UNIVERSIDAD CARLOS III DE MADRID

ESCUELA POLITÉCNICA SUPERIOR

PROYECTO FIN DE CARRERA

DESCUBRIMIENTO DE SERVICIOS Y ROUTING EN REDES MANET CON DISPOSITIVOS MAEMO

INGENIERíA DE TELECOMUNICACIÓN

Autor: **Fernando Hoyos Leyva**

Tutora: **Mª Celeste Campo Vázquez**

2 de diciembre de 2009

Agradecimientos

En primer lugar, a mi familia, porque sin ellos seguramente no estaría donde estoy ahora. A mis padres, por darme todo el cariño y el apoyo que se puede recibir. Nunca podré agradeceros lo suficiente lo que habeis hecho por mí. A mi hermana Cristina, por acompañarme durante todos estos años y por ser una de las personas más importantes en mi vida. También a mi tía Maika, por darme tantos ánimos para seguir adelante cuando estaba decaído, siempre has tenido más confianza en mí y en mi trabajo que yo mismo.

A mi tutora Celeste, que siempre me has ayudado y apoyado en la realización de este proyecto, gracias por todos los conocimientos que me has transmitido tanto en lo personal como en lo profesional. Creo que no podría haber tenido un tutor de proyecto mejor.

A mis amigos de toda la vida, por estar ahí después de tantos años y por ser ese tipo de amigos que sólo se pueden contar con los dedos de las manos. A Santi, por ser el compañero perfecto de tantas aventuras vividas. A Lucas, por estar siempre ahí cuando te he necesitado. A Sara, por ser una amiga fiel dispuesta a ayudar en todo lo que puedas. A Raquel, tan espontánea y dicharachera como siempre, no cambies por nada del mundo. Y especialmente a Cris, por hacerme ver que la vida se puede contemplar desde otra perspectiva, llena de matices que nunca antes había conocido, y por el que merece la pena vivir y disfrutar a tu lado.

A mis amigos de la universidad, que después de compartir tantas horas de clases y prácticas habéis llegado a ser más que simples compañeros. Ha sido un placer conoceros a todos y pasar tan buenos momentos con vosotros. Sobre todo a Rubén, por hacer más fácil todos estos años de duro trabajo, eres una de las personas que más admiro y estoy completamente seguro de que en el futuro llegarás a ser un excelente investigador.

A mis amigos Erasmus de Suecia, que más que amigos habéis sido como unos hermanos para mí. Gracias por hacer de ese año en Göteborg uno de los mejores de mi vida, siempre lo recordaré con un cariño especial.

En definitiva, a todas aquellas personas que a lo largo de mi vida me han animado, apoyado y ayudado a intentar llegar a ser mejor persona.

Gracias a todos.

Índice general

Índice de figuras

Índice de cuadros

Parte I

Introducción

Capítulo 1

Introducción

En los últimos años la popularidad de los dispositivos móviles ha ido creciendo exponencialmente en nuestra sociedad (portátiles, PDAs, móviles, Internet Tablets, etc.). Esto se debe principalmente a la movilidad y autonomía que ofrecen al usuario, pudiendo ser utilizadas sin tener que estar obligatoriamente en un lugar fijo. Con el auge de Internet en la Sociedad de la Información se demuestra que una de las principales motivaciones de las personas es la de conocer e intercambiar información con otras. Por ello hoy en día es imprescindible la tecnología que permite que varios dispositivos móviles puedan comunicarse entre sí, como por ejemplo Bluetooth, Wi-Fi, WiMax, 3G, etc. Estas tecnologías inalámbricas dan una libertad e independencia que nunca podrán ser proporcionadas por las conexiones de cable, aunque también es cierto que son algo más limitadas en otros aspectos (duración de la batería, mayor complejidad, menor ancho de banda, mayores amenazas en la seguridad, etc.).

1.1. Visión general

Este proyecto se centra en un tipo de redes móviles denominadas MANET *(Mobile Ad-hoc NETworks)*, que se caracteriza principalmente por ser una red descentralizada (no existe ningún nodo central o punto de acceso fijo), donde todos los nodos que participan en la red son considerados iguales. Debido a las características propias de los dispositivos que forman parte de este tipo de redes, como restricción de ancho de banda y energía (por el uso de baterías), la mayoría de protocolos y aplicaciones utilizados habitualmente deben ser modificados y optimizados. Este proyecto trata especialmente el protocolo de routing y el descubrimiento de servicios en redes MANET.

Probablemente el protocolo de routing sea uno de los protocolos más importantes para ser optimizado, ya que constantemente utiliza ancho de banda para que los dispositivos móviles conozcan cuáles son los otros nodos que componen la red (topología de red) y así poder comunicarse entre ellos. Pero aunque cada dispositivo sepa qué otros nodos existen en la red, sin una aplicación que diga los servicios que ofrece cada nodo o busque aquellos que necesita el dispositivo (conexión a Internet, servicio de impresora, manos libres, etc.), la funcionalidad de la red no está completa (ver Figuras 1.1 y 1.2). Para ello se utilizan los protocolos de descubrimiento de servicios.

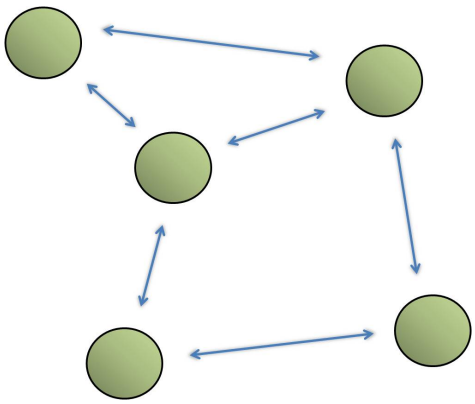

Figura 1.1: Topología de red sin descubrimiento de servicios

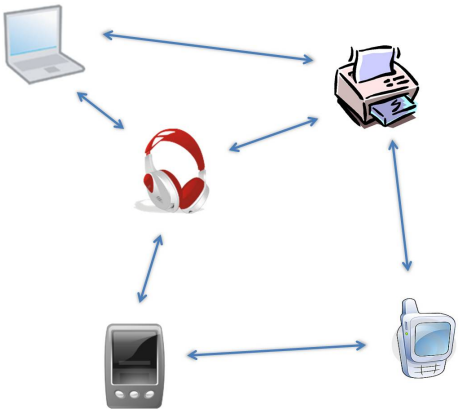

Figura 1.2: Topología de red con descubrimiento de servicios

En nuestro caso comprobaremos si las optimizaciones realizadas por los protocolos anteriormente mencionados son adecuados para ser utilizados en redes MANET. En el caso del protocolo de routing estudiaremos el protocolo OLSR [1] (*Optimized Link State Routing*) y en el caso del descubrimiento de servicios, el objeto de estudio será el protocolo SDM [52] (*Service Discovery Mechanism*).

Para comprobar si las optimizaciones planteadas funcionan, nos centraremos más en su implementación y uso en dispositivos reales (y más próximos al usuario final) que en su simulación a través de un simulador de red. El dispositivo real utilizado es el terminal de Nokia N800 Internet Tablet. Una de las principales características de este terminal es la utilización de Maemo, una plataforma de desarrollo de software abierto desarrollada por Nokia en colaboración con otros proyectos de software abierto tales como Linux, Debian, GNOME, etc. Esto hace que podamos desarrollar e incluir nuestras propias aplicaciones en el terminal, tanto la implementación del protocolo de routing como el mecanismo de descubrimiento de servicios.

1.2. Objetivos

Este proyecto se encuadra en la línea de investigación de la tesis doctoral de Maribel Vara [52] para el diseño de un nuevo protocolo de descubrimiento de servicios específico para su utilización en redes MANET. Este protocolo, denominado SDM (*Service Discovery Mechanism*), tiene la particularidad de estar integrado en un protocolo de routing (OLSR) en lugar de a nivel de aplicación. El objetivo principal de este proyecto es la implementación en un dispositivo real del protocolo SDM, pues aunque en general la mayoría de las prestaciones que se especifican en el diseño de un protocolo se pueden comprobar con simulaciones, las medidas realizadas en dispositivos reales son más fiables y precisas, además de poderse utilizar dicha implementación en un futuro para aplicaciones reales.

Los objetivos concretos que se han planteado en la realización de este proyecto son los que se enumeran a continuación:

- Estudiar los diferentes tipos de redes existentes que utilizan la tecnología WLAN (estándar 802.11), centrándose principalmente en las características y aplicaciones de las redes MANET (*Mobile Ad-hoc NETworks*).

- Estudiar el estado del arte referente a los protocolos de routing en redes MANET, centrándose principalmente en el protocolo OLSR (*Optimized Link State Routing*) y la implementación utilizada en este proyecto (demonio olsrd).

- Estudiar la plataforma de desarollo de software abierto Maemo con sus principales características y componentes.

- Estudiar el estado del arte referente a los protocolos de descubrimiento de servicios tanto en redes cableadas como en redes MANET, centrándose principalmente en el protocolo SDM (*Service Discovery Mechanism*).

- Implementar el protocolo de descubrimiento de servicios SDM en dispositivos reales como un plugin del demonio olsrd, pudiendo ser ejecutado en sistemas basados en Linux (plataforma de desarrollo Maemo).

- Realizar pruebas con varios dispositivos Maemo (Nokia N800) para comprobar el correcto funcionamiento del protocolo SDM y obtener datos relevantes sobre el consumo de batería, el retardo que se produce al anunciar servicios y el consumo del ancho de banda para diferentes configuraciones.

- Comparar el protocolo SDM con otro protocolo de descubrimiento de servicios denominado Mercury [53] (basado en OLSR al igual que SDM) para contrastar las prestaciones de ambos protocolos y determinar sus principales ventajas e inconvenientes.

- Redactar la documentación y la memoria asociada a este proyecto.

1.3. Esquema del proyecto

La estructura de la memoria es la siguiente:

En el capítulo 2, se introducen las redes MANET (*Mobile Ad-hoc NETworks*), enumerando sus principales características, sus modos de funcionamiento y las situaciones en las que se suele utilizar este tipo de redes. También se describe brevemente el funcionamiento de un protocolo de routing y los diferentes tipos que existen actualmente.

En el capítulo 3, se estudia el protocolo de routing OLSR, su funcionamiento, sus principales características y los diferentes tipos de mensajes que se envían a través de la red. También se mencionan las principales novedades que presenta la última versión de este protocolo (OLSR versión 2).

En el capítulo 4, se presenta la plataforma de desarrollo de software abierto Maemo, describiéndose los principales componentes que posee, como el kernel, las principales librerías de sistema o el framework de aplicaciones.

En el capítulo 5, se introducen los protocolos de descubrimiento de servicios, las principales arquitecturas y modos de funcionamiento, además de las dos líneas de investigación que existen actualmente en los protocolos específicos para redes MANET (implementados a nivel de aplicación o integrados con el protocolo de routing).

En el capítulo 6, se describe el funcionamiento del protocolo SDM, las principales razones por las que está diseñado para ser utilizado en redes MANET y la estructura de los mensajes que un dispositivo envía al resto de la red para anunciar sus servicios.

En el capítulo 7, se describe la implementación del protocolo de descubrimiento de servicios SDM como un plugin para el demonio olsrd, centrándose en su arquitectura y la manera de implementar sus diferentes funciones.

En el capítulo 8, se documentan las diferentes pruebas realizadas para comprobar el correcto funcionamiento y el rendimiento del protocolo de descubrimiento de servicios SDM, además de una comparación con otro protocolo de

descubrimiento de servicios (Mercury).

Finalmente, en el capítulo 9 se resumen las principales conclusiones del proyecto y se proponen sugerencias para futuros trabajos.

Como parte de la memoria hemos incluido una serie de anexos para aclarar y completar algunas de las contribuciones de este proyecto. Así:

En el anexo A, se describe el presupuesto del proyecto incluyendo los honorarios según los baremos orientativos del Colegio Oficial de Ingenieros de Telecomunicación (COIT).

En el anexo B, se presenta el entorno de desarrollo Maemo y las principales aplicaciones que se utilizan para crear un paquete Debian (.deb) instalable en el dispositivo Maemo.

En el anexo C, se incluyen las tablas con todos los valores obtenidos en las pruebas realizadas para medir el rendimiento del protocolo de descubrimiento de servicios SDM.

Parte II

Estado del arte

Capítulo 2

Redes MANET

En la actualidad existen en el mercado numerosas tecnologías inalámbricas para que varios dispositivos se comuniquen entre sí a través de ondas y sin la necesidad de utilizar cable, tales como 3G HSPA [8], WLAN [9] (estándar 802.11) o Bluetooth [10]. También se está invirtiendo una gran cantidad de recursos en la investigación de futuras tecnologías inalámbricas que aumenten enormemente las capacidades de las actuales y se aproximen a las conseguidas por las tecnologías de cable. Ejemplos de este tipo son WiMAX [11] o LTE [12] (*Long Term Evolution*).

La mayoría de estas tecnologías inalámbricas se basan en el principio de una comunicación punto a punto, donde los nodos móviles se comunican directamente con un punto de acceso centralizado (que suele ser fijo y conectado a otras redes a través de cable, usualmente Internet). Pero una red MANET [7] *(Mobile Ad-hoc NETworks)* se basa en una red descentralizada y además multisalto, es decir, la información puede pasar por varios nodos antes de llegar a su destino si la distancia entre origen y destino es muy grande. De esta manera los nodos intermedios actúan como routers, reenviando la información en cada salto a través de la red.

2.1. Tecnología WLAN (estándar 802.11)

Aunque las redes MANET no están restringidas a una tecnología en concreto, se suelen utilizar interfaces WLAN para montar este tipo de redes (ya sea en cualquiera de las versiones del estándar IEEE 802.11 [13] - 802.11a, 802.11b, 802.11g o el más moderno 802.11n). Las características de la tecnología WLAN en cobertura y limitación de energía hacen que sea la más adecuada para este

tipo de redes. Otras tecnologías como Bluetooth o 3G HSPA también podrían ser utilizadas, pero mientras que la primera posee un rango de transmisión muy pequeño (unas 10 veces menor que WLAN), la segunda sólo permite el modo de acceso a un nodo centralizado (contrario a la filosofía de las redes MANET)

Las dos configuraciones más utilizadas en las comunicaciones con dispositivos WLAN son:

- **Modo infraestructura**

  La red móvil consiste en un conjunto de dispositivos móviles que se conectan a un punto de acceso (típicamente colocado en un lugar fijo y conectado por cable a otras redes). Esta configuración de red móvil es de tipo centralizado, ya que el punto de acceso es el dispositivo por el que pasa todo el tráfico que intercambian los diferentes dispositivos móviles que forman la red (si el punto de acceso falla, la red entera cae).

  Un ejemplo del modo infraestructura es la red utilizada en cualquier hogar para que un dispositivo móvil (portátil, PDA, etc.) tenga acceso a Internet a través de un router inalámbrico, que suele estar conectado por cable Ethernet [14] al proveedor de servicios ISP para tener acceso a Internet (ver Figura 2.1).

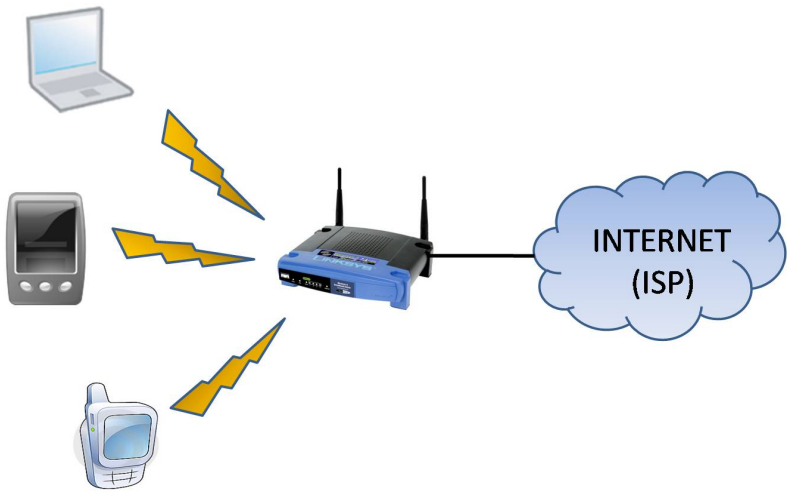

Figura 2.1: Modo infraestructura en una red móvil

- **Modo ad-hoc**

  Este modo de configuración es el utilizado en las redes MANET, ya que se caracteriza en que no existe ningún dispositivo que centralice todo el

tráfico generado en la red, sino que éste es distribuido indistintamente por todos los dispositivos. Por lo tanto los diferentes nodos establecen comunicación directamente unos con otros sin la necesidad de utilizar para ello un punto de acceso. Un ejemplo del modo ad-hoc es el intercambio de información entre móviles (imágenes, tonos, etc.) o la conexión de éste a un portátil con un manos libres (ver Figura 2.2).

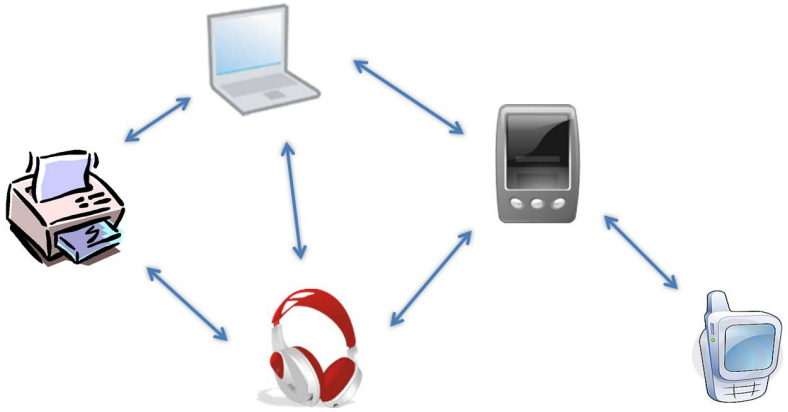

Figura 2.2: Modo ad-hoc en una red móvil

2.2. Redes en modo ad-hoc

Aunque la mayoría de redes que se despliegan son de modo infraestructura (que requieren centralizar todo el tráfico en un punto de acceso), existen diferentes escenarios y aplicaciones donde lo más conveniente (o lo único) que se puede desplegar es una red en modo ad-hoc. Ejemplos importantes de redes móviles ad-hoc pueden ser comunicaciones militares, operaciones de rescate, conferencias, redes de sensores, etc.

Estos ejemplos se caracterizan en que no pueden depender de una conectividad organizada y centralizada, ya que la topología de la red es muy diversa y dinámica (cambia muy rápidamente), y los dispositivos utilizados están restringidos por su limitada capacidad de ancho de banda y energía. Además la red tiene que ser fácilmente desplegable y auto-configurable, pudiendo ser una red autónoma o con acceso a redes externas como Internet. También hay que tener en cuenta que las redes móviles son más propensas a sufrir ataques en su seguridad [23] (spoofing, eavesdropping, DoS, etc.) que las redes fijas de cable, aunque al ser las redes MANET descentralizadas proporcionan algo más

de robustez ya que no existe un punto crítico de fallo (como por ejemplo un servidor centralizado).

En principio el modo ad-hoc sólo permite la transmisión de paquetes a aquellos dispositivos que están en su área de cobertura, pero las aplicaciones mencionadas anteriormente necesitan conectarse con otros nodos que están situados a una distancia mayor. Para ello se puede utilizar a dispositivos vecinos como routers para que reenvíen la información al destinatario. Esta capacidad de las redes MANET de utilizar los dispositivos para retransmitir la información y así alcanzar dispositivos más lejanos se denomina multisalto (debido a que la información da varios saltos desde que sale del origen hasta que llega al destino).

Tomando como ejemplo la topología de red de la Figura 2.2, el portátil quiere enviar un mensaje al teléfono móvil, pero éste se encuentra fuera de su área de cobertura. Si esta red soporta la opción multisalto, el portátil podrá intercambiar información con el teléfono móvil a través de la PDA, que en este caso actuará como router reenviando la información. Para que una red MANET tenga la capacidad de multisalto, es imprescindible que exista un protocolo de routing que establezca la topología de red, definiendo aquellos dispositivos que actuarán como routers y las rutas de tráfico para saber a quién se deben enviar los paquetes para que la información llegue correctamente a su destino.

2.3. Protocolos de routing en redes MANET

La función principal de un protocolo de routing es la de establecer y mantener las rutas por donde viaja toda la información que genera la red, es decir, que cada dispositivo sepa a qué otro nodo de la red dentro de su rango de transmisión debe enviar cada paquete. Para ello cada dispositivo posee una tabla de routing donde aparece dicha información, siendo la misión del protocolo la de crear y mantener esas tablas correctamente, definiendo así la topología de la red. Estos protocolos especifican la estructura de los mensajes que se intercambian todos los dispositivos para que conozcan la topología de la red y de esta manera actualizar la tabla de routing. Ejemplos típicos de protocolos de routing para redes IP fijas son RIP [21] (*Routing Information Protocol*) y OSPF [22] (*Open Shortest Path First*).

Al contrario que en las redes fijas (o móviles en modo infraestructura), todos los dispositivos pueden actuar como routers, por lo que cada emisor tiene una ruta específica por la que enviar los paquetes a cada destino de la red MANET. Por esta razón los protocolos de routing implementados habitualmente en redes

fijas no son apropiados para redes MANET. Además hay que tener en cuenta otras propiedades que hacen el diseño del protocolo de routing bastante más complejo, ya que la comunicación se produce de forma distribuida (y no de forma centralizada vía routers como en las redes fijas), la topología de una red de estas características cambia con mucha rapidez y el ahorro de energía es fundamental (las baterías proporcionan energía por un intervalo de tiempo limitado).

Las características más importantes de un protocolo de routing en redes MANET son:

- Creación y organización de rutas de tráfico de forma automática y autónoma.
- Capacidad de multi-salto, donde las rutas no presenten bucles infinitos.
- Mantenimiento de la topología de red hasta en las redes más dinámicas (con muchos cambios en poco tiempo).
- Convergencia rápida de la red.
- Mínima sobrecarga en el tráfico de la red.
- Que se pueda utilizar en grandes redes con muchos dispositivos (escalabilidad).

En redes MANET existen tres tipos diferentes de protocolos de routing para conseguir todas las funcionalidades enumeradas anteriormente: los denominados protocolos reactivos, los proactivos y los híbridos, diferenciándose entre sí sobre todo en la forma de mantenimiento de las rutas de tráfico.

2.3.1. Protocolos reactivos

La filosofía de los protocolos de routing reactivos es la de transmitir la información de la topología de red bajo demanda, es decir, solamente cuando se quiere iniciar una comunicación con un nodo que no tiene una ruta establecida. En este caso el protocolo de routing intentará establecer tal ruta para interconectar ambos dispositivos.

Para explicar el funcionamiento de este tipo de protocolo de routing tomaremos como referencia la topología de red de la Figura 2.3. Si por ejemplo el nodo A quiere establecer una comunicación con el nodo F, enviará una petición

de ruta (mensaje RREQ - *Route Request*) a sus vecinos (nodos B y C) preguntándoles si son el nodo F. Como ninguno de ellos es el nodo F, reenviarán la misma petición a sus vecinos y así sucesivamente hasta que la petición le llega al nodo F.

En ese momento (ver Figura 2.3), el nodo F enviará al nodo E una respuesta (mensaje RREP - *Route Reply*) confirmando que es el nodo F. De esta manera el nodo E sabe que el nodo F es uno de sus vecinos y reenvía la respuesta al nodo C, averiguando así que la manera de llegar al nodo F es a través del nodo E. Por último, la respuesta llega por fin al nodo A por parte del nodo C. A partir de este momento (y mientras la ruta tenga validez) el nodo A sabe que si quiere comunicarse con el nodo F tiene que hacerlo a través del nodo C (en este tipo de protocolos los nodos no conocen la topología entera de la red, solamente a quién deben enviar el paquete).

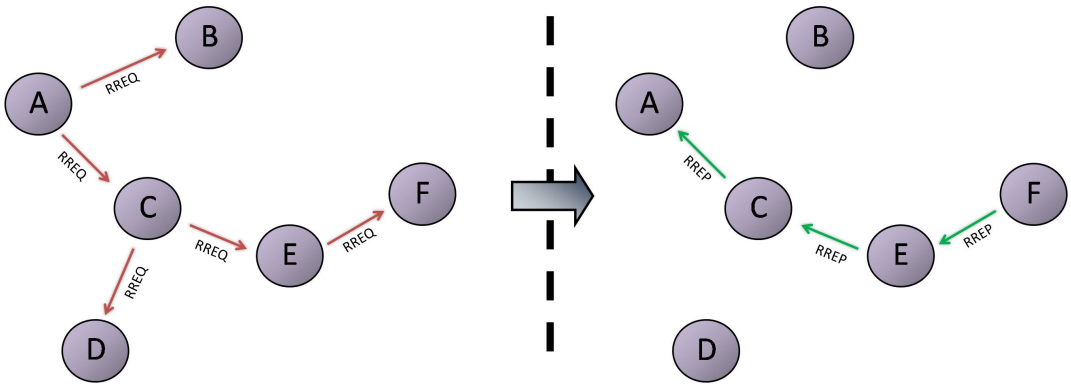

Figura 2.3: Petición de ruta del nodo A al nodo F y su correspondiente respuesta

La ventaja de este tipo de protocolos es que sólo utilizan ancho de banda cuando es estrictamente necesario (ya que mientras la ruta tenga validez no se envían peticiones de ruta). Entre los inconvenientes, cabe destacar la existencia de un retraso importante cuando se quiere establecer una comunicación nueva y la sobrecarga en la red cuando se realiza una petición de ruta (ya que se transmite por inundación a toda la red). Ejemplos de protocolos de routing reactivos en redes MANET son AODV [24] (*Ad-hoc On Demand Distance*), DSR [26] (*Dynamic Source Routing*) y DYMO [27] (*Dynamic MANET On-demand*).

2.3.2. Protocolos proactivos

Mientras que los protocolos de routing reactivos esperan a que algún nodo quiera establecer una nueva conexión para descubrir la ruta entre origen y destino, los protocolos de routing proactivos están constantemente buscando los diferentes caminos por donde se puede enviar la información, tomando la iniciativa del intercambio de mensajes entre los distintos dispositivos que conforman la red para tener su topología lo más actualizada posible.

En este tipo de protocolos todos los nodos conocen la topología entera de la red, sabiendo en todo momento cuál es el mejor camino para que la información llegue al destino (por ejemplo si existen varios caminos elegir el más corto o en el que se produzcan menos saltos).

Entre los protocolos proactivos en redes MANET más utilizados se encuentran el TBRPF [28] (*Topology Dissemination Based on Reverse-Path Forwarding*), el FSR [29] (*Fisheye State Routing*) y el OLSR [1] (*Optimized Link State Routing*). Este último protocolo es descrito en profundidad en el capítulo 3.

La ventaja de este tipo de protocolos de routing es que todas las rutas son mantenidas todo el tiempo, descubriendo cada poco tiempo la existencia de una nueva ruta. De esta manera no existe ningún retraso en las comunicaciones y las rutas siempre están disponibles para el intercambio de información. Por el contrario, el precio a pagar por tener siempre actualizado la topología de red de todos los nodos es una constante sobrecarga en el tráfico de la red debido al continuo intercambio de mensajes entre los diferentes dispositivos que la conforman.

2.3.3. Protocolos híbridos

Los protocolos híbridos combinan las características de los protocolos reactivos y proactivos. El mayor ejemplo de este tipo de protocolos es el ZRP [30] (*Zone Routing Protocol*). Su funcionamineto consiste en dividir la topología de la red en diferentes zonas, utilizando un tipo de protocolo determinado según convenga, basándose en las ventajas e inconvenientes que presenta cada uno.

Si el emisor y el destino se encuentran dentro de la misma zona (rutas locales), se utiliza un protocolo proactivo (cualquiera de los descritos en la sección 2.3.2) para que no haya un retraso inicial en las comunicaciones locales. En cambio, si el emisor y el destingo están en diferentes zonas (rutas globales), se utiliza un protocolo reactivo (cualquiera de los descritos en la sección 2.3.1), eliminando así la necesidad de tener que guardar la topología completa de la

red. Además, para controlar el tráfico entre las diferentes zonas dentro de la red MANET, el protocolo ZRP define una técnica denominada BRP (*Bordercast Resolution Protocol*), que difunde las peticiones de ruta (*Route Request*) a las demás zonas cuando el destinatario se encuentra fuera de la zona del emisor.

Capítulo 3

Protocolo OLSR

OLSR [1] (*Optimized Link State Routing*) es un protocolo de routing diseñado para redes móviles ad-hoc, especialmente cuando dichas redes son grandes y densas (con muchos terminales disponibles en la red). Entre las características principales de OLSR se encuentra su capacidad de multi-salto, su proactividad en el momento de encontrar una ruta y la mejora en el ancho de banda para dispositivos móviles. El protocolo OLSR se basa principalmente en el clásico algoritmo de estado de enlace (cuyo máximo representante es el protocolo OSPF), adaptándolo a las redes móviles ad-hoc.

La principal diferencia con este algoritmo es la forma de reenviar los mensajes que se intercambian los dispositivos para conocer la topología de la red. En el protocolo OSPF, cuando un dispositivo quiere dar a conocer alguna información, inunda la red enviando mensajes de broadcast a todos sus vecinos, éstos lo reenvían a sus respectivos vecinos y así sucesivamente hasta que llega a todos los rincones de la red.

En cambio en OLSR, en lugar de reenviar un mensaje a todos los nodos que se encuentran en el área de cobertura, sólo se reenvía a un conjunto de nodos estratégicamente colocados llamados MPR (*Multipoint Relays*) capaces de llegar ellos solos al mayor número de nodos posible. Si éstos a su vez reenvían el mensaje a sus respectivos nodos MPR, se conseguirá que todos los nodos de la red reciban el mensaje de broadcast habiéndose enviado un menor número de mensajes con el consiguiente ahorro de tráfico (ancho de banda) y de energía, parámetros críticos en las redes móviles ad-hoc. Esta característica del protocolo OLSR será explicada detalladamente en el apartado 3.1.2.

Aunque el protocolo OLSR se puede utilizar para cualquier tipo de red móvil ad-hoc, su diseño es especialmente adecuado para redes grandes y densas con

muchos dispositivos a interconectar, pues es en este tipo de escenarios donde la técnica de los nodos MPR es más eficiente. Cuanto más grande y densa sea una red, mayor optimización se alcanzará comparándola con el algoritmo clásico de estado de enlace.

El documento RFC 3626 separa el protocolo OLSR en dos partes: el núcleo funcional (*core functionality*) y un conjunto de funciones auxiliares. El núcleo funcional son aquellas funciones indispensables para que OLSR funcione correctamente y que deben ser implementadas en todo dispositivo que quiera utilizarlo. Las funciones auxiliares son una extensión que otorga al protocolo una mayor redundancia y compatibilidad con otras redes que no sean MANET. Según el RFC 3626, el protocolo permite el uso de nodos heterogéneos, es decir, nodos con diferentes implementaciones y por lo tanto que contengan un diferente conjunto de funciones auxiliares.

A continuación se explicarán cada una de las funciones que proporciona OLSR para que un conjunto de nodos pueda conocer la topología de la red y mantener así actualizadas sus tablas de routing, describiendo después la estructura principal de un paquete OLSR y los principales tipos de mensajes que existen (`HELLO`, `TC` y `MID`).

3.1. Funcionamiento del protocolo OLSR

Desde que un nodo entra en una red MANET hasta que la red entera se actualiza y crea todas las rutas posibles desde cualquier nodo de la red hasta el nuevo dispositivo entrante, se llevan a cabo ciertas acciones del protocolo OLSR para actualizar cada una de las tablas de routing. Las principales funciones o acciones que toma el protocolo se describen en los siguientes apartados:

3.1.1. Descubrimiento de vecinos

Cuando un nodo entra en una red móvil ad-hoc, no sabe qué dispositivos forman esa red. El primer paso será conocer quiénes son sus vecinos (aquellos dispositivos que están en su área de cobertura y por lo tanto pueden llegar a él de un solo salto). El mecanismo para la detección de los vecinos es el intercambio periódico de mensajes `HELLO` (para un mayor detalle de este tipo de mensajes ver apartado 3.3.1).

Una posible secuencia de intercambio de mensajes `HELLO` entre dos vecinos que todavía no han establecido conexión puede ser la que se muestra en la

Figura 3.1.

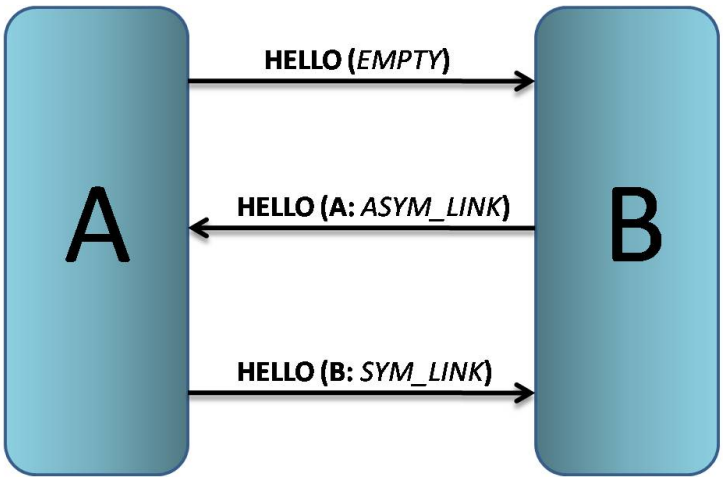

Figura 3.1: Descubrimiento de vecinos

En primer lugar, cuando un nodo entra en la red (nodo A) envía a todos los vecinos un mensaje `HELLO` con su campo de vecinos vacío. Cuando algún nodo recibe este mensaje (nodo B) actualiza su tabla de routing pues sabe que puede recibir mensajes de A. Entonces envía un mensaje `HELLO` actualizado, donde incluye al nodo A como su vecino pero con un enlace de tipo asimétrico (ya que lo único que sabe es que puede recibir mensajes de A, pero no al revés). En este momento el nodo A actualiza su tabla de routing considerando a B como vecino. Por último cuando el nodo A envíe otra vez un mensaje `HELLO`, incluirá a B como vecino y esta vez será de tipo simétrico (nodos A y B pueden enviar y recibir mensajes entre ellos).

Estos mensajes `HELLO` son enviados periódicamente, dando a conocer a los demás nodos quiénes son tus vecinos (sólo alcanza un ámbito local ya que no pueden ser reenviados a otros nodos). Si un dispositivo tiene varias interfaces, hay que considerar a cada interfaz perteneciente a una red de vecinos independiente, ya que puede que algún nodo sólo pertenezca al área de cobertura de una de las interfaces, como se puede observar en la Figura 3.2).

En este caso, los vecinos del nodo A son, por un lado C y D (a través de la interfaz a1), y por otro lado el nodo B (a través de la interfaz a2), siendo los mensajes `HELLO` diferentes en cada interfaz.

Al recibir un mensaje `HELLO`, un nodo también puede descubrir el conjunto de nodos disponibles a través de sus vecinos (pero que no se encuentran en su área de cobertura). Este conjunto de nodos se denomina vecinos de segundo salto (*2-hop neighbor set*), en el que pertenecen aquellos nodos, que no siendo

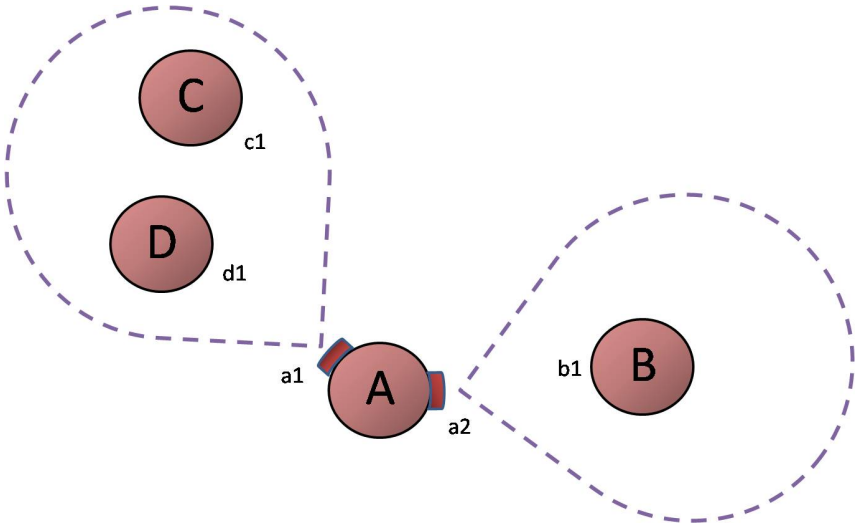

Figura 3.2: Descubrimiento de vecinos en múltiples interfaces

vecinos tuyos, están incluidos en un mensaje `HELLO` recibido de un vecino. Este tipo de nodos será fundamental en el cálculo por parte de cada dispositivo de la red de su conjunto de nodos MPR.

