ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA DE TELECOMUNICACIÓN
UNIVERSIDAD POLITÉCNICA DE CARTAGENA

Proyecto Fin de Carrera

“Diseño e implementación de un entorno de simulación para redes de sensores inalámbricos”

AUTOR: Ana Belén Corral Ignoto
DIRECTOR: Esteban Egea López
CODIRECTOR: Alejandro S. Martínez Sala

Julio / 05

Autor	Ana Belén Corral Ignoto
E-mail del Autor	abcorral@ono.com
Director(es)	Esteban Egea López
E-mail del Director	esteban.egea@upct.es
Codirector(es)	Alejandro Santos Martínez Sala
Título del PFC	"Diseño e implementación de un entorno de simulación para redes de sensores inalámbricos"
Descriptores	

Resumen

Recientemente ha surgido un gran interés por la investigación y experimentación de las redes de sensores inalámbricos, debido a la incorporación y uso de ellas en ámbitos muy dispares. Entre los problemas que estas redes plantean podemos encontrar la limitación de los recursos energéticos de los dispositivos así como la propagación de los datos a través de redes multisalto.

El proyecto aquí presentado se centra en el diseño y desarrollo de un entorno de simulación para redes de sensores inalámbricos así como la implementación de dos protocolos característicos de estas redes que permitan validar a su vez el funcionamiento del simulador. Estos protocolos son S-MAC, protocolo de control de acceso al medio que destaca por mejorar las características de consumo de potencia de los nodos, y DSR, protocolo de enrutamiento el cual utiliza para lograr su función las propiedades que presentan las redes distribuidas. Para lograr todos estos objetivos hemos hecho uso de la librería de libre distribución OMNeT++ por su modularidad y flexibilidad.

Titulación	Ingeniería de Telecomunicación
Intensificación	Planificación y Gestión de Telecomunicaciones
Departamento	Tecnologías de la Información y las Comunicaciones
Fecha de Presentación	Julio – 2005

Índice

Índice figuras y tablas

Capítulo 1
Introducción

1.1 Fundamentos del proyecto

Los sensores han sido tradicionalmente elementos indispensables en los procesos industriales debido a la capacidad que proporcionaban de monitorizar y manipular las magnitudes físicas involucradas en los diferentes procesos productivos. La conectividad entre los sensores se realizaba mediante el uso de redes cableadas tradicionales. Actualmente las líneas de investigación intentan conseguir una mayor distribución de sensores de monitorización, dispuestos en cualquier localización, autónomos e inalámbricos.

El continuo desarrollo de estas particulares redes ha provocado su incorporación y uso en ámbitos muy dispares. Desde la monitorización ambiental (humedad, temperatura, luz, etc.) fundamental para el desarrollo de la domótica, pasando por las aplicaciones militares, industriales, médicas o comerciales.

Sin embargo a pesar de los grandes desarrollos que se están produciendo en este campo de investigación, son muchos los problemas que quedan hoy en día por solucionar, problemas derivados directamente de las condiciones distribuidas e inalámbricas de los dispositivos que forman las redes de sensores.

Entre los problemas que comentábamos anteriormente podemos destacar principalmente dos. En primer lugar los sensores se encuentran directamente ligados con el entorno que los rodea, a diferencia de las redes de datos tradicionales. Esto provoca que a la hora de diseñar el buen funcionamiento de una red de sensores, los parámetros a considerar sean muy elevados, acrecentados por la característica inalámbrica del medio de transmisión utilizado.

En la mayoría de los casos los dispositivos usados para la implementación de redes de sensores inalámbricos se caracterizan por ser pequeños, autónomos, muy numerosos y con importantes limitaciones energéticas. Todos estos factores originan que los estudios analíticos a realizar sean muy complejos y los estudios experimentales muy costosos. Por ello vemos la necesidad que existe por parte de los investigadores y desarrolladores de realizar simulaciones previas antes de la implantación física de estas redes.

Por todo ello, los simuladores de redes de sensores inalámbricos se han convertido en herramientas fundamentales para el desarrollo, evolución e implantación de estas redes. Además estos simuladores necesitan modelos adicionales que no están presentes en los simuladores de redes tradicionales lo que obliga a diseñarlos cuidadosamente.

1.2 Objetivos

El objetivo del proyecto es estudiar protocolos que den solución a los problemas anteriormente expuestos. Para ello se realizará un simulador de redes de sensores para el que se implementarán dos novedosos protocolos específicos de este tipo de redes.

Primero estudiaremos el protocolo de acceso al medio S-MAC como una posible solución tanto a los problemas antes mencionados como a los originados por las limitaciones impuestas en el consumo de potencia de los sensores.

A su vez, como una posible solución al enrutamiento de los datos en las redes distribuidas analizaremos el protocolo de nivel de red DSR.

Por último, una vez realizados todos estos estudios previos, desarrollaremos un entorno de simulación de redes de sensores mediante el uso de la herramienta OMNeT++, así como la programación de los dos protocolos antes citados.

La característica modular del entorno desarrollado permitirá la incorporación de nuevos protocolos para las diferentes capas que constituirán los nodos de la red, así como la incorporación de diferentes tipos de canales de transmisión, número de capas en los nodos, topologías, etc.

Un objetivo adicional es que el simulador pueda ser extendido con la incorporación de nuevos módulos y protocolos. Para facilitar esta tarea junto al código facilitado se presentará una amplia documentación, en formato HTML.

1.3 Herramientas utilizadas

Para el desarrollo de este proyecto hemos hecho uso de las siguientes herramientas:

OMNeT++ [1]

Entorno de simulación de eventos discretos, modular y orientado a objetos. Debido a la importancia que esta herramienta tiene para la realización del proyecto aquí presentado, la comentaremos de forma más amplia en el *Capítulo 3* de este documento.

Kdevelop [2]

Entorno de programación de código abierto desarrollado por KDE (*K Desktop Environment*). Programa completo de programación visual, que se caracteriza por contar con varias herramientas para la concepción de interfaces y de generación automática de código. En nuestro caso, Kdevelop ha sido usada únicamente como editor de códigos C++.

Doxygen [3]

Sistema de extracción de documentación de código fuente de lenguajes tales como C, C++, Java, CORBA, PHP, etc. Mediante su uso podemos obtener documentación en diferentes formatos como son HTML, Latex, RTF, PDF, páginas man, postscript y XML. La introducción en los comentarios del código realizado de palabras claves propias de este programa puede proporcionar la obtención de una documentación clara, concisa y fácil de usar.

Capítulo 2
Estudio previo

2.1 Redes inalámbricas

2.1.1 Introducción

Desde hace relativamente poco tiempo, estamos viviendo lo que puede significar una revolución en el uso de las tecnologías de la información tal y como actualmente las conocemos. Las redes inalámbricas (*Wireless Networks*, WN) constituyen un sistema de comunicación de datos flexible utilizado como alternativa a las redes cableadas o como extensión de éstas.

En este tipo de redes los datos se propagan en un medio de transmisión no guiado mediante ondas electromagnéticas transmitidas y recibidas por antenas. Mediante el uso de estas tecnologías de radiofrecuencia se permite a los usuarios de las redes una mayor movilidad al minimizarse las conexiones cableadas.

Las aplicaciones de las redes inalámbricas son muy elevadas, y cada vez surgen nuevos ámbitos de desarrollo en campos muy diversos.

2.1.2 Características de las redes inalámbricas

Para poder comprender mejor las propiedades que poseen estas redes, las cuales pueden ser positivas o negativas, realizaremos este estudio comparándolas con las de las redes tradicionales, las redes cableadas.

2.1.2.1 Ventajas

En ámbito general, las principales ventajas que las redes inalámbricas presentan sobres las redes cableadas tradicionales son las siguientes:

Movilidad

Frente a las redes tradicionales la liberación de los cables permite a los usuarios acceder a la información y a los servicios en cualquier lugar y en cualquier momento. Este uso de información en tiempo real supone mayor productividad y posibilidad de servicios.

Facilidad y rapidez de despliegue

En el proceso de instalación de una red podemos encontrar muchos lugares en los que es difícil instalar cables. Incluso en edificios modernos y preparados, las instalaciones pueden constituir un proceso lento y muy costoso.

Mediante el uso de redes inalámbricas, se origina una minimización importante de todos estos problemas.

Flexibilidad

A la facilidad de despliegue antes citada se une la simplicidad de reconfiguración de la red. Las redes inalámbricas interconectan un elevado número de usuarios a

través de estaciones base. Un incremento o decremento de este número de usuarios, una vez instaladas las antenas y las estaciones base, se convierte únicamente en una cuestión administrativa (proceso de autorización de usuarios).

Reducción de costes

Uno de los mayores costes de la factura final del despliegue de una nueva red es sin duda alguna la instalación del cableado. Esto se ve incrementado si en la topología de la red se dan cambios frecuentes o el medio en el que se encuentra es muy dinámico. Además, en ciertos entornos, como el industrial, el cableado puede constituir un problema realmente serio por las dificultades que plantea su instalación (imposibilidad de realizar obra civil, condiciones hostiles, maquinaria).

Escalabilidad

Realizar un cambio de topología, en una red inalámbrica, es bastante sencillo y el tratamiento es el mismo tanto si la red es pequeña como si es grande.

2.1.2.2 Inconvenientes

A pesar de todas estas características antes expuestas, que han sido las que han originado la rápida extensión de las redes inalámbricas en nuestro entorno, no todo son ventajas. Estas redes presentan ciertos inconvenientes que han frenado una implantación de las mismas aún más rápida:

Canal físico

La propia disponibilidad del mismo es un inconveniente: el espectro radioeléctrico, en el cual se realiza la transmisión de los datos, está limitado y se necesita licencia para su uso en muchos casos.

Así mismo, la propagación de la información presenta numerosos problemas, como las interferencias o la propagación multicamino, lo que se traduce en tasas de error elevadas. Como consecuencia directa el hardware necesario sea más complejo y los equipos más lentos en la mayoría de los casos.

Seguridad

La naturaleza de las transmisiones en el medio no guiado hace que el acceso a la red esté disponible para cualquiera, por lo que es necesaria la implantación de ciertas medidas de seguridad que eviten el uso de la red por parte de elementos no autorizados.

2.1.3 Clasificación

Debido a la multitud de tecnologías de comunicaciones inalámbricas que nos encontramos debido al auge de estos dispositivos en los últimos años, nos vemos en la necesidad de realizar una clasificación. El criterio utilizado para la misma será en función de la extensión del área que cubren.

Redes de área personal [4]

Las denominadas redes de área personal inalámbricas (*Wireless Personal Area Network*, WPAN) se caracterizan por tener una cobertura de corto alcance y por estar orientadas a la sustitución de cables en dispositivos más simples. Dentro del ámbito de estas redes podemos integrar las siguientes tecnologías:

- IrDA.- Sus principales características son su limitación por su corto alcance, necesidad de una visión sin obstáculos entre los dispositivos que se comunican y las bajas velocidades que alcanza (hasta 115kbps).

- Bluetooth.- Opera en la banda de 2,4GHz, con un rango espacial de unos 10m y tasas de transferencia de 1Mbps.

Redes de área local [4]

Las redes de área local inalámbricas (*Wireless Local Area Network*, WLAN) incluyen tecnologías que operaran en bandas sin licencia ISM (*Industrial, Scientific and Medical*) a 2,4GHz y 5GHz en las que se permite hasta 1W de potencia de transmisión por equipo. En este grupo podemos destacar los siguientes estándares de transmisión:

- IEEE 802.11.- Tecnología inalámbrica que incluye en sus especificaciones el nivel físico y el nivel de enlace de datos. Dentro de este estándar podemos destacar tres variantes, las cuales se diferencian en el nivel físico usado:

    1. 802.11a

    Proporciona de 25 a 54Mbps en la banda sin licencia ISM de 5GHz, usando modulación OFDM (*Orthogonal Frequency Division Multiplexing*) y códigos correctores de errores (*Forward Error Correcting*, FEC).

    2. 802.11b

    Hasta 11Mbps de datos brutos en la banda ISM de 2,4GHz, usando técnicas de transmisión de espectro ensanchado por secuencia directa (*Direct Sequence Spread Spectrum*, DDSS). Si se usan técnicas de salto de frecuencia (*Frequency Hopping Spread Spectrum*, FHSS) se consigue hasta 2Mbps.

    3. 802.11g

    Compatible con los dispositivos que implementan IEEE 802.11b. Permite obtener hasta 54Mbps en la bada de 2,4GHz.

- HIPERLAN.- Estándar europeo especificado por el ETSI (*European Telecommunication Standards Institute*). Podemos encontrar cuatro implementaciones:

    1. HIPERLAN/1

    Usado como extensión de redes cableadas trabaja en las bandas de frecuencia de 5,15GHz y 17,1GHz, alcanzando tasa de transferencia de 23,5Mbps.

    2. HIPERLAN/2

    Usa la banda de 5GHz y ofrece tasas comprendidas entre los 6Mbps y los 56Mbps.

    3. HIPERLAN/3 o HIPERACCESS

    Utilizado principalmente para proporcionar conectividad a equipos situados a grandes distancias (hasta 5Km) que utilicen HIPERLAN/2. Alcanza los 25Mbps.

    4. HIPERLAN/4 o HIPERLINK

    Diseñado para conectar redes HIPERACCESS o puntos de acceso HIPERLAN mediante enlaces de gran velocidad pero distancias reducidas (150m). HIPERLINK opera en la banda de los 17GHz.

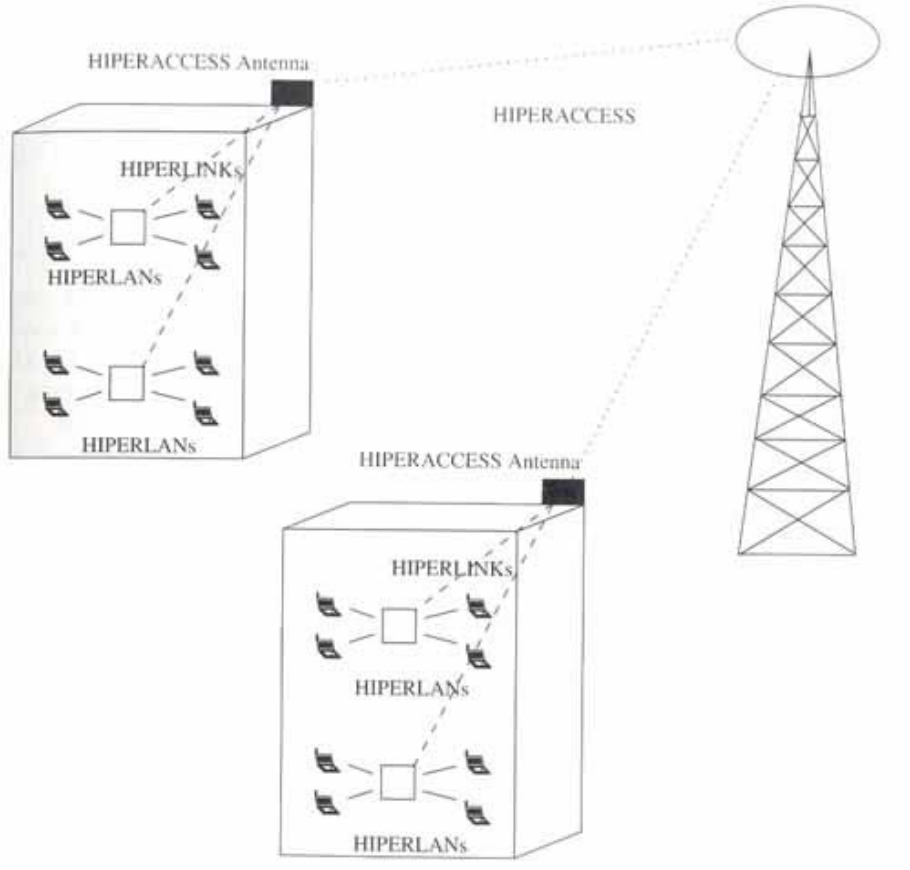

Figura 2.1: Implementación de redes HIPERLAN

Wireless WAN's y MAN's [5]

En este grupo podemos englobar tecnologías con características muy diversas:

- WAN's (*Wide Area Networks*).- Redes de área extendida inalámbricas también denominadas redes inalámbricas celulares. Se caracterizan por que dividen el área de cobertura en celdas consideradas generalmente hexagonales. Estándares tales como AMPS (*Advanced Mobile Phone System*), GSM (*Global System for Mobile Communications*), D-AMPS (*Digital-AMPS*) o UMTS (*Universal Mobile Telecommunications System*) pertenecerían a este subconjunto.
- MAN's (*Metropolitan Area Networks*).- Redes de área metropolitana. Dentro del gran abanico de tecnologías que constituyen este grupo debemos incluir los enlaces punto a punto basados en microondas y las redes inalámbricas de bucle local (*Wireless Local Loop*, WLL). Éstas últimas se caracterizan por proporcionar servicios de voz y datos al usuario final mediante enlaces inalámbricos. Destacamos dos enfoques:

    1. LMDS (*Local Multipoint Distribution Service*)

    Opera a frecuencias que rondan los 28GHz en Estados Unidos y 40GHz en Europa. Tecnología inalámbrica de banda ancha usada para proveer servicios de voz, datos, Internet y video. Celdas de 1 a 2Km que pueden proporcionar capacidades de transmisión de 155Mbps.

    2. MMDS (*Multichannel Multipoint Distribution*)

    2,5GHz en la banda con licencia y 5,8GHz en la banda ISM. Al ser menores las frecuencias utilizadas, las celdas alcanzan radios de

cobertura de 45Km, aunque la capacidad de transmisión por celda se ve reducida a 36Mbps.

2.1.4 Arquitectura

Atendiendo a la arquitectura de red, las WN's pueden clasificarse en dos tipos: redes inalámbricas centralizadas y redes inalámbricas distribuidas.

Las primeras de ellas, las redes inalámbricas centralizadas (*Centralizad Wireless Networks*), se caracterizan por la existencia de un nodo central que realiza la función de enlace entre los distintos equipos que constituyen la red. Este nodo puede encaminar la información que circula por la red bien a una red cableada convencional, bien hacia otras redes o equipos distintos. Para poder realizar la comunicación con este nodo central o punto de acceso (*Access Point*, AP), los dispositivos de la red deben estar en el radio de cobertura del AP.

Figura 2.2: Ejemplo de red inalámbrica centralizada

En cambio, las redes inalámbricas distribuidas (*Distributed Wireless Networks*) también denominadas redes ad-hoc, se caracterizan porque los dispositivos que constituyen la red pueden comunicarse entre si siempre y cuando ambos se encuentren dentro de su radio de cobertura. La comunicación entre los dispositivos es directa, sin la intervención de ningún equipo central. Veremos más adelante como las redes que centran nuestro estudio, es decir, las redes de sensores inalámbricos, presentan generalmente esta disposición.

Figura 2.3: Ejemplo de red inalámbrica distribuida

2.1.5 Control de acceso al medio [6]

2.1.5.1 Introducción

Las características especiales que presenta el medio de transmisión inalámbrico hacen que las consideraciones establecidas para el desarrollo de los protocolos de acceso al medio de redes cableadas no nos sirvan en su totalidad y debamos platearnos una serie nueva de factores para la implementación de protocolos eficientes.

Los protocolos de acceso al medio (*Medium Access Control*, MAC), estudiados extensamente desde la década de los años 70, establecen las reglas que permiten gestionar cómo y cuándo cada uno de los usuarios de la red podrá utilizar el medio común de transmisión. Si no se considerase ningún tipo de protocolo MAC o éste estuviera mal implementado en el sistema de comunicación, podrían ocurrir conflictos si más de un usuario quisiera realizar una transmisión al mismo tiempo.

La aparición de las transmisiones en medios inalámbricos y las particularidades que estos poseen ha originado la necesidad de desarrollar nuevos protocolos o adaptar los ya existentes.

Para comprender mejor la forma en que trabajan estos protocolos en primer lugar presentaremos las principales propiedades de los medios no guiados que los protocolos deberán ser capaces de solventar. A continuación mostraremos una relación de los principales parámetros utilizados para determinar el rendimiento alcanzado y por último presentaremos una breve clasificación de los protocolos MAC inalámbricos.

2.1.5.2 Propiedades de los medios inalámbricos

Como comentamos anteriormente, la existencia de una serie de propiedades características del medio inalámbrico obliga a que los protocolos MAC desarrollados sean capaces de solucionar las complicaciones que estas peculiaridades ocasionan.

Podemos destacar las siguientes:

Transmisión half-duplex

En los sistemas inalámbricos es muy complicado realizar transmisión y recepción de datos de forma simultánea. Esto es debido a que la propia señal transmitida provoca interferencias en el sistema emisor (auto-interferencias), siendo su potencia de varios órdenes de magnitud superior a la señal que esperamos recibir de otro nodo. Por tanto realizar la detección de una señal es imposible si estamos realizando una transmisión.

Esto a su vez provoca la incapacidad de detectar colisiones en el medio no guiado. Debido a esto los protocolos se centran en disminuir la probabilidad de colisión utilizando métodos de prevención de colisión denominados *Collision Avoidance* (CA).

Canal variante en el tiempo

La no utilización de elementos que guíen la señal mientras se transmite origina que los mecanismos de propagación tales como reflexiones, difracciones y dispersiones provoquen la superposición de la señal, llegando al receptor con diferentes retardos y atenuaciones (propagación multicamino). También es posible que debido a la deficiente transmisión de la señal, ésta pierda potencia produciéndose un desvanecimiento de la misma.

Por todo esto, los protocolos de medios inalámbricos suelen incorporar mecanismos que realizan un establecimiento previo de la comunicación entre nodos, de tal manera que mediante el envío de mensajes cortos se puede determinar la calidad del enlace en ese momento, antes de la realización de la transmisión.

Existencia de errores de ráfaga

Debido al factor antes expuesto y a las variaciones de potencia que experimentan las señales transmitidas en los medios inalámbricos, estos presentan mayores errores que los tradicionales medios guiados.

Mientras que la tasa de error de bit (*Bit Error Rate*, BER) es del orden de 10^{-6} en los medios guiados, en el medio inalámbrico alcanza valores de 10^{-3} o superiores. Además en los medios cableados los errores son debidos, en la mayoría de los casos, a la acción del ruido aleatorio. En cambio en los enlaces inalámbricos los errores de bits se producen en largas ráfagas debido a los desvanecimientos de la señal antes comentados.

Los protocolos suelen incorporar en sus especificaciones ciertas herramientas destinadas a solventar los problemas de errores de ráfagas de bits y pérdida de paquetes. Entre ellos podemos nombrar los siguientes:

- Utilización de paquetes de tamaño más pequeño.
- Códigos FEC.
- Métodos de retransmisión de paquetes. Una estrategia ampliamente utilizada por los diferentes protocolos MAC para medios inalámbricos es la del envío de paquetes de reconocimiento (*Acknowledgments*, ACK).

La detección de portadora depende de la posición del nodo

En la transmisión en espacio libre, la potencia de la señal decrece con el cuadrado de la distancia. Como resultado de esto, la detección de portadora por parte del elemento receptor es función de la posición relativa de éste con respecto al nodo transmisor. Además, como sabemos, sólo los nodos que estén en el radio de cobertura del transmisor podrán recibir la señal generada.

