligned} \frac{T_{rx}(t)}{T_{data}(t)} &\approx \frac{nc(t)(\frac{B_pC - B'_p}{C} + B_l) + Km(t)(B + B_a - B'_l)}{BKm(t)} = \\ &= 2\frac{\frac{B_pC - B'_p}{C} + B_l}{B\rho}\frac{n}{v(n)} + \frac{B + B_a - B'_l}{B} \end{aligned} \tag{2.20}$$

Estas ecuaciones son aproximaciones, ya que los paquetes pueden estar parcialmente transmitidos en el instante t. Sin embargo, estas aproximación es apropiada para valores grandes de t. Entonces, cuando t tiende a infinito, ξ es igual a

$$\begin{aligned} \xi =& \frac{2B'_p}{BC\rho}\frac{n}{v(n)} + \frac{B'_l + B_a}{B} + \\ &+ \frac{P_{rx}}{P_{tx_p}}[2\frac{\frac{B_pC - B'_p}{C} + B_l}{B\rho}\frac{n}{v(n)} + \frac{B + B_a - B'_l}{B}] \end{aligned} \tag{2.21}$$

Resumiendo, nótese que el coeficiente ξ depende de la relación entre potencia de recepción y potencias de transmisión, que es un parámetro del *hardware*, y de la carga de tráfico y las longitudes de los mensajes, que son parámetros del protocolo. Además ξ es una función de $n/v(n)$, que, a su vez, es una variable aleatoria para topologías aleatorias como la considerada.

Proposición 1 *La media de la variable aleatoria $n/v(n)$ viene expresada por:*

$$E\{\frac{n}{v(n)}\} = \frac{1}{\overline{\boldsymbol{v}}} = \frac{1}{(n-1)[1-\exp{(-\frac{d_S^2}{4\sigma^2})}]} \tag{2.22}$$

donde $\overline{\boldsymbol{v}}$ denota el número medio de vecinos de un nodo en una red de n nodos.

La expresión de $\overline{\boldsymbol{v}}$ en nuestro modelo corresponde al número medio de vecinos que tiene un nodo cuando los nodos se posicionan según una distribución normal. Su valor se obtiene de la siguiente manera: sea n el número total de nodos en la red y $\boldsymbol{v}$ la variable aleatoria "número de vecinos de un nodo", y puesto que la posición de los nodos se elige de manera independiente, tenemos que:

$$\Pr[\boldsymbol{v} = i] = \binom{n-1}{i}\eta^i(1-\eta)^{(n-1-i)}, \text{para } i\text{=0, } \ldots, n\text{-1} \tag{2.23}$$

donde η es la probabilidad de que dos nodos elegidos de manera independiente sean vecinos, es decir, la probabilidad de que la distancia entre dos nodos (d) sea menor que d_S (sección 2.4.1). De la fdp de d^2 obtenida en la sección 2.4.2, sabemos que $\eta = \Pr[d^2 \leq d_S^2] = 1 - \exp{(-\frac{d_S^2}{4\sigma^2})}$. Además, el momento de primer orden de $\boldsymbol{v}$ será:

$$\overline{\boldsymbol{v}} = \sum_{i=1}^{n-1} i\binom{n-1}{i}\eta^i(1-\eta)^{(n-1-i)} = \eta(n-1) \tag{2.24}$$

Sin embargo, aunque podemos calcular el valor de $\overline{\boldsymbol{v}}$, no tenemos una prueba directa de (2.22). En su lugar, se ha comprobado su corrección mediante cálculo numérico. En la figura 2.4 se muestran algunos de los resultados. Para valores de σ de 50, 100 y 150 se han lanzado redes aleatorias de n nodos y se ha calculado la media y la desviación típica de $n/v(n)$ utilizando 10000 muestras. Las líneas en la figura 2.4 representan la media $\pm$ la desviación típica obtenida en los experimentos, mientras que los puntos representan el cálculo analítico de $1/\overline{\boldsymbol{v}}$. Como se puede observar, la probabilidad se concentra alrededor del valor esperado a medida que n crece.

Usando la expresión previa, el valor medio de ξ viene dado por

$$\overline{\boldsymbol{\xi}} = \frac{2B'_p}{BC\rho}\frac{1}{\overline{\boldsymbol{v}}} + \frac{B'_l + B_a}{B} + \frac{P_{rx}}{P_{tx_p}}[2\frac{\frac{B_pC - B'_p}{C} + B_l}{B\rho}\frac{1}{\overline{\boldsymbol{v}}} + \frac{B + B_a - B'_l}{B}] \tag{2.25}$$

La figura 2.5 representa $\overline{\boldsymbol{\xi}}$ evaluado respecto a ρ para diferentes tamaños del paquete de Datos (B) usando $n = 100$ nodos. Los parámetros utilizados para calcular estas gráficas son B_p = B'_p = 100 bits, B_l = B'_l = 400 bits, B_a = 400 bits, $C = 20$ y P_{rx} = 35.2 mW, P_{tx_p} = 76.2 mW. Las longitudes de paquete se han elegido así por ser valores representativos de una configuración de WSN y el consumo de potencia corresponde a los valores reales de consumo de un Mica2. Las curvas muestran que el valor esperado de ξ disminuye a medida que ρ se incrementa y que su valor es mayor para valores menores de B.