3.1.2. MultiPoint Relaying (MPR)

Una vez que un nodo descubre quiénes son sus vecinos (tanto los que se encuentran en su área de cobertura como los de segundo salto), el siguiente paso es calcular quiénes van a ser sus nodos MPR. Tal y como se ha dicho previamente, ésta es la principal diferencia con respecto al protocolo clásico de estado de enlace y su principal virtud para la adaptación del protocolo a redes móviles ad-hoc. Los nodos MPR serán aquellos vecinos que, elegidos correctamente, pueden reenviar cualquier mensaje a todos los vecinos de segundo salto (es indispensable que los enlaces entre estos vecinos sean de tipo simétrico). Cuanto menor sea este número de nodos MPR con respecto al número de vecinos, mayor eficiencia obtendrá el protocolo pues introducirá una menor sobrecarga de tráfico en la red. Debido a que el conjunto de vecinos es diferente para cada nodo, el conjunto de nodos MPR también será diferente para cada uno de los nodos que forman la red.

En el algoritmo clásico, si un nodo quiere enviar un mensaje de broadcast a toda la red, inunda la red enviándoselo a todos sus vecinos, éstos a su vez lo reenvían a los suyos y así sucesivamente hasta que toda la red ha recibido el mensaje. En cambio en OLSR, si un nodo quiere enviar un mensaje de

broadcast a toda la red, igualmente envía su mensaje a todos los vecinos, pero en esta ocasión sólo reenviarán el mensaje aquellos nodos seleccionados como MPR (pues sólo con este conjunto es posible alcanzar a todos los vecinos de segundo salto).

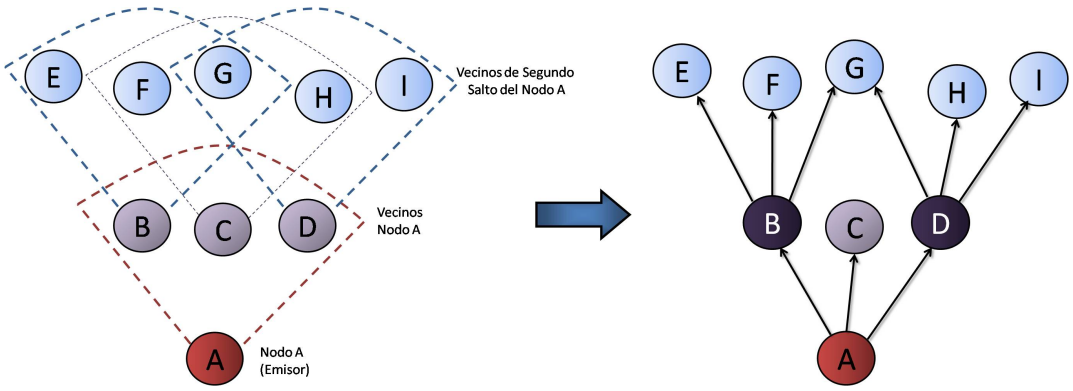

Figura 3.3: Selección de los nodos MPR

En el caso de la Figura 3.3, todos los nodos de segundo salto de A pueden ser alcanzados a través de B y D, y por tanto son elegidos nodos MPR por A. Por lo tanto cuando el nodo A envíe un mensaje de broadcast, solamente los nodos B y D reenviarán dicho mensaje a sus vecinos. De esta manera se ahorra el ancho de banda que utilizaría el nodo C para reenviar los mensajes a los nodos F, G y H (inundación de la red en el algoritmo clásico).

En el caso de redes cableadas, para que la inundación del mensaje en la red sea más eficiente no se suele retransmitir por la misma interfaz que ha recibido el mensaje. Pero en el caso de redes móviles esto no es posible ya que la transmisión se propaga en espacio libre y lo usual es que la misma interfaz que recibe el mensaje sea la misma que lo retransmite. Por esta razón, además de la limitación en ancho de banda y energía disponible de los dispositivos móviles, hace imprescindible la reducción en lo posible del número total de retransmisiones, consiguiéndose gracias a la utilización de los nodos MPR. Otra manera de reducir estas retransmisiones en OLSR es mantener una caché de mensajes duplicados donde se averigua si un mensaje ya ha sido procesado o retransmitido (a través del número de secuencia del mensaje), siendo en ese caso descartado por el nodo.

La eficiencia del protocolo OLSR en el ahorro del tráfico cuando se inunda la red con un mensaje de broadcast se puede observar en las Figuras 3.4 y 3.5, donde solamente los nodos MPR (en oscuro) reenvían la información, ahorrándose así retransmisiones innecesarias pues al final todos los nodos de la red reciben el mensaje que envió el emisor (nodo central).

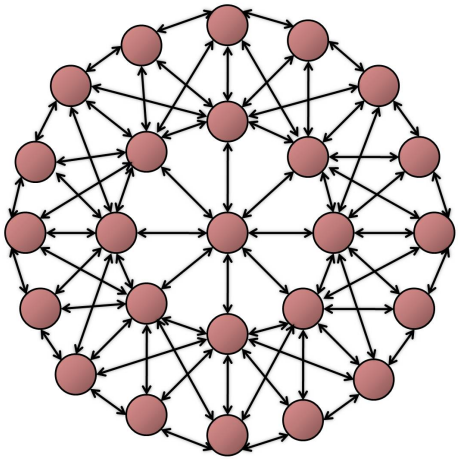

Figura 3.4: Inundación de una red sin nodos MPR

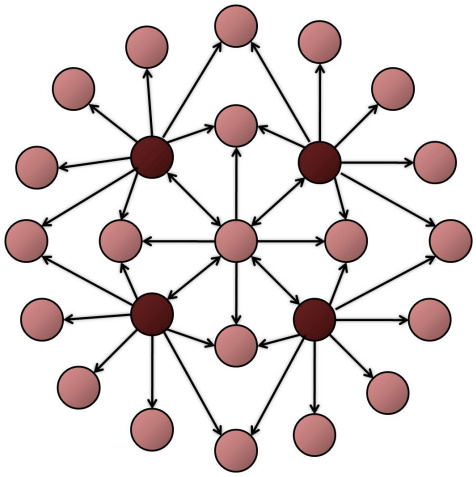

Figura 3.5: Inundación de una red con nodos MPR

Debido a que en una red móvil la duración de la batería de los dispositivos puede ser mayor o menor según el tipo de dispositivo, el protocolo OLSR permite a los nodos que anuncien en los mensajes `HELLO` su voluntad (*willingness*) de retransmitir mensajes a otros nodos y por lo tanto de convertirse en nodos MPR. Existen 8 niveles de voluntad, siendo el más bajo el número 0 (no retransmite paquetes - etiqueta `WILL_NEVER`) y el más alto el número 7 (siempre dispuesto a reenviar paquetes - etiqueta `WILL_ALWAYS`). Si un nodo tiene nivel 7 de voluntad siempre debería ser elegido como nodo MPR.

El conjunto óptimo de nodos MPR (menor número de nodos que alcanza a todos los vecinos de segundo salto) es muy difícil de calcular, habiéndose demostrado que se trata de un problema de decisión de tipo NP-completo [3]. Para resolver este problema, el protocolo OLSR utiliza un método heurístico sencillo para decidir quiénes son los nodos MPR de cada dispositivo. Este método tiene en cuenta tanto el número de vecinos de segundo salto que es capaz de alcanzar cada nodo como su nivel de voluntad (*willingness*) para retransmitir un mensaje.

La explicación de este método heurístico escapa al alcance de este proyecto, siendo posible su análisis y estudio exhaustivo en [4]. Desde que apareció el protocolo OLSR otros métodos para el cálculo del conjunto de nodos MPR han sido propuestos. Por ejemplo en [5] sugieren que la selección de los nodos MPR debería también estar basada en la movilidad de los nodos dentro de la red, dando preferencia a aquellos nodos más estables (con menor movilidad). Aunque el uso de los nodos MPR ha sido definido principalmente para el protocolo de routing OLSR, investigaciones recientes han propuesto otros usos a

la selección de nodos MPR. Por ejemplo en [31] se propone la estimación de las posiciones geográficas de los nodos que componen la red MANET basándose en la conectividad de los nodos MPR.

3.1.3. Cálculo de la topológía de la red

Una vez realizado el descubrimiento de vecinos y el cálculo de los nodos MPR, lo único que falta por hacer antes de actualizar la tabla de routing es diseminar por toda la red mensajes con la información de los enlaces de cada nodo para que cada dispositivo de la red pueda obtener la topología de la red. Dicha información se envía a través de mensajes `TC` (para un mayor detalle de este tipo de mensajes ver apartado 3.3.2).

De toda la información obtenida a través de los mensajes `HELLO`, lo que un nodo inunda a toda la red es la información relativa de aquellos nodos que le han elegido como nodo MPR. De esta manera se consigue, además de una sensible mejora en la eficiencia al reenviar los mensajes de broadcast sólo los nodos MPR, dos optimizaciones importantes:

- El mensaje `TC` se reduce con respecto al algoritmo clásico de estado de enlace, ya que en lugar de almacenar en el mensaje la información sobre todos tus vecinos, sólo envías información de aquellos vecinos que te han elegido como nodo MPR, reduciéndose así el tamaño del mensaje `TC`.
- Como sólo se envía información de los vecinos que te han elegido como nodo MPR, sólo deberán generar este tipo de mensaje aquellos nodos que hayan sido elegidos como nodos MPR por algún vecino, absteniéndose el resto. Esto supone un ahorro en tráfico y ancho de banda.

Al ser la topología de la red diseminada solamente por aquellos nodos elegidos como MPR, al calcular la tabla de routing éstos se convertirán en los routers que encaminarán el tráfico en la red, reenviando los mensajes de una parte a otra de la red.

Una vez recibidos los mensajes `TC` por parte de los nodos, se puede llevar a cabo el cálculo de la tabla de routing para obtener la topología de la red y saber a qué vecino debo enviar cualquier paquete para que éste sea encaminado a través de la red MANET y entregado correctamente a su destinatario.

Una tabla de routing está formada principalmente por la dirección IP destino, la dirección IP del siguiente salto y el valor del salto (número de nodos que tiene que cruzar un paquete para llegar a su destino). Los pasos a seguir en el cálculo de la tabla de routing de cualquier nodo son los siguientes:

1. Primero se añaden entradas de routing para los vecinos a partir de los mensajes `HELLO` recibidos con un valor de salto igual a 1. El siguiente salto es el propio vecino.

2. Por cada vecino y a partir de sus mensajes `HELLO`, se añaden los vecinos de segundo salto con valor de salto igual a 2, siendo el siguiente salto el vecino correspondiente.

3. Ahora, por cada mensaje `TC` recibido cuyo origen sea un nodo con valor de salto n = 2 (vecino de segundo salto), se añaden todos los nodos que aparecen en dicho mensaje (y que no estén en la tabla de routing) con un valor de salto n + 1 = 3. El siguiente salto será en este caso el nodo origen con valor de salto n = 2.

4. Se incrementa el valor de salto n en 1 y se repite el paso 3 hasta que no queden más entradas en la tabla de routing con valor de salto n + 1. Este paso se repite hasta que no queden más mensajes `TC` por procesar.

5. Por último se tiene que observar si algún nodo posee varias interfaces a partir de la información obtenida con los mensajes `MID` para actualizar la tabla de routing (para un mayor detalle de este tipo de mensajes ver apartado 3.3.3). Para ello se añade una entrada en dicha tabla pero poniendo, en lugar de la dirección IP principal del nodo, la dirección IP de su interfaz correspondiente (el valor de salto y el salto siguiente siguen siendo los mismos).

El procedimiento heurístico anterior es el típico algoritmo de encontrar el camino más corto (*shortest-path*), recogido en el RFC 3626 del protocolo OLSR [1]. La tabla de routing debe ser actualizada cuando se detecte un cambio en alguno de los vecinos de primer y segundo salto (mensajes `HELLO`), en alguno de los nodos de la red (mensajes `TC`) o en alguna interfaz adicional de un nodo (mensajes `MID`).

3.2. Estructura del paquete OLSR

Todos los mensajes pertenecientes al protocolo OLSR poseen la misma estructura (ver Figura 3.6), cuyos campos de cabecera más importantes son:

- **Packet Length:** Longitud del paquete en bytes incluyendo la cabecera

- **Packet Sequence Number:** Número de secuencia que se incrementa por cada nuevo paquete OLSR que se envía. Cada interfaz posee su propio número de secuencia.

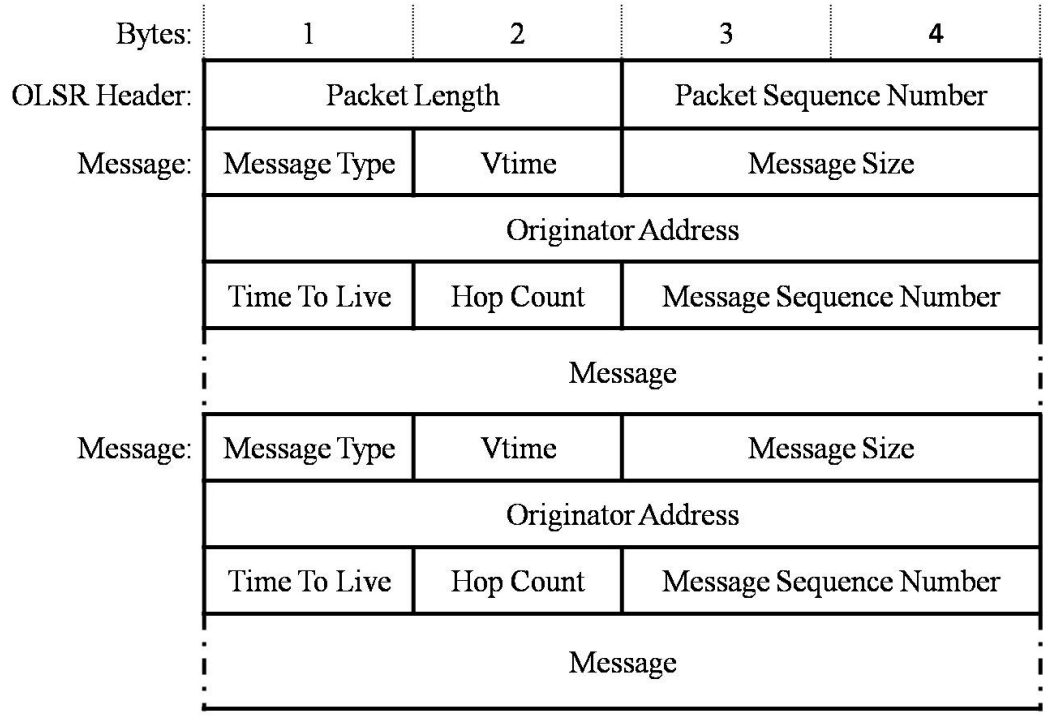

Figura 3.6: Paquete genérico OLSR, obtenido de [1]

Dentro de un mismo paquete puede haber uno o varios mensajes OLSR. Cada mensaje posee su propia cabecera cuyos campos más importantes son:

- **Message Type:** Tipo de mensaje OLSR
- **Vtime:** Intervalo de tiempo después de la recepción en el que se considera el mensaje recibido como válido.
- **Message size:** Tamaño del mensaje en bytes incluyendo la cabecera del mensaje.
- **Originator address:** Principal dirección IP del origen del mensaje.
- **Time To Live (TTL):** Máximo número de saltos que el mensaje puede ser reenviado hasta ser descartado y desechado por la red.

- **Hop Count:** El número de veces que ha sido retransmitido el mensaje (número de saltos entre el origen y el receptor del mensaje).
- **Message Sequence Number:** Número de secuencia de mensaje que se incrementa por cada nuevo mensaje OLSR que se envía. De esta manera se evita que un nodo pueda procesar un mensaje que ya ha recibido anteriormente.

Los paquetes están encapsulados en datagramas UDP para ser transmitidos a través de la red. El puerto asignado por IANA (*Internet Assigned Numbers Authority*) para el uso exclusivo de OLSR es el 698.

3.3. Tipos de mensajes OLSR

En el núcleo funcional del protocolo OLSR se definen tres tipos principales de mensajes, aunque en la sección de funciones auxiliares existe otro tipo de mensaje OLSR (mensajes `HNA`) para que un nodo se anuncie como puerta de enlace a otras redes externas (principalmente Internet).

Los tres principales mensajes OLSR se explican en los siguientes apartados:

3.3.1. Mensajes `HELLO`

Los mensajes `HELLO` se utilizan en las tareas de mantenimiento de enlaces, en la detección de nodos vecinos y en la señalización de nodos MPR. Su estructura se puede observar en la Figura 3.7.

Los campos del mensaje `HELLO` más importantes son:

- **Htime:** Intervalo de tiempo antes de la emisión del siguiente mensaje `HELLO` (debido a que este tipo de mensajes se envían periódicamente).
- **Willingness:** Indica la voluntad del nodo a encaminar y reenviar tráfico para otros nodos. Es un número del 0 al 7 según el nivel de voluntad, utilizándose este valor para el cálculo del conjunto de nodos MPR (ver apartado 3.2.2).
- **Link Code:** Especifica el estado del enlace entre el interfaz del emisor y la lista de interfaces vecinas (por ejemplo si el enlace es simétrico, asimétrico o se ha perdido la conexión). También da información de si el vecino ha sido seleccionado como nodo MPR por el emisor.

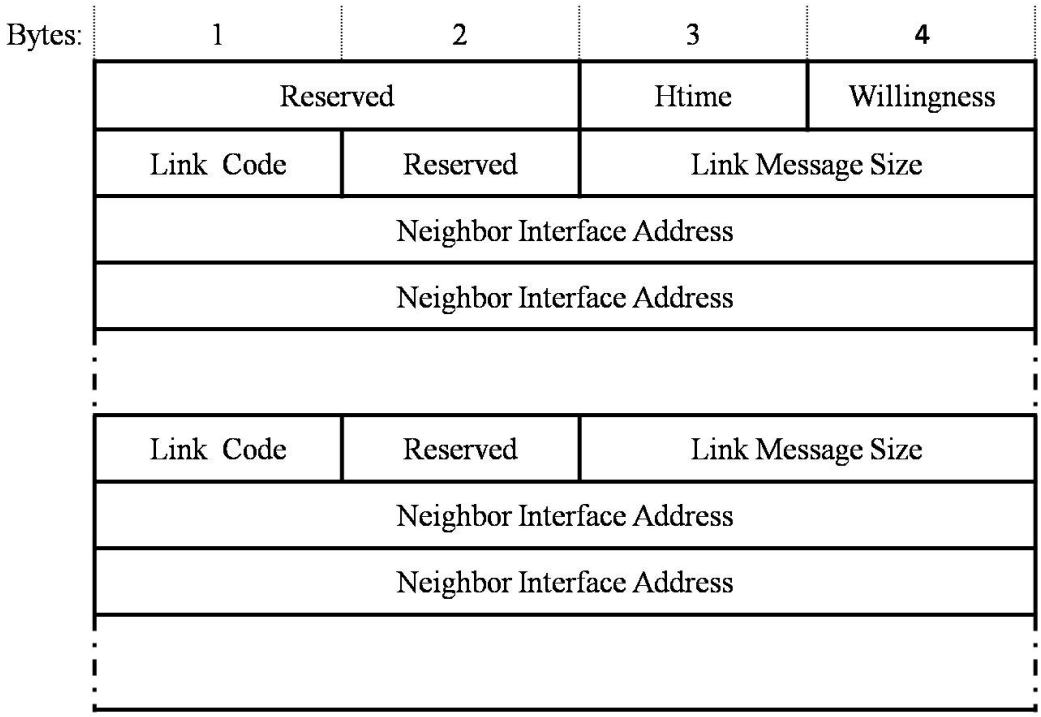

Figura 3.7: Estructura del mensaje `HELLO`, obtenido de [1]

- **Link Message Size:** Tamaño del mensaje contado en bytes, desde el principio del campo `Link Code` hasta el siguiente campo `Link Code` o hasta el final del mensaje.

- **Neighbor Interface Address:** Dirección IP de las interfaces vecinas del nodo emisor.

3.3.2. Mensajes `TC` (*Topology Control*)

Los mensajes TC se utilizan en la tarea de anuncio de nuevos enlaces para el cálculo de la topología de red por parte de cada nodo. Su estructura se observa en la Figura 3.8.

Los campos del mensaje `TC` más importantes son:

- **ANSN (*Advertised Neighbor Sequence Number*):** Número de secuencia de mensaje que se incrementa, no cuando un nodo genera un mensaje `TC`, sino cada vez que dicho nodo detecta un cambio en su conjunto

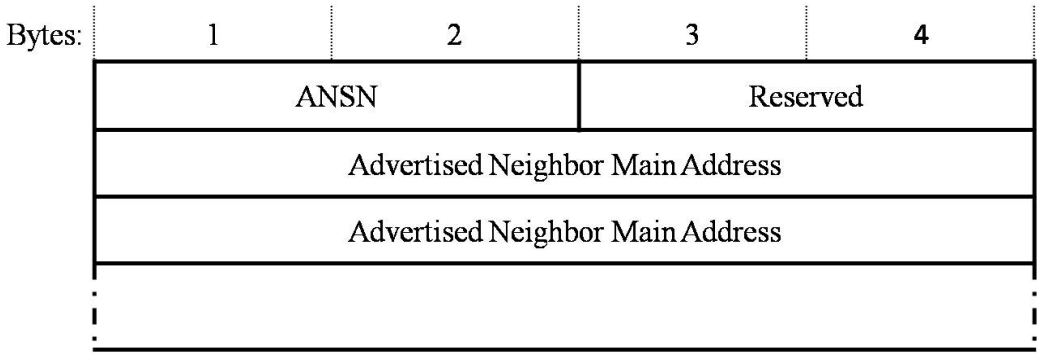

Figura 3.8: Estructura del mensaje `TC` (*Topology Control*), obtenido de [1]

de vecinos. Gracias a este número de secuencia el receptor del mensaje sabe si éste contiene la información más reciente de dicho nodo.

- **Advertised Neighbor Main Address:** Contiene la principal dirección IP del nodo vecino que le ha seleccionado como nodo MPR. También puede contener direcciones IP de otros vecinos para dar una mayor redundancia e información a los demás nodos de la red (pero incrementando también el tamaño del mensaje y la sobrecarga de tráfico a la red).

3.3.3. Mensajes `MID` (*Multiple Interface Declaration*)

Los mensajes MID se utilizan en la tarea de declarar la presencia de múltiples interfaces en un nodo. Su estructura se puede ver en la Figura 3.9.

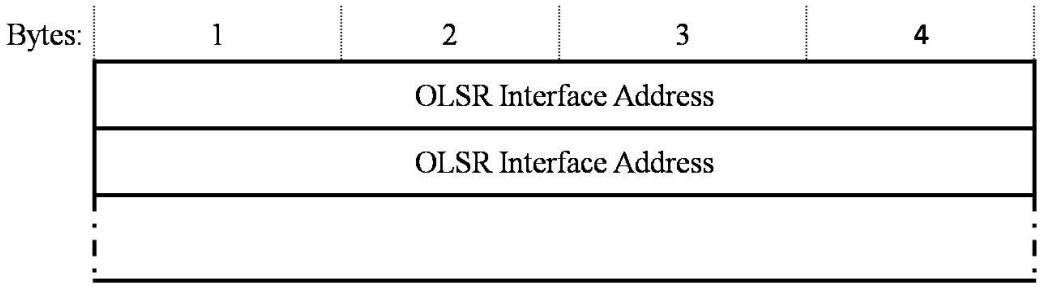

Figura 3.9: Estructura del mensaje MID (*Multiple Interface Declaration*), obtenido de [1]

El único campo del mensaje `MID` es:

- **OLSR Interface Address:** Contiene una de las múltiples direcciones que posee el nodo emisor, excluyendo de este conjunto la dirección IP principal del nodo (que ya está indicada en la dirección de origen del mensaje OLSR).

3.4. OLSR versión 2

Actualmente se está desarrollando la versión 2 del protocolo OLSR [56] para hacerla más flexible y eficiente, manteniendo en general las principales propiedades que caracterizan al protocolo, como la selección de nodos MPR o el cálculo de la topología de la red. Una de las principales diferencias entre la versión 1 y 2 de OLSR es el formato de los mensajes. En OLSRv1 cada uno de los mensajes tiene un número fijo de campos (obligatorios) con su valor correspondiente. En cambio OLSRv2 utiliza en sus mensajes un número variable de bloques denominados TLV [57] (Type-Length-Value). Cada bloque TLV da un valor a un atributo específico y está compuesto de 3 campos: el nombre del atributo, la longitud del bloque en octetos y su valor. En cada mensaje de OLSRv2 existirá una cantidad variable de bloques TLV, algunos obligatorios según el tipo de mensaje y otros opcionales dependiendo de la información de la que disponga el nodo. De esta manera el intercambio de información entre los nodos es más flexible y eficiente.

También se ha reducido el número de tipos de mensajes. Si en OLSRv1 existen 4 tipos de mensajes (`HELLO, TC, MID y HNA`), en OLSRv2 sólo se utilizan los mensajes `HELLO` y los mensajes `TC`, ya que las direcciones de las múltiples interfaces anunciadas por los mensajes `MID` se incluyen en los mensajes `TC` y los mensajes `HNA` se han convertido en bloques TLV opcionales de tipo `GATEWAY` (para anunciar acceso a otro tipo de redes que no sean MANET).

El descubrimiento de servicios en OLSRv2 se produce a través del protocolo NHDP [58] (*NeighborHood Discovery Protocol*), separando de esta manera el protocolo de descubrimiento de vecinos con el de routing, convirtiéndolo así en un protocolo más modular y flexible. El protocolo NHDP define el formato de los mensajes `HELLO` y la actualización de la información que necesitan los nodos, tanto la información local como la de los vecinos. El protocolo NHDP ha sido diseñado de tal manera que pueda ser utilizado por otros protocolos de routing, no solamente por OLSRv2.

3.5. Resumen

El protocolo OLSR es un protocolo de tipo proactivo utilizado en redes móviles ad-hoc, consiguiendo una mayor eficiencia en redes grandes y densas (con muchos dispositivos) gracias a la selección de unos nodos denominados MPR (MultiPoint Relaying), que son los que reenvían los paquetes en la red y encaminan el tráfico a sus respectivos destinatarios, actuando de esta manera como routers.

Como se ha visto en el apartado 3.2, la funcionalidad del protocolo OLSR puede dividirse en tres módulos principales: el descubrimiento de vecinos (tanto de primer como de segundo salto), la selección de los nodos MPR y el anuncio de los nodos por la red para el cálculo de la topología de red y la consiguiente actualización de las tablas de routing de los nodos.

También se ha visto los principales tipos de mensajes y sus campos más importantes. Los mensajes `HELLO` se utilizan para conocer el estado del enlace con los vecinos, calcular el conjunto de vecinos de segundo salto y el cálculo de nodos MPR (para que reenvíen el tráfico generado cuando se quiere inundar la red con algún mensaje de broadcast). Los mensajes `TC` se utilizan para anunciar a los demás nodos de la red quiénes son aquellos vecinos que te han seleccionado como nodo MPR, sabiendo de esta manera que pueden alcanzar a dichos nodos a través de ti. Los mensajes `MID` se utilizan para conocer si algún nodo posee múltiples interfaces por las que llegar a él. Con toda la información obtenida de estos tres mensajes se puede calcular y/o actualizar la tabla de routing de cada nodo, que servirá para que todos los nodos sepan a qué vecino y por qué interfaz deben enviar cualquier paquete para que llegue correctamente a su destino.

Por último se ha descrito brevemente la nueva versión de OLSR y sus principales diferencias con la versión anterior, principalmente en cuanto al formato de los paquetes, la introducción de los bloques TLV para aumentar la eficiencia y flexibilidad del protocolo y la separación de los protocolos de descubrimiento de vecinos (NHDP) y descubrimiento de rutas propiamente dicho para otorgar mayor modularidad al protocolo de routing OLSR.

Capítulo 4

Plataforma de desarrollo Maemo

Maemo [2] es una plataforma de desarrollo de software abierto creada por Nokia en colaboración con otros proyectos tales como Linux, Debian [32], GNOME [33], etc. Su principal característica es la base sobre la que está implementada toda la pila de software de la plataforma Maemo: el kernel del sistema operativo Linux. Este sistema es uno de los más versátiles que existen ya que es capaz de ser implementado en gran cantidad de dispositivos totalmente diferentes, desde pequeños relojes hasta grandes servidores. Además, al ser de código abierto, cualquiera con conocimientos en sistemas operativos puede adaptarlo según sus necesidades (un ejemplo de esto es la gran cantidad de distribuciones de sistemas operativos que están basados en el kernel de Linux, tales como Ubuntu [66], Debian [32], Mandriva, Fedora, etc.).

Las características más importantes de la plataforma Maemo son:

- Primera plataforma de software en llevar la arquitectura basada en Linux a los dispositvos móviles.
- Diseñado para potenciar las capacidades multimedia y de Internet en los dispositivos móviles.
- Desarrollado por varias de las mejores comunidades de desarrollo de software de código abierto.
- Optimizado para las aplicaciones Internet 2.0 en dispositivos móviles basados en Linux.

La filosofía de Maemo es la de llevar las aplicaciones típicas utilizadas en equipos de sobremesa a dispositivos móviles tales como PDAs o teléfonos móviles (que utilizan otro tipo de arquitectura hardware denominado ARM [34]).

Su objetivo es por tanto que el sistema sea facilmente extensible y portable, pudiendo exportar sin ninguna dificultad aplicaciones ya existentes de GNU/Linux.

Para ello la plataforma Maemo utiliza en su arquitectura de interfaz de usuario uno de los frameworks más utilizados en la actualidad denominado GNOME [33], especialmente el conjunto de widgets GTK+. Esto permite la portabilidad de aplicaciones entre las diferentes plataformas, siendo además de código abierto la mayoría de las librerías utilizadas. La única parte del código de la plataforma Maemo que no es abierto consiste en algunos componentes de la interfaz de usuario y los drivers de los dispositivos móviles, pertenecientes o bien a Nokia o bien a terceras empresas.

Actualmente, el software desarrollado por la plataforma Maemo se utiliza en los dispositivos móviles de Nokia denominados *Nokia Internet Tablets*, entre los que se encuentran los modelos Nokia N770, N800 (ver Figura 4.1), N810 y el nuevo N900 (con la renovada versión 5 de Maemo).

Figura 4.1: Nokia N800 Internet Tablet

4.1. Componentes de la plataforma Maemo

Maemo está basado en gran parte en las componentes de código abierto pertenecientes a la distribución de Linux Debian [32], construyendo en GNU/Linux el sistema operativo y en GNOME/GTK+ la arquitectura de interfaz de usuario. Es por ello que la pila de software de la arquitectura principal de la plataforma Maemo posee en gran medida los mismos módulos que la distribución Debian de Linux, pudiendo ver sus componentes principales en la Figura 4.2.

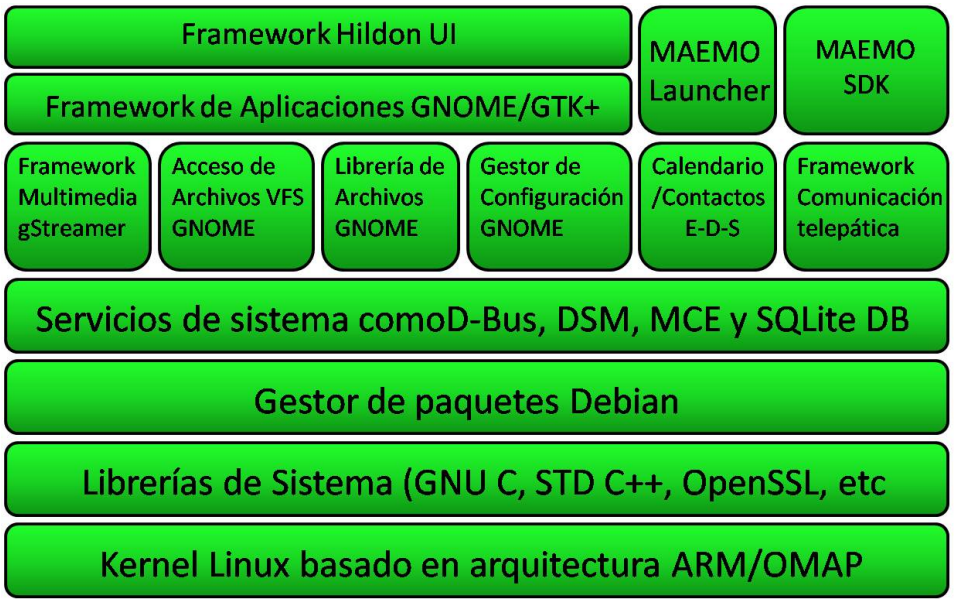

Figura 4.2: Arquitectura de software de la plataforma Maemo, basado en [2]

En los siguientes apartados se explica brevemente cada uno de los módulos principales representados en la pila de software de la Figura 4.2.

4.1.1. Kernel

Maemo utiliza la versión 2.6 del kernel del sistema operativo Linux, cuyo núcleo posee una arquitectura monolítica, es decir, concentra todas las funcionalidades posibles (planificación, sistema de archivos, redes, controladores, gestión de memoria, etc.) dentro de un programa grande y complejo. El kernel de Maemo está basado en la arquitectura ARM, que implementa los drivers de hardware (USB, LCD, WLAN, etc) encima de las funcionalidades que ofrece el kernel de Linux.

4.1.2. Librerías de sistema

La plataforma Maemo utiliza las librerías estándar GNU C, al igual que la mayoría de las distribuciones Linux. Para la abstracción del hardware, Maemo provee una capa de abstracción denominada HAL (*Hardware Abstraction Layer*), siendo capaz de cargar el driver correspondiente de cualquier dispositivo que se conecte, manteniendo su estado actual y ofreciendo a las aplicaciones que lo soliciten las funcionalidades de dicho dispositivo.

4.1.3. Servicios del sistema

El canal de comunicaciones primario entre las diferentes aplicaciones es DBUS, que también es el encargado de las comunicaciones entre el sistema y las aplicaciones. Las comunicaciones se realizan por paso de mensajes entre las aplicaciones. El sistema también ofrece otros servicios como una base de datos para almacenar información del usuario (SQLite 3), conocer el estado del dispositivo (DSM) o de la batería (BME).

4.1.4. Framework de aplicaciones

El propósito de un framework de aplicaciones es la de proveer una estructura de aplicación estándar para ayudar al desarrollo de las aplicaciones de software, especialmente aquellas que tengan una interfaz gráfica de usuario. El framework de aplicaciones de Maemo se denomina *Hildon*, y está parcialmente basado en la misma tecnología que utiliza el framework GNOME, sobre todo en los componentes de tipo GTK+. *Hildon* está especialmente diseñado para adaptarse a las limitaciones de los dispositivos móviles, realizando optimizaciones tales como:

- Reducción de la resolución de pantalla.
- Reducir requerimientos de memoria.
- Mejora de la velocidad para dispositivos móviles.
- Guardado automático del estado del dispositivo (ante posible cierre del sistema por falta de batería).
- Soporte para aplicaciones con pantalla táctil.

4.2. Conectividad de dispositivos Maemo

Después de una breve introducción a la plataforma de desarrollo Maemo y sus principales componentes, se presentan las tecnologías que pueden utilizar los dispositivos Maemo para interconectarse entre sí, que es al fin y al cabo la parte que más nos puede interesar debido a la naturaleza del proyecto.

Principalmente, varios dispositivos Maemo pueden interconectarse entre sí a través de:

- Wireless LAN (principal tecnología para la conexión a Internet).
- Bluetooth (para la conexión de corto alcance entre dispositivos).

El sistema de conectividad de Maemo está implementado con las librerías estándar de C que provee el kernel de Linux, utilizando el demonio IC (*Internet Connection*) que se encarga tanto de las conexiones WLAN como de las conexiones Bluetooth (para este caso se utiliza la implementación de Bluetooth para Linux denominado BlueZ).

El gestor de estas conexiones (*Connection Manager*) provee al dispositivo Maemo de una interfaz para manejar las conexiones con Internet y con otros teléfonos, mostrando también el estado de las conexiones activas en ese momento (con sus correspondientes estadísticas de paquetes transmitidos, recibidos, etc.).

Los dos subsistemas a los que pueden acceder los dispositivos Maemo con la tecnología inalámbrica descrita anteriormente se explican en los siguientes apartados:

4.2.1. Acceso a teléfonos móviles

El subsistema encargado de las conexiones con otros teléfonos móviles se denomina *Phone Access*, el cual provee una herramienta para encontrar otros teléfonos móviles y descubrir qué servicios pueden ofrecer a través de Bluetooth. Entre dispositivos Maemo la compatibilidad es total, aunque cualquier teléfono móvil que soporte el protocolo de descubrimiento de servicios de Bluetooth [35] (SDP), el perfil de red Dial-Up (DUN) y el perfil de transferencia de archivos (FTP) puede conectarse a un dispositivo Maemo. La conexión con un dispositivo se produce bajo demanda, es decir, cuando una aplicación requiere una conexión con ese dispositivo en concreto se envía un mensaje solicitando dicha conexión.

