

ÍNDICE GENERAL

INTRODUCCIÓN	1
CAPÍTULO 1 INTRODUCCIÓN A LAS REDES INALÁMBRICAS DE SENSORES	3
1.1 ¿Qué son redes inalámbricas de sensores?	3
1.1.1 Características	3
1.1.2 Componentes	5
1.2 Arquitectura de una red inalámbrica de sensores	6
1.2.1. Capa Física	7
1.2.2. Capa de Enlace	7
1.2.3. Capa de Red	8
1.2.4. Capa de transporte	8
1.2.5. Capa de aplicación	8
1.3 Actividades de las RIS	8
1.3.1 Planificación de la red	9
1.3.2 Establecimiento o implantación	9
1.3.3 Administración	9
1.3.4 Monitoreo	9
1.3.5 Comunicación	10
1.4 Estándar 802.15.4 (Zigbee)	10
1.4.1 Objetivos	10
1.4.2 Características	11
1.5 Restricciones de las RIS	11
1.5.1 Batería	12
1.5.2 Alcance de la señal de transmisión	12
1.5.3 Condiciones del medio inalámbrico	12
CAPÍTULO 2 ENTORNO DE TRABAJO.....	14
2.1 Kit MicaZ de Crossbow	14
2.1.1 Nodos MPR2600 con chip de radio CC2420	15
2.1.2 Estación base MIB520	16
2.2 Ambiente de Desarrollo	17
2.3 TinyOS	18
2.4 NesC	19
CAPÍTULO 3 TÉCNICAS DE CHANNEL SURFING.....	20
3.1 Channel Surfing	20

3.2 Tipos de Nodos	21
3.2.1 Nodo aislado	21
3.2.2 Gateway o Nodo fronterizo	22
3.3 Técnicas de Channel Surfing	22
3.3.1 Coordinated Channel Switching	22
3.3.2 Spectral Multiplexing	23
3.4 Selección de una técnica de Channel Surfing	28
CAPÍTULO 4 EL PROTOCOLO ASMCS	29
4.1 Descripción	29
4.2 Características	30
4.3 Funcionamiento	31
4.3.1 Estructuras de Datos	32
4.3.2 Configuración y parámetros por omisión	33
4.3.3 Mensajes	35
4.3.4 Fases	37
4.3.5 Interfaces Send y Receive	41
4.3.6 Incluir ASMCS en una Aplicación	42
CAPÍTULO 5 EXPERIMENTOS	43
5.1 Mediciones	43
5.2 Resultados	44
5.5.1 Confiabilidad de la comunicación	45
5.5.2 Latencia en la transmisión de mensajes	45
5.5.3 Consumo de energía	46
CONCLUSIONES Y RECOMENDACIONES	47
REFERENCIAS	49
Apéndice A. CÓDIGO FUENTE DEL PROTOCOLO ASMCS	51

ÍNDICE DE FIGURAS

Figura 1.1 Red Inalámbrica de Sensores conectada a Internet.	5
Figura 1.2 Pila de Protocolos y planos de gestión de una RIS (Akyildiz <i>et al.</i> , 2002)	7
Figura 2.1 Kit MicaZ de Crossbow Technology Inc.	15
Figura 2.2 Nodo Crossbow MPR2600	15
Figura 2.3. Estación base Crossbow MIB520 dentro y fuera de la carcasa	17
Figura 2.4. MoteView mostrando los datos obtenidos por los sensores de la red	18
Figura 3.1. Alineamiento de canales entre 802.11 (canales no solapados) y 802.15.4	21
Figura 3.2. Ejecución de la técnica Coordinated Channel Switching	23
Figura 3.3. Escenario de interferencia y slots en Spectral Multiplexing Síncrona	25
Figura 3.4. Comunicación en cada slot con la técnica Spectral Multiplexing Síncrona	25
Figura 3.5. Escenario de interferencia y slots en Spectral Multiplexing Asíncrona	26
Figura 3.6. Comunicación en cada slot con la técnica Spectral Multiplexing Asíncrona	27
Figura 4.1. Ubicación de protocolo ASMCS en la pila de protocolos de una RIS	29
Figura 4.2. Formato de una tabla de vecinos con n nodos	32
Figura 4.3. Formato de la estructura que guarda todos los buffers de los vecinos	33
Figura 4.4. Formato del mensaje de hello (HelloMsg)	35
Figura 4.5. Formato del mensaje de anuncio (AdvertisementMsg)	36
Figura 4.6. Formato del mensaje de datos (DataMsg)	37
Figura 5.1 Ejemplo de topología estrella con 3 motes	44
Figura 5.2 Ejemplo de topología multihop con 3 motes	44

ÍNDICE DE TABLAS

Tabla 2.1. Especificación técnica de los nodos Crossbow MPR2600

16

INTRODUCCIÓN

Los constantes avances tecnológicos han conllevado al desarrollo de pequeños dispositivos llamados nodos sensores o motes los cuales son capaces de medir, procesar y enviar, de manera inalámbrica, una serie de datos que permiten conocer el estado actual del medio donde se encuentran. Una característica importante de estos aparatos es su autonomía en cuanto a la energía que utilizan, ya que cada uno es provisto de una fuente de alimentación propia (baterías) que le permite, siempre que se haga una adecuada utilización de la misma, estar activo por un largo período de tiempo en el cual no se requiere ningún tipo de intervención humana. Actualmente, debido a su bajo coste, los motes han permitido acceder a lugares que debido a sus características físicas son imposibles de alcanzar para los humanos.

Una red inalámbrica de sensores (RIS) es una interconexión de motes que permite el flujo de información desde un punto a otro. En la mayoría de los casos, las RIS poseen uno o varios nodos sumideros (sink) que se encargan de recolectar la información, realizar algún procesamiento adicional y enviarla hacia otras redes, servidores o internet. De esta manera se tiene el control sobre variables específicas de un ambiente que se desee estudiar y se está en la capacidad de tomar acciones y actuar a distancia sobre una situación determinada.

Sin embargo, las numerosas tecnologías inalámbricas que existen hoy en día y que comparten la misma banda de frecuencia (2,4 GHz a 2,48 GHz) interfieren entre ellas y causan una caída importante en el desempeño de las aplicaciones. Dichas aplicaciones pueden requerir una alta confiabilidad a la hora de enviar sus datos debido a que pueden estar implementadas en ambientes militares, de salud, urbanos, ambientales, etc.

En el presente trabajo se estudia el problema de la interferencia en las redes inalámbricas específicamente en el caso de las RIS que operan bajo el estándar IEEE 802.15.4 y se propone una solución basada en técnicas de *Channel Surfing* que permite evitar este tipo de inconveniente. Esta medida, la cual recibe el nombre de protocolo ASMCS (*Asynchronous Spectral Multiplexing Channel Surfing*), consiste en emplear el mecanismo Espectral Multiplexing Asíncrono para llevar a cabo la comunicación a través de aquellos canales de frecuencia que no se encuentren afectados por algún tipo de interferencia en un momento dado.

Con el objetivo de evaluar el desempeño del protocolo desarrollado, se han realizado distintas mediciones que permiten comparar ciertos datos como la tasa de paquetes enviados, el consumo de energía y el tiempo que tarda un paquete en llegar desde un nodo de la red a

otro. Estas pruebas se han desarrollado bajo la utilización de ASMCS y en la ausencia del mismo para poder comparar el desempeño de la red con la introducción de esta técnica propuesta.

Para explicar los aspectos estudiados y el desarrollo de ASMCS, esta memoria se divide en 5 capítulos. En el capítulo 1 se presenta un resumen de lo que son las RIS y se explican sus elementos fundamentales, así como también se explica el estándar IEEE 802.15.4 y se mencionan las restricciones más importantes que afectan a las redes inalámbricas de sensores. En el capítulo 3 se explica el concepto de Channel Surfing, las diferentes técnicas existentes para implementar una solución de este tipo y se muestra un análisis que debe ser realizado para escoger el mecanismo más adecuado para un proyecto. Posteriormente, en la sección número 5, se presentan los detalles del protocolo desarrollado en el presente proyecto (ASMCS), sus características, funcionamiento, mensajes y estructuras de datos utilizadas en la implementación de la mencionada técnica de Channel Surfing. Finalmente, en el capítulo 6 se muestran los resultados de las mediciones realizadas del desempeño de ASMCS en cuanto a la confiabilidad de la comunicación, latencia en el envío de mensajes y consumo de energía.

Capítulo 1 INTRODUCCIÓN A LAS REDES INALÁMBRICAS DE SENSORES

1.1 ¿Qué son redes inalámbricas de sensores?

Una red inalámbrica de sensores (por sus siglas, RIS) es un sistema distribuido conformado por un número de dispositivos sensores cuya función principal es reportar información sobre ciertas condiciones del lugar donde se encuentra desplegada. La comunicación entre estos sensores es llevada a cabo de manera inalámbrica hasta que los datos son llevados a un lugar donde se centraliza y procesa dicha información para ser intercambiada con otras redes o simplemente para ser analizada en el mismo lugar.

Los datos que pueden recolectar los motes varían de acuerdo a las necesidades del lugar de estudio y de la configuración de la aplicación que poseen instalada. Comúnmente se recaudan cantidades de luz, temperaturas, aceleraciones y grados de humedad aunque actualmente la gran mayoría dispone de puertos de expansión que permiten adaptar sensores adecuados para obtener los valores de la variable física que se desee monitorear, independientemente de la naturaleza de la misma.

Según los requerimientos del estudio a realizar, los nodos sensores pueden ser desplegados de distintas maneras que van desde la colocación intencional en un lugar específico de la región en cuestión hasta ser lanzados desde un avión sobre el área de interés. He aquí la necesidad que se tiene en muchos casos de poseer una red auto configurable que sea capaz de permanecer operativa por largos períodos de tiempo (meses o incluso años) sin ningún tipo de mantenimiento por parte de una persona.

1.1.1 Características

Aún cuando actualmente hay muchas variaciones de lo que son las Redes Inalámbricas de Sensores (RIS), existe un conjunto de características que podemos encontrar en casi cualquiera de ellas, las cuales son:

- Fundamentalmente son diseñadas para la detección o recolección de datos de un fenómeno específico que se desea estudiar.
- Una RIS puede estar compuesta por un número cualquiera de nodos que puede ir desde unos pocos hasta miles de ellos.
- En la mayoría de los casos, el flujo de información va desde los nodos sensores hasta una estación base (nodo sink) que se encarga de recolectar los datos para su posterior análisis.
- Los intercambios de información son originados por consultas o por eventos que interesan ser reportados.
- La cantidad de energía de la que dispone cada nodo es limitada y, en algunos casos, imposible de reemplazar por lo cual esto constituye un factor de gran importancia a tomar en cuenta durante su diseño e implantación.
- La topología que constituye la red es mayormente estática aunque hoy en día hay casos en los que los nodos cambian de lugar una vez que ya se ha iniciado el proceso de recolección de datos.
- El costo de cada nodo es tan bajo que, en algunos casos, son desechados al momento de culminar el estudio o la monitorización que se desea realizar.
- La red debe ser capaz de tolerar fallos parciales o permanentes de los nodos así como también debe permitir la inclusión de algunos nodos que pueden restituir la comunicación en secciones críticas de la misma.
- Se emplea principalmente las comunicaciones de difusión (broadcast) en lugar de las comunicaciones punto a punto (unicast).
- Generalmente los nodos no poseen un identificador único universal, tal como lo es un número IP. Una de las propuestas para direcciones en las redes de sensores es la utilización de coordenadas espaciales para nombrar datos, porque los datos de los sensores están asociados con el contexto físico donde el fenómeno detectado ocurre (Bulusu et al., 2001).
- La seguridad, tanto física como a nivel de la comunicación, es más limitada que en los enfoques de redes inalámbricas convencionales debido a los escasos recursos de cálculo de los nodos. Sin embargo es un tema a considerar cuando se estudian fuentes de datos que pueden ser confidenciales.

1.1.2 Componentes

Las RIS están compuestas por nodos que se pueden distinguir de acuerdo a las funcionalidades que realiza en la red, como se puede observar en la Figura 1.1. A continuación se explican los más comunes.

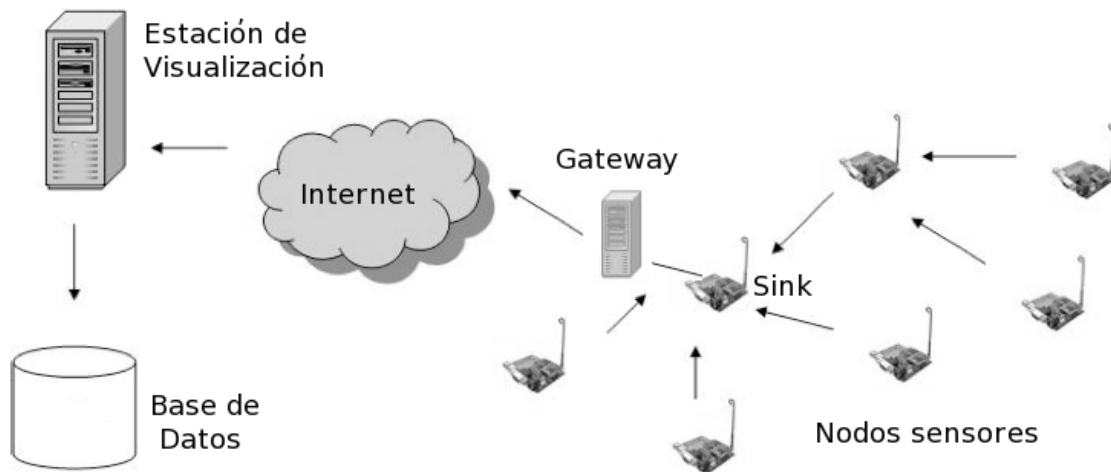


Figura 1.1 Red Inalámbrica de Sensores conectada a Internet.

1.1.2.1. Nodos Sensores o Motes

Estos dispositivos poseen funcionalidades muy básicas en cuanto a sus componentes e interfaces y generalmente son operados con baterías. Además, pueden enviar alarmas si las condiciones que están monitorizando cambian sustancialmente en función de los parámetros para los que estaban programados (Bruin, 2005).

Estos dispositivos fueron desarrollados originalmente por Intel Research en colaboración con la Universidad de California en Berkeley en el Centro para la Investigación en Tecnologías de la Información CITRIS (Information Technology Research in the Interest of Society) (Intel, 2004a).

Un mote está compuesto generalmente por una unidad de microcontrolador o unidad de procesamiento sencilla, una pequeña memoria, sensores, una fuente de alimentación eléctrica y un dispositivo de comunicación inalámbrico, que le permite al mote comunicarse e intercambiar datos unos a otros (Intel, 2004b) (Intel, 2005b).

Cada tarea que realiza un mote en un ciclo de operación consume su batería y acercan al dispositivo a su muerte. Para evitar esto, el mote es activado sólo durante un porcentaje pequeño de tiempo. Se activa para realizar lecturas programadas o para la recepción o transmisión de datos de sus dispositivos vecinos y luego pasa a un estado de ahorro de energía, en el que consume muy poca potencia, al tener la mayoría de sus componentes electrónicos apagados.

1.1.2.2. Estación Base o Nodo Sink

En la mayoría de los casos la estación base está representada por un mote, igual a los que conforman el resto de la red, que se encuentra conectado a otro dispositivo que permite el intercambio de información entre la RIS y otras redes. Sin embargo, existen componentes especiales que poseen distintas prestaciones con respecto a los nodos ordinarios de la red, como por ejemplo una antena más grande o un procesador más potente que permite gestionar un volumen de datos superior al de un nodo sensor.

Una característica muy importante de los nodos sink es que a menudo están conectados permanentemente a una fuente de energía que evita su desconexión de la red y, por lo tanto, el programa que aquí se ejecuta puede despreocuparse de este asunto.

1.1.2.3. Gateway

Aunque en muchos casos se trata del mismo nodo Sink conectado a un ordenador, también se pueden encontrar dispositivos que ejecutan funciones adicionales. Algunos pueden ser capaces de transformar los datos para que la información recolectada sea entendida por otro protocolo o almacenarla en una base de datos para ser mostrada de manera más ordenada.

1.2 Arquitectura de una red inalámbrica de sensores

En una RIS la pila de protocolos está diseñada en base a cinco capas que son: capa física, capa de enlace, capa de red, capa de transporte y capa de aplicación. Este diseño está realizado con el objetivo de cuidar tanto la energía como el estado de las rutas, integrar los datos con los distintos protocolos de red, establecer una comunicación eficiente en cuanto a la energía empleada y promover esfuerzos cooperativos entre todos los nodos de la red (Akyldiz *et al.*, 2002).

Por otra parte también se incluyen tres planos de gestión los cuales son: plano de energía, plano de movilidad y plano de manejo de tareas. Estos planos tienen como función la óptima utilización de la energía, el mantenimiento de rutas hacia el nodo sink y las funciones que deben cumplir los nodos dependiendo de su estado, ubicación, etc.

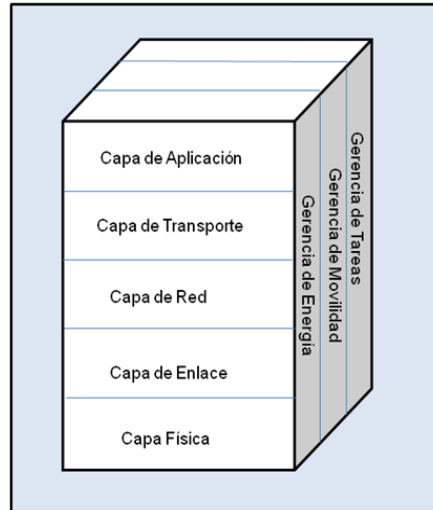


Figura 1.2 Pila de Protocolos y planos de gestión de una RIS (Akyildiz *et al.*, 2002)

La reciente investigación en el área de Redes Inalámbricas de Sensores provoca que, a día de hoy, no se posea una arquitectura completamente demarcada en cada nivel. Por el contrario, lo que se consigue muchas veces es un diseño denominado en inglés *cross-layered* donde se rompe un poco lo que son las estrictas funcionalidades de cada nivel. A continuación se explican detalladamente cada una de las capas que forman parte de esta pila de protocolos:

1.2.1. Capa Física

Esta capa es la responsable de la selección de la frecuencia, detección de la señal, modulación y generación de la señal de transmisión. Al mismo tiempo se encarga de activar o desactivar la radio y de obtener los valores indicativos de la calidad de la señal recibida a través de las medidas de la calidad del enlace (en inglés *Link Quality Indicator*, LQI) y la fuerza de esta señal (en inglés *Received Signal Strength Indication*, RSSI).