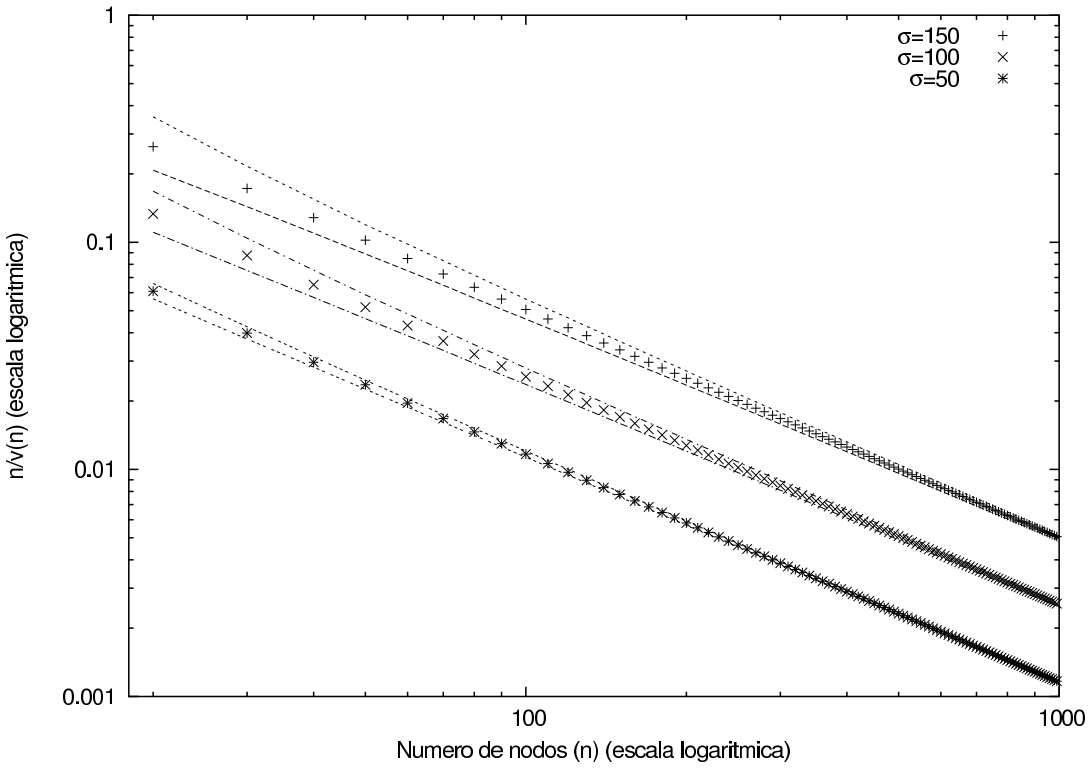

Figura 2.4: $n/v(n)$ evaluado respecto a n para diferentes σ

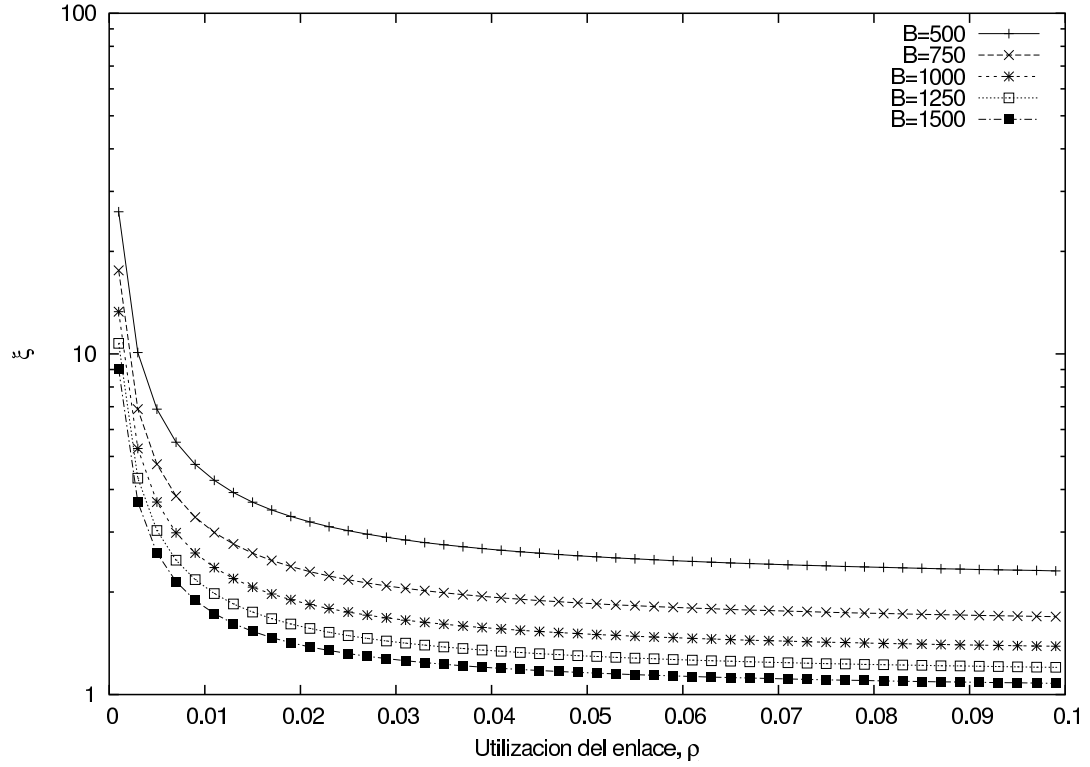

Figura 2.5: $\overline{\boldsymbol{\xi}}$ evaluado respecto a ρ para diferentes tamaños de paquete (B)

2.6. Evaluación de $\overline{s}$

Como se mostró en la sección 2.3, el ahorro de energía depende del valor del coeficiente s. Este coeficiente es una función de la función de densidad de probabilidad de la variable aleatoria $\boldsymbol{Q}$, introducida en la sección 2.3:

$$s = \sum_{j=1}^{p} \frac{P_{tx_j}}{P_{tx_p}} \text{Pr}[Q{=}Q_j] \tag{2.26}$$

De hecho, s es en realidad una función de la posición de los nodos y del número de nodos. Ya que la posición es aleatoria, s es también aleatoria.

Proposición 2 *La media de la variable aleatoria s es*

$$\begin{aligned}\overline{\boldsymbol{s}} = & \frac{\sum_{j=1}^{j=p-1} P_{tx_j}[\exp{(-\frac{(\frac{Q_{j-1}}{\Omega})^{2/\alpha}}{4\sigma^2})} - \exp{(-\frac{(\frac{Q_j}{\Omega})^{2/\alpha}}{4\sigma^2})}]}{P_{tx_p}[1-\exp{(-\frac{d_S^2}{4\sigma^2})}]} + \\ & + \frac{\exp{(-\frac{(\frac{Q_{p-1}}{\Omega})^{2/\alpha}}{4\sigma^2})} - \exp{(-\frac{d_S^2}{4\sigma^2})}}{[1-\exp{(-\frac{d_S^2}{4\sigma^2})}]}\end{aligned} \tag{2.27}$$

El propósito de esta sección es demostrar la proposición anterior.