Durante la transmisión en medios inalámbricos se producen tres factores, muy importantes a la hora de ser tenidos en cuenta en la implementación de protocolos MAC: nodos ocultos, nodos expuestos y captura. Para facilitar la explicación de estos términos haremos uso de la *Figura 2.4*.

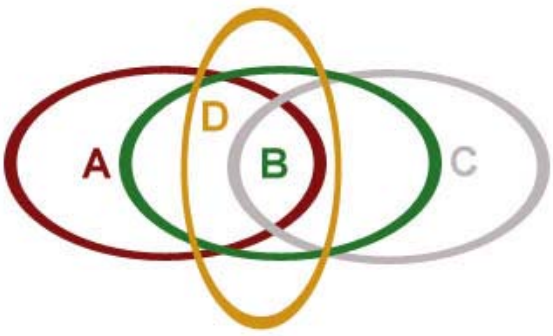

Figura 2.4: Nodos ocultos, nodos expuestos y captura

Supongamos que el nodo A, tras detectar que el medio está libre, comienza a transmitir información hacia el nodo B. Al mismo tiempo C también inicia una transmisión hacía B, produciéndose una colisión. Ni A ni C se han percatado de la

transmisión del otro nodo, pues ambos se encuentran en radios de cobertura distintos. A este fenómeno se le conoce con el nombre de Nodos Ocultos (*Hidden Nodes*).

Ahora es B quien transmite datos hacía el nodo A. El nodo C detectará que el canal está ocupado, por lo cual no realizará ningún tipo de emisión. Si C estuviera interesado en transmitir información hacía cualquier otro destino que no fuera B, no lo haría, por lo que se estaría produciendo un desaprovechamiento del canal. Este caso recibe el nombre de Nodos Expuestos (*Exposed Nodes*).

El último de los casos, el denominado Captura (*Capture*) se produciría si el nodo B pudiera recibir y decodificar la señal procedente de D aún en presencia de una señal transmitida por el nodo A, ya que ésta última tiene menos potencia que la del nodo D al estar a mayor distancia. El acceso al canal sería injusto, por tanto, para algunos de los nodos de la red.

2.1.5.3 Parámetros para medidas de rendimiento

Para determinar si el protocolo MAC desarrollado consigue solventar los problemas ocasionados por las características especiales de los medios inalámbricos, tal y como hemos visto en el apartado anterior, es necesario realizar medidas de ciertos parámetros, algunos de ellos heredados de las redes cableadas tradicionales y otros propios de las WN's.

Entre estos parámetros podemos destacar los siguientes:

- Retardo.- Tiempo transcurrido entre que el paquete accede a la capa de enlace de datos donde se ejecuta el protocolo MAC hasta que se transmite completamente. Depende del protocolo utilizado y las características del tráfico existentes en la red.
- Throughput.- Fracción de la capacidad del canal utilizada para la transmisión de datos. Uno de los principales objetivos de los protocolos MAC es maximizar este parámetro.
- Justicia.- Si no se han definido prioridades en la transmisión, un protocolo es más justo si logra que todos los nodos tengan la misma probabilidad de ocupar el canal.
- Estabilidad.- El protocolo debe procurar que el funcionamiento de la red sea lo más estable posible, tanto si existen muchas o pocas fuentes.
- Robustez ante el desvanecimiento del canal.- El desvanecimiento del canal puede provocar que el enlace entre dos nodos esté deshabilitado durante ciertos periodos de tiempo. El protocolo debe evitar que se pierda información debido a esto.
- Consumo de potencia.- Muchos dispositivos inalámbricos se caracterizan por disponer de baterías limitadas. Por tanto, los protocolos MAC deben tener este factor en cuenta a la hora de implementar su modo de funcionamiento. Este será uno de los parámetros principales utilizados para valorar el rendimiento del protocolo MAC inalámbrico que propondremos a lo largo de este documento.
- Soporte de datos multimedia.- Debido a la convergencia de las redes de voz, video y datos, los protocolos MAC han tenido que evolucionar para incorporar mecanismos que permitan un tratamiento determinado de los paquetes según la información que transporten.

2.1.5.4 Clasificación de los protocolos MAC inalámbricos

Atendiendo a la arquitectura que presente la red y según el tipo de control que sobre ella se realice, los protocolos MAC inalámbricos pueden clasificarse en dos grandes grupos: centralizados y distribuidos.

Protocolos MAC centralizados

Debido a la existencia, en este caso, de una estación base (*Base Station*, BS), todos los problemas relacionados con el acceso al medio por parte de los nodos que forman la red inalámbrica, se centralizan en la BS. Desaparece también el problema de los nodos ocultos, pues todas las comunicaciones se realizarán a través de este punto central de comunicación.

Los protocolos MAC centralizados se dividen a su vez en tres grandes grupos:

- Protocolos de acceso aleatorio.- En este tipo de protocolos los nodos compiten por obtener el canal y así poder realizar la transmisión. Si un único nodo intenta transmitir a la vez, el proceso se realizará con éxito. En cambio, si múltiples nodos intentan transmitir al mismo tiempo, el problema de las colisiones que surge se resuelve mediante el uso del algoritmo de resolución de contención (*Contention Resolution Algorithm*, CRA) que posea el protocolo en uso.

  Algunos ejemplos serian: ISMA (*Idle Sense Multiple Access*), RAP (*Randomly Addressed Polling*) y RAMA (*Resource Auction Multiple Access*).

- Protocolos de acceso garantizado.- Los nodos acceden en este caso al medio de manera ordenada. Dicho orden se establece o bien mediante el uso de una arquitectura cliente-servidor (el servidor determina mediante sondeos periódicos quién será el siguiente nodo a transmitir) o bien mediante el uso de testigos (el nodo que posea el testigo o *token* será el encargado de realizar la transmisión).

  Dentro de este grupo encontramos los siguientes protocolos: Zhang's Proposal, DTMP (*Disponsable Token MAC Protocol*) y Acampora's Proposal.

- Protocolos de acceso híbrido.- Reúnen, como su propio nombre indica, características de los dos grupos anteriores. Se suelen usar en las redes de comunicaciones celulares. A su vez se dividen en dos grupos: RRA (*Random Reservation Access*) y Demand Assignment Protocols.

Protocolos MAC distribuidos

Con la excepción de ALOHA, todos los protocolos MAC distribuidos se basan en los principios detección de portadora y prevención de colisión (CA). Surge por tanto la necesidad, por parte de estos protocolos de subsanar los problemas que ocasiona los fenómenos de Nodos Ocultos y Nodos Expuestos, antes comentados.

Para ello, los protocolos CSMA (*Collision Avoidance Multiple Access*) implantan dos mecanismos distintos de previsión de colisiones:

- Señalización fuera de banda (*Out-of-band Signaling*).- Los nodos que constituyen la red una vez han escuchado que se está realizando una transmisión, emiten una señal fuera de banda, indicando que el canal está ocupado.

- Control de establecimiento (*Control Handshaking*).- Entre nodo emisor y receptor se produce un intercambio de señales de establecimiento de conexión, RTS (*Request to Send*) y CTS (*Clear to Send*), de tal manera que cualquier otro nodo que detecte dichas señales sabe que el canal está ocupado y evita tomarlo hasta que los nodos anteriores hayan concluido.

Dentro de este grupo podemos destacar los siguientes algoritmos: DFWMAC (*Distributed Foundation Wireless MAC*) y EY-NPMA (*Elimination Yield – Non-Preemptive Priority Multiple Access*).

Como veremos más adelante, el protocolo de acceso al medio para las redes de sensores que proponemos en este documento, S-MAC (*Sensor Médium Access Control*) pertenece al segundo de estos grupos.

2.2 Redes de sensores inalámbricos

2.2.1 Introducción

Las redes de sensores inalámbricos (*Wireless Sensor Network*, WSN) pueden ser consideradas como un tipo particular de redes móviles ad-hoc (*Mobile Ad-hoc NETwork*, MANET). Estas redes constituyen un campo actual y emergente de estudio donde se combina el desarrollo de computadores, comunicaciones inalámbricas y dispositivos móviles.

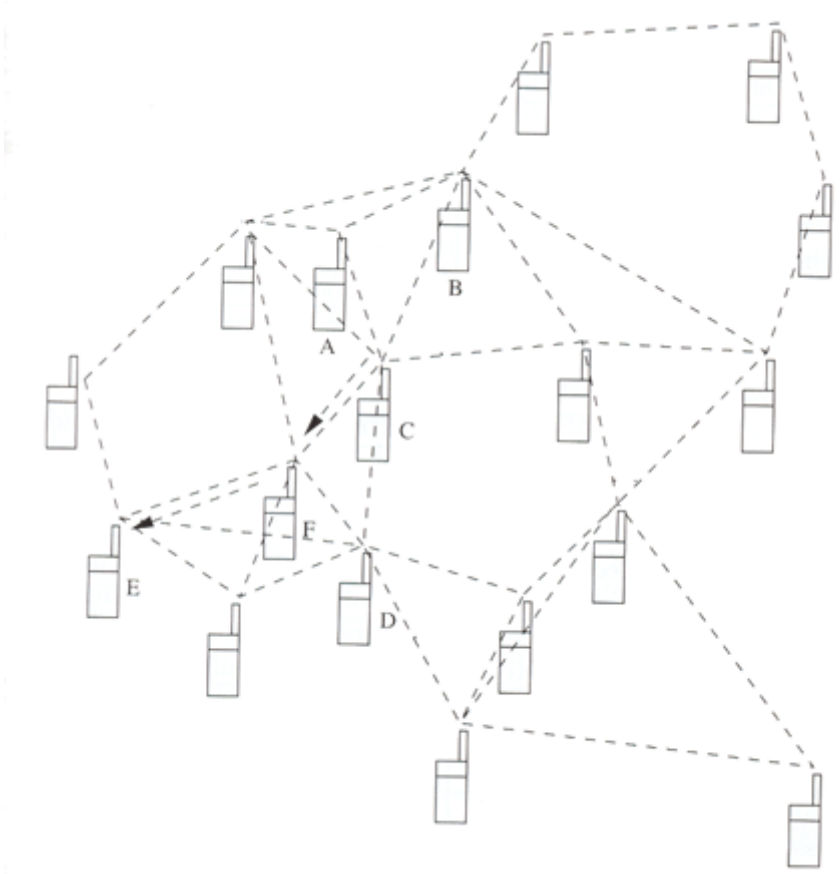

Figura 2.5: Ejemplo de la disposición de los elementos de una WSN's

Las WSN's se caracterizan por estar formadas por un conjunto de nodos inalámbricos, en general mucho más numerosos que en una red MANET, desarrollados para la realización de medidas y procesado distribuido de distintas y variadas magnitudes físicas: temperatura, movimiento, presión, ruido, luz, etc. Esto ha originado que su uso se extienda en ámbitos tan dispares como las aplicaciones militares, comerciales, médicas, agrarias, etc.

La prevención de incendios forestales es un ejemplo concreto de la aplicación de este tipo de redes, pues la distribución estratégica y distribuida de una WSN formada por un número elevado de dispositivos en un bosque puede proporcionar información exacta de donde se ha provocado un incendio de tal manera que permita su extinción antes de que se vuelva incontrolable. Los sensores red colaboran entre ellos para poder superar los obstáculos del entorno, transmitiendo los datos a través de la red distribuida hasta alcanzar el dispositivo encargado del procesamiento de la información.

Entre las características más importantes que poseen este tipo de redes, las cuales serán comentadas ampliamente en el siguiente apartado, podemos destacar que están compuestas por un elevado número de dispositivos, lo que provoca la necesidad de que sean, a su vez, autónomos y auto-organizados. Además los sensores se caracterizan por estar limitados en recursos tales como memoria, capacidad de procesado, etc.

Debido a los recientes avances tecnológicos tanto a nivel electrónico (analógico y digital) como en el diseño de dispositivos de radiofrecuencia hoy en día podemos encontrar sensores más económicos, pequeños y con más bajos niveles de consumo de potencia. Además las WSN's son capaces de agrupar cada vez a un mayor numero de nodos constituyendo así eficientes redes inalámbricas de millones de sensores que comunicándose entre sí de forma eficiente procesan grandes cantidades de información distribuida.

2.2.2 Características [7]

A pesar de derivar directamente de las WN's, las redes de sensores poseen un elevado número de características propias. A lo largo de este apartado intentaremos dar a conocer algunas de ellas, pues su consideración es fundamental para el posterior desarrollo de protocolos y aplicaciones.

Topología y mantenimiento

En general los nodos que forman las redes de sensores se caracterizan por estar aleatoriamente distribuidos sin seguir ninguna topología regular. Debido a ello es recomendable que su mantenimiento y configuración sea completamente autónomo (no requiera de la intervención humana) mediante el uso de algoritmos distribuidos.

Limitaciones energéticas

Uno de los principales cuellos de botella que encontramos en las operaciones realizadas por los sensores es, como ya hemos comentado en varias ocasiones a lo largo de este documento, la disponibilidad energética de los nodos. Los sensores en la mayoría de los casos poseen baterías que se caracterizan por no poder ser recargadas, lo cual convierte este problema en la principal restricción a la hora de desarrollar nuevos protocolos. Aumentar el tiempo de vida de un sensor implicará, por tanto, disminuir los niveles de tolerancia o limitar la precisión de los resultados obtenidos.

Hardware y software específico

El microcontrolador, el sistema operativo y las aplicaciones desarrolladas para las WSN's deben tener muy en cuenta las limitaciones energéticas antes expuestas.

Sincronización de los dispositivos

Para que el tratamiento de la información que se propaga por una red de sensores se realice de forma correcta, los nodos deben sincronizarse. Por ello en las WSN's deben imponerse procesos de acceso al medio por división múltiple en el

tiempo (*Time Division Multiple Access*, TDMA) y ordenación temporal para que la detección de los eventos se produzca sin ambigüedades.

Enrutamiento dinámico

Las redes de sensores deben ser capaces de adaptarse a los cambios de conectividad de los nodos. Por ello, los protocolos de enrutamiento utilizados deben estar preparados para incluir o excluir a nodos de sus rutas.

Restricciones temporales

Si bien las WSN's deben soportar comunicaciones en tiempo real entre los diferentes nodos, esto no debe perjudicar a las características de retardos, ancho de banda u otros parámetros de calidad de servicio de las redes (*Quality of Service*, QoS).

Seguridad

Atendiendo al uso final para el cual esté desarrollada la red de sensores, la seguridad en las comunicaciones puede ser un factor muy importante a la hora de determinar los protocolos que se desarrollaran en las diferentes capas de los nodos. Este es el claro ejemplo de las redes de sensores utilizadas en el ámbito militar.

2.2.3 Arquitectura de las WSN's [8]

Las redes de sensores se distribuyen sobre el área de trabajo tal y como se representa en la *Figura 2.6*.

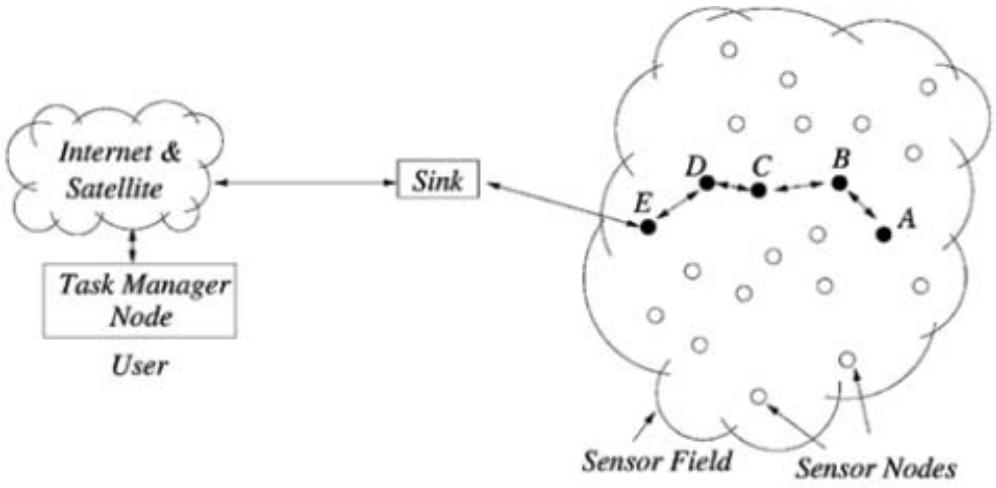

Figura 2.6: Estructura general de una red de sensores

Cada uno de los nodos de la red posee funcionalidades que le permiten la recolección de datos y su enrutamiento hasta llegar al usuario final. Hasta que la información recapitulada llega al usuario ésta debe pasar a través de una estructura distribuida de nodos ad-hoc y alcanzar al elemento *sink* o sumidero, el cual se encarga de centralizar toda esta información. Desde este punto hasta el nodo central de procesamiento, los datos pueden llegar vía Internet o conexión por satélite.

Para poder comprender la forma en que se comunican y trabajan los distintos elementos que constituyen los nodos, debemos atender al gráfico de la *Figura 2.7*. En ella vemos como internamente los sensores están constituidos por una pila de

protocolos. Dicha pila combina las características propias de las distintas capas (heredadas de las redes de datos tradicionales) con las nuevas propiedades de las redes de sensores (energía, movilidad, entorno, etc.).

En términos generales la pila de protocolos está compuesta por cinco capas: aplicación, transporte, red, enlace de datos y físico. En cada una de ellas se han desarrollado múltiples protocolos.

A continuación comentaremos brevemente cada una de estas capas. Además desarrollaremos los conceptos propios de dos protocolos, uno perteneciente a la capada red y otro a la de enlace de datos, ya que ambos han sido programados como objetivo fundamental del proyecto aquí presentado: S-MAC y DSR (*Dynamic Source Routing*).

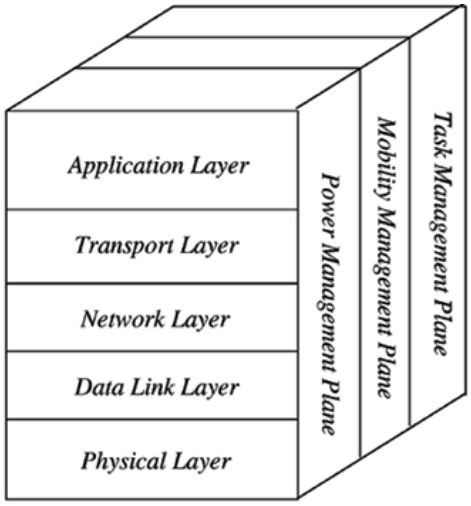

Figura 2.7: Pila de protocolos de las redes de sensores

Capa de aplicación

Dependiendo de las características de los parámetros a medir por los sensores de la red, y cual sea la funcionalidad que se les quiera dar, las aplicaciones que se pueden desarrollar para ser ejecutadas en esta capa de la pila pueden ser múltiples. Además si tenemos en cuenta que los campos de acción de las WSN's cada vez son mayores, nos encontramos ante una amplia línea de desarrollo e investigación.

Capa de transporte

En general, en las WSN's existen dos implementaciones distintas de esta capa. Cuando la comunicación se realiza entre el nodo sumidero y el nodo central de procesamiento de datos podemos encontrarnos con protocolos propios de las redes tradicionales cableadas: TCP (*Transmission Control Protocol*) y UDP (*User Datagram Protocol*). En cambio, cuando las transmisiones se realizan entre el nodo *sink* y los sensores, debido a las limitaciones existentes de memoria y energía, se han desarrollado protocolos de transporte a partir de las especificaciones de UDP.

Capa de red

En muchos casos se hace necesario enviar información desde un sensor de la red a uno o un grupo concreto de sensores. Para ello se requerirán protocolos de enrutamiento de bajo coste, de tal manera que se complementen bien con las limitaciones de los dispositivos así como con la arquitectura distribuida de la red.

Como ejemplo de aplicaciones de esta capa se ha desarrollado el protocolo DSR que será comentado detalladamente en el *Apartado 2.2.3.2* como parte de los objetivos marcados en este proyecto.

Capa de enlace de datos

La capa de enlace de datos es la responsable del control de errores, el acceso al medio, la multiplexación y la detección de tramas de datos. Además también debe de asegurar la manera fiable las conexiones punto a punto o punto a multipunto para la realización de las comunicaciones entre los dispositivos de la red.

En este caso como ejemplo de la implementación de un protocolo de enlace de datos hemos desarrollado S-MAC, que comentaremos en el *Apartado 2.2.3.1*.

Capa física

La selección de frecuencia, detección de portadora, modulación y encriptación de datos, son algunas de las funciones que debe realizar el nivel físico de los nodos de la red.

Esta capa se caracteriza por estar estrechamente unida al hardware utilizado para la implementación de las WSN's.

2.2.3.1 Protocolo S-MAC [9]

2.2.3.1.1 Introducción

Diseñar un protocolo de enlace de datos para WSN's requiere la consideración de una serie importante de atributos. En primer lugar hay que considerar la eficiencia energética del protocolo, pues el tiempo de vida de los nodos de la red es un parámetro crítico del diseño. En segundo lugar el protocolo ha de ser capaz de cubrir las expectativas de escalabilidad de la red, topología y densidad de nodos. Y por último una serie de atributos donde podemos incluir justicia, latencia, caudal y ancho de banda. Debemos destacar que mientras esta serie de factores son primordiales en las redes cableadas tradicionales, en las WSN's pasan a ser secundarios primando más la eficiencia energética.

Pues bien, S-MAC es un nuevo protocolo de enlace de datos, diseñado explícitamente para redes de sensores inalámbricos, que mientras reduce el consumo de potencia logra también buenas características de escalabilidad y prevención de colisiones mediante el uso de un esquema de contención y planificación.

Analizando el funcionamiento de una red de sensores podemos determinar las principales causas que provocan un gasto innecesario de energía. Éstas son:

- Colisiones.
- Escuchas de paquetes no dirigidos al nodo.
- Envío y recepción de paquetes de control.
- Escuchar el canal de transmisión cuando éste está vacío.

S-MAC reduce este gasto innecesario de energía aunque esto conlleve una pérdida en las características de justicia y latencia de la red. Ahora bien, la característica de justicia en una red de sensores no es de las más destacadas, ya que en general en estas redes todos los nodos de la red cooperan para alcanzar un objetivo común, por lo que si un nodo tiene la necesidad de enviar una mayor cantidad de información acaparará el canal un intervalo de tiempo mayor sin perjudicar para ello al resto de los dispositivos.

La característica de latencia de la red también se ve afectada por el intento que hace S-MAC de reducir el gasto energético. Cuando los nodos perciben que el canal está ocupado y que la información transmitida no está dirigida a ellos, pasan a un estado de sueño o *sleep* en el cual no procesan ninguna de las señales que les llegan

a su nivel físico. Por tanto, habrá que esperar a que estos nodos despierten (pasen a estado *idle*) para poder enviarles información.

2.2.3.1.2 Características de diseño

A lo largo de este apartado comentaremos los tres procedimientos que implementa S-MAC para reducir el consumo de energía. Además determinaremos una serie de características sobre las WSN's y las aplicaciones que ejecutaremos en ellas.