Los componentes más importantes de este subsistema son:

- **Interfaz de selección de teléfonos:** Interfaz de usuario que gestiona las operaciones realizadas con los teléfonos móviles a los que se conecta el dispositivo Maemo.
- **Interfaz general de Bluetooth:** Interfaz de usuario que maneja las conexiones con todos los dispositivos Bluetooth (no sólo teléfonos móviles).
- **Búsqueda Bluetooth:** Realiza la búsqueda de todos los dispositivos Bluetooth que estén disponibles en el área de cobertura.

- **Descubrimiento de servicios Bluetooth:** Comprueba qué servicios ofrece cada dispositivo Bluetooth, usando para ello el protocolo SDP.

4.2.2. Acceso a Internet

El subsistema encargado de las conexiones del dispositivo Maemo a Internet se denomina *Internet Access*, el cual provee aplicaciones con conexiones TCP/IP. Estas conexiones pueden ser realizadas tanto con tecnología WLAN (a través de un punto de acceso wireless) o con tecnología Bluetooth (usando el protocolo Point-to-Point (PPP) para establecer un enlace desde un modem en el teléfono).

Los componentes más importantes de este subsistema son:

- **Demonio IC (*Internet Connection*):** Establece la conexión a Internet ya sea a través de WLAN o Bluetooth. Entre los principales objetivos de este demonio está el control de que sólo exista una conexión activa a Internet en un momento dado, autenticar la conexión, obtener la dirección IP a través de DHCP y proveer las estadísticas sobre el uso de dicha conexión.

- **Interfaz gráfico de la conectividad a Internet:** Este paquete contiene las aplicaciones gráficas para que el usuario pueda configurar los parámetros de la conexión al punto de acceso de Internet, También escanea las redes WLAN disponibles (a petición del usuario).

- **Demonio WLAN Connection:** Este demonio gestiona las conexiones del dispositivo Maemo a la red WLAN. Una de las características más importantes de este demonio es que cierra la conexión a Internet cuando no hay aplicaciones que lo estén usando para así ahorrar batería. Por lo tanto, las conexiones se crearán solamente cuando sea requerido por una aplicación.

- **EAP (*Extensible Authentication Protocol*):** Se encarga de la seguridad en redes WLAN, haciéndose cargo de la autenticación en los puntos de acceso a Internet. Tiene una interfaz de usuario para que éste introduzca la contraseña y de esta manera autenticarse en la red WLAN. Una vez se ha procedido a la autenticación, se le entrega un mensaje al demonio IC para que establezca la conexión a Internet a través de ese punto de acceso.

Por último, mencionar que la librería utilizada para la conectividad con Internet se denomina *LibConIC*. Esta librería se encarga de dar las funciones necesarias a las aplicaciones de software para la gestión de las conexiones de Internet en los dispositivos Maemo.

Capítulo 5

Descubrimiento de servicios

En los últimos años la popularidad de las tecnologías de la información ha crecido de forma exponencial en gran parte debido a que su diseño ha permitido a las personas con pocos conocimientos técnicos a saber utilizarlas sin grandes complicaciones. Para ello se ha conseguido que la aparente complejidad que conlleva la creación y el uso de una red como Internet se pueda esconder en una interfaz amigable y sencilla para el usuario. Uno de las mecanismos cruciales en este intento por simplificar los aspectos más tecnológicos en el uso de la red es el descubrimiento automático de servicios.

Cada dispositivo que compone una red posee diferentes tipos de equipamiento y asume uno o varios papeles dentro de esa red (conexión a Internet, servicio de impresora, manos libres, sensores, cañón de proyección, bases de datos, etc.). Todas estas aplicaciones pueden ser descritas como servicios que pueden ser compartidos y accedidos por cualquier dispositivo que participa en la red de forma automática y sin importar dónde se encuentra o a quién pertenezca. Por lo tanto cuando un dispositivo entra por primera vez en una red puede saber a través de intercambio de mensajes qué servicios le ofrece dicha red sin necesidad de conocer previamente dichos servicios ni las direcciones IP u otros atributos de los dispositivos que la ofrecen.

En resumen, el descubrimiento de servicios sirve para que los dispositivos conozcan todas las funcionalidades o servicios que le ofrece la red, así como también anunciar sus propias capacidades a los demás nodos. De esta manera estos servicios podrán ser utilizados de manera fácil y sencilla sin que los dispositivos tengan la necesidad de conocer la arquitectura de la red que funciona por debajo.

5.1. Entidades participantes

El descubrimiento de servicios utiliza un protocolo específico que determina la estructura de los mensajes que se deben intercambiar los diferentes dispositivos: unos para dar a conocer los servicios que ofrece y otros para averiguar qué servicios están disponibles dentro de esa red. Los agentes participantes en este intercambio de mensajes son los siguientes:

- **Cliente:** Es la entidad que requiere uno o varios servicios específicos ofrecidos por la red para ser utilizados en alguna aplicación del dispositivo.
- **Servidor:** Es la entidad que posee una o varias características o servicios que pueden servir a los demás dispositivos de la red. Por lo tanto es el agente que ofrece dichos servicios a través de la red.
- **Directorio:** Es un servidor que almacena de manera parcial o completa la información sobre los servicios que ofrece cada uno de los nodos que componen la red.

5.2. Arquitecturas

Con respecto a la propagación de la información sobre los servicios que ofrece cada dispositivo en la red, los protocolos de descubrimiento de servicios se pueden dividir en tres arquitecturas:

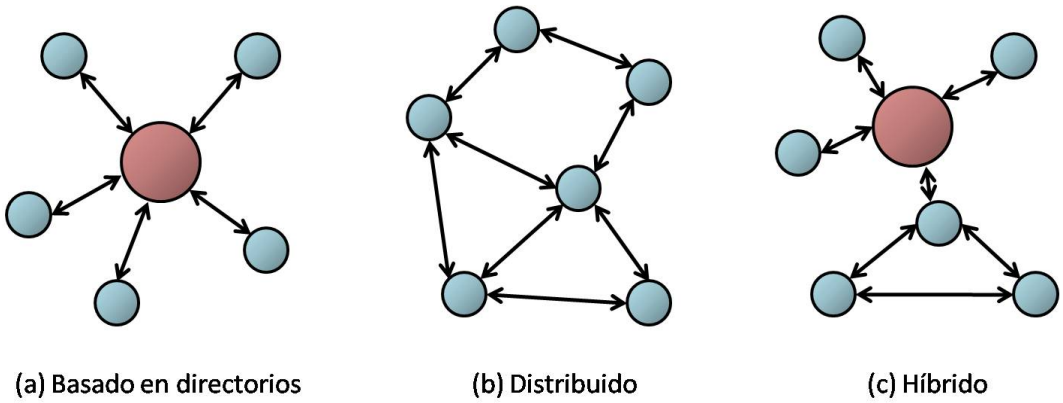

Figura 5.1: Tipos de arquitecturas en el descubrimiento de servicios

5.2.1. Basada en directorios

En este tipo de arquitectura uno o varios nodos de la red actúan como directorios de servicios registrando la información que les mandan los servidores. De esta manera los clientes, en lugar de solicitar la información a los propios servidores, van a buscarla a estos directorios (ver Figura 5.1(a)). Luego los directorios de servicios son el único punto de unión entre los clientes y los servidores.

5.2.2. Distribuida

En este modelo los clientes y los servidores se comunican directamente entre sí para conocer los servicios que ofrece la red sin la necesidad de directorios (ver Figura 5.1(b)). De esta manera los clientes no tienen que seleccionar a un directorio ni éstos necesitan sincronizarse cada poco tiempo para mantener la consistencia de su base de datos. Además al ser distribuida no existen puntos críticos de fallo en la red, por lo que el descubrimiento de servicios se hace más robusto frente a ataques. Por el contrario al no haber intermediarios las peticiones y los avisos deben ser enviados por toda la red aumentando así el tráfico y el ancho de banda.

5.2.3. Híbrida

Como en toda arquitectura híbrida, se intenta combinar las ventajas de las demás arquitecturas (basada en directorios y distribuida). Para ello la información de los servicios que ofrece la red se encuentra tanto en los directorios como en los servidores (ver Figura 5.1(c)). Si un cliente tiene a su alcance a un directorio, elegirá a éste para obtener la información que requiere como en la arquitectura basada en directorios. Si esto no es posible el cliente se comunicará directamente con el servidor como en la arquitectura distribuida. De esta manera se evita que existan puntos críticos de fallo en la red, ya que si un directorio falla el cliente siempre tiene la posibilidad de conectarse con el servidor directamente.

5.3. Modos de funcionamiento

Independientemente de la arquitectura, el protocolo de descubrimiento de servicios tiene dos modos de funcionamiento según la actitud de los clientes y

los servidores a la hora de solicitar y anunciar los servicios que ofrecen respectivamente.

5.3.1. Modo push (proactivo)

En este modo los servidores anuncian los servicios que tienen disponibles al resto de la red sin que ningún cliente se los haya solicitado previamente. El único cometido de los clientes es *escuchar* estos mensajes y seleccionar aquellos servicios que necesiten. Al igual que en los protocolos de routing, en este modo proactivo el ancho de banda consumido en la red es mayor pero tiene la ventaja de que no existe retraso inicial ya que los clientes poseen la información de los servicios que necesitan de antemano.

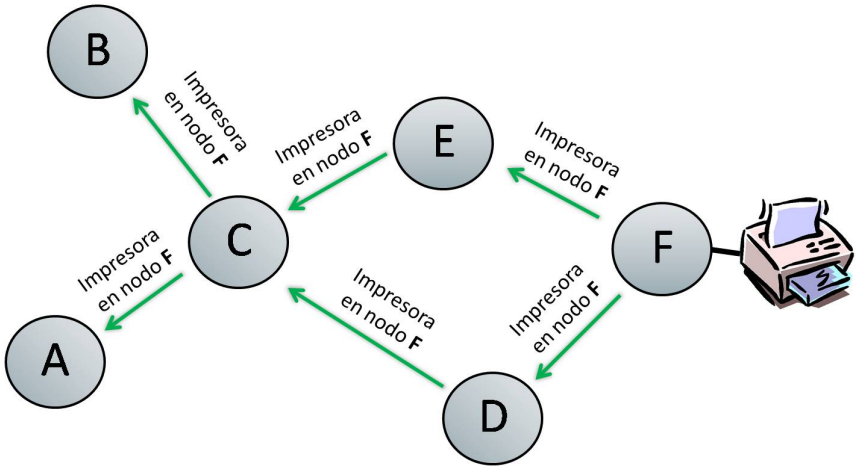

Figura 5.2: Descubrimiento de servicios en modo *push* (proactivo)

En la Figura 5.2 se observa un ejemplo de modo proactivo en el que el nodo F anuncia una impresora al resto de nodos. De esta manera, si algún nodo después necesita una impresora sabe, sin tener que preguntar a nadie, quien puede proporcionarle dicho servicio.

5.3.2. Modo pull (reactivo):

En este modo los clientes solicitan un servicio cuando lo necesitan inundando la red con mensajes de broadcast (o unicast si existen directorios). Los servidores en este caso sólo responden las solicitudes que le llegan de los clientes anunciando que ese servicio está disponible. Como en el tipo reactivo de los protocolos de routing, al ser un servicio por demanda el retraso inicial entre

que el cliente solicita el servicio hasta que el servidor responde es considerable, pero en cambio sólo se utiliza el ancho de banda estrictamente necesario.

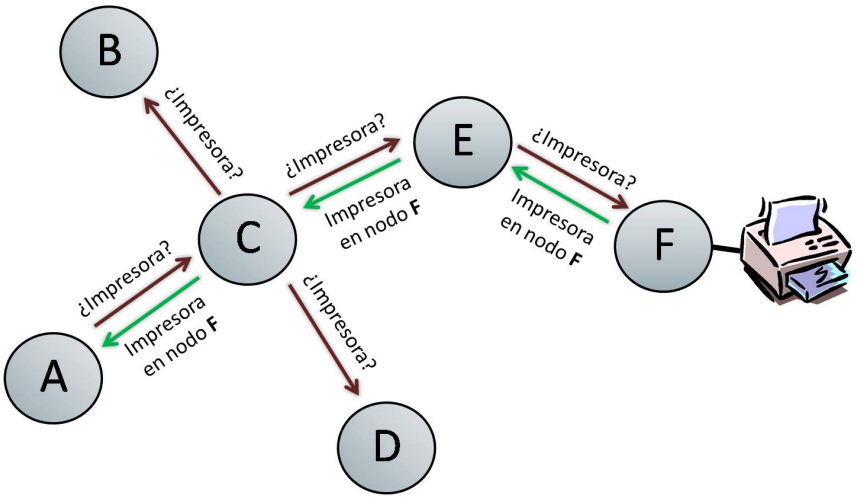

Figura 5.3: Descubrimiento de servicios en modo pull (reactivo)

En la Figura 5.3 se observa un ejemplo de modo reactivo en el que el nodo A necesita utilizar una impresora y envía por toda la red una petición de dicho servicio. El nodo que puede proporcionarle dicho servicio (en nuestro caso el nodo F) le envía una respuesta al nodo A para indicarle que él posee una impresora disponible.

5.4. Protocolos de descubrimiento de servicios

Después de ver en qué consiste principalmente el descubrimiento de servicios y los diferentes tipos de arquitectura y modos de funcionamiento, los principales protocolos de descubrimiento de servicios son presentados con una breve introducción de cada uno de ellos. Hay que tener en cuenta que la mayoría de los siguientes mecanismos son utilizados en redes fijas, con una infraestructura estable y por lo tanto no son adecuados para redes MANET. Sin embargo hay que destacar que los investigadores se han basado principalmente en estos ejemplos para diseñar los protocolos de descubrimiento de servicios de redes MANET.

Los protocolos de descubrimiento de servicios más importantes son:

- **Service Location Protocol [41] (SLP):** Desarrollado dentro del IETF (*Internet Engineering Task Force*), está basado en tres componentes:

`User Agents` (UA) – que son los clientes que solicitan los servicios, `Service Agents` (SA) – que son los servidores que anuncian los servicios, y `Directory Agents` (DA) – que son los directorios que contienen la información de los servicios que se ofrecen en toda la red, recogiendo tal información de los servidores y respondiendo a los clientes. La información de los servicios en los mensajes está en forma de Service URL.

- **Simple Service Discovery Protocol [42] (SSDP):** Desarrollado dentro del IETF (aunque nunca ha pasado del estado de *draft*), es utilizado por Microsoft como parte de su arquitectura UPnP (*Universal Pug and Play*) en sistemas Windows. Al igual que SLP, está basado en tres componentes: `SSDP service` – que representa a los servidores, `SSDP client` – que representa a los clientes, y `SSDP proxy` – que representa a los directorios. La diferencia entre SLP y SSDP radica básicamente en que no es obligatorio utilizar los SSDP proxy y en la forma de comunicación entre los diferentes agentes, que en el caso de SSDP se realiza a través de HTTP.

- **DNS Service Discovery [43] (DNS-SD):** Desarrollado por Apple como parte de la arquitectura Bonjour (antiguamente conocida como Rendevouz), este protocolo utiliza los mensajes empleados en DNS [15] para encontrar servicios, pudiendo ser usado tanto para obtener nombres, tipos u otros atributos de los servicios que se ofrecen en la red. Al utilizar DNS en lugar de HTTP, el protocolo DNS-SD es más simple que SSDP.

- **Jini [44]:** Desarrollado por Sun Microsystems, Jini es un tipo de tecnología de computación distribuida orientada a objetos basada en Java que además del descubrimiento de servicios permite la invocación y notificación de servicios. Jini posee una arquitectura basada en directorios denominada *Jini Lookup Service* (JLS). A través de JLS, los dispositivos pueden tanto ofrecer sus servicios como solicitarlos a través de objetos que contienen atributos y métodos en el lenguaje de programación Java. Como Jini depende de Java, requiere en cada dispositivo la utilización de la máquina virtual de Java (*Java Virtual Machine*).

- **Bluetooth Service Discovery Protocol [35] (SDP):** Especialmente diseñado para redes conectadas con tecnología Bluetooth, posee una arquitectura distribuida en el que el cliente SDP crea una petición PDU (*Protocol Data Unit*) respondida por aquel servidor que contenga el servicio requerido por el cliente. Esta respuesta puede contener información adicional u otros atributos del servicio.

En [55] se describen detalladamente más protocolos de descubrimiento de servicios.

5.5. Descubrimiento de servicios en MANET

En una red MANET el descubrimiento de servicios suele ser más complejo que en una red fija, debido sobre todo al dinamismo y a la movilidad de los nodos, donde cada dispositivo puede llegar a cambiar varias veces de red a lo largo de un período de tiempo relativamente corto. También hay que tener en cuenta la restricción de este tipo de redes en el ancho de banda y en el nivel de energía por el uso de baterías, así como la escalabilidad de la red (posibilidad de que hayan cientos o miles de nodos dentro de la misma red).

Otro de los problemas de las redes MANET es que al ser una red distribuida, la mayoría de aplicaciones donde se necesita un servidor centralizado (tales como DNS [15], SIP [16], DHCP [17], etc.) deben ser modificadas para poder proporcionar una infraestructura sin servidores, donde los nodos puedan comunicarse entre sí directamente sin la necesidad de intermediarios. Recientes investigaciones han propuesto resolver este problema integrando el proceso de búsqueda de otros nodos en el protocolo de routing, tanto en el caso de DNS [18], de SIP [19] como de DHCP [20].

Esto mismo ocurre con el descubrimiento de servicios basado en directorios. Debido a la movilidad existente en una red MANET, la selección del directorio más próximo al nodo cambiaría bastante a menudo con el consiguiente aumento del tráfico en la red. Además la sincronización de dichos directorios sería muchísimo más compleja pudiendo llegar a dar falsos positivos con mayor asiduidad (es decir, que el directorio anuncie un servicio cuando el servidor que lo ofrece ya no está disponible en la red MANET). Por lo tanto la mayoría de propuestas de protocolos de descubrimiento de servicios para redes MANET se decantan por una arquitectura distribuida.

En la actualidad existen dos maneras de diseñar un protocolo de descubrimiento de servicios en redes MANET: la que está implementada dentro del nivel de aplicación de la torre de protocolos OSI [45] (ver Figura 5.4(a)), por lo que se considera al descubrimiento de servicios como una aplicación cualquiera independiente de los niveles inferiores y del protocolo de routing, y la que está integrada dentro del protocolo de routing (ver Figura 5.4(b)) debido a la ligera similitud entre ambos protocolos, consiguiendo reducir así la sobrecarga de tráfico en la red [46].

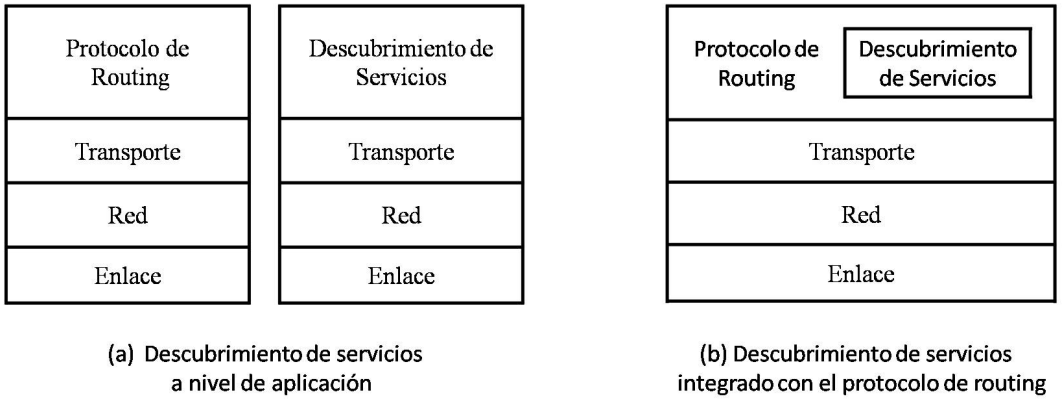

Figura 5.4: Descubrimiento de servicios en redes MANET

5.5.1. A nivel de aplicación

La manera más intuitiva de diseñar un protocolo de descubrimiento de servicios es hacerlo independiente al protocolo de routing ya que poseen objetivos diferentes dentro de una red MANET. Por lo tanto se implementa en el nivel de aplicación por encima de las capas más bajas de la red (enlace, red y transporte) para anunciar y solicitar servicios dentro de la red. Este tipo de implementación posee varias ventajas: que el mecanismo de descubrimiento de servicios es totalmente independiente del protocolo de routing, que la arquitectura puede extenderse a través de varias redes ya que no depende de las capas más bajas de la red y que al ser un diseño modular separando muy claramente cada capa, los protocolos de cada una de ellas pueden ser reemplazados sin afectar a las demás. Debido a la transparencia entre el descubrimiento de servicios a nivel de aplicación y las demás capas, la portabilidad y compatibilidad de este protocolo en diferentes redes MANET está asegurada.

La mayoría de las propuestas de descubrimiento de servicios en redes MANET siguen esta tendencia, entre las que destacan:

- **Pervasive Discovery Protocol [47] (PDP):** Es un protocolo totalmente distribuido en el que cada dispositivo posee un cliente (PDP User Agent), un servidor (PDP Service Agent) y una caché para almacenar tanto los servicios locales como los ofrecidos por el resto de dispositivos de la red. El protocolo PDP integra en su funcionamiento tanto el modo *push* como el modo *pull* de la siguiente manera: actúa en modo *pull* (reactivo) pues los servidores esperan a que un cliente solicite un servicio para responderlo, pero al recibir dicha solicitud los servidores difunden la respuesta a toda la red para que los dispositivos lo almacenen en su

caché local, actuando de esta manera en modo *push* (proactivo).

- **Konark [48]:** Su diseño está basado en un modelo peer-to-peer usando micro-servidores HTTP. Para clasificar los servicios utiliza una estructura en árbol, permitiendo búsquedas tanto para servicios generales (por ejemplo un servicio de impresión) como para servicios más específicos (por ejemplo un servicio de impresión a láser en color y a doble cara). Konark admite tanto los modos de funcionamiento *push* como *pull*, donde los clientes y los servidores solicitan o anuncian los servicios según sus necesidades. También utiliza una caché local en cada dispositivo para reducir el número de solicitudes de servicios.

- **SLPManet [49]:** Adaptación para redes MANET del protocolo Service Location Protocol [41] (SLP). El cambio más significativo entre ambos protocolos es la eliminación de los `Directory Agents` (DA) para convertir la arquitectura en distribuida. Como consecuencia de esto, los `Service Agents` (SA) sólo contestan a las solicitudes de los `User Agents` (UA). Como la mayoría de los protocolos en redes MANET, cada dispositivo contiene una caché local que incrementa la eficiencia del protocolo pero que puede dar falsos positivos si la topología de la red cambia. Este problema se encuentra en los protocolos de descubrimiento de servicios a nivel de aplicación, ya que al ser independiente del nivel de red no puede acceder a la información de routing y por tanto a la topología de la red.

5.5.2. Integrado con el protocolo de routing

En este caso lo que se propone es extender el protocolo de routing para que incluya una opción de descubrimiento de servicios debido a la similitud entre ambos tipos de protocolo. El objetivo final es conseguir optimizar y reducir la sobrecarga que se produce en el descubrimiento de servicios, que obviamente es mayor si se ejecuta a nivel de aplicación pues habría cabeceras de niveles inferiores tanto para el protocolo de routing como para el protocolo de descubrimiento de servicios, mientras que si está integrado ambos tipos de protocolos compartirían las cabeceras de niveles inferiores (ver Figura 5.4). Así se consigue una mayor eficiencia en la diseminación por toda la red de mensajes, tanto para el descubrimiento de rutas como para el de servicios, además del consiguiente ahorro en energía.

El precio a pagar por conseguir dicha eficiencia es la dependencia total con el protocolo de routing en el que se integra, siendo sólo compatibles con aquellas redes de parecidas características en las que se utilice el mismo protocolo de

routing. Además, esta solución viola la independencia entre las diferentes capas que conforman la red, teniendo que tener mucha precaución si alguna capa inferior es alterada ya que podría perjudicar al funcionamiento del descubrimiento de servicios.

Entre los mecanismos de descubrimiento de servicios integrados con el protocolo de routing hay que diferenciar entre los que se integran con un protocolo reactivo (por ejemplo AODV), con un protocolo proactivo (por ejemplo OLSR) y con un híbrido (por ejemplo ZRP):

Integrados con un protocolo de routing reactivo (AODV)

- **Service Discovery and Interaction with Routing protocols in Ad hoc Network [50] (SEDIRAN):** Este mecanismo de descubrimiento de servicios se integra en el protocolo de routing AODV que encapsula tres nuevos paquetes en el mensaje de respuesta RREP: la solicitud del servicio DREQ (*Discovery Request*), la respuesta a dicha solicitud DREP (*Discovery Reply*) y el anuncio de un servicio ADVM (*Advertisement Message*). Por lo tanto para difundir tanto las solicitudes como los anuncios de los servicios por toda la red el protocolo utiliza el mensaje de respuesta RREP (no diseñado para enviar mensajes de broadcast), teniendo que realizar por tanto algunos ajustes en el protocolo de routing. Esto conlleva que aquel nodo que no tenga instalado SEDIRAN no podrá reenviar los mensajes que le lleguen haciendo que el mecanismo de descubrimiento de servicios no funcione correctamente.

- **AODV Service Discovery [51] (AODV-SD):** Este protocolo modifica los formatos de los mensajes RREQ y RREP del protocolo de routing AODV para solicitar servicios y responder a dichas solicitudes respectivamente. Este protocolo permite el uso tanto de la arquitectura distribuida como de la arquitectura híbrida. En el caso de la arquitectura híbrida los anuncios del directorio son encapsulados en los mensajes de solicitud RREQ y los registros de los clientes en el directorio son encapsulados en los mensajes de respuesta RREP. Además cada nodo posee una caché interna que contiene una lista con la información de los servicios de los demás nodos de la red.

Integrados con un protocolo de routing proactivo (OLSR)

- **Service Discovery Mechanism [52] (SDM):** Este protocolo introduce dentro del paquete genérico OLSR un nuevo mensaje denominado

Service Discovery Message (SDM), el cual contiene únicamente anuncios de servicios (modo *push*), aprovechando así el carácter proactivo del protocolo OLSR. Cada nodo posee una caché interna que almacena todos los servicios disponibles, tanto los que ofrece el propio dispositivo como los del resto de nodos. El problema de este protocolo es que al ser el mensaje SDM tan pequeño (solamente 8 bytes), no puede ofrecer una descripción detallada de los servicios que se ofrecen en la red. En el siguiente capítulo se encuentra una descripción detallada de este protocolo al ser el elegido para su implementación en la plataforma de desarrollo Maemo.

- **Mercury [53]:** Al igual que el protocolo SDM, Mercury introduce un nuevo mensaje denominado *Mercury Service Discovery* (MSD) dentro del paquete genérico OLSR, pero en cambio tiene la posibilidad tanto de solicitar como de anunciar servicios (modos *pull* y *push*). Otra diferencia con el protocolo SDM es la forma de describir los servicios ya que utiliza para ello un filtro Bloom. Este tipo de filtros son capaces de en un pequeño espacio de memoria (128 bytes) contener un resumen de **todos** los servicios disponibles en el dispositivo (hasta un máximo de 256). Para ello crea una función hash para cada servicio, mapeando su resultado en la lista de servicios disponibles. El problema de utilizar este tipo de filtros es que puede llegar a dar un falso positivo si un nodo ofrece muchos servicios y se solapan sus resultados en el espacio de memoria asignado.

Integrados con un protocolo de routing híbrido (ZRP)

- **Extended ZRP [54] (E-ZRP):** En esta ocasión se extiende el protocolo híbrido ZRP para incluir descubrimiento de servicios. Dentro de cada zona donde actúa el protocolo de routing proactivo, cada nodo incluye en los mensajes `HELLO` todos los servicios que ofrece y los envía a todos sus vecinos. De la misma manera, en los mensajes donde los nodos difunden quiénes son sus vecinos incluyen también los servicios que ofrecen éstos para que el resto de los nodos dentro de la zona lo *escuchen* y lo almacenen en su caché interna. En esta propuesta no se menciona nada sobre el descubrimiento de servicios entre las diferentes zonas que componen la red MANET.

En este proyecto se implementará en un dispositivo Maemo el mecanismo de descubrimiento de servicios SDM (integrado con el protocolo de routing OLSR) para verificar la eficiencia que se consigue con este tipo de propuesta al reducir la cantidad de mensajes y por tanto la sobrecarga en la red MANET (además de acceso a la información de niveles de red como las tablas de routing)

a cambio de una menor compatibilidad e independencia con las capas inferiores de la red.

5.6. Resumen

Este capítulo ha tratado de explicar la importancia creciente del descubrimiento de serivicios en las redes de comunicaciones actuales, desarrollándose constantemente protocolos que intentan mejorar su eficiencia en determinadas situaciones y tipos de redes. Se han definido los agentes que participan en este tipo de protocolos (cliente, servidor y directorio) y algunas de las arquitecturas propuestas como la basada en directorios o la distribuida. También se ha realizado una explicación de los modos de funcionamiento: reactivo (modo *pull*) y proactivo (modo *push*), con sus ventajas e inconvenientes en aspectos como el tiempo de retardo en el descubrimiento del servicio o en el consumo del ancho de banda. Algunos de los protocolos de descubrimiento de servicios más importantes (como SLP [41] o Jini [44]) han sido descritos brevemente.

Dentro del estudio de los protocolos de descubrimiento de servicios se encuentran aquellos diseñados especialmente para su utilización en redes MANET, existiendo dos líneas de implementación: los diseñados a nivel de aplicación y los que están integrados en un protocolo de routing para aprovechar sus características y así conseguir una mayor optimización a cambio de una menor compatibilidad. Por último se ven los protocolos de descubrimiento de servicios diseñados específicamente para redes MANET más importantes, tanto a nivel de aplicación como integrados en un protocolo de routing.

Parte III

Desarrollo del proyecto

Capítulo 6

Service Discovery Mechanism

Service Discovery Mechanism [52] (SDM) es un protocolo de descubrimiento de servicios tipo *push* (proactivo) integrado dentro del protocolo de routing OLSR para redes MANET. Su principal característica es la sencillez de su estructura, estando especialmente diseñado para redes MANET donde el ancho de banda sea escaso, ya que los mensajes SDM que se transmiten a la red son pequeños (cada servicio solamente ocupa 8 bytes). A cambio de esta sencillez, el precio a pagar es la codificación de los servicios anunciables (teniendo que haber un consenso entre todas las implementaciones del protocolo) y unas descripciones de estos servicios no muy detallados, haciendo que los nodos no puedan realizar búsquedas muy específicas.

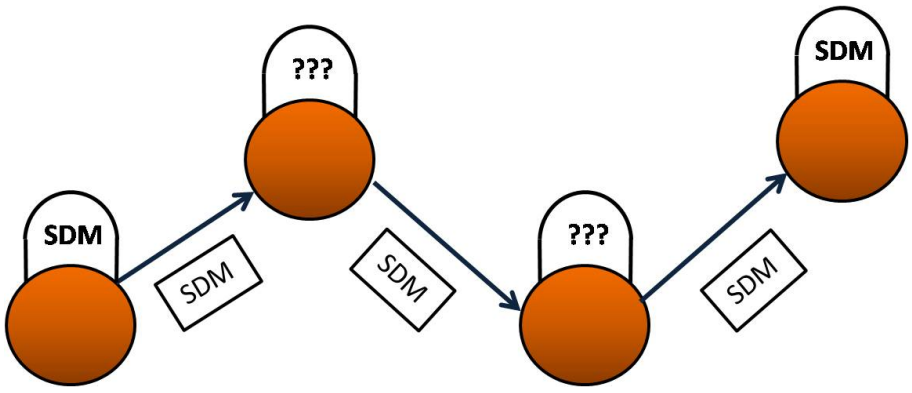

Figura 6.1: Reenvío correcto de mensajes SDM aunque los nodos centrales (elegidos como nodos MPR) no tienen implementado el protocolo SDM

Al integrarse dentro del protocolo OLSR, se aprovecha tanto de su naturaleza proactiva para anunciar los servicios al resto de nodos como del método

de reenvío de paquetes mediante los nodos MPR (sección 3.1.2), con el consiguiente ahorro en ancho de banda. Además la utilización del formato de paquete OLSR para encapsular los mensajes SDM hace posible el correcto funcionamiento de este protocolo aunque algunos nodos dentro de la red MANET no tengan instalada una implementación y por tanto no sepan interpretar dichos mensajes. Esto es debido a que el protocolo OLSR establece que los nodos MPR deben reenviar todo paquete OLSR aunque no comprendan el contenido del mensaje que hay encapsulado (en este caso el mensaje SDM, como se puede ver en la Figura 6.1).

Para almacenar los servicios disponibles en la red MANET este protocolo posee dos cachés: una destinada a los servicios locales y otra para los servicios anunciados por los demás nodos de la red. Tanto para anunciar como para buscar servicios, las aplicaciones que utilicen este protocolo harán uso de estas cachés, siendo clave su implementación debido a la limitada memoria que suelen poseer los dispositivos de una red MANET.

6.1. Estructura del mensaje SDM

La estructura del mensaje del protocolo SDM donde los nodos anuncian los servicios que poseen se puede observar en la Figura 6.2:

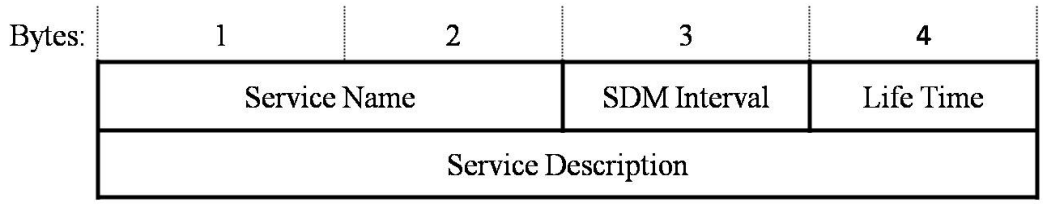

Figura 6.2: Estructura del mensaje SDM

Los campos del mensaje SDM más importantes son:

- **Service Name:** Nombre del servicio que anuncia el nodo, como por ejemplo una impresora. El máximo número de servicios anunciables es 65536 (16 bits).
- **SDM Interval:** Indica cada cuánto tiempo (en segundos) el nodo envía un mensaje de actualización de este servicio al resto de la red. Este parámetro se utiliza para evaluar la cantidad de mensajes perdidos de un determinado servicio y así averiguar si un nodo se ha desconectado

de la red, ya que debido al dinamismo de las redes MANET es bastante común que los nodos entren y salgan de la red constantemente.

- **Life Time:** Indica la disponibilidad (en segundos) del servicio para el resto de nodos. Si pasado este tiempo no se ha recibido ninguna actualización de este servicio se da por hecho que ya no está disponible en la red MANET y se elimina de la caché. Su valor máximo es de 255 segundos.

- **Service Description:** Información detallada del servicio para que el resto de nodos puedan realizar una búsqueda más específica de servicios. En el ejemplo de la impresora, la descripción informaría sobre si ésta imprime en blanco y negro o a color, si a una cara o doble cara, etc.

6.2. Funcionamiento del protocolo SDM

El funcionamiento del protocolo SDM es bastante sencillo, consistiendo principalmente en el anuncio de servicios por parte de los nodos, la recepción y procesamiento de esos mensajes por parte del resto de nodos, la búsqueda en la caché cuando una aplicación requiere de un servicio en concreto y el envío de un mensaje de notificación cuando un servicio deja de estar disponible o el nodo que lo anuncia se va a desconectar de la red MANET.

6.2.1. Anuncio de servicios

Al principio, todas las aplicaciones que utilizan el protocolo SDM anuncian todos los servicios que poseen al resto de nodos de la red tal y como está descrito en la sección 5.3.1 (modo *push*, ver Figura 5.2 en la página 44). Cada servicio se codifica con un número ya establecido en la implementación del protocolo, encapsulando en un paquete OLSR tantos mensajes SDM como servicios anuncie el nodo, incrementándose su tamaño en 8 bytes por cada nuevo servicio anunciado. Por lo tanto el tamaño total del paquete OLSR junto con los mensajes SDM será:

$$Cabecera\ OLSR + Mensaje\ SDM * Numero\ servicios = 16 + 8 * S\ bytes$$

Estos mensajes se enviarán periódicamente para que los demás nodos tengan la información sobre los servicios disponibles en la red lo más actualizada posible. Este intervalo de tiempo común a todos los servicios se denomina `SDM`

`Interval` y es un parámetro configurable por el usuario. Si `SDM Interval` es un período de tiempo muy pequeño, los demás nodos tendrán siempre su tabla de servicios disponibles actualizada, pero el ancho de banda utilizado es mayor. En cambio si `SDM Interval` es muy grande, el ancho de banda utilizado es muy pequeño pero es posible que al no estar la tabla de servicios lo más actualizada posible se produzca un falso positivo, es decir, que el nodo se conecte con un dispositivo para utilizar un servicio y éste ya no está disponible (o el nodo se ha desconectado de la red).