Demostración:

Una transmisión de datos entre dos nodos con TPC utiliza el nivel de potencia Q_j si la distancia entre dos nodos está en el intervalo $D_j = [d_{min_j}, d_{max_j})$. Si la utilización del un enlace es equiprobable, entonces Pr[$\boldsymbol{Q} = Q_j$] es simplemente la probabilidad de que la distancia entre dos nodos esté en el intervalo D_j. Los algoritmos de TPC en el nivel MAC deben seleccionar un valor de potencia de transmisión que garantice una probabilidad de error dada $\widehat{p_e}$ en el receptor (o equivalentemente, que la SNR sea mayor que un determinado umbral). Si Pr[Nodos en alcance] es la probabilidad de que los nodos estén en alcance. Sea $Q_0 = 0$. Entonces de las ecuaciones (2.45) y (2.38) obtenemos que:

$$\begin{aligned}&\text{Pr}[\boldsymbol{Q} = Q_j]\text{Pr}[\text{Nodes in range}] = \\ &= \text{Pr}[Q_{j-1} \leq \widehat{\boldsymbol{P_{tx}}}, Q_j \geq \widehat{\boldsymbol{P_{tx}}}] = \\ &= \text{Pr}[Q_{j-1} \leq \widehat{\boldsymbol{P_{tx}}} \leq Q_j] = \\ &= \text{Pr}[Q_{j-1} \leq \boldsymbol{d}^{\alpha}\Omega \leq Q_j] = \\ &= \text{Pr}[\tfrac{Q_{j-1}}{\Omega} \leq \boldsymbol{d}^{\alpha} \leq \tfrac{Q_j}{\Omega}] = \\ &= \text{Pr}[(\tfrac{Q_{j-1}}{\Omega})^2 \leq (\boldsymbol{d}^{\alpha})^2 \leq (\tfrac{Q_j}{\Omega})^2] = \\ &= \text{Pr}[(\tfrac{Q_{j-1}}{\Omega})^{\frac{2}{\alpha}} \leq \boldsymbol{d}^2 \leq (\tfrac{Q_j}{\Omega})^{\frac{2}{\alpha}}] = \\ &= \exp{(-\frac{(\frac{Q_{j-1}}{\Omega})^{2/\alpha}}{4\sigma^2})} - \exp{(-\frac{(\frac{Q_j}{\Omega})^{2/\alpha}}{4\sigma^2})}\end{aligned} \tag{2.28}$$

para $j = 1, \ldots, p-1$. Para el cuanto de potencia con índice p hay una región entre $(\frac{Q_j}{\Omega})^{\frac{2}{\alpha}}$ y d_S^2 donde la recepción es todavía posible, pero con una probabilidad de error mayor. En este caso, la probabilidad asociada al último cuanto (potencia nominal) es:

$$\begin{aligned} &\Pr[\boldsymbol{Q} = Q_j]\Pr[\text{Nodes in range}] = \\ &= \Pr[(\tfrac{Q_{p-1}}{\Omega})^{\frac{2}{\alpha}} \leq \boldsymbol{d}^2 \leq d_S^2] = \\ &= \exp\left(-\frac{(\frac{Q_{p-1}}{\Omega})^{2/\alpha}}{4\sigma^2}\right) - \exp\left(-\frac{d_S^2}{4\sigma^2}\right) \end{aligned} \tag{2.29}$$

La probabilidad de que los nodos estén en rango es simplemente $\Pr[\text{Nodes in range}] = \Pr[\boldsymbol{d^2} \leq d_S^2]$. Por tanto, de la ecuación (2.26), el coeficiente medio s para una distribución normal es:

$$\begin{aligned} \overline{\boldsymbol{s}} &= \frac{\sum_{j=1}^{j=p} P_{tx_j}\Pr[\boldsymbol{Q} = Q_j]}{P_{tx_p}} = \\ &= \frac{\sum_{j=1}^{j=p-1} P_{tx_j}[\exp\left(-\frac{(\frac{Q_{j-1}}{\Omega})^{2/\alpha}}{4\sigma^2}\right) - \exp\left(-\frac{(\frac{Q_j}{\Omega})^{2/\alpha}}{4\sigma^2}\right)]}{P_{tx_p}[1 - \exp\left(-\frac{d_S^2}{4\sigma^2}\right)]} + \\ &+ \frac{\exp\left(-\frac{(\frac{Q_j}{\Omega})^{2/\alpha}}{4\sigma^2}\right) - \exp\left(-\frac{d_S^2}{4\sigma^2}\right)}{[1 - \exp\left(-\frac{d_S^2}{4\sigma^2}\right)]} \end{aligned} \tag{2.30}$$

Adicionalmente se ha verificado la proposición 2 utilizando cálculo numérico. La figura 2.6 muestra los resultados. Para valores de σ de 50, 100 y 150 se lanzaron/establecieron redes de n nodos y se calculó la media y la desviación típica de s utilizando 10000 muestras para cada punto. Las líneas en la figura 2.6 son la media $\pm$ la desviación típica obtenida en los experimentos, mientras que los puntos representan el cálculo analítico de $\overline{\boldsymbol{s}}$. Se puede observar de nuevo que la probabilidad se concentra alrededor del valor esperado a medida que n crece.

La figura 2.7 muestra la evaluación de esta expresión para diferentes valores del parámetro σ, y suponiendo Ω = -97.5 dB, y α = 3.95 (los mismos valores usados en los ejemplos previos). Como se podía esperar, se obtienen valores bajos de s (es decir, valores altos de L) cuando la σ es baja, ya que la distancia entre los nodos es menor. Nótese también que la función resultante tiene un intervalo de crecimiento rápido para $\sigma \in (10, 50)$. Para valores $\sigma > 100$ la función crece asintóticamente hacia un valor límite de $\overline{\boldsymbol{s}}$ = 0.78.