Aplicaciones y red

Las redes de sensores están compuestas por un elevado número de dispositivos para, de esta manera, aprovechar la ventaja de la proximidad a la magnitud a medir y simplificar el procesamiento de las señales. Además al ser más numerosos, los nodos están a distancias más pequeñas y las comunicaciones entre ellos son de corto alcance, lo que reduce el consumo de potencia.

Por otro lado, las WSN's deben estar dedicadas a la ejecución de una determinada aplicación o a colaborar junto con otras redes a la consecución de un fin común. Por ello, los criterios de justicia a la hora de usar el canal de transmisión quedan relegados a la consecución de los objetivos comunes de todos los dispositivos.

Periodos *sleep* y *listen*

En gran cantidad de redes de sensores los dispositivos están en estado activo o *idle* durante largos periodos de tiempo caracterizados por la inactividad y la no ocurrencia de eventos. Por ello S-MAC reduce los periodos *idle*, introduciendo un nuevo estado en el cual el nodo no procesa datos (periodo de *sleep*).

El esquema básico de esta sucesión de estados puede observarse en la *Figura 2.8*. Cada uno de los nodos de la red pasará al estado de *sleep* durante cierto tiempo y una vez se despierte escuchará el canal por si algún otro nodo de la red desea comunicarse con él.

Durante el periodo de sueño, los nodos detendrán la recepción de mensajes y planificarán un temporizador que les indique el momento de comenzar nuevamente la ejecución de sus funciones. Cabe destacar que la duración de los periodos del ciclo de trabajo de los sensores dependerá del escenario en el cual implantemos la WSN. Por simplicidad, se recomienda utilizar los mismos valores para todos los nodos de la red.

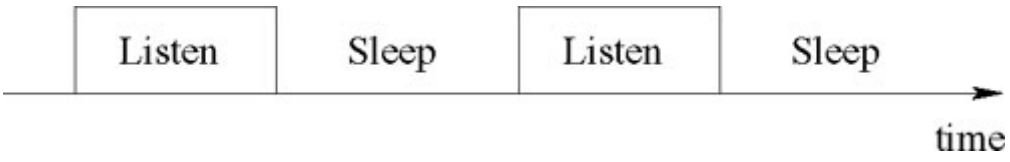

Figura 2.8: Periodos *listen* y *sleep*

Todos los nodos de la red pueden elegir libremente cuando cambiarán el valor de sus estados, aunque para reducir el control sobre ellos es preferible que se sincronicen con sus nodos vecinos. Es decir, los nodos se irán a dormir y se despertarán al mismo tiempo. Ahora bien, en una red distribuida no todos los nodos son vecinos de todos, lo cual provoca la aparición de diferentes esquemas de sincronización.

Para solucionar este problema los nodos de la red informan del esquema de sincronización que siguen mediante la transmisión de un mensaje broadcast a todos sus vecinos.

Otra característica a destacar es que los nodos de la red forman una topología plana, es decir, cada uno de los nodos puede comunicarse con cualquier otro de la red sin importar que planificaciones conozcan. De esta manera este esquema es muy eficaz para el tratamiento de topologías altamente variantes.

Ahora bien, la latencia de la red durante la transmisión de un mensaje se va incrementando a medida que se van produciendo la sucesión de periodos de sueño de los nodos. Por ello, determinar la duración de estos periodos es fundamental para que las aplicaciones ejecutadas tengan valores de latencia aceptables.

Como comentábamos anteriormente, los nodos de la red deben ser capaces de elegir un esquema de sincronización y poder intercambiarlo con sus vecinos. Para ello, los dispositivos almacenan una tabla de planificadores.

Los pasos a seguir por los nodos para la elección y mantenimiento de los esquemas de sincronización son los siguientes:

1. Cuando un nodo aparece en la red y no sigue ningún tipo de planificación, escucha el canal durante un cierto periodo de tiempo. Si durante dicho intervalo no recibe la transmisión por parte de algún vecino de una planificación, generará aleatoriamente un tiempo para pasar a estado *sleep* y lo comunicará mediante mensaje broadcast a sus vecinos.
2. Si el nodo recibe, antes de generar una planificación propia, el esquema de un nodo vecino, lo adopta como propio y lo reenvía nuevamente a la red.
3. Si el nodo recibe la planificación procedente del nodo vecino después de haber concluido el tiempo inicial de búsqueda, adoptará ambas planificaciones, es decir, se despertará tanto cuando lo determine su planificación como la del nodo vecino.

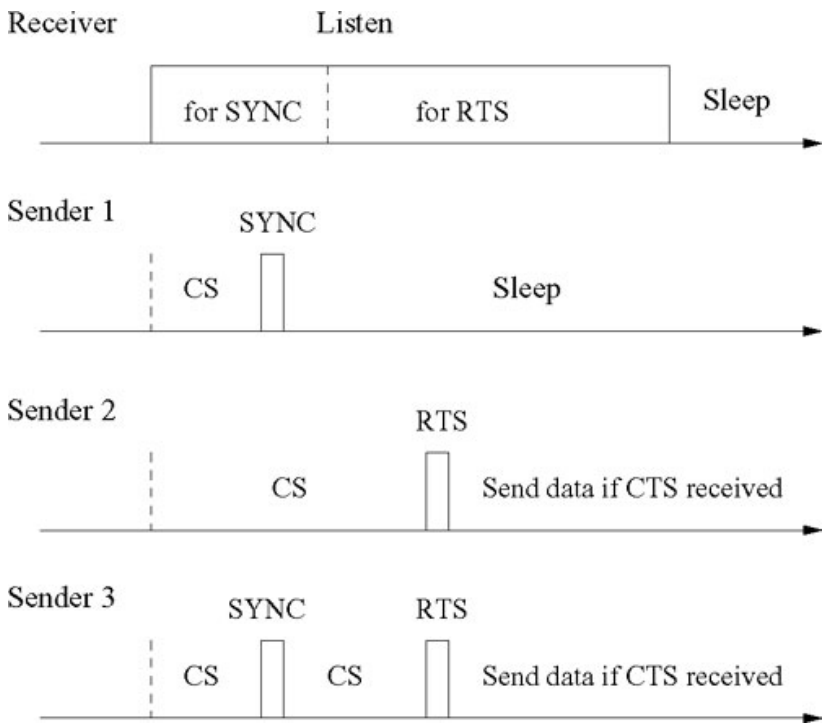

Figura 2.9: Ejemplo de sincronización entre nodos

La actualización de las planificaciones a seguir por un nodo se logra mediante el envió de paquetes de sincronismo (SYNC). Esto paquetes se caracterizan por tener una duración muy corta e incluir tanto la dirección del remitente como el instante de tiempo en el que el destino pasará a estado de *sleep*.

Observando la *Figura 2.9* vemos como el periodo de *listen* se divide en dos partes. Durante la primera de ellas los nodos reciben los paquetes SYNC y durante la segunda se inicia la transmisión de los paquetes RTS. Cabe destacar que el mecanismo de contención utilizado por S-MAC es el mismo que el del estándar IEEE 802.11, es decir, uso de los paquetes CTS y RTS.

Prevención de colisiones y escucha de paquetes no dirigidos al nodo

Como comentábamos anteriormente para prevenir las colisiones S-MAC hace uso del intercambio de mensajes RTS y CTS. Así es capaz de solucionar el problema de los nodos ocultos.

Antes de comenzar la transmisión de un paquete los nodos escuchan el canal. Si este está ocupado se van a dormir hasta que se les informa que el canal está libre. La transmisión de mensajes broadcast no requiere de mecanismos de contención, en cambio el envío de mensajes unicast sí, siendo la secuencia de transmisión la siguiente: RTS/CTS/DATA/ACK.

Cuando un nodo recibe un paquete destinado a otro, en su cabecera va almacenada la duración de dicha transmisión. El nodo, a continuación, almacenará en una variable NAV (*Network Allocation Vector*) la duración de la transmisión y no considerará el canal libre hasta que dicha variable expire.

Por otro lado, para evitar que los nodos gasten energía recibiendo mensajes que no están destinados a ellos S-MAC determina que los nodos vecinos de aquellos que si están involucrados en la transmisión (emisor y receptor) pasen a estado de *sleep* una vez hayan recibido los paquetes RTS o CTS y hayan determinado el tiempo que durará la comunicación.

Paso de mensajes

Cuando los nodos tienen que transmitir entre ellos mensajes de gran tamaño, estos se dividen en unidades de datos más pequeñas. Estos fragmentos son transmitidos en forma de ráfaga, es decir, uno tras otro, reservando el control del medio hasta que se haya realizado la transmisión de todos ellos.

Debemos destacar que el envío de los datos en forma de ráfaga implica la utilización de un único par de paquetes RTS/CTS para la contención de las colisiones. Además al fragmentar los datos en el caso de que se produjeran errores de transmisión no deberíamos realizar el envío del paquete original completo, sino simplemente del fragmento corrupto.

2.2.3.2 Protocolo DSR [10]

2.2.3.2.1 Introducción

Los protocolos de nivel de red o protocolos de encaminamiento suelen dividirse en dos grandes grupos: los protocolos dirigidos por tablas de encaminamiento y los protocolos por demanda iniciada en el origen.

Los primeros de ellos, entre los que se encuentran los protocolos AODV (*Ad Hoc On-Demand Distance Vector Routing*), DSDV (*The Destination-Sequenced Distance - Vector Routing Protocol*) y WRP (*The Wireless Routing Protocol*), tratan de mantener en cada uno de los nodos la información actualizada de distintas rutas en una o varias

tablas. Ante cambios de la topología de la red, los dispositivos responderán propagando la nueva información de ruta alcanzada.

En cambio, los protocolos por demanda iniciada en el origen crean rutas únicamente cuando un nodo desea enviar un paquete. Una vez obtenida la ruta, mediante distintos procesos, esta se mantiene en una tabla hasta que o bien el nodo destino pasa a ser inalcanzable o bien la ruta es borrada de la tabla. Ejemplos de este tipo de protocolos son TORA (*Temporary Ordered Routing Algorithm*), ABR (*Associative-Based Routing*) y DSR, protocolo que desarrollaremos a continuación.

2.2.3.2.2 Características de diseño

El protocolo DSR se caracteriza por obtener la ruta desde el nodo origen al destino mediante el uso del proceso de encaminamiento en el origen, es decir, el emisor conoce la ruta completa salto a salto a través de la red distribuida hacia el nodo destino.

Para mantener y descubrir estas rutas DSR hace uso de dos mecanismos que funcionan de forma conjunta: Descubrimiento de rutas (*Route Discovery*) y Mantenimiento de rutas (*Route Maintenance*).

Además existen dos mejoras a incorporar al protocolo que también comentaremos.

Descubrimiento de rutas

Cuando un nodo origen quiere enviar a un determinado destino un paquete, debe introducir en la cabecera del mensaje la ruta que debe seguir para que pueda llegar al destino. En dicha ruta debe reflejarse la secuencia de saltos que el paquete debe dar en su camino hacia el destino.

Si el nodo emisor consulta la lista donde almacena las rutas que conoce (caché de rutas o *Route Cache*) y encuentra una entrada que posee como receptor el nodo destino, almacenará en la cabecera dicha ruta y procederá a enviar el mensaje. Si por el contrario no existe tal entrada, comenzará la ejecución del mecanismo de Descubrimiento de rutas, para encontrar de forma dinámica un camino que enlace el nodo emisor y el nodo destino.

Para poder explicar mejor el proceso aquí expuesto haremos uso de la *Figura 2.10*, en la cual podemos observar como el nodo A intenta descubrir la ruta hasta el nodo destino E.

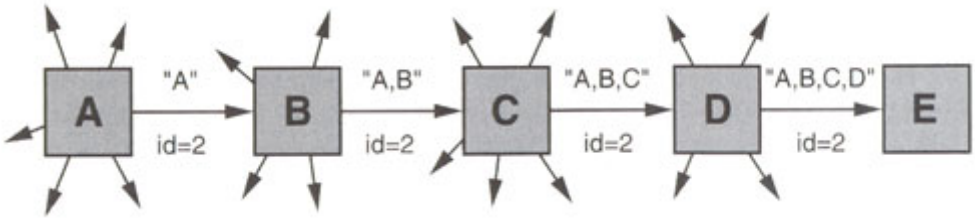

Figura 2.10: Ejemplo del proceso de Descubrimiento de rutas

El proceso de Descubrimiento de rutas se inicia en el nodo A mediante el envío de un mensaje broadcast denominado *Route Request.* Este mensaje es recibido por, prácticamente, todos los nodos que están en el rango de cobertura de A. Cada *Route Request* está compuesto por la siguiente información:

- Identificación del nodo origen y destino de la ruta.
- Identificador de petición único determinado por el nodo origen (2 en el ejemplo).

- Lista de registros con las direcciones de cada nodo intermedio a través de los cuales el *Route Request* ha ido propagándose. Inicialmente la lista sólo contiene el identificador del nodo origen (A).

Cuando el mensaje broadcast llega a otro nodo pueden ocurrir dos cosas. Si dicho nodo es el destino de la petición, devuelve al emisor un mensaje unicast de tipo *Route Reply*. En él se almacenará una copia del registro de nodos que conforman la ruta hasta alcanzarle. Cuando A reciba este mensaje, almacenará la ruta en su *Route Cache* para utilizarla más tarde cuando quiera enviar paquetes al nodo destino.

Si por el contrario el nodo B ha recibido ya un *Route Request* del nodo emisor con el mismo identificador de petición y de dirección destino, o si la dirección del propio nodo está incluida en la lista de nodos que almacena el paquete, entonces el nodo descarta la petición. En cualquier otro caso, el nodo añade su dirección a la lista y propaga el *Route Request*. Atendiendo a nuestro ejemplo, el mensaje broadcast se irá propagando desde A hasta E, nodo que iniciará la retransmisión del *Route Reply*. Para hacerle llegar a A la respuesta el nodo E utilizará el camino recién creado.

Al iniciarse el proceso de descubrimiento, el nodo emisor guarda una copia del paquete original en un buffer local denominado *Send Buffer*. El *Send Buffer* contiene una copia de cada paquete que no puede ser transmitido por el nodo emisor debido a que aún no tiene la ruta en encaminamiento hacia el nodo destino del paquete. Cada paquete en el *Send Buffer* tiene asociado un temporizador con la hora en la que fue introducido en el buffer. Si pasado un cierto tiempo, determinado como parámetro de diseño del protocolo, no se ha recibido ninguna ruta o bien se reinicia el proceso de descubrimiento un número limitado de veces o bien se descarta el mensaje.

Mantenimiento de rutas

Al encaminar un paquete utilizando una determinada ruta, cada nodo que transmite el paquete es responsable de la confirmación de que el mensaje ha sido recibido por el siguiente salto.

En el ejemplo que mostramos en la *Figura 2.11*, el nodo A desea enviar un paquete al nodo E utilizando ruta calculada en el paso anterior.

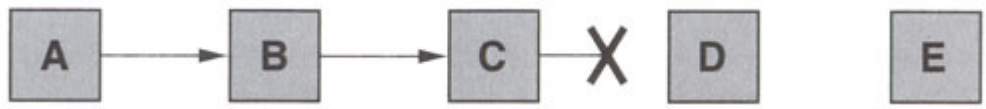

Figura 2.11: Ejemplo del proceso de Mantenimiento de rutas

Observando la gráfica vemos como el nodo A es responsable de la recepción del paquete en el nodo B, el nodo B es responsable de la entrega del paquete en el nodo C y así sucesivamente.

Si pasado un cierto tiempo no se recibe confirmación de que el paquete ha sido recibido por el siguiente nodo de la ruta, tras un número máximo de intentos, el nodo que es responsable de la entrega del paquete en el siguiente nodo debe devolver un mensaje de tipo *Route Error* al nodo emisor del paquete indicando el enlace por el cual no ha podido ser encaminado el mensaje. En nuestro ejemplo el nodo C enviará un mensaje de error al nodo B, el cual lo retransmitirá al nodo origen A. Una vez recibido el *Route Error* por parte de A, éste eliminará de su caché de rutas el enlace para ese nodo.

En cambio, si la recepción del mensaje se realiza de forma correcta, el nodo destino enviará un mensaje unicast de tipo *Route Ack*, que se irá transmitiendo hasta llegar al nodo fuente.

Actualización de *Route Cache* en nodos intermedios

A medida que los nodos encaminan la información de enrutamiento hacia los nodos destino y/o fuente, pueden incorporar esa información a sus propias *Route Cache* de cara a poder utilizarla en futuros envíos.

Respuestas a los *Route Request*

Para minimizar el tiempo que tarda un nodo destino en descubrir la ruta hacia el origen cuando un nodo recibe un *Route Request* que no está dirigido a él, busca en su *Route Cache* la existencia de ruta para el nodo destino de la petición. Si la encuentra, el nodo devuelve un *Route Reply* al emisor en vez de continuar la propagación de la petición. En el mensaje *Route Reply*, el nodo completa la ruta, concatenando la lista de nodos que almacenaba el mensaje con la que tenía en su caché.

Sin embargo, antes de transmitir el paquete *Route Reply*, el nodo debe verificar que la lista resultante a enviar en la *Route Reply* no contenga nodos duplicados o el número de saltos que forman la ruta sea superior al establecido como longitud máxima de ruta permitida (según las especificaciones del protocolo 10 saltos).

Capítulo 3
Entornos de simulación de redes de sensores

3.1 Introducción

Para realizar el diseño y desarrollo del simulador de redes de sensores que se presenta en este proyecto ha sido necesaria la utilización de un entorno de simulación que reuniera características tales que nos permitieran alcanzar los objetivos marcados.

Debido al continuo desarrollo de las redes de sensores, el número de entornos que podemos encontrar, tanto no comerciales como comerciales, es cada vez más numeroso. Por ello, en este capitulo intentaremos mostrar una visión general de algunos de ellos, así como determinar las características que han provocado la elección de OMNeT++ como entorno de simulación utilizado para la implementación del simulador de redes de sensores inalámbricos desarrollado.

A continuación, por tanto, estudiaremos qué necesitamos para la realización de simulaciones y cuáles son las herramientas que podemos encontrar para realizar dichas actividades.

3.2 ¿Por qué usar simuladores?

La implantación de las infraestructuras de una red de sensores, debido al elevado número de nodos que la constituyen, implica un gran esfuerzo y coste. Por ello, la realización de estudios previos que comprueben el correcto funcionamiento de los protocolos implementados en la red es un paso fundamental y necesario.

Por otro lado, las investigaciones realizadas sobre las WSN's han determinado la necesidad de estudiar el comportamiento de este tipo de red y de los protocolos implementados en ella frente la existencia de dos factores específicos:

- La operabilidad de los protocolos ejecutados en las distintas capas de los nodos de la red está determinada generalmente por las variables físicas medidas por los sensores. Es decir, dependiendo del objetivo para el cual haya sido desarrollada la red de sensores, la variabilidad de los parámetros del medio a medir determinará tanto el tráfico como la topología de la red.
- El control del gasto energético de la red. En la mayoría de los casos los nodos que constituyen las WSN's funcionan con baterías no recargables. Por ello, limitar el uso innecesario de energía por parte del nodo durante su ciclo de trabajo es un proceso fundamental, que debe ser considerado en los estudios previos a la implantación de la red.

Debido, por tanto, a estos factores, la realización de análisis teóricos de una WSN es un proceso muy complicado y se suele recurrir a modelos simplificados de la red, lo que da lugar a resultados poco fiables y limitados.

Vemos así que la realización de simulaciones previas es esencial antes de la implantación de las WSN's, sobre todo si ello conlleva nuevos protocolos y

funcionalidades de red. Este hecho ha provocado un auge en las herramientas de simulación disponibles. Sin embargo, la obtención de resultados fiables y representativos mediante simulaciones no es una tarea sencilla. Hay dos aspectos claves que debemos evaluar antes de la realización de las simulaciones pertinentes:

- La utilización del modelo correcto.- La simulación de dispositivos hardware idealizados, protocolos y canales de transmisión poco realistas pueden dar lugar a resultados equivocados. Un buen modelo de simulación basado en supuestos sólidos originará resultados bastante fiables. Pero mantener un nivel de detalle elevado de las características de la red puede originar la necesidad de un nivel de procesamiento demasiado alto. Por ello se debe establecer la necesidad de un compromiso: Precisión contra rendimiento y escalabilidad.
- La elección de la mejor herramienta para el modelo en cuestión.- El esfuerzo de ejecutar un modelo de simulación completo es considerable. Se hace fundamental, por tanto, el uso de una herramienta que nos facilite este trabajo. Sin embargo cada una de estas herramientas presenta distintas características que las hacen adecuadas para un tipo de simulación u otro. Algunas de ellas están desarrolladas para la obtención de resultados más precisos, mientras que otras se caracterizan por presentar un interfaz más simple para el usuario. Por esto es fundamental conocer que es lo que nos puede ofrecer cada una de las herramientas y como éstas pueden adaptarse a los requerimientos del modelo sometido a estudio.

3.3 Modelo básico para los simuladores de WSN's [11]

Una de las innovaciones que añaden los simuladores de WSN's es la inclusión en los modelos de nuevos componentes característicos de las redes de sensores, como pueden ser los referentes al consumo de energía y potencia o modelos que determinan de manera exacta las condiciones ambientales o parámetros de entorno de la red.

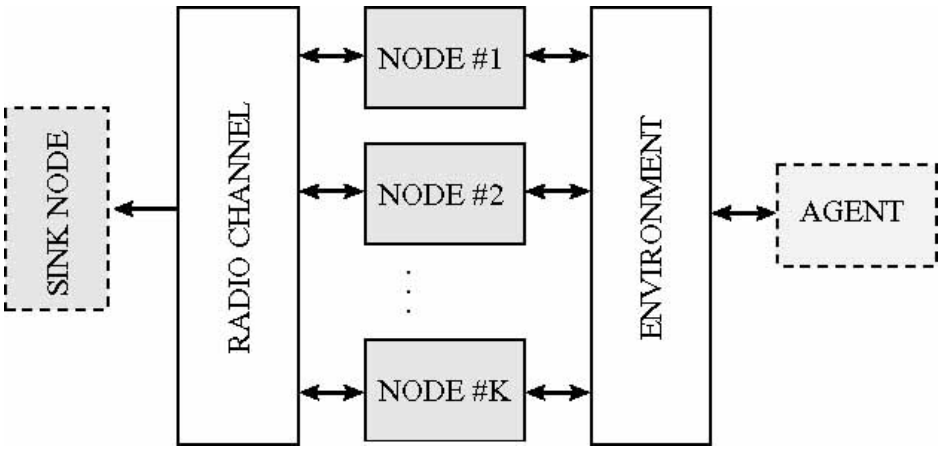

Figura 3.1: Modelo de red para las WSN's

En la *Figura 3.1*, podemos observar los componentes básicos que constituyen el modelo de una red de sensores inalámbricos. Dichos componentes serán descritos brevemente a continuación:

Nodos

Dispositivo encargado de supervisar una o varias variables físicas. La información recapitulada por el nodo se transmite hacia los demás elementos de la red a través del canal inalámbrico común, haciendo uso de una pila de protocolos encargados de las comunicaciones.

Para poder explicar el funcionamiento interno de un nodo, es necesario saber como se produce el intercambio de información entre los elementos que lo constituyen. Para ello haremos uso del esquema de módulos de la *Figura 3.2*.