6.2.2. Caché de servicios locales

La primera vez que se anuncia un servicio, éste se almacena en una caché interna del nodo para tener una lista de todos los servicios que está anunciando. Así se podrán crear fácilmente los mensajes SDM cada vez que se actualicen los servicios y además servirá como una primera búsqueda de servicios cuando una aplicación del nodo requiera utilizar uno en concreto. El formato de la caché interna de servicios locales se observa en la Figura 6.3.

< Service Name >	< Life Time >	< Service Description >
Impresora	255 s	Color | Laser

Figura 6.3: Formato de la caché de servicios locales

La caché está compuesta por el nombre del servicio `<Service Name>`, su descripción `<Service Description>` y el tiempo de vida esperado `<Life Time>` (configurable para cada servicio, siendo el máximo 255 segundos para un servicio fijo).

6.2.3. Procesamiento de mensajes SDM

Cuando un mensaje SDM es recibido por un nodo de la red, éste almacena en la caché de servicios disponibles (excluyendo los servicios locales), todos los servicios anunciados en dicho mensaje. Su formato puede observarse en la Figura 6.4.

Si el servicio es nuevo se crea una nueva entrada en la caché guardando el nombre del servicio `<Service Name>`, la dirección IP del nodo que anuncia el servicio `<IP Address>`, el tiempo de vida del servicio `<Life Time>` y la

< Service Name >	< IP Address >	< Life Time >	< Service Description >
Impresora	192.168.x.y	255 s	Color | Laser

Figura 6.4: Formato de la caché de servicios anunciados por el resto de nodos

descripción detallada del servicio `<Service Description>`. Si dicha entrada ya existe al recibir el mensaje SDM, lo único que se hace es actualizar el tiempo de vida del servicio.

También existe otra caché cuya información sirve para detectar si un nodo se ha desconectado de la red al no recibir mensajes SDM de ese nodo. Su formato se ilustra en la Figura 6.5.

< IP Address >	< SDM Interval >	< Last Time >
192.168.x.y	10 s	4 s

Figura 6.5: Formato de la caché de tiempos de cada nodo

Esta caché almacena desde cuánto tiempo hace que se recibió el último mensaje SDM `<Last Time>` por cada nodo de la red `<IP Address>`, así como el intervalo periódico con el que envía actualizaciones de los servicios `<SDM Interval>`. Por lo tanto al saber a qué intervalo se deben recibir los mensajes SDM podremos saber si algún nodo se ha desconectado de la red si no recibimos mensajes SDM de ese nodo en un tiempo determinado.

6.2.4. Búsqueda de servicios

Cuando una de las aplicaciones del nodo requiera un servicio, al ser un protocolo de descubrimiento de servicios de tipo *push* (ver sección 5.3.1), en lugar de mandar un mensaje de petición de servicio al resto de la red buscará en la caché interna del nodo dicha información. En primer lugar se mira en la caché local de servicios por si el propio nodo ya posee dicho servicio. Si el servicio no es prestado por el propio nodo, el protocolo mira en la caché de servicios disponibles en los demás nodos de la red. Si hay alguna coincidencia se le comunica a la aplicación para que pueda conectarse a dicho servicio a través de la red. Si en cambio no hay ningún resultado satisfactorio para la aplicación

se supone que tal servicio no existe en la red MANET y la aplicación no puede hacer uso de él en este momento.

6.2.5. Indisponibilidad de un servicio

Si en un momento dado uno de los servicios que se anuncian en la red no está disponible, el nodo que anuncia dicho servicio tiene la obligación de mandar un mensaje de notificación al resto de nodos de la red para que lo eliminen de sus respectivas cachés internas y de esta manera no se produzcan falsos positivos. Después de enviar el mensaje de notificación el nodo eliminará el servicio que poseía de su caché de servicios locales para que no vuelva a ser anunciado, ahorrando así ancho de banda.

El mensaje de notificación de la indisponibilidad de un servicio tiene la misma estructura del mensaje SDM (ver Figura 6.2), consistiendo en anunciar el servicio que no está disponible con la particularidad de que el campo `Life Time` tiene un valor igual a 0. De esta manera al recibir el mensaje el resto de nodos, actualizarán las entradas respectivas a este servicio, y al ser 0 su tiempo de vida éste será eliminado del registro en el acto.

6.3. SDM en redes MANET

El protocolo de descubrimiento de servicios SDM, al estar integrado en el protocolo OLSR, se centra en redes móviles ad-hoc caracterizadas principalmente por su topología dinámica en la que los nodos entran y salen constantemente de la red. Esto puede provocar que se produzca una tasa de falsos positivos muy alta si las cachés de servicios de los nodos no están lo suficientemente actualizadas. Un ejemplo de falso positivo por no actualizar convenientemente la caché de servicios se observa en la Figura 6.6.

En este ejemplo el nodo A desea un servicio de voz sobre IP, mira en su caché de servicios y descubre que el nodo B posee dicho servicio (ya que fue anunciado previamente como se indica en el apartado 6.2.1). Pero al intentar conectarse con el nodo B para utilizar el servicio VoIP, el nodo A se encuentra con que es imposible realizar la conexión, produciéndose un falso positivo. Esto se ha debido a la movilidad del nodo B, que al irse moviendo por la red MANET ha llegado un momento en que se ha salido del área de cobertura de la red y su rango de transmisión no alcanza a ningún nodo de la red.

Estos falsos positivos no son deseables ya que al intentar conectarse un

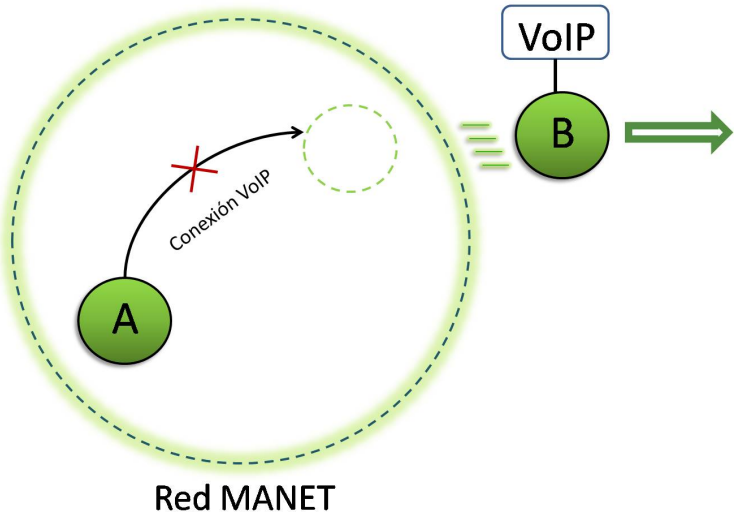

Figura 6.6: Falso positivo al intentar conectarse con un servicio no disponible

nodo con otro en el que el servicio ya no está disponible se incrementa el ancho de banda (el nodo suele reintentar la conexión a los pocos segundos). Este incremento se hace más significativo en una red MANET, donde el ancho de banda es un parámetro fundamental. Además, desde que se intenta la conexión, se produce el fallo y se busca otro nodo que posea el mismo servicio pasa una cantidad de tiempo que puede llegar a resultar molesta para el usuario. Por lo tanto uno de los aspectos del diseño de cualquier protocolo de descubrimiento de servicios para este tipo de redes debe ser intentar eliminar en lo posible estos falsos positivos.

En el protocolo de descubrimiento de servicios SDM, además del tiempo de vida de cada servicio `(Life Time)` y la notificación de servicios no disponibles, se aplica otro método para la detección temprana de falsos positivos. Este método consiste en la detección de pérdida de mensajes SDM de algún nodo perteneciente a la red MANET. En el envío de mensajes SDM, cada nodo indica el intervalo de tiempo entre actualizaciones de los servicios disponibles `(SDM Interval)`, por lo tanto si desde la última recepción de un mensaje SDM ha pasado ese intervalo y no se ha recibido otro mensaje del mismo nodo se supondrá que éste se ha perdido. Si se pierden un número determinado de mensajes SDM de forma consecutiva (parámetro `SDM Lost`, configurable por el usuario), se supone que por alguna razón (movilidad, batería, etc.) el nodo se ha desconectado de la red MANET y se borran de la caché todos los servicios disponibles de dicho nodo aunque su tiempo de vida `(Life Time)` sea mayor que 0.

Al ser este tipo de redes de naturaleza inalámbrica, la probabilidad de que se pierda algún paquete es mayor que en las redes cableadas. Veamos que ocurre si se pierde algún paquete:

- **Anuncio de servicios:** Si el paquete que se pierde es un mensaje SDM anunciando servicios no afecta al funcionamiento del protocolo, ya que por lo general el tiempo de vida (`Life Time`) es mayor que el intervalo de tiempo entre 2 actualizaciones (`SDM Interval`), y para que se borren los servicios de un nodo deben perderse varios mensajes SDM de forma consecutiva como se ha explicado anteriormente.

- **Notificación de servicio no disponible:** Si el paquete que se pierde es un mensaje SDM notificando que un servicio del nodo no va a estar más tiempo disponible, los demás nodos no podrán borrar de sus cachés dicho servicio, teniendo que esperar a que caduque su tiempo de vida (`Life Time`) para poder borrarlos. En este caso la probabilidad de exista un falso positivo aumenta.

6.4. Resumen

En este capítulo se ha explicado el funcionamiento del protocolo de descubrimiento de servicios SDM, pensado para ser utilizado en redes MANET debido a su escasa utilización de ancho de banda (cada servicio sólo ocupa 8 bytes). El precio a pagar por ello es una codificación de los servicios (cada uno posee un número determinado) que provoca que todos los nodos deban tener la misma implementación del protocolo para que no haya incongruencias en el anuncio de servicios. Además las descripciones que se pueden dar de cada servicio no pueden tener muchos detalles a no ser que también vayan codificadas (por lo que tendría que haber un mayor convenio entre las diferentes implementaciones del protocolo).

SDM es un protocolo sencillo, destinado a redes no muy complicadas en las que no se vayan a anunciar muchos tipos de servicios diferentes, pero que gracias a su integración en el protocolo OLSR y su naturaleza proactiva le hace muy adecuado para aquellas redes con mucha movilidad y una topología de red muy dinámica. Además esta proactividad puede ser configurada por el usuario a través de los parámetros `SDM Interval` (intervalo de tiempo entre dos actualizaciones de servicios) y `SDM Lost` (número de mensajes consecutivos perdidos para que se consideere a un nodo fuera de la red MANET), a cambio de un mayor o menor consumo del ancho de banda.

Por último cabe destacar que la definición realizada del protocolo SDM en este capítulo no se corresponde exactamente con la provista por [52]. Tanto la notificación de un servicio que deja de estar disponible como la utilización del parámetro `SDM Interval` para descubrir que un nodo ha dejado la red MANET (y evitar así falsos positivos) son especificaciones diseñadas por el autor de este proyecto.

Capítulo 7

Implementación de SDM

En este capítulo se explicará la implementación del protocolo *Service Discovery Mechanism* (SDM) para dispositivos reales (en nuestro caso para un Nokia N800), su arquitectura y los algoritmos más importantes para otorgar la funcionalidad deseada a la aplicación. Al ser un protocolo de descubrimiento de servicios integrado en el protocolo de routing OLSR, necesitaremos una implementación de dicho protocolo que sea flexible y permita la comunicación con la implementación del protocolo SDM. En nuestro caso utilizaremos el demonio `Unik olsrd` desarrollado por Andreas Tønnesen [25]. Esta implementación posee una interfaz de plugins que permite añadir nuevas funcionalidades al protocolo OLSR sin tener que modificar el código del demonio olsrd. Así la implementación de SDM está pensada como un plugin para este demonio.

A continuación se realizará una breve descripción del demonio olsrd, su interfaz de plugins y una explicación detallada de toda la implementación del protocolo SDM para olsrd.

7.1. Implementación de OLSR

Desde el momento en que apareció el protocolo OLSR muchas implementaciones han sido creadas, tanto para ser probadas en simuladores de red como para ser utilizadas en escenarios con dispositivos reales.

Con respecto a las implementaciones diseñadas para ser utilizadas en simuladores de red como ns-2 [38], la más destacada es la realizada por la Universidad de Murcia (UM-OLSR [36]). Entre las implementaciones para ser instaladas en dispositivos reales se encuentran la realizada por el equipo Hipercom en INRIA (OOLSR [37], en lenguaje C++), la realizada por Andreas Tønnesen

para su Master Thesis (olsrd [6], en lenguaje C) y la implementada por la Naval Research Laboratory de la U.S. army (NRL-OLSR [39], en lenguaje C++).

De entre todas las implementaciones citadas anteriormente, la utilizada en este proyecto para estudiar las prestaciones del protocolo OLSR en dispositivos Maemo es el demonio olsrd implementado por Andreas Tønnesen para su tesis de Máster [25].

La elección de esta implementación con respecto a las demás existentes se debe principalmente a dos factores:

- El demonio olsrd está implementado principalmente para la plataforma GNU/Linux. Al estar realizado en el lenguaje de programación C, tiene muy pocas dependencias aparte de la librería estándar, siendo la principal dependencia la librería de hilos POSIX *libpthread*. Por lo tanto debería compilar en todos los modernos sistemas de GNU/Linux. Como la plataforma de desarrollo Maemo está basada en la plataforma GNU/Linux, en principio no debería existir ningún problema de compatibilidad ni comprometer su funcionalidad al ser exportado a esta tecnología.

- La implementación de olsrd posee una interfaz para la carga de plugins (es decir, librerías que se pueden cargar dinámicamente en tiempo de arranque) con la que cualquiera puede ajustar, modificar o extender el protocolo OLSR sin tener que cambiar ninguna línea de código original del núcleo de olsrd. Además a través de esta interfaz de plugins un programador puede aprovechar la estructura del protocolo OLSR para diseñar diferentes funcionalidades, aunque no estén estrictamente relacionadas con el routing. Esta interfaz de plugins será utilizada para la implementación del mecanismo de descubrimiento de servicios.

  En [40] se da una explicación detallada del funcionamiento de esta interfaz de plugins para el demonio olsrd, donde además se proponen ciertas extensiones, como por ejemplo un mecanismo de seguridad para el protocolo OLSR, un mecanismo de autoconfiguración de nodos, un plugin para saber el nivel de la batería de cada dispositivo y un plugin para detectar dinámicamente nodos que sirvan como puertas de enlace (*gateways*) a Internet.

7.2. Características de olsrd

Las características más importantes de esta implementación de OLSR son:

- **Modularidad:** El demonio olsrd está basado en módulos más o menos independientes que interactúan entre sí, por lo que se pueden añadir o quitar varios módulos al programa sin que afecte a su integridad. Esta característica es muy importante para la carga de *plugins*, que son módulos externos cargados en tiempo de arranque para añadir o cambiar funcionalidades al programa.

- **Estructuras de datos consistentes:** En cualquier protocolo de routing, la mayor parte de la computación es realizada en tablas o listas (como por ejemplo las tablas con los vecinos del nodo, las tablas de reenvío, etc.). Por lo tanto, estas tablas deben ser consistentes, teniendo todas ellas la misma estructura.

- **Transparencia IP:** En principio, el demonio olsrd debería funcionar para cualquier versión IP ya sea IPv4 o IPv6. Por ello todas las funciones tienen que ser implementadas para manejar direcciones tanto de 32 bits (IP versión 4) como direcciones de 128 bits (IP versión 6).

- **Código independiente de la plataforma:** Todo el código que sea dependiente de la máquina en la que se está procesando el demonio debería ir encapsulado en un módulo para separarlo del resto del código independiente de la plataforma. De esta manera la portabilidad es más sencilla de realizar al tener sólo que implementar el código dependiente de cada máquina en concreto.

7.3. Módulos del demonio olsrd

Una vez enunciadas las características más importantes, se explicará brevemente cada una de las partes en las que consiste el demonio olsrd para entender cómo funciona. Los módulos principales del demonio olsrd se representan en la Figura 7.1.

7.3.1. Socket Parser

El socket parser es el encargado de gestionar un conjunto de sockets que *escuchan* el enlace para recibir los paquetes procedentes de otros nodos. Por lo tanto, la tarea del socket parser es detectar si alguno de los sockets registrados ha recibido algún paquete y enviárselo al OLSR packet parser para que analice el tráfico de datos recibido. Los sockets pueden ser registrados (si se inicia una

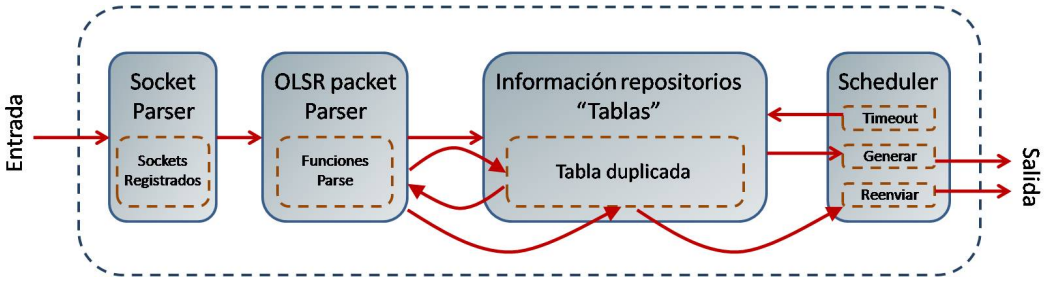

Figura 7.1: Esquema del demonio olsrd, basado en [25]

nueva conexión) o eliminados (si se pierde la conexión) en cualquier momento, pudiendo encargarse tanto de conexiones tipo cliente como de tipo servidor.

7.3.2. OLSR Packet Parser

El OLSR packet parser se encarga de analizar todo el tráfico recibido por los sockets, pudiendo realizar las siguientes acciones:

- Descartar el paquete si éste no es válido.
- Procesar el paquete si su contenido es un mensaje OLSR.
- Reenviar el paquete si éste es válido pero no contiene ningún mensaje concerniente al protocolo OLSR.

Si el paquete debe ser procesado porque ha sido recibido por el puerto UDP 698 y contiene cualquier tipo de información relacionado con el protocolo OLSR (mensaje `HELLO`, `TC` o `MID`), se actualizan las tablas correspondientes en el módulo de repositorios de información según el procedimiento descrito en el RFC 3626 [1].

7.3.3. Repositorio de información (tablas)

Una vez que se ha procesado el mensaje OLSR, hay que almacenar toda la información obtenida o actualizarla en las diferentes tablas o registros que existen según el tipo de mensaje (la tabla de vecinos de primer salto, de segundo salto, de topología de red, etc.). Tal y como se mencionó en el capítulo 3, estas tablas contienen toda la información necesaria para conocer el actual estado de la red y su topología.

Si ha habido cambios en alguna de estas tablas se procederá a recalcular todas las rutas posibles, incluyendo el cálculo de los nodos MPR. Las entradas de todas estas tablas poseen una etiqueta de tiempo (*timestamp*) donde se indica la última vez que se actualizó dicha entrada. De esta manera se puede verificar la validez de cada entrada, teniendo que ser borradas de sus respectivas tablas si su *timestamp* ha caducado.

7.3.4. Scheduler

El scheduler es el módulo que se encarga de gestionar los diferentes eventos que se van produciendo a lo largo de la ejecución del demonio olsrd. Para que cualquier evento pueda ser lanzado por el scheduler, antes debe ser registrado por el módulo de olsrd que quiera ejecutar dicho evento en el momento indicado. Por defecto, el scheduler comprueba si hay registrado un nuevo evento cada 0.1 segundos, aunque este parámetro puede ser configurado por el usuario según sus necesidades.

Los eventos más habituales ejecutados por el scheduler son:

- Transmisión de un mensaje en un intervalo de tiempo determinado, registrando la generación de dicho mensaje en el scheduler.
- Comprobar si alguna entrada en las tablas de los repositorios de información ha caducado, declarándola por tanto no válida y eliminándola de su tabla correspondiente. Esto se realiza registrando una función de *timeout* en el scheduler.

Una vez que ha puesto en marcha todos los eventos programados para ese momento, el proceso del scheduler se duerme (es decir, pasa a un estado inactivo) durante un intervalo de tiempo determinado (por defecto 0.1 segundos), volviendo a activarse pasado ese tiempo y comprobando si hay algún nuevo evento que gestionar.

7.4. Interfaz de plugins

La gran ventaja que posee el demonio olsrd frente a otras implementaciones del protocolo OLSR es su interfaz de plugins, permitiendo extender el código del protocolo sin tener que alterar el núcleo de olsrd (lo que podría ocasionar situaciones de inestabilidad si no se maneja con cuidado). Esta interfaz soporta la carga de librerías de forma dinámica para hacer uso de funciones auxiliares

creadas por personas ajenas a la implementación de olsrd (plugins). También se pueden *suplantar* los métodos ya implementados en el núcleo de olsrd para añadir, cambiar o extender su funcionalidad.

Una librería cargada dinámicamente (del inglés *Dynamically Loadable Library* (DLL)) es una porción de código que contiene datos y funciones pero que a diferencia de una aplicación, este tipo de librerías no se enlazan después de la compilación, sino en el momento en que se carga la aplicación. De entre todas las librerías DLL que existan para una aplicación, solamente se cargarán aquellas indicadas en los archivos de configuración de dicha aplicación. En el caso de olsrd, el archivo de configuración es `olsrd.conf`, que se encuentra en la carpeta `/etc`.

Debido a que el protocolo OLSR permite el reenvío de paquetes aunque su tipo sea desconocido para el nodo MPR, los plugins pueden dividirse en dos grandes grupos:

- Aquellos que cambian o extienden la funcionalidad propia del protocolo OLSR, pudiendo por ejemplo cambiar la estructura del mensaje `HELLO` o `TC`, el método de reenvío de paquetes OLSR a través de los nodos MPR, la manera de ajustar el nivel de voluntad *(willingness)* de cada nodo según su nivel de batería o el diseño de las tablas que contienen la información de routing.

- Aquellos que se aprovechan del método de reenvío de paquetes para crear sus propios tipos de mensaje y diseminarlos por toda la red. Estos mensajes auxiliares pueden ser utilizados para proveer un dominio de nombres de servicio (DNS) en una red MANET, compartir la información de la batería entre los nodos [25] o diseminar información sobre anuncio y búsqueda de servicios [53]. La implementación del protocolo SDM pertenece a este segundo grupo de plugins.

Para realizar todos estos cambios ya sea de un grupo u otro, la interfaz de plugins, siguiendo las instrucciones dadas por el plugin cargado, registra y/o quita funciones tanto en el socket parser (interceptando los mensajes de entrada) como en el packet parser o en el scheduler, pudiendo también actualizar muchas variables presentes en las tablas que maneja el protocolo de routing. Además puede manipular todo los paquetes OLSR que salgan del nodo con destino el resto de la red (cambiando cualquier valor de cualquier campo OLSR). En resumen, un plugin puede virtualmente manipular cada parte del demonio olsrd, tal y como se observa en la Figura 7.2.

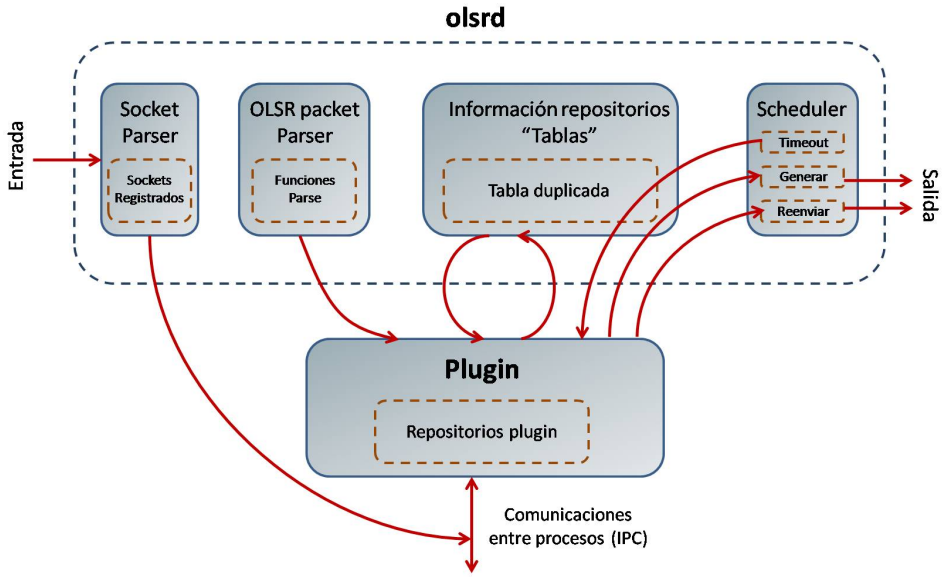

Figura 7.2: Interacción entre la interfaz de plugins y olsrd. Figura basada en [25]

Un ejemplo de un método del demonio olsrd modificado a través de la interfaz de plugins para la implementación del protocolo SDM es `olsr_parser`. Este método sirve para procesar un paquete SDM y obtener toda la información de los servicios anunciados por el resto de nodos de la red. Además reenvía el paquete SDM si ha sido elegido previamente como nodo MPR a través del método `olsr_forward_message`. Esta función es añadida al módulo de packet parser en la inicialización del plugin, mientras que los métodos que actualizan las cachés periódicamente se registran en el scheduler.

7.5. Visión general de la implementación

Aunque el demonio olsrd está escrito en C, los plugins que se añaden pueden ser escritos en cualquier lenguaje de programación que pueda ser compilado como una librería dinámica (archivos .dll en Windows o .so en Linux). Aún así la implementación del protocolo de descubrimiento de servicios SDM también está escrito en C debido a que necesita muy pocas dependencias y la plataforma Maemo, al estar diseñada para dispositivos móviles, tiene menor número de librerías y software por defecto que los ordenadores convencionales. De esta manera es más fácil portarla a otras plataformas y sistemas operativos.

El código fuente de la implementación está contenido en la carpeta `/src`,

que se compone de los siguientes archivos:

- `sdm_plugin.h`: Cabecera donde se definen las estructuras de datos de las cachés de servicios, tanto la local como las anunciadas por el resto de nodos, así como la estructura del mensaje SDM.
- `sdm_plugin.c`: Contiene las funciones principales del protocolo SDM: creación y envío de mensajes SDM, obtención de información de servicios a partir de un mensaje SDM recibido, actualización de las distintas cachés, detección de la no disponibilidad de un servicio o nodo, etc.
- `ipc_socket.h`: Cabecera donde se indica el número máximo de aplicaciones que pueden anunciar y buscar servicios en un nodo y el puerto de acceso al plugin (en nuestro caso el puerto 5555).
- `ipc_socket.c`: Contiene los métodos que gestionan la comunicación entre las aplicaciones que anuncian y buscan servicios con el plugin SDM a través de sockets IPC (*InterProcess Communication*). Este archivo contiene funciones para abrir, configurar y cerrar sockets, así como el envío y recepción de mensajes a través de ellos.
- `services.h`: Cabecera donde van codificados los posibles servicios que pueden ser anunciados por un nodo.
- `services.c`: Contiene los métodos que sirven para codificar (al enviar) y decodificar (en recepción) los posibles servicios que pueden ser anunciados por un nodo.
- `Makefile`: Archivo que sirve para compilar el plugin SDM y crear la librería dinámica correspondiente.

Esta carpeta `/src` debe estar incluida en la carpeta `/lib` dentro de la carpeta del demonio olsrd junto con el resto de plugins. Para compilar e instalar el plugin SDM se ejecuta el archivo Makefile de la siguiente manera:

```
.../src$ make
.../src$ make install
```

Al ejecutar estos comandos se crea la librería dinámica del plugin SDM denominada `olsrd_sdm.so.x.y` (donde `x.y` es la versión del plugin), colocándolo en la carpeta `/usr/lib`. Este plugin se carga al inicio del demonio olsrd si se indica en el archivo de configuración `/etc/olsrd.conf` de acuerdo con [25], teniendo que dar un valor a los dos parámetros configurables por el usuario: `SDM Interval` y `SDM Lost`.

7.6. Arquitectura del plugin SDM

La arquitectura de la implementación del plugin SDM para el demonio olsrd con sus principales componentes se ilustra en la Figura 7.3.

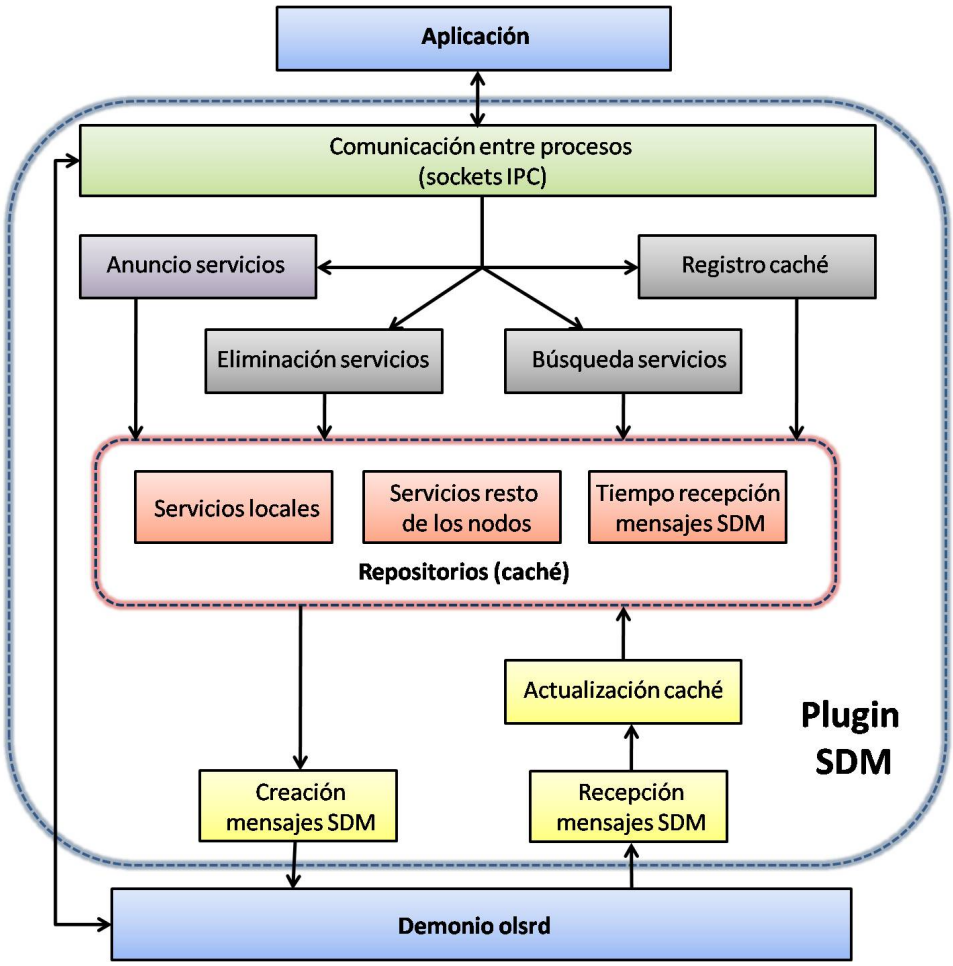

Figura 7.3: Arquitectura de la implementación del plugin SDM

Las partes más importantes de esta arquitectura son las siguientes:

- **Comunicación entre procesos (Sockets IPC):** Este módulo gestiona la comunicación bidireccional entre el plugin SDM y las aplicaciones (situadas en capas superiores) que anuncian o requieren servicios del resto de la red. También gestiona parte de la comunicación entre el demonio olsrd y el plugin SDM.

- **Funciones de gestión de servicios:** Estas funciones realizan las acciones que desean realizar las aplicaciones (o el propio demonio olsrd, por ejemplo indicándole al plugin que hay que anunciar servicios cada cierto período de tiempo). Estas acciones van desde anunciar servicios, eliminarlos de la caché enviando un mensaje de notificación, buscar servicios en la caché, mirar el contenido de la caché, etc.

- **Repositorios (caché):** Los repositorios son tablas donde está contenida toda la información acerca de los servicios y de los nodos que los anuncian para el correcto funcionamiento del protocolo SDM. Se dividen en tres tablas:

  - La caché de servicios locales, donde se almacenan los servicios que posee el propio nodo.
  - La caché de servicios anunciados por el resto de nodos en la red y que están disponibles para ser utilizados por las aplicaciones de las capas superiores.
  - La caché donde se almacenan los tiempos de recepción de mensajes SDM por cada uno de los nodos de la red para detectar una posible desconexión de dichos nodos (ver sección 6.3).

- **Interfaz del demonio olsrd:** Módulo que se encarga principalmente del envío de mensajes SDM una vez que han sido creados a partir de la caché de servicios locales y de la recepción de mensajes SDM de otros nodos para obtener todos los servicios disponibles en la red y almacenarlos en la caché correspondiente (siendo actualizados si dichas entradas ya existían).

En los próximos apartados cada uno de estos módulos será descrito detalladamente.

7.7. Interfaz del demonio olsrd

Aunque el interfaz con el demonio olsrd se encarga tanto del envío de mensajes SDM ya creados previamente como el procesamiento de los mensajes SDM recibidos, en este apartado sólo nos centraremos en la recepción ya que el envío consiste simplemente en empaquetar el mensaje SDM en un paquete OLSR y enviarlo al resto de nodos de la red por la interfaz correspondiente.

El objetivo de la recepción de mensajes SDM es obtener toda la información posible de ese mensaje y actualizar en consonancia los repositorios de servicios para que así cualquier aplicación del nodo, al buscar cualquier servicio que necesite, siempre obtenga la información más reciente y no se produzca un falso positivo.

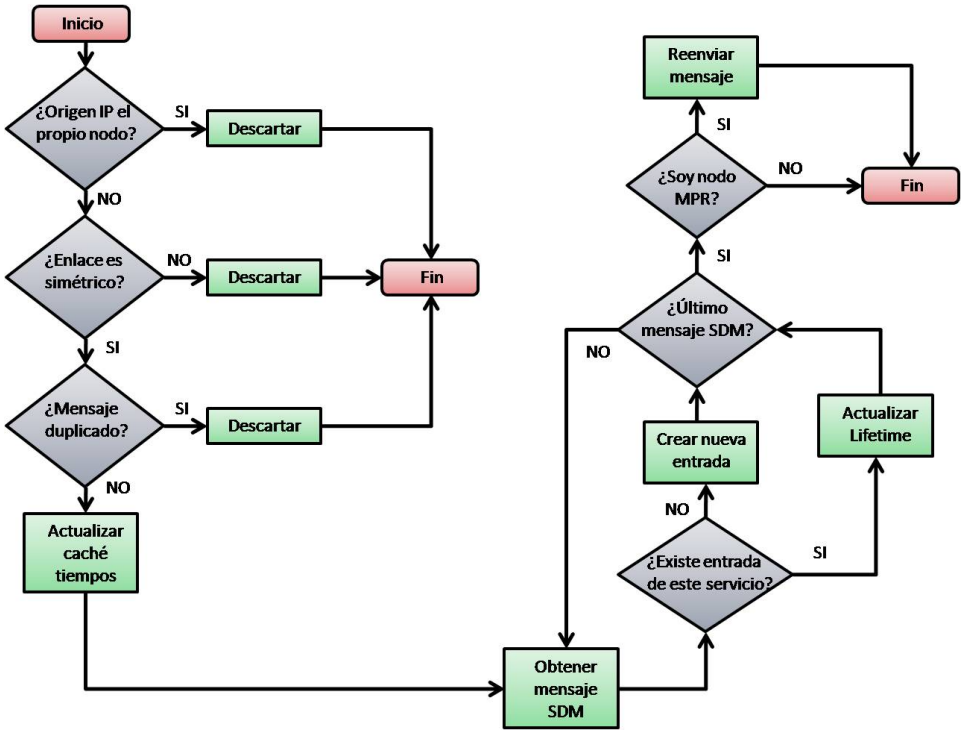

Figura 7.4: Diagrama de flujo de la recepción de mensajes SDM

El procesamiento de un mensaje SDM recibido se muestra en el diagrama de flujo de la Figura 7.4, que se puede distribuir en cuatro etapas:

1. Comprobación de que el mensaje SDM recibido es correcto y está preparado para su procesamiento. En primer lugar se comprueba si la IP del paquete es la del propio nodo, descartándolo si fuera verdad. Después se comprueba si el enlace entre el nodo que envió el paquete y el propio nodo es simétrico, descartándolo si no es así. Por último se mira en el número de secuencia del paquete para comprobar si este mensaje ya ha sido procesado previamente o no (mensaje duplicado), descartándolo si ya ha sido procesado.

2. Si en la comprobación resulta que el mensaje SDM es correcto, antes de obtener información de cualquier servicio se actualiza la caché de tiempos

donde se almacena la última vez que se recibió un mensaje SDM de cada nodo para poder detectar desconexiones en la red. También se almacena en esta caché el intervalo de tiempo supuesto entre dos mensajes SDM de cada nodo.