2.7. Evaluación de $\overline{L}$

Apoyándonos en los resultados previos, podemos enunciar la siguiente proposición:

Proposición 3 *Para valores grandes de n, la media de la variable aletoria L converge a*

$$\overline{\boldsymbol{L}}|_{n \gg 1} \approx \frac{1 + \overline{\boldsymbol{\xi}}}{\overline{\boldsymbol{s}} + \overline{\boldsymbol{\xi}}} \tag{2.31}$$

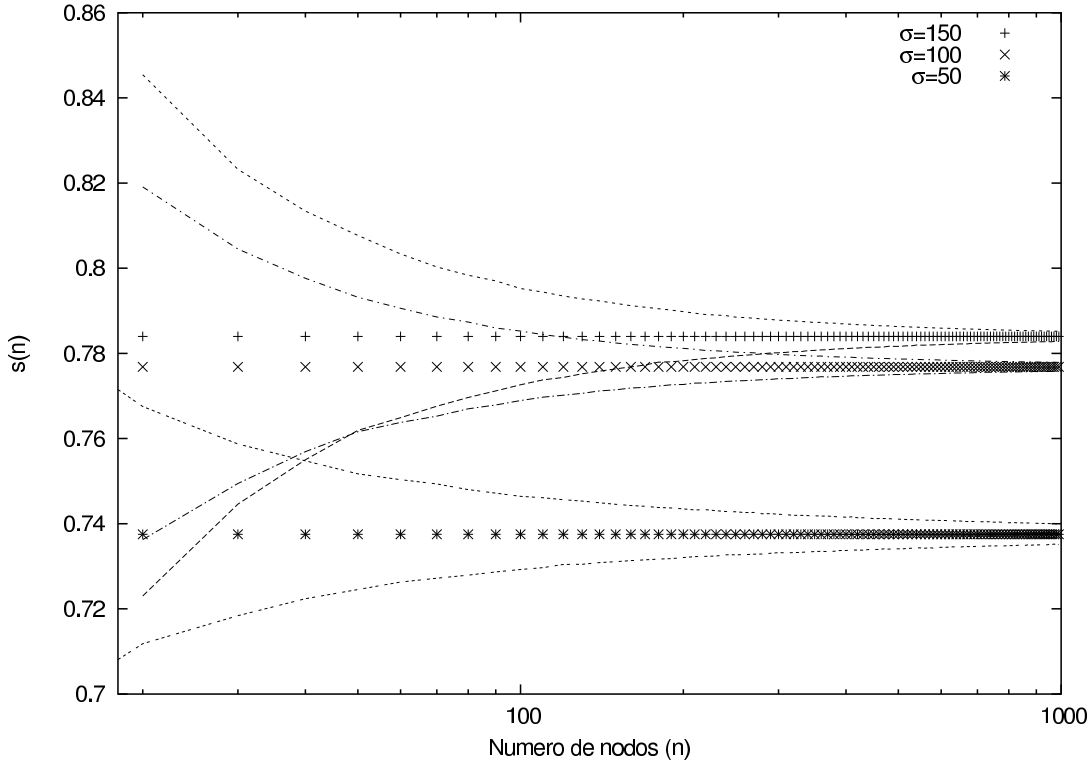

Figura 2.6: s evaluado respecto a n para diferentes σ

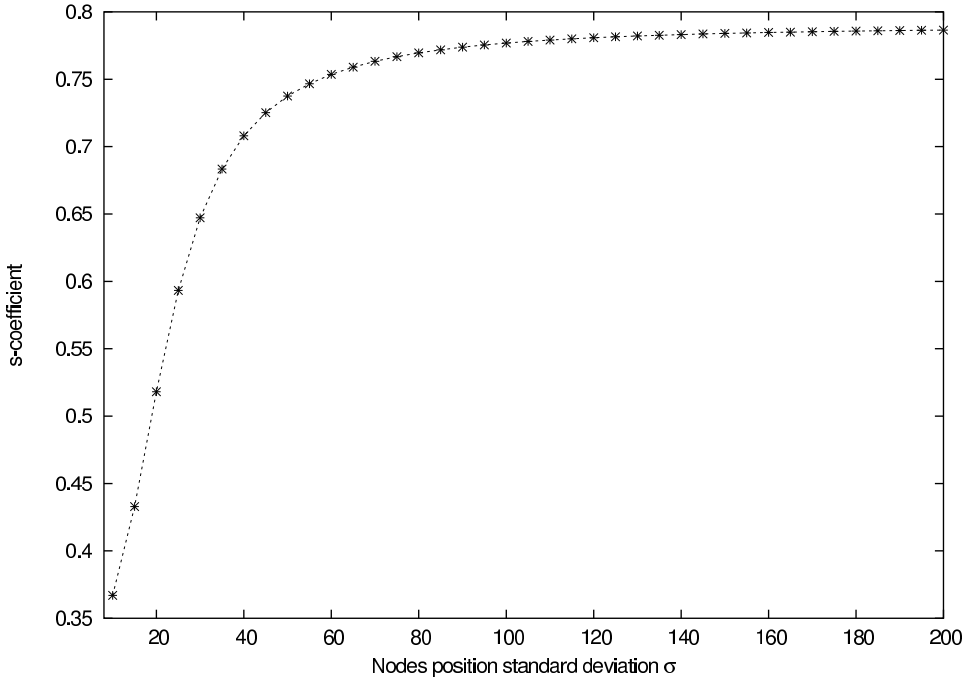

Figura 2.7: $\overline{s}$ evaluado respecto a σ

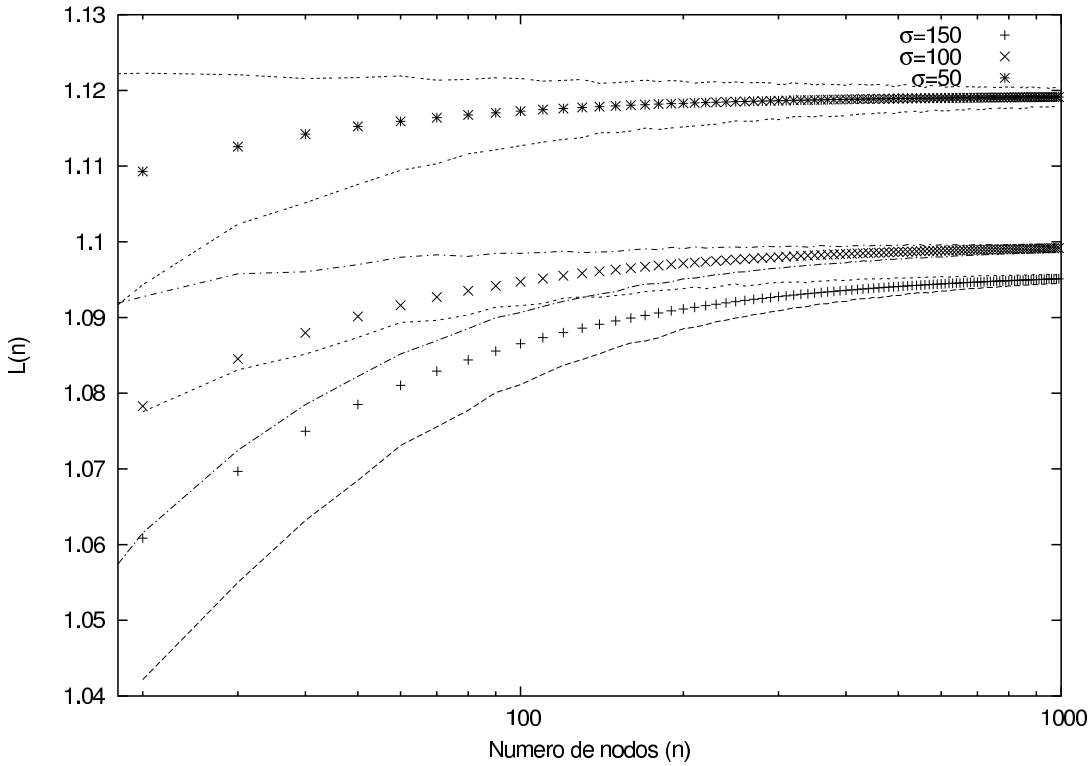

Figura 2.8: L evaluada respecto a n para diferentes σ

La razón de este comportamiento asintótico es la concentración de la función de probabilidad de la media de ξ y s a medida que n crece. Entonces, para valores grandes de n la incertidumbre en el valor de ξ y s es muy pequeña, y, por tanto, también lo es la de L. Estas razones nos permiten sustituir las variables aleatorias por sus valores esperados en la ecuación (2.31). Sin embargo, tampoco tenemos una demostración formal de la proposición 3. De nuevo, recurrimos al cálculo numérico para comprobar este enunciado. La figura 2.8 muestra la evaluación de L para valores crecientes de n. Los experimentos son similares a los mostrados en las figuras 2.4 y 2.6. Para cada punto se lanzaron 10000 redes aleatorias y se calculó el valor medio de L. Se obtuvo el valor de la media $\pm$ la desviación típica que se representa con líneas en la figura 2.8. Los resultados analíticos se muestran con puntos. La configuración usada fue similar a la de los ejemplos anteriores, B = 800 bits, B_p = B'_p = 100 bits, B_l = B'_l = 400 bits, B_a = 400 bits, $C = 20$, P_{rx} = 35.2 mW y P_{tx_p} = 76.2 mW.

Los resultados obtenidos pueden ser aplicados a cualquier protocolo MAC con operación ranurada simplemente ajustando los parámetros del modelo. El parámetro $\overline{s}$ depende del modelo de propagación considerado, del hardware empleado y de la distribución de los nodos (ver sección 2.4). El parámetro $\overline{\xi}$ depende fundamentalmente del modelo de tráfico y del protocolo de comunicaciones subyacente, para el que es necesario ajustar los parámetros de las ecuación (2.25), $B_p, B'_p, B_l, B'_l, B, B_a, C$, descritos en la sección 2.4.4.

En las siguientes secciones aplicaremos los resultados previos para evaluar las mejoras que introduciría el uso de TPC en dos protocolos MAC representativos en WSN: uno TDMA (LMAC) y otro de contienda (S-MAC). En este estudio mantendremos el modelo de propagación (*path-loss*), el hardware (nodos Mica2), y la distribución de los