1.2.2. Capa de Enlace

Tiene como función principal detectar la trama de los mensajes, controlar el acceso al medio, controlar los errores en la transmisión y multiplexar los datos. El control de acceso al

medio es el encargado de dos importantes funcionalidades: creación de la estructura de la red estableciendo enlaces de comunicaciones para la transmisión de datos y compartir el medio y los recursos de comunicación con el resto de los nodos de la red para lograr un efectivo intercambio de datos. Esta última es realizada mediante el algoritmo de Acceso Múltiple por Detección de Portadora (en inglés *Carrier Sense Multiple Access*, CSMA).

1.2.3. Capa de Red

Se encarga de proveer interconexión con redes externas donde el nodo sink puede realizar tareas de gateway. De igual manera puede comprender protocolos de enrutamiento que establezcan y administren las posibles rutas entre las cuales deben circular los datos. Actualmente se necesita, a este nivel, mejoras en los protocolos para poder afrontar efectivamente grandes cambios en la topología (Akyildiz *et al.*, 2002).

1.2.4. Capa de transporte

Actualmente es la capa menos estudiada actualmente en las RIS, pero cumple una importante función cuando se desea acceder a los nodos o al gateway desde Internet o algún otro tipo de red externa. Debe ser implementada siempre que las aplicaciones que se ejecutan en los nodos necesiten controlar el flujo de datos que circula entre ellas.

1.2.5. Capa de aplicación

Aún cuando esta capa se encuentra un poco inexplorada, existen varias propuestas interesantes sobre protocolos de aplicación como por ejemplo: Sensor Management Protocol (SMP), Task Assignment, Data Advertisement Protocol (TADAP) y Sensor Query and Data Dissemination Protocol los cuales permiten el manejo de datos a este nivel así como también sincronización de los relojes y movimientos de sensores.

1.3 Actividades de las RIS

Las principales actividades que lleva a cabo una Red Inalámbrica de Sensores están comprendidas en cinco grandes grupos: planificación de la red, establecimiento o implantación, administración, monitoreo y comunicación (Ruiz, 2002). A continuación se explica con detalle cada grupo de actividades.

1.3.1 Planificación de la red

Comprende todo el estudio previo que debe realizarse sobre la RIS a implantar. Durante esta fase se debe hacer un análisis de los requerimientos funcionales y no funcionales, variables a monitorizar y realizar una cuidadosa selección de los dispositivos y protocolos a emplear según las necesidades que debe cubrir la red.

Además, la planificación comprende la elaboración de un *site survey*, el cual es un estudio detallado del lugar donde se instalará la Red Inalámbrica de Sensores que permite determinar qué factores de la red se deben configurar para garantizar su buen funcionamiento (ubicación de los nodos, interferencias a evitar, zonas de ruido que pueden afectar a la RIS, etc.). El *site survey* debe ser realizado periódicamente tiempo debido a que las condiciones físicas del lugar de estudio pueden variar en el tiempo.

1.3.2 Establecimiento o implantación

Es la fase en la cual se ejecuta la instalación de los nodos y la configuración de la red. Los nodos pueden ser colocados intencionalmente en los lugares determinados por el estudio de *site survey* o pueden ser desplegados al azar. En cualquier caso, antes de comenzar con sus funciones de monitoreo, los nodos pueden realizar algunas pruebas para determinar su ubicación o formar grupos (González et al., 2006). Estas actividades son desarrolladas, en la mayoría de los casos actuales, de manera automática por los mismos nodos los cuales tienen la capacidad de auto organizarse y responder efectivamente frente diversos fallos en la red.

1.3.3 Administración

Una vez que la RIS se encuentra operativa, se deben tratar de utilizar todos los recursos disponibles de la manera más eficiente posible. Uno de los objetivos fundamentales de una buena administración es alargar tanto como sea necesario el tiempo de vida de la red, ya sea ejecutando medidas que eviten el rápido agotamiento de la energía de la que disponen los nodos o corrigiendo aquellos contratiempos que puedan surgir durante el desempeño de las funciones de la red.

1.3.4 Monitoreo

El monitoreo es la actividad principal para la que fue diseñada e implementada la red. Incluye las actividades de sensado, a través de las cuales se obtienen los datos necesarios del entorno estudiado, y el procesamiento de dicha información para que posteriormente sea conducida hasta la(s) estación(es) base.

1.3.5 Comunicación

Consiste en el envío de los datos recolectados por cada nodo hasta la estación base para su análisis o traslado hacia otra red. Esta actividad se realiza mediante el intercambio de mensajes entre los motes el cual puede variar según los protocolos utilizados. Además, de acuerdo a la topología de red que se haya formado dichas transmisiones pueden incluir varios saltos hasta alcanzar su destino final.

Las tecnologías empleadas en este proceso poseen ciertas limitaciones en cuanto alcance y resistencia contra agentes que puedan interferir en el proceso lo cual provoca que los protocolos involucrados en esta tarea deban tratar de solventar para lograr una buena comunicación.

1.4 Estándar 802.15.4 (Zigbee)

Es un estándar que surge bajo la necesidad de poder utilizar dispositivos de bajo coste y poco consumo de energía en ambientes industriales, de salud y agrícolas donde las tecnologías tradicionales no están capacitadas para su adecuada aplicación. Esta filosofía bajo la cual nace Zigbee es conocida como *Low Power Wireless Personal Area Network*, por sus siglas en inglés LP-WPAN (González et al., S/F).

1.4.1 Objetivos

Los objetivos fundamentales que busca alcanzar 802.15.4 son (González et al., S/F):

- **Muy bajo consumo de potencia:** una de las principales limitaciones de una red Zigbee es el consumo eléctrico ya que únicamente están equipados con una pequeña batería que debe durar meses o incluso años.
- **Muy bajo coste de implementación:** según el objetivo al que se vaya a dedicar la red se necesitará numerosos dispositivos que en algunos casos serán desechados una vez terminado el estudio en cuestión. Por esta razón el costo final de los componentes que implementan dicha red debe ser muy pequeño.

Las bandas en las que opera el estándar 802.15.4 corresponden a frecuencias libres ISM y son:

- 868 MHz con un canal de 20 Kbps.
- 902-928 MHz la cual permite 10 canales de 40 Kbps.

- 2.4-2.48 GHz con 16 canales de 250 Kbps.

Por causa de las prestaciones que ofrece la banda de frecuencia de 2.4 GHz, gran cantidad de motes fabricados comercialmente la utilizan de la misma manera que lo hacen dispositivos Wi-Fi (802.11 b/g), Bluetooth, 802.15.3, algunos modelos de teléfonos inalámbricos y ciertos modelos de hornos microondas. Esto plantea cuestionamientos interesantes sobre las estrategias a utilizar, por los distintos estándares, para evitar que se interfieran unos a otros.

1.4.2 Características

Algunas de las características de las Redes inalámbricas LR-WPAN 802.15.4 son (IEEE Std 802.15.4, 2003):

- Tasas de transferencia de 250 kb/s, 40 kb/s, y 20 kb/s.
- Operación en topologías de estrella o punto a punto.
- Capacidad para manejar direcciones cortas de 16 bit (short) o direcciones extendidas de 64 bits.
- Capacidad para manejar la reserva garantizada de slots de tiempo GTS (en inglés, *Guaranteed Time Slots*).
- Posibilidad de acceder a los canales de radio mediante protocolos (CSMA-CA) (*Carrier Sense Multiple Access with Collision Avoidance*).
- Capacidad de administrar protocolos con reconocimiento de mensajes para manejar la confiabilidad de los envíos de información.
- Bajo consumo de potencia.
- Detección de energía (en inglés *Energy detection*, ED).
- Indicadores de la calidad de la señal (LQI).
- Manejo de 16 canales en la banda de frecuencia de 2450 MHz, 10 canales en la banda de frecuencia de 915 MHz y 1 canal en la banda de frecuencia de 868 MHz.

1.5 Restricciones de las RIS

La capacidad del estándar Zigbee de poder transmitir a tasas de velocidad de hasta 250 Kbps, poseer muy bajo coste de producción, ofrecer la posibilidad de trabajar casi en cualquier ambiente, entre otros, provoca que ciertos factores tengan que ser "perjudicados" durante su diseño e implementación. A causa de esto encontramos ciertas restricciones que implican un cuidadoso análisis de la red a desarrollar para no dejar de considerar factores

claves. Las más resaltadas a lo largo del desarrollo de las Redes Inalámbricas de Sensores se detallan a continuación:

1.5.1 Batería

Como ya se ha mencionado anteriormente, el consumo de energía es uno de los factores críticos que más atención comprende al momento del diseño de una RIS. Debido a que la energía eléctrica es provista por una fuente finita como lo son las baterías y el tiempo de vida de la red, en algunos casos, debe alcanzar años, se debe asegurar que los nodos estarán disponibles por un tiempo determinado.

Para lograr superar este inconveniente se deben emplear técnicas que desactiven la mayor cantidad de funciones posibles de los nodos y que sincronicen el trabajo que realizan con el fin de optimizar el tiempo en el cual los nodos se hayan completamente operativos.

1.5.2 Alcance de la señal de transmisión

Una RIS soporta tasas de transmisión de hasta 250 Kbps (sin contar encabezados) en distancias que van desde los 10 hasta los 300 metros, sin embargo se deben emplear antenas externas si se quiere lograr un alcance más allá de los 100 metros. La razón por la cual el máximo alcance de la señal es 300 metros se debe a que si se quisiera llegar a una distancia más lejana, el consumo de energía sería demasiado alto como para ser aceptado dentro de este esquema de bajo consumo (González et al., S/F).

Por esta razón se debe intentar limitar la potencia de la señal para no malgastar la tan preciada energía eléctrica.

1.5.3 Condiciones del medio inalámbrico

Dado que las RIS que operan bajo el estándar IEEE 802.15.4, estas redes trabajan en un rango de frecuencia libre, en el cual cualquier otra tecnología tiene el derecho de transmitir, la posibilidad de encontrar algún elemento que pueda ocasionar interferencia en la comunicación entre dos o más nodos es bastante alta.

Los elementos comunes que pueden interferir críticamente a una Red Inalámbrica de Sensores, hasta el punto de bloquear su comunicación, son una red WI-FI y un horno microondas ubicado a menos de un metro de distancia de algún nodo de la RIS (Chiasserini et al., 2003). Por otra parte, la evolución de la tecnología Bluetooth (IEEE 802.15.1) en los

últimos años, especialmente a partir de la versión 1.2 prácticamente ha eliminado los problemas de interferencia entre dichas tecnologías inalámbricas y las redes 802.15.4 (González et al., S/F).

De esta problemática nacen una serie de mecanismos y técnicas que intentan solventar esta situación a través de distintas maneras. Algunas de estas medidas se basan en mecanismos colaborativos de coexistencia, en los cuales ambas tecnologías intercambian información intencionalmente para sincronizarse y organizar sus transmisiones, o en mecanismos no colaborativos de coexistencia, en los que cada red adquiere la información sobre la interferencia presente en el medio a través de la detección de la misma ya sea por medio de transmisiones erróneas o sensando del medio (Chiasserini et al., 2003).

Una de estas estrategias no colaborativas es el Channel Surfing el cual se define como la habilidad de adaptar la frecuencia en la que trabaja un nodo como respuesta a la necesidad de evitar las interferencias presentes en el medio (Wenyuan et al., 2007).

En este Proyecto Final de Carrera se analizan las distintas técnicas de Channel Surfing, se realiza la elección de la más adecuada para el caso de estudio, se implementa un protocolo para su utilización en RIS y se presentan los resultados obtenidos en cuanto a las mejoras introducidas con respecto a la confiabilidad y seguridad de la comunicación.

Capítulo 2 ENTORNO DE TRABAJO

En esta sección se describen los dispositivos utilizados durante el proceso de desarrollo del protocolo ASMCS así como también el conjunto de software que sirvió de apoyo durante el transcurso de esta actividad. Todo esto comprende desde los motes programados y con los cuales se realizaron los experimentos pertinentes hasta el sistema operativo que utilizan (TinyOS) y el lenguaje en el que se escriben los programas que allí se ejecutan (NesC). Por último se ofrece una explicación sobre la aplicación MoteView la cual permite mostrar, a través de una interfaz gráfica en el ordenador, los datos que se van recibiendo en el nodo sink o estación base.

2.1 Kit MicaZ de Crossbow

Para el desarrollo del presente Proyecto Final de Carrera se utilizó un kit MicaZ, que se puede observar en la Figura 2.1, comercializado por la marca Crossbow Technology Inc. el cual incluye los siguiente elementos:

- 6 nodos MPR2600.
- 2 Estaciones Base MIB520.
- Guía de usuario.
- CD con el software necesario para el desarrollo de este tipo de aplicaciones.



Figura 2.1 Kit MicaZ de Crossbow Technology Inc.

2.1.1 Nodos MPR2600 con chip de radio CC2420

El modelo de mote utilizado en el desarrollo de este proyecto es el Crossbow MPR2600 (ver Figura 2.2) el cual cuenta con un microprocesador Atmel Atmega 128L de 8 bits con una frecuencia de 8 MHz. Para la transmisión de datos posee un chip de radio marca Chipcon modelo CC2420, que transmite en la banda de frecuencia de 2.4 GHz cumpliendo con el estándar IEEE 802.15.4. Este chip permite la transmisión de datos a 250 Kbps. En cuanto a la memoria del dispositivo, ostenta 4 KB de SRAM, 128 KB destinados a los programas que ejecuta.



Figura 2.2 Nodo Crossbow MPR2600

Con respecto al suministro de electricidad cuenta con 2 pilas de tipo AA las cuales le brindan la posibilidad de estar activo durante 5 días consumiendo un alto nivel de energía y hasta 20 años en modo inactivo.

Los datos técnicos de estos dispositivos se presentan más detalladamente en la Tabla 2.1.

Tabla 2.1. Especificación técnica de los nodos Crossbow MPR2600

Processor/Radio Module	MPR2600CA	Remarks
Processor Performance		
Program Flash Memory	128K bytes	
Measurement Serial Flash	512K bytes	>100,000 measurements
Configuration EEPROM	4 K bytes	
Serial Communications	UART	0 V to 3 V transmission levels
Analog to Digital Converter	10 bit ADC	8 channels, 0 V to 3 V input
Other Interfaces	Digital I/O,I2C,SPI	
Current Draw	8 mA	Active mode
	< 15uA	Sleep mode
RF Transceiver		
Frequency band ¹	2400 MHz to 2483.5 MHz	ISM band, programmable in 1 MHz steps
Transmit (TX) data rate	250 kbps	
RF power	3 dBm (min), 0 dBm (typ)	
Receive Sensitivity	-90 dBm (min), -94 dBm (typ)	
Adjacent channel rejection	45 dB	+ 5 MHz channel spacing
	30 dB	- 5 MHz channel spacing
Current Draw	19.7 mA	Receive mode
	11 mA	TX, -10 dBm
	14 mA	TX, -5 dBm
	17.4 mA	TX, 0 dBm
	20 μ A	Idle mode, voltage regular on
	1 μ A	Sleep mode, voltage regulator off
Electromechanical		
External Power	2.1 V - 3.6 V	
Size (in)	0.95 x 0.95	LCC68
(mm)	24.13 x 24.13	
Weight (oz)	0.11	
(gms)	3	

2.1.2 Estación base MIB520

La estación base MIB520 (ver Figura 2.3) Es un dispositivo que comprende las mismas prestaciones que un nodo MPR2600 pero que dispone de un puerto USB el cual permite conectarlo a un ordenador. De esta manera se pueden instalar las aplicaciones programadas en los distintos motes que conforman la red. Aunque la placa que lo conforma presenta algunas diferencias, las capacidades de procesamiento y memoria son exactamente iguales a las del modelo ya descrito anteriormente. La diferencia más importante se encuentra en que este aparato recibe la energía eléctrica del ordenador y por ello no debe realizar consideraciones sobre su gasto.



Figura 2.3. Estación base Crossbow MIB520 dentro y fuera de la carcasa

Otra característica importante de la estación base MIB520 es que permite realizar una conexión entre la Red Inalámbrica de Sensores y el ordenador para intercambiar datos y poder realizar un análisis más amplio y preciso sobre el estudio en cuestión.

2.2 Ambiente de Desarrollo

El kit MicaZ de Crossbow incluye también un conjunto de software que facilita el desarrollo de aplicaciones para Redes Inalámbricas de Sensores que trabajan bajo el sistema operativo TinyOS. Además también contiene ciertos programas que colaboran en la recolección y muestra de datos para su interpretación por parte del usuario.

El contenido del CD se resume a continuación:

- MoteConfig: herramienta gráfica que permite programar los motes.
- CygWin: emulador del sistema operativo Linux que incluye las funcionalidades necesarias para la compilación e instalación de los programas a ejecutar en los nodos.
- Programmers Notepad: pequeño IDE que permite programar cómodamente en varios lenguajes.
- MoteView: interfaz que permite mostrar y ordenar los datos recolectados por la estación base (ver Figura 2.4).