3. Una vez actualizada la caché de tiempos, se procede a la obtención de la información de los servicios anunciados por el nodo en cuestión. Por cada servicio anunciado se comprueba si ya existe tal entrada en la caché, siendo actualizado el tiempo de vida (`Life Time`) si es así o creando una nueva entrada en la caché si es la primera vez que se tiene constancia de la disponibilidad de ese servicio por ese nodo. Esta misma operación se realiza con cada uno de los servicios anunciados por el nodo.

4. Por último se comprueba si el nodo que recibe el mensaje SDM ha sido elegido como nodo MPR por algún vecino según el criterio propuesto por el protocolo OLSR (ver sección 3.1.2). Si el nodo es MPR, reenviará el mismo mensaje SDM a todos sus vecinos para que también actualicen sus respectivas cachés y así se difunda el mensaje SDM por toda la red.

7.8. Repositorios (caché)

Los repositorios son las tablas donde se almacena toda la información relativa a los servicios locales, a los servicios anunciados por otros nodos y al momento en que se recibió el último mensaje SDM de cada nodo. En este apartado se explicará cómo se ha implementado cada una de estas tablas y qué estructura se ha elegido para representar estos datos.

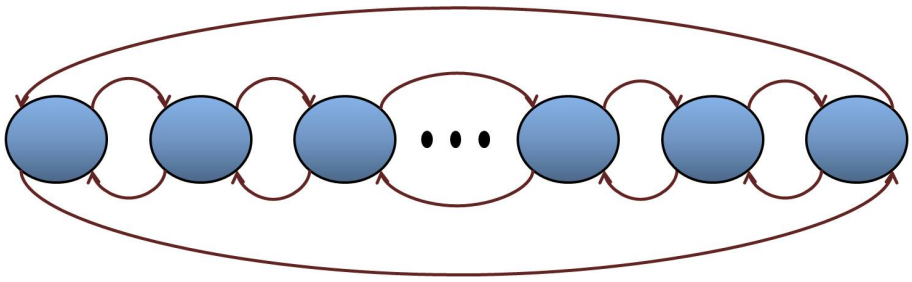

Figura 7.5: Doble lista enlazada

El elemento principal para almacenar los datos en los repositorios es la doble lista enlazada (ver Figura 7.5), que consiste en un conjunto de entradas que, además de toda la información necesaria en esa lista, posee información

adicional: un puntero al elemento previo de la lista y otro puntero al siguiente elemento de la lista. De esta manera mirando la información contenida en un elemento de la lista es posible dirigirte o bien al elemento previo o bien al siguiente elemento de la lista, pudiendo recorrer así la lista de una manera fácil y sencilla.

Después de considerar otras opciones como arrays o árboles binarios, se ha utilizado esta estructura de datos sobre todo por ser dinámica (no hace falta establecer en su inicialización un tamaño máximo de entradas) y por no ser necesario el conocimiento de otros elementos de la lista excepto el actual (muy adecuado para eliminar elementos de la lista). Además el orden de los elementos de la lista enlazada puede ser diferente del orden de los elementos almacenados en memoria, permitiendo la inserción y eliminación de entradas en cualquier punto de la lista.

7.8.1. Caché de servicios locales

La caché de servicios locales es una doble lista enlazada, donde cada elemento está definido en el plugin SDM como una estructura en C, cuyos campos son:

```
struct ownservice {
  int serviceType;
  long serviceDescription;
  int lifetime;
  int socket[MAX_IPC_CLIENTS];
  int advertisingSockets;
  struct ownservice *next;
  struct ownservice *prev;
};
```

`Service Type` es el servicio anunciado por el propio nodo, `Service Description` es la descripción de dicho servicio y `LifeTime` es el tiempo de vida estándar del servicio (el valor que se enviará en el mensaje SDM al resto de nodos). `Socket[MAX_IPC_CLIENTS]` es un array donde se almacenan los identificadores de las aplicaciones que anuncian dicho servicio y `AdvertisingSockets` es el número de aplicaciones almacenados en el array (explicación del funcionamiento de los sockets en el apartado 7.10). Por último `Next` y `Prev` son los elementos siguiente y anterior respectivamente de la lista doble enlazada.

7.8.2. Caché de servicios anunciados por otros nodos

La caché de servicios anunciados por otros nodos consiste en una lista doble enlazada donde cada elemento representa un servicio determinado. Además cada elemento tiene un puntero a otra lista doble enlazada donde se almacenan las direcciones IP de aquellos dispositivos que anuncian dicho servicio, la descripción del servicio que anuncian y el momento de tiempo en que se actualizó dicha entrada (ver Figura 7.6).

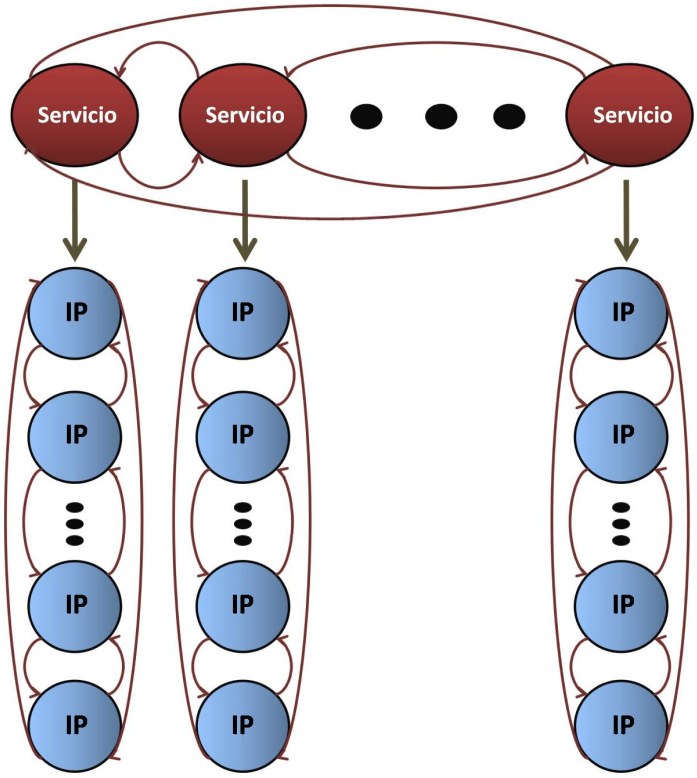

Figura 7.6: Estructura de datos de la caché de servicios externos

Cuando un nodo desea buscar un servicio determinado, el proceso de búsqueda en la caché de servicios anunciados por otros nodos que realiza el plugin SDM es el siguiente:

1. Recorre la lista doble enlazada (principal) donde se almacenan los servicios hasta que encuentra aquel servicio que está buscando.

2. Una vez encontrado el elemento correspondiente al servicio buscado, pasa a la lista doble enlazada (secundaria) almacenada en ese elemento y la

recorre para mostrar los datos de todos los nodos que anuncian dicho servicio en la red.

Al principio se pensó en implementar esta caché como un array donde cada posición se correspondería con un servicio determinado, almacenando en cada una de dichas posiciones una lista doble enlazada con los datos de los nodos que anunciaban dicho servicio, ya que así se aceleraría el proceso de búsqueda de servicios al realizar en un solo paso la localización del servicio que se está buscando. Pero al ser el número máximo de servicios 65536, habría que inicializar entonces un array de 65536 posiciones donde probablemente sólo se utilizarían unas pocas, desperdiciando mucha memoria del dispositivo, aspecto bastante crítico en los dispositivos móviles utilizados en redes MANET.

Implementando la caché con una lista doble enlazada en lugar de un array, el tiempo de proceso de búsqueda de un servicio en concreto es algo más lento pero a cambio se ahorra muchísimo espacio en la memoria del dispositivo móvil.

Cada elemento de la lista doble enlazada principal (aquella que almacena los servicios) está definido en el plugin SDM como una estructura en C, cuyos campos son:

```
struct foreignservice {
  int serviceType;
  struct sdentry *node;
  struct foreignservice *next;
  struct foreignservice *prev;
};
```

`ServiceType` es el servicio que representa a ese elemento de la lista (codificado con un número definido por el protocolo), `Node` es el puntero al primer elemento de la lista doble enlazada que almacena la información de los dispositivos que anuncian dicho servicio. Por último `Next` y `Prev` son los elementos siguiente y anterior respectivamente de la lista doble enlazada.

Cada elemento de la lista doble enlazada secundaria (aquella que almacena los datos de los nodos que anuncian dicho servicio) está definido en el plugin SDM como una estructura en C, cuyos campos son:

```
struct sdentry {
  union olsr_ip_addr originator;
  long serviceDescription;
  time_t timer;
```

```
  struct sdentry *next;
  struct sdentry *prev;
};
```

`Olsr_IP_addr` es la dirección IP del nodo que anuncia el servicio, `Service Description` es la descripción de dicho servicio y `Timer` es el momento de tiempo en el que se recibió el último mensaje SDM anunciando dicho servicio (pudiendo calcular así el tiempo de vida `(Life Time)` mediante la diferencia entre el momento actual y `Timer`). Por último `Next` y `Prev` son los elementos siguiente y anterior respectivamente de la lista doble enlazada.

7.8.3. Caché de tiempos

La caché de tiempos, al igual que la caché de servicios locales, es una doble lista enlazada, donde cada elemento está definido en el plugin SDM como una estructura en C, cuyos campos son:

```
struct sdmupdate {
  union olsr_ip_addr originator;
  time_t updateTime;
  int sdmInterval;
  struct sdmupdate *next;
  struct sdmupdate *prev;
};
```

`Olsr_IP_addr` es la dirección IP del nodo que anuncia el servicio, `Update Time` es el momento de tiempo en el que se recibió el último mensaje SDM de dicho nodo y `SDMInterval` es el intervalo de tiempo que hay que esperar para recibir otro mensaje SDM del mismo nodo. Por último `Next` y `Prev` son los elementos siguiente y anterior respectivamente de la lista doble enlazada.

7.9. Funciones de gestión de servicios

En este apartado se explicará cómo se ejecuta cada acción que puede realizar el plugin SDM tanto para anunciar, buscar o eliminar servicios. Estas funciones por lo tanto añaden, recorren y eliminan datos de la caché según lo que la aplicación requiera.

7.9.1. Anuncio de servicios

El plugin SDM puede anunciar los servicios disponibles en el nodo tanto si una aplicación ha añadido un nuevo servicio o si es el momento de enviar una actualización de los servicios al resto de la red (cada *SDM Interval*). En ambos casos el plugin SDM realiza lo siguiente:

1. Empieza a recorrer la caché de servicios locales y por cada entrada crea un mensaje SDM, definido en el plugin como una estructura en C, cuyos campos son:

   ```
   struct sdmsg {
     uint16_t serviceType;
     uint8_t sdmInterval;
     uint8_t lifetime;
     uint32_t description;
   };
   ```

   `ServiceType` es el tipo de servicio que se anuncia codificado en 16 bits, `SDMInterval` es el intervalo entre dos mensajes SDM (8 bits), `LifeTime` es el tiempo de vida del servicio (8 bits) y `Description` es la descripción del servicio (32 bits). La estructura del mensaje SDM se puede ver con mayor detalle en la sección 6.1.

2. Junta todos los mensajes SDM en un paquete OLSR, rellenando los campos que sean necesarios tales como la dirección IP del nodo o el número de secuencia (ver apartado 3.2).

3. Envía el paquete OLSR por la interfaz correspondiente hacia el resto de la red MANET.

7.9.2. Búsqueda de servicios

Cuando una aplicación requiere la utilización de un servicio, primero se recorre la caché de servicios locales por si hay alguna coincidencia, enviándosela en tal caso a la aplicación a través del socket via IPC. Tanto si existe en la caché de servicios locales como si no, se mira en la caché de servicios anunciados por el resto de nodos, recorriendo la lista enlazada primaria hasta encontrar el servicio correspondiente y enviando después la información de cada uno de los elementos de la lista secundaria (pues corresponden al mismo servicio). Se deja en poder de la aplicación el elegir qué servicio es el más apropiado según su descripción (o su proximidad).

7.9.3. Eliminación de servicios

En el momento en que una aplicación deja de anunciar un servicio (debido a que éste ya no está disponible), el plugin SDM recorre la caché de servicios locales y elimina el servicio no disponible. Después crea un mensaje SDM como el visto en el apartado anterior, poniendo un valor de `Life Time` igual a 0, lo introduce dentro de un paquete OLSR y lo envía por la interfaz correspondiente hacia el resto de la red MANET.

7.9.4. Actualización de la caché

Para un correcto funcionamiento del protocolo SDM es imprescindible la actualización de la caché para descubrir nuevos servicios y prevenir falsos positivos. La caché se actualiza en los casos siguientes:

1. Cuando se recibe un mensaje SDM con los servicios anunciados por otro nodo. En este caso se procesan todos los mensajes SDM encapsulados en el paquete OLSR y se actualiza el tiempo de vida de cada servicio (si ya existía esa entrada) o se añade un nuevo registro a la caché en caso contrario.

2. Cada 0.5 segundos se recorre tanto la caché de servicios locales como la de servicios anunciados por otros nodos para detectar si algún servicio ha expirado (es decir, su `Life Time` es igual o menor que 0). Para calcular el tiempo de vida se realiza la diferencia entre el momento actual y el momento en que se recibió el último mensaje SDM anunciando dicho servicio.

   También se recorre la caché de tiempos cada 0.5 segundos para detectar si se ha producido la desconexión de algún nodo, calculando para ello el tiempo que ha pasado desde que se recibió el último mensaje SDM de ese nodo. Si excede del límite (`SDM Lost` $*$ `SDM Interval`) se considera que el nodo ya no está disponible en la red, eliminándose de la caché todos aquellos servicios que anunciaba dicho nodo.

Se contempló la posibilidad de actualizar la caché, en lugar de periódicamente cada 0.5 segundos, cada vez que se produjera un evento, es decir, cada vez que se enviara o recibiera un mensaje SDM, que se añadiera, buscara o eliminara un servicio, que se mostrara en pantalla el contenido de alguna caché, etc. Realizándolo de esta manera se asegura que se actualiza la caché el menor número de veces posible permaneciendo ésta estable, ya que hasta que no

se produce alguno de estos eventos no se accede a la caché. Así el tiempo de procesamiento del plugin SDM es menor, pero en cambio su implementación es bastante más difícil ya que se tiene que comprobar que antes de cada uno de los posibles eventos se actualiza la caché, pues si no daría lugar a inestabilidades e incongruencias de la información almacenada. Como el intervalo entre dos eventos no suele ser muy grande (se envían y reciben mensajes SDM constantemente), se decidió que no merecía la pena complicar la implementación del plugin SDM para ahorrar algo de tiempo de procesamiento.

7.9.5. Observación del contenido de la caché

Si la aplicación lo requiere, en lugar de buscar un servicio concreto, existe la posibilidad de mirar el contenido de la caché para conocer los servicios que están disponibles, tanto locales como anunciados por el resto de nodos. Esta observación consiste en recorrer las listas enlazadas de la caché correspondiente y enviar a la aplicación toda la información almacenada (tipo de servicio, descripción, dirección IP, etc.).

7.10. Comunicación entre procesos (IPC)

La conexión entre las aplicaciones y el plugin SDM para anunciar y buscar servicios se realiza a través de una función de comunicación entre procesos (IPC - *InterProcess Communication*). IPC es un conjunto de técnicas para el intercambio de información entre procesos o aplicaciones, pudiendo estar éstas en el mismo terminal o en otro conectado a través de la red. Entre las técnicas más conocidas de IPC se encuentran el paso de mensajes, la utilización de semáforos, llamadas a procedimiento remoto (RPC), etc. En nuestro caso emplearemos sockets para la comunicación IPC entre las aplicaciones y el plugin SDM, utilizando para ello el protocolo TCP/IP a través de la interfaz de *loopback* (interfaz virtual que representa al propio ordenador, siendo 127.0.0.1 su dirección IP).

Los sockets implementan una arquitectura cliente-servidor, donde existe un servidor que está *escuchando* (en nuestro caso el plugin SDM) hasta que un cliente quiere empezar una comunicación (en nuestro caso la aplicación de usuario), iniciándose una conexión en la que tanto cliente como servidor pueden transmitirse información a través de la red (o a través del propio ordenador si se utiliza la interfaz de *loopback*).

El plugin SDM permite que varias aplicaciones se conecten de manera si-

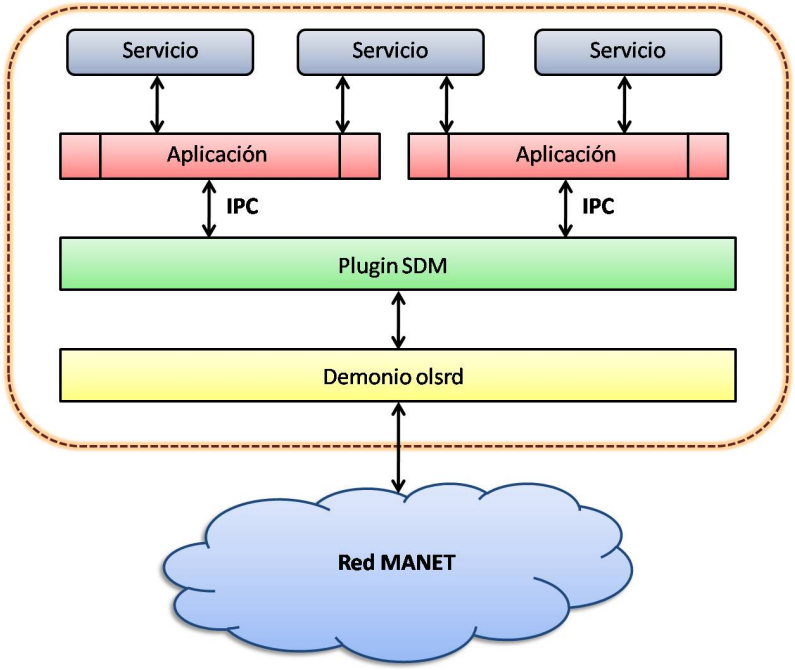

Figura 7.7: Estructura de la conexión de las aplicaciones al plugin SDM

multánea, estando limitado su número por el parámetro `MAX_IPC_CLIENTS` definido en el fichero `ipc_socket.h`. Estas aplicaciones anuncian uno o varios servicios, pudiendo compartir dos aplicaciones el mismo servicio, como se observa en la Figura 7.7.

La conexión al plugin se establece mediante un socket TCP al propio nodo (*localhost*) a través de la interfaz *loopback* (127.0.0.1) en el puerto `IPC_PORT` (en nuestro caso 5555) definido en el fichero `ipc_socket.h`. Para probar que el funcionamiento de dicha conexión es correcta se puede utilizar el comando `telnet` a través del terminal (provisto en la mayoría de sistemas operativos) de la siguiente manera:

```
telnet localhost 5555
```

Una vez conectado con el plugin SDM, se le pueden enviar comandos a través del terminal para anunciar, buscar y/o eliminar servicios. La estructura de dichos comandos es la siguiente:

```
<Comando>  <Servicio>  <Valor>
```

A continuación se enumerarán los diferentes comandos que se le pueden enviar al plugin SDM:

Advertisement (Anuncio de servicios)

Una aplicación puede registrar un servicio para que se anuncie a toda la red a través del comando `ADVR <Servicio> <Lifetime>`. Si por ejemplo una aplicación quiere anunciar una impresora y cree que su tiempo de vida es de 150 segundos (pudiendo actualizarlo cada intervalo de tiempo `SDM Interval`), el comando que enviará al plugin SDM será:

```
ADVR Printer 150
```

Si la aplicación no conoce el tiempo de vida `(Life Time)`, puede omitirse este parámetro dándole el plugin SDM un tiempo de vida por defecto de 255 segundos (máximo permitido).

Request (Búsqueda de servicios)

Una aplicación puede buscar un servicio a través del comando `RQST <Servicio>`. Si por ejemplo una aplicación quiere buscar una impresora en toda la red, el comando que enviará al plugin SDM será:

```
RQST Printer
```

Si mirando la caché de servicios el plugin SDM descubre que existe alguna impresora disponible en algún nodo de la red (por ejemplo en el nodo con dirección IP 192.168.0.50), le enviará el siguiente mensaje a la aplicación:

```
SERVICE FOUND: Printer AT 192.168.0.50
```

Si en cambio el servicio se encuentra en el propio nodo el mensaje que se envía a la aplicación es el siguiente:

```
SERVICE FOUND: Printer AT LOCALHOST
```

Withdrawn (Eliminación de servicios)

Una aplicación puede eliminar un servicio y anunciar al resto de la red que ya no está disponible a través del comando `WTDR <Servicio>`. Si por ejemplo una aplicación estaba anunciando una impresora pero ésta ya no se encuentra disponible y por tanto quiere eliminarla, el comando que enviará al plugin SDM será:

```
WTDR Printer
```

Look (Observación del contenido de la caché)

Una aplicación puede observar el contenido de cualquier caché del plugin SDM (local, del resto de nodos y de tiempos) a través del comando `LOOK <Caché>`. A continuación se pondrá un ejemplo de cada uno de los tres posibles comandos (uno por cada tipo de caché) y un posible resultado que el plugin SDM envía a la aplicación:

`LOOK Local` (Caché de servicios locales)

```
ServiceName     Description     LifeTime
===========     ===========     ========
Printer              0             255
```

`LOOK Foreign` (Caché de servicios anunciados por el resto de nodos)

```
ServiceName     IPAddress     Desc.     Life
===========     =========     =====     ====
Printer        192.168.0.50     0        255
```

`LOOK Time` (Caché de tiempos)

```
IPAddress       SDMInterval     LastUpdate
=========       ===========     ==========
192.168.0.50         5              2
```

Exit (Desconexión de la aplicación con el plugin SDM)

Una aplicación puede desconectarse del plugin SDM a través del comando `EXIT SDM`. Todas aquellos servicios anunciados únicamente por la aplicación que se desconecta serán eliminados de la caché, enviándose una notificación de cada servicio eliminado al resto de nodos de la red.

7.10.1. Registro de sockets

Los sockets que utilizan las aplicaciones para comunicarse con el plugin SDM tienen un número que los identifica. Cada servicio que está almacenado en la caché de servicios locales posee un array que contiene a los identificadores de aquellas aplicaciones que lo anuncian. Por lo tanto, cuando una aplicación anuncia un servicio, si éste no existe crea un nuevo elemento en la lista enlazada, pero si éste ya existe lo que hace es registrarse en él almacenando su identificador en el array correspondiente. Así, un servicio puede estar anunciado por una o más aplicaciones (ver Figura 7.7).

Cuando una aplicación desea eliminar un servicio, lo que hace es eliminar su identificador del array perteneciente a dicho servicio. Entonces el plugin SDM comprueba si hay alguna otra aplicación anunciándolo también. Si descubre que dicho servicio era únicamente anunciado por la aplicación que desea eliminarlo, el servicio será eliminado también de la caché y notificado mediante un mensaje SDM al resto de nodos.

Algo parecido ocurre cuando una aplicación se desconecta del plugin SDM. Entonces se elimina el identificador de la aplicación de todos los servicios anunciados por dicha aplicación, siendo eliminados de la caché de servicios locales y notificado mediante mensajes SDM al resto de nodos de la red aquellos servicios que eran anunciados únicamente por la aplicación (ver Figura 7.8).

7.11. Resumen

En este capítulo se ha explicado el diseño de la implementación del protocolo OLSR (demonio olsrd), su interfaz para cargar plugins (librerías cargadas dinámicamente) y el plugin que implementa el protocolo de descubrimiento de servicios SDM. Cabe destacar que tanto el demonio olsrd como su interfaz de plugins ha sido implementado por Andreas Tønnesen [25], mientras que el plugin SDM ha sido implementado por el autor de este proyecto. En primer lugar se ha descrito brevemente la arquitectura y las características principales del

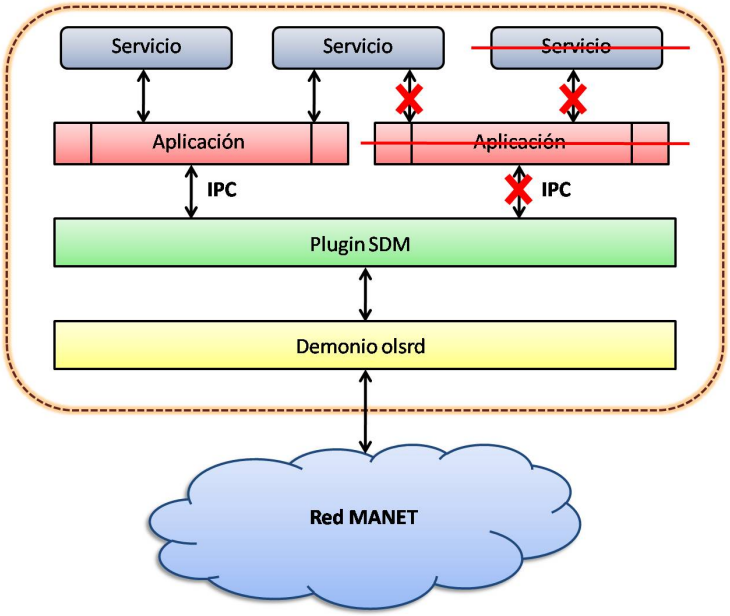

Figura 7.8: Eliminación de una aplicación y los servicios que anuncia

demonio olsrd, explicando luego cada uno de los módulos de los que está compuesto esta implementación. En segundo lugar se ha definido el funcionamiento de la interfaz de plugins, es decir, del módulo que permite extender el código de olsrd con diferentes funcionalidades (en este caso la integración de un protcolo de descubrimiento de servicios en el protocolo de routing OLSR) sin tener que modificar el núcleo del código original. Por último se enumeran las especificaciones de la implementación del protocolo de descubrimiento de servicios SDM como plugin del demonio olsrd. Después de una visión general de cómo está implementado, se describe su arquitectura, sus componentes y módulos más destacados, explicando más tarde el diagrama de flujo de la recepción de mensajes SDM. Una parte importante del capítulo es la manera en la que están implementados cada una de las cachés de servicios a través de listas dobles enlazadas, definiéndose para ello las estructuras utilizadas. Por último se explica cómo implementa el plugin cada una de las posibles funciones que puede realizar el protocolo SDM y la comunicación con los procesos que quieren anunciar y/o buscar servicios con el plugin SDM a través de una sencilla interfaz.

Capítulo 8

Medidas experimentales del plugin SDM en escenarios reales

En este capítulo se presentarán las diferentes técnicas que existen para comprobar que el protocolo SDM cumple con los requisitos especificados. Después de razonar la elección de una de estas técnicas, se explicará brevemente todos los escenarios de pruebas empleados en el análisis del plugin SDM, los resultados obtenidos y las conclusiones a las que se llegan a partir de dichos resultados.

8.1. Introducción

Después de diseñar e implementar cualquier protocolo de red hay que realizar un análisis exhaustivo de su comportamiento para determinar si cumple con las expectativas y requisitos especificados inicialmente. Para realizar dicho análisis se pueden utilizar varias técnicas según el nivel de abstracción empleado (lo que da lugar a una mayor flexibilidad a cambio de parecerse menos a la realidad). Entre las técnicas más conocidas se encuentran:

- **Modelo analítico:** El modelo analítico es un modelo matemático que intenta a través de ecuaciones y fórmulas describir el comportamiento de un sistema complejo. Por ejemplo el funcionamiento electrónico de un diodo viene determinado por la ecuación de Shockley:

$$I = I_s(e^{\frac{qV}{nkT}} - 1)$$

  En el estudio de un protocolo se suelen utilizar modelos estadísticos ya

que no se sabe con precisión cómo van a actuar los nodos de la red en un momento determinado, de ahí la utilización de probabilidades (en nuestro caso la probabilidad de que un nodo anuncie un servicio en un instante determinado).

Los modelos matemáticos son bastante precisos y no requieren de muchos recursos, aunque a menudo se realizan ciertas simplificaciones que hacen que el modelo difiera de la realidad (por ejemplo, es bastante habitual en los modelos matemáticos de redes de comunicaciones suponer que no hay degradación de la señal en la transmisión, algo que obviamente no es cierto). Además, si el sistema a estudiar es muy complejo puede que el modelo matemático sea tan complicado que muy pocas personas sean capaces de entenderlo en profundidad.

- **Simulación:** Según R. Shannon [59] la simulación es "el proceso de diseñar un modelo de un sistema real y llevar a término experiencias con él, con la finalidad de comprender el comportamiento del sistema o evaluar nuevas estrategias - dentro de los limites impuestos por un cierto criterio o un conjunto de ellos - para el funcionamiento del sistema". Estos modelos son obtenidos a partir de aproximaciones y simplificaciones del sistema real para poder realizar una evaluación factible en un ordenador (y que no requiera de una cantidad de tiempo inviable para el experimento).

  Los simuladores son ampliamente utilizados (casi imprescindibles) en la evaluación del comportamiento de protocolos en redes de comunicaciones, siendo los más famosos ns-2 [38] y OMNeT++ [60]. Esto es debido a que se obtiene una idea bastante buena del comportamiento del protocolo sin tener que emplear muchos recursos en experimentos reales, además puede ser repetido y verificado por cualquier persona y escalable hasta cientos o miles de nodos en la red (algo impensable en un escenario de pruebas real).

- **Experimento en el mundo real:** Por último nos encontramos con la técnica de evaluación que más se asemeja al uso habitual que se le dará al sistema en entornos reales. En este tipo de técnica, todos los componentes (tanto de hardware como de software) son completamente funcionales y aptos para su uso en el mundo real, aunque se diferencia de éste en que el entorno de los experimentos está controlado, teniendo que determinar por ejemplo el número de nodos, el valor de los parámetros más importantes del protocolo, etc.

  El test definitivo consistiría en probar todas las configuraciones y escenarios reales posibles, algo imposible de realizar debido a que los recursos

(de tiempo, dinero y humanos) son limitados. Además los resultados de los experimentos son difíciles de repetir y por lo tanto de verificar por terceras personas.

En la mayoría de protocolos que se diseñan, para verificar que cumplen con todos los requisitos, se suele utilizar primero la simulación y, si ésta es satisfactoria, realizar una implementación en un entorno real y probarlo en situaciones determinadas.

Así ha ocurrido con el protocolo de descubrimiento de servicios SDM. En [52] se realizan pruebas con el simulador ns-2 para comprobar que todo funciona correctamente, mientras que el objetivo de este proyecto es implementar dicho protocolo en un dispositivo real (Nokia N800) y proponer una serie de escenarios posibles para medir su rendimiento con aplicaciones reales.

8.2. Características de los escenarios

Los escenarios utilizados para probar las características y medir la eficiencia del protocolo de descubrimiento de servicios SDM están compuestos entre 1 y 4 dispositivos Nokia N800 y N810 (basados en la plataforma Maemo) con sistema operativo OS2008 Diablo. Después de crear el paquete Debian (.deb) correspondiente al demonio olsrd junto con el plugin SDM e instalarlo a través del gestor de aplicaciones, serán necesarias la utilización de varias herramientas para medir tanto el nivel de batería de los dispositivos como el ancho de banda utilizado, que son:

- **Advanced-power:** Advanced-power [61] es una aplicación Maemo diseñada para mostrar el nivel de batería con mayor detalle que la que muestra el sistema operativo por defecto. Además del nivel de batería también muestra información sobre la temperatura, memoria RAM utilizada, nivel de iluminación de la pantalla, etc.

- **tcpdump:** Herramienta cuyo cometido es principalmente analizar el tráfico que circula por la red, capturando todos los paquetes transmitidos y recibidos por la interfaz correspondiente y mostrando información de dichos paquetes por pantalla (número de paquetes recibidos, dirección IP fuente y destino, puerto origen y destino, *timestamp*, etc.).

- **tcpstat:** Herramienta que se utiliza para obtener datos estadísticos del tráfico que circula por la red a partir de un fichero creado por *tcpdump*.

Entre la información estadística que se puede obtener se encuentra el tamaño medio de los paquetes, la desviación estándar del tamaño de cada paquete, el ancho de banda utilizado (en bits por segundo), etc.

A continuación se muestran los resultados de las medidas experimentales tomadas en los escenarios anteriormente descritos y las conclusiones obtenidas. Las pruebas que se han realizado para observar el correcto funcionamiento y medir el rendimiento del protcolo de descubrimiento de servicios SDM son las siguientes:

- **Consumo de energía del protocolo OLSR:** Medición del consumo energético que produce la ejecución del protocolo OLSR junto con el protocolo SDM en los dispositivos móviles.
- **Cambio de ruta según nivel de *willingness*:** Comprobación de que la elección de una ruta depende del nivel de voluntad del dispositivo que actúa como router para ser elegido como nodo MPR (*willingness*).
- **Ancho de banda consumido:** Cálculo del ancho de banda consumido por el protocolo OLSR junto con el protocolo SDM en función del número de servicios anunciados y del intervalo de tiempo entre mensajes SDM para diferentes configuraciones de red (2, 3 y 4 nodos en línea).
- **Retardo del protocolo SDM:** Cálculo experimental del retardo existente en el protocolo de descubrimiento de servicios SDM tanto en el anuncio de servicios como en la notificación que se envía al resto de la red cuando se elimina algún servicio.
- **Comparativa con el protocolo Mercury:** Comparación de rendimiento del protocolo SDM con el protocolo de descubrimiento de servicios Mercury [53], que al igual que SDM se integra dentro del protocolo de routing OLSR.

8.3. Consumo de energía del protocolo OLSR

En esta prueba se medirá el consumo energético que produce la ejecución del protocolo OLSR junto con el protocolo SDM en los dispositivos móviles, así como la comparación del gasto energético entre un nodo que no es elegido MPR con otro que sí lo es (teniendo en este caso que reenviar los mensajes que reciba al resto de la red). El escenario de pruebas está compuesto por 4

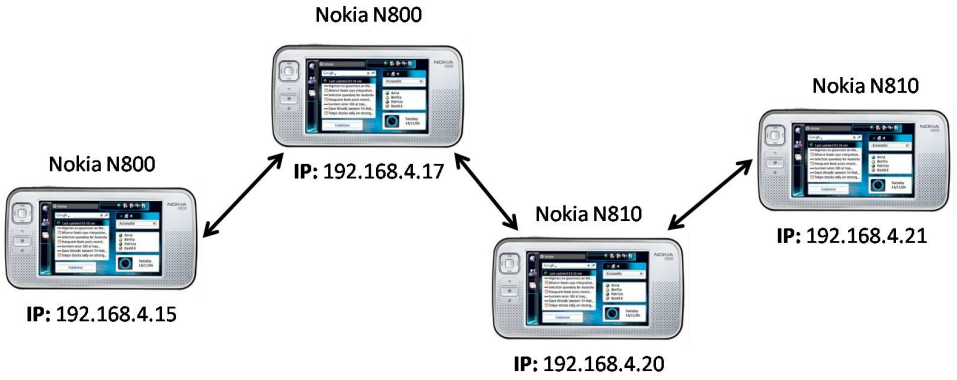

Figura 8.1: Escenario de pruebas con cuatro nodos en línea

dispositivos Nokia N800 y N810 colocados en línea como se puede observar en la Figura 8.1.

Después de comprobar que cada dispositivo está cargado totalmente, se procede a la ejecución del demonio olsrd en cada uno de los dispositivos, anunciando también cada uno de ellos 4 servicios a través del plugin SDM. La prueba consiste en ir midiendo el nivel de batería de cada dispositivo cada 10 minutos hasta un máximo de 1 hora para ver cómo progresa el consumo de energía a lo largo del tiempo. En principio debería ocurrir que los nodos que han sido elegido como MPR (en nuestro caso los dos centrales) consumen más batería que aquellos que no lo son ya que deben reenviar los mensajes tanto del protocolo OLSR (mensajes `TC`) como del protocolo SDM. Para la realización de esta prueba suponemos que los dispositivos del mismo modelo (N800 y N810) consumen la misma batería si se ejecutan las mismas aplicaciones.

Como se puede comprobar en el Cuadro 8.1, en 1 hora se gasta entre un 16 % y un 18 % de batería ejecutando tanto el demonio olsrd como el plugin SDM. Además para averiguar lo que gasta solamente esta aplicación se ha medido el consumo de batería de los dispositivos móviles en reposo con todas las aplicaciones que vienen por defecto activadas, como por ejemplo el módulo Wifi, la herramienta Advanced Power para medir el consumo de batería e incluso los terminales donde serían ejecutados las aplicaciones (para intentar consumir la misma energía excepto la gastada expresamente por el demonio olsrd y el plugin SDM).