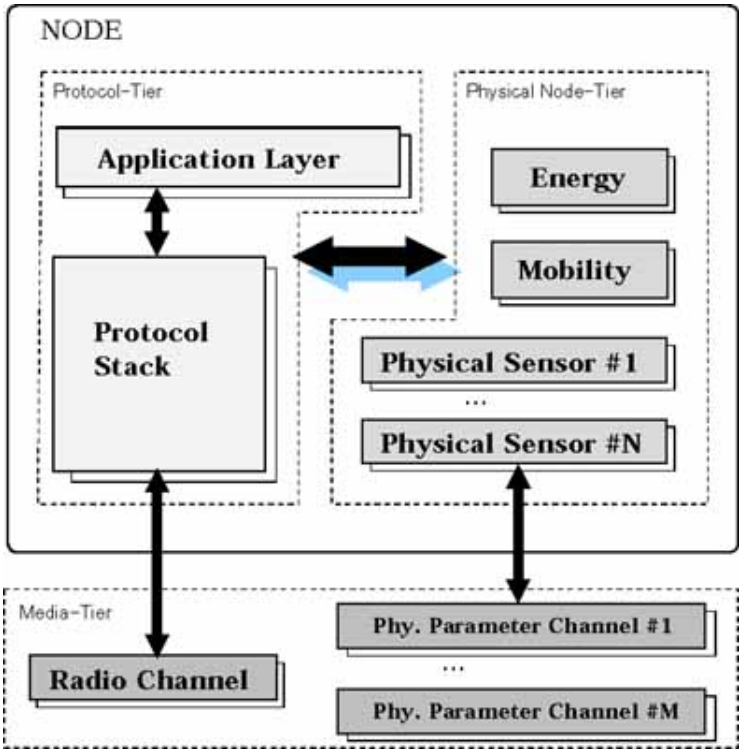

Figura 3.2: Modelo de los nodos de las WSN's

- Protocol-tier.- Submódulo que abarca todos los protocolos de comunicación implementados en el nodo. Generalmente este módulo está a su vez constituido por tres capas: capa de control de acceso al medio, capa de red o encaminamiento y capa de aplicación. Las funcionalidades propias de este submódulo están estrechamente vinculadas con los datos y estados determinados por el módulo físico del nodo, el cual explicaremos a continuación. Por ello es fundamental para el correcto funcionamiento de la red el establecimiento de un método eficiente de intercambio de información entre estos módulos.
- Physical Node-tier.- La composición que presenta este submódulo varia dependiendo de las aplicaciones específicas para las cuales este desarrollado el nodo. Suele incluir el sensor físico así como el módulo de energía, encargado de simular el consumo de potencia del hardware, y el módulo de movilidad, el cual controla la posición del nodo.
- Media-tier.- Este submódulo relaciona a los nodos de la red simulada con las características que poseería un entorno real. La conexión puede realizarse o bien a través de un canal de radio o bien a través de uno o varios canales físicos.

Puede añadirse al nodo un componente adicional, un módulo de topología, que se encargue de controlar cual es la posición del nodo sobre la superficie de trabajo.

Entorno

Una de las principales diferencias que encontramos entre las WSN's y los modelos tradicionales de redes es la influencia que sobre las simulaciones realizadas tiene el entorno. Por ello, en el modelo de simulación se incluye un módulo encargado de modelar la generación y propagación de las magnitudes físicas a medir por los sensores que forman la red inalámbrica, así como la comunicación entre los nodos.

Canal de transmisión inalámbrico

Como vimos en los apartados anteriores, el tipo de medio utilizado en las transmisiones va a condicionar la forma en que se realiza la propagación de las señales entre los elementos de la red. Por ello en algunos modelos muy detallados se añaden las características del terreno sobre el cual se van a realizar las simulaciones. De esta manera se pretende obtener resultados más fiables y reales.

Nodos sumideros (*Sink Nodes*)

Nodos especiales, encargados de recibir y procesar los datos que se van propagando por la red. Pueden también interrogar a los distintos sensores y solicitar información específica.

Agentes (*Agents*)

Generador de eventos. El agente puede provocar la variación de la magnitud física que se propaga a través de la red y que debe ser medida por los sensores.

3.4 Características de los simuladores [11]

Debido a la variedad de entornos y medios donde se pueden establecer las WSN's, la gran cantidad de protocolos que podemos encontrar para cada una de las capas de las que están formadas los nodos y los distintos módulos que pueden formar parte de una red de sensores, el marco que deben abarcar los simuladores de redes de sensores inalámbricos es muy amplio.

Por todo ello es necesario determinar cuales son las principales propiedades que los entornos de simulación de las WSN's deben poseer, independientemente de los parámetros que configuremos. Destacamos las siguientes características:

Disponibilidad y reutilización

Simular redes de sensores en proceso de desarrollo implica la necesidad de que las herramientas utilizadas ofrezcan la posibilidad de modificar o incluir nuevos modelos de simulación. Un diseño cuidadoso del entorno con interfaces claros y un alto grado de modularidad ofrecerá grandes facilidades a la hora de añadir y/o modificar parámetros o funcionalidades del simulador en cuestión. Esta propiedad es interesante cuando las características del entorno de simulación permiten que éste sea utilizado para la realización de múltiples simulaciones y por distintos investigadores. Cuando nos referimos a herramientas específicas desarrolladas para un tipo concreto de WSN's, solamente algunos de los módulos mantendrán su generalidad (nivel físico, canal de comunicaciones, etc.) mientras que el resto deberá ser desarrollado íntegramente.

Además también es conveniente que estos entornos incluyan modelos comunes ya desarrollados y depurados, con la finalidad de usarlos como base para pruebas de nuevas simulaciones. En general los simuladores suelen incluir la pila de protocolos TCP/IP (*Transmission Control Protocol/Internet Protocol*), considerada como soporte

estándar mínimo, así como modelos de movilidad, enrutamiento y propagación propios de las WSN's.

Rendimiento y escalabilidad

Ambos parámetros constituyen una de las mayores preocupaciones a la hora de ejecutar simulaciones con una herramienta determinada.

Mientras que el rendimiento del simulador está estrechamente ligado a la memoria, al procesador y a la capacidad de almacenamiento, las necesidades de escalabilidad nos determinaran la eficacia del lenguaje de programación utilizado.

La realización de simulaciones con cientos de miles de nodos sigue siendo hoy en día un problema, pues los tiempos de simulación necesarios aumentan de forma considerable al tener que añadir la interacción del entorno sobre las comunicaciones que se producen en los medios inalámbricos, tal y como comentamos en el apartado anterior, así como las consideraciones de consumo de energía, movilidad y propagación.

Soporte para la definición de experimentos y resultados mediante el uso de scripts de simulación

En general, son muchas las variables que se deben obtener en la simulación de una WSN's. El simulador utilizado para tal efecto debe ser capaz de proporcionarnos información sobre el número de nodos de la red, topología, movilidad de los sensores, gasto energético durante la simulación, parámetros del entorno, magnitud física medida, características del canal, etc.

Por ello es necesario que el entorno sea capaz de facilitar al usuario scripts de simulación que permitan, no solo la introducción de múltiples valores para diferentes parámetros, sino también la obtención de estadísticas y resultados de múltiples simulaciones.

Soporte gráfico, de depuración y ejecución paso a paso

Que el simulador utilizado proporcione un soporte gráfico, es siempre una ayuda fundamental a la hora de encontrar y eliminar errores. Una rápida visualización puede proporcionarnos información, por ejemplo, sobre el estado de las variables de la simulación, la interacción entre los diferentes módulos o los eventos futuros de la ejecución.

Además la existencia de herramientas de composición de módulos nos proporciona una representación visual de los elementos de la red, muy útil en simulaciones con pocos nodos. Sin embargo, para simulaciones de mayor escala, esta funcionalidad no resulta muy práctica.

Por último, la posibilidad de observar los resultados obtenidos mediante plotters que permitan la generación de gráficas proporciona también un valor añadido al simulador.

3.5 Clasificación de los simuladores de WSN's

Una vez expuestas las principales características que debemos buscar en los entornos de simulación de WSN's, expondremos a continuación una clasificación de los mismos. Dentro de cada una de las categorías presentaremos algunos de ellos, centrándonos en los no comerciales o de código abierto.

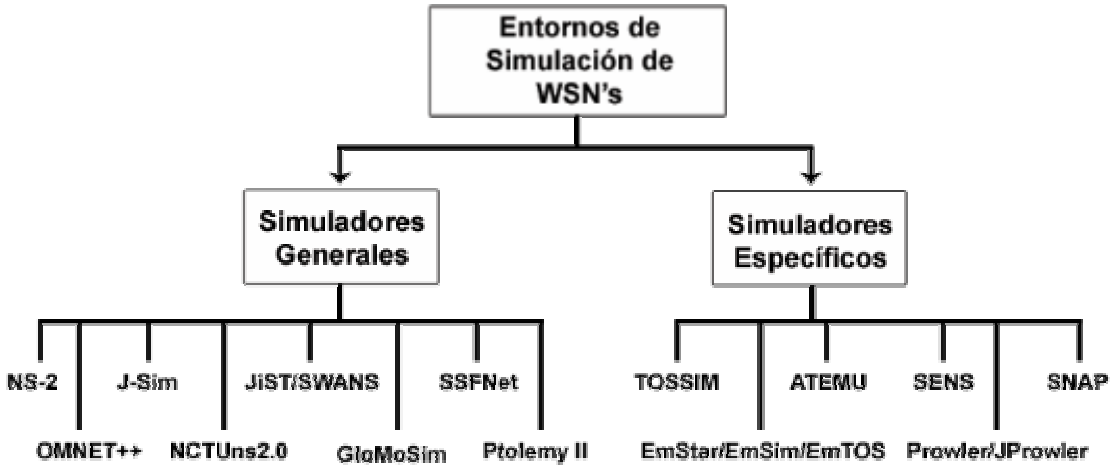

Figura 3.3: Entornos de simulación para WSN's

A grandes rasgos, podemos hablar de la existencia de dos tipos de entornos de simulación de redes de sensores inalámbricos: los entornos generales y los específicos.

3.5.1 Simuladores generales

Dentro de este apartado hablaremos, principalmente, de ocho herramientas de simulación destacando entre ellas a OMNeT++, la cual, como ya comentamos en el *Capitulo 1*, ha sido la seleccionada para la implementación del simulador desarrollado como objetivo de este proyecto. Por ello, comentaremos brevemente las características principales del resto de simuladores para después desarrollar OMNeT++ de forma más detallada en el *apartado 3.5.1.1*.

NS-2 [12]

Simulador de eventos discretos desarrollado en C++. Se caracteriza por tener soporte para gran cantidad de protocolos en todas las capas. Utiliza OTcl [13] como lenguaje de script. Además proporciona el soporte más amplio de modelos de comunicación, incluyendo protocolos propios de las redes ad-hoc y específicos de las WSN's, tales como Directed Diffusion [14] o SMAC [9].

NS-2 ve ampliada sus capacidades mediante la incorporación de proyectos tales como SensorSim [15] y NRL [16], los cuales permiten modelar distintos tipos de WSN's.

Con NS-2 podemos realizar simulaciones de hasta 1.000 nodos, realizando para ello una serie de pasos de optimización previos. En contrapunto a estas ventajas nos encontramos con un soporte gráfico, basado en Nam, de muy poca calidad.

De los simuladores presentados en esta categoría, NS-2 es el más popular de todos. Ha sido la herramienta de test esencial para las investigaciones sobre redes realizadas y se podría esperar que a medida que surjan nuevos protocolos de WSN's estos sean agregados. Sin embargo, la aparición de herramientas específicas, las cuales comentaremos en el siguiente apartado, desarrolladas para estos tipos determinados de protocolos y redes de sensores ha frenado la evolución esta herramienta.

J-Sim [17]

Entorno de simulación en Java basado en el diseño por componentes software. Entre las características de J-Sim debemos destacar la inclusión de soporte para una amplia lista de protocolos, conteniendo modelos detallados de WSN's que incorporan algoritmos de difusión y enrutamiento de datos. Estos modelos, a su vez, destacan por ser reutilizables y fácilmente modificables, ofreciendo así J-Sim máxima flexibilidad.

J-Sim incorpora bibliotecas GUI (*Graphical User Interface*) que proporcionan soporte para depuración y control de errores así como animaciones gráficas. Como lenguaje de script, la herramienta utiliza Jacl [18].

Comparando J-Sim con NS-2, vemos que ambas herramientas son capaces de soportar simulaciones con un número de nodos muy parecido. Ahora bien, mientras que J-Sim consume menos memoria que NS-2, este último proporciona unos tiempos de ejecución mucho mejores (de hasta el 41%).

NCTUns2.0 [19]

Simulador de eventos discretos integrado en el kernel de UNIX. Los paquetes de la capa de red son enrutados a través de interfaces virtuales que simulan capas inferiores de los nodos y dispositivos físicos. Esta característica permite la realización de simulaciones con datos procedentes de aplicaciones reales, incluso de sensores instalados para tal efecto. Como contrapunto, la incorporación de nuevos módulos para la simulación de WSN's no es una tarea sencilla.

Esta herramienta dispone también de la biblioteca GUI así como un gran número de protocolos y dispositivos de red pertenecientes a WLAN's. Por el contrario NCTUns2.0 no incorpora diseños específicos para WSN's.

Por último destacar las características del soporte gráfico que añade esta herramienta.

JiST/SWANS [20]

Simulador de eventos discretos desarrollado en Java. Los diferentes modelos a simular son implementados y compilados en Java, introduciendo las características propias del modelo y mediante el uso de una máquina virtual de Java (*Java Virtual Machine*, JVM).

A pesar de que esta herramienta nos permite la utilización de software existente desarrollado en lenguaje Java, una de sus principales desventajas es la carencia de modelos de simulación y protocolos incorporados. Actualmente sólo proporciona un simulador de redes ad-hoc denominado SWANS.

Utilizando Jython [21] como lenguaje de scripts es capaz de realizar simulaciones de hasta 10^6 nodos obteniendo tiempos de ejecución de hasta dos órdenes de magnitud mejores que NS-2 o GloMoSim, simulador que desarrollaremos a continuación. También supera las características de consumo de memoria de ambos simuladores.

GloMoSim [22]

Entorno de simulación basado en Parsec, lenguaje de programación derivado de C al que se le ha añadido la semántica necesaria para creación de entidades de simulación y comunicación mediante mensajes en una gran variedad de arquitecturas paralelas.

Mediante el uso de este paralelismo, se han logrado realizar simulaciones mediante GloMoSim de hasta 10.000 nodos.

SSFNet [23]

Conjunto de modelos de red implementados en Java construidos sobre SSF (*Scalable Simulation Framework*). SSF es un estándar de simulación en paralelo por eventos discretos desarrollado tanto en Java como en C++.

Ptolemy II [24]

Permite la utilización simultánea de diferentes modelos de computación: tiempo continuo, flujo de datos, eventos discretos, etc. También trata de modelar, diseñar y simular sistemas concurrentes en tiempo real.

Los modelos desarrollados bajo esta herramienta pueden construirse a partir de clases ya existentes o mediante la combinación de ellas. Además el soporte gráfico de esta herramienta asegura una forma sencilla e intuitiva a la hora de componer los modelos y representar los resultados obtenidos.

OMNeT++ [1]

Entorno de simulación de eventos discretos, modular y orientado a objetos, caracterizado por poseer una interfaz gráfica muy potente. Al pertenecer a la categoría de simuladores generales, mediante OMNeT++ no podemos simplemente realizar simulaciones de redes de comunicaciones, sino que debido a la arquitectura genérica y flexible que posee, su campo de acción abarca áreas de investigación tales como: modelación de tráfico de redes de comunicaciones, colas de dispositivos de red, protocolos y multiprocesadores y sistemas hardware distribuidos, validación de arquitecturas hardware, evaluación de aspectos de rendimiento en sistemas software, etc.

Si bien una de las principales desventajas de OMNeT++ era la carencia de protocolos disponibles en las bibliotecas del entorno, el hecho de que se esté convirtiendo en una herramienta popular está provocando la continua aportación de nuevos modelos por parte de los usuarios del entorno. Así podemos encontrar como recientemente se han incorporado características de movilidad, localización de nodos y protocolos MAC específicos de WSN's.

Ahora bien, debido a que estas incorporaciones han sido desarrolladas por grupos de investigación independientes, no comparten un interfaz común, lo que produce grandes dificultades a la hora de combinarlas.

3.5.2 Simuladores específicos

A lo largo de esta sección describiremos algunos de los entornos específicos de simulación para redes de sensores más relevantes. Estos simuladores están preparados para emular y simular el software y el hardware de determinadas WSN's.

En estos tipos de simuladores nos enfrentamos a un nuevo compromiso: pérdida de generalidad en el desarrollo de los modelos frente a la exactitud y realismo de los resultados obtenidos.

Los modelos desarrollados para las simulaciones están fuertemente ligados a las aplicaciones para las cuales se usarán. Además en las diferentes capas de los nodos, los protocolos implementados se acercan a las realizaciones reales finales que se ejecutarán en los sensores. Esto facilita la detección de errores que posteriormente podrían surgir al realizar la implementación física de la WSN una vez concluidas las simulaciones pertinentes. En contraposición, los simuladores que pertenecen a este grupo están estrechamente ligados a un determinado hardware o software (generalmente MICA Motes [25]).

TOSSIM [26]

Simulador de eventos discretos a nivel de bit y emulador de TinyOS [27]. TOSSIM simula la ejecución de código nesC en dispositivos MICA que operan bajo sistema operativo TinyOS. También proporciona un modelo para el canal radio.

Tras compilar los elementos hardware emulados junto con los componentes reales del TinyOS mediante el uso del compilador nesC, obtenemos un ejecutable que incorpora tanto los datos del sistema operativo como los correspondientes a la capa física simulada. Si a esto le añadimos la posibilidad que ofrece TOSSIM de introducir datos desde fuentes externas el resultado que nos proporciona este simulador es una herramienta perfecta para el análisis de redes formadas por nodos TinyOS/MICA.

Como limitación podemos destacar que debido a que esta herramienta está orientada más al estudio del sistema operativo TinyOS y a las aplicaciones que derivan de él no es posible mediante ella obtener información del consumo de energía de los nodos. Además, todos los nodos de la red deberán ejecutar los mismos protocolos, lo que limita el uso del simulador para el estudio de redes heterogéneas.

Otra cuestión importante a comentar es que en este entorno, a medida que el tráfico de la red aumenta, el rendimiento del simulador se degrada de forma considerable. Esto se debe principalmente, como antes comentamos, a que TOSSIM está orientado a nivel de bit. Aún así, puede ejecutar simulaciones de alrededor de miles de nodos.

EmStar/EmSim/EmTOS [28]

EmStar es un entorno de programación de software orientado al desarrollo de aplicaciones para unas plataformas especiales denominadas microservers (sistemas ad-hoc caracterizados por poseer hardware con mejores características que los sensores convencionales). Este entorno incluye una extensión de un microkernel de Linux así como librerías y herramientas. De estas últimas podemos destacar EmSim (simulador de microservers) y EmCee.

Además se ha desarrollado EmTOS que consiste en una extensión de EmStar que permite la ejecución de aplicaciones nesC/TinyOS en EmStar.

ATEMU [29]

Emulador del procesador AVR que se utiliza en la plataforma MICA. Se caracteriza por que mientras que el entorno simula el canal, las operaciones a realizar por los nodos son emuladas instrucción por instrucción.

ATEMU posee también una biblioteca con otros dispositivos hardware para las simulaciones como por ejemplo temporizadores, transceptores, etc. De esta manera esta herramienta proporciona la posibilidad de realizar simulaciones usando otros sistemas operativos y aplicaciones (distintas de TinyOS) y de construir para ellos redes heterogéneas con distintos tipos de sensores.

SENS [30]

Simulador de eventos discretos implementado en C++. SENS utiliza un modelo de sensor simplificado, constituido únicamente por tres niveles (capa de aplicación, capa de red y capa física) más una capa adicional formada por las características del entorno y el canal de transmisión. Permite la utilización directa de código nesC.

Prowler/JProwler [31]

Simulador de eventos discretos que funciona debajo de MATLAB. Se desarrollo con la finalidad de optimizar los parámetros de red. La versión implementada en Java recibe el nombre de JProwler.

SNAP [32]

Plataforma integrada para simulación y despliegue de hardware. Mediante matrices de SNAP's (denominadas *Network on a Chip*) se han conseguido simular redes de 100.000 nodos.

3.6 ¿Por qué OMNeT++?

OMNeT++ [1] (*Objective Modular Network Testbed in C++*) ha sido elegida entre todas las herramientas antes expuestas como entorno de desarrollo del simulador de WSN, objetivo de este proyecto, por las siguientes propiedades:

- OMNeT++ se caracteriza por ser una herramienta fácil de utilizar y por poseer un diseño en componentes extensible.
- Como comentaremos más adelante, las funcionalidades propias de los módulos que constituirán el simulador están desarrolladas en lenguaje de programación C++. El uso de OMNeT++ nos permite el uso de este lenguaje y de esta forma aprovechar toda su potencia y rendimiento.
- A pesar de estar interesados en el desarrollo de un simulador de redes de sensores inalámbricos, buscamos que éste sea lo más general posible y que su uso pueda expandirse a otros estándares de WN, como por ejemplo IEEE 802.11. OMNeT++ nos proporciona esta generalidad.
- Debido a la necesidad de usar el simulador como herramienta docente era necesaria la utilización de una interfaz gráfica potente.

Además, OMNeT++ facilita el desarrollo de modelos de simulación, pues evita que el programador tenga que preocuparse de las acciones relacionadas directamente con la simulación por eventos discretos. El programador únicamente debe implementar el comportamiento de las entidades del modelo, los módulos, las cuales se programan en C++, y definir la topología de la red mediante el uso del lenguaje de alto nivel propio de OMNeT++, NED. De la planificación de eventos, del paso de mensajes entre los módulos, de la actualización del tiempo de simulación, etc., se ocupan las librerías. OMNeT++ también proporciona clases que representan colas, vectores, paquetes o histogramas, entre otras.

En los apartados siguientes expondremos las principales características de OMNeT++.

3.7 Modelado en OMNeT++

En los siguientes apartados del documento mostraremos una visión global del entorno de simulación OMNeT++, destacando las características más importantes de la herramienta.

3.7.1.1 Componentes

De los elementos que constituyen el entorno de simulación aquí desarrollado podemos numerar los siguientes:

- Librerías de componentes utilizados para las simulaciones.
- Compilador para el lenguaje NED.
- Librería GUI para la ejecución de las simulaciones.

- Ejecución de las simulaciones mediante línea de comandos.
- Herramientas para la visualización de vectores gráficos (*Plove*).
- Utilidades varias, entre las que podemos destacar generadores de semillas de números aleatorios.
- Documentación y material de ayuda.

3.7.1.2 Conceptos básicos

Son varios los elementos que OMNeT++ proporciona al usuario para que describa la estructura del sistema que desea simular. A continuación los iremos desarrollando.