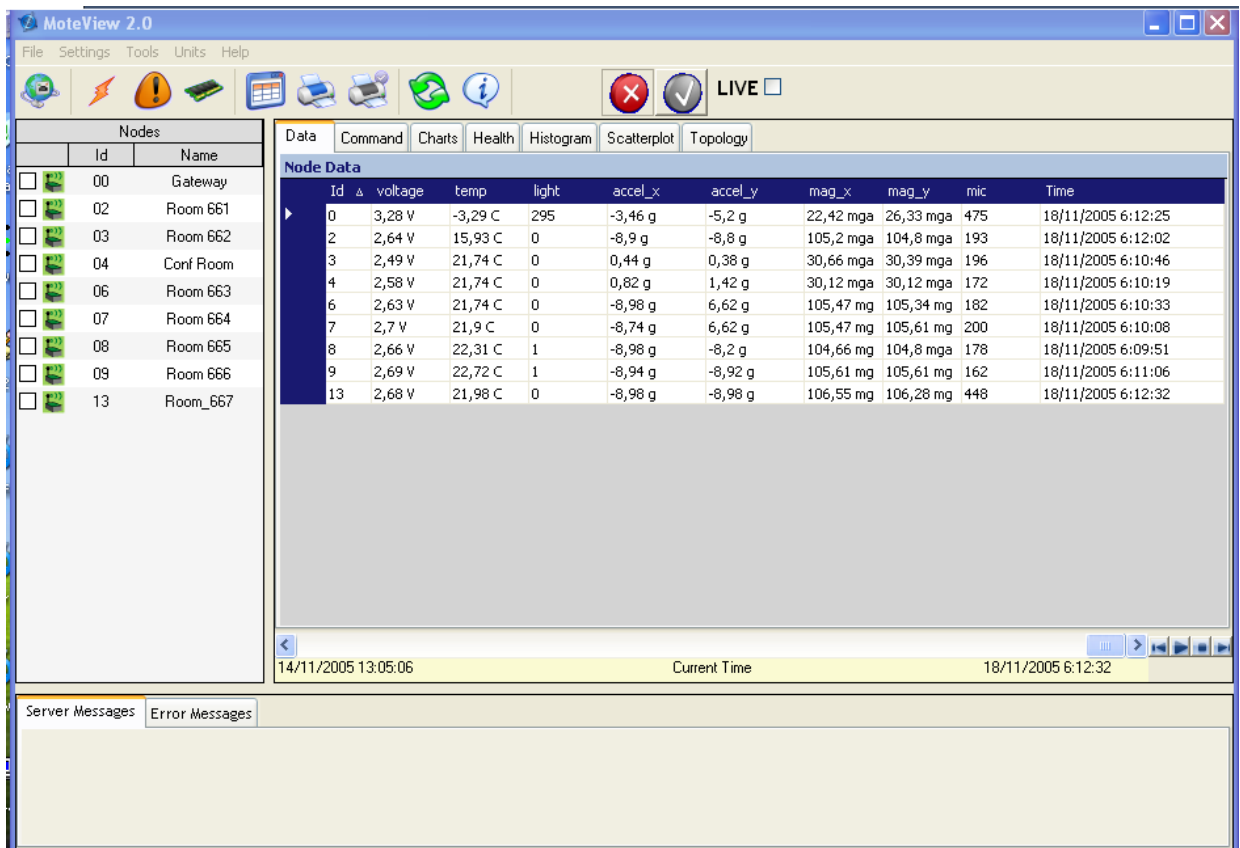


Figura 2.4. MoteView mostrando los datos obtenidos por los sensores de la red

- gcc: compilador del lenguaje C.
- TortoiseSVN: herramienta gráfica para la gestión de repositorios de control de versión.
- WinMerge: software que permite la comparación de código fuente.
- Aplicaciones de ejemplo y módulos TinyOS para ser cargados en los motes junto con la aplicación desarrollada.

2.3 TinyOS

TinyOS es un sistema operativo libre y orientado a eventos diseñado para la programación de sensores embebidos. Originalmente fue desarrollado por investigadores adscritos al Departamento de Ingeniería Eléctrica y Ciencias de la Computación en la Universidad de California en Berkeley (TinyOS et al., 2004).

Está escrito en un nuevo lenguaje llamado NesC, del cual se habla a continuación, y se encuentra estructurado en distintos módulos que gestionan diferentes componentes del nodo. El chip de radio, los sensores para tomar mediciones de las variables externas o internas del

mote, control de acceso al medio, timers, etc. Cuando la aplicación es compilada se genera un archivo ejecutable que contiene todos los módulos necesarios para su desempeño en el nodo el cual no es más que el enlace entre los componentes que el usuario ha decidido incluir y los suyos propios.

En la actualidad TinyOS se encuentra en la versión 2.1.0 que contempla una serie de mejoras respecto a la versión 2.0.2 en cuanto al manejo de memoria e hilos, soporte para más plataformas y la incorporación de nuevos protocolos de disseminación de información. (TinyOS et al., 2004).

2.4 NesC

Es un lenguaje orientado a eventos que extiende a C pero con esta funcionalidad que lo hace muy útil para la programación de Redes inalámbricas de Sensores. Fue desarrollado en la UC Berkeley y desde su comienzo estuvo diseñado para dispositivos embebidos.

Un programa en NesC es el enlace entre varios componentes por parte del usuario programador. Un componente representa un programa por sí mismo que es capaz de ejecutar una función específica. Se vale de otros componentes para complementar y agilizar sus funcionalidades. Se divide en tres partes:

1. Configuración: declaración de las interfaces involucradas en el componente. También es llamado wiring.
2. Interfaz: declaración de funciones llamadas comandos que sirven para interactuar con otros componentes.
3. Módulo: código que implementa las funciones del componente así como también los comandos que se ofrecen en la interfaz.

Capítulo 3 TÉCNICAS DE CHANNEL SURFING

3.1 Channel Surfing

La constante producción de dispositivos inalámbricos que transmiten en frecuencias de radio libres ha provocado problemas de coexistencia en bandas como en la ISM de 2.4 GHz. A día de hoy, en casi cualquier lugar podemos encontrar varios aparatos electrónicos que utilizan este medio ya sea para temas de comunicación o para otras funciones (Routers Wi-Fi, teléfonos inalámbricos, hornos microondas, dispositivos Bluetooth, entre otros). Esto nos conduce a pensar que intencionalmente o no, los servicios ofrecidos por las Redes Inalámbricas de Sensores pronto serán altamente perturbados o, en el peor de los casos, bloqueados.

Además, existe una continua intención de utilizar las Redes Inalámbricas de Sensores en monitoreo de ambientes críticos como lo son:

- La salud de pacientes en un hospital.
- Procesos industriales.
- Operaciones militares.
- Colaboración en desastres naturales.
- Monitoreo sísmico de estructuras.

En vista de estos problemas han surgido diversas medidas que pueden ayudar a las RIS en este esfuerzo por alcanzar una comunicación confiable. Channel Surfing es un mecanismo que permite evitar fallos en la comunicación provocados por interferencias presentes en una Red Inalámbrica de Sensores. Su funcionamiento consiste en evadir las interferencias presentes mediante la variación de la frecuencia de radio utilizada en la transmisión de datos (Wenyuan et al., 2007).

Al utilizar esta estrategia, aquellos nodos que se auto detecten como aislados (incapaces de comunicarse con alguno de sus vecinos) se cambiarán inmediatamente de canal para esperar oportunidades de reconexión con el resto de la red. Por otra parte, los nodos que

todavía mantengan comunicación con otros vecinos y detecten que son incapaces de obtener alguna señal de un nodo en particular, se cambiarán temporalmente de canal en busca de este nodo aislado para reincorporarlo a la red.

En la Figura 3.1 se pueden observar cada uno de los canales del estándar IEEE 802.15.4 y los canales del estándar IEEE 802.11 que no están solapados.

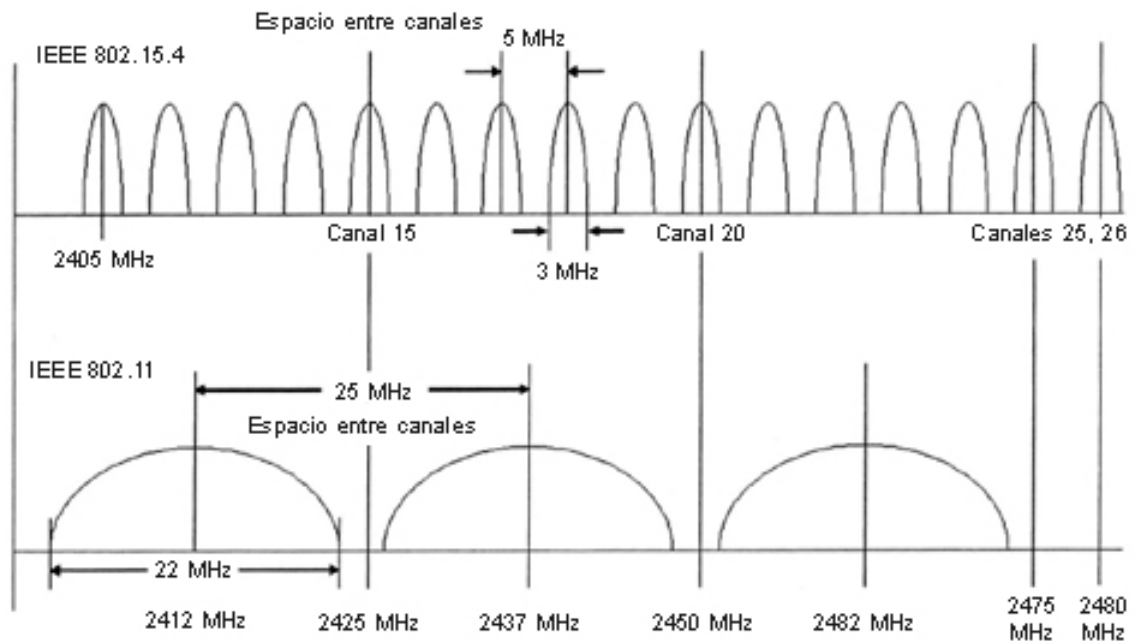


Figura 3.1. Alineamiento de canales entre 802.11 (canales no solapados) y 802.15.4

3.2 Tipos de Nodos

Al utilizar la estrategia de Channel Surfing para evitar fallos en la comunicación a causa de interferencias, los nodos de la RIS cumplirán distintos roles. Algunos seguirán actuando como nodos normales, pero otros tendrán que cumplir funciones de nodos gateway enlazando las dos partes de la red que se encuentran incomunicadas.

3.2.1 Nodo aislado

Un nodo aislado es cualquier nodo de la red que ha quedado incomunicado por causa de una interferencia en el canal en el que se encuentra. Dicho mote no tiene capacidad de enviar ni recibir mensajes con algún vecino.

3.2.2 Gateway o Nodo fronterizo

Es llamado Gateway o nodo fronterizo aquel mote que es incapaz de comunicarse con alguno o varios de sus vecinos pero que todavía guarda comunicación con nodos que se encuentran en el canal actual. Tiene como función buscar, en los canales de frecuencia siguientes, al nodo aislado y moverse entre aquellos canales donde se encuentre algún vecino para mantener la comunicación en la red. Para ello debe configurar el tiempo que pasa en cada canal según la cantidad de nodos con los que puede comunicarse en cada uno y el tiempo que le toma, aproximadamente, enviar y recibir los datos pertinentes.

3.3 Técnicas de Channel Surfing

Existen varias maneras de aplicar la estrategia del Channel Surfing, en (Wenyuan et al., 2007) se formulan dos soluciones: en primer lugar Coordinated Channel Switching en la cual los nodos gateway participan en llevar a toda la red hasta un canal nuevo donde se reconstruirá nuevamente. La segunda técnica, Spectral Multiplexing, consiste en la multiplexación, por parte de los nodos gateway, entre el viejo y nuevo canal conectando a los nodos que operan entre ambas frecuencias.

3.3.1 Coordinated Channel Switching

Consiste en evadir la interferencia cambiando toda la red de canal. Esta simple estrategia incluye una fase de transición en la cual los nodos pasan a la nueva banda de radio, seguidamente la conexión es restituida.

La ejecución de esta técnica pasa por varias fases, las cuales se explican con la ayuda de la Figura 3.2. En primer lugar se presenta la interferencia, por parte del objeto *X*, afectando a los nodos *D*, *I*, *J* y *O* (Figura 3.2 - a) los cuales, al percatarse de su situación, se cambian directamente al canal 2. Por otra parte los nodos fronterizos *C*, *E*, *H*, *K*, *N*, *P*, *S* y *T* notan que no pueden comunicarse con sus vecinos en el canal actual y se deciden a probar en el siguiente donde encontrarán a los nodos aislados (Figura 3.2 - b). En este momento los nodos gateway volverán al canal inicial temporalmente para difundir un mensaje que indique el canal al cual debe cambiarse toda la red (Figura 3.2 - c). Luego de reenviar este mensaje a través de todos los nodos, la red completa pasará a comunicarse en el canal 2 de manera permanente (Figura 3.2 - d).

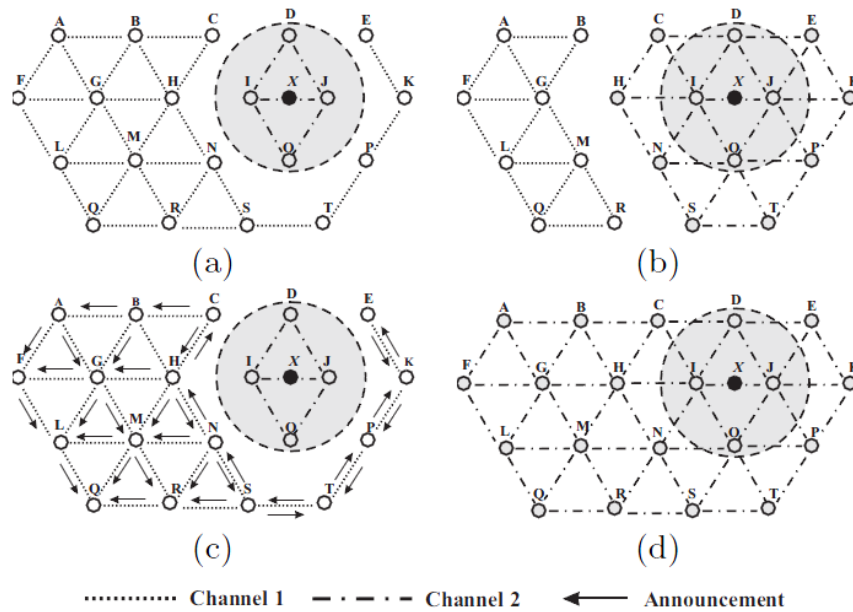


Figura 3.2. Ejecución de la técnica Coordinated Channel Switching

Una ventaja de esta técnica, a parte de su simplicidad, es que aunque el mensaje de difusión no llegue a todos los nodos de la red, aquellos que no lo reciban se considerarán aislados y pasarán al nuevo canal por sí mismos.

Por otra parte, representa un costo notable el migrar a toda la red de su canal de radio por causa de una interferencia que afecta a un pequeño porcentaje de nodos de la misma. Sin embargo, cuando la región bloqueada es lo suficientemente extensa, Coordinated Channel Switching constituye la estrategia más adecuada.

3.3.2 Spectral Multiplexing

Se basa en aplicar una estrategia similar a Coordinated Channel Switching pero sólo en la región de la red que se encuentre afectada por la interferencia en cuestión. El resto de la red se mantiene trabajando en el canal original mientras que los nodos de frontera (multiplexing nodes) se encargan de transmitir, en ambos canales, la información necesaria para que la red mantenga su conectividad (Wenyuan et al., 2007).

El tiempo que un nodo gateway pasa en cada canal es el factor más crítico que se debe manejar en este tipo de técnica con el fin de evitar o disminuir al máximo el número de paquetes no entregados o no recibidos por causa de una descoordinación en los canales de radio utilizados para un momento dado. Idealmente los nodos gateway deberían tener dos

interfaces de radio, pero a día de hoy sabemos que esto es muy difícil de lograr debido a que el transmitir y recibir mensajes son los factores que más batería consumen. De cualquier manera, el tiempo que debe pasar un nodo gateway en un canal en el cual tiene vecinos con los que comunicarse viene dado por el número de dichos nodos, los datos que debe enviar y recibir y el tiempo de overhead que se emplea en esta labor. De esta forma se define un ciclo de trabajo como el proceso realizado por un nodo gateway una vez pasa por todos aquellos canales dónde debe realizar las actividades de envío, recepción y, de ser el caso, búsqueda de vecinos aislados.

Con el fin de lograr una correcta sincronización entre los canales de radio utilizados surgen dos subtécnicas que ofrecen alternativas distintas para esta necesidad y que, según sea el caso, poseen sus respectivas ventajas y factores en contra

3.3.2.1 Spectral Multiplexing Síncrona

Se caracteriza por el hecho de que toda la red es gobernada por un reloj general que mantiene sincronizados a todos los nodos involucrados en el proceso. En este caso el ciclo de trabajo de todos los nodos es dividido en tantos slots como canales habilitados tenga la red para operar. Cada slot es asignado a un canal y durante ese tiempo sólo tienen permitido enviar o recibir mensajes aquellos nodos que se encuentren en dicha banda de frecuencia (Wenyuan et al., 2007).

El proceso de comunicación durante el uso de esta práctica viene explicado por la Figura 3.3 (a) en la cual se presenta la siguiente situación: La interferencia sobre el nodo *D* lo aísla de la red y su padre, el nodo *C*, se convierte en un nodo gateway que debe realizar la multiplexación en ambos canales mientras que el resto de los nodos permanecen en el canal 1. En la Figura 3.3 (b) se presenta el planificador global para la red donde se establecen dos slots: uno para las actividades del canal 1 y el otro para las del canal 2. En el primer slot, el nodo *C* podrá enviar los mensajes que necesite a su padre (Nodo *A*) pero no podrá recibir ninguno del nodo *D*. Posteriormente, al momento en que el planificador indique el cambio al segundo slot, sólo podrán transmitir los nodos *C* y *D* en el canal correspondiente.

Esta situación se repetirá mientras que la interferencia perdure afectando la comunicación de la red o durante todo el tiempo de vida de la misma. Por supuesto, la primera alternativa requiere un mecanismo adicional que verifique la ausencia de la interferencia para poder promover la vuelta al estado inicial de la red.