Observando las diferencias entre los niveles de batería medidos, se llega a la conclusión de que el gasto energético producido solamente por el demonio olsrd junto con el plugin SDM anunciando 4 servicios es aproximadamente de un 1 %, algo muy poco significativo para el nivel total de batería del dispositivo

	192.168.4.15 (Nokia N800)	192.168.4.17 (Nokia N800)	192.168.4.20 (Nokia N810)	192.168.4.21 (Nokia N810)
Inicio	90.70%	90.60%	94.90%	92.90%
10 min	87.50%	87.20%	91.70%	89.80%
20 min	84.60%	84.20%	89.10%	87.30%
30 min	81.70%	81.20%	86.40%	84.90%
40 min	78.70%	78.20%	83.70%	82.30%
50 min	75.70%	75.10%	81.10%	79.80%
60 min	72.70%	72.10%	78.40%	77.40%

Cuadro 8.1: Nivel de batería de cada dispositivo

	192.168.4.15 (Nokia N800)	192.168.4.17 (Nokia N800)	192.168.4.20 (Nokia N810)	192.168.4.21 (Nokia N810)
10 min	3.20%	3.40%	3.20%	3.10%
20 min	6.10%	6.40%	5.80%	5.60%
30 min	9.00%	9.40%	8.50%	8.00%
40 min	12.00%	12.40%	11.20%	10.60%
50 min	15.00%	15.50%	13.80%	13.10%
60 min	18.00%	18.50%	16.50%	15.50%

Cuadro 8.2: Porcentaje de batería consumido por cada dispositivo

Nokia N800 y N810.

El hecho de que el nivel de la batería medido con la herramienta Advanced Power no fuera del 100 % cuando la batería estaba totalmente cargada (como se muestra en el Cuadro 8.1 en el inicio de la prueba) fue bastante extraña ya que no se realizó ninguna acción anterior que pudiera consumir batería. Probablemente la medición de la batería con esta herramienta sea correcta aunque a lo mejor algo aproximada. Cabe destacar que para la realización de esta prueba se intentó utilizar una aplicación más fiable denominada Nokia Energy Profiler [70], pero la falta de una versión compatible con la plataforma Maemo hizo imposible su utilización para medir el consumo de batería.

Mirando los resultados en el Cuadro 8.2, se observa que el modelo N800 consume más batería que el modelo N810 ejecutando las mismas aplicaciones (alrededor de un 2 % más en 1 hora). Esto puede ser debido a que al empezar cada modelo desde diferentes niveles de batería (en los N800 empiezan alrededor del 90 % y los N810 alrededor del 94 %) su consumo energético no sea el mismo, ya que se ha comprobado en [71] que el gasto de energía en las baterías de los móviles no es lineal, sino que se asemeja más a una función sigmoide (ver Figura 8.2).

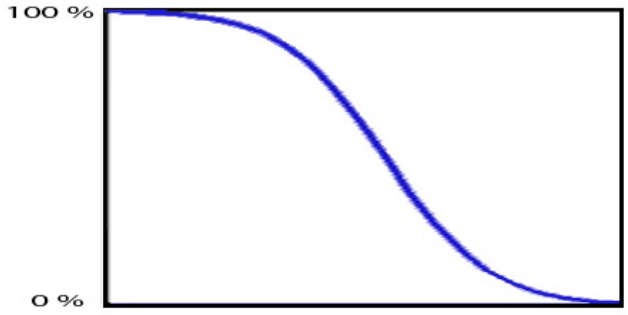

Figura 8.2: Consumición de las baterías en los móviles (función sigmoide)

Esto significa que cuanto más cargada está la batería más lentamente desciende su nivel (hasta aproximadamente el 50 %, donde a partir de ese momento ocurre justamente lo contrario). Así, en los N810, al estar sus baterías más cargadas que las de los N800, su consumo de batería es algo menor.

Otro aspecto a destacar es la diferencia entre los modelos que no han sido elegidos nodos MPR con los que sí lo han hecho. Al ser la consumición de energía diferente para el modelo N800 y N810, las comparaciones las vamos a realizar entre dispositivos del mismo modelo (como se observa en la Figura 8.1, hay un nodo MPR y otro que no lo es por cada modelo). Tal y como se esperaba, los nodos que han sido elegidos como nodos MPR consumen más batería que los que no han sido elegidos (un 0.5 % más en el caso del modelo N800 y un 1 % más para el modelo N810).

Debido a que el consumo extra de batería por ser nodo MPR es muy pequeño podemos llegar a la conclusión de que la elección de un nodo como MPR no afecta de manera significativa en el consumo de energía de dicho dispositivo. Aun así, como se verá en la sección siguiente, sí es importante elegir a un nodo como MPR o no según el nivel de batería que tenga dicho dispositivo.

8.4. Cambio de ruta según nivel de *willingness*

En esta prueba se comprobará que la elección de una ruta depende del nivel de voluntad del dispositivo que actúa como router para ser elegido como nodo MPR (*willingness*). Para ello se ha establecido un escenario de pruebas con cuatro nodos en forma de rombo como se puede observar en la Figura 8.3. De esta manera los paquetes que van del dispositivo situado más a la izquierda al dispositivo situado más a la derecha (nodos externos) tendrán que elegir a uno de los dos nodos centrales como nodo MPR para que reenvíe dichos paquetes.

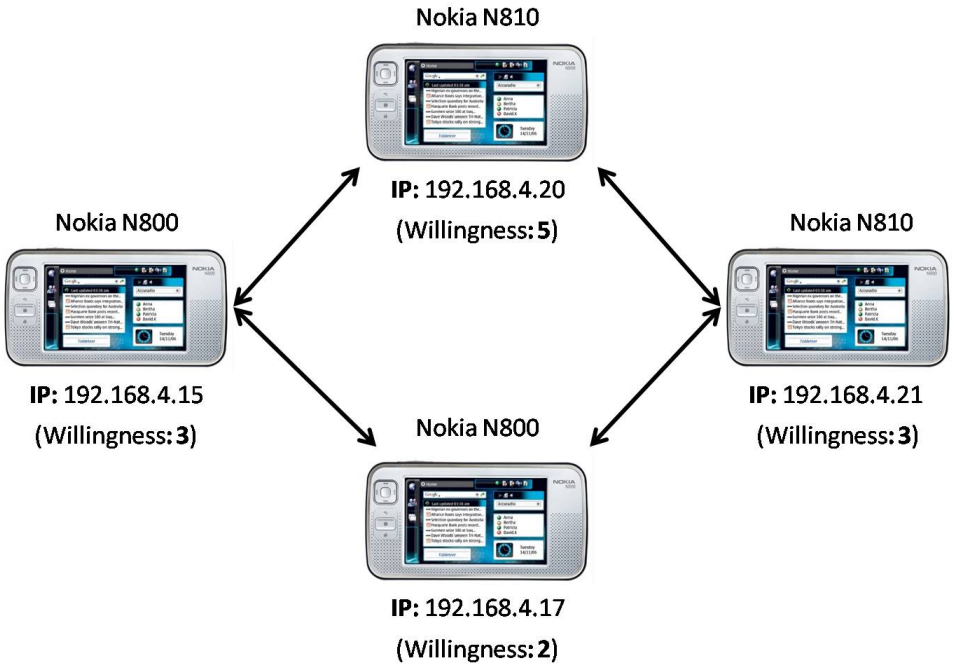

Figura 8.3: Escenario de pruebas con cuatro nodos en forma de rombo

Según el RFC del protocolo OLSR [1], cada dispositivo tiene un nivel de voluntad (*willingness*) para que sea elegido como nodo MPR determinado principalmente por el nivel de batería que le queda a cada dispositivo. Si un dispositivo puede elegir entre dos nodos para que le reenvíen los paquetes al resto de la red, en teoría elegirá aquél con mayor nivel de *willingness* ya que, suponiendo que ambos nodos permanecen estáticos, hay más posibilidades de que dicho dispositivo permanezca conectado a la red (sólo se desconectaría de ella si se le acabara la batería). Por lo tanto, en el ejemplo de la Figura 8.3, al tener el nodo central superior un mayor nivel de *willingness*, los nodos externos deberían elegirle como nodo MPR para que le reenviaran los paquetes.

Las pruebas que se realizarán para comprobar experimentalmente este aspecto del protocolo OLSR son las siguientes:

1. Una vez creada la red y pasado algún tiempo para que se estabilicen las rutas, comprobar que el nodo central que ha sido elegido MPR es el que tiene mayor nivel de voluntad (*willingness*).

2. Si se desconecta de la red el nodo que tiene menor nivel de *willingness*, comprobar que este hecho no afecta a la ruta ya establecida y por lo tanto no se produce un retardo en la red para encontrar una nueva ruta y que ésta se estabilice. En este caso se calculará el tiempo que tarda la red en darse cuenta de que el nodo se ha desconectado y por lo tanto en borrarlo de sus tablas de routing.

3. Si se desconecta de la red el nodo que tiene mayor nivel de *willingness*, comprobar que este hecho sí afecta a la ruta ya establecida, teniendo que buscar una ruta alternativa que provocará un retardo en la transmisión de los paquetes hasta que la topología de la red almacenada en cada uno de los dispositivos se estabilice a través de los mensajes OLSR. En este caso se calculará tanto el tiempo que tarda la red en darse cuenta de que el nodo se ha desconectado como el tiempo que tardan los nodos externos en encontrar una ruta alternativa, eligiendo al otro nodo central que queda conectado en la red como nodo MPR.

Al realizar dichas pruebas, los resultados obtenidos han resultado ser totalmente diferentes a lo esperado. En primer lugar se comprobó que, a pesar de tener diferentes niveles de *willingness*, los nodos externos elegían indistintamente a uno u otro nodo central como nodo MPR para que reenviara los paquetes al resto de la red. Revisando las especificaciones de diseño del demonio olsrd nos dimos cuenta que para la elección de un nodo MPR, dicha implementación del protocolo OLSR no se basa en el nivel de *willingness*, sino en otra característica denominada ETX (*Expected Transmission Count*).

El parámetro ETX, descrita por Douglas S.J. De Couto en su tesis doctoral [62], se define como el número esperado de transmisiones (y retransmisiones) requeridas para entregar con éxito un paquete a lo largo de un enlace. Este parámetro sirve para medir la calidad del enlace en ambas direcciones de transmisión en redes wireless multi-salto y así encontrar rutas de alta capacidad. El rango de este parámetro se encuentra entre 1 e infinito, siendo $ETX = 1$ una transmisión perfecta y $ETX = \infty$ una transmisión nula. Por lo tanto aquellos enlaces que tengan el mínimo valor de ETX son aquellos que poseen la mayor capacidad de transmisión.

Así, en la implementación realizada por Andreas Tønnesen del protocolo OLSR [25], la elección de una ruta u otra se basa en el parámetro ETX (calidad del enlace) y no en el nivel de *willingness*. A continuación se muestra la tabla de routing del nodo situado a la izquierda en la red de la Figura 8.3 con dirección IP 192.168.4.15.

```
Table: Routes

Destination      Gateway IP      Metric      ETX      Interface
===========      ==========      ======      ===      =========
192.168.4.21     192.168.4.17         2      2.048        wlan0
192.168.4.20     192.168.4.20         1      1.024        wlan0
192.168.4.17     192.168.4.17         1      1.024        wlan0
```

Como se puede observar, en la prueba realizada el valor del parámetro ETX es el mismo para ambos nodos centrales, por lo tanto el demonio olsrd elige indistintamente a uno u otro nodo como MPR para reenviar los paquetes, sin tener en cuenta el nivel de *willingness*.

Cuando se desconecta de la red el nodo que no ha sido elegido como nodo MPR, la ruta que existe entre los nodos externos no se ve afectada tal y como se esperaba, tardando una media de 6.4 segundos en darse cuenta la red que el nodo se ha desconectado y por tanto de borrar la información de dicho nodo de las tablas de routing del resto de nodos de la red.

Cuando se desconecta de la red el nodo que sí ha sido elegido como nodo MPR, la ruta que existe entre los nodos externos deja de existir y a través de los mensajes OLSR se tiene que buscar una ruta alternativa para llegar al otro lado de la red. En principio se esperaba que el tiempo que tarda la red en detectar que el nodo se ha desconectado fuera menor que el tiempo que tarda en encontrar la ruta alternativa ya que empezaría a buscar dicha ruta cuando se diese cuenta que el nodo dejaba de estar presente en la red. Pero las pruebas demuestran que, mientras el tiempo que tarda en darse cuenta la red que el nodo se ha desconectado es, al igual que antes, de 6.4 segundos, el tiempo que tarda la red en encontrar una ruta alternativa es solamente de 4.2 segundos.

Esto se debe a que la red no empieza a buscar una ruta alternativa cuando detecta que el nodo se ha desconectado, sino que se guía una vez más por el parámetro ETX. Nada más desconectarse el nodo, al no recibirse paquetes de éste, el parámetro ETX de los enlaces que utilizan dicho nodo empieza a aumentar su valor, y automáticamente al detectarse que el valor de ETX es mayor que el del otro nodo central, se elige como nodo MPR al que sigue conectado y

se cambia la ruta establecida para que los nodos externos se comuniquen entre sí. De esta manera se acelera el proceso de búsqueda de una ruta alternativa cuando se desconecta el nodo que había sido elegido como MPR.

La conclusión que se obtiene de esta prueba es que el demonio olsrd sigue un criterio diferente al especificado en el RFC del protocolo OLSR para la elección del nodo MPR. En lugar de tener en cuenta el nivel de *willingness*, elige las mejores rutas a través del parámetro ETX que mide la calidad de los enlaces entre los nodos. Personalmente creo que la elección de rutas a través del parámetro ETX es una excelente idea, ya que es posible que una ruta sea más rápida aunque tenga que dar más saltos poque sus enlaces son de muchísima mayor capacidad que otra ruta en la que sólo hay que dar un salto entre los dos nodos pero dicho enlace es de pésima calidad.

Aún así, creo que en la elección de las rutas habría que tener en cuenta también el nivel de *willingness* además del parámetro ETX, ya que si un dispositivo tiene poco nivel de *willingness* probablemente sea porque tiene menos batería y por tanto mayores posibilidades de que se le acabe dicha batería y se desconecte de la red. Si esto ocurriera se perdería por un tiempo la disponibilidad de los servicios del protocolo SDM hasta que se encontrara una nueva ruta, pudiendo darse el caso de falsos positivos. En conclusión, creo que habría que hacer una ponderación entre el nivel de *willingness* y el valor del parámetro ETX para llegar a encontrar las mejores rutas posibles dentro de la red MANET.

8.5. Cálculo teórico del ancho de banda

En este apartado se va a calcular el ancho de banda consumido teóricamente por el protocolo SDM para un número de nodos N, los cuales anuncian un número de servicios determinado en un intervalo de tiempo dado (`SDM Interval`). De esta manera se puede realizar una comparación con las medidas experimentales que aparecen en el apartado 8.6 para comprobar si éstas tienen sentido y se aproximan al valor teórico. Cabe destacar que en el ancho de banda calculado experimentalmente se encuentran tanto paquetes del protcolo OLSR como del protocolo SDM, por lo que también se calcula teóricamente el ancho de banda consumido por el protocolo OLSR para un número de nodos N. Además hay que tener en cuenta en el cálculo del ancho de banda teórico las cabeceras de los niveles de enlace (Ethernet), red (IP) y transporte (UDP), presente en todos los paquetes OLSR y SDM que circulan por la red MANET. Estas cabeceras siempre tienen un tamaño constante, mostrándose sus valores

en la Figura 8.4, sumando todas ellas 42 bytes.

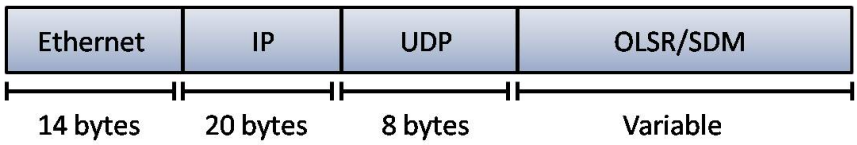

Figura 8.4: Tamaño de las cabeceras del paquete OLSR

Para calcular teóricamente el ancho de banda consumido por el protocolo de routing OLSR y el protocolo de descubrimiento de servicios SDM en la red MANET, suponemos que todos los nodos están en línea, es decir, su rango de transmisión solamente alcanza a los dispositivos contiguos, como se observa en la Figura 8.5.

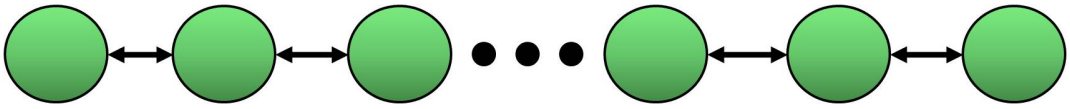

Figura 8.5: Configuración de una red MANET en línea

En primer lugar se calculará el ancho de banda que consume cada nodo individualmente (sin retransmisiones) en función del número de servicios anunciados y el valor del intervalo de tiempo `SDM Interval`. Después se calculará el ancho de banda consumido para una configuración de red de 5 nodos (en la que los nodos centrales retransmiten los mensajes al resto de la red). Por último, una vez visto cómo se calcula para un ejemplo concreto, se generalizará para N nodos, obteniendo una fórmula general del ancho de banda consumido teóricamente por el protocolo de descubrimientos de servicios SDM para una cantidad de nodos N que anuncian un número específico de servicios en un intervalo de tiempo dado (suponiendo que todos los nodos anuncian el mismo número de servicios en el mismo intervalo de tiempo).

8.5.1. Ancho de banda consumido por 1 nodo

Tal y como se vio en el apartado 6.2.1, el tamaño de un mensaje SDM viene dado por:

$$Cabecera\ OLSR + Mensaje\ SDM * Numero\ servicios = 16 + 8 * S\ bytes$$

Al ser un solo nodo el que compone la red MANET, los mensajes `HELLO` que envía dicho nodo no contienen a ningún vecino, y por tanto en este caso

no se produce el envío de mensajes TC. Tal y como se vio en el apartado 3.3.1, el tamaño del mensaje `HELLO` viene dado por:

$$Cabecera\ (OLSR + HELLO) + 4 * Numero\ vecinos = (16 + 8) + 4 * V\ bytes$$

Si se envía un mensaje SDM cada `SDM Interval`, el ancho de banda consumido por 1 nodo individualmente será igual a (en bits por segundo):

$$Bandwidth = \frac{8 * (42 + 20)}{HELLO\ Interval} + \frac{8 * (42 + 16 + 8 * Servicios)}{SDM\ Interval}\ bps$$

Como se puede observar, a cada uno de los paquetes OLSR hay que sumarle los 42 bytes correspondientes a las cabeceras Ethernet, IP y UDP. Cabe destacar que aunque generalmente el tamaño de la cabecera de `OLSR + HELLO` son 24 bytes, en el caso de 1 nodo es solamente 20 bytes, ya que se ahorra los 4 bytes que contienen la información del tipo de enlace que posee con sus vecinos (en nuestro caso simétrico) al no tener vecinos que anunciar (ver apartado 3.3.1).

8.5.2. Ancho de banda consumido por 5 nodos

Una vez que sabemos cuánto consume cada nodo individualmente, tenemos que calcular el consumo del ancho de banda cuando existen varios nodos en la red, ya que se producen retransmisiones que aumentan el ancho de banda. Así, dependiendo de la posición del nodo en la red, el número de mensajes OLSR y SDM enviados varía en función de las retransmisiones que debe realizar para que todos los mensajes SDM de todos los nodos lleguen a toda la red MANET. Cómo la configuración de una red en línea es simétrica, solamente se necesitará calcular el ancho de banda en la mitad de las posiciones pues el resto consume lo mismo. Además, al existir más de un nodo en la red, se enviarán (y retransmitirán) mensajes `TC` para que los nodos de la red MANET conozcan la topología de la red. Tal y como se vio en el apartado 3.3.2, el tamaño del mensaje `TC` viene dado por:

$$Cabecera\ (OLSR + TC) + 4 * Numero\ vecinos = (16 + 4) + 4 * V\ bytes$$

Nodo externo:

Cuando un nodo externo envía un mensaje OLSR o SDM, dicho mensaje deberá ser reenviado por todos los nodos que le siguen hasta llegar al otro extremo (ver Figura 8.6).

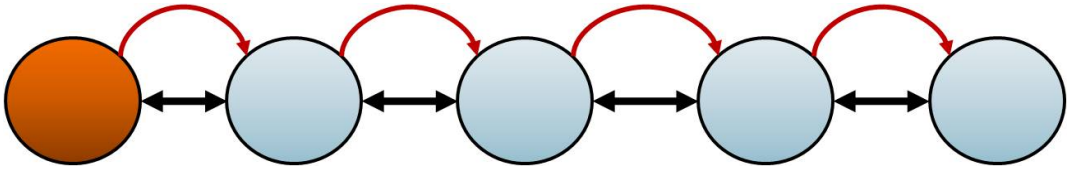

Figura 8.6: Mensajes entre un nodo externo y el resto de la red

Por lo tanto el ancho de banda consumido por los mensajes OLSR y SDM que envía dicho nodo es igual a:

$$
\begin{aligned}
= & \quad \frac{8 * (42 + 24 + 4 * (1\ Vecino)) * 4(Nodos)}{HELLO\ Interval} + \\
+ & \quad \frac{8 * (42 + 20 + 4 * (1\ Vecino)) * 4(Nodos)}{TC\ Interval} + \\
+ & \quad \frac{8 * (42 + 16 + 8 * Servicios) * 4(Nodos)}{SDM\ Interval}\ bps
\end{aligned}
$$

Nodo interno:

Cuando un nodo interno envía un mensaje OLSR o SDM, dicho mensaje llega a dos nodos de un solo salto (el nodo externo y el nodo central), teniendo que ser reenviado por dichos nodos para poder llegar al otro extremo de la red (ver Figura 8.7). Por lo tanto tiene 2 vecinos que serán anunciados tanto en los mensajes `HELLO` como en los mensajes `TC`.

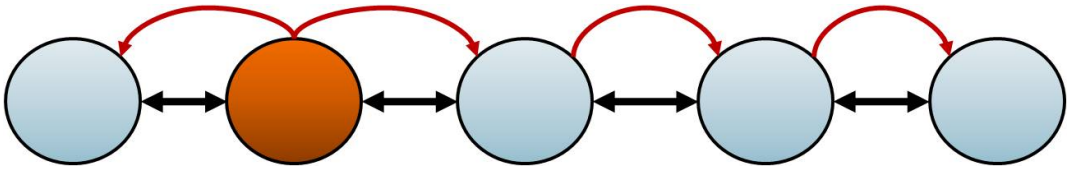

Figura 8.7: Mensajes entre un nodo interno y el resto de la red

Por lo tanto el ancho de banda consumido por los mensajes OLSR y SDM que envía dicho nodo es igual a:

$$Bandwidth = \frac{8 * (42 + 24 + 4 * (2\ Vecinos)) * 3(Nodos)}{HELLO\ Interval} + \\ + \frac{8 * (42 + 20 + 4 * (2\ Vecinos)) * 3(Nodos)}{TC\ Interval} + \\ + \frac{8 * (42 + 16 + 8 * Servicios) * 3(Nodos)}{SDM\ Interval}\ bps$$

Nodo central:

Cuando el nodo central envía un mensaje OLSR o SDM, dicho mensaje llega a los dos nodos internos de un solo salto, teniendo que ser reenviado por dichos nodos a los extremos de la red (ver Figura 8.8). Al igual que en el caso anterior, tiene 2 vecinos que serán anunciados tanto en los mensajes `HELLO` como en los mensajes `TC`.

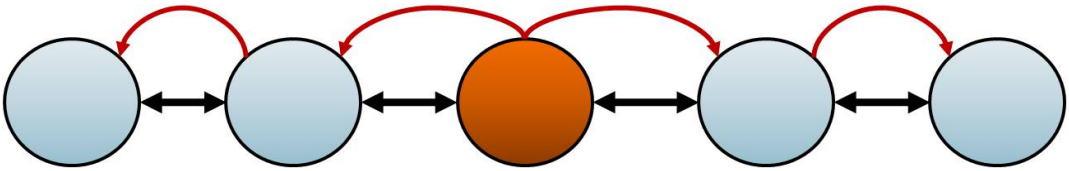

Figura 8.8: Mensajes entre el nodo central y el resto de la red

Por lo tanto el ancho de banda consumido por los mensajes OLSR y SDM que envía dicho nodo es igual a:

$$Bandwidth = \frac{8 * (42 + 24 + 4 * (2\ Vecinos)) * 3(Nodos)}{HELLO\ Interval} + \\ + \frac{8 * (42 + 20 + 4 * (2\ Vecinos)) * 3(Nodos)}{TC\ Interval} + \\ + \frac{8 * (42 + 16 + 8 * Servicios) * 3(Nodos)}{SDM\ Interval}\ bps$$

Como se puede observar, el ancho de banda consumido por el nodo central es el mismo que el consumido por los nodos internos.

De esta manera, el consumo total de ancho de banda teórico para una red de 5 nodos en línea es la media de los anchos de banda consumidos individualmente por cada nodo, cuya expresión final es:

$$BW = \frac{3 * Bandwidth(Nodo\ interno) + 2 * Bandwidth(Nodo\ externo)}{5\ (Nodos)}\ bps$$

8.5.3. Ancho de banda consumido por N nodos

Una vez que se ha visto el cálculo del consumo de ancho de banda teórico para un ejemplo concreto (5 nodos en línea), se generalizará para N nodos. En el apartado anterior se comprobó que, según la posición del nodo dentro de la red, el consumo de ancho de banda es diferente. Los nodos externos sólo tienen a un dispositivo contiguo, por lo que necesitan que su mensaje OLSR y SDM sea reenviado a todos los demás nodos de la red excepto al que llega de un solo salto, y por tanto su mensaje OLSR o SDM será enviado (y retransmitido) $N-1$ veces, siendo N el número de nodos en línea. El ancho de banda consumido por los nodos externos viene dado por:

$$\begin{aligned} Bandwidth &= \frac{8 * (42 + 24 + 4 * (1\ Vecino)) * (N-1)}{HELLO\ Interval} + \\ &+ \frac{8 * (42 + 20 + 4 * (1\ Vecino)) * (N-1)}{TC\ Interval} + \\ &+ \frac{8 * (42 + 16 + 8 * Servicios) * (N-1)}{SDM\ Interval}\ bps \end{aligned}$$

En cambio los nodos internos tienen a dos dispositivos contiguos, por lo que necesitan que su mensaje sea reenviado al resto de nodos de la red excepto aquellos a los que llega de un solo salto, y por tanto su mensaje OLSR o SDM tiene que ser enviado (y retransmitido) $N-2$ veces, con N el número de nodos de la red. El ancho de banda consumido por lo nodos internos viene dado por:

$$\begin{aligned} Bandwidth &= \frac{8 * (42 + 24 + 4 * (2\ Vecinos)) * (N-2)}{HELLO\ Interval} + \\ &+ \frac{8 * (42 + 20 + 4 * (2\ Vecinos)) * (N-2)}{TC\ Interval} + \\ &+ \frac{8 * (42 + 16 + 8 * Servicios) * (N-2)}{SDM\ Interval}\ bps \end{aligned}$$

Calculando la media se obtiene el ancho de banda consumido teóricamente por el protocolo de routing OLSR más el protocolo de descubrimiento de servicios SDM para una configuración de N nodos en línea.

$$BW = \frac{(N-2) * Bandwidth(Interno) + 2 * Bandwidth(Externo)}{N\ (Nodos)}\ bps$$

A continuación se muestran las tablas de los valores teóricos del ancho de banda consumido para una configuación de 1, 2, 3 y 4 nodos calculados con las fórmulas obtenidas anteriormente. En cada una de estas configuraciones se ha calculado el ancho de banda en función de, o bien el número de servicios (manteniendo el valor de `SDM Interval` constante e igual a 20 segundos) o bien el intervalo de tiempo entre mensajes SDM (manteniendo el número de servicios constante e igual a 5). Los valores de los intervalos entre mensajes `HELLO` y `TC` son los dados por defecto en el RFC del protocolo OLSR [1] (2 y 5 segundos respectivamente).

La tabla con los valores teóricos del ancho de banda consumido en función del número de servicios se puede observar en el Cuadro 8.3.

	1 servicio	5 servicios	10 servicios	15 servicios	20 servicios
1 Nodo	274.40	287.20	303.20	319.20	335.20
2 Nodos	412.00	424.80	440.80	456.80	472.80
3 Nodos	500.00	521.33	548.00	574.67	601.33
4 Nodos	610.00	642.00	682.00	722.00	762.00

Cuadro 8.3: Ancho de banda teórico en función del número de servicios anunciados

La tabla con los valores teóricos del ancho de banda consumido en función del intervalo entre mensajes SDM (`SDM Interval`) se puede observar en el Cuadro 8.4.

La comparación entre los valores de ancho de banda teóricos y los obtenidos experimentalmente en el laboratorio con dispositivos reales se encuentra en el apartado 8.6.3.

	1 seg.	10 seg.	20 seg.	30 seg.	40 seg.
1 Nodo	1032.00	326.40	287.20	274.13	267.60
2 Nodos	1169.60	464.00	424.80	411.73	405.20
3 Nodos	1762.67	586.67	521.33	499.56	488.67
4 Nodos	2504.00	740.00	642.00	609.33	593.00

Cuadro 8.4: Ancho de banda teórico en función del valor de SDM Interval

8.6. Ancho de banda consumido

En esta prueba se calculará experimentalmente el ancho de banda consumido por el protocolo SDM integrado en el protocolo de routing OLSR en función de varios parámetros fundamentales (número de servicios anunciados, intervalo de tiempo entre mensajes SDM, etc.), observando así su incidencia en el consumo de ancho de banda medio por dispositivo (en bps). En principio los parámetros del protocolo OLSR que afectan al ancho de banda (como el intervalo entre mensajes `HELLO` y `TC`) siempre tendrán un valor constante, que será el asignado por defecto en el RFC del protocolo OLSR [1]. Por lo tanto, el intervalo entre mensajes `HELLO` será de 2 segundos, mientras que el intervalo entre mensajes `TC` será de 5 segundos.

Los parámetros fundamentales del protocolo de descubrimiento de servicios SDM que afectan significativamente al ancho de banda consumido por la red MANET en función de su valor son:

- **Número de servicios anunciados por cada dispositivo:** En teoría, a mayor número de servicios anunciados por cada dispositivo, mayor número de mensajes SDM y por lo tanto mayor ancho de banda consumido. En esta prueba se mantendrá un intervalo de tiempo entre mensajes SDM constante de 20 segundos y se calculará el ancho de banda para el caso en que los nodos anuncian 1, 5, 10, 15 y 20 servicios (todos los nodos de la red anuncian el mismo número de servicios).
- **Intervalo entre mensajes SDM enviados por el mismo nodo (`SDM Interval`):** En teoría, un intervalo de tiempo entre mensajes SDM mayor significa a la larga un menor número de mensajes enviados y por lo tanto un menor ancho de banda consumido. En esta prueba se mantendrá un

número de servicios anunciados por cada nodo constante e igual a 5, calculándose el ancho de banda para el caso en que el valor de `SDM Interval` sea igual a 1, 10, 20, 30 y 40 segundos.

Además se obtendrá el valor del ancho de banda consumido por el protocolo SDM integrado en el protocolo OLSR para configuraciones de red en la que se encuentran 1, 2, 3 y hasta 4 nodos en línea (ver Figura 8.9) para poder comparar cuanto aumenta el ancho de banda consumido en función del número de nodos (en teoría, a mayor número de nodos, mayor cantidad de retransmisiones y por lo tanto mayor ancho de banda consumido).

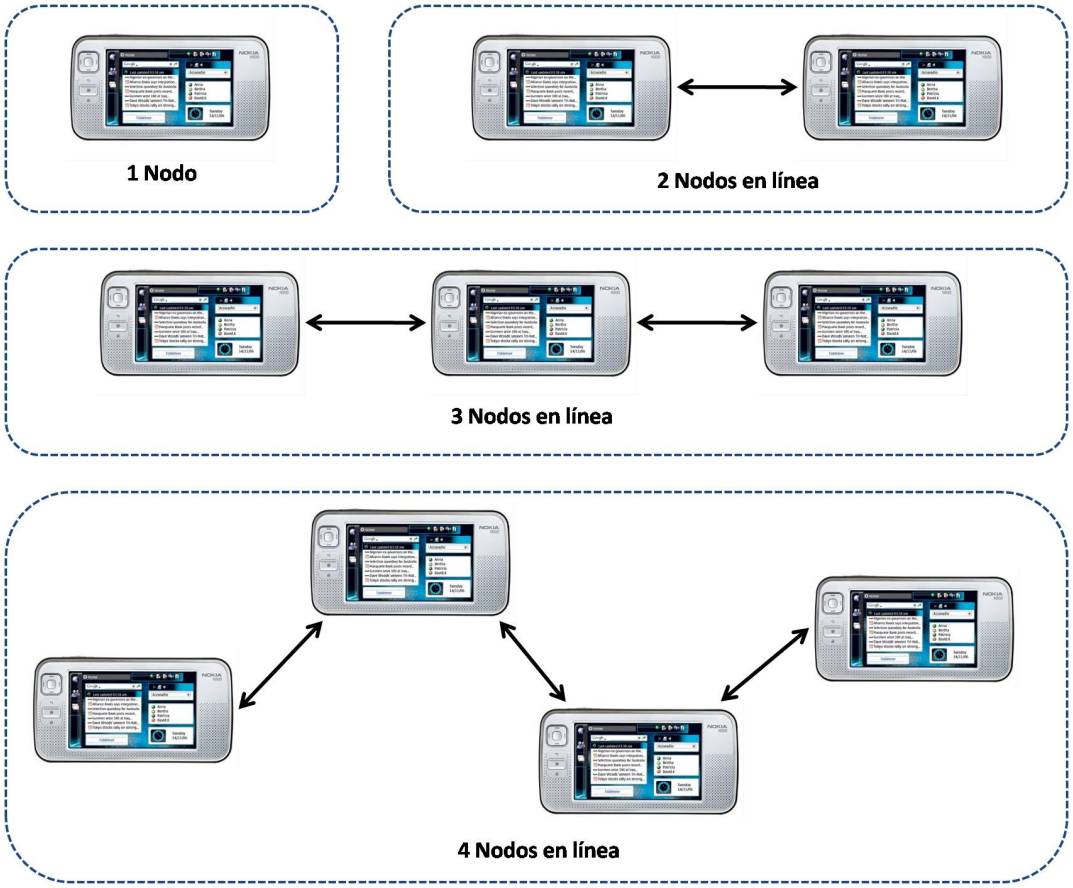

Figura 8.9: Configuraciones de red MANET utilizadas en la prueba

Como en las configuraciones de red donde hay 3 o más nodos en línea se producen retransmisiones en los nodos internos, el ancho de banda consumido por cada nodo de la red es diferente, por lo que para calcular el ancho de banda total de la red MANET se realiza la media. Además, se han realizado 3 tandas

de medidas para cada valor de los diferentes parámetros, siendo el valor final la media de estos 3 valores, obteniendo a su vez la desviación estándar y el intervalo de confianza al 95 % (aunque en este capítulo sólo se muestran los valores finales, todas las medidas obtenidas están recogidas en el anexo C para cada una de las pruebas realizadas). Aunque se ha obtenido el intervalo de confianza al 95 % por cada valor, en las gráficas que se observarán a continuación no aparecen ya que al ser tan pequeñas no se discernían claramente y se ha decidido no mostrarlas para no crear confusión.

8.6.1. Número de servicios anunciados por dispositivo

Los resultados experimentales obtenidos sobre el ancho de banda consumido (en bps) por cada una de las configuraciones de red en función del número de servicios anunciados por cada dispositivo cuando el intervalo entre mensajes SDM es constante se pueden observar en el Cuadro 8.5.

	1 servicio	5 servicios	10 servicios	15 servicios	20 servicios
1 Nodo	234.84	248.64	266.49	282.35	296.78
2 Nodos	355.66	372.11	387.38	404.48	417.92
3 Nodos	426.32	441.86	464.91	496.12	518.67
4 Nodos	499.53	530.14	574.17	609.78	650.89

Cuadro 8.5: Ancho de banda en función del número de servicios anunciados

Además en la Figura 8.10 se encuentra la gráfica correspondiente a estos resultados, pudiéndose discernir de una manera más clara como varía el ancho de banda tanto si el número de servicios aumenta como si el número de nodos en la red aumenta, llegando a las siguientes conclusiones:

1. El ancho de banda consumido por la red MANET aumenta linealmente cuando aumenta el número de servicios anunciado por cada nodo. Este resultado era lo esperado, ya que el número de mensajes SDM encapsulados en el paquete OLSR aumenta de forma lineal con el número de servicios y por lo tanto el valor del ancho de banda.