nodos (normal bivariada) expuestos en la sección 2.4. Por tanto, nos concentraremos en el cálculo de $\overline{\xi}$.

2.7.1. Ejemplo TDMA: L-MAC

El protocolo *Lightweight Medium Access Protocol* (LMAC) [VanH04a] es un protocolo TDMA propuesto para redes de sensores, basado en una modificación del protocolo Eyes-MAC (E-MAC) [VanH04b]. Cada nodo posee una ranura que selecciona mediante la información sobre el estado de las ranuras disponibles (asignada o libre) que le transmiten sus vecinos inmediatos (a un salto de distancia). Las principales limitaciones de LMAC son que el número de ranuras es fijo (32) y que los nodos escuchan todas las ranuras aunque no estén ocupadas. En cuanto un nodo selecciona una ranura, transmite siempre o bien un paquete de control (preámbulo) o bien el preámbulo acompañado de un paquete de datos.

Nuestro modelo genérico descrito en la sección 2.4.4 puede ser directamente aplicado a la operación del protocolo. L-MAC usa un acceso TDMA puro, y notifica directamente a sus vecinos que se va a producir una transmisión en el preámbulo del paquete. Así, la fase de señalización/notificación incluida en nuestro modelo (ver Fig. 2.3) no es necesaria (por tanto será $B_l = 0$). Adicionalmente, el periodo de actividad es $C = 32$ y no hay asentimientos $B_a = 0$. Por tanto, una ranura en el protocolo L-MAC inclye sólo un preámbulo y la transmisión de datos (cuando corresponda).

El mecanismo de TPC que proponemos se puede utilizar aquí sin modificar la topología de la red, ya que los nodos siempre transmitirán el preámbulo a la potencia nominal. La sección de Datos siguiente se podría enviar con niveles de potencia controlados. Por tanto, para aplicar el análisis previo a LMAC, simplemente tenemos que ajustar los parámetros al funcionamiento de LMAC:

- n = 100 nodes
- Data size, B = 800 bits.
- Preamble, B_p = B'_p = 96 bits.
- Signaling, B_l = B'_l = 0 bits.
- Auxilary, B_a = 0 bits.
- Preamble period, C = 32.
- P_{rx} = 35.4 mW.
- P_{tx_p} = 76.2 mW.

En la figura 2.9 se muestra la evaluación de L, según la ecuación (2.31), particularizada a LMAC respecto a σ para diferentes cargas de los enlaces (ρ). Los resultados muestran ahorros energéticos considerables, en el orden del 10-20 % para valores medios y altos de σ. El ahorro es mayor para valores bajos de σ y altos de la carga ρ. Esta tendencia es razonable ya que los nodos estan cerca unos de otros y transmiten un número alto de paquetes, con lo que el TPC es efectivo. LMAC además es un protocolo con poca sobrecarga de paquetes de control, lo que refuerza la efectividad del TPC.