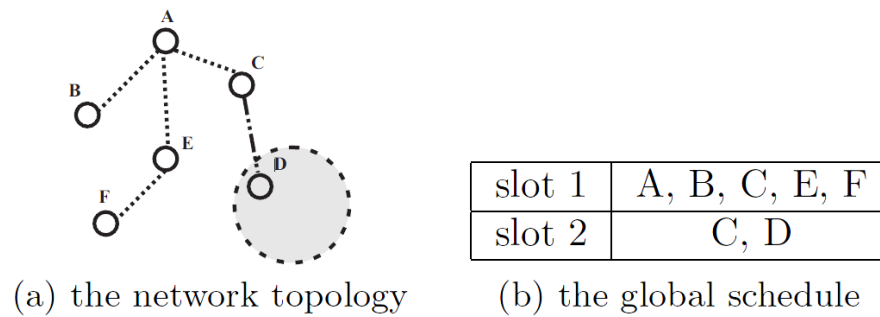


Figura 3.3. Escenario de interferencia y slots en Spectral Multiplexing Síncrona

La banda de frecuencia en la que se encuentra cada nodo y la capacidad que tiene cada uno para transmitir sus mensajes son mostradas en la Figura 3.4. Como se puede apreciar, los nodos *A*, *B*, *E* y *F* no pueden comunicarse en el segundo slot ya que el planificador común de la red establece que sólo los nodos que se encuentren en el segundo canal pueden hacerlo.

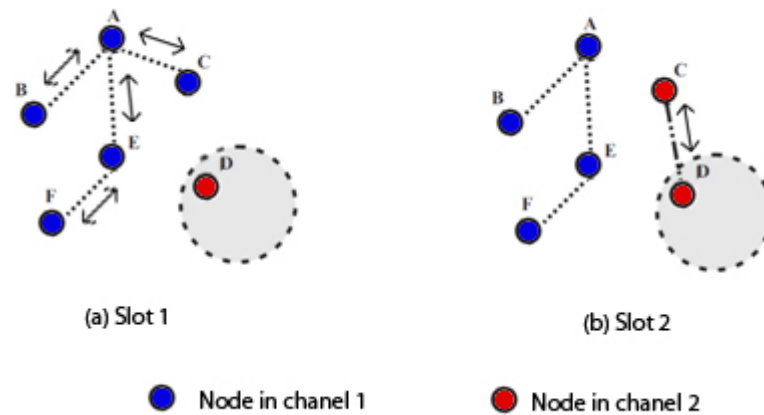


Figura 3.4. Comunicación en cada slot con la técnica Spectral Multiplexing Síncrona

Para lograr un único planificador el nodo sink debe iniciar el proceso difundiendo un mensaje a modo de flooding por toda la red que le indique a todos y cada uno de los motes el momento en que inicia y termina cada slot. El retraso que presenta este esquema se reduce al tiempo que toma enviar, propagar a través de aire y recibir un mensaje y para evitar algún tipo de alteraciones en estos tiempos, la estación base realiza este proceso periódicamente. Sin embargo, para asegurar una correcta sincronización, los nodos gateway deben enviar este mensaje rápidamente a través de todos canales en los que trabajan.

Para que el nodo sink pueda tener una lista con los canales en los cuales opera la red, tan pronto como un nodo fronterizo descubra que uno de sus vecinos se ha movido de canal para evadir una fuente de interferencia, debe calcular el tiempo de duración de sus slots basándose en su número de vecinos y el tráfico que representan y enviar un mensaje a la

estación base con esta información la cual calculará la duración independiente de cada slot como el tiempo mínimo de todos los que ha recibido para dicho slot.

Finalmente se debe reconocer que la naturaleza determinística de esta estrategia hacer que funcione correctamente más allá de lo complicado que pueda ser el escenario. Sin embargo, se debe pagar un coste extra que corresponde al overhead generado en el sistema para mantener dicha sincronización.

3.3.2.2 Spectral Multiplexing Asíncrona

Es la técnica de Channel Surfing donde cada nodo se preocupa únicamente de sus vecinos y de los canales donde se ubican así como también de establecer un planificador propio que permita la correcta sincronización de los canales a utilizar para la transmisión de datos (Wenyuan et al., 2007).

En la Figura 3.5 (a) se presenta la misma topología que en el caso de Spectral Multiplexing síncrona con la interferencia aislando al nodo *D*. Sin embargo, en la Figura 3.5 (b) se puede observar la diferencia, en cuanto a la técnica anterior, con respecto a los nodos que tienen permitido transmitir en cada intervalo de tiempo. En este caso, todos los nodos de la red pueden comunicarse con aquellos que se encuentren en su mismo canal.

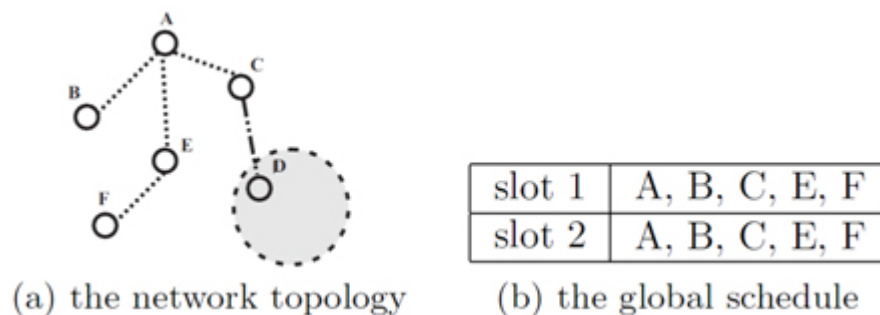


Figura 3.5. Escenario de interferencia y slots en Spectral Multiplexing Asíncrona

La vía más simple de aplicar esta medida por parte de un nodo gateway es saltando, a modo de round robin, a través de los canales dónde se encuentre un nodo con el cual se deben intercambiar datos. De cualquier forma, una estrategia como esta provocaría una alta tasa de paquetes perdidos (alrededor de 60%) que se puede reducir si se disminuye al máximo la duración de cada slot.

Una manera más eficiente de manejar el asunto de la sincronización es mediante el envío de mensajes de advertencia cada vez que el nodo gateway llegue a un canal. Así, los nodos que se encuentran en dicha banda de frecuencia de radio, conocerán el tiempo que el nodo fronterizo estará disponible para la transmisión y recepción de mensajes. Además, determinar la duración del slot queda completamente en manos del nodo gateway el cual deberá considerar el número de vecinos con los cuales debe contactar en cada canal para lograr calcular un tiempo que optimice las prestaciones y desempeño de la red.

Con esta medida se logra una tasa de paquetes perdidos muy inferior y, de conseguirse un tiempo ideal para cada slot y un tamaño adecuado para cada buffer, los resultados serán notablemente mejores que en el caso de la utilización del algoritmo de Round Robin.

A continuación se ilustra, a través de la Figura 3.6, el comportamiento de los nodos de la red a través de cada slot de tiempo. Como se puede apreciar, lo único que varía, es la comunicación entre el nodo *C* y los nodos *A* y *D* dependiendo del canal dónde se encuentre el nodo gateway.

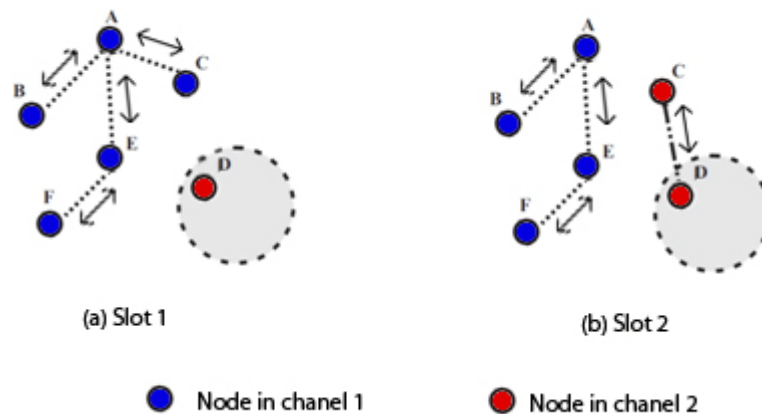


Figura 3.6. Comunicación en cada slot con la técnica Spectral Multiplexing Asíncrona

En resumen, se puede concluir que esta técnica de Channel Surfing es mucho más liviana en cuanto al overhead que genera en toda la red y flexible con respecto al cálculo de slots. Por ello es muy recomendable su uso cuando se prevé que la región afectada por la interferencia puede ser pequeña y regular ya que para regiones aisladas muy extensas, la red tendrá muchos nodos gateway generando gran cantidad de procesamiento que podría simplificarse y agruparse utilizando la técnica Spectral Multiplexing Síncrona.

3.4 Selección de una técnica de Channel Surfing

En el presente Proyecto Final de Carrera se diseña e implementa un protocolo llamado ASMCS para el cual se ha seleccionado la inclusión de la técnica de Channel Surfing Spectral Multiplexing asíncrona con el fin de incluir una medida que garantice la comunicación en toda la red a pesar de las interferencias que puedan existir en el ambiente que se desee estudiar.

Esta elección se debe a la intención de focalizar el estudio sobre una de estas estrategias y de esta manera, poder realizar un análisis más profundo sobre las mejoras que se introducen en la Red Inalámbrica de Sensores. Adicionalmente se toman en cuenta las adversidades que puede añadir el empleo de dicha técnica de Channel Surfing al funcionamiento de la red (mayor consumo de batería, aumento del delay en la entrega de mensajes, entre otros).

Para un estudio más completo se propone el diseño e implementación de las otras dos técnicas de Channel Surfing para su comparación, en cuanto a prestaciones, respecto a los distintos escenarios que se pueden suscitar en un ambiente monitorizado a través de una RIS.

Capítulo 4 EL PROTOCOLO ASMCS

ASMCS (Asynchronous Spectral Multiplexing Channel Surfing) es un protocolo que opera entre la capa de enlace y la capa de red de la pila de protocolos establecidos para una Red Inalámbrica de Sensores (ver Figura 4.1). Tiene como función principal mantener la comunicación de la red conectando a los nodos, que hayan sido aislados por causa de alguna interferencia, a través de canales de frecuencia de radio que no estén sometidos a dicha obstrucción. Para realizar esta compleja labor, emplea una técnica de Channel Surfing conocida como Spectral Multiplexing Asíncrona (ver sección 3.3.2.2).

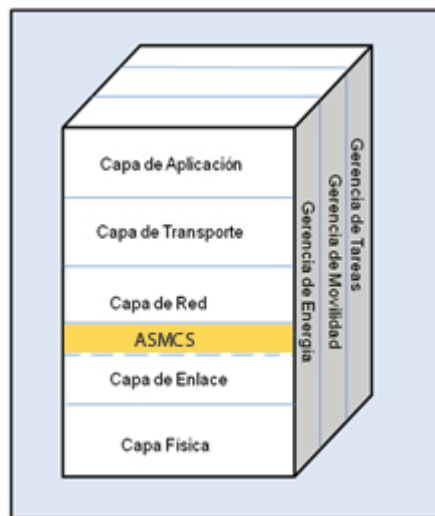


Figura 4.1. Ubicación de protocolo ASMCS en la pila de protocolos de una RIS

4.1 Descripción

ASMCS consta de un componente implementado en TinyOS, el cual está compuesto por un módulo y 3 interfaces. Se encuentra modularizado de manera tal que pueden emplearse cualesquiera protocolos en las capas superiores sin interferir con su funcionamiento. El código del protocolo se encuentra en el Apéndice A.

Su labor se basa esencialmente en guardar información sobre aquellos nodos con los que mantiene comunicación directa y vigilar que no se pierda contacto con alguno de ellos, en

cuyo caso, debe ir en su búsqueda en otro canal para enlazarlo de nuevo a la red y evitar la pérdida de comunicación. ASMCS no almacena información de otros nodos de la red que no sean sus vecinos directos ya que esto es responsabilidad del Protocolo de Red que se esté empleado.

El proceso de búsqueda de un vecino aparentemente aislado por causa de un elemento que genera interferencia en el medio, se lleva a cabo consultando en el resto de canales siguiendo una secuencia preestablecida y esperando en cada uno un tiempo permitiendo que ambos nodos coincidan en un canal y se restituya la topología de la red.

Una vez que un nodo haya encontrado a algún vecino aislado, pasa a ser un nodo gateway o nodo fronterizo el cual debe desplazarse por todos aquellos canales en los que tiene algún nodo con el cual comunicarse. Así se logra el correcto flujo de la comunicación en la red sin que afecte, a las capas superiores de la pila de protocolos, los canales en los que se encuentra cada nodo.

Todo nodo dispone de un buffer para cada vecino en el cual almacena aquellos mensajes que le deben enviar cuando dicho vecino se encuentra en otro canal diferente al suyo. En el momento en que ambos coincidan en alguna frecuencia de radio, estos mensajes serán entregados a su destinatario.

Esta información de los vecinos es almacenada en estructuras de datos en forma de tablas donde se incluyen las direcciones, canal de frecuencia, indicador de gateway (si el vecino es un nodo fronterizo o no) y el número de mensajes por enviar a dicho nodo. De la misma manera se dispone de los correspondientes buffers que sirven para guardar, temporalmente, los mensajes por entregar a cada vecino al momento de coincidir en algún canal.

4.2 Características

El protocolo ASMCS posee las siguientes características fundamentales:

- **Aumenta la tolerancia a fallos de la red:** gracias al empleo de la técnica Spectral Multiplexing Asíncrona, ASMCS permite superar caídas de comunicación en la red provocadas por la presencia de agentes generadores de interferencia.

- **Fácil utilización con cualquier protocolo de encaminamiento:** ASMCS provee las interfaces necesarias para que cualquier protocolo de la capa de red pueda enviar y recibir mensajes aprovechándose de las mejoras que introduce en la comunicación (ver sección 4.3.5).
- **Configurable:** los parámetros de uso del protocolo ASMCS pueden ser configurados, mediante el uso de los comandos que provee para establecer los valores de estas variables, por cualquier otro componente de la aplicación que se esté implementando (ver sección 4.3.2).
- **Agrega poco retardo a la comunicación:** los pocos milisegundos que le añade a la transmisión de datos el uso del protocolo ASMCS permiten disponer de esta funcionalidad sin temer la entrega de datos a destiempo significativo.
- **Aumento del consumo de batería:** dado que el protocolo ASMCS debe enviar mensajes propios durante la búsqueda de un vecino posiblemente aislado, así como también cada vez que un nodo gateway cambie el canal de radio en uso, la inclusión de este protocolo disminuye la vida de las baterías de los motes. Sin embargo, dicho consumo eléctrico puede ser tan despreciable como las necesidades de duración de batería de la red lo determinen.

De esta manera se presentan los beneficios y desventajas de incluir el protocolo ASMCS en un Red Inalámbrica de Sensores. Como en todo diseño e implementación de un RIS, se debe realizar un estudio que determine si el entorno a monitorear requiere de la inclusión de una técnica como la aquí propuesta o, por el contrario, si la red puede mantenerse y cumplir sus funciones sin ella.

4.3 Funcionamiento

El funcionamiento del protocolo ASMCS es detallado a continuación mediante la explicación de cada una de las estructuras de datos que maneja, las funciones que permiten su configuración, las distintas fases por las que pasan los nodos que lo utilizan, los mensajes que emplea para su administración y las facilidades que provee para su utilización en la implementación de una red inalámbrica de sensores.

4.3.1 Estructuras de Datos

Para la correcta administración de la información de sus vecinos cada nodo guarda, en una tabla llamada NeighborsTable, sus direcciones, una etiqueta que indica si el nodo es gateway o no, los canales donde se encuentran, el número de mensajes *hello* que no han contestado y el valor de la potencia de la señal del último mensaje recibido. En la Figura 4.2 se ilustra el formato de esta tabla.

0	1	2	3	4
01234567890123456789012345678901234567890123456789012345678901234567				
Address 1	gateway flag 1	actual channel 1	uncontested hellos 1	last rssi 1
Address 2	gateway flag 2	actual channel 2	uncontested hellos 2	last rssi 2
...
Address n	gateway flag n	actual channel n	uncontested hellos n	last rssi n

Figura 4.2. Formato de una tabla de vecinos con n nodos

Cada vez que se necesita conocer el dato de alguno de los vecinos es buscado en esta tabla y al mismo tiempo, siempre que se recibe algún mensaje, es actualizada con los nuevos valores obtenidos. El número máximo de vecinos que puede acoger la tabla está delimitado por el parámetro MAX_NEIGHBORS_NUMBER.

Con el objetivo de que no se pierda mensaje alguno por causa de que el emisor y el destinatario se encuentren en canales distintos, siempre que se le deba enviar algún dato a un nodo que se encuentre en otra frecuencia de radio, el mensaje se guardará en el buffer correspondiente a dicho nodo y, en cuanto ambos coincidan en el mismo canal, los mensajes serán enviados. Además la estructura que contiene el buffer también guarda el número de mensajes que deben ser enviados al vecino en cuestión y la posición, en el arreglo de mensajes, del último mensaje enviado.

En la Figura 4.3 se muestra la estructura de datos que almacena cada uno de los buffers de los vecinos, en los cuales se almacenan los mensajes que no se pueden enviar por causa de diferir en el canal de radio utilizado para un momento dado.

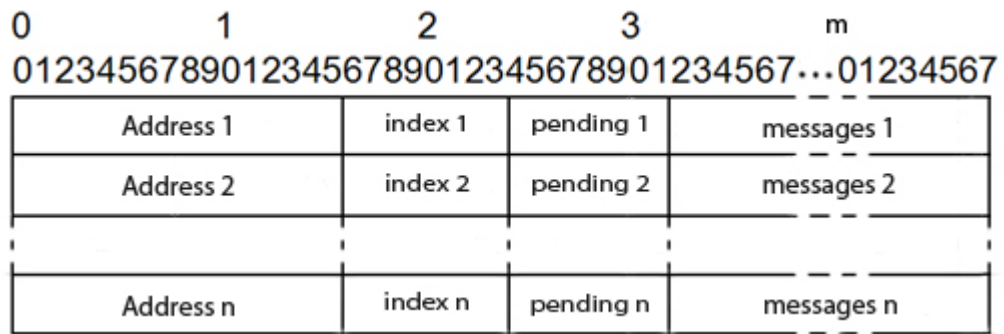


Figura 4.3. Formato de la estructura que guarda todos los buffers de los vecinos

De esta manera, cuando un nodo reciba un mensaje indicando que el vecino con "Address 1" se encuentra en su mismo canal, le enviará "pending 1" mensajes los cuales se encuentran en el arreglo "messages 1" a partir de la posición "index 1". Por otra parte el número máximo de mensajes que puede guardar cada buffer por vecino está determinado por la constante `MAX_BUFFER_SIZE`.