Módulos jerárquicos

Un modelo o red de simulación en OMNeT++ consiste en un conjunto de entidades que realizan una actividad y se comunican entre sí. Cada una de estas entidades recibe el nombre de módulo. Estos módulos pueden estar formados a su vez por la unión de otros, constituyendo entre todos ellos una jerarquía anidada de nodos, en la cual no existe límite de profundidad, lo que puede dar lugar a estructuras bastante complejas.

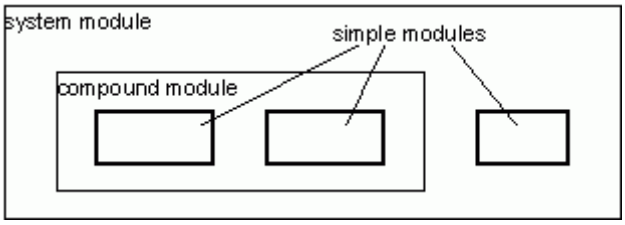

Figura 3.4: Ejemplo de módulo en OMNeT++

Atendiendo a la figura anterior vemos como un modelo está constituido inicialmente por un módulo de sistema (*system module*). Éste a su vez está formado por una serie de submódulos, los cuales pueden ser simples (*simple module*) o compuestos (*compound module*).

El usuario implementará los módulos simples usando el lenguaje de programación C++, así como las distintas clases desarrolladas en las librerías que proporciona el entorno.

La característica de OMNeT++ de ser un entorno orientado a objetos nos permite, una vez definido un módulo, usar varias instancias de él en la construcción del modelo sin tener que programar tantas clases como módulos tenga nuestro simulador.

Mensajes

Los módulos que componen el modelo a simular se comunican mediante mensajes, los cuales pueden representar tramas, paquetes de red, trabajos, clientes o cualquier otra entidad móvil del sistema.

Los mensajes pueden contener estructuras de datos bastante complejas, incluso incluir en su interior otros tipos de mensajes.

OMNeT++ considera los mensajes entre módulos eventos, por ello el tiempo de simulación avanza cada vez que se recibe un mensaje, siempre y cuando la recepción de dicho mensaje avance en el tiempo, es decir, los instantes de generación y recepción sean distintos. Para la implementación de temporizadores los mensajes pueden ser enviados y recibidos por el mismo módulo.

Puertas y enlaces

Para enviarse mensajes entre ellos, los módulos hacen uso de las puertas y las conexiones. Las puertas son interfaces de entrada y salida de los módulos. En cambio, las conexiones o enlaces son las rutas o caminos que los mensajes deben de seguir desde un módulo a otro. Estos enlaces, creados dentro de un único nivel de jerarquía, se pueden realizar mediante el uso de canales o interconectando las puertas de los módulos directamente.

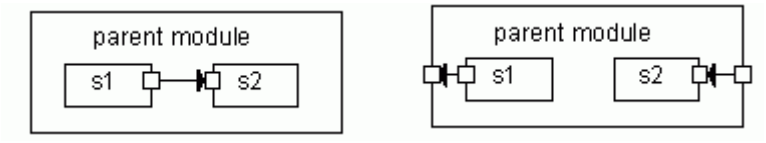

Figura 3.5: Conexión de módulos mediante puertas y enlaces

Simulación de transmisión de mensajes

Las conexiones antes descriptas entre los módulos pueden modelarse estableciendo valores para tres parámetros, muy importantes a la hora de simular comunicaciones en redes:

- Retardo de propagación.- Tiempo, medido en segundos, que tarda un mensaje en llegar desde su origen al destino cuando viaja a través del canal.
- BER, tasa de error de bit.- Probabilidad de que un bit se transmita incorrectamente.
- Tasa de transmisión.- Tiempo, medido en bits/segundo, que tarda un mensaje en ser transmitido.

Estos parámetros pueden ser o no definidos por el usuario de manera independiente en los diferentes enlaces del modelo. Además OMNeT++ proporciona la posibilidad de definir parámetros propios y característicos de un tipo de enlace.

Parámetros

Los módulos pueden aceptar parámetros al ser definidos. Mediante ellos podemos ajustar el comportamiento del módulo en cuestión o bien determinar la topología de la red a simular (número de nodos, tipo de distribución utilizada para establecer los nodos sobre la superficie de trabajo, etc.).

Los parámetros pueden ser de tipo *string*, *numeric* o *pointer*. Dentro del campo de los valores numéricos podemos encontrar desde expresiones usando otros parámetros, llamadas a otras funciones o valores de entrada mediante interacción con el usuario.

Programación de algoritmos

Tal y comentamos anteriormente el usuario debe implementar el comportamiento de los módulos simples del modelo mediante el desarrollo y programación de funciones en C++. Gracias a las características que posee este lenguaje de programación (librerías de simulación, potencia, flexibilidad, polimorfismos, abstracción, etc.) los códigos ejecutados en los módulos pueden alcanzar un alto grado de complejidad.

Durante la ejecución de las simulaciones las funciones de los módulos simples se ejecutan en paralelo, implementadas como corrutinas. De esta manera, la descripción de los algoritmos se realiza de forma intuitiva pudiendo ser aprovechada como base para implementar otros métodos de descripción.

Es importante también destacar la característica de OMNeT++ de ser un entorno orientado a objetos. Esto permite, por tanto, que los elementos que constituyen el modelo sean representados como objetos, instancias de una misma clase. Además el simulador proporciona en sus librerías clases muy útiles para la realización de las simulaciones: módulos (*cModule*), puertas (*cGate*), conexiones (*cChannel*), mensajes (*cMessage*), contenedores (*cQueue, etc.*), estadísticas (*cStatistic*), etc.

3.7.1.3 El lenguaje NED

Una vez el usuario ha definido las funciones a realizar por los módulos de su modelo debe indicarle a OMNeT++ cuales son esos módulos y la conectividad que existe entre ellos. Para ello, hay que hacer uso del lenguaje de descripción de topologías NED. Mediante él se describen cuales son los parámetros de los módulos así como las puertas que poseen.

Los ficheros que contienen la topología de la red a simular poseen, por lo general, el sufijo .ned. Estos ficheros no son usados directamente por OMNeT++ sino que son traducidos a lenguaje C++ mediante el uso del compilador nedc.

Una descripción NED se compone de las siguientes secciones, dispuestas en ordena arbitrario:

- Importaciones.- Rutas de los ficheros NED que contengan la descripción de cualquier módulo usado y no declarado en el propio fichero.
- Definición de canales.
- Declaración de módulos simples y compuestos.
- Declaración del módulo del sistema.

Como ejemplo de la estructura del lenguaje NED mostramos a continuación varios fragmentos del fichero .ned creado para la implementación del simulador de redes de sensores inalámbricos objetivo de este proyecto.

```
simple Smac
       parameters:
              dutyCycle : numeric,
              timeWakeup : numeric,
              syncflag : numeric,
              data_cw : numeric,
              sync_cw : numeric,
              sync_period : numeric,
              dataPktSize : numeric,
              syncPktSize : numeric,
              controlPktSize : numeric,
              fragthreshold : numeric;
       gates:
              in:  fromNet, fromPhysical;
              out: toNet, toPhysical;
endsimple
```

Tabla 3.1: Descripción del módulo simple Smac

En la tabla anterior podemos observar claramente la declaración de los parámetros propios del módulo así como la definición de las puertas.

Observando la *Tabla 3.2* podemos observar la topología del único módulo compuesto del simulador, el módulo Node. Vemos como además de los parámetros y puertas propias del módulo aparece la definición de los submódulos que lo

constituyen, así como las coordenadas de su disposición dentro del módulo compuesto y las conexiones entre ellos.

```
module Node
      parameters:
            id: numeric,
            energyType: string,
            netType: string,
            macType: string,
            phyType: string;
      gates:
            in: in;
            out: out;
      submodules:
            physical_layer: phyType like Physical;
                  display:"p=80,350";
            mac_layer: macType like Mac;
                  display:"p=80,250";
            net_layer: netType like Net;
                  display:"p=80,150";
            application_layer: Application;
                  display:"p=80,50";
            energy_module: energyType like Energy;
                  display:"p=200,250";
            mobility_module: Mobility;
                  display:"p=200,150";
      connections nocheck:
            application_layer.fromNet <-- net_layer.toApplication;
            application_layer.toNet --> net_layer.fromApplication;

            net_layer.fromMac <--  mac_layer.toNet;
            net_layer.toMac --> mac_layer.fromNet;

            mac_layer.fromPhysical <-- physical_layer.toMac;
            mac_layer.toPhysical --> physical_layer.fromMac;

            out <-- physical_layer.toNode;
            in --> physical_layer.fromNode;
endmodule
```

Tabla 3.2: Descripción del módulo compuesto Node

Por último mostramos en la siguiente tabla la definición de la red implementada por el simulador, utilizando, nuevamente el lenguaje NED de descripción de topologías.

```
network simulator: WirelessNet
      parameters:

endnetwork
```

Tabla 3.3: Descripción del módulo simulator

3.7.1.4 Construcción de una simulación ejecutable

El objetivo de OMNeT++ es la generación de un programa de simulación ejecutable y autónomo, que se pueda usar en cualquier máquina, siempre y cuando ésta posea una plataforma igual a aquella en la que se compiló (Linux, Windows, etc.).

En el siguiente apartado presentaremos la secuencia de acciones necesarias para la generación de este programa:

1. En primer lugar se programan las clases que ejecutarán los módulos simples del simulador. Dichas clases determinan el comportamiento de los mismos.
2. Se crea el fichero NED que describe la topología de la red. En él se deben establecer los módulos simples y los compuestos, así como las interconexiones entre ellos si las hubiera. Por último también se debe determinar la estructura del módulo que defina la red completa a simular, el módulo del sistema. Cabe destacar que toda esta información no tiene porqué estar englobada en el mismo fichero (uso de importaciones en el fichero NED).
3. Se crea un fichero de simulación, denominado *omnetpp.ini*. Este fichero sirve para ir variando los parámetros del simulador sin tener que recompilar las fuentes. El formato del fichero se divide en varios grupos o secciones llamadas [General], [Cmdenv], [Tkenv], [Parameters], [Slaves], [Run1], etc., cada uno de los cuales contiene diferentes entradas. Al igual que ocurría con los archivos .ned, en este fichero también se puede añadir otros archivos de simulación, con extensión .ini. En nuestro caso, hemos incluido los parámetros más generales del simulador en el archivo *omnetpp.ini* (número de nodos, dimensiones, etc.). Los específicos de los protocolos implementados los podemos encontrar en ficheros de simulación propios.

```
[General]
network = simulator
rng-class = "cLCG32"
ini-warnings = yes
num-rng = 1
snapshot-file = snapShot.sna
output-vector-file = xputVect.vec
output-scalar-file = yputVect.sca
sim-time-limit = 500000s
total-stack-kb = 32768

[Cmdenv]
module-messages = no
runs-to-execute = 1
default-run = 1
express-mode=yes
performance-display = no

[Tkenv]
default-run= 3
use-mainwindow = yes
animation-speed = 2.0

[Parameters]

simulator.num_nodes = 25
simulator.manager.num_dims = 2
simulator.manager.x_dim = 400
simulator.manager.y_dim = 400
simulator.manager.z_dim = 400


include smac.ini
include dsr.ini
include runs.ini
```

Tabla 3.4: Fragmento del fichero de simulación *omnetpp.ini*

4. A continuación se debe definir la compilación conjunta de todas las clases que constituyen el simulador. Para ello se ejecutar el script

opp_makemake, proporcionado por OMNeT++, el cual crea un fichero *Makefile*. Cuando todos los ficheros del modelo a simular se encuentran en el mismo directorio, el script anterior debe ejecutarse en dicha localización. En nuestro caso, para una mejor estructuración, hemos dividido el modelo en una serie de directorios, por lo cual la ejecución del comando *opp_makemake* no es tan sencillo. Por ello se ha generado un script para facilitar esta labor.

5. Una vez se han obtenido los archivos *Makefile*, se ejecuta el comando *make* necesario para la construcción del archivo ejecutable. El nombre del ejecutable será el mismo que el del directorio en el cual se esté trabajando. Para solucionar el problema de las múltiples localizaciones del código desarrollado, también se ha desarrollado un script como en el paso anterior.
6. Ejecución del programa obtenido, y posterior obtención y procesamiento de los resultados.

Aunque el proceso no es visible para el usuario, antes de construir el ejecutable, los ficheros NED y los ficheros de mensajes (ficheros con extensión .msg) necesitan ser convertidos a C++. Para ello OMNeT++ hace uso del compilador de NED (nedc) y el compilador de mensajes (opp_msgc). El proceso completo podemos observarlo en la siguiente figura:

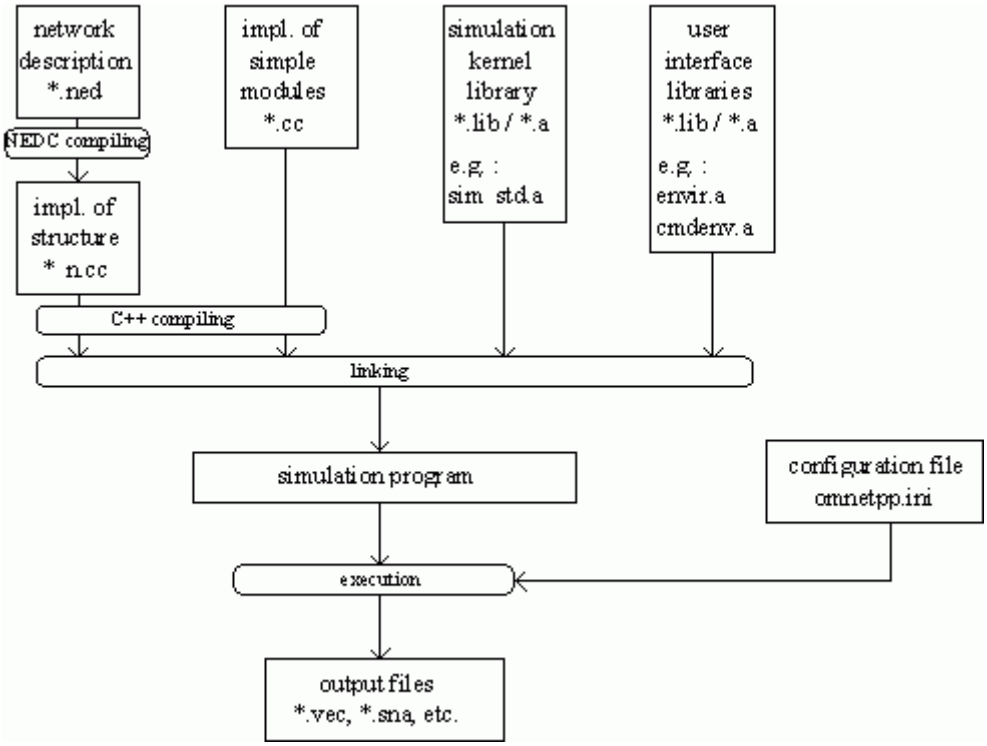

Figura 3.6: Construcción de un archivo ejecutable mediante OMNeT++

3.7.1.5 Interfaces de usuario

OMNeT++ proporciona dos interfaces diferentes a los usuarios: Tkenv y Cmdenv.

Tkenv es un entorno de ventanas basado en gráficos Tcl/Tk. En él las simulaciones se realizan de forma interactiva, permitiendo la depuración de la

simulación. Este interfaz es fundamental en los procesos previos del desarrollo de un modelo, pues nos ofrece una representación gráfica del estado de la simulación en cualquier instante de su ejecución. Entre sus características podemos destacar las siguientes:

- Animación de los mensajes.
- Visualización gráfica de las estadísticas.
- Posibilidad de observar cada módulo en una ventana diferente.
- Pila de eventos futuros.
- Cuatro velocidades distintas de ejecución (*step, run, fast run* y *express run*).

En la siguiente figura mostramos el entorno Tkenv que nos ofrece OMNeT++.

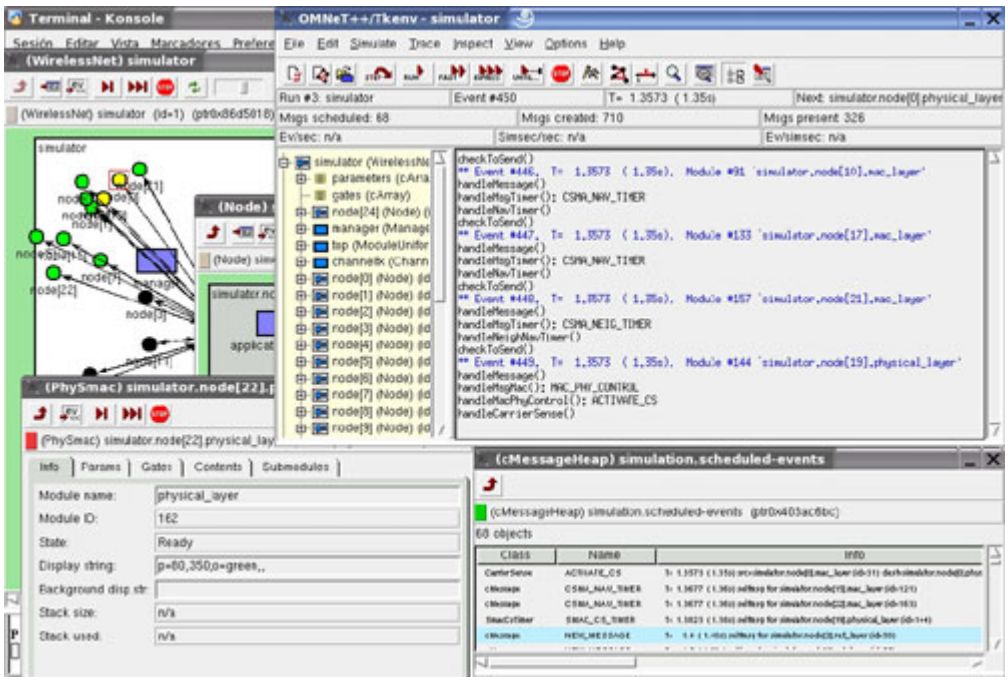

Figura 3.7: Entorno Tkenv

Cmdenv es el entorno utilizado para la realización de las simulaciones mediante línea de comandos. Se caracteriza por ser un interfaz pequeño, portable y rápido que compila y se ejecuta en todo tipo de plataformas.

Para compilar un programa y trabajar en modo Cmdenv debemos añadir al comando *opp_makemake – f* la opción *–u*.

Capítulo 4
Diseño y desarrollo del simulador

4.1 Introducción

Han sido necesarias dos fases de desarrollo para la consecución del objetivo principal marcado para este proyecto: el diseño e implementación de un simulador de redes de sensores inalámbricos.

La primera de ellas ha consistido en el desarrollo de un simulador genérico de redes inalámbricas, entre las cuales podemos incluir las redes ad-hoc y las WSN's. Para ello se ha diseñado una arquitectura de módulos mediante el uso de la herramienta OMNeT++, tal y como hemos comentado en el *Capítulo 3*.

Por tanto a lo largo de este apartado expondremos esta estructura de módulos jerárquicos. También haremos referencia a la clase o clases desarrolladas en lenguaje de programación C++ que estos módulos utilizan para la implementación de sus funciones en el simulador. Además comentaremos ciertas características referentes a la organización de los archivos así como de los ficheros de configuración y topología.

Por último expondremos una peculiaridad que nos encontramos en las WSN's, la dependencia cruzada entre las capas que constituyen la arquitectura de los nodos. Veremos los diferentes mecanismos que existen para solucionar este conflicto y cual ha sido en nuestro caso el utilizado para ello.

Durante la segunda de las fases se ha implementado un simulador de redes de sensores inalámbricos. Se ha seguido como modelo de desarrollo el comentado ampliamente en el *Apartado 3.3*. Aún así, debido a la alta complejidad que dicho objetivo conlleva han quedado por desarrollar y añadir al simulador los modelos de hardware de los sensores físicos de la red y los modelos de entorno y canales del sensor (ver *Figura 3.2*). Expondremos esta última fase en el *Capítulo 5* del documento.

Debido a que junto al código desarrollado se presenta una amplia documentación, generada mediante el programa Doxygen, en la cual se puede observar diagramas de clases, variables, funciones, etc., no realizaremos una explicación exhaustiva del código implementado, aunque si comentaremos las funciones y variables más importantes, necesarias para comprender el funcionamiento del simulador.

4.2 Arquitectura del simulador

En la *Figura 4.1* mostrada a continuación podemos observar los módulos básicos que componen el simulador.

Atendiendo a los módulos simples nos encontramos con manager (*Manager*), top (*Topology*) y channeltx (*ChannelTx*). Únicamente el último de estos presenta conectividad directa con los nodos que componen la red. Estos nodos se comunicarán con el resto a través del módulo ChannelTx, mediante el cual representamos el canal de transmisión inalámbrico.

Todos los nodos de la red, número variable cuya modificación de valor se realiza en el archivo de configuración *omnetpp.ini*, son instancias del mismo módulo compuesto. Este módulo, node (*Node*), esta formado a su vez por otros seis módulos simples. Mediante este diseño se pretende seguir el modelo de nodo para WSN's mostrado en la *Figura 3.2* y desarrollado ampliamente en el *Apartado 3.3* de este documento.

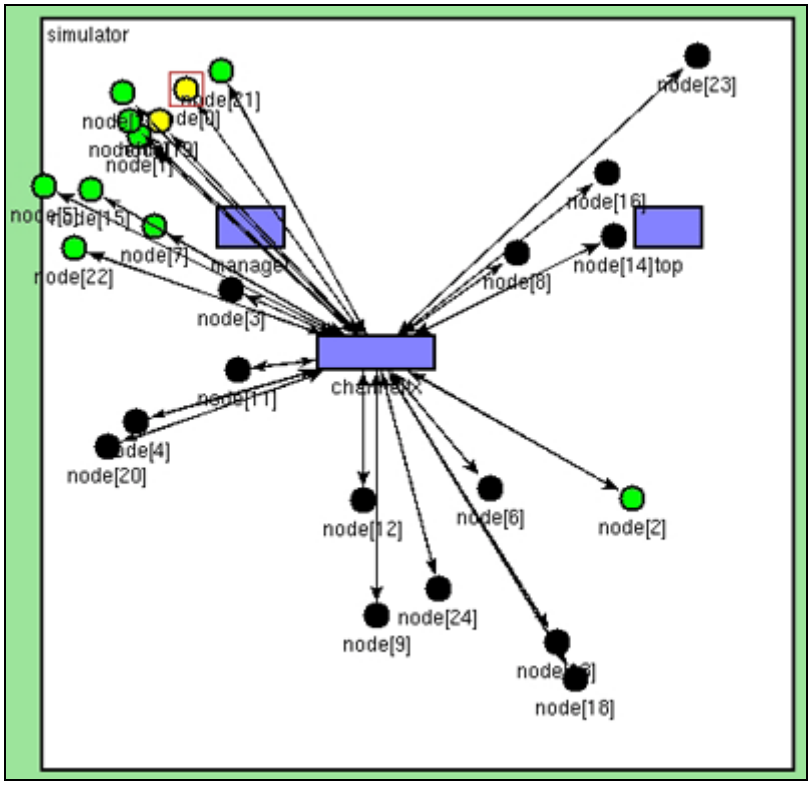

Figura 4.1: Estructura de módulos del simulador

Antes de comenzar la ejecución propiamente dicha de una simulación, los módulos deben inicializar los valores de una serie de variables. Para ello usamos la función *initialize()* propia de OMNeT++. Pero debido a la interrelación que existe entre ellos este proceso de inicialización no puede realizarse de manera secuencial. Manager, por ejemplo, necesita las coordenadas de los nodos determinadas por el módulo Topology para realizar el cálculo de las distancias medias de la red.