2. El ancho de banda consumido por la red MANET aumenta cuanto mayor es el número de nodos presentes. Esto es debido a que el número de

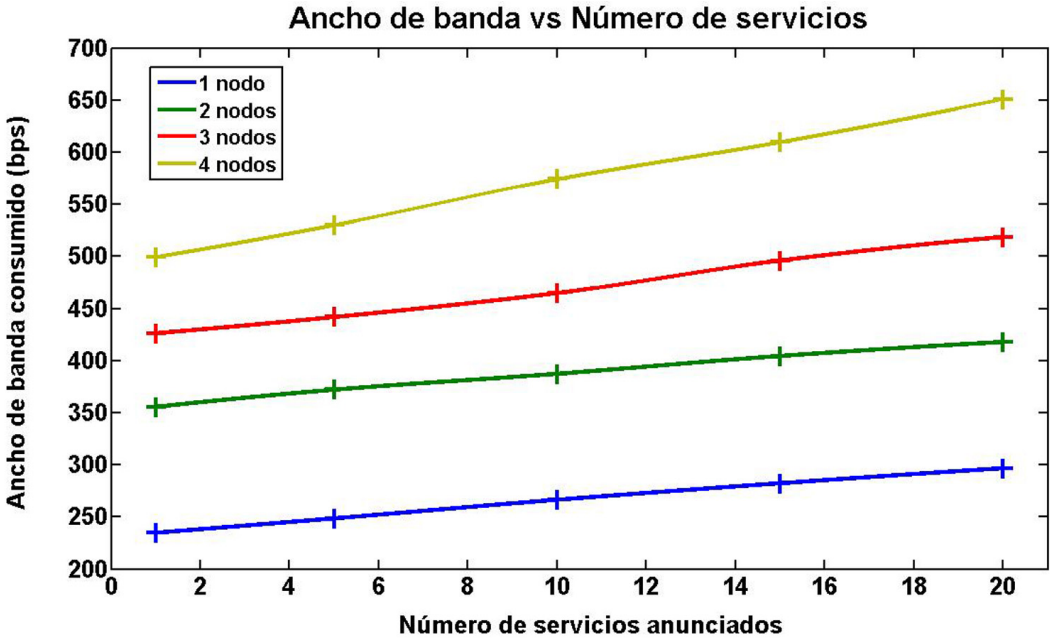

Figura 8.10: Ancho de banda en función del número de servicios anunciados

retransmisiones aumenta ya que los mensajes OLSR y SDM necesitan llegar a más nodos de la red, aumentando de igual manera el ancho de banda.

Aún así, el motivo por el que el ancho de banda en la configuración de 2 nodos es mayor que en el de 1 nodo es algo diferente, puesto que en el caso de 2 nodos no hay retransmisiones y sin embargo el ancho de banda es mayor como se observa en la Figura 8.10. En este caso el aumento del ancho de banda se debe al protocolo de routing OLSR, porque al tener cada nodo un vecino empiezan a retransmitirse mensajes `TC` (algo que no ocurre en la configuración de 1 nodo pues éste no tiene vecinos), aumentando también el tamaño de los mensajes `HELLO` pues anuncia la presencia de 1 vecino. El envío de nuevos mensajes y el aumento de tamaño del otro tipo de mensajes hacen que el ancho de banda consumido aumente en consecuencia.

8.6.2. Intervalo entre mensajes (`SDM Interval`)

Los resultados experimentales obtenidos sobre el ancho de banda consumido (en bps) por cada una de las configuraciones de red en función del intervalo entre mensajes (`SDM Interval`) cuando el número de servicios anunciados por

cada dispositivo es constante se pueden observar en el Cuadro 8.6.

En la Figura 8.11 se encuentra la gráfica correspondiente a estos resultados, pudiéndose discernir de una manera más clara como varía el ancho de banda tanto si el intervalo entre mensajes SDM aumenta como si el número de nodos en la red aumenta, llegando a las siguientes conclusiones:

	1 seg.	10 seg.	20 seg.	30 seg.	40 seg.
1 Nodo	861.65	281.96	248.64	239.86	234.77
2 Nodos	943.24	401.89	372.11	362.65	355.41
3 Nodos	1251.38	489.94	441.86	427.99	420.86
4 Nodos	1642.88	591.50	530.14	514.76	503.42

Cuadro 8.6: Ancho de banda en función del valor de SDM Interval

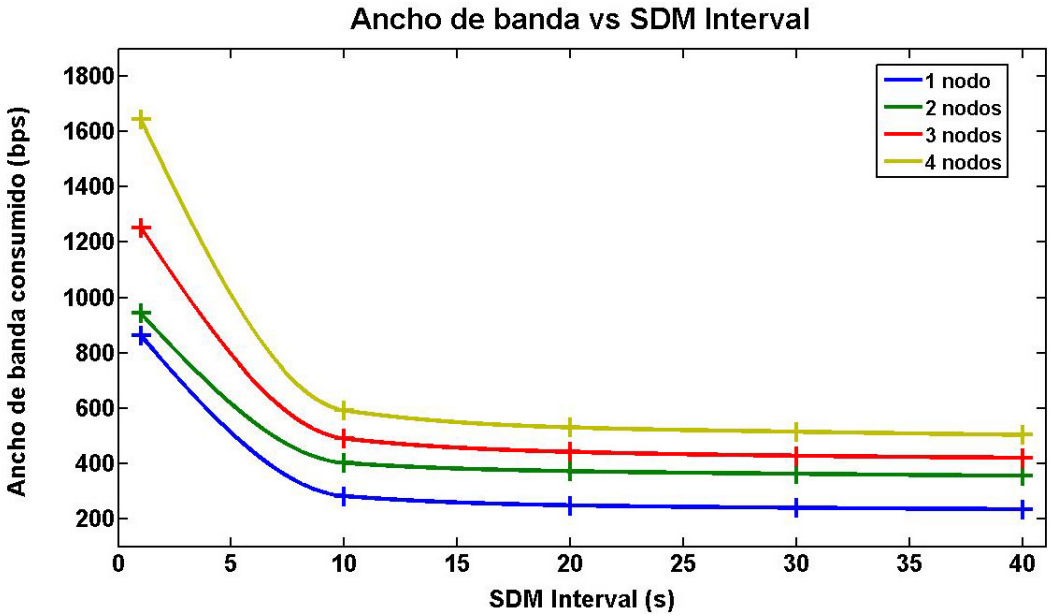

Figura 8.11: Ancho de banda en función del valor de SDM Interval

1. El ancho de banda consumido por la red MANET disminuye exponencialmente cuando aumenta el intervalo de tiempo entre mensajes SDM.

Cuando el valor de `SDM Interval` es muy pequeño, por ejemplo 1 segundo, la cantidad de tráfico es considerable ya que constantemente se envían mensajes SDM, además de las retransmisiones en configuraciones de red con más de 2 nodos (se puede ver claramente en la Figura 8.11 que el aumento de ancho de banda consumido es considerable cuando pasa de 2 a 3 nodos). Al disminuir de forma exponencial, también se puede observar que a partir de 20 segundos el ancho de banda consumido no disminuye significativamente aunque aumentemos el intervalo entre mensajes SDM, por lo que 20 segundos podría resultar un buen valor de `SDM Interval` por defecto.

2. Al igual que en el apartado anterior, el ancho de banda consumido por la red MANET aumenta cuanto mayor es el número de nodos presentes, siendo principalmente a causa del número de retransmisiones de los mensajes OLSR y SDM que se producen en la red. También hay que tener en cuenta el aumento de tamaño de los mensajes TC y OLSR al aumentar el número de vecinos (en el paso de 1 nodo a 2 nodos como se comentó en el apartado anterior).

8.6.3. Comparación con el valor teórico

Una vez analizados los valores del ancho de banda obtenidos experimentalmente con una red MANET de dispositivos reales, se procederá a su comparación con los valores teóricos obtenidos en el apartado 8.5. Su cálculo se ha podido realizar gracias a que se conoce el tamaño de los mensajes OLSR, de los mensajes SDM y de las cabeceras que le preceden (Ethernet, IP y UDP), el intervalo entre cada uno de los mensajes y el número de retransmisiones aproximadas que deben realizar los nodos internos para que cada mensaje llegue a todos los rincones de la red MANET.

El Cuadro 8.7 muestra los valores reales y teóricos del ancho de banda consumido (en bps) para una configuración de red de 4 nodos en función del número de servicios anunciados por cada dispositivo cuando el intervalo entre mensajes SDM es constante. Su gráfica correspondiente se puede observar en la Figura 8.12.

El Cuadro 8.8 muestra los valores reales y teóricos del ancho de banda consumido (en bps) para una configuración de red de 4 nodos en función del intervalo entre mensajes (`SDM Interval`) cuando el número de servicios anunciados por cada dispositivo es constante. Su gráfica correspondiente se puede observar en la Figura 8.13.

	1 servicio	5 servicios	10 servicios	15 servicios	20 servicios
Teoría	610.00	642.00	682.00	722.00	762.00
Real	499.53	530.14	574.17	609.78	650.89

Cuadro 8.7: Tabla con los valores teóricos y reales del ancho de banda en función del número de servicios anunciados

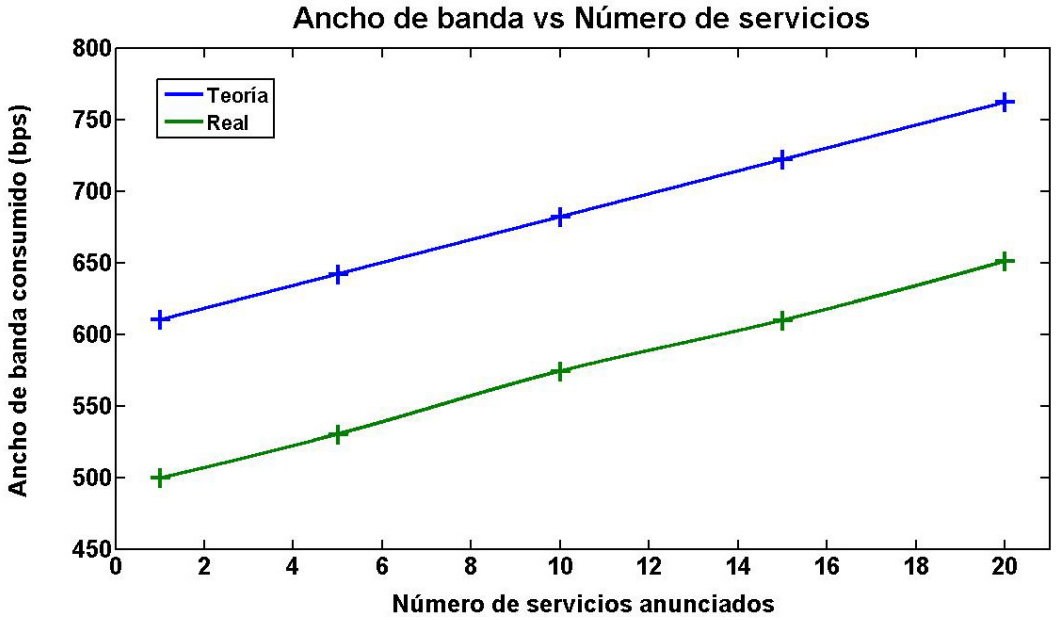

Figura 8.12: Comparación del ancho de banda real con el teórico en función del número de servicios anunciados

Si se analizan las dos gráficas de las Figuras 8.12 y 8.13 se llega a la conclusión de que el valor teórico, tanto en función del número de servicios anunciados como del intervalo entre mensajes SDM, es mayor que el valor real. Esto se debe principalmente a la propiedad de *piggybacking* que posee el protocolo OLSR, que es capaz de encapsular varios mensajes, ya sean `HELLO`, `TC` o de otro protocolo diferente (en nuestro caso SDM), en un mismo paquete OLSR. Por lo tanto cualquier mensaje puede ser enviado con otros mensajes (si van a ser transmitidos al mismo tiempo), ahorrándose así ancho de banda, ya que se envían varios mensajes pero con una sola cabecera (Ethernet, IP, UDP y OLSR). En los cálculos teóricos presentes en el apartado 8.5 se ha supuesto que todos los mensajes se han enviado individualmente, con el consiguiente aumento en el ancho de banda respecto a los valores reales.

	1 seg.	10 seg.	20 seg.	30 seg.	40 seg.
Teoría	2504.00	740.00	642.00	609.33	593.00
Real	1642.88	591.50	530.14	514.76	503.42

Cuadro 8.8: Tabla con los valores teóricos y reales del ancho de banda en función del valor de SDM Interval

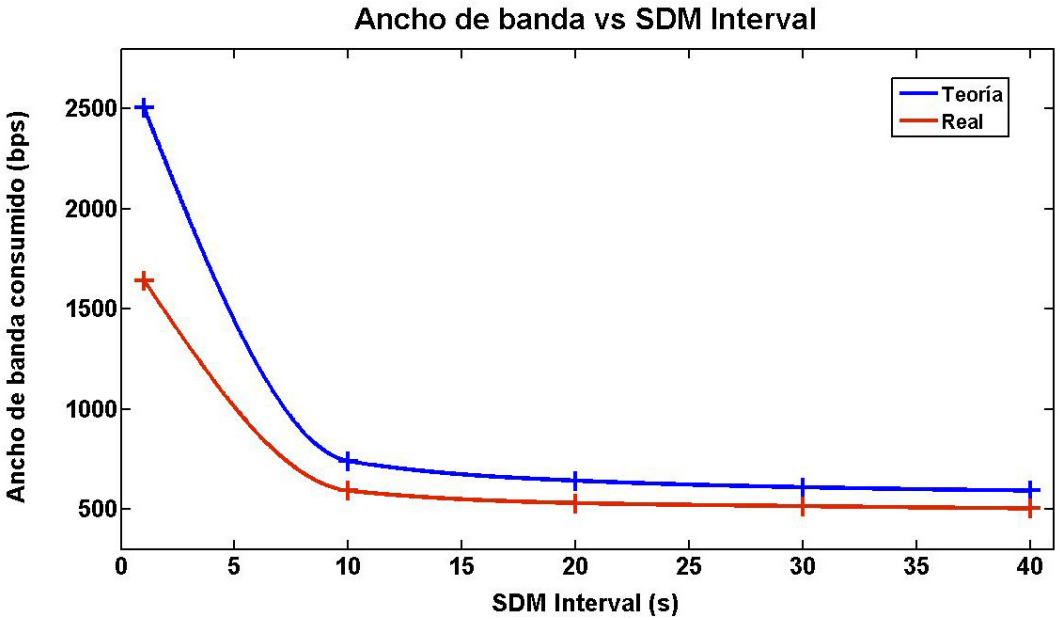

Figura 8.13: Comparación del ancho de banda real con el teórico en función del valor de SDM Interval

Para comprobar que esta diferencia entre los valores teóricos y reales se debe a la propiedad de *piggybacking* que posee el protocolo OLSR, se va a calcular el ancho de banda (en bps) que se ahorra el protocolo al encapsular los mensajes teniendo en cuenta la diferencia del número de paquetes teóricos enviados con los que se envían realmente y el tamaño de la cabecera de dichos paquetes.

Para el cálculo teórico del número medio de paquetes enviados hay que tener en cuenta que depende del intervalo de tiempo entre mensajes, ya sean `HELLO`, `TC` o `SDM` (el número de servicios anunciados por el protocolo SDM no afecta al número de paquetes enviados sino al tamaño de dichos paquetes). Por lo tanto el número medio de paquetes por minuto viene dado por:

$$Paquetes = \frac{60\ segundos}{HELLO\ Interval} + \frac{60\ segundos}{TC\ Interval} + \frac{60\ segundos}{SDM\ Interval}\ bps$$

En el Cuadro 8.9 se muestran los valores del número medio de paquetes por minuto calculado teóricamente en función del valor de `SDM Interval` y para una configuración de red de 4 nodos, sabiendo que el intervalo entre mensajes `HELLO` es de 2 segundos, que el intervalo entre mensajes `TC` es de 5 segundos y que los mensajes `HELLO` no se retransmiten al resto de la red. También se muestran en el Cuadro 8.9 los valores del número medio de paquetes medidos experimentalmente y su diferencia con los valores teóricos.

	1 seg.	10 seg.	20 seg.	30 seg.	40 seg.
Teoría	210	75	68	65	64
Real	82	51	49	46	46
Diferencia	**128**	**24**	**19**	**19**	**18**

Cuadro 8.9: Tabla con la diferencia en el número de paquetes teóricos y reales enviados por minuto

Con la diferencia entre los valores teóricos y reales se va a proceder al cálculo del ancho de banda que se ahorra el protocolo OLSR al utilizar la técnica de *piggybacking*, sabiendo que el tamaño de la cabecera del paquete OLSR viene dado por:

$$Cabecera\ (Ethernet + IP + UDP + OLSR) = 14 + 20 + 8 + 4 = 46\ bytes$$

Luego el ancho de banda que se ahorra el protocolo OLSR al utilizar la técnica de *piggybacking* es igual a:

$$Bandwidth = \frac{8 * (Numero\ paquetes * Cabecera\ paquete)}{60\ segundos}\ bps$$

En el Cuadro 8.10 se muestran los valores del ancho de banda ahorrado gracias a la técnica de *piggybacking* en función del valor de `SDM Interval` y para una configuración de red de 4 nodos. Como se puede apreciar, el valor de `SDM Interval` donde más se ahorra ancho de banda es 1 segundo ya que tanto el intervalo de los mensajes `HELLO` (2 segundos) como el de los mensajes `TC` (5

segundos) son múltiplos de `SDM Interval`, por lo que probablemente en cada paquete OLSR que contenga uno de estos mensajes también lleve encapsulado un mensaje SDM, aprovechando al máximo la técnica de *piggybacking*. Aún así, a pesar del ahorro considerable en ancho de banda con respecto a otros valores de `SDM Interval`, el ancho de banda consumido para este intervalo es muy grande (ver Figura 8.13), haciéndolo inviable para su uso en redes MANET (donde el ancho de banda es muy limitado).

1 seg.	10 seg.	20 seg.	30 seg.	40 seg.
785.07	147.20	116.53	116.53	110.40

Cuadro 8.10: Tabla con los valores del consumo de ancho de banda ahorrado al utilizar *piggybacking*

Por último en las gráficas de las Figuras 8.14 y 8.15 se compara el ancho de banda consumido teóricamente con la suma del ancho de banda real más el ancho de banda ahorrado, tanto en función del número de servicios anunciados como en función del intervalo entre mensajes SDM. Como se puede observar en estas gráficas, al sumarle el ancho de banda ahorrado los valores teóricos y reales son muy similares, demostrando que las diferencias en los valores del ancho de banda de las Figuras 8.12 y 8.13 se deben a la propiedad de *piggybacking* que posee el protocolo OLSR para encapsular varios mensajes (`HELLO`, `TC` o `SDM`) en un paquete OLSR. Además se demuestra que esta técnica es muy útil y eficaz ya que se consume menos ancho de banda del requerido, especialmente indicado para redes MANET donde las limitaciones en ancho de banda son muy grandes.

8.7. Retardo del protocolo SDM

En esta prueba se calculará experimentalmente el retardo existente en el protocolo de descubrimiento de servicios SDM tanto en el anuncio de servicios como en la notificación que se envía al resto de la red cuando se elimina algún servicio. El retardo obtenido en este caso consiste en el tiempo desde que se da la acción de anunciar o eliminar un servicio hasta que dicho mensaje SDM es procesado por el destinatario. Dicho retardo se ha calculado entre los nodos externos para una configuración de red donde hubiera 2, 3 y 4 nodos en línea (ver Figura 8.9). A continuación se explica el proceso que tiene que seguir cada

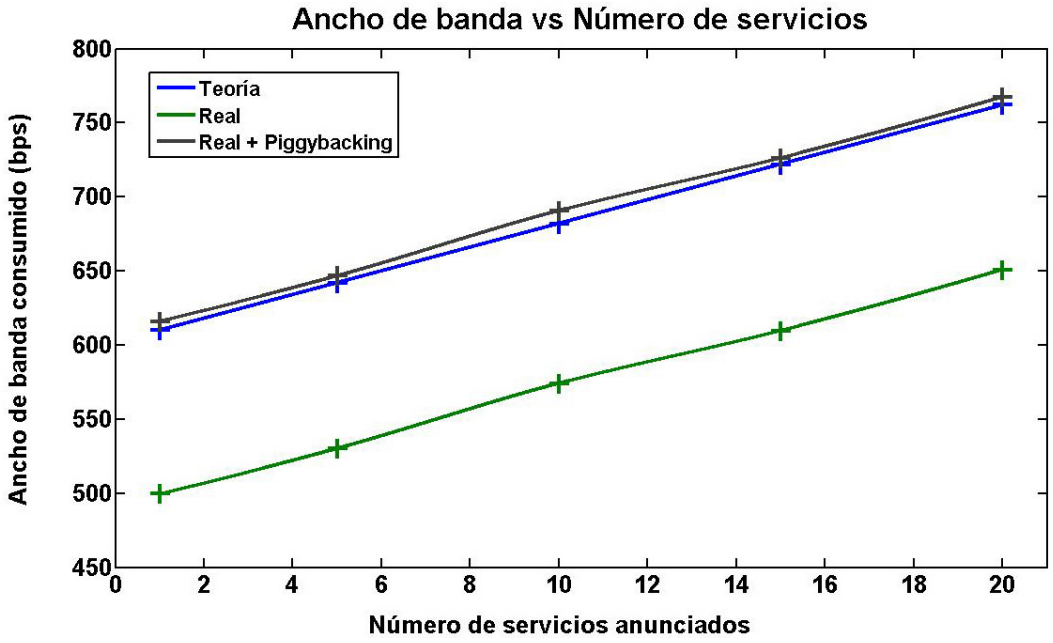

Figura 8.14: Comparación del ancho de banda real con el teórico en función del número de servicios anunciados (Piggybacking)

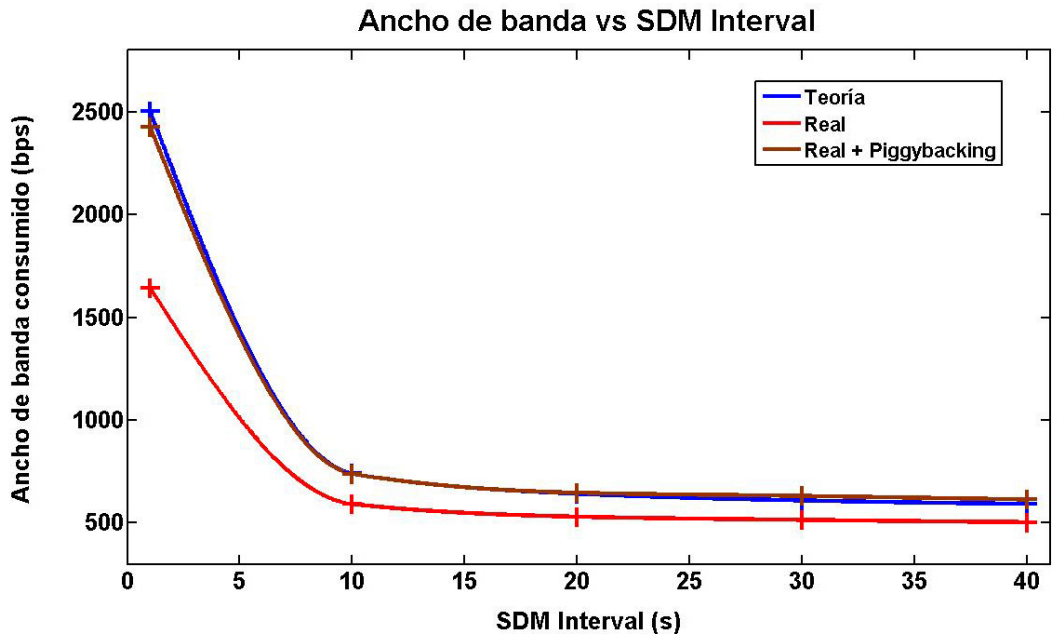

Figura 8.15: Comparación del ancho de banda real con el teórico en función del valor de SDM Interval (Piggybacking)

acción (anuncio y eliminación de servicios) para poder entender los valores reales que se darán posteriormente.

- **Anuncio de servicios:** El retardo en el anuncio de servicios consiste en el almacenamiento del servicio que se va a anunciar en la caché de servicios locales, la creación del mensaje SDM con todos los servicios a anunciar, el encapsulamiento del mensaje SDM en el paquete OLSR, el envío de dicho paquete a través de la interfaz correspondiente y la propagación hasta el destinatario (posibles retransmisiones por el camino), el procesamiento del mensaje SDM, la actualización de la caché de tiempos y el almacenamiento del servicio en la caché de servicios anunciados por otros nodos, tal y como se explica en el apartado 6.2.1.

- **Eliminación de servicios:** El retardo en la notificación de la eliminación de servicios consiste en el borrado del servicio a eliminar de la caché de servicios locales, la creación del mensaje SDM con la notificación del servicio eliminado, el encapsulamiento del mensaje SDM en el paquete OLSR, el envío de dicho paquete a través de la interfaz correspondiente y la propagación hasta el destinatario (posibles retransmisiones por el camino), el procesamiento del mensaje SDM y la actualización de la caché de tiempos y del campo `Lifetime` del servicio a eliminar en la caché correpondiente con un valor igual a 0, tal y como se explica en el apartado 6.2.5.

Aunque teóricamente la eliminación de un servicio va desde que se da la orden en el nodo origen hasta que se borra dicho servicio en la caché del nodo destino, en nuestro caso solamente vamos a medir el retardo hasta que se actualiza el campo `Lifetime` del servicio a eliminar con un valor igual a 0. Esto es debido a que la eliminación del servicio propiamente dicho (borrado del elemento de la lista doble enlazada en la caché de servicios anunciados por otros nodos) se produce en el chequeo que realiza la aplicación (en nuestro caso cada 0.5 segundos) para comprobar si algún servicio ha caducado (ver sección 7.9.4). Al ser este valor de espera variable no se ha incluido para la obtención del retardo (pues desde que se actualiza la caché no se sabe exactamente cuánto hay que esperar para que se realice su chequeo y se elimine el servicio correspondiente). Además al poder ser cambiado el valor de dicha espera se podría manipular de alguna manera el valor del retardo si el borrado del servicio en la caché correspondiente fuera incluido.

Para esta prueba se han realizado 5 tandas de medidas para cada configuración de red diferente, siendo el valor final la media de estos 5 valores, obteniendo

a su vez la desviación estándar y el intervalo de confianza al 95 % (aunque en este capítulo sólo se muestran los valores finales, todas las medidas obtenidas están recogidas en el anexo C para cada uno de las pruebas realizadas). Al contrario que en el consumo del ancho de banda, el intervalo de confianza al 95 % en este caso sí tiene un valor significativo y por lo tanto es mostrado en las gráficas correspondientes.

Los resultados experimentales obtenidos sobre el retardo en el anuncio y eliminación de servicios para una configuración de red de 2, 3 y 4 nodos se pueden observar en el Cuadro 8.11. La gráfica correspondiente a estos valores se encuentra en la Figura 8.16.

	2 Nodos	3 Nodos	4 Nodos
Anuncio servicios	0.0357	0.9862	1.7421
Eliminación servicios	0.0344	0.8506	1.4613

Cuadro 8.11: Retardo en el anuncio y eliminación de servicios en función del número de nodos

Como era de esperar, en la gráfica de la Figura 8.16 se observa que cuanto mayor es el número de nodos presentes en la red, mayor es el retardo producido por el envío de un mensaje SDM de un extremo al otro de la red MANET. Aún así también se puede apreciar que dicho aumento es cada vez menor según el número de nodos se hace más mayor. Esto es debido a que como se ha explicado anteriormente, el retardo consiste tanto en las transmisiones (y retransmisiones) del mensaje a través de la red (que aumenta con el número de nodos), como en el procesamiento del mensaje SDM por parte de los extremos (que aproximadamente es constante independientemente del número de nodos).

Si comparamos el retardo que se produce en el anuncio de servicios con el que se produce en la eliminación de servicios, se llega a la conclusión de que ambos retardos tienen valores muy similares (parte de sus intervalos de confianza al 95 % se solapan), aunque se aprecia que el tiempo que se tarda en eliminar un servicio y notificarlo al resto de la red es algo menor que en anunciarlo. La razón de esta diferencia consiste en la parte del procesamiento del mensaje por parte del origen y del destino, ya que antes de almacenar un servicio tanto en la caché de servicios locales como en la de servicios externos se debe recorrer la lista de servicios entera para comprobar que no exista ya dicho servicio en la caché, al contrario que en la eliminación de un servicio pues

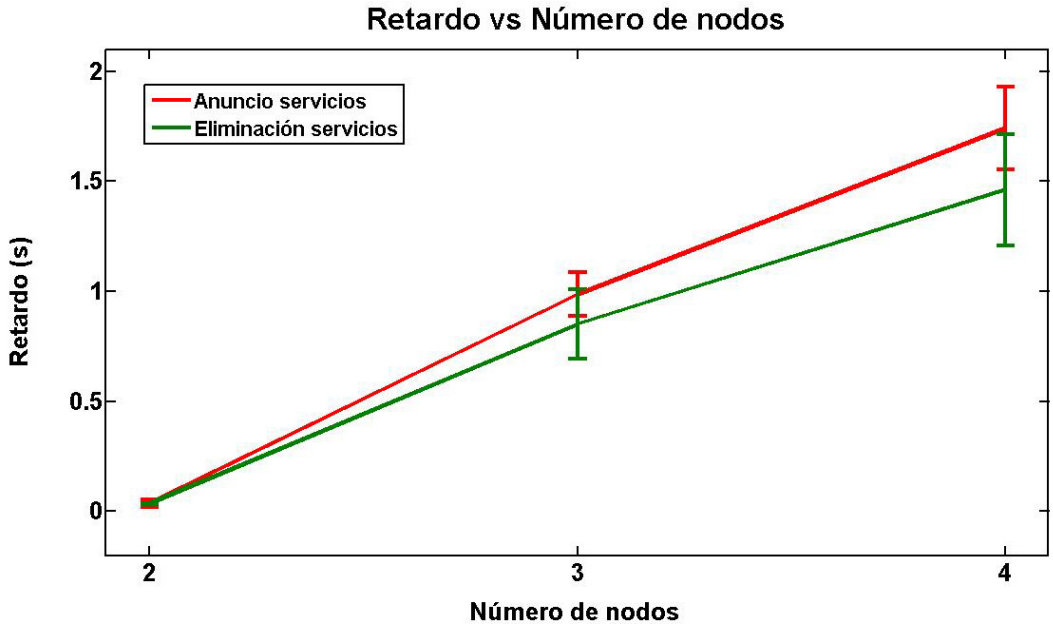

Figura 8.16: Comparación del retardo de anuncio de servicios con el retardo de eliminación de servicios

cuando encuentra dicho servicio ya no continúa recorriendo la lista. El hecho de recorrer la lista entera antes de almacenar el servicio en la caché hace que el retardo en el anuncio de servicios sea algo mayor que en su eliminación.

Aunque el borrado de un servicio ya no disponible en la caché de servicios externos no ha sido incluido en el retardo de eliminación de servicios debido al tiempo de espera que existe entre el procesamiento del mensaje SDM y el chequeo de la caché, se ha calculado expresamente el tiempo de borrado del servicio (desde que la aplicación empieza a recorrer la lista hasta que la información relativa al servicio es liberado de la memoria), dando lugar a un valor aproximado de 0.2 segundos.

Por último cabe destacar que a mayor número de nodos el intervalo de confianza al 95 % es mayor (dando lugar también a una mayor separación entre el valor del retardo en el anuncio de servicios y en su eliminación), algo comprensible si se tiene en cuenta que a mayor número de nodos mayor es el número de acciones a realizar (procesamiento de mensajes, retransmisiones, etc.) y por lo tanto la varianza del resultado aumenta.

8.8. Comparativa con el protocolo Mercury

Por último se va a realizar una comparación de rendimiento con el protocolo de descubrimiento de servicios Mercury [53], diseñado por Joakim Flathagen en 2008 para su tesis de máster en la universidad de Oslo. Al ser realizada la implementación del protocolo Mercury en lenguaje C y para el demonio olsrd (al igual que el protocolo SDM), la portabilidad a un dispositivo Maemo se ha podido conseguir sin grandes dificultades. En primer lugar se realizará una pequeña descripción de este protocolo para poder entender posteriormente los resultados experimentales obtenidos con dispositivos reales, habiendo realizado pruebas sobre el ancho de banda consumido y el retardo en el envío de mensajes. Después de mostrarse los valores de dichas pruebas se hará un estudio comparativo entre ambos protocolos de descubrimiento de servicios.

8.8.1. Descripción del protocolo Mercury

El protocolo Mercury introduce un nuevo tipo de mensaje al protocolo OLSR denominado *Mercury Service Discovery* (MSD), con el que es capaz tanto de anunciar como de solicitar servicios al resto de nodos de la red, siendo por tanto su modo de funcionamiento una mezcla de los modos *push* y *pull* (a diferencia del protocolo SDM que solamente puede anunciar servicios). Para la descripción de servicios, el protocolo Mercury utiliza filtros Bloom, que son capaces de en un pequeño espacio de memoria (128 bytes) contener un resumen de **todos** los servicios disponibles en el dispositivo (hasta un máximo de 256). Para ello crea una función hash para cada servicio, mapeando su resultado en la lista de servicios disponibles. El problema de utilizar este tipo de filtros es que puede llegar a dar un falso positivo si un nodo ofrece muchos servicios y se solapan sus resultados en el espacio de memoria asignado.

El funcionamiento del protocolo Mercury es el siguiente:

- Cuando un servicio empieza a estar disponible en un nodo, éste lo anuncia una sola vez a toda la red. El resto de nodos poseen una caché en la que almacenan temporalmente los servicios anunciados por dicho nodo (el tiempo de almacenamiento es un parámetro configurable por el usuario).

- Cuando un nodo requiere un servicio, primero mira en la caché de servicios locales para comprobar si éste se encuentra disponible en algún nodo de la red. Si no es así, envía una solicitud al resto de nodos de la red a través de un mensaje MSD (modo *pull*). Al recibir dicha solicitud, el

resto de nodos miran en su caché de servicios locales, y si poseen el servicio que pide el nodo, envían a toda la red (y no solamente al nodo que solicita el servicio) una respuesta indicando la disponibilidad de dicho servicio a través de él. De esta manera se inunda la información por toda la red (modo *push*), por si se da el caso de que algún otro nodo está también interesado en utilizar dicho servicio y así no necesita mandar otra solicitud.

La diferencia principal del protocolo Mercury con el protocolo SDM se basa principalmente en el modo de descubrir los servicios, mientras que Mercury solicita y anuncia servicios (modo *push* y *pull*), SDM sólo anuncia servicios (modo *push*). En las pruebas realizadas para este apartado se ha eliminado la caché de servicios del protocolo Mercury (poniendo un valor de tiempo de permanencia en la caché de 0 segundos). De esta manera cuando un nodo solicita un servicio siempre tiene que enviar un mensaje de petición al resto de la red, anulando de esta manera las ventajas que proporciona el modo *push* del protocolo Mercury. Así se puede también comparar experimentalmente las ventajas e inconvenientes que presentan los protocolos de descubrimiento de servicios que trabajan en modo *push* y *pull*.

8.8.2. Comparación del ancho de banda consumido

En la comparación del ancho de banda consumido por los protocolos Mercury y SDM hay que tener en cuenta varias cosas. En primer lugar, el modo de funcionamiento de ambos protocolos es diferente. Mientras en el protocolo SDM se ha medido el ancho de banda consumido en función del número de servicios anunciados por cada dispositivo, en el protocolo Mercury se ha medido el ancho de banda consumido en función del número de servicios pedidos por minuto por uno de los dispositivos al resto de la red (y sus correspondientes respuestas del resto de nodos). Para ambos protocolos, todos los nodos que componían la red MANET poseían los mismos servicios, por lo que al realizarse una petición en el protocolo Mercury, el resto de nodos de la red contestaban a esa petición.

Los resultados experimentales obtenidos sobre el ancho de banda consumido por los protocolos Mercury y SDM en función del número de servicios anunciados/pedidos cuando el intervalo entre mensajes SDM es constante (20 segundos) y el tiempo de permanencia de la caché en Mercury es nulo se pueden observar en el Cuadro 8.12. Su gráfica correspondiente se puede observar en la Figura 8.17.