4.3.2 Configuración y parámetros por omisión

Para obtener y establecer el valor de algunos parámetros de funcionamiento de protocolo ASMCS, se proveen comandos "get" y "set" los cuales permiten configurar el protocolo a la medida de la aplicación en desarrollo. Dichos parámetros y sus valores por defecto son:

- `ALLOWED_HELLO_LOSS`: establece el número de mensajes hello que se permiten dejar de recibir antes de iniciar la búsqueda del nodo del que se espera una respuesta. El valor inicial es 3.
- `HELLO_INTERVAL`: indica el tiempo que transcurre entre el envío de dos mensajes hello. Dado que cada vez que se recibe un mensaje el número de hellos no recibidos es colocado en cero, este parámetro debe ser ligeramente superior al tiempo en que la aplicación espera recibir mensajes de sus vecinos, así se limita su envío a los casos en que realmente se necesita. De no especificarse alguno, el valor de esta constante es 0,5 segundos.
- `RSSI_THRESHOLD`: anuncia el valor mínimo de fuerza de señal recibida permitido para considerar al vecino no aislado. Si al recibir algún mensaje dicha potencia está por debajo de este valor, al momento de no obtener respuesta a un mensaje hello ya se considerará al vecino como aislado y se comenzará su búsqueda. Su valor por defecto es -87 dbm.

- CHANNEL_HOP: establece el número de canales que se saltarán en cada cambio de frecuencia mientras se esté buscando a algún vecino aislado. Su valor inicial es 4 y puede ser modificado en cualquier momento por la aplicación que use el protocolo ASMCS.
- NEIGHBOR_TIME_WAIT: constituye el tiempo máximo que un nodo esperará en el nuevo canal por recibir un mensaje del nodo aislado. De no recibirse alguno, se asumirá que el nodo tampoco puede comunicarse en ese rango de frecuencias y se probará en el siguiente. Su valor predeterminado es de 0,6 segundos.

Para poder utilizarlos se debe hacer uso de la interfaz ASMCSControl incluyendo la instrucción correspondiente tanto en el módulo como en el componente de la aplicación que se esté desarrollando. Para más detalles ver la sección 4.3.6

Por otra parte, hay una serie de constantes que no pueden ser modificadas en tiempo de ejecución ya que podrían comprometer la estabilidad del protocolo o porque para este momento su valor no tendría efecto alguno. Por esta razón dichos valores se deben modificar directamente en el archivo asmchannelsurfing.h. Dichos parámetros son:

- INI_CHANNEL: indica el canal en el cual los nodos empezarán a funcionar. Su valor predeterminado es 15 ya que representa el primer rango de frecuencia que no es cubierto por los aparatos comunes que operan bajo el estándar 802.11.
- MAX_BUFFER_SIZE: establece la cantidad máxima de mensajes que se pueden almacenar mientras su destinatario y el nodo en cuestión no coinciden en el mismo canal. Tiene como valor inicial 20 y, de sobrepasarse este límite, los mensajes más antiguos serán sobrescritos por los más recientes.
- MAX_NEIGHBORS_NUMBER: es el número máximo de vecinos que puede almacenar un nodo de la red. Su valor por omisión es 40 de acuerdo con un estudio sobre la capacidad por canal en una red que opera bajo el estándar 802.15.4 (Skogholt et al., 2006).

Como se puede apreciar en esta sección, ASMCS permite adaptar su funcionamiento a las necesidades de la aplicación que desee utilizarlo para obtener el máximo rendimiento posible.

Todos estos valores han sido calculados heurísticamente mediante una serie de pruebas que han logrado definirlos como los más adecuados para una aplicación de

monitorización que tome medidas aproximadamente cada medio segundo. Aunque esta consideración puede aumentar significativamente el gasto de batería de los nodos y elevar su coste de procesamiento, asegura que toda aplicación que utilice el protocolo ASMCS y no lo configure debidamente de acuerdo a sus necesidades, tolerará satisfactoriamente los fallos de comunicación producidos por las interferencias presentes en el entorno de estudio.

Para obtener el máximo rendimiento posible, se sugiere la correcta configuración de los parámetros adaptables del protocolo de acuerdo a los tiempos esperados de transmisión de datos de la aplicación.

4.3.3 Mensajes

ASMCS utiliza dos tipos de mensajes para su propia gestión y uno para que la aplicación que lo utilice pueda intercambiar datos con el resto de los nodos de la red. Es importante destacar que si no son utilizadas las interfaces que provee el protocolo para el envío de información, la técnica de Channel Surfing no será aplicada correctamente y los resultados que se obtendrán no serán adecuados.

Los mensajes de gestión del protocolo ASMCS se explican en las secciones siguientes.

4.3.3.1. Mensaje Hello (HelloMsg)

Este mensaje permite conocer si se ha perdido comunicación con un nodo de la red para comenzar su búsqueda en el siguiente canal de frecuencia de radio. La estructura de este tipo de mensajes puede verse en la Figura 4.4.

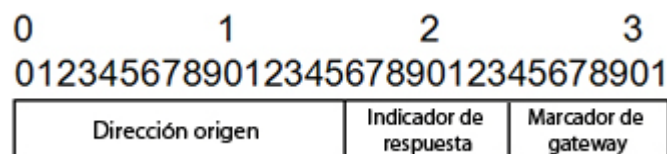


Figura 4.4. Formato del mensaje de hello (HelloMsg)

Los parámetros que se envían en este mensaje son:

1. Dirección de origen (`source_address`): permite saber al receptor del mensaje qué nodo se interesa por conocer su estado.
2. Indicador de respuesta (`is_reply`): anuncia si el presente mensaje de hello es respuesta a alguno anterior o si es generado intencionalmente por su emisor.
3. Marcador de gateway (`gateway_flag`): indica si el nodo emisor del mensaje es un gateway, para que el receptor tenga conocimiento de ello.

4.3.3.2. Mensaje de Anuncio (AdvertisementMsg)

Este mensaje es utilizado por aquellos nodos que se encuentran buscando a algún vecino probablemente aislado. Además, el mensaje de anuncio es empleado siempre que un nodo gateway llega a un canal de su ciclo, ya que a través de él notifica el tiempo que permanecerá en dicho canal. De esta forma, aquellos nódos que deban enviarle un mensaje al nodo gateway en cuestión, conocerán el tiempo del que disponen para ello. En la figura 4.5 se muestra un gráfico con la estructura de un mensaje de anuncio en el cual se pueden observar sus campos y el tamaño, en bits, de los mismos.

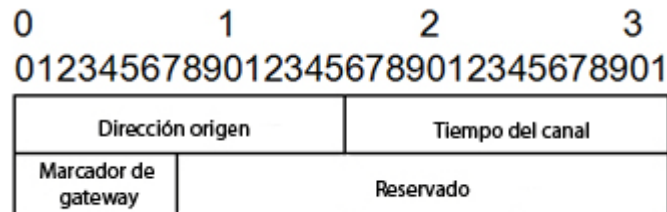


Figura 4.5. Formato del mensaje de anuncio (AdvertisementMsg)

Los campos que componen un mensaje de anuncio son:

1. Dirección de origen (*source_address*): permite saber al receptor del mensaje qué nodo se interesa por conocer su estado.
2. Tiempo del canal (*channel_time*): anuncia el tiempo que el nodo emisor permanecerá en el canal actual. Luego de transcurrir este tiempo, los nodos que tengan que entregarle algún mensaje no podrán hacerlo debido a que el nodo habrá vuelto a cambiar de canal.
3. Marcador de gateway (*gateway_flag*): indica si el nodo emisor del mensaje es un gateway, para que el receptor tenga conocimiento de ello.
4. Reservado (*reserved*): espacio disponible para futuras expansiones del protocolo.

4.3.3.3. Mensaje de Datos (DataMsg)

Es el mensaje que utiliza el protocolo ASMCS para enviar los datos que la aplicación necesite encapsulándolos y agregándole la cabecera necesaria para la propia administración. Al igual que con todos los mensajes anteriores, sirve para actualizar los datos del nodo emisor en la tabla de vecinos del mote que lo recibe.

En la Figura 4.6 se pueden observar los campos del mensaje de datos junto a su respectivo tamaño en bits.

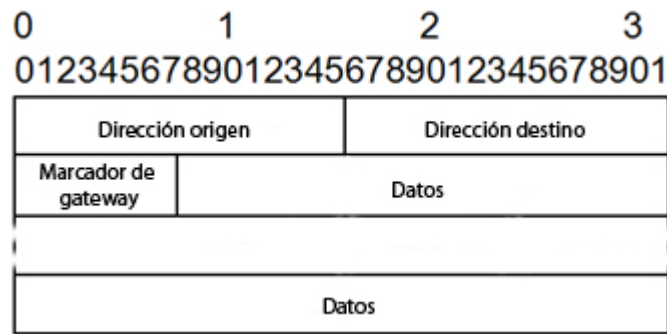


Figura 4.6. Formato del mensaje de datos (DataMsg)

Los campos que posee este tipo de mensajes son:

1. Dirección de origen (source_address): permite saber al receptor del mensaje qué nodo se interesa por conocer su estado.
2. Dirección de destino (dest_address): establece la dirección a la que van dirigidos los datos.
3. Marcador de gateway (gateway_flag): indica si el nodo emisor del mensaje es un gateway, para que el receptor tenga conocimiento de ello.
4. Datos (payload): corresponde a los datos que envía la aplicación. Puede tener un tamaño de hasta 16 bytes representado por cualquier estructura que maneje el protocolo de la capa de red.

4.3.4 Fases

EL protocolo ASMCS se ejecuta en distintas fases a medida que se van desarrollando las actividades normales de la red. En cada una de ellas se llevan a cabo acciones diferentes y toma decisiones en base a la información de la red que posee. El número total de etapas es cuatro y se explican a continuación.

4.3.4.1. Fase de Inicio

El protocolo inicializa sus estructuras de datos y establece como cana inicial el valor contenido en la variable "radio_channel" la cual contiene el valor de la constante INI_CHANNEL. En este momento ASMCS no realiza acción alguna y únicamente se encuentra a la espera de alguna actividad por parte de la aplicación.

4.3.4.2. Determinación de la Interferencia

Una vez que los mensajes de la aplicación hayan empezado a circular, el protocolo ASMCS poseerá información suficiente como para determinar cuándo un nodo de la red ha sido aislado por causa de alguna interferencia en el rango de frecuencias en uso.

Para saber que una región de la red está sometida a una interferencia que no permite la correcta comunicación, un nodo esperará no recibir respuesta a un número determinado de mensajes hello hasta considerar a algún vecino como aislado. El tiempo que transcurrirá desde que un nodo deja de recibir mensajes de algún vecino hasta que está seguro de que ha perdido la comunicación con él, viene dado por la siguiente fórmula:

$$t_{\text{vecino aislado}} = \text{HELLO_INTERVAL} \times \text{ALLOWED_HELLO_LOSS}$$

Ecuación 4.1 Cálculo del tiempo en asumir el aislamiento de un vecino

Otro mecanismo que se utiliza para saber que un nodo o región de la red se encuentra bajo una interferencia que amenaza con cortar el flujo de información es la inspección del RSSI (indicador de fuerza de señal recibida). De esta manera se puede saber cuándo un enlace entre dos o más nodos está en peligro de perderse y desarrollar acciones que puedan evitar tener que esperar a que la comunicación sea cortada por completo. Para ello está fijado un valor (RSSI_THRESHOLD) que establece el umbral sobre el cual debe estar la potencia de la señal recibida antes de comenzar a tomar medidas que permitan recuperar la posible comunicación perdida.

Una razón importante por la cual se utiliza este indicador y no el LQI (Link Quality Indicator) es debido a que el RSSI es menos variable entre cada recepción de mensajes (Srinivasan et al., S/F). De esta manera se puede evitar asumir que un nodo ha sido bloqueado a raíz de una interferencia y emprender su búsqueda por el resto de canales disponibles, cuando únicamente ha sufrido algún tipo de alteración que ha provocado la transmisión errónea de un sólo mensaje.

4.3.4.3. Búsqueda de un Nodo Aislado

En el momento en que un nodo pierde comunicación con algún otro, no debe suponer que se encuentra totalmente aislado de la red. Primero debe asegurarse que es incapaz de intercambiar información con alguno de sus vecinos enviando mensajes hello. Luego de esto

el nodo en cuestión supondrá que se encuentra bloqueado por la interferencia y se dirigirá al siguiente canal para encontrar a sus vecinos o ser encontrado por los nodos con los cuales guardaba comunicación anteriormente. Si por el contrario el nodo percibe que, aunque no logra comunicarse con un nodo de la red, aún puede enviar y recibir mensajes con sus otros vecinos en la frecuencia de radio actual, se convertirá en un gateway entre el nodo con el que ha perdido comunicación y la porción de red con la cual aún puede interactuar.

Justo cuando un nodo pasa al siguiente canal, ya sea para encontrar a un vecino con el que ha perdido comunicación o para ser encontrado por el resto de la red, enviará un mensaje de anuncio para hacerle saber a los nodos que ya se encuentran en ese canal que él también se comunicará en ese rango de frecuencias y esperará `NEIGHBOR_TIME_WAIT` segundos hasta finalizar la búsqueda allí o suponer que el nodo buscado tampoco puede comunicarse en la frecuencia actual y pasar al siguiente. Adicionalmente, los nodos que reciban este mensaje, podrán actualizar su tabla de vecinos con esta nueva información.

Un nodo saltará `CHANNEL_HOP` canales en cada intento por encontrar a un vecino aislado. Con esta propiedad se busca acelerar el proceso en el cual ambos nodos vuelven a encontrarse teniendo en cuenta que es muy probable que una fuente de interferencia, al afectar un determinado canal, perjudique también a los canales adyacentes al mismo.

La secuencia de saltos que un nodo ha de seguir cuando se dispone a cambiar de canal, ya sea para buscar alguno o para ser encontrado, es calculada mediante el número de canales que se saltarán en cada cambio. Esta sucesión empieza en el canal 11 (2,4 GHz) y llega hasta el 26 (2,48 GHz) tal y como lo especifica el estándar 802.15.4 sin embargo, como se anunció desde un principio, el canal inicial en el que se comunicarán los nodos será el 15 para evitar de antemano interferencias por parte del estándar 802.11. Cuando este el cálculo sugiera que el próximo canal será mayor que 26, se establecerá este último y, de darse el caso en que 26 sea el canal actual, ASMCS escogerá como próximo canal el 11. De esta manera se volverá a buscar un canal libre de interferencias a lo largo de toda la secuencia disponible.

4.3.4.4. Restablecimiento de la comunicación y sincronización

Luego de que un nodo ha encontrado a otro que suponía perdido, introducirá los nuevos datos en su tabla de vecinos para conocer los canales en los cuales debe transmitir cada mensaje, dependiendo del destinatario del mismo. En este momento, el nodo que actúa como gateway modificará su planificador para establecer el tiempo exacto que debe dedicar a

cada canal en los que debe moverse. El tiempo que este nodo debe pasar en un canal determinado viene dado principalmente por el número de nodos, con el que puede comunicarse en dicho canal, más otros factores como el overhead del sistema. En resumen, la fórmula queda expresada en la Ecuación 4.2.

Tiempo en canal $c = n^{\circ}$ vecinos en $c * (\text{tiempo de recepción} + \text{tiempo de envío} + \text{overhead})$

Ecuación 4.2. Tiempo que un nodo gateway permanecerá en el canal c

Si se calcula el tiempo máximo que un nodo tarda en vaciar su buffer de mensajes pendientes (4 ms a una tasa de 250 Kbps), el tiempo que tardaría en recibir el máximo número de mensajes que cada nodo puede tener almacenados para él (4 ms a una tasa de 250 Kbps) y se le suma el posible tiempo empleado en overhead del sistema y en el control de colisiones de transmisiones, se obtiene un valor cercano a los 10 ms. Esto permite determinar el tiempo que un nodo gateway pasará en cada canal como:

Tiempo en canal $c = n^{\circ}$ vecinos en $c * 10$ ms

Ecuación 4.3. Tiempo que un nodo gateway permanecerá en el canal c

Con el objetivo de que los nodos que mantienen comunicación con un gateway sepan el instante en el cual deben comunicarse con él, dicho gateway enviará un mensaje de anuncio justo en el momento en el cual haya entrado al canal en cuestión. En este mensaje se indicará también el tiempo que permanecerá disponible para recibir los mensajes que deban entregársele. Este mensaje será contestado sólo por los nodos gateway que lo reciban para que ambos estén enterados que coinciden en la frecuencia que están utilizando. Además, los mensajes que un nodo gateway deba entregarle al otro deben ser enviados antes que cualquier mensaje pendiente que se deba enviar al resto de vecinos, así se evita que el destinatario salga del canal de radio en el que han coincidido.

De esta manera, se define la duración de un ciclo de trabajo para un nodo gateway como el tiempo en el cual se ubica en cada uno de los canales en los que existe al menos un nodo con el cual puede establecer comunicación. Cabe destacar que un nodo de la red que no ejerce funciones de gateway no realiza este tipo de ciclo de trabajo debido a que siempre se encuentra en el mismo canal de frecuencia.

Las distintas situaciones que debe enfrentar un fronterizo en cada ciclo son las siguientes:

- Si el nodo gateway recibe un mensaje con destino "nodo A" y este se encuentra en el canal actual, el mensaje es enviado de inmediato.
- Si el nodo al que debe transmitir un mensaje no se encuentra en el rango de frecuencias actual, dicho mensaje se colocará en el buffer correspondiente a dicho vecino y será enviado tan pronto como el nodo gateway se encuentre en el canal indicado.

Igualmente las acciones que ejecuta un nodo gateway en cada canal son:

- Envío del mensaje advertencia para indicar su presencia en el canal actual.
- Recepción de respuestas del mensaje de advertencia por parte de los otros nodos fronterizos.
- Envío de mensajes que tengan como destino algún nodo gateway.
- Envío de los mensajes al resto de los nodos.
- Espera de mensajes para recibir.
- Cambio de frecuencia de radio al canal siguiente.