Para solucionar este problema de dependencia hemos hecho uso de una versión ampliada de la función de inicialización, *initialize (int stage)*, la cual se caracteriza por determinar cual será el código de la función a ejecutar según el valor que tome la variable *stage*.

Cada módulo determinará previamente de forma individual cuantas fases de inicialización poseerá y éstas se irán ejecutando de forma progresiva, es decir, primero la fase 1 de todos los módulos, luego la 2, y así de forma secuencial. Cabe destacar que los módulos de la red no tienen porqué tener el mismo número de fases de inicialización, tal y como se muestra en la *Figura 4.2*.

A continuación pasaremos a desarrollar más detenidamente los módulos antes comentados, destacando sus principales funcionalidades así como las clases que posibilitan la realización de dichas actividades.

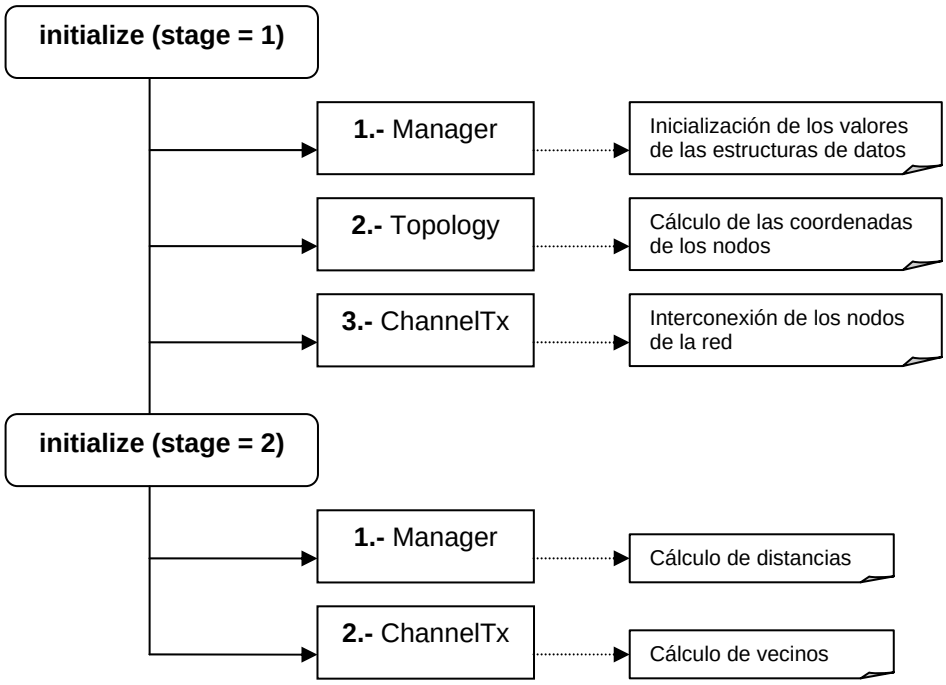

Figura 4.2: Fases de inicialización del simulador

4.2.1 Manager

El módulo simple Manager, considerado el núcleo central del simulador, se caracteriza por poseer entre sus funciones la definición de las estructuras básicas de almacenamiento de datos del simulador. Para ello hace uso de la clase específica InfoNeighbour y de la estructura de datos *info_node* definida en la clase Manager.

InfoNeighbour se utiliza para que en cada uno de los nodos de la red almacene información referente a sus vecinos. Entre esta información podemos destacar el identificador del nodo y la distancia existente entre el dispositivo en cuestión y su vecino. Además la clase aporta métodos para la obtención y modificación de estos valores.

Cada nodo, por tanto, poseerá tantos punteros a objetos de la clase InfoNeighbour como vecinos posea. Para almacenar estos punteros se hará uso de la estructura de C++ *info_node* donde también podremos encontrar otro tipo de información, esta vez referente al nodo: identificador, coordenadas de posición sobre la superficie de trabajo, valor de sensibilidad asignado para realizar el cálculo de vecinos, etc.

En la primera fase de inicialización del módulo Manager, tal y como hemos comentado en el apartado anterior, se crean las estructuras de datos de los nodos antes comentadas y se le dan valores. Cabe destacar que el valor de la potencia de transmisión del nodo se obtiene o bien mediante un fichero de datos o bien con un valor único introducido mediante el uso del archivo de configuración.

Una vez se ha realizado la distribución inicial de los nodos sobre la superficie de trabajo y se han calculado las relaciones de vecindad existentes entre los nodos el módulo Manager calcula una serie de valores de distancias, los cuales son almacenados en el fichero obtenido como resultado de la simulación y mostrados a través del interfaz de usuario:

- Distancia cuadrática media de un nodo con respecto cualquier otro de la red.
- Distancia cuadrática media de un nodo.

- Distancia cuadrática media total entre nodos.
- Distancia media total entre nodos.

Destacar que la determinación inicial de las coordenadas de los nodos y los vecinos las realizan otros de los módulos que componen el simulador. Estas y otras funciones serán comentadas más adelante.

Manager realiza también otra serie de actividades. Entre ellas encontramos la actualización de los valores de los niveles de potencia y las coordenadas de los nodos a medida que estas variables van modificándose durante la ejecución de las simulaciones.

Por último, una vez que el módulo concluye su ejecución realiza la recolección de todas las estadísticas de las capas de los nodos de la red y las almacena en el fichero de resultados.

4.2.2 Topology

Una vez concluye la ejecución de la primera fase de la inicialización del módulo Manager, comienza la inicialización del módulo Topology que se caracteriza por tener una única fase.

Topology es un módulo simple y polimórfico, es decir, observando la definición que de este bloque se realiza en el archivo .ned podemos ver como no se le incluyen ni puertas ni enlaces ni parámetros. Es, por tanto, un módulo vacío.

El módulo se encarga de realizar la distribución de los nodos que constituyen la red sobre el área de trabajo. Además también realiza la representación gráfica de los dispositivos según las coordenadas calculadas por él.

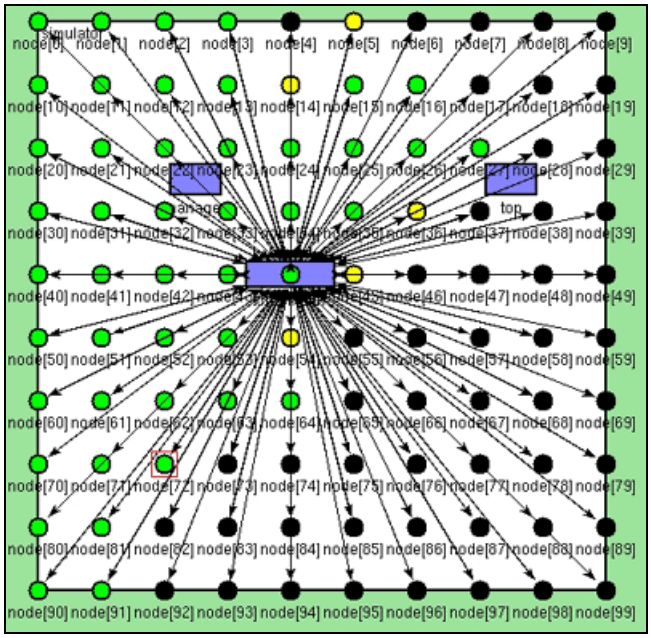

Figura 4.3: Distribución en rejilla de 100 nodos sobre la superficie de trabajo

Para la realización de estas actividades el módulo de topología de la red puede hacer uso de cuatro clases hijas de Topology. Todas ellas poseen parámetros propios incluidos para su posible modificación en el archivo *omnetpp.ini* y realizan el cálculo de las coordenadas de los nodos siguiendo diferentes criterios de ejecución:

- ModuleUniformTopology: Cálculo de las coordenadas iniciales de los nodos siguiendo una distribución uniforme.
- ModuleNormalTopology: Cálculo de las coordenadas iniciales de los nodos siguiendo una distribución normal con media *mean* y desviación típica *std*.
- GridTopology: La distribución de los nodos se realiza en forma de *grid* o rejilla.
- FileTopology: Las coordenadas de los nodos las introducimos haciendo uso de un fichero de texto.

En la *Figura 4.3* podemos observar un ejemplo de la ejecución del módulo Topology ejecutando las funciones determinadas por la clase GridTopology.

4.2.3 ChannelTx

El módulo simple ChannelTx al igual que Manager posee un proceso de inicialización en dos fases. En la primera de ellas, una vez Manager ha determinado los valores de potencia de los nodos y Topology ha calculado las coordenadas de los mismos, ChannelTx realiza la conexión de los dispositivos de la red utilizándose él mismo como intermediario de esta unión. Mediante este módulo, por tanto, modelaremos el canal inalámbrico de comunicaciones. Gráficamente podemos observar las conexiones entre los nodos en la *Figura 4.4*.

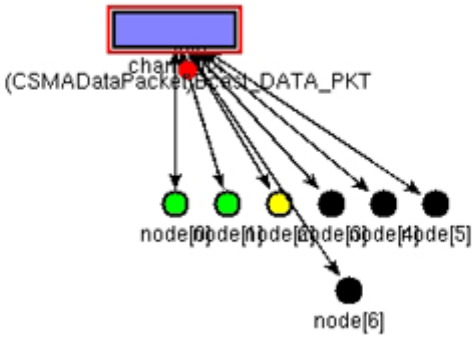

Figura 4.4: Funcionamiento del canal de transmisión

En la segunda de las fases el módulo que representa el canal de comunicaciones determina los rangos de cobertura de todos los nodos de la red, identificando cuales son sus vecinos. Según el valor de la variable *static_simulation*, que podemos modificar en el archivo de configuración, este cálculo de vecinos iniciales se puede realizar de dos formas:

- Simulación estática: Este modelo está preparado para simulaciones donde los nodos de la red no se desplacen. Por ello se calculan los vecinos a un determinado nodo estableciendo cuales son los que se encuentran en el rango de cobertura del nodo en cuestión. Esta cobertura puede ser calculada para varios valores de potencia, almacenados en el vector *matrix_power*.
- Simulación dinámica: Mediante este tipo de simulación el canal de transmisión ChannelTx realiza el cálculo de los vecinos a un nodo en cuestión haciendo uso de un único valor de potencia de transmisión, almacenado en la variable *tx_power* de la estructura *info_node*.

Una vez calculados los vecinos, cuando el canal recibe un mensaje por una de sus puertas lo reenviará a todos aquellos dispositivos que estén en el mismo rango de cobertura. Cabe destacar que si los niveles de potencia de los dispositivos van

variando debe realizarse nuevamente el cálculo de los vecinos, pues los radios de cobertura se irán modificando. ChannelTx incorpora también esta funcionalidad.

Por último destacar que el modelo de propagación utilizado en la clase ChannelTx sigue la siguiente ecuación:

$$Pr = \frac{Pt \cdot k}{d}$$

donde *Pr* es la potencia recibida y *d* la distancia que existe entre emisor y receptor. El valor de la variable *k* viene determinado por la siguiente expresión:

$$k = \frac{c^2}{16 f^2 \pi^2}$$

donde *c* es la velocidad de la luz ($3 \cdot 10^8$ m/s) y *f* la frecuencia a la que transmiten los dispositivos.

4.2.4 Node

Atendiendo a la *Figura 4.5* podemos observar la estructura compuesta del módulo Node. De los seis módulos simples que lo forman cuatro de ellos están interconectados (Application, Net, Mac y Physical) mientras que los dos restantes, Mobility y Energy, se encuentran en su interior sin una aparente conexión con el resto de las capas del nodo.

Las clases bases programadas para determinar las funciones básicas a ejecutar por las capas Application, Net, Mac y Physical se caracterizan principalmente por su simplicidad. Podemos destacar las siguientes actividades implementadas:

- Determinación de las puertas de entrada y/o salida de los módulos.
- Reenvío de los mensajes entre las capas de los nodos hasta alcanzar el nivel superior, si el mensaje procede del canal.
- Reenvío de los mensajes entre las capas de los nodos hasta alcanzar el canal de transmisión, si el mensaje ha sido generado por la capa de aplicación del nodo.

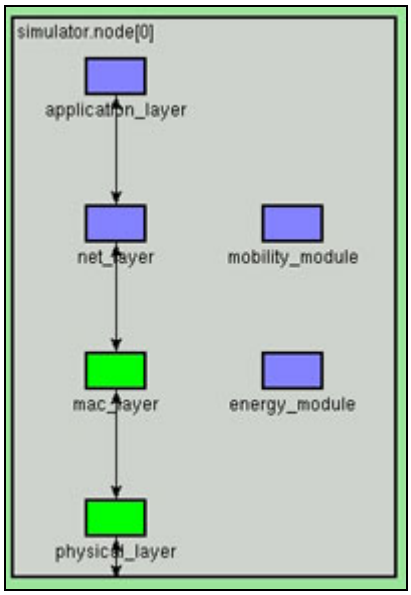

Figura 4.5: Diagrama de módulos del los nodos del simulador

Cuando se desee que alguna de estas capas ejecute funciones específicas, propias de algún tipo de protocolo, se deberá implementar una nueva clase derivada de las clases padre Physical, Mac, Net o Application.

Para indicar al simulador que clases serán las que ejecutarán las capas de los nodos haremos uso del archivo *omnetpp.ini*, tal y como podemos ver en la *Tabla 4.1*. En él podremos determinar la pila de protocolos de los nodos de la red, así como si la red será heterogénea o no, pudiendo tener nodos con diferentes protocolos en las distintas capas del modelo.

```
simulator.mactype = "Smac"
simulator.phytype = "PhySmac"
simulator.node[*].netType = "Dsr"
simulator.node[*].energyType = "Energy"
```

Tabla 4.1: Determinación de clases en las capas de los nodos del simulador

Como parte de los objetivos de este proyecto se han desarrollado las siguientes clases que enumeramos a continuación y algunas de las cuales serán desarrolladas ampliamente en el siguiente capítulo:

- Nivel de red: Clase Dsr. Mediante su uso los nodos de la red ejecutarán el protocolo de enrutamiento DSR. Cabe destacar que esta clase incorpora generación de mensajes, por lo cual no ha sido necesario el desarrollo de una clase que realice dicha función en el nivel de aplicación.
- Nivel de enlace de datos: Clases Smac y CSMA. La primera de ellas será desarrollada ampliamente en el siguiente capítulo y mediante ella los nodos de la red podrán usar el protocolo de enlace de datos S-MAC.
- Nivel físico: Clases PhySmac y PhySmacCs. Ambas clases se caracterizan por su gran semejanza y por implementar protocolos de nivel físico muy ligados a la ejecución en la capa de enlace del protocolo S-MAC. Su diferencia estriba en que PhySmacCs incorpora la posibilidad de que el canal de transmisión introduzca errores en los paquetes de datos que reciben los nodos.

Por último debemos tratar el tema de las estadísticas calculadas por las diferentes clases ejecutadas en las capas de los nodos durante las simulaciones.

Como ya comentamos anteriormente, el módulo Manager se encarga de realizar la recolección de estas estadísticas una vez ha concluido la simulación. Para trabajar con estadísticas en el simulador se ha desarrollado la clase Stats derivada de la clase propia de OMNeT++ cObject e independiente de cualquiera de los módulos que componen el simulador. Cada vez que una de las capas de los nodos desea tener una variable donde calcular estadísticos, crea una instancia de la clase Stats. Esta clase proporciona parámetros de tal manera que permitirán a Manager identificar posteriormente a que nodo pertenece esa variable, en que capa del modelo se ha generado, que tipo de variable es, etc.

4.3 Características

A continuación vamos a comentar ciertas características del simulador desarrollado, necesarias para un mejor conocimiento del mismo y facilitar así su posterior uso. Entre ellas podemos destacar la organización de los archivos que constituyen el simulador, los ficheros de configuración y los de topología.

Versión y plataforma

El simulador desarrollado está basado en la versión de OMNeT++ v3.0 y ha sido implementado bajo la plataforma Linux.

Organización en directorios

Para facilitar las posibles ampliaciones del entorno desarrollado se han distribuido todos los archivos programados en una serie de directorios:

- *html*: Directorio creado automáticamente donde el programa de extracción de documentación Doxygen ha almacenado todos los archivos generados como resultado de su ejecución. Para poder hacer uso de la documentación se deberá abrir el archivo *index.htm* en un navegador web. Editando el archivo *Doxyfile* que encontramos en el directorio raíz podremos determinar cuales han sido las opciones utilizadas para el desarrollo de la documentación.
- *core*: Directorio núcleo del simulador. En esta localización se encuentran las clases bases utilizadas para determinar las funciones básicas a realizar por los módulos del simulador. Por ejemplo, en este directorio se encuentra la clase Mac. Si no implementásemos ningún tipo especial de protocolo de acceso al medio, la capa de enlace de datos ejecutaría el código simplificado determinado por esta clase.

  A su vez dentro del directorio *core* encontramos otros dos, *include* y *msg*. En el primero de ellos podemos encontrar los archivos de cabecera (extensión .h) donde declaramos las constantes del simulador, las propias de las distintas capas que constituyen los nodos y las librerías de C++ usadas para la compilación (*global.h*). Las constantes relativas a las capas de los nodos se caracterizan por ser genéricas, es decir, independientes de cualquiera de los protocolos desarrollados.

  En el otro de los directorios, *msg*, podemos encontrar las clases que determinan la estructura de los mensajes genéricos del simulador, es decir, los que son independientes de los protocolos.
- *topology*: En este directorio podemos encontrar las cuatro clases derivadas de la clase padre Topology comentadas anteriormente.
- *channel*: Aunque en este caso no almacenemos en su interior ninguna clase que determine características diferentes del canal de transmisión, se ha añadido para su posible uso en futuras modificaciones del simulador.
- *statistic*: Directorio donde se encuentran los archivos .cc y .h que definen la clase Stats.
- *application*: Actualmente no se utiliza. Destinado al almacenamiento de las clases a ejecutar por la capa de aplicación de los dispositivos.
- *net*: Directorio de almacenamiento de los diferentes protocolos a ejecutar en la capa de red. Encontramos en su interior el directorio *dsr*, lugar donde almacenamos el código fuente de la clase Dsr.
- *mac*: Idénticas características que *net*, pero orientado a la capa de enlace de datos. En su interior nos encontramos con dos protocolos distribuidos en en los siguientes emplazamientos: *smac* y *csma*.
- *physical*: Directorio destinado a la localización de los protocolos del nivel físico. Actualmente se encuentra desarrollado el protocolo PhySmac.

omnetpp.ini

En el directorio raíz podemos encontrar el fichero de simulación *omnetpp.ini*. Debido a las características de generalidad que buscamos en el simulador son muchos los parámetros modificables incluidos en este archivo. A continuación

comentaremos algunos de los más importantes, los situados en la sección [Parameters] del archivo:

- *simulator.man*: Cadena de caracteres donde indicamos la clase que se ejecutará en el módulo Manager.
- *simulator.topology*: Clase que determinará la distribución de los nodos en la superficie.
- *simulator.num_nodes*: Número de nodos que constituyen la red.
- *simulator.channeltype*: Clase a ejecutar por el módulo que representa el canal de transmisión.
- *simulator.mactype*: Cadena de caracteres que determina el protocolo de la capa de enlace de datos.
- *simulator.netType:* Protocolo de enrutamiento de la capa de red de los nodos.
- *simulator.manager.output_file*: Nombre del fichero de resultados.
- *simulator.manager.num_dims*: Número de dimensiones que tendrá el espacio donde se realicen las simulaciones.
- *simulator.manager.static_simulation*: Variable cuyo valor determina el tipo de cálculo de vecinos a realizar por el canal.
- *simulator.manager.powers*: Nombre del fichero con los valores de potencia de los nodos.
- *simulator.manager.init_tx_power*: Potencia de transmisión de los nodos.
- *simulator.manager.sensibility_node*: Sensibilidad de los nodos de la red.
- *simulator.manager.frequency*: Frecuencia a la que transmiten los dispositivos.

Observando las variables aquí expuestas vemos que se caracterizan por ser generales, es decir, independientes de los protocolos establecidos en las capas de los nodos. Para añadir las variables propias de estos protocolos se recomienda incluir en el archivo de configuración *omnetpp.ini* tantos archivos de extensión .ini como protocolos se vayan a usar. En nuestro caso hemos añadido los archivos *smac.ini* y *dsr.ini* al final del archivo tal y como quedó reflejado en la *Tabla 3.4*.

simulator.ned

Junto con el archivo de configuración podemos encontrar en el directorio raíz de la aplicación el archivo *simulator.ned*. Este fichero está escrito en el lenguaje de descripción de topologías NED y mediante él se ha establecido la estructura general que presenta el entorno.

Como comentamos en apartados anteriores en este archivo encontramos tanto la descripción de los módulos simples de la red como los compuestos.

A pesar que los módulos del modelo están ya definidos en el fichero si deseamos generar un nuevo protocolo para una determinada capa del nodo y éste requiere parámetros o puertas de enlace distintas a las del módulo básico será necesario agregar una definición de un nuevo módulo o modificar la ya existente.

4.4 Acoplamiento entre capas

En 1984, la Organización Internacional de Estandarización (ISO) desarrolló un modelo para describir los entornos de red, en el cual el funcionamiento de los dispositivos quedaba dividido en una serie de capas abstractas. Era el modelo de Interconexión de Sistemas Abiertos u OSI (*Open Systems Interconnection*).

En este modelo, ampliamente extendido, el propósito de cada capa es proveer servicios para la siguiente capa superior, resguardándola de los detalles de como los servicios se realizan realmente. Cada capa cree que está intercambiando información con la capa asociada del dispositivo con el que se comunica, cuando realmente cada capa se comunica sólo con las adyacentes.

Ahora bien, a medida que se produce la implantación de las WSN's y se estudian sus características se puede observar que el modelo OSI no es el mejor referente para representar este tipo de redes.

Mientras que en OSI las capas, aunque abstractas, tiene bien definidas sus funcionalidades y la comunicación entre las mismas, en una red de sensores inalámbricos las limitaciones de energía y capacidad de memoria y procesamiento hacen que estas fronteras no estén muy bien definidas. Además no se sabe con seguridad "quién" y "cuándo" necesitará la información de una determinada capa. Se produce, por tanto, lo que se denomina dependencia cruzada entre capas.

Para solucionar este problema podemos actuar de dos maneras: usando el mecanismo de la pizarra (*blackboard*) o mediante el uso de eventos.

El primero de estos mecanismos consiste en que cada vez que en la red se genere información que puede ser considerada necesaria para otras partes del sistema, estos datos se almacenan en una zona de memoria a la cual pueden tener acceso todos los dispositivos del sistema que precisen de esa información.

El segundo de los métodos se corresponde con el paradigma del editor/suscriptor (*publisher/suscriber*), y es el que hemos utilizado para la implementación del simulador desarrollado.