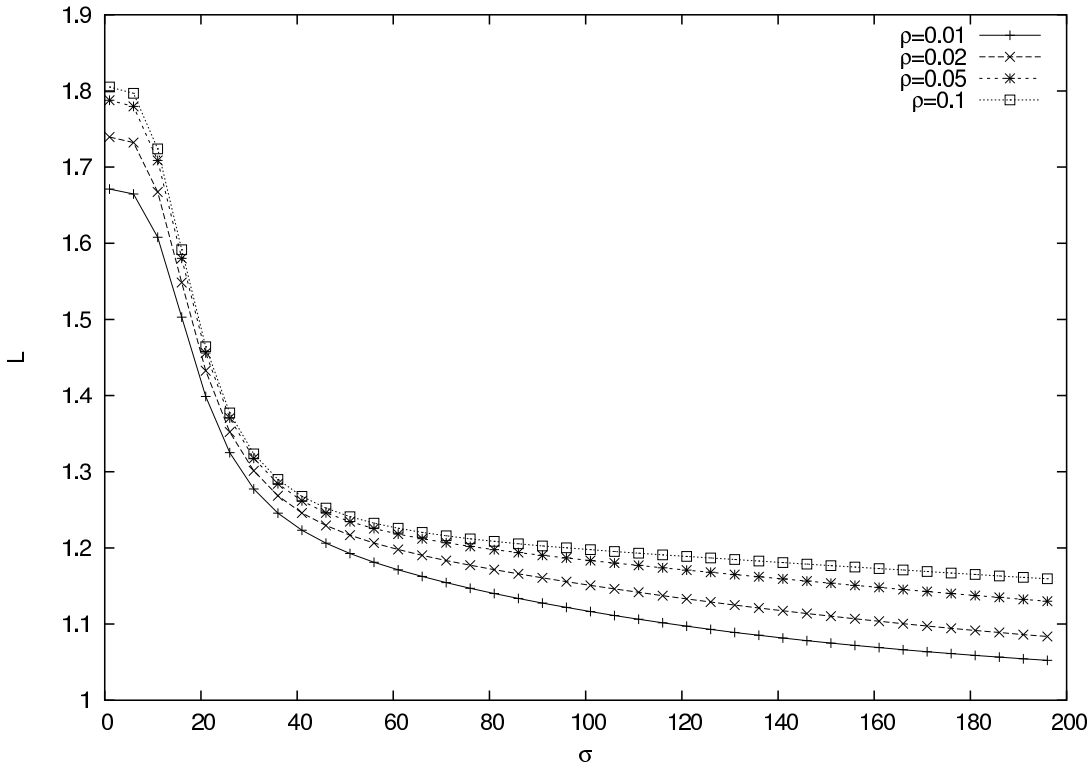

Figura 2.9: L respecto a σ para diferentes cargas de los enlaces (ρ) para el protocolo LMAC

2.7.2. Ejemplo de contienda: S-MAC

El protocolo S-MAC [Ye04] es un protocolo basado en contienda que utiliza periodos de escucha y sueño sincronizados. La sincronización de estos periodos se consigue gracias a un paquete corto de sincronización (paquete SYNC), que es periódicamente difundido por las estaciones. Así, la operación del protocolo es ranurada, y similar al modelo representado en la figura 2.3. Las colisiones se evitan mediante un mecanismo tipo CSMA/CA, es decir, con la secuencia RTS/CTS/Datos/ACK. Los paquetes RTS/CTS son transmitidos en la fase de escucha/notificación indicada en la figura 2.3. En la etapa de intercambio de mensajes se transmiten el paquete de Datos y el ACK. El análisis previo puede ser aplicado a S-MAC de la misma forma en la que se aplicó a LMAC, es decir, ajustando los parámetros de las ecuaciones (2.16), (2.17) y (2.18). Como valores para evaluar el protocolo hemos escogido los que se utilizan en la implementación de referencia de S-MAC bajo TinyOS [SMIMPL], que son los siguientes:

- n = 100 nodes
- Data size, B = 800 bits.
- Preamble, B_p = 727 bits, B'_p = 100 bits.
- Signaling, B_l = 1226 bits, B'_l = 100 bits.
- Auxilary, B_a = 100 bits.

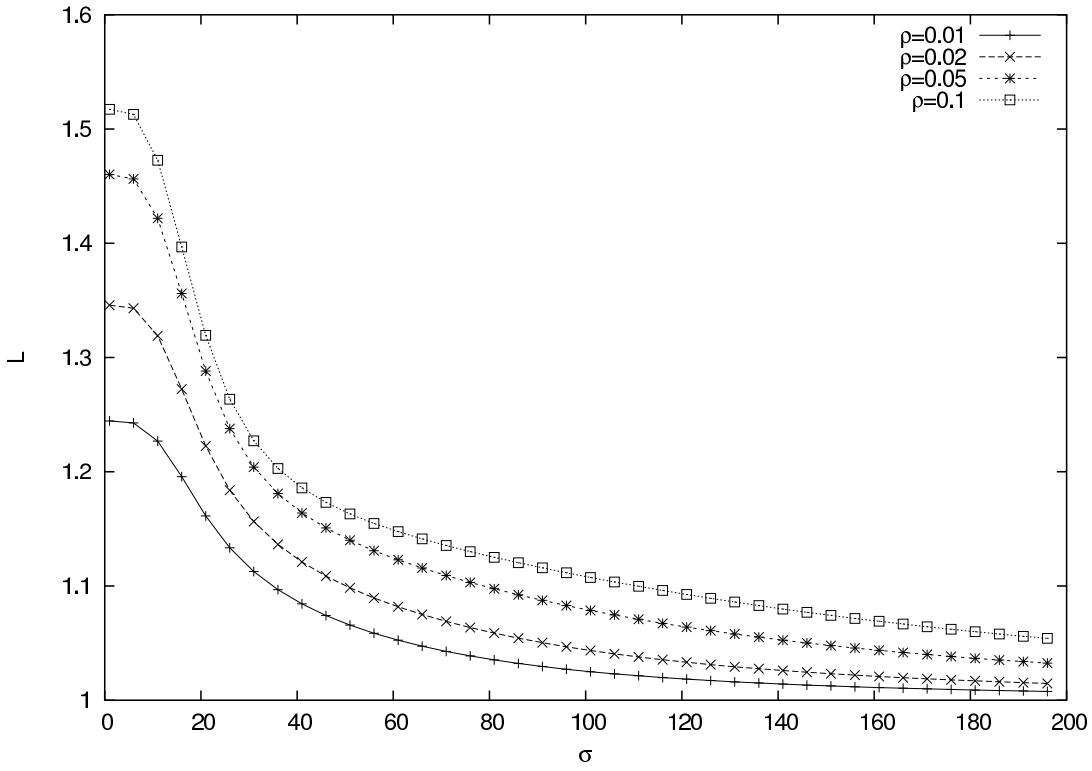

Figura 2.10: L respecto a σ para diferentes cargas de los enlaces (ρ) para el protocolo S-MAC