4.3.5 Interfaces Send y Receive

Con el objetivo de formar parte de un diseño modular y ubicarse, en la pila de protocolos, por debajo de la capa de red, ASMCS provee una interfaz para enviar mensajes y una interfaz para recibirlos. Los nodos que deseen enviar algún dato, incluyendo esta técnica de channel surfing, lo podrán hacer indicando simplemente el nodo destino y el mensaje que necesitan enviar. El protocolo le añadirá su encabezado y pasará a enviarlo según sus estrategias.

Por otra parte, cuando ASMCS reciba un mensaje, dichos datos serán desencapsulados del encabezamiento del protocolo y entregados a la capa superior para su propia administración.

En la siguiente sección se incluye el conjunto de pasos que debe seguir cualquier aplicación que desee hacer uso del protocolo ASMCS.

4.3.6 Incluir ASMCS en una Aplicación

La división en componentes de los distintos programas elaborados en el sistema operativo TinyOS hacen que su utilización, por parte de cualquier aplicación, sea muy simple.

En el caso de protocolo ASMCS se deben seguir los siguientes pasos:

1. Colocar en el directorio TOS_ROOT/tos/interfaces los archivos de interfaz del protocolo ASMCS: ASMCS`Send.nc`, ASMCS`Receive.nc` y ASMCS`Control.nc`.
2. Conectar, en el archivo de configuración, el componente ASM`ChannelSurfing.nc`.
3. Anunciar el uso de la interfaz ASMCS`Control.nc` en el módulo de la aplicación que se esté implementando.

De esta manera cualquier aplicación puede hacer uso del protocolo ASMCS. Los nombres de los comandos para su configuración se hallan en el código del mismo el cual está presentado en el apéndice A. Adicionalmente, en el apéndice B, se incluye una guía básica de programación en TinyOS que permite aprender a programar una aplicación sencilla y a utilizar componentes ya implementados.

Capítulo 5 EXPERIMENTOS

Con la finalidad de determinar el rendimiento ofrecido por el protocolo ASMCS y probar la factibilidad de su utilización en cuanto al consumo de recursos del sistema, se desarrollaron una serie de experimentos con distintas topologías de red que permitieron observar el comportamiento del protocolo en distintos escenarios.

Aunque estas pruebas están limitadas por el número de nodos empleados, los resultados han sido coherentes y confiables. Sin embargo, se realizaron simulaciones parciales del protocolo en una topología de tipo estrella (nodo sink rodeado de el resto de motes) que avalan los resultados aquí presentados.

5.1 Mediciones

Para probar el desempeño del protocolo ASMCS se tomaron en cuenta diversos parámetros y dos tipos de topología de red distintas. Todo esto con la finalidad de analizar, en el número máximo de casos posibles, el comportamiento del protocolo frente a las distintas situaciones que pueden surgir durante el ciclo de vida de una Red Inalámbrica de Sensores.

Los parámetros de referencia que se tomaron en cuenta para cada uno de los experimentos fueron los siguientes:

- Tasa de paquetes recibidos en relación a los enviados.
- Tiempo que tarda un mensaje en ir desde el nodo sink hasta cada nodo y volver de nuevo a la estación base (ver Figura 5.2).
- Consumo de energía.

Las topologías formadas para el estudio de las variables anteriormente mencionadas fueron las siguientes:

- Topología Estrella: aunque representa la disposición de los nodos más sencilla, una gran cantidad de Redes Inalámbricas de Sensores es establecida de esta forma.

Consiste simplemente en la conexión directa de cada nodo sensor con la estación base sin ningún mote intermedio (Figura 5.1)

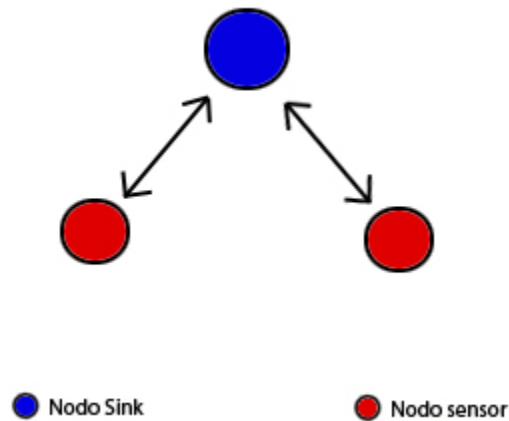


Figura 5.1 Ejemplo de topología estrella con 3 motes

- Topología multihop: ubica a los nodos de la RIS de manera tal que para que algunos de ellos tengan que comunicarse con el sink, sus datos tengan que ser retransmitidos por algunos nodos intermedios (Figura 5.2)

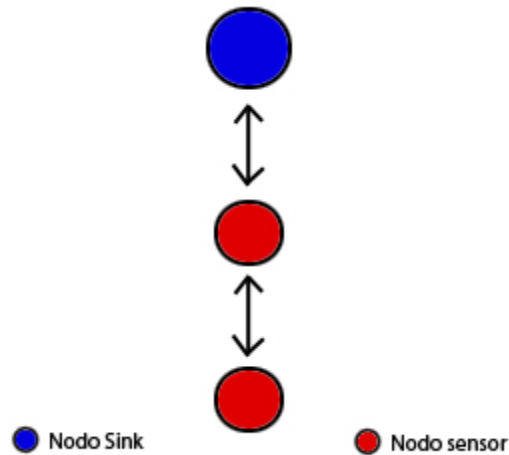


Figura 5.1. Ejemplo de topología multihop con 3 nodos

5.2 Resultados

Los resultados de los experimentos propuestos se muestran a continuación en tres secciones distintas donde se analiza por separado cada uno de los tres parámetros a estudiar.

La aplicación programada en los motes realiza una petición periódica, por parte del Nodo Sink al resto de los nodos de la red, del valor obtenido por su sensor de temperatura. En el momento en que cada nodo recibe este mensaje se dispone a calcular dicho valor y enviarlo de vuelta a la estación base.

Para cada prueba se ha ejecutado la aplicación empleando el protocolo ASMCS y posteriormente sin utilizarlo. Adicionalmente dicho programa utiliza un protocolo de encaminamiento (a nivel de capa de red) que se encarga del correcto flujo de datos entre el nodo sink y cada uno de los sensores.

5.5.1 Confiabilidad de la comunicación

Gracias a que el objetivo principal del empleo de una técnica de Channel Surfing es aumentar la tasa de paquetes entregados a pesar de las interferencias presentes en el medio de comunicación, no debe sorprender que al incluir el protocolo ASMCS el error en las transmisiones haya descendido hasta casi lograr el 100% de paquetes entregados.

En concreto, al ejecutar la aplicación sin la activación del protocolo ASMCS y encender una fuente de interferencia (Transmisión entre un Router WI-FI y un ordenador portátil) la tasa de paquetes satisfactoriamente entregados estuvo alrededor del 65%. Sin embargo, al incluir el protocolo ASMCS, dicha relación logró ubicarse entre 95% y 98% de mensajes correctamente entregados a su destinatario.

Con estos resultados se muestra el éxito que se puede obtener, en cuanto a confiabilidad de la comunicación, al aplicar una técnica de Channel Surfing.

5.5.2 Latencia en la transmisión de mensajes

El retardo generado por la utilización del protocolo ASMCS puede ser estudiado de una manera distinta ya que se le puede colocar una cota al mismo de acuerdo con los parámetros de configuración colocados.

Por una parte, antes de que se produzca interferencia alguna en la red, el retardo generado por la administración de los vecinos que realiza el protocolo ASMCS es descartable. Sin embargo, cuando algún nodo es interferido y la conexión de la red es restituida, el tiempo que tardará un mensaje en pasar a través de un nodo gateway que enlaza al nodo aislado con el resto de la red es, como máximo, 10 mili segundos por cada vecino que posea en dicha

frecuencia. Esto se debe a que, tal y como se explicó anteriormente, un nodo fronterizo no tardará más de 10 milisegundos en enviar y recibir los mensajes que sean requeridos por un nodo cualquiera de la red.

En base a estos cálculos, las pruebas realizadas se mantuvieron fieles a las extrapolaciones que se podían hacer, en el caso de una topología multihop se obtuvo un retardo adicional en la transmisión del mensaje entre 15 ms y 19 ms. Nótese que este tiempo está comprendido desde que el mensaje de petición es enviado desde el nodo sink hasta que se obtiene la respuesta del último nodo sensor).

5.5.3 Consumo de energía

Los estudios realizados en cuanto a uno de los parámetros de mayor importancia en el diseño e implementación de Redes Inalámbricas de Sensores se mostraron satisfactorios en relación con la confiabilidad agregada a la comunicación.

Al incluir el protocolo ASMCS en la aplicación de monitoreo sin tomar en cuenta configuración alguna de los parámetros pertinentes, se obtuvo un consumo de energía de un 15% mayor que en el caso de no utilizarlo. Sin embargo, al ajustar los parámetros del protocolo a los tiempos de medición y envío de datos de la aplicación, se logró disminuir este porcentaje y alcanzar un 10%.

De esta manera se puede afirmar que dependiendo de los requerimientos del fenómeno a estudiar mediante la implementación de una Red Inalámbrica de Sensores, puede ser necesaria la inclusión del protocolo ASMCS para aumentar su tolerancia a fallos en cuanto a las interferencias que puedan surgir.

CONCLUSIONES Y RECOMENDACIONES

El aumento en la producción de tecnologías que utilizan las bandas libres de frecuencia de radio aumentan la probabilidad de que se produzcan interferencias en el medio las cuales pueden llegar a bloquear parcial o totalmente alguna red inalámbrica que comparta dicho rango. En estos casos, uno de los estándares más perjudicados es el 802.15.4 debido a su poco ancho de banda y carencia de potencia suficiente en los dispositivos para la transmisión.

Debido a esta razón surge la necesidad de diseñar e implementar una serie de técnicas que permitan, a las Redes Inalámbricas de Sensores, ser tolerantes a estos riesgos y mantener su comunicación a pesar de las obstrucciones que pueden presentarse. Una causa importante de esta necesidad es la creciente utilización de este tipo de redes en el monitoreo de procesos críticos de industrias médicas, militares, químicas, entre otros.

En vista de todo esto, en el presente proyecto final de carrera se ha diseñado e implementado un protocolo llamado ASMCS el cual utiliza la técnica de Channel Surfing, conocida como Spectral Multiplexing Asíncrona, con el objetivo de enlazar aquellos nodos o regiones de la red que han sido aislados o bloqueados por causa de alguna fuente de interferencia. Esta tarea es conseguida mediante el cambio del canal de frecuencia empleado por los nodos en la transmisión de datos.

Sin embargo, se han tenido que analizar sus prestaciones en cuando a ciertos parámetros que son de suma preocupación cuando se realiza alguna actividad con Redes Inalámbricas de Sensores. Dichos factores son, entre otros de menos importancia, el consumo de batería y el retardo sufrido por la transmisión de mensajes que, en algunos casos, deben llegar rápidamente a la estación base. En relación a estos estudios se ha podido determinar que el uso de esta técnica logra elevar, hasta casi alcanzar un 100%, la tasa de mensajes transmitidos de manera satisfactoria aumentando, proporcionalmente a las necesidades de fiabilidad en la comunicación, el consumo de energía y el retardo en la transmisión de paquetes.

Debido a estas circunstancias se propone realizar un estudio que determine las necesidades de la aplicación que se desee emplear para determinar el grado de confiabilidad

requerido y decidir el grado de robustez que se quiere alcanzar al incluir el uso del protocolo ASMCS. Adicionalmente se deben configurar los parámetros que determinan las condiciones de funcionamiento del protocolo para buscar el comportamiento idóneo que le brinde a la red tanto un funcionamiento prolongado como la capacidad de afrontar las interferencias presentes en su ambiente de desempeño.

Por otra parte, se proponen una serie de tareas que pueden llegar a complementar el protocolo aquí desarrollado así como también mejorar el estudio sobre las técnicas de Channel Surfing. Dichas propuestas son:

- Incluir un protocolo de seguridad que permita la encriptación de los mensajes transmitidos por el protocolo ASMCS y que determine la secuencia de canales a cubrir mediante el uso de una clave privada. Esta funcionalidad permitiría agregar una alta habilidad al protocolo para superar aquellas interferencias producidas de manera intencional y que buscan entorpecer el correcto funcionamiento de la red.
- Con la finalidad de determinar las ventajas de cada una de las técnicas de Channel Surfing de acuerdo al escenario donde se desarrolle la red, se plantea la implementación del resto de estrategias y la comparación de cada una de acuerdo a situaciones distintas que puedan surgir como el bloqueo parcial, total o irregular de la red.

Finalmente se concluye que, como en la mayoría de los casos de estudio de Redes Inalámbricas de Sensores, se debe hacer un análisis previo que determine las verdaderas necesidades de la RIS a implementar para poder decidir correctamente tanto los requerimientos de confiabilidad necesarios en la transmisión de datos, como las exigencias de duración del tiempo de operación de la misma.

REFERENCIAS

Akyildiz, I., Su, W., Sankarasubramaniam, Y., y Cayirci, E. "A Survey on Sensor Networks". Revista IEEE Communications, Nov. 2002.

Bruin, Erik. "Helping Organizations Realize the Business Benefits of RFID and Wireless Sensor Network Technologies". Nov. 2005.

Chiasserini, C., Rao, R., "Coexistence Mechanisms for Interference Mitigation in the 2.4-GHz ISM Band". IEEE Transactions on Wireless Communications, Vol. 2, No. 5, Septiembre de 2003.

Crossbow Technology Inc. "Avoiding RF Interference Between WiFi and Zigbee". S/F.

Crossbow Technology Inc. "Readme & Quick Start Guide". 2007.

Crossbow Technology Inc. "MICAz OEM EDITION". S/F. Disponible en:

http://www.xbow.com/Products/Product_pdf_files/Wireless_pdf/MICAz_OEM_Edition.pdf

Crossbow Technology Inc. "USB INTERFACE BOARD MIB520". S/F. Disponible en:

http://www.xbow.com/Products/Product_pdf_files/Wireless_pdf/MIB520_Datasheet.pdf

Gay, D., Levis, P., Culler, D., Brewer, E., "nesC 1.2 Language Reference Manual", Ago. 2005.

González, R., Barenco, C., Pereira, W., Villapol, M., Roa, L., Dávila, N., Cárdenas, N., Landaeta, M., Cataldo, G., López, J., "Papel de Posicionamiento del Proyecto: Aplicaciones de Tecnologías Relacionadas a Redes Inalámbricas de Sensores en el Negocio de Petróleo y Gas". Sep. 2006.

González, R., Dávila, N., Lobalsamo, G., "Pruebas de alcance y coexistencia de IEEE 802.15.4 con otras tecnologías IEEE en la banda de frecuencia de 2.4 GHz". S/F.

IEEE Std 802.15.4-2003. Part 15.4: Wireless Medium Access Control (MAC) and Physical Layer (PHY) Specifications for Low-Rate Wireless Personal Area Networks (LR-WPANs) IEEE Standard for Information technology Telecommunications and information exchange between systems Local and metropolitan area networks Specific requirements. 12 de Mayo del 2003.

Intel. "Instrumenting the World An Introduction to Wireless Sensor Networks". Version 1, Feb. 2004. Disponible en:

http://www.intel.com/research/print/overview_instrument_world.pdf

Intel. "Intel Mote Sensor Nets / RFID", 2004. Disponible en:

<http://www.intel.com/research/exploratory/motes.htm>

Intel. "Intel Motes and Wireless Sensor Networks" Sep. 2005. Disponible en:

<http://www.intel.com/research/downloads/snoverviewcd.pdf>

Ruiz, L., "Um arquitetura para o gerenciamento de redes de sensores sem fio". Technical Report DCC/UFGM RT.005/2002. Departamento de Ciência da Computação da Universidade Federal de Minas Gerais, 2002.

Skogholt, M., " Practical Evaluation of IEEE 802.15.4/ZigBee Medical Sensor Networks". Norwegian University of Science and Technology Department of Electronics and Telecommunications. Jun. 2006.

TinyOS Community Forum. "An open-source OS for the networked sensor regime". <http://www.tinyos.net/>. US Berkeley, 2004.