Para ello se ha generado una clase denominada Event. A partir de ella se han derivado clases hijas que hacen referencia a diferentes tipos de información necesaria para el funcionamiento correcto de los protocolos. Podemos encontrar clases referentes a los siguientes tipos de información:

- Cambio de estado de la radio del dispositivo (ChangeRadioStateEvent).
- Cambio de las coordenadas del nodo (ChangeCoordEvent).
- Cambio del nivel de potencia del sensor (ChangePowerEvent).
- Cambio de estado del nivel de enlace de datos (ChangeMacStateEvent).

Cuando los módulos se crean, se suscriben a la información que desean recibir, una vez ésta se ha producido. Por tanto, los módulos tendrán acceso a estos datos sin necesidad de tener que leerlos de una memoria general del sistema.

Mediante el uso de la función *handleEvent()* los módulos que generan eventos envían esta información a todos los dispositivos que se han inscrito a ella, es decir, los nodos reciben la notificación de la ocurrencia de un evento de su interés. Este proceso de suscripción se realiza mediante la función *setEventListener()*.

Capítulo 5
Modelos implementados

5.1 Introducción

Como ya hemos comentado en la introducción del capítulo anterior, durante la consecución del objetivo fundamental de este proyecto, el diseño de un simulador de redes de sensores inalámbricos, se han seguido dos fases.

La primera de ellas, la implementación de un simulador general de redes inalámbricas, se desarrollo y completó en el *Capítulo 4* de este documento. A continuación nos encargaremos de exponer la segunda de las fases: la implementación de un simulador de redes de sensores inalámbricos siguiendo el modelo básico de WSN's [11].

Para ello se han añadido al simulador genérico desarrollado dos protocolos específicos de redes de sensores. El primero de ellos, S-MAC, es un protocolo de acceso al medio cuya principal característica son las aportaciones que realiza para solventar el problema de la limitación energética de los dispositivos que constituyen estas redes. Ahora bien veremos que como contrapunto a esta mejora S-MAC está estrechamente vinculado al nivel físico del nodo. Esto provoca la necesidad de desarrollar de forma conjunta tanto el protocolo de la capa de enlace de datos como el ejecutado en la capa física.

DSR, protocolo de nivel de red, es el segundo de los módulos incorporados al simulador. Este protocolo se caracteriza por realizar enrutamiento de los paquetes transmitidos por los nodos aprovechando la arquitectura distribuida de las redes ad-hoc. Debido a que los dispositivos inalámbricos se caracterizan por tener capacidades de procesamiento limitadas, los nodos que ejecutan el protocolo DSR aprovechan la información de ruta de todos los paquetes de datos que son encaminados a través de ellos.

Ahora bien, a pesar de haber agregado al simulador estos dos módulos para poder comparar realmente el modelo de nodo aquí desarrollado con el expuesto en el *Capítulo 3* nos faltaría incorporar las características hardware de los sensores físicos de la red y los modelos de entorno y canales de los sensores.

5.2 S-MAC (Sensor Medium-Access Control)

En el siguiente apartado expondremos la implementación realizada del protocolo de enlace de datos específico para redes de sensores, el protocolo S-MAC.

Como ya hemos comentado en varios apartados, S-MAC se caracteriza por reducir el consumo de potencia de los dispositivos que lo ejecutan mediante el uso de un ciclo de trabajo que incorpora periodos de actividad o *idle* y periodos de sueño o *sleep* cuando el nodo no puede utilizar el canal de transmisión. Para realizar tal función, es necesaria una vinculación especial entre el protocolo de la capa de enlace de datos y el protocolo que el dispositivo está ejecutando en el nivel físico.

Por ello, adaptar el simulador para la ejecución del protocolo S-MAC en los nodos del mismo ha supuesto el desarrollo no sólo de la clase Smac orientada a las capas Mac de los nodos, sino también la implementación de PhySmac, clase que proporciona todas las funcionalidades de la capa física. Las principales características de ambas serán comentadas en el siguiente apartado.

A continuación explicaremos dos principales estructuras de datos utilizadas por S-MAC, la lista de planificaciones y la lista de vecinos, y para finalizar enumeraremos y expondremos los principales mensajes utilizados por este protocolo. Estos mensajes se dividen en temporizadores, paquetes de control y mensajes propios del protocolo.

Cabe destacar que debido a que las clases Smac y PhySmac están estrechamente ligadas, la capa física debe tener un conocimiento pleno de la estructura de los mensajes utilizados por la capa Mac. Esta característica se opone claramente al criterio establecido en el modelo OSI, por el cual las actividades realizadas por una determinada capa deben ser transparentes para cualquiera otra de la pila.

5.2.1 Diseño

La ejecución de la clase Smac en la capa de enlace de datos comienza con la ejecución de la función propia de OMNeT++ *initialize().* Durante la única fase que caracteriza a la versión simple de esta función Smac establece los valores de las variables propias del protocolo. Estos valores se introducen en el simulador haciendo uso del fichero *smac.ini* el cual ha sido incluido en el archivo general de configuración del simulador *omnetpp.ini*.

Como comentábamos anteriormente, es imposible hablar de Smac sin desarrollar simultáneamente el protocolo implementado para el nivel físico de los nodos, PhySmac. La característica que intenta aportar S-MAC a los dispositivos en relación al control del gasto energético obliga a que ambas capas estén estrechamente relacionadas. Vemos aquí un claro ejemplo del fenómeno de acoplamiento entre capas, ampliamente comentado en el *Apartado 4.4*. Debido a esta vinculación entre capas las variables propias de PhySmac también han sido introducidas en el archivo *smac.ini*.

Smac no comenzará su funcionamiento normal hasta que el nivel físico haya inicializado sus variables, mediante el uso también de la función *initialize()*, y le envíe un mensaje del tipo INIT_SMAC. En este mensaje el nivel físico envía al Mac información necesaria para realizar el cálculo de los tiempos de transmisión de los paquetes. Entre estos datos encontramos:

- *slotTime:* Tiempo de slot.
- *Bit_tx_time:* Tiempo de transmisión de un bit.
- *guardTime:* Tiempo de guarda.
- ENCODE_RATIO: Ratio de codificación del código empleado en el nivel físico. Al utilizar código Manchester, el valor de esta constante es de 2.

Una vez Smac recibe este mensaje y realiza los cálculos temporales pertinentes genera las variables estadísticas que precisa haciendo uso de instancias a la clase Stats, tal y como comentamos en el *Apartado 4.2.4* cuando hablábamos de la estructura de los nodos del simulador. Por último en esta fase previa el protocolo realiza la inicialización de las dos estructuras de datos que utiliza: la lista de vecinos y la lista de planificadores.

A continuación el módulo queda en espera de la llegada de mensajes procedentes de cualquiera de las tres entradas que posee el módulo Mac: el nivel de red, el nivel físico o la llegada de un temporizador generado por el mismo.

Del nivel de red, la clase Smac únicamente puede recibir un tipo de mensaje característico del simulador y no del protocolo de nivel superior utilizado, *NetworkPdu*. Este paquete, que desarrollaremos más ampliamente cuando tratemos la implementación del protocolo DSR, almacena en su interior la dirección Mac del nodo destino y el mensaje generado por la capa superior. El contenido de este mensaje es ajeno totalmente para Smac, no necesita conocerlo para realizar correctamente sus funciones.

Una vez Smac determina si el mensaje se enviará mediante un procedimiento broadcast o unicast, lo prepara encapsulándolo en un mensaje del tipo *SmacDataPacket* y almacenándolo en el buffer *dataPkt* a la espera de que comience un periodo de *idle* de su ciclo de trabajo.

Procedentes del nivel físico pueden llegar tres tipos de mensajes. En primer lugar nos encontramos con los de datos. Una vez Smac ha comprobado que estos han sido recibidos correctamente (podría haberse producido una colisión, por ejemplo) son enviados a la capa superior de la pila de protocolos del nodo.

Otros de los tipos de mensajes procedentes del nivel físico son los utilizados por Smac para realizar el control de acceso al medio: RTS, CTS y ACK. También podemos encontrar los mensajes tipo SYNC. Mediante su uso el nodo transmite a los demás cuales son sus planificadores, es decir, que sincronización va a seguir para establecer su ciclos de sueño y de trabajo.

Por último Smac y PhySmac intercambian mensajes de los denominados de control. A este grupo pertenecen los mensajes INIT_SMAC, PKT_SENT y PHY_STATE. El segundo de ellos indica a Smac que se ha trasmitido un paquete. Dependiendo del tipo el protocolo realizará unas u otras funciones tales como cambio de estado de la capa, envío de mensajes de control a la capa de red, etc.

Mediante la recepción del mensaje PHY_STATE el nivel físico informa al Mac de que se ha producido un cambio de estado. Estos cambios en el valor de la variable *radioState* pueden estar originados o bien por la acción directa de la capa de enlace de datos sobre el nivel físico (por ejemplo, cuando el nivel Mac obliga a la radio del dispositivo a pasar a estado de detección de portadora o CR_SENSE) o bien por la actividad independiente del canal físico (es el caso que se produce cuando el nivel físico pasa al estado de recepción o RADIO_RX debido a la detección de una portadora en el canal de transmisión).

El último de los tipos de mensajes que pueden llegar al nivel Mac de los nodos son los temporizadores. Entre las funciones que realizan podemos destacar que los comienzos de las diferentes fases del ciclo de trabajo de Smac vienen determinados por la llegada de este tipo de mensajes. A medida que se van cumpliendo cada uno de los intervalos de los que consta el ciclo se va planificando el siguiente con la duración temporal determinada para tal efecto a la vez que se van realizando funciones propias del protocolo. Por ejemplo, si el nodo se despierta y hay planificado un mensaje para enviar, el nivel Mac avisará al físico de que debe activar un proceso de detección de portadora para determinar si el canal está o no ocupado.

Cabe destacar que todos los valores temporales utilizados por estos temporizadores son modificables accediendo al archivo *smac.ini*.

Además debemos comentar que Smac está preparado también para realizar fragmentación. Si los mensajes procedentes del nivel de red superan un cierto umbral determinado por la variable *fragthreshold* el paquete se fragmentará en varias unidades siendo transmitidas una tras otra al nodo destino. Éste será el encargado de

unir los fragmentos y una vez completado el mensaje original Smac se encargará de transmitirlo a la capa superior. Para evitar errores en el proceso de fragmentación se han incluido en las cabeceras de los mensajes números de secuencia y éste no se realiza cuando los mensajes son broadcast.

Atendiendo ahora al protocolo PhySmac implementado para realizar las funciones propias del nivel físico necesarias para la ejecución del protocolo S-MAC, vemos como tras realizar la inicialización de las variables propias de esta capa, crear las estadísticas pertinentes siguiendo el mismo proceso que Smac y enviar a la capa superior el mensaje INIT_SMAC que permite el cálculo de los valores temporales de los mensajes emitidos, el nivel físico se queda a la espera de la llegada de mensajes procedentes también de tres vías: el nivel superior o Mac del que pueden llegar paquetes de control o de datos, el canal de transmisión o el cumplimiento de temporizadores.

Entre las funciones principales de PhySmac podemos destacar la realización de la detección de portadora y la transmisión y recepción de los mensajes que se deben enviar al canal de transmisión.

Por último destacar que para aprovechar las ventajas gráficas que nos ofrece OMNeT++ se ha utilizado un código de colores que nos permitirá ver a simple vista el estado en que se encuentran los niveles físico y Mac de la red. Si las capas están en color rojo implicará que el nodo se encuentra en periodo *sleep* o de sueño. Si en cambio son de color amarillo implicará que el nodo se encuentra realizando el proceso de detección de portadora. Por último los estados de transmisión, recepción e *idle* se representarán con color verde.

5.2.2 Estructuras de datos

Además del buffer donde Smac almacena el mensaje que está pendiente de ser transmitido (*dataPkt*) y los punteros a los mensajes que están en proceso de envío o recepción (*pktTx* y *pktRx*) existen dos estructuras de datos que tienen una gran importancia en el funcionamiento del protocolo: la lista de vecinos y la lista de planificaciones.

schedTable

Vector donde almacenaremos como máximo SMAC_MAX_NUM_SCHEDULES punteros a objetos de la clase SchedTable.

SchedTable

Clase derivada de cObject que proporciona a los nodos de la red una estructura de datos donde almacenar la información referente a cada una de las planificaciones de las que tiene conocimiento durante la ejecución de la simulación.

Entre la información que podemos encontrar en esta clase destacamos el parámetro *nodesSched* en el cual el nodo va recopilando el número de dispositivos que siguen cada una de las planificaciones conocidas por él.

neighbList

Vector donde almacenaremos como máximo SMAC_MAX_NUM_NEIGHBORS punteros a objetos de la clase NeighbList.

NeighbList

Clase derivada de cObject cuya función es ofrecer a los nodos de la red una estructura de datos donde almacenar la información referente a los vecinos que conocen.

Los parámetros de información que la clase NeighbList proporciona son los siguientes:

- *nodeId:* Identificador del nodo vecino.
- *schedId*: Si un nodo de la red es vecino de un dispositivo en cuestión, este último debe conocer la planificación que sigue. Mediante esta variable identificamos cual de las planificaciones almacenadas en el vector *schedTable* sigue el nodo vecino.
- *state*: Debido a la dificultad añadida de eliminar elementos de un vector, si estos no se encuentran al principio o final del contador, y a la posibilidad de que nodos que eran vecinos dejen de serlo, se ha añadido a la clase este parámetro. Si el valor de la variable es 0 implicará que la posición del vector está libre y podrá ser utilizada para almacenar información referente a un nuevo vecino. De igual manera si un nodo deja de ser vecino el valor de la variable pasará de 1 a 0.
- *active:* Variable que nos indica si hemos recibido actividad de un determinado vecino. Si cuando actualizamos la lista de vecinos esta variable tiene valor 0, consideraremos que el vecino se ha perdido.
- *txSeqNum:* Variable donde almacenamos el número de secuencia de transmisión para un determinado vecino.
- *rxSeqNum:* Variable de almacenamiento del número de secuencia de recepción para un determinado vecino.

A medida que van sucediéndose en los nodos del entorno los ciclos de trabajo, cada vez que concluye un periodo de sincronismo el código desarrollado incrementa en una unidad el valor de la variable *numSyncTimeNeigh*. Cuando dicha variable iguala o supera el resultado de multiplicar la constante UPDATE_NEIGHG_PERIOD por la variable *sync_period* Smac realiza la actualización de la lista de vecinos teniendo en cuenta los valores almacenados en los objetos NeighList. A continuación, se realiza también la revisión de la lista de planificadores pues podría darse el caso de que se perdieran todos los nodos conocidos de una determinada planificación, lo que implicaría prescindir de dicha planificación.

Por último comentar que teniendo en cuenta las características de actualización de los dos vectores utilizados, en los cuales puede producirse la necesidad de eliminar algún elemento de los almacenados, parecería más correcta la utilización de listas. Pero en este caso ha primado más la posibilidad que ofrecen los contenedores de tipo vector de acceder a los elementos almacenados de forma directa que las características de eliminar e insertar elementos en posiciones intermedias que proporcionan las listas. El número máximo de vecinos permitido por S-MAC es 20 lo que implica que la elección del contenedor adecuado no se convierte en un punto crítico del diseño realizado.

5.2.3 Mensajes

Resumimos brevemente los mensajes más importantes de Smac.

SmacControlFrame

Mensajes utilizados por Smac para realizar el control de acceso al medio inalámbrico implementado. En la cabecera de estos paquetes, entre los que encontramos los mensajes del tipo RTS, CTS y ACK, se definen los siguientes campos o variables:

- *type:* Tipo de mensaje.

- *dstAddr*: Dirección Mac del nodo destino.
- *srcAddr:* Dirección Mac del nodo origen del mensaje.
- *duration:* Duración temporal del mensaje.
- *crc:* Variable preparada para el almacenamiento del código de redundancia cíclica o CRC (*Cyclic Redundancy Check*).

SmacDataPacket

Paquete de datos donde Smac encapsula el mensaje procedente de nivel de red, una vez el nodo ha conseguido el control del medio. Además de los campos comentados anteriormente, los mensajes de datos añaden dos más:

- *sequence:* En caso de realizar fragmentación del mensaje, en esta variable se almacena el número de secuencia.
- *numFragment:* Número total de fragmentos que constituyen el mensaje original. Utilizado por el nodo receptor para realizar la reconstrucción del paquete.

SmacSyncFrame

Mensaje utilizado por los nodos para informar a los vecinos cual es la planificación seguida. Entre la información transmitida podemos destacar la almacenada en la variable *sleepTime*, tiempo que le queda al nodo para pasar a estado *sleep*.

MacPhyControl

Mensajes de control que el nivel Mac intercambia con el físico. Entre ellos destacamos *CarrierSense* mediante el cual el nivel físico inicia el proceso de detección de portadora.

MacNetControl

Mensajes de control que la capa de enlace de datos envía al protocolo de nivel de red que están ejecutando los nodos.

Temporizadores

Agrupando los utilizados en la capa Mac y en la física podemos destacar los siguientes:

- *SmacGeneTimer*: Temporizador genérico utilizado tanto para determinar el tiempo de espera de la llegada de la primera planificación como de los mensajes de control CTS o ACK.
- *SmacRecvTimer*: Mediante su uso PhySmac simula el periodo de tiempo en que el nodo tarda en recibir un mensaje.
- *SmacSendTimer*: Idéntico significado que el mensaje anterior, pero con respecto al envío de paquetes.
- *SmacCsTimer*: Duración, calculada en la capa Mac, que el nivel físico está realizando la detección de portadora.
- *SmacCounterTimer*: Temporizador que marca el fin de los diferentes periodos de los ciclos de trabajo de las planificaciones que sigue el nodo.

5.3 DSR (Dynamic Source Routing Protocol)

A continuación vamos a comentar la implementación realizada del protocolo DSR, protocolo de enrutamiento por demanda iniciada en el origen. Mediante el desarrollo en C++ de la clase Dsr y su posterior ejecución en los módulos simples Net de los nodos de la red se consigue que dichos dispositivos ejecuten DSR como protocolo de nivel de red.

Seguiremos el mismo esquema de desarrollo que con S-MAC. En primer lugar comentaremos las características principales del protocolo. No entraremos a nombrar todas y cada una de las funciones que incorpora la clase, pues dicha información se encuentra desarrollada y recopilada en la documentación suministrada para tal efecto.

En el siguiente apartado hablaremos de las estructuras de datos utilizadas. Al igual que ocurría con el protocolo antes expuesto, la buena gestión de estas estructuras es fundamental para un correcto funcionamiento del protocolo y una mejora del rendimiento del mismo.

Y, por último, enumeraremos los principales mensajes utilizados por DSR.

5.3.1 Diseño

El comienzo de la ejecución de la clase Dsr comienza con la función propia de OMNeT++ *initialize()*. En este caso hacemos uso de la versión simple, es decir, Dsr tendrá una única fase de inicialización.

Debido a que no se ha implementado protocolo de nivel de aplicación, una vez se ha completado la inicialización de las variables propias del protocolo determinamos cuales van a ser los nodos fuentes de la simulación. Para ello en el archivo de configuración desarrollado para el protocolo DSR, *dsr.ini*, indicaremos previamente cuantas fuentes tendremos en el simulador. Dicho número deberá ser menor o igual al número de nodos.

Si el dispositivo resulta ser nodo fuente (dicha condición la determinará la función *nodeSource()* utilizando para ello variables tales como el identificador del nodo, valores de probabilidad, etc.) generará el automensaje de comienzo de ejecución START_SEND, exclusivo de los dispositivos fuente. Cuando el nodo reciba dicho mensaje generará un temporizador que le indicará el momento en que debe comenzar el proceso de envío de paquetes de datos. El temporizador recibe el nombre de NEW_MESSAGE.

El código generado está preparado para concluir la simulación cuando alguno de los niveles de red de los nodos fuentes ha conseguido enviar con éxito un determinado número de mensajes. Dicho número lo establece la variable *num_menssage* la cual inicializamos en el fichero de configuración del protocolo. Además hemos añadido aleatoriedad a la producción de mensajes mediante el uso de la función *probSend()*. Dicha función calcula un valor de probabilidad, bien siguiendo una distribución normal o bien una uniforme según el valor que tome la variable *distributions*. Si el resultado de ejecutar *probSend()* es menor o igual que la constante PROBABILITY, entonces el nodo en cuestión comenzará el proceso de emisión de paquetes. Si es mayor, generará un nuevo temporizador NEW_MESSAGE y esperará a que éste se cumpla para inicializar nuevamente el proceso.

Una vez el nodo determina la dirección destino del paquete de datos, busca en su caché de rutas si posee un camino para alcanzarla. Si dicha ruta existe realiza el envío utilizando para ello las funciones determinadas por el protocolo de la capa

inferior, la de enlace de datos. En nuestro caso podremos hacer uso del protocolo CSMA o del S-MAC.

Ahora bien, si en la *Route Cache* no encontramos ninguna ruta definida se inicia en el nodo el proceso de Descubrimiento de rutas realizando para ello el envío de un mensaje broadcast del tipo *Route Request*.

A continuación el proceso implementado seguiría las directivas ya explicadas en el *Apartado 2.2.3.2* donde explicábamos el modo de funcionamiento del protocolo DSR:

1. Cuando el mensaje *Route Request* llega a un nodo que no conoce la ruta hacía el destino, éste reenvía el paquete broadcast añadiéndose él a la ruta. Este proceso lo realiza siempre y cuando el número total de nodos de la ruta sea inferior o igual a constante del protocolo NUM_MAX_HOPS.
2. Si el nodo receptor del *Route Request* es el destino o conoce una ruta hacía él, reenvía un mensaje unicast a la fuente del mensaje utilizando como camino la ruta que ha establecido. El mensaje es del tipo *Route Reply*.
3. Cuando el nodo origen recibe el *Route Reply* que esperaba comienza la transmisión del paquete de datos que generó.
4. Una vez ha llegado el mensaje de datos al destino buscado, los nodos comienzan la transmisión de mensajes *Route Ack* para confirmar que el envío se realizó de forma correcta.
5. Si no es así, es decir, si en alguno de los enlaces los nodos detectan que no puede realizarse correctamente el enrutamiento se iniciará la transmisión de paquetes *Route Error*, de tal manera que se pueda informar a la fuente de la imposibilidad de realizar el envío.

A medida que al nodo fuente le van llegando paquetes *Route Reply*, nada nos asegura que la primera de las rutas que procese sea la mejor, es decir, la que tenga un menor número de saltos. Por ello se han implementado dos modos de funcionamiento en la clase Dsr.

Cuando el nodo fuente inicia un proceso de Descubrimiento de rutas genera el temporizador DISCOVERY_TIMER. Según el valor que toma la variable *waitTime*, el tratamiento que se realiza a este temporizador es distinto.

Para *waitTime* igual a 1 el nodo fuente almacenará todas las rutas que vayan llegando hasta que se cumpla el temporizador de descubrimiento. A continuación procesará las rutas utilizando la de menor número de saltos.

Si el valor de la variable es 0, el nodo usará la primera de las rutas que llegue y cancelará el temporizador. El resto de rutas que se vayan recibiendo no se descartarán, se almacenarán en la caché de rutas para un posible uso futuro.