- Preamble period, C = 20.
- P_{rx} = 35.4 mW.
- P_{tx_p} = 76.2 mW.

La figura 2.10 muestra L respecto a σ para diferentes cargas de los enlaces (ρ) al igual que se hizo para LMAC. Los resultados, por el contrario, muestran un peor comportamiento del TPC que en el caso de LMAC. El ahorro medio está por debajo del 10 % para valores medios y altos de σ. La razones de este empeoramiento son dos. Por una parte, con S-MAC se transmiten más paquetes de control (RTS y CTS) que con LMAC. Puesto que estos paquetes se envían siempre a la potencia nominal, se malgasta más energía. Por otra parte, en LMAC los nodos van a dormir justo después del preámbulo (si no son los receptores del paquete de datos). En S-MAC, por el contrario, todos los nodos tienen que esperar hasta que el periodo de escucha termina para dormir. Es decir, el periodo de escucha del canal de los protocolos basados en contienda reduce los beneficios del uso de TPC.

2.7.3. Discusión sobre los resultados teóricos

En la introducción expresamos nuestro interés en calcular el mayor ahorro energético que podría proporcionarnos un mecanismo TPC ideal. Para ello, realizamos dos asunciones fundamentales en este trabajo: (1) todos los nodos conocen de antemano la potencia

necesaria para alcanzar a sus vecinos, y (2) no existen colisiones. Por tanto, los resultados obtenidos deben interpretarse consecuentemente: muestran una estimación del ahorro que un mecanismo de TPC podría ofrecer. Claramente, las condiciones del escenario de despliegue real determinarán el ahorro final, condicionadas a la posible aparición de colisiones y donde los nodos deberán implementar mecanismos que les permitan decidir la potencia de transmisión. En cualquier caso, el método analítico presentado permite estimar rápidamente el desempeño del mecanismo de TPC para decidir si vale la pena o no su utilización en la capa MAC.

2.8. Pruebas experimentales del mecanismo de TPC

Tras el estudio teórico presentado en las secciones anteriores hemos desarrollado una serie de pruebas experimentales del mecanismo de control de potencia. El objetivo es demostrar la viabilidad práctica del mismo, sobre diferentes protocolos utilizados en redes de sensores. Como plataforma de desarrollo se han empleado sensores tanto de la familia Mica2 (chip de radiocomunicación CC1000 a 868/916 MHz), como motes del tipo MicaZ (chip de radiocomunicación CC2400). Ambos integrados permiten la selección de niveles de transmisión durante su funcionamiento.

Las pruebas se han realizado para protocolos con mecanismo de acceso tipo CSMA/CA. Este mecanismo permite implementar de modo sencillo el control de potencia:

1. El nodo receptor, mide tras recibir el paquete de control RTS, la relación señal a ruido del mensaje. Para ello, se muestrea periódicamente el nivel de señal (mediante la interfaz de control del chip de radiocomunicaciones), y se mide su diferencia entre los instantes de recepción del paquete y los instantes inmediatamente anteriores (donde el canal está desocupado).

2. El nodo receptor calcula el decremento de señal que es posible realizar en transmisión para que el mensaje llegue con una relación señal a ruido mínima (en los experimentos se ha elegido como valor umbral 20 dB), y responde con el paquete CTS, en el que indica en un nuevo campo el valor de este nivel.

3. El transmisor, corrige la potencia al nivel indicado en el paquete CTS, y transmite los datos.

En los experimentos, a fin de poder comparar correctamente la mejora energética, se ha procedido del siguiente modo:

- Se utilizan dos nodos: uno con el mecanismo corrector activo y el otro operando en el modo normal, es decir, transmitiendo los datos a la potencia nominal.

- Los nodos se transmiten datos de longitud fija $800bits$ con un protocolo de "ping-pong". Es decir, tras recibir el paquete de datos, el receptor (A) lo envía de nuevo al nodo original (B), y éste de nuevo al nodo (A), y así sucesivamente. De este modo ambos nodos transmiten el mismo paquete en idénticas condiciones, pero con mecanismos diferentes. Así es posible medir el comportamiento relativo entre

ambas variantes. El nivel de carga del protocolo se modula introduciendo un retardo entre la recepción y la retransmisión de los mensajes. En las pruebas este tiempo es de 2 segundos.

Concretamente, se ha procedido a la implementación de este mecanismo usando como protocolo de soporte el S-MAC, en su implementación de referencia para los motes de la familia Mica2, sobre sistema operativo TinyOS. Los resultados muestran niveles de ahorro significativo, incluso en espacios cerrados, y dependientes de la carga, como así lo indican los resultados teóricos. En el ambiente controlado del laboratorio se alcanzan ahorros entorno al 25 %, superiores al caso del despliegue aleatorio modelado en el análisis teórico de este capítulo.

2.9. Conclusiones del capítulo

En este capítulo se ha desarrollado un método analítico para calcular el ahorro proporcionado por un mecanismo genérico de control de potencia de transmisión en el nivel MAC, que puede ser utilizado en combinación con la mayoría de las propuestas actuales para el MAC en redes de sensores. Se calcula la relación entre la energía consumida si no se aplica TPC respecto a la que se consume si se aplica TPC. Esta relación se denomina tasa L. El ahorro energético medio, medido a través de la tasa L, converge si el número de nodos es muy alto (la asunción habitual para redes de sensores). La tasa L debe ser interpretada como el límite superior del ahorro de energía obtenible a través del mecanismo de control de potencia, y su cálculo puede realizarse rápidamente ajustando los parámetros de operación de la red en las ecuaciones. El cálculo de L se ha particularizado para el caso en el que los nodos se posicionan según una distribución normal, con un modelo genérico de redes de sensores y un patrón de tráfico razonable en estas redes.