Wenyuan, X., Wade, T., Yanyong, Z. "Channel Surfing: Defending Wireless Sensor Networks from Interference". Abr. 2007

Apéndice A

Código fuente del protocolo ASMCS

A continuación se presenta cada uno de los archivos que componen el protocolo ASMCS.

```

/**
 * Archivo asmchannelsurfing.h
 *
 * Contiene las definiciones de los parámetros de configuración del
 * protocolo ASMCS así como sus valores por defecto. De la misma manera
 * define los distintos tipos de mensajes y las estructuras de datos que
 * permiten guardar la información de los nodos vecinos y sus buffers de
 * datos.
 *
 * Componente: AMChannelSurfing
 * Modulo:     AMChannelSurfingM
 *
 * @author Cristiam Da Silva
 *
 * @modified 15/JUN/2009
 *
 * Barcelona, Junio de 2009.
 */

/*
 * Parámetros de configuración
 */
enum {
    AM_CS_HELLO = 0,
    AM_CS_ADVERTISEMENT = 1,
    AM_CS_DATA = 2,

    INI_CHANNEL=15,
    UNKNOW_CHANNEL=10,
    ALLOWED_HELLO_LOSS=3,
    HELLO_INTERVAL=500,
    RSSI_THRESHOLD=-87,
    CHANNEL_HOP=4,
    MAX_BUFFER_SIZE=20,
    MAX_NEIGHBORS_NUMBER=40,
    NEIGHBOR_TIME_WAIT=600,
};

/*
 * Estructura del mensaje de hello
 */
typedef struct HelloMsg {
    uint16_t source_address;
    uint8_t is_reply;
    uint8_t gateway_flag;
} __attribute__((packed)) HelloMsg;

/*
 * Estructura del mensaje de advertencia
 */
typedef struct AdvertisementMsg {
    uint16_t source_address;

```

```

uint16_t channel_time;
uint8_t gateway_flag;
} __attribute__((packed)) AdvertisementMsg;

/*
 * Estructura del mensaje de datos
 */
typedef struct DataMsg {
    uint16_t source_address;
    uint16_t dest_address;
    uint8_t gateway_flag;
    void* payload;
} __attribute__((packed)) DataMsg;

/*
 * Estructura que guarda la información de un vecino
 */
typedef struct Neighbor {
    uint16_t address;
    uint8_t gateway_flag;
    uint8_t actual_channel;
    uint8_t uncontested_hellos;
    int8_t last_rssi;
} Neighbor;

/*
 * Estructura que guarda la información todos los vecinos
 */
typedef struct NeighborsTable {
    Neighbor row[MAX_NEIGHBORS_NUMBER];
} NeighborsTable;

/*
 * Estructura que guarda la información de un buffer
 */
typedef struct Buffer {
    uint8_t index;
    uint8_t pending;
    uint16_t address;
    void* messages[MAX_BUFFER_SIZE];
} Buffer;

/*
 * Estructura que guarda la información todos los buffers
 */
typedef struct NeighborsBuffers {
    Buffer neighbor_buffer[MAX_NEIGHBORS_NUMBER];
} NeighborsBuffers;

```

```

/**
 * Archivo ASMChannelSurfing.nc
 *
 * Archivo de configuraciones y Wiring de las interfaces y componentes que
 * provee y utiliza el protocolo ASMCS.
 *
 * Componente: AMChannelSurfing
 * Modulo:     AMChannelSurfingM
 *
 * @author Cristiam Da Silva
 *
 * @modified 15/JUN/2009
 *
 * Barcelona, Junio de 2009.
 */

#include "asmchannelsurfing.h"

configuration ASMChannelSurfing {

provides {
    interface ASMCSSend;
    interface ASMCSReceive;
}
}
implementation
{
    components Main,
        ASMChannelSurfingM,
        CC2420RadioC,
        GenericComm as Comm,
        LedsC,
        TimerC;

    Main.StdControl -> TimerC.StdControl;
    Main.StdControl -> ASMChannelSurfingM.StdControl;
    Main.StdControl -> Comm.Control;

    ASMCSSend = ASMChannelSurfingM.ASMCSSend;
    ASMCSReceive = ASMChannelSurfingM.ASMCSReceive;

    ASMChannelSurfingM.CC2420Control -> CC2420RadioC;

    ASMChannelSurfingM.SendHelloMsg -> Comm.SendMsg[AM_CS_HELLO];
    ASMChannelSurfingM.SendAdvertisementMsg -> Comm.SendMsg[AM_CS_ADVERTISEMENT];
    ASMChannelSurfingM.SendDataMsg -> Comm.SendMsg[AM_CS_DATA];

    ASMChannelSurfingM.ReceiveHelloMsg -> Comm.ReceiveMsg[AM_CS_HELLO];
    ASMChannelSurfingM.ReceiveAdvertisementMsg -> Comm.ReceiveMsg[AM_CS_ADVERTISEMENT];
    ASMChannelSurfingM.ReceiveHelloMsg -> Comm.ReceiveMsg[AM_CS_DATA];

    ASMChannelSurfingM.Leds -> LedsC;

    ASMChannelSurfingM.SendBufferTimer -> TimerC.Timer[unique("Timer")];
    ASMChannelSurfingM.HelloTimer -> TimerC.Timer[unique("Timer")];
    ASMChannelSurfingM.SearchTimer -> TimerC.Timer[unique("Timer")];
    ASMChannelSurfingM.CicleTimer -> TimerC.Timer[unique("Timer")];
}
}

```

```

/**
 * Archivo ASMCSSend.nc
 *
 * Interfaz que le permite a una aplicación que utilice el protocolo ASMCS
 * enviar datos a otros nodos de la red.
 *
 * Componente: AMChannelSurfing
 * Modulo: AMChannelSurfingM
 *
 * @author Cristiam Da Silva
 *
 * @modified 15/JUN/2009
 *
 * Barcelona, Junio de 2009.
 */

```

```

interface ASMCSSend
{
    /**
     * Comando que permite el envío de mensajes por parte de la aplicación.
     *
     * @param dest destinatario del mensaje a enviar.
     * @param data mensaje que desea enviar la aplicación.
     */
    command void sendData(uint16_t dest, void* data);
}

```

```

/**
 * Archivo ASMCSReceive.nc
 *
 * Interfaz que permite a una aplicación que utlice el protocolo ASMCS
 * recibir mensajes de datos de otros nodos de la red.
 *
 * Componente: AMChannelSurfing
 * Modulo: AMChannelSurfingM
 *
 * @author Cristiam Da Silva
 *
 * @modified 15/JUN/2009
 *
 * Barcelona, Junio de 2009.
 */

```

```

interface ASMCSReceive
{
    /**
     * Evento que se dispara en el momento en que la aplicación recibe
     * un mensaje de datos por medio del protocolo ASMCS.
     *
     * @param m mensaje de datos recibido.
     */
    event void receive(DataMsg* m);
}

```

```

/**
 * Archivo ASMChannelSurfingM.nc
 *
 * Módulo que implementa el funcionamiento del protocolo ASMCS y sus
 * interfaces
 *
 * Componente: AMChannelSurfing
 * Modulo: AMChannelSurfingM
 *
 * @author Cristiam Da Silva
 *
 * @modified 15/JUN/2009
 *
 * Barcelona, Junio de 2009.
 */

```

```

module ASMChannelSurfingM
{
  provides {
    interface StdControl;
    interface ASMCSControl;
    interface ASMCSSend;
    interface ASMCSSReceive;
  }
  uses {
    interface CC2420Control;
    interface SendMsg as SendHelloMsg;
    interface SendMsg as SendAdvertisementMsg;
    interface SendMsg as SendDataMsg;
    interface ReceiveMsg as ReceiveHelloMsg;
    interface ReceiveMsg as ReceiveAdvertisementMsg;
    interface ReceiveMsg as ReceiveDataMsg;
    interface Timer as SendBufferTimer;
    interface Timer as HelloTimer;
    interface Timer as SearchTimer;
    interface Timer as CicleTimer;
    interface Leds;
  }
}
implementation
{
  // Datos locales
  bool sending_packet = FALSE;
  bool buffer_time_over = TRUE;
  uint8_t im_gateway = 0;
  uint8_t neighbor_gateway_channel;
  uint16_t neighbor_searching;

  // Mensajes utilizados
  TOS_Msg msg_data;
  DataMsg *data_msg = (DataMsg *)msg_data.data;
  TOS_Msg msg_hello;
  HelloMsg *hello_msg = (HelloMsg *)msg_hello.data;
  TOS_Msg msg_adv;
  AdvertisementMsg *adv_msg = (AdvertisementMsg *)msg_adv.data;

  // Inicialización de los parámetros de configuración
  uint8_t radio_channel = INI_CHANNEL;
  uint8_t allowed_hello_loss = ALLOWED_HELLO_LOSS;
  uint16_t hello_interval = HELLO_INTERVAL;
  int8_t rssi_threshold = RSSI_THRESHOLD;
  uint8_t channel_hop = CHANNEL_HOP;
  uint16_t neighbor_time_wait = NEIGHBOR_TIME_WAIT;
}

```

```
// Estructuras de datos de vecinos y buffers
NeighborsTable neighbors_table;
NeighborsBuffers neighbors_buffers;

// Reporte del estado del mote a traves de los leds.
void red() { call Leds.redToggle(); }
void green() { call Leds.greenToggle(); }
void yellow() { call Leds.yellowToggle(); }
```

```
////////////////////////////////////
//////////////////////////////////// GETS & SETS //////////////////////////////////////
////////////////////////////////////
```

```
command uint8_t ASMCSCControl.getAllowedHelloLoss(){
    return allowed_hello_loss;
}
```

```
command result_t ASMCSCControl.setAllowedHelloLoss(uint8_t hello_loss){
    allowed_hello_loss = hello_loss;

    return SUCCESS;
}
```

```
command uint8_t ASMCSCControl.getHelloInterval(){
    return hello_interval;
}
```

```
command result_t ASMCSCControl.setHelloInterval(uint8_t h_interval){
    hello_interval = h_interval;

    return SUCCESS;
}
```

```
command int8_t ASMCSCControl.getRSSIThreshold(){
    return rssi_threshold;
}
```

```
command result_t ASMCSCControl.setRSSIThreshold(int8_t r_threshold){
    rssi_threshold = r_threshold;

    return SUCCESS;
}
```

```
command uint8_t ASMCSCControl.getChannelHop(){
    return channel_hop;
}
```

```
command result_t ASMCSCControl.setChannelHop(uint8_t c_hop){
    channel_hop = c_hop;

    return SUCCESS;
}
```

```

command uint8_t ASMCSCControl.getNeighborTimeWait(){
    return neighbor_time_wait;
}

command result_t ASMCSCControl.setNeighborTimeWait(uint8_t n_time_wait){
    neighbor_time_wait = n_time_wait;

    return SUCCESS;
}

////////////////////////////////////
//////////////////////////////////// TABLE & BUFFER ///////////////////////////////////
////////////////////////////////////

/**
 * Inicializa los valores de la tabla de vecinos
 */
void initializeNeighborTable(){
    int i;
    for(i=0 ; i<MAX_NEIGHBORS_NUMBER ; i++){
        neighbors_table.row[i].address = MAX_NEIGHBORS_NUMBER+1;
    }
}

/**
 * Permite obtener un vecino de la tabla por su dirección
 */
Neighbor getNeighborFromTable(uint16_t address){
    Neighbor n;
    int i;

    n.address = MAX_NEIGHBORS_NUMBER+1;

    for(i=0 ; i<MAX_NEIGHBORS_NUMBER ; i++){
        if(neighbors_table.row[i].address == address){
            return neighbors_table.row[i];
        }
    }

    return n;
}

/**
 * Agrega un nodo a la tabla de vecinos
 */
result_t insertNeighborInTable(Neighbor n){
    int i;
    Buffer b;

    for(i=0 ; i<MAX_NEIGHBORS_NUMBER ; i++){
        if(neighbors_table.row[i].address == MAX_NEIGHBORS_NUMBER+1){
            neighbors_table.row[i] = n;
            b.index = 0;
            b.pending = 0;
            b.address = n.address;
            neighbors_buffers.neighbor_buffer[i] = b;
            return SUCCESS;
        }
    }

    return FAIL;
}

```

```

/**
 * Permite conocer el último valor del RSSI obtenido de la señal
 * recibida por un nodo
 */
int8_t getNeighborRSSI(uint16_t address){
    int i;

    for(i=0 ; i<MAX_NEIGHBORS_NUMBER ; i++){
        if(neighbors_table.row[i].address == address){
            return neighbors_table.row[i].last_rssi;
        }
    }
}

/**
 * Establece el indicador de gateway de un vecino
 */
void setNeighborGatewayFlag(uint16_t address,int8_t gw_flag){
    int i;

    for(i=0 ; i<MAX_NEIGHBORS_NUMBER ; i++){
        if(neighbors_table.row[i].address == address){
            neighbors_table.row[i].gateway_flag = gw_flag;
            i = MAX_NEIGHBORS_NUMBER;
        }
    }
}

/**
 * Actualiza los valores de RSSI, hellos no contestados y canal
 * actual de un nodo vecino
 */
void setNeighborRSSI(uint16_t address,int8_t rssi){
    int i;

    for(i=0 ; i<MAX_NEIGHBORS_NUMBER ; i++){
        if(neighbors_table.row[i].address == address){
            neighbors_table.row[i].last_rssi = rssi;
            neighbors_table.row[i].uncontested_hellos = 0;
            neighbors_table.row[i].actual_channel = radio_channel;
            i = MAX_NEIGHBORS_NUMBER;
        }
    }
}

/**
 * Permite conocer cuantos mensajes hello no se han recibido
 * de un determinado nodo
 */
uint8_t getHelloLoss(uint16_t address){
    int i;

    for(i=0 ; i<MAX_NEIGHBORS_NUMBER ; i++){
        if(neighbors_table.row[i].address == address){
            return neighbors_table.row[i].uncontested_hellos;
        }
    }
}

```

```

/**
 * Incremente el número de mensajes hello que no han sido
 * respondidos por un vecino
 */
void addHelloLoss(uint16_t address){
    int i;

    for(i=0 ; i<MAX_NEIGHBORS_NUMBER ; i++){
        if(neighbors_table.row[i].address == address){
            neighbors_table.row[i].uncontested_hellos += 1;
            i = MAX_NEIGHBORS_NUMBER;
        }
    }
}

/**
 * Coloca en 0 el número de mensajes hello no contestados por
 * un vecino
 */
void resetHelloLoss(uint16_t address){
    int i;

    for(i=0 ; i<MAX_NEIGHBORS_NUMBER ; i++){
        if(neighbors_table.row[i].address == address){
            neighbors_table.row[i].uncontested_hellos = 0;
            i = MAX_NEIGHBORS_NUMBER;
        }
    }
}

/**
 * Reestablece el número de mensajes hello no contestados por
 * todos los vecinos que se encuentran en un canal de radio
 */
void resetAllChannelHelloLoss(){
    Neighbor n;
    int i;
    uint16_t addr;

    for(i=0 ; i<MAX_NEIGHBORS_NUMBER ; i++){
        n = neighbors_table.row[i];
        addr = n.address;

        if(addr != MAX_NEIGHBORS_NUMBER+1 && n.actual_channel == radio_channel){
            n.uncontested_hellos = 0;
        }
    }
}

/**
 * Busca un nodo en la tabla de vecinos, de no encontrarlo, lo inserta.
 * Si por el contrario el nodo ya ha sido agregado anteriormente, actualiza
 * sus datos
 */
result_t insertOrUpdateNeighbor(uint8_t address,int8_t rssi,uint8_t gw_flag){
    Neighbor n = getNeighborFromTable(address);

    if(n.address == MAX_NEIGHBORS_NUMBER+1){
        n.address = address;
    }
}

```

```

    n.gateway_flag = gw_flag;
    n.actual_channel = radio_channel;
    n.uncontested_hellos = 0;
    n.last_rssi = rssi;
    if(insertNeighborInTable(n)){
        return SUCCESS;
    }
}
else{
    setNeighborRSSI(address,rssi);
    setNeighborGatewayFlag(address,gw_flag);
    return SUCCESS;
}

return FAIL;
}

/**
 * Inicializa los valores de la tabla de bufffers de vecinos
 */
void initializeNeighborsBuffer(){
    int i;

    for(i=0 ; i<MAX_NEIGHBORS_NUMBER ; i++){
        neighbors_buffers.neighbor_buffer[i].index = MAX_BUFFER_SIZE+1;
    }
}

/**
 * Agrega un mensaje a la cola de mensajes por enviar a un
 * nodo vecino
 */
result_t insertMessageInBuffer(uint16_t destination,void* msg){
    int i;
    Buffer b;

    for(i=0 ; i<MAX_NEIGHBORS_NUMBER ; i++){
        b = neighbors_buffers.neighbor_buffer[i];

        if(b.address == destination){
            b.messages[b.index++] = msg;

            if(b.pending < MAX_BUFFER_SIZE){
                b.pending++;
            }

            if(b.index == MAX_BUFFER_SIZE+1){
                b.index = 0;
            }
            return SUCCESS;
        }
    }

    return FAIL;
}

```

```

////////////////////////////////////
//////////////////////////////////// SEARCH & SINCHRONIZE ///////////////////////////////////
////////////////////////////////////

/**
 * Función que envía un mensaje de advertencia para anunciar que el
 * nodo ha llegado a un canal o que está buscando a algún vecino aislado
 */
void sendAdvertisement(uint16_t dest){

    adv_msg->source_address = TOS_LOCAL_ADDRESS;
    adv_msg->channel_time = neighbor_time_wait;
    adv_msg->gateway_flag = im_gateway;

    call SendAdvertisementMsg.send(dest,sizeof(AdvertisementMsg),&msg_adv);
}

/**
 * Calcula el tiempo que un nodo gateway permanecerá en un canal de
 * acuerdo al número de vecinos que allí posee
 */
int getCicleTimer(uint8_t r_channel){
    Neighbor n;
    int i;
    uint16_t neighbors = 0;

    for(i=0 ; i<MAX_NEIGHBORS_NUMBER ; i++){
        n = neighbors_table.row[i];

        if(n.actual_channel == r_channel){
            neighbors++;
        }
    }

    return 10*neighbors;
}

/**
 * Porta al nodo gateway al siguiente canal que debe cubrir para
 * mantener la comunicación de la red
 */
void goToNextCicleChannel(uint8_t r_channel){
    Neighbor n;
    int i;
    uint8_t temp_channel = r_channel;

    if(temp_channel == 26){
        temp_channel=11;
    }else{
        if(temp_channel+channel_hop>26){
            temp_channel = 26;
        }else{
            temp_channel+=channel_hop;
        }
    }

    for(i=0 ; i<MAX_NEIGHBORS_NUMBER ; i++){
        n = neighbors_table.row[i];

        if(n.actual_channel != radio_channel && n.actual_channel == temp_channel){
            radio_channel = temp_channel;
            resetAllChannelHelloLoss();
            call CC2420Control.TunePreset(radio_channel);
        }
    }
}

```

```

        sendAdvertisement(TOS_BCAST_ADDR);
        call CicleTimer.start(TIMER_ONE_SHOT,getCicleTimer(radio_channel));

    }else if(i+1 == MAX_NEIGHBORS_NUMBER){
        goToNextCicleChannel(temp_channel);
    }
}

}

/**
 * Cambia el canal de de radio por el siguiente en la secuencia establecida
 */
void changeChannel(){
    if(radio_channel == 26){
        radio_channel=11;
    }else{
        if(radio_channel+channel_hop > 26){
            radio_channel = 26;
        }else{
            radio_channel+=channel_hop;
        }
    }
    resetAllChannelHelloLoss();
    call CC2420Control.TunePreset(radio_channel);
}

/**
 * Función que se encarga de buscar a un vecino aislado por una interferencia
 * en la secuencia de canales disponibles
 */
void searchNeighbor(uint16_t address){
    neighbor_searching = address;
    changeChannel();
    call SearchTimer.start(TIMER_ONE_SHOT,neighbor_time_wait);

    sendAdvertisement(neighbor_searching);
}

////////////////////////////////////
//////////////////////////////////// STD CONTROL //////////////////////////////////////
////////////////////////////////////

/**
 * Inicializa el componente
 */
command result_t StdControl.init() {
    call Leds.init();

    return SUCCESS;
}

/**
 * Detiene la ejecución de la aplicación
 */
command result_t StdControl.stop() {
}