En cualquiera de los casos, si se cumple el temporizador y la cola de almacenamiento de rutas está vacía, se reinicia el proceso de descubrimiento. Cabe destacar que se ha establecido un número máximo de reintentos para el descubrimiento de rutas desde un nodo origen a un destino. Este valor está determinado por la constante MAX_NUM_DISC.

Antes de pasar a comentar las estructuras de datos más importantes utilizadas en la clase Dsr vamos a comentar tres funciones implementadas en el código mediante las cuales se han alcanzado otras características importantes del diseño: *updateRoute()*, *sendingMessage()* y *handelMacNetControl()*.

La primera de ellas surge de la necesidad de implementar en el protocolo DSR el funcionamiento promiscuo de los nodos, es decir, debido a la limitación de procesamiento que caracteriza a los sensores de las WSN's cada vez que se recibe un mensaje el protocolo DSR obtiene la máxima cantidad de información posible que puede aportar un paquete. Para ello, mediante el uso de *updateRoute()*, cada vez que el dispositivo recibe una ruta a la que pertenece obtiene de ella todos los posibles caminos a cualquiera de los nodos que como él la componen. De igual manera, si la información que nos llega de la ruta es que esta es errónea mediante el uso de *updateRoute()* el nodo eliminará de su caché de rutas todas aquellas que posean el enlace defectuoso.

La función *sendingMessage()*, la segunda a comentar, surge por la necesidad de implementar dos características del protocolo. La primera de ellas es la condición de nodos enrutadores que poseen todos los dispositivos de la red. Debido a esto, los nodos pueden tener la necesidad simultánea de enviar diferentes tipos de paquetes. Por ello, cada vez que se genera uno nuevo, independientemente del tipo que sea, éste se almacena en una cola. A continuación, siempre que el estado del nivel de la capa de enlace de datos lo permita, se ejecuta *sendingMessage()*. Esta función realizará el envío del primero de los elementos existentes en la cola, el cual no tiene que ser el que ha provocado su ejecución.

Según el tipo de paquete a transmitir la función realiza también cierto control de errores. Por ejemplo, un nodo fuente no debe enviar un mensaje de tipo *Route Ack*.

Ahora bien, independientemente de paquete a enviar, la función *sendingMessage()* también realiza el incremento del número de retransmisiones que podemos realizar de un paquete.

El control de estas retransmisiones lo realiza la tercera de las funciones a comentar, *handleMacNetControl()*. Según el tipo de paquete de control que llegue realizará un tipo de función u otro:

- MAC_FINISH: Si el nivel Mac nos informa que la transmisión del paquete ha sido realizada y éste es de datos, quitamos de la pila de transmisión el mensaje y lanzamos un temporizador de espera del mensaje de confirmación *Route Ack*. Cabe destacar que el tiempo de duración de este temporizador debe ser proporcional a la posición del nodo en la ruta utilizada para la transmisión de los datos.
- MAC_ERROR: Según los niveles Mac implementados, estos sólo nos informarán de errores en el envío de un paquete si es unicast. Por ello, si el nodo recibe este mensaje comprueba que el número de retransmisiones no se ha superado. Si es así reinicia el proceso ejecutando la función *sendingMessage()*. Ahora bien, si el número de retransmisiones es superior a la constante MAX_NUM_RTX, se descarta el envío.

5.3.2 Estructuras de datos

Para reducir la cantidad de información que debe manejar el protocolo DSR y evitar el envío de mensajes que ya han sido procesados, debido a las características broadcast de algunos de ellos, son varias las estructuras de datos utilizadas. A continuación expondremos una relación de las mismas a la vez que comentaremos los principales criterios de su elección.

routeInfo

Estructura de datos básica de la implementación realizada. En ella se almacena toda la información de lo que Dsr considera una ruta:

- *dstAddr*: Dirección destino.
- *numHops*: Número de saltos de la ruta.
- *vector <int> route*: Vector de número enteros donde se van almacenando de forma progresiva a medida que se construye la ruta los identificadores de los nodos que la forman.
- *vector <int> error*: Vector de dos posiciones donde se almacenan los identificadores de los nodos donde exista un enlace defectuoso. El enlace debe pertenecer a la ruta antes definida.

Para almacenar la ruta se ha utilizado en contenedor tipo vector debido a las características que nos ofrece de acceso directo a cualquier elemento así como la posibilidad de insertar y eliminar elementos de forma rápida al final del mismo.

Cada vez que a lo largo del documento hemos hablado de que en los mensajes se almacenaba la ruta a seguir, nos referíamos a esta estructura.

bufferAckTx

Contendor tipo lista donde se almacenan los paquetes de datos que el nivel de red ha enviado pero que aún están pendientes de confirmación mediante la recepción de un mensaje *Route Ack*.

Debido a que las listas nos permiten una rápida inserción y eliminación de elementos en cualquiera de las posiciones de las que consta hemos hecho uso de ellas, pues no sabemos a priori cual de los mensajes que un nodo ha enviado va a recibir antes una confirmación de recepción.

AckInfo

Clase derivada de cObject. Realmente la lista *bufferAckTx* no almacena el paquete de datos en sí, sino una instancia a un objeto de esta clase que contiene los siguientes parámetros:

- *DsrNetworkPacket * pkt:* Puntero al paquete de datos del cual pedimos confirmación.
- *list <AckInfo *>::iterator it*: Iterador donde almacenamos la posición en la lista donde se ha almacenado el objeto *AckInfo*.
- *DsrAckTimer * timer:* Puntero al temporizador generado por el nodo para controlar la llegada del mensaje *Route Ack*.

Si se cumple el temporizador y el mensaje de confirmación no ha llegado mediante el iterador almacenado en el objeto podremos determinar que paquete de la lista es el que está sin conformar. A continuación podremos o reenviarlo o eliminarlo si hemos superado el número de retransmisiones.

En cambio, si lo que llega es un mensaje de confirmación *Route Ack* mediante el puntero al temporizador podremos cancelarlo.

arrayRequest

Debido a las características broadcast de los mensajes *Route Request* puede darse el caso de que los nodos reciban mensajes de este tipo que ya han sido contestados por ellos. Para solucionar esta posible generación de tormentas de broadcast se ha incorporado a la clase el vector *arrayRequest* cuya longitud es igual al número de nodos de la red. La estructura de datos que almacenamos en las posiciones del vector, la clase DiscInfo, será explicada a continuación.

Hemos hecho uso en este caso de un vector debido, como ya comentamos antes, a la facilidad para acceder a una determinada posición del mismo. En este

caso, las posiciones son fijas, pues cada nodo de la red ocupa una situación en el vector igual a su identificador.

DiscInfo

Clase derivada de cObject. El vector *arrayRequest* antes comentado almacena punteros a instancias de esta clase. Define los siguientes parámetros:

- *DiscoveryAckTimer * timer:* Puntero al temporizador generado por el nodo para controlar la llegada de mensajes *Route Request*. Por cada uno de los mensajes *Route Request* contestado por el dispositivo, se generará un temporizador que almacenaremos en la correspondiente posición del vector.
- *bool flag:* Variable booleana que nos indica si hemos recibido con anterioridad un mensaje de descubrimiento procedente de este nodo.

Cuando se cumple el temporizador inicializamos los valores del objeto, estableciendo el valor de la variable *flag* a 0. Mientras el temporizador no se cumpla, el nodo no contestará *Route Request* procedentes de ese nodo. Cabe destacar que un determinado dispositivo no puede iniciar un proceso de descubrimiento antes de haber concluido otro, por lo que no se puede caer en el error de no contestar mensajes de búsqueda de rutas hacía distintos destinos procedentes del mismo nodo, siempre y cuando estos temporizadores y los de descubrimiento estén sincronizados.

routeCache

Lista mediante la cual creamos la *Route Cache* del protocolo DSR. Dado que las rutas que almacenaremos tienen tiempo de vida, lanzaremos un temporizador por cada una de las que se creen y pasado un cierto tiempo si no son utilizadas serán borradas, o pueden desaparecer durante su uso y por tanto deben ser eliminadas, hemos hecho uso de un contenedor tipo lista.

Route

Clase derivada de cObject. En la lista *routeCache* almacenaremos instancias a objetos de esta clase, cuyos principales parámetros son los siguientes:

- *routeInfo route:* Estructura del tipo *routeInfo* ya antes comentada. En los parámetros de los que consta almacenaremos toda la información de la ruta pertinente.
- *DsrRouteTimer * timer:* Puntero al temporizador generado por el nodo para controlar el tiempo que una ruta debe permanecer en la caché de rutas.
- *list <Route *>::iterator it*: Iterador donde almacenamos la posición en la lista donde se ha almacenado el objeto *Route.*

Si se produce la llegada de un mensaje *DsrRouteTimer* mediante el uso del iterador almacenado podremos eliminar la ruta de la caché. En cambio, si se decide reutilizar la ruta almacenada, mediante el uso del puntero al temporizador podremos cancelarlo y volver a planificarlo de nuevo. También es posible mediante este proceso eliminar una ruta que ha resultado ser defectuosa.

bufferTx

Contenedor tipo cola donde almacenamos los mensajes que el nodo desea enviar. Mediante su uso aprovechamos las características que nos ofrece a la hora de apilar los mensajes: El primero que entra es el primero en salir (*First In First Out,* FIFO).

neigbDisc

Vector donde almacenamos los procesos de descubrimiento de rutas realizados para cada nodo de la red. Esta estructura de datos solamente la poseen los nodos que actúan como fuentes de mensajes.

bufferDisc

Cola de almacenamiento temporal donde se guardan las rutas que van llegando en los mensajes *Route Reply*. Su uso se centra cuando realizamos simulaciones haciendo uso del tiempo de espera de llegada de *Route Reply.*

5.3.3 Mensajes

Son muchos los mensajes de los que hace uso DSR para completar sus funciones. Por ello haremos únicamente un resume de los más importantes.

NetworkPdu

Si bien, como hemos visto en el desarrollo del protocolo S-MAC, el nivel Mac y el nivel físico tiene que tener perfecto conocimiento de la estructura de los mensajes de ambas capas, en el caso del nivel de red hemos conseguido establecer una total independencia entre capas.

Una vez que el protocolo Dsr ha construido el mensaje correspondiente, independientemente del tipo que sea, genera un paquete del tipo *NetworkPdu* en el cual almacena en la variable *dstAddrMac* la dirección o identificador del siguiente nodo en la ruta y encapsula el mensaje que ha construido.

Cuando este mensaje llega al nivel Mac, éste desencapsula el mensaje procedente del nivel de red, construye su propio paquete con la información dada (la dirección MAC del nodo siguiente) y vuelve a encapsular el mensaje DSR, considerándolo como una única unidad de datos y sin tener en cuenta las cabeceras que lo componen.

DsrNetworkPacket

Tipo de mensaje característico de DSR del que derivan los paquetes que realizan las funciones más importantes del protocolo:

- *RouteReply*
- *RouteRequest*: Añade el identificador de paquete, el cual coincide con el identificador del nodo que genera el mensaje.
- *RouteAck*
- *RouteError*
- *DataNetworkPacket*

Como parámetros que lo forman podemos destacar el tipo de paquete, la estructura *routeInfo* y el número de retransmisiones del mensaje.

DsrTimer

Automensajes generados por el protocolo. Dentro de esta clase podemos encontrar a su vez otra serie que derivan directamente de *DsrTimer*:

- *DsrDiscoveryTimer*
- *DiscoveryAckTimer*
- *DsrAckTimer*
- *DsrRouteTimer*

Capítulo 6
Conclusiones y líneas futuras

6.1 Conclusiones

En los primeros capítulos de este proyecto se realizó un estudio de las redes inalámbricas y en especial de las redes de sensores inalámbricos. Dicho estudio nos mostró los nuevos problemas que estas redes plantean y la necesidad de desarrollar nuevos protocolos específicos para las WSN's que permitan solventar estas dificultades.

A continuación se realizó el estudio de un protocolo de acceso al medio, S-MAC, como una posible solución a las limitaciones energéticas de los dispositivos que constituyen estas redes, además de conseguir gestionar el control de acceso al medio.

Para dar respuesta enrutamiento de los datos en estas redes distribuidas se analizó también el protocolo de nivel de red DSR.

Para poder desarrollar, diseñar y probar estos protocolos nos hemos visto, por tanto, obligados a usar simuladores de redes de sensores.

Por ello tras analizar las herramientas de código abierto disponibles para tal fin se decidió la implementación de un simulador de redes de sensores inalámbricos basado en el entorno OMNeT++.

Se decidió su utilización por estar basado en C++ lo que permitía incorporar al simulador criterios de herencia y polimorfismo, y poder aprovechar las altas prestaciones de modularidad y flexibilidad que caracterizan a este entorno. Además OMNeT++ es una de las herramientas para simulación de redes de comunicaciones más usadas en ámbitos de investigación y académicos, y se deseaba la utilización del simulador desarrollado como herramienta docente.

Tras realizar todos estos estudios previos se ha implementado un entorno de simulación de redes de sensores inalámbricos en dos fases. En la primera de ellas se ha desarrollado un simulador genérico de redes inalámbricas. Nos hemos centrado en lograr una arquitectura jerárquica, modular y con funciones bien definidas en los distintos módulos de la red.

La obtención del simulador de WSN's en sí se ha conseguido en la segunda de las fases. Para ello hemos implementado las clases que contienen las funciones propias de los protocolos antes estudiados y se han añadido al entorno.

Por último se ha generado una amplia documentación en formato HTML para facilitar una futura extensión del simulador mediante la incorporación de nuevos módulos, protocolos y características.

6.2 Líneas futuras

El continuo desarrollo de las aplicaciones de las redes de sensores ha provocado su aplicación en ámbitos totalmente inimaginables hasta hace unos años.

Ello ha originado que las líneas de investigación de los simuladores para este tipo de redes estén todavía abiertas y en continua evolución.

Además, teniendo en cuenta los estudios previos realizados, vemos como los problemas a solventar en el ámbito de las WSN's son múltiples y muy variados. Por ello a continuación mostramos una relación de las posibles mejoras y evoluciones que podrían realizarse al simulador en fases futuras de desarrollo, de tal manera que se pueda adaptar a estos nuevos cambios:

- Incorporar al modelo de nodo del simulador las características de hardware de los sensores físicos que formarán la red.
- Añadir los modelos de entorno y canales al sensor, siguiendo el esquema de módulos mostrado en la *Figura 3.2*.
- El modelo de nodo desarrollado en el simulador se caracteriza por tener una estructura de capas fijada. Es decir, la pila de protocolos ejecutada en los dispositivos consta de cuatro niveles: aplicación, red, enlace de datos y capa física. La incorporación de nuevas capas implica un conocimiento avanzado de la estructura del simulador, complicando con ello la evolución del mismo. Por ello se propone independizar las estructuras prefijadas mediante la modificación del código fuente de tal manera que las capas, puertas y conexiones se identifiquen por niveles y no por una nomenclatura heredada del modelo OSI.
- Aunque forman parte del nodo, las capas de movilidad y energía no están desarrolladas. Por tanto, agregar modelos de movimiento y de consumo energético siguiendo patrones marcados por dispositivos reales sería otra posible línea de desarrollo del simulador.
- Es necesario realizar optimizaciones del código para poder simular redes grandes (miles de nodos). En particular, optimizar el cálculo de la propagación radio de paquetes.
- Es necesaria una revisión de las interfaces y de la estructura en general para simplificar el entorno de simulación. Este desarrollo ha sido una primera versión que se refinará a medida que se vaya utilizando.

Acrónimos

- ABR: Associative-Based Routing
- ACK: Acknowledgment
- AMPS: Advanced Mobile Phone System
- AODV: Ad Hoc On-Demand Distance Vector Routing
- AP: Access Point
- BER: Bit Error Rate
- BS: Base Station
- CA: Collision Avoidance
- CRA: Contention Resolution Algorithm
- CRC: Cyclic Redundancy Check
- CSMA: Collision Avoidance Multiple Access
- CTS: Clear to Send
- D-AMPS: Digital-AMPS
- DDSS: Direct Sequence Spread Spectrum
- DFWMAC: Distributed Foundation Wireless MAC
- DSR: Dynamic Source Routing
- DSDV: Destination-Sequenced Distance -Vector Routing Protocol
- DTMP: Disposable Token MAC Protocol
- ETSI: European Telecommunication Standards Institute
- EY-NPMA: Elimination Yield – Non-Preemptive Priority Multiple Access
- FEC: Forward Error Correcting
- FIFO: First In First Out
- FHSS: Frequency Hopping Spread Spectrum
- GSM: Global System for Mobile Communications
- GUI: Graphical User Interface
- IP: Internet Protocol
- ISM: Industrial, Scientific and Medical
- ISMA: Idle Sense Multiple Access
- ISO: International Organization for Standardization
- JVM: Java Virtual Machine
- KDE: K Desktop Environment
- LMDS: Local Multipoint Distribution Service
- MAC: Medium Access Control

- MAN: Metropolitan Area Network
- MANET: Mobile Ad-hoc Network
- MMDS: Multi-channel Multipoint Distribution
- NAV: Network Allocation Vector
- OFDM: Orthogonal Frequency Division Multiplexing
- OMNeT++: Objective Modular Network Testbed in C++
- QoS: Quality of Service
- OSI: Open Systems Interconnection
- RAMA: Resource Auction Multiple Access
- RAP: Randomly Addressed Polling
- RRA: Random Reservation Access
- RTS: Request to Send
- S-MAC: Sensor Medium-Access Control
- SSF: Scalable Simulation Framework
- TCP: Transmission Control Protocol
- TDMA: Time Division Multiple Access
- TORA: Temporary Ordered Routing Algorithm
- UDP: User Datagram Protocol
- UMTS: Universal Mobile Telecommunications System
- WAN: Wide Area Network
- WLAN: Wireless Local Area Network
- WLL: Wireless Local Loop
- WN: Wireless Network
- WPAN: Wireless Personal Area Network
- WRP: Wireless Routing Protocol
- WSN: Wireless Sensor Network

Referencias

[1] OMNeT++: http://www.omnetpp.org/

[2] Kdevelop: http://www.kdevelop.org/

[3] Doxygen: http://www.doxygen.org/

[4] C. Siva Ram Murthy and B.S. Manoj, “Ad hoc wireless networks, architectures and protocols”, Chapter 2: Wireless LAN’s and PAN’s. Ed. Prentice Hall, 2004

[5] C. Siva Ram Murthy and B.S. Manoj, “Ad hoc wireless networks, architectures and protocols”, Chapter 3: Wireless WAN’s and MAN’s. Ed. Prentice Hall, 2004

[6] Ajay Chandra V. Gummalla, John O. Limb, “Medium Access Control Protocols”, Georgia Institute of Technology

[7] C. Siva Ram Murthy and B.S. Manoj, “Ad hoc wireless networks, architectures and protocols”, Chapter 12: Wireless Sensor Networks. Ed. Prentice Hall, 2004

[8] I.F. Akyildiz, W. Su, Y. Sankarasubramaniam, E. Cayirci, “Wireless sensor networks: a survey”, Broadband and Wireless Networking Laboratory, School of Electrical and Computer Engineering, Georgia Institute of Technology, Atlanta, GA 30332, USA

[9] Wei Ye, John Heidemann, Deborah Estrin, “An Energy-Efficient MAC Protocol for Wireless Sensor Networks”, USC/ISI Technical Report ISI-TR-543, September 2001

[10] Charles E. Perkins, “Ad hoc networking”. Ed. Addison-Wesley, 2001

[11] E. Egea-López, J. Vales-Alonso, A. S. Martinez-Sala, P. Pavón-Mariño, J. García-Haro, ``Simulation tools for wireless sensor networks'', Proc. Int. Symposium on Performance Evaluation of Computer and Telecommunication Systems (SPECTS 2005), Philadelphia, PA, July 2005. To appear

[12] The Network Simulator, NS–2: http://www.isi.edu/nsnam/ns/

[13] MIT Object Tcl: http://bmrc.berkeley.edu/research/cmt/cmtdoc/otcl

[14] C. Intanagonwiwat, R. Govidan, D. Estrin, J. Heidemann, F. Silva, “Directed diffusion for wireless sensor networking” IEEE/ACM Transactions on Networking, vol. 11, issue 1, pp. 2–16, February 2003

[15] S. Park, A. Savvides, M. B. Srivastava. “SensorSim: A Simulation Framework for Sensor Networks.” In Proc. ACM Modeling, Analysis and Simulation of Wireless and Mobile Systems (MSWiM 2000), Boston, MA, pp. 104–111, August 2000

[16] NRL’s Sensor Network Extension to NS-2: http://nrlsensorsim.pf.itd.nrl.navy.mil/

[17] J-Sim: http://www.j-sim.org

[18] Jacl, Java implementation of Tcl8.x: http://www.tcl.tk/software/java

[19] NCTUns 2.0 Network Simulator and Emulator: http://nsl.csie.nctu.edu.tw/nctuns.html

[20] R. Barr, Z. J. Haas, R. van Renesse, “JiST: Embedding Simulation Time into a Virtual Machine.” In Proc. 5th EUROSIM Congress on Modeling and Simulation, Paris, France, September 2004

[21] Jython: http://www.jython.org

[22] Global Mobile Information Systems Simulation Library (GloMoSim): http://pcl.cs.ucla.edu/projects/glomosim/

[23] Scalable Simulation Framework (SSF): http://www.ssfnet.org

[24] Ptolemy II. Heterogeneous model and design: http://ptolemy.eecs.berkeley.edu/ptolemyII

[25] MICA Motes: http://www.xbow.com

[26] P. Levis, N. Lee, M. Welsg, D. Culler, “TOSSIM: Accurate and Scalable Simulation of Entire TinyOS Applications.” In Proc. 1st ACM Int. Conf. Embedded Networked Sensor Systems (SenSys), Los Angeles, CA, pp. 126–137, 2003

[27] TinyOS: Open-source operating system for wireless embedded sensor networks: http://www.tinyos.net

[28] L. Girod, J. Elson, A. Cerpa, T. Stathopoulos, N. Ramanathan, D. Estrin, “EmStar: A software Environment for Developing and Deploying Wireless Sensor Networks.” In Proc. USENIX 2004, Boston, MA, pp. 283–296, 2004

[29] J. Polley, D. Blazakis, J. McGee, D. Rusk, J. S. Baras, M. Karir, “ATEMU: A Fine-grained Sensor Network Simulator.” In Proc. 1st IEEE Int. Conf. Sensor and Ad-hoc Communication Networks (SECON’04), Santa Clara, CA, October 2004

[30] S. Sundresh, W. Kim, G. Agha, “SENS: A Sensor, Environment and Network Simulator”. In Proc. 37th ACM Annual Symposium on Simulation, Washington, DC, pp. 221, 2004

[31] PROWLER: Probabilistic Wireless Network Simulator: http://www.isis.vanderbilt.edu/projects/nest/prowler

[32] C. Kelly, V. Ekanayake, R. Manohar, “SNAP: A Sensor-Network Asynchronous Processor”. In Proc. 9th ACM Int. Symposium on Asynchronous Circuits and Systems, Washington, DC, pp. 24, 2003