La principal conclusión extraída del trabajo expuesto en este capítulo respecto a los protocolos MAC apropiados para ser usados con el mecanismo de TPC, es que merece la pena utilizarlo con protocolos tipo TDMA, en los que se pueden esperar ahorros del 10-20 %. Los protocolos basados en contienda no consiguen todos los beneficios del TPC ya que permanecen más tiempo a la escucha y suelen utilizar más paquetes de control, reduciendo la eficiencia a un 5-10 %.

Apéndice A. Modelo del canal de radio

Sean P_{tx} y P_{rx} la potencia de transmisión y de recepción de la señal, respectivamente, y d la distancia entre dos nodos. EL canal de transmisión puede ser considerado un canal de banda estrecha invariante en el tiempo, que puede ser modelado aproximadamente por un modelo *path-loss* [Rap02], donde:

$$P_{rx}(d) = P_{tx}(\frac{\lambda}{4\pi d_0})^2(\frac{d_0}{d})^\alpha \tag{2.32}$$

donde α es el coeficiente de *path loss* calculado a una distancia de referencia d_0.

Hay dos contribuciones principales al ruido en la transmisión: el ruido del canal, que se supone habitualmente ruido blanco gaussiano con densidad espectral de potencia $N_0 = KT$, con K la constante de Boltzmann y T la temperatura absoluta; y el ruido interno del receptor, caracterizado por la figura de ruido ($F \approx$ 10-15 dB). Sea E_b la energía por bit, R la tasa binaria y B el ancho de banda de transmisión. La relación señal a ruido (*Signal-to-Noise Ratio*, SNR) es:

$$\frac{S}{N} = \frac{P_{rx}}{KTBF} = \frac{E_b}{N_0}\frac{R}{B} \tag{2.33}$$

Los sensores Mica2 utilizan la modulación NCFSK, con lo que la probabilidad de error de bit (b_e) se expresa [Pro01]:

$$b_e = \frac{1}{2}\exp(-\frac{1}{2}\frac{E_b}{N_0}) \tag{2.34}$$

Consecuentemente, la probabilidad de error de paquete (p_e) de tamaño n bits es:

$$p_e = 1 - (1 - b_e)^n \tag{2.35}$$

Entonces, dada una probabilidad de error objetivo, la potencia de transmisión necesaria se puede calcular de las ecuaciones anteriores:

$$\frac{\widehat{E_b}}{N_0} = -2\ln(2\widehat{b_e}) = -2\ln(2[1 - (1 - \widehat{p_e})^{\frac{1}{n}}]) \tag{2.36}$$

Y de la ecuación (2.33),

$$\begin{aligned} &\frac{\widehat{P_{rx}}}{KTBF} = -2\ln(2[1 - (1 - \widehat{p_e})^{\frac{1}{n}}])\frac{R}{B} \Rightarrow \\ &\widehat{P_{rx}} = -2KTFR\ln(2[1 - (1 - \widehat{p_e})^{\frac{1}{n}}]) \end{aligned} \tag{2.37}$$

Entonces de la expresión (2.32), $\widehat{P_{tx}}$ se puede escribir:

$$\widehat{P_{tx}} = d^\alpha\Omega \tag{2.38}$$

dónde,

$$\Omega = -2KTRF\ln(2[1 - (1 - \widehat{p_e})^{\frac{1}{n}}])(\frac{4\pi d_0}{\lambda})^2(\frac{1}{d_0})^\alpha \tag{2.39}$$

Apéndice B. Función de densidad de probabilidad de la distancia cuadrática entre puntos elegidos independientemente con una distribución normal bivariada

Teorema 1 *La fdp de $\boldsymbol{d^2}$, distancia cuadrática entre puntos elegidos independientemente con una distribución normal bivariada, es:*

$$f_{d^2}(x) = \frac{1}{4\sigma^2}\exp(-\frac{x}{4\sigma^2}) \tag{2.40}$$

Demostración: Primero, nótese que:

$$\begin{aligned} \boldsymbol{\Delta X} = \boldsymbol{X_i} - \boldsymbol{X_j} &= N(0,\sigma) - N(0,\sigma) = \\ &= N(0,\sqrt{2}\sigma) \\ \boldsymbol{\Delta Y} = \boldsymbol{Y_i} - \boldsymbol{Y_j} &= N(0,\sigma) - N(0,\sigma) = \\ &= N(0,\sqrt{2}\sigma) \end{aligned} \tag{2.41}$$

Por tanto,

$$(\frac{\boldsymbol{\Delta X}}{\sqrt{2}\sigma})^2 + (\frac{\boldsymbol{\Delta Y}}{\sqrt{2}\sigma})^2 = \chi_2^2 \tag{2.42}$$

Entonces,

$$\boldsymbol{d^2} = \boldsymbol{\Delta X}^2 + \boldsymbol{\Delta Y}^2 = 2\sigma^2\chi_2^2 \tag{2.43}$$

Ya que $\boldsymbol{\chi_2^2} = \mathbf{exp}(\frac{\mathbf{1}}{\mathbf{2}})$, la función de distribución de $\boldsymbol{d^2}$ es:

$$\begin{aligned} \Pr[d^2 \leq x] = \Pr[2\sigma^2\chi_2^2 \leq x] = \Pr[\chi_2^2 \leq \tfrac{x}{2\sigma^2}] = \\ = \int_0^{\frac{x}{2\sigma^2}} \frac{1}{2}\exp\,(-\frac{u}{2})du = \\ = 1 - \exp\,(-\frac{x}{4\sigma^2}) \end{aligned} \tag{2.44}$$

Y, finalmente, el fdp de $\boldsymbol{d^2}$ es la derivada de la anterior:

$$f_{d^2}(x) = \frac{d(1-\exp\,(-\frac{x}{4\sigma^2}))}{dx} = \frac{1}{4\sigma^2}\exp(-\frac{x}{4\sigma^2}) \tag{2.45}$$

Por tanto, queda demostrado.