```



```

//////////////////////////////////////
////////////////////////////////////// TIMERS ////////////////////////////////////////
//////////////////////////////////////

```

```

/**
 * Evento producido cuando finaliza el tiempo en el cual estar  disponible
 * un nodo gateway para que le sean entregados sus mensajes retenidos
 */

```

```

event result_t SendBufferTimer.fired() {
    Neighbor n;
    int i;

    buffer_time_over = TRUE;

    for(i=0 ; i<MAX_NEIGHBORS_NUMBER ; i++){
        n = neighbors_table.row[i];

        if(n.address == neighbor_gateway_channel){
            n.actual_channel = UNKNOWN_CHANNEL;
        }
    }

    return SUCCESS;
}

```

```

/**
 * Evento producido cuando transcurre un HELLO_INTERVAL y se deben
 * reparar los nodos que no han respondido dichos mensajes
 */

```

```

event result_t HelloTimer.fired(){
    Neighbor n;
    int i;
    uint16_t addr;

    for(i=0 ; i<MAX_NEIGHBORS_NUMBER ; i++){
        n = neighbors_table.row[i];
        addr = n.address;

        if(addr != MAX_NEIGHBORS_NUMBER+1 && n.actual_channel == radio_channel){

            if(getHelloLoss(addr) == allowed_hello_loss){
                resetHelloLoss(addr);
                searchNeighbor(addr); //
                i=MAX_NEIGHBORS_NUMBER+1;

            }else if(n.uncontested_hellos > 0 && n.last_rssi < rssi_threshold){
                resetHelloLoss(addr);
                searchNeighbor(addr);
                i=MAX_NEIGHBORS_NUMBER+1;

            }else{
                addHelloLoss(addr);
                sendHello(addr,0);
            }
        }
    }

    return SUCCESS;
}

```

```

/**
 * Indica que ha culminado el tiempo de buscar a un vecino aislado
 * en el canal actual y que se debe probar en el siguiente
 */
event result_t SearchTimer.fired(){
    searchNeighbor(neighbor_searching);

    return SUCCESS;
}

/**
 * Indica que ha culminado el tiempo que el nodo gateway debe permanecer
 * en el canal actual y que debe mudarse al siguiente en la secuencia
 */
event result_t CicleTimer.fired() {

    goToNextCicleChannel(radio_channel);

    return SUCCESS;
}

////////////////////////////////////
//////////////////////////////////// DATA //////////////////////////////////////
////////////////////////////////////

/**
 * Indica la transmisión satisfactoria de un mensaje de datos
 */
event result_t SendDataMsg.sendDone(TOS_MsgPtr msg, result_t success) {

    atomic sending_packet = FALSE;

    return SUCCESS;
}

/**
 * Indica la recepción de un mensaje de datos
 */
event TOS_MsgPtr ReceiveDataMsg.receive(TOS_MsgPtr msg){
    DataMsg* m = (DataMsg*)msg->data;

    if(insertOrUpdateNeighbor(m->source_address, msg->strength, m->gateway_flag)){

        signal ASMCSReceive.receive(m);
    }

    return msg;
}

/**
 * Función que envía los mensajes que tienen por destinatario un nodo
 * gateway que acaba de llegar al canal y que se acumularon al no
 * estar presente
 */
void sendBuffer(uint16_t destination){
    int i;
    Buffer nb;

    for(i=0 ; i<MAX_NEIGHBORS_NUMBER ; i++){
        nb = neighbors_buffers.neighbor_buffer[i];

```

```

if(nb.address == destination){
    while(nb.pending > 0 && !buffer_time_over){
        if(nb.index == 0){
            call ASMCSSend.sendData(nb.address,nb.messages[MAX_BUFFER_SIZE]);
            nb.index = MAX_BUFFER_SIZE;
        }else{
            call ASMCSSend.sendData(nb.address,nb.messages[--nb.index]);
        }
        nb.pending--;
    }
}
}
}

////////////////////////////////////
////////////////////////////////////  ADVERTISEMENT  //////////////////////////////////////
////////////////////////////////////

/**
 * Indica la transmisión satisfactoria de un mensaje de anuncio
 */
event result_t SendAdvertisementMsg.sendDone(TOS_MsgPtr msg, result_t success)
{
    return SUCCESS;
}

/**
 * Indica la recepción de un mensaje de anuncio
 */
event TOS_MsgPtr ReceiveAdvertisementMsg.receive(TOS_MsgPtr msg){
    AdvertisementMsg* m = (AdvertisementMsg*)msg->data;

    if(m->source_address == neighbor_searching){
        call SearchTimer.stop();
        im_gateway = 1;
        call CicleTimer.start(TIMER_ONE_SHOT, 20);
        neighbor_searching = TOS_LOCAL_ADDRESS;
    }else{
        sendAdvertisement(m->source_address);
    }

    if(insertOrUpdateNeighbor(m->source_address,msg->strength,m->gateway_flag)){
        if(m->gateway_flag == 1){
            buffer_time_over = FALSE;
            neighbor_gateway_channel = m->source_address;
            call SendBufferTimer.start(TIMER_ONE_SHOT, m->channel_time);
            sendBuffer(m->source_address);
        }
    }

    return msg;
}

```

```

////////////////////////////////////
//////////////////////////////////// HELLO //////////////////////////////////
////////////////////////////////////

/**
 * Indica la transmisión satisfactoria de un mensaje hello
 */
event result_t SendHelloMsg.sendDone(TOS_MsgPtr msg, result_t success) {

    return SUCCESS;
}

/**
 * Indica la recepción de un mensaje hello
 */
event TOS_MsgPtr ReceiveHelloMsg.receive(TOS_MsgPtr msg){
    HelloMsg* m = (HelloMsg*)msg->data;

    if(insertOrUpdateNeighbor(m->source_address,msg->strength,m->gateway_flag)){
        if(m->is_reply == 0){
            sendHello(m->source_address,1);
        }
    }

    return msg;
}
}

```

Apéndice B

Guía básica de TinyOS y NesC

TinyOS es un sistema operativo orientado a eventos, diseñado para ser utilizado en redes inalámbricas de sensores embebidos (*embedded*). Originalmente fue desarrollado por investigadores adscritos al Departamento de Ingeniería Eléctrica y Ciencias de la Computación en la Universidad California en Berkeley y actualmente es distribuido bajo licencia de software libre.

A diferencia de otros sistemas operativos para motes, TinyOS resalta por su arquitectura basada en componentes y por estar explícitamente diseñado para ser usado en motes. Además, con TinyOS es posible conformar redes inalámbricas ad hoc, ya que es capaz de detectar nodos vecinos y con ciertas heurísticas y algoritmos determinar cuáles son las mejores rutas por la cual deben transitar los datos. Por esto, es considerado el estándar de facto en lo que se refiere a sistemas o ambientes de operación para RIS.

TinyOS está escrito en NesC, un lenguaje de programación también diseñado para dispositivos embebidos. Dicho lenguaje es una extensión del lenguaje C, pero orientado a una programación basada en eventos que es ideal para los motes, ya que por lo general dichos dispositivos deben estar dormidos y sólo se despiertan cuando eventos especiales ocurran como la adquisición de un dato o la recepción de un mensaje.

De acuerdo a (Intel, 2004a), TinyOS en su nivel más básico es un planificador encargado de gestionar las actividades de los componentes modulares que lo conforman. Ejecuta un único programa a la vez que está constituido por componentes proveídos por el sistema operativo y por otros componentes que son necesarios para que la aplicación diseñada pueda ser desplegada en una red de sensores. El programa antes descrito tiene dos tipos de hilos de ejecución: las tareas y los manejadores de eventos.

Las tareas son funciones ejecutadas en diferido o en algún momento del tiempo, por lo general en el futuro cercano. El planificador de estas tareas es de tipo FIFO, lo que quiere decir que se ejecutan en el orden en el que llegan a una cola. Ninguna tarea puede, por sí misma, interrumpir la ejecución de otra tarea o tomar el control de la unidad de procesamiento

del mote. Cuando una tarea cualquiera finalice su ejecución, le corresponde a la siguiente iniciar su ejecución.

Los manejadores de eventos son funciones que deben ser ejecutadas como respuesta a un suceso detectado por el hardware. Al igual que las tareas, los manejadores de eventos se ejecutan hasta culminarse por completo, y a diferencia de las tareas dichos manejadores sí pueden interrumpir a otras tareas e incluso interrumpir a otros manejadores de eventos.

Debido a que una tarea o un manejador de eventos pueden verse interrumpidos por otro manejador, es posible que se generen condiciones de carrera en ciertas regiones críticas de alguna aplicaciones. El compilador de NesC es capaz de detectar e indicarle al programador potenciales condiciones de carrera al momento de generar el código. Además, el lenguaje nesC señala las áreas del código de una aplicación que se deben ejecutar de manera atómica. Cuando un bloque de código es declarado con la palabra clave ‘atomic’, el manejador no cede el control a ningún otro evento hasta no haber terminado la ejecución de este bloque.

TinyOS es un sistema operativo que permite regular el manejo de la energía de un mote, ya que incluye componentes para controlar el encendido y el apagado de los dispositivos, aparte de tener la capacidad de colocarlos en modo de hibernación o de bajo consumo. Para el ahorro de energía, TinyOS permite que los nodos procesen la información que obtienen, y sólo si la información es interesante en el contexto de la aplicación, se procede a comunicar los resultados. (Intel, 2004a).

B.1.1. Conceptos básicos de nesC y TinyOS

Los programas se basan en un conjunto de componentes, los cuales son enlazados para la formación del programa entero. Estos componentes poseen tareas que se ejecutan de manera concurrente. El modelo de concurrencia está basado en los hilos de ejecución de TinyOS.

Para especificar el comportamiento de los componentes se utilizan las interfaces, que pueden ser proporcionadas o usadas. Las interfaces proporcionadas representan las funcionalidades que el componente provee al usuario. Las interfaces usadas representan las funcionalidades que el componente necesita para llevar a cabo su trabajo.

Los componentes son enlazados unos a otros a través de sus interfaces. Esto incrementa la eficiencia, permite un diseño robusto y un mejor entendimiento del programa.

B.1.1.1. Interfaces

Una interfaz en NesC representa una iteración bidireccional entre dos componentes, conocidos como el proveedor (*provider*) y el usuario (*user*) (Gay *et al.*, 2005). Las iteraciones de las interfaces se especifican a través de dos funciones: los comandos y los eventos.

Una interfaz tiene un único nombre, opcionalmente puede tener parámetros y contiene las declaraciones de sus comandos y eventos. Una interfaz con parámetros es llamada una interfaz genérica.

Ejemplo:

```
interface Queue<t> {  
    async command void push(t x);  
    async command t pop();  
    async command bool empty();  
    async command bool full();  
}
```

B.1.1.2. Componentes

Un componente en NesC tiene un nombre, argumentos de manera opcional, una especificación y una implementación (Gay *et al.*, 2005). La primera parte de la definición de un componente es la especificación, que es una declaración de elementos provistos o usados, donde cada elemento es una declaración de una interfaz, un comando o un evento.

Ejemplo de especificación de un componente.

```
module Simple {
    provides interface Init<int> as MyInit;
    uses interface SendMsg as MyMessage;
}
implementation {...}
```

Hay tres formas posibles de implementar un componente: módulos, componentes binarios y configuraciones

B.1.1.3. Módulos

Los módulos implementan una especificación de un componente en código C (Gay *et al.*, 2005). Este debe implementar todos los comandos y eventos provistos por el módulo, es decir, todos los comandos provistos por las interfaces y todos los eventos usados por las interfaces. Un módulo puede llamar cualquiera de estos comandos o señalar cualquiera de los eventos.

Llamadas a comandos y señalización de eventos

Un comando es llamado con la palabra clave *call*, mientras un evento es señalado con la palabra clave *signal*. La ejecución de comandos y eventos se realiza de manera inmediata, es decir, *call* y *signal* se comportan de manera similar a una llamada a función.

Por ejemplo, en un módulo que usa la interfaz Send del tipo SendMessage, la llamada a un comando sería de la forma:

```
call Send.send(1, sizeof(Message), &msg1)
```

B.1.1.4. Tareas

Las tareas permiten ofrecer la concurrencia interna dentro de un componente (Gay *et al.*, 2005). Ellas pueden llamar comandos y señalar eventos.

Las tareas se definen usando la palabra clave `task`. Por ejemplo:

```
task void myTask() { ... }
```

Las tareas se fijan anteponiéndole la palabra clave `post` a la tarea. Por ejemplo

```
post myTask();
```

B.1.1.5. Declaraciones Atómicas

Las declaraciones atómicas garantizan la ejecución completa del código que encierran, como si ningún otro cómputo ocurriera simultáneamente. Estas declaraciones son usadas para implementar la exclusión mutua y las actualizaciones concurrentes de estructuras de datos. Se recomienda que las declaraciones atómicas sean cortas. Un ejemplo simple de una declaración atómica es:

```
bool busy; // global
void f() { // called from an interrupt handler
    bool available;
    atomic {
        available = !busy;
        busy = TRUE;
    }
    if (available) do_something;
    atomic busy = FALSE;
}
```

B.1.1.6 Componentes binarios

Los componentes binarios son declarados con la palabra clave `component` y no tienen sección de implementación. Los programas que usan componentes binarios deben ser enlazados con un archivo objeto que le proporcione la implementación del componente binario, este archivo objeto puede ser el resultado de la compilación de otro programa en `nesC`.

El archivo objeto debe proveer las definiciones de los comandos y eventos proporcionados en el componente binario, y puede llamar sus comandos y eventos usados.

B.1.1.7. Configuraciones

Las configuraciones se usan para conectar un componente con otro, enlazando las interfaces usadas con las interfaces provistas. La sección de implementación consiste de una lista de elementos de configuración, estos pueden ser: componentes (*components*) o declaraciones de enlace (*wiring statements*).

B.1.1.8. Wiring

Las declaraciones de enlace (*wiring statements*) permiten conectar dos elementos denominados puntos finales (comandos, eventos o interfaces) (Gay *et al.*, 2005), que pueden ser externos o internos. Un elemento externo es un elemento de la especificación de la configuración, mientras que un elemento interno es un elemento de la especificación de un componente de la configuración.

Sean S1 y S2 dos puntos finales respectivamente, entonces existen tres tipos de declaraciones de enlace las cuales son:

- **S1 = S2**

S1 y S2 son externos, uno es proporcionado y el otro es usado. Si uno es interno y el otro es externo, ambos son proporcionados o usados.

- **S1- > S2**

Ambos son internos. Uno es proporcionado y el otro usado.

- **S1 < -S2**

Es equivalente a S2- > S1.

B.1.1.9. Concurrencia

NesC asume un modelo de ejecución que consiste en ejecución completa de tareas (*run to-completion task*), y manejadores de interrupciones (*interrupt handlers*) que son señalados de forma asíncrona por el hardware (Gay *et al.*, 2005). Un planificador para nesC puede ejecutar las tareas en cualquier orden, pero debe cumplir con la regla de la ejecución completa de la tarea (el planificador estándar de TinyOS sigue una política FIFO).

Debido a que las tareas no son apropiativas y se ejecutan de forma completas ellas son atómicas entre sí, pero no son atómicas con respecto a los manejadores de interrupciones. Debido a este modelo de ejecución, los programas en nesC son susceptibles a lo que se conoce como condiciones de carrera (*race conditions*). Esta condición es evitada accediendo al estado compartido sólo en declaraciones atómicas.

El código de un programa en nesC se divide formalmente en dos partes:

- **Código Síncrono (SC):** código (funciones, comandos, eventos, tareas) que sólo es accesible desde tareas.
- **Código Asíncrono (AC):** código que es accesible desde al menos un manejador de interrupciones.

Aunque la no apropiatividad elimina las condiciones de carrera entre las tareas, cabe la posibilidad que éstas se presenten entre código síncrono y asíncrono, así como entre código asíncrono y asíncrono. Para prevenir esto nesC genera advertencias cuando se produce una violación de las siguientes reglas:

- **Race-free invariant 1:** si una variable x se está escribiendo en un AC, entonces todos los accesos a x deben ocurrir en secciones atómicas.
- **Race-free invariant 2:** si una variable x es leída en un AC, entonces todas las escrituras sobre x deben ocurrir en secciones atómicas.

B.1.2. Instalación de TinyOs en Ubuntu

A continuación se presentan los pasos para la instalación de TinyOS en Ubuntu, en alguna distribución posterior a la Hardy.

1. Ir a System → Administration → Synaptic Package Manager
2. En Synaptic Package Manager ir a Settings → Repositories
3. Ubicarse en la pestaña 3rd party Software. Agregar un nuevo repositorio seleccionado el botón Add y copiar las siguiente línea:

```
deb http://tinynos.stanford.edu/tinynos/dists/ubuntu hardy main
```

Nota: Si la distribución es Heron, igualmente colocar Hardy.

4. Hacer click en Ok y recargar todos los respositorios.
5. Después que se ha cargado el nuevo repositorio, se puede buscar a través del botón Search los paquetes relacionados con TinyOS y marcar para luego ser instalados. En caso que se esté utilizando la distribución de Ubuntu Heron, puede al realizar la búsqueda los paquetes no aparezcan. Pasa solucionar esto, se debe realizar la búsqueda por repositorios presionando el botón Origin.
6. Una vez que la instalación ha finalizado, se debe establecer un shell environment. Para esto, se debe abrir una consola (shell) y escribir gedit ~/.bashrc y añadir las siguientes líneas al final del archivo que se abre:

```
#added line below for tinynos
export TOSROOT=/opt/tinynos-2.1.0
export TOSDIR=$TOSROOT/tos
export CLASSPATH=$TOSROOT/support/sdk/java/tinynos.jar
export MAKERULES=$TOSROOT/support/make/Makerules
export PATH=/opt/msp430/bin:$PATH
```

Nota: Asegurarse que en la línea export TOSROOT=/opt/tinynos-2.1.0 se esté colocando la versión correcta de TinyOS.

7. Cerrar la consola.

Para verificar que la instalación haya finalizado correctamente se deben seguir estos pasos:

1. Abrir una nueva consola (shell).
2. Escribir `echo $MAKERULES` y si todo está correcto, se mostrará el path:

```
/opt/tinyos-2.1.0/support/make/Makerules
```