	1 servicio	5 servicios	10 servicios	15 servicios
SDM	499.53	530.14	574.17	609.78
Mercury	483.89	511.00	534.40	561.82

Cuadro 8.12: Ancho de banda del protocolo SDM y Mercury en función del número de servicios anunciados/pedidos

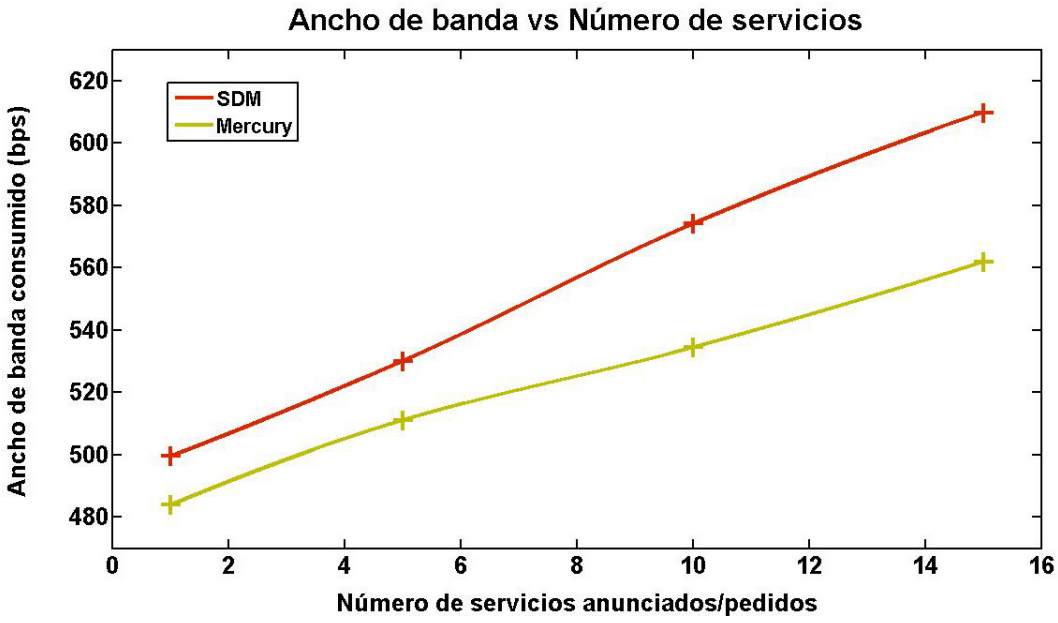

Figura 8.17: Comparación del ancho de banda consumido por SDM y Mercury en función del número de servicios anunciados/pedidos

Analizando la gráfica de la Figura 8.17 se podría llegar a la conclusión de que para un mismo número de servicios anunciados/solicitados, el protocolo SDM consume mayor ancho de banda que el protocolo Mercury, pero al no estar realmente midiendo lo mismo es difícil ver qué protocolo consume más ancho de banda (en el protocolo SDM todos los nodos anuncian servicios, por lo que toda la red conoce los servicios de cada uno, mientras que en el protocolo Mercury solamente es un nodo el que solicita servicios al resto de la red). De modo que lo que se puede intentar hacer es una equivalencia, es decir, ver cuántos servicios por minuto tiene que solicitar un nodo para que consuma el mismo ancho de banda que un determinado número de anuncios de servicios. Observando la gráfica, se ve que el anuncio de 5 servicios con el protocolo SDM equivale a que un nodo solicite 10 servicios por minuto con el protocolo Mercury, algo que se puede dar con bastante frecuencia en cualquier red MANET. En conclusión, ambos protocolos tienen unos niveles de consumo de ancho de banda razonables, y aunque sea bastante difícil probarlo en la realidad debido a sus diferentes modos de funcionamiento, parece ser que el protocolo SDM consume un mayor de ancho de banda que el protocolo Mercury, algo de esperar conociendo su naturaleza proactiva.

8.8.3. Comparación del consumo energético

Para hacernos una ligera idea de qué protocolo consume más energía, se ha realizado una comparativa del número medio de paquetes por minuto generados tanto por el protocolo SDM como por el protocolo Mercury, ya que según [72] el consumo energético del envío de un paquete por una interfaz WLAN se divide en dos componentes: un consumo fijo por el hecho de transmitir un paquete y otro consumo variable dependiendo del tamaño del paquete, siendo el consumo fijo bastante más alto que el variable. Por lo tanto, aquel protocolo que genere menos paquetes por minuto probablemente consumirá un menor nivel de batería.

En el Cuadro 8.13 se muestran los valores del número medio de paquetes por minuto generado por el protocolo SDM en función del intervalo de tiempo entre mensajes SDM y en el Cuadro 8.14 se muestran los valores del número medio de paquetes por minuto generado por el protocolo Mercury en función del número de servicios solicitados por minuto.

Aunque como se ha comentado anteriormente es muy difícil comparar ambos protocolos al ser su modo de funcionamiento diferente, se observa que los valores de los Cuadros 8.13 y 8.14 son muy similares, por lo que se puede llegar a la conclusión de que el consumo energético de ambos protocolos es muy pa-

1 seg.	10 seg.	20 seg.	30 seg.	40 seg.
82	51	49	46	46

Cuadro 8.13: Número medio de paquetes por minuto (Protocolo SDM)

1 Servicio	5 Servicios	10 Servicios	15 Servicios
45	50	54	56

Cuadro 8.14: Número medio de paquetes por minuto (Protocolo Mercury)

recido para valores razonables tanto en el intervalo de tiempo entre mensajes SDM como en el número de solicitudes por minuto (ya que cuando el valor de `SDM Interval` es 1 segundo, el número medio de paquetes se dispara y por tanto su consumo energético).

8.8.4. Comparación del retardo

A continuación se muestra la comparación del retardo que se produce en los protocolos Mercury y SDM. Al igual que en el apartado anterior, debido a las diferencias entre ambos protocolos el concepto de retardo es algo diferente. Mientras que en el protocolo SDM el retardo es el tiempo que existe entre que un extremo de la red envía un mensaje SDM anunciando un servicio y el otro extremo lo procesa, en el protocolo Mercury el retardo es el tiempo que tarda en enviarse un mensaje de petición al otro extremo, que éste lo procese y enviar la respuesta al nodo origen (ver Figura 8.18).

Los resultados experimentales obtenidos sobre el retardo en el anuncio/petición de servicios en los protocolos Mercury y SDM para una configuración de red de 2, 3 y 4 nodos se pueden observar en el Cuadro 8.15. Su gráfica correspondiente se puede observar en la Figura 8.19, donde se muestra que el retardo del protocolo Mercury en la petición de un servicio (y su consiguiente respuesta) es mayor que en el retardo del protocolo SDM en el anuncio de un servicio como cabía de esperar, puesto que el camino de los mensajes en el protocolo Mercury es de ida (petición) y vuelta (respuesta) mientras que el camino de los mensajes en el protocolo SDM es sólo de ida (anuncio). Además se aprecia claramente que a mayor número de nodos, mayor es la diferencia entre los retardos de ambos protocolos ya que la distancia entre los extremos

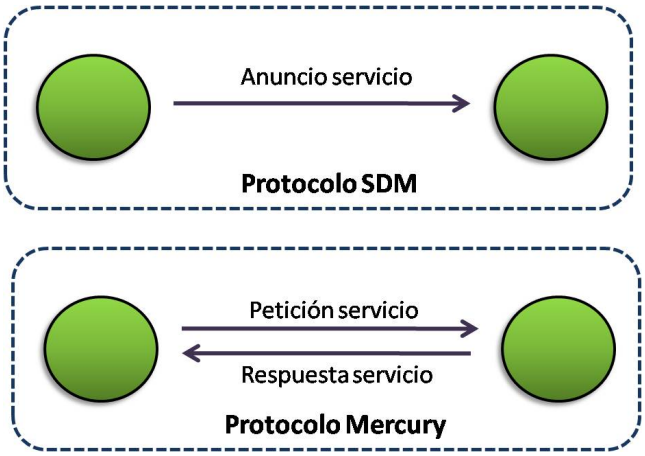

Figura 8.18: Concepto de retardo en el protocolo SDM y Mercury

es más grande y por tanto se tarda más tiempo en recorrer el camino de vuelta (principal diferencia en el retardo de los dos protocolos).

	2 Nodos	3 Nodos	4 Nodos
SDM	0.0357	0.9862	1.7421
Mercury	0.1321	1.7330	2.7533

Cuadro 8.15: Retardo en el anuncio/petición de servicios del protocolo SDM y Mercury

Por último se ha comparado el retardo del protocolo Mercury en la petición y respuesta de un servicio obtenido experimentalmente en este proyecto con el que aparece en la tesis de máster realizada por el autor del protocolo Mercury, Joakim Flathagen [53]. En la gráfica de la Figura 8.20 se muestran los valores de ambos experimentos para una configuración de red de 2, 3 y 4 nodos, viéndose claramente que el retardo medido por Joakim Flathagen es inferior al obtenido en este proyecto. La causa principal de esta diferencia radica en los dispositivos utilizados para la creación y transmisión de mensajes Mercury, ya que mientras en [53] se emplean portátiles equipados con tarjetas WLAN, en este proyecto se emplean dispositivos móviles N800, muchísimo más limitados en procesamiento y memoria que los portátiles. Esto hace que los N800 empleen mucho más tiempo tanto en crear las peticiones (y respuestas) en el nodo origen (y destino) y en procesarlas tanto en el nodo destino (peticiones) como en el nodo origen (respuestas), además del procesamiento para su retransmisión.

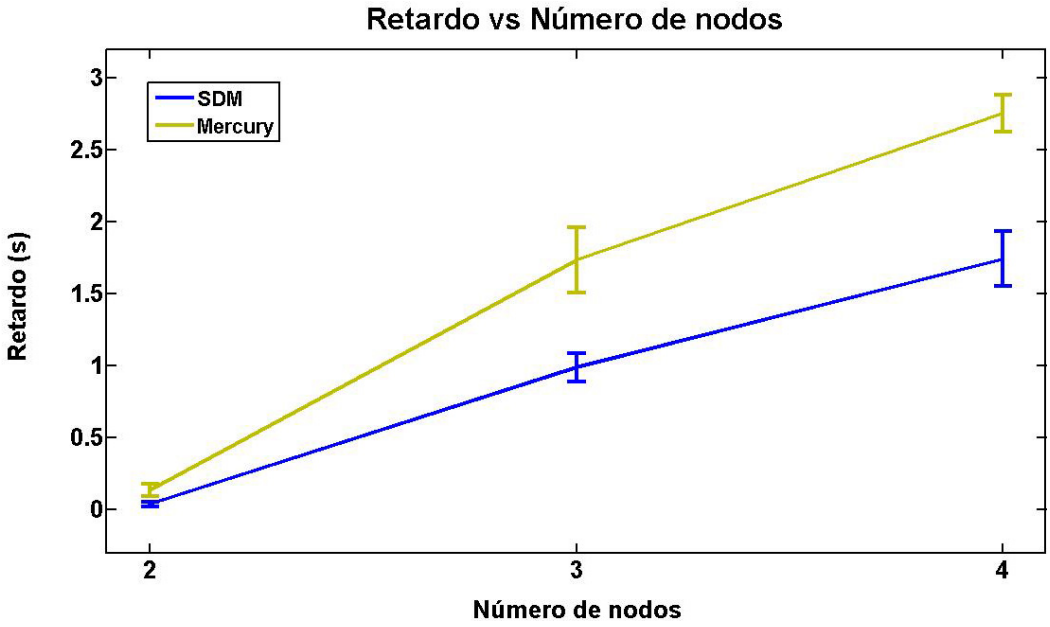

Figura 8.19: Comparación del retardo de anuncio de servicios de SDM y el retardo de petición de servicios de Mercury

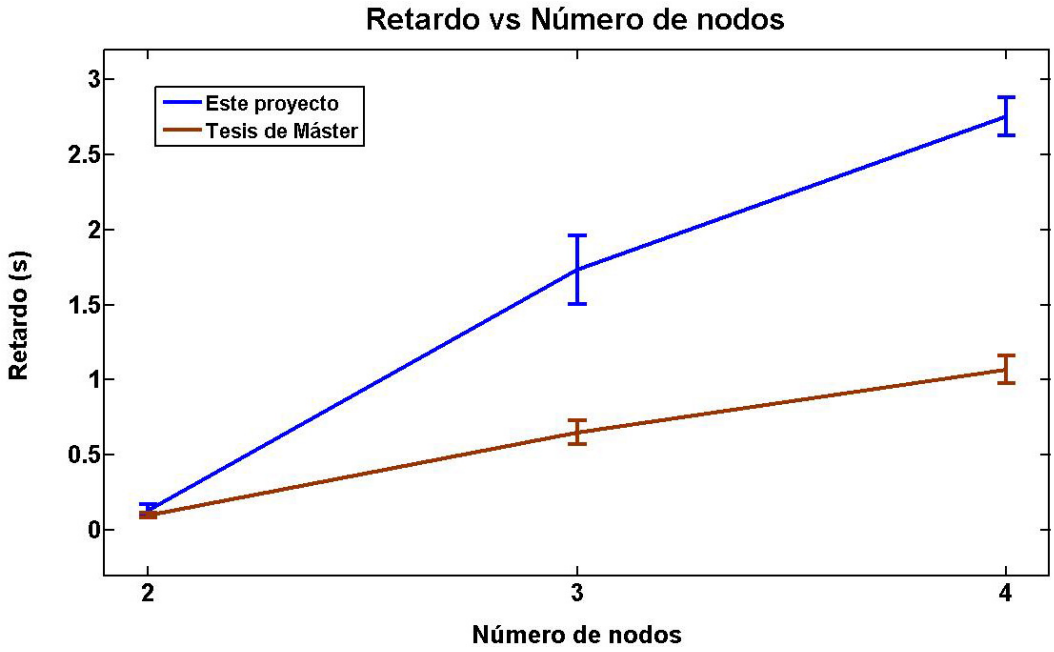

Figura 8.20: Comparación del retardo de petición de servicios de Mercury en este proyecto y en la tesis de máster [53]

Gracias a estas pruebas se han demostrado las ventajas e inconvenientes que presentan los protocolos de descubrimiento de servicios en modo *push* y *pull*. El protocolo de descubrimiento de servicios SDM (modo *push*) consume mayor ancho de banda debido a su naturaleza proactiva pero en cambio el retardo en anunciar sus anuncios es menor (además de que el tiempo que tarda en solicitar un servicio es nulo pues la información ya está almacenada en la caché del propio dispositivo). Por otra parte, el protocolo de descubrimiento de servicios Mercury (modo *pull*) tiene un mayor retardo al solicitar un servicio pero en cambio el ancho de banda consumido en general es menor, si no hay muchísima demanda y dependiendo de la actividad de los nodos presentes en la red MANET.

8.9. Conclusiones

En este capítulo se han realizado varias pruebas (consumo de energía, de ancho de banda, retardo, etc.) para comprobar experimentalmente el funcionamiento y rendimiento del protocolo de descubrimiento de servicios SDM. Las conclusiones a las que se pueden llegar son las siguientes:

- Con respecto al nivel de energía consumido, se observa que el demonio olsrd no consume muchísima batería de los dispositivos Nokia (alrededor de un 1 % para los nodos externos y un 1.5 % para los nodos internos en 1 hora). Se ha demostrado también que los nodos internos consumen mayor batería pues tienen que retransmitir los paquetes de los demás nodos al resto de la red (aproximadamente un 50 % más de consumo, aunque no es significativo para el nivel de batería total del dispositivo).

- Se ha comprobado que en la implementación del protocolo OLSR utilizado no se tiene en cuenta el nivel de *willingness* para seleccionar la mejor ruta por donde encaminar los paquetes OLSR, sino un parámetro denominado ETX que mide la calidad del enlace en ambas direcciones (enlace simétrico). A pesar de que este parámetro es un buen criterio para descubrir aquellas rutas que tienen una mayor capacidad en el envío de paquetes, no sería mala idea combinarlo también con el nivel de *willingness* pues se debería tener en cuenta aquellos dispositivos que tengan mayor nivel de batería (tienen menos posibilidades de desconectarse de la red si atendemos al criterio de consumo de batería).

- En el consumo del ancho de banda, se ha demostrado que existen tres parámetros fundamentales que le afectan de una manera crucial: el núme-

ro de servicios anunciados, el intervalo entre mensajes SDM y el número de nodos presentes en la red. De estos tres parámetros, sólo uno de ellos es configurable por el usuario, porque tanto el número de servicios como el de nodos dependen de las características de la red MANET y de los servicios disponibles en cada dispositivo. En cambio el intervalo entre mensajes SDM si puede ser configurado por el usuario, llegando a la conclusión de que un valor de `SDM Interval` de aproximadamente 20 segundos es razonable para que las cachés de servicios de los nodos estén apropiadamente actualizadas sin llegar a consumir un ancho de banda excesivo.

Además se han comparado los valores medidos experimentalmente con los teóricos, concluyendo que gracias a la técnica de *piggybacking* (donde varios mensajes OLSR y/o SDM son encapsulados en un mismo paquete OLSR) es posible ahorrar un ancho de banda significativo. En el caso de SDM, este ahorro será mayor si el valor de `SDM Interval` es múltiplo de los intervalos entre mensajes `HELLO` y `TC`, que en nuestro caso es 10, por lo que la decisión de asignar un valor de tiempo entre mensajes SDM de 20 segundos sigue siendo apropiada para no consumir mucho ancho de banda y tener a la vez las cachés debidamente actualizadas.

- Con respecto al retardo que se produce al anunciar y eliminar servicios, se observa que depende principalmente del número de nodos (y por tanto de retransmisiones) que se deben realizar para que un mensaje SDM llegue a todos los rincones de la red MANET. Además estos retardos tienen valores muy similares, aunque anunciar servicios produce un retardo algo mayor pues tiene que recorrer la lista entera antes de almacenar el nuevo servicio en la caché correspondiente. Cabe destacar que este retardo sólo afecta al anunciar un servicio la primera vez, ya que al ser procesado por otro nodo, éste lo almacena en su caché y cuando necesite utilizar dicho servicio lo mirará en su caché siendo la respuesta inmediata.

Por último se ha realizado una comparación con el protocolo de descubrimiento de servicios Mercury [53]. Aunque ambos protocolos tienen un modo de funcionamiento diferente (mientras que SDM tiene una naturaleza proactiva, Mercury es reactivo ya que hay que solicitar los servicios cuando se requieren en lugar de anunciarlos al resto de la red), se ha intentado medir en ambos protocolos el ancho de banda consumido y el retardo producido al enviar paquetes. Gracias a esta comparación se ha podido discernir las ventajas e inconvenientes que presentan cada uno de los modos de funcionamiento (*push* y *pull*) en un protocolo de descubrimiento de servicios.

Parte IV

Conclusiones y trabajos futuros

Capítulo 9

Conclusiones

Las redes MANET (*Mobile Ad-hoc NETworks*), caracterizadas por su descentralización y el dinamismo presente en la topología de la red (siempre cambiante con la llegada y salida constante de dispositivos), están empezando a ser utilizadas habitualmente en situaciones específicas donde la única posibilidad que existe es el despliegue de este tipo de redes (como por ejemplo en escenarios militares o de emergencia en zonas de difícil acceso o rurales).

Ante las limitaciones de los dispositivos empleados en este tipo de redes, la necesidad de utilizar protocolos y aplicaciones optimizadas para este tipo de escenarios es fundamental en el desarrollo de nuevos tipos de interconexiones en el que las personas puedan intercambiarse información entre ellas (a veces de vital interés), sobre todo en aquellas zonas donde se hace más difícil el uso de infraestructuras y por tanto el uso de las redes MANET se hace imprescindible.

En este proyecto se ha abordado el estudio de dos de los tipos de protocolos fundamentales en el despliegue y proliferación de cualquier red: el protocolo de routing y el protocolo de descubrimiento de servicios. Ambos protocolos son complementarios, pues mientras uno encuentra quiénes son el resto de nodos existentes en la red, el otro descubre los servicios que ofrecen para poder utilizarlos en su propio provecho. Debido a que su cometido es similar, se ha contribuido a las investigaciones ya realizadas en este campo con la implementación de un protocolo de descubrimiento de servicios denominado SDM que se caracteriza por integrarse en el protocolo de routing OLSR para aprovecharse de la información que obtiene este protocolo, ahorrando además ancho de banda, recurso muy limitado en las redes móviles ad-hoc.

Este capítulo presenta las contribuciones principales de este proyecto, un resumen de los resultados más significativos, posibles líneas de investigación

futuras para continuar con el trabajo realizado, sugerencias para optimizar el protocolo SDM implementado y las conclusiones finales del autor de este proyecto.

9.1. Contribuciones del proyecto

El objetivo principal del proyecto ha sido investigar las principales líneas de trabajo en el campo de los protocolos de routing y descubrimiento de servicios, centrándose en su mayor parte en las redes MANET, y en la implementación del protocolo de descubrimiento de servicios SDM como un plugin del demonio olsrd para comprobar su funcionamiento en dispositivos comerciales y medir su rendimiento en escenarios reales. Las principales contribuciones que se han realizado en este proyecto son las siguientes:

- Realizar un estudio del estado del arte de las diferentes configuraciones en las que se emplea la tecnología WLAN (estándar 802.11) para crear redes de comunicaciones y ser empleados en los incontables escenarios y situaciones que se pueden dar en la vida real. También se ha realizado un estudio del arte en el área de los protocolos de routing y de descubrimiento de servicios, tanto los utilizados a nivel general como los especializados y optimizados para su uso en redes MANET. De ambos protocolos se han descrito brevemente sus modos de funcionamiento y los ejemplos más empleados en la actualidad. Además se ha realizado un estudio sobre la plataforma de desarrollo de software abierto Maemo, basada en Linux y definida por Nokia junto con otros grupos de desarrollo de software libre, especificando sus principales características y componentes.

- Implementar el protocolo de descubrimiento de servicios SDM (*Service Discovery Mechanism*), diseñado específicamente para su utilización en redes MANET y que posee la particularidad de que está integrado en el protocolo de routing OLSR (*Optimized Link State Routing*). Esta implementación está pensada para ser utilizada en situaciones cotidianas de la vida real, haciendo posible el descubrimiento de servicios por parte de cualquier aplicación distribuida a través de una sencilla interfaz, y aunque en principio está enfocada para ser utilizada en dispositivos que utilicen la plataforma Maemo, al estar basada en Linux se pueden realizar portabilidades sin ninguna dificultad a otros equipos que utilicen también sistemas basados en Linux como Ubuntu, Mandriva, Fedora, etc.

- Evaluar la implementación del protocolo SDM en diferentes escenarios y

situaciones habituales que se pueden dar con un número determinado de dispositivos comerciales (Nokia N800) para comprobar el correcto funcionamiento del protocolo y medir su rendimiento en aspectos tales como la batería utilizada, el ancho de banda consumido o el retardo producido al anunciar servicios o notificar la eliminación de algún servicio. Además se ha realizado la comparación con otro protocolo de descubrimiento de servicios denominado Mercury, también integrado en el protocolo de routing OLSR pero con una visión de anunciar y solicitar servicios diferente a SDM. De esta manera se han podido obtener las principales ventajas e inconvenientes que presenta el utilizar uno u otro protocolo en las redes MANET.

9.2. Resultados más significativos

Al realizar el estudio sobre los protocolos de routing y descubrimiento de servicios se ha comprobado que existen muchísimas formas y maneras de realizar una misma tarea, cada cual con sus ventajas e inconvenientes. En los protocolos de routing están los que tienen una naturaleza proactiva anunciándose al resto de la red, consumiendo así más ancho de banda pero reduciendo el retardo, y los que son reactivos, esperando a que le soliciten información. Algo parecido ocurre con los protocolos de descubrimiento de servicios, donde existen aquellos que funcionan en modo *push*, anunciando sus servicios aunque no se los soliciten, y los que deben pedir el servicio al resto de la red para ver si algún nodo lo tiene (modo *pull*). Tanto en los protocolos de routing como en los de descubrimiento de servicios existen protocolos intermedios que intentan aunar las ventajas de ambos modos de funcionamiento (reactivo y proactivo).

Al realizar el estudio sobre la plataforma de desarrollo de software Maemo se ha descubierto la infinidad de posibilidades que ofrece dicho sistema, que al ser de código abierto todo grupo de desarrolladores que lo desee puede crear sus propias aplicaciones para que los usuarios de este sistema se beneficien y aprovechen al máximo las características que ofrecen los dispositivos en los que está integrado Maemo. Además al estar basado en Linux se pueden realizar sin ninguna dificultad (a excepción de la interfaz gráfica que puede ocasionar algún problema) la portabilidad de los miles de programas ya existentes para sistemas GNU/Linux. El soporte que le da una gran empresa como es Nokia asegura un futuro muy prometedor a esta plataforma de desarrollo de software.

Al implementar y evaluar el protocolo de descubrimiento de servicios SDM se ha comprobado su correcto funcionamiento y la disponibilidad de poder ser

ya utilizado en aplicaciones y situaciones de la vida real. Cualquier aplicación puede emplear este protocolo para anunciar sus servicios a través de redes MANET. En cuanto a las pruebas realizadas se ha llegado a la conclusión de que la implementación del protocolo SDM no gasta mucha batería de los dispositivos utilizados (Nokia N800), que se produce un consumo de ancho de banda razonable para un protocolo de estas características aunque también depende mucho de los parámetros establecidos (número de servicios anunciados, intervalo de tiempo entre mensajes, número de nodos existentes en la red, etc.) y que el retardo para descubrir si un servicio requerido está en la red o no es casi nulo gracias a la naturaleza proactiva del protocolo SDM.

9.3. Trabajo futuro

Durante la implementación del protocolo de descubrimiento de servicios SDM nuevas ideas han surgido para mejorar y optimizar el protocolo, pero debido a los limitados recursos de tiempo disponibles no han podido llevarse a cabo en este proyecto. Las principales líneas de trabajo para continuar este proyecto son:

- Añadir soporte para IP versión 6 al protocolo SDM, pues actualmente sólo soporta IP versión 4 y cada vez más protocolos soportan ambos tipos de versiones IP hasta que la migración de IP versión 4 a IP versión 6 se complete en un futuro.
- Extender la implementación del protocolo SDM para incluir soporte a la nueva versión del protocolo de routing OLSR (OLSR versión 2). Esto podría ser realizado creando los mensajes SDM como una estructura TLV (*Type-Length-Value*), tal y como se define en el RFC correspondiente [57].
- Definir una manera de describir los servicios en el protocolo SDM, ya que en el diseño del protocolo no se indica nada de cómo deben ser descritos los servicios (valor del campo `Service Description`). Podría ser por ejemplo en texto plano o codificado al igual que el tipo de servicio, intentando diseñarlo de tal manera que contenga bastante información a pesar del poco espacio dedicado a ello (4 bytes), pero que no sea muy complicado de procesar y comprender por los dispositivos. Así se podrían hacer peticiones más específicas y conseguir que la elección que se realice sea la más favorable para los intereses del dispositivo.
- Aprovechar más en profundidad las características y la información que obtiene el protocolo de routing OLSR (almacenada en los repositorios)

para optimizar el funcionamiento del protocolo SDM e intentar ahorrar así consumo en el ancho de banda y disminuir en lo posible los falsos positivos. Por ejemplo se podría aprovechar la información contenida en los repositorios OLSR para descubrir cuándo se ha desconectado un nodo de la red y eliminar así todos los servicios que anuncia dicho nodo de la caché correspondiente (como complemento al método de detección en la desconexión de nodos del protocolo SDM explicado en el apartado 6.3).

- Implementar la posibilidad de que un mismo nodo pueda anunciar dos servicios del mismo tipo pero diferenciándose en su descripción (por ejemplo un nodo anuncia que posee dos impresoras, una en blanco y negro y la otra en color, y dependiendo de lo que requieran el resto de nodos se podrá elegir entre una u otra impresora). Ahora mismo, tal y como está implementado el protocolo SDM, solamente se puede anunciar un servicio por cada tipo ya que no se tiene en cuenta el campo de descripción de servicios. Si en un futuro se utiliza este campo para poder realizar búsquedas más específicas, debería ser posible que un dispositivo pudiera anunciar un número indeterminado de servicios pertenecientes al mismo tipo aunque con descripciones diferentes, mejorando significativamente las posibilidades que ofrece el protocolo SDM.

- Ajustar los intervalos de tiempo configurables por el usuario tanto del protocolo de routing OLSR (`HELLO Interval` y `TC Interval`) como del protocolo de descubrimiento de servicios SDM (`SDM Interval`) para llegar a un acuerdo entre el ancho de banda consumido y la rapidez con la que se descubren nuevos nodos y servicios, teniendo muy en cuenta las características presentes en los diferentes escenarios utilizados.

- Implementar una interfaz gráfica para el descubrimiento de servicios de la red, pudiendo con dicha aplicación aprovechar tanto la información que proporciona el protocolo OLSR como el protocolo SDM para visualizar la topología completa de la red y los servicios que anuncia cada uno de los nodos existentes.

- Extender varias aplicaciones ya existentes para que puedan emplear el protocolo de descubrimiento de servicios SDM (a través de una sencilla interfaz) anunciando los servicios disponibles y solicitando los servicios requeridos, demostrando así el correcto funcionamiento de este protocolo en entornos reales.

9.4. Sugerencias al protocolo SDM

En este apartado se realiza una sugerencia en el diseño del protocolo de descubrimiento de servicios SDM, ya que al realizar su implementación nos dimos cuenta de que es un protocolo demasiado estricto, en el que todos los campos tienen siempre un tamaño fijo, cuando en realidad los servicios que se anuncian en las redes MANET pueden llegar a ser muy diferentes entre sí teniendo que cubrir las necesidades de cada servicio. Esta diferencia se refiere sobre todo a la descripción de los servicios, pues habrá servicios que necesiten búsquedas muy específicas (por ejemplo una impresora puede tener muchas características importantes: si imprime a color o en blanco y negro, a una o doble cara, en qué planta del edificio se encuentra, etc.), mientras que puede que haya otros servicios que no necesiten casi especificaciones.

Por esta razón se sugiere que el campo del mensaje SDM `Service Description` tenga un tamaño variable, así dependiendo del tipo de servicio que se quiera anunciar se introducirá mayor o menor información según el criterio de la aplicación que anuncia el servicio. De esta manera se podrán realizar búsquedas más específicas, introduciendo mayor información que los 4 bytes actuales si así lo requieren, o no introducir ninguna descripción del servicio si no es necesario o para ahorrar ancho de banda si éste es muy limitado en la red MANET desplegada. El tamaño (en bytes) del campo variable de `Service Description` vendría especificado por el campo `Length`, que ocuparía 1 byte e iría inmediatamente después del campo `Service Name`, que en esta ocasión pasaría a ocupar 1 byte como se puede observar en la Figura 9.1.

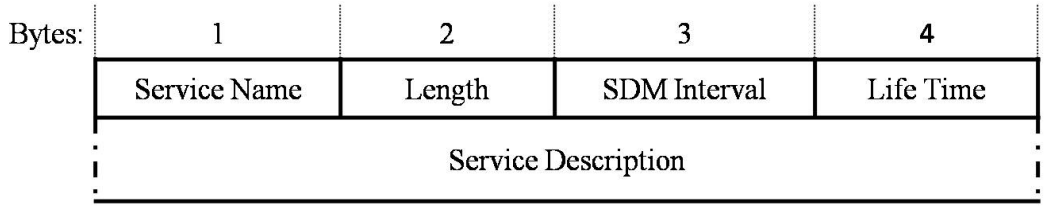

Figura 9.1: Estructura de un mensaje SDM con descripción del servicio variable

Con esta modificación se pretende otorgar cierta flexibilidad a las aplicaciones que anuncian servicios para que según su criterio añadan o no más información que describa el servicio que anuncian. Además se intenta que la descripción del servicio tenga un mayor peso en la elección de los servicios requeridos aunque esto conlleve una complejidad mayor en dicha elección (hay más información a tener en cuenta) pues las posibilidades que ofrece esta manera de elegir servicios son bastante más amplias que teniendo sólo en cuenta

el tipo de servicio. Por esta razón se reduce el número de tipos de servicios (de 65536 servicios a solamente 256) y se añade un campo con el tamaño de la descripción del servicio (hasta un máximo de 255 bytes).

Otorgando al protocolo SDM dicha flexibilidad en el anuncio de los servicios, se podrán realizar anuncios y búsquedas más personalizadas y específicas, atendiendo a las necesidades reales de los otros dispositivos y a las características del entorno en el que esté desplegada la red MANET.

9.5. Conclusiones finales del autor

Este proyecto ha presentado el trabajo que he realizado para el estudio de los protocolos de routing y descubrimiento de servicios existentes en redes MANET, además de implementar el protocolo SDM integrado en el protocolo OLSR en un dispositivo Maemo. Una vez finalizado este proyecto, creo que el descubrimiento de servicios es una de las técnicas más importantes para minimizar la interacción del usuario y así poder crear aplicaciones en las que automáticamente se conecte con los servicios requeridos por el usuario sin que éste tenga que conocer toda la tecnología que existe por debajo de las redes móviles ad-hoc. Gracias a las mejoras en las prestaciones de los dispositivos utilizados en este tipo de redes, cada vez existen más aplicaciones y servicios que ofrecer al usuario, ampliando las posibilidades de uso de forma ilimitada. Esperando a que aparezca alguna aplicación que haga que las redes MANET terminen de despegar de manera comercial y lleguen al usuario final, existen dos áreas en las que se considera imprescindible el despliegue de este tipo de redes debido a la falta de infraestructuras: redes de primeros auxilios en respuesta a situaciones de emergencia y redes tácticas militares para la comunicación entre los integrantes de la contienda.

En mi opinión, para que el despliegue de este tipo de redes llegue a convertirse en algo común entre los usuarios finales todavía hace falta realizar muchas investigaciones en este campo, diseñando nuevos y mejores protocolos tanto de routing como de descubrimiento de servicios que intenten optimizar al máximo la eficiencia en el retardo y sobre todo en el consumo del ancho de banda, debido a los limitados recursos que presentan las redes MANET. Uno de los aspectos que creo que se deberían centrar los investigadores es en la creación de un estándar para la descripción de los diferentes servicios que se pueden anunciar en una red, porque el problema que existe actualmente es que cada protocolo de descubrimiento de servicios define su propia manera de describir servicios, que en la mayoría de los casos no son compatibles entre

sí. Esto provoca una falta de interoperabilidad entre los diferentes protocolos que obliga a que todos los dispositivos utilicen el mismo tipo de protocolo de descubrimiento de servicios, o lo que es peor, que tengan que utilizar varios puesto que no todos los nodos de la red soportan el mismo protocolo, haciendo su uso ineficiente y dando lugar a un mayor consumo de ancho de banda.

La comunidad investigadora debería realizar un esfuerzo para diseñar una manera sencilla y eficiente de describir servicios que fuera implementado por la mayoría de protocolos de descubrimiento de servicios para que éstos puedan, a lo mejor no aprovechar las máximas posibilidades que ofrece el protocolo (si ha creado su propia manera de describir servicios), pero sí comprender el mensaje recibido de una manera básica para que todos los dispositivos sean compatibles entre sí en lo referente a anunciar y solicitar servicios. Si no llega dicho consenso entre los investigadores, otro remedio sería diseñar un algoritmo para que, al igual que el servidor DHCP otorga direcciones IP a los nuevos dispositivos que entran en la red, los nodos existentes en la red se pusieran de acuerdo para elegir qué protocolo de descubrimiento de servicios se va a utilizar, basado principalemente en la elección de aquel protocolo que la mayoría de dispositiivos tenga implementado. De todas maneras es preferible la creación de un estándar, ya que en este segundo método existe la posibilidad de que haya dispositivos que por una razón u otra no tengan implementado este protocolo y se queden por tanto sin poder anunciar ni solicitar servicios al resto de la red MANET.

Con respecto a la plataforma de desarrollo Maemo, creo que aunque ahora mismo no es muy común entre los sistemas de dispositivos móviles (viéndose superado ampliamente por Windows Mobile, Symbian [68] o Android [69]), en un futuro próximo será una plataforma muy a tener en cuenta que ofrecerá muchas posibilidades al usuario final, sobre todo después de la noticia aparecida últimamente en los medios de comunicación [63] de que Nokia va a integrar la plataforma Maemo en sus móviles de alta gama (serie N) a partir del año 2012.

Lo que sí es seguro es que la tecnología wireless cada vez está más presente en nuestras vidas cotidianas, donde las redes centralizadas están migrándose a redes distribuidas donde las posibilidades que te ofrece la movilidad son enormes, por lo que creo que en los próximos años las redes móviles ad-hoc continuarán evolucionando hasta poco a poco irse integrando en el usuario final, apareciendo nuevas aplicaciones que por sus limitados recursos no necesiten de infraestructuras para poderse ejecutar, haciendo de las redes MANET el mejor tipo de red a utilizar en estos casos. Además espero que los protocolos de descubrimiento de servicios tengan un papel crucial en el desarrollo de las futuras redes de comunicaciones, especialmente en las redes MANET.

Parte V

Anexos

Anexo A

Presupuesto

El presupuesto del proyecto se puede dividir en honorarios y equipos. El equipo utilizado en la realización de este proyecto es el siguiente: 1 portátil Dell XPS M1530 (Intel Core 2 Duo a 2.50 GHz, 4 GB de RAM y Windows Vista con Service Pack 1 instalado), 2 terminales Nokia N800 y 2 terminales Nokia 810.

El software utilizado en este proyecto es gratuito, siendo éstas las principales aplicaciones: VMware, para cargar máquinas virtuales, Maemo SDK, para desarrollar aplicaciones Maemo, Eclipse, un entorno de programación con el plugin ESbox (específico para crear aplicaciones Maemo) y Wireshark, una aplicación que analiza el tráfico de la red.

El presupuesto del equipo se observa en el Cuadro A.1.

Concepto	Unidades	Coste (Unidad)	Coste (Total)
Portátil	1	800 €	800 €
Nokia N800	2	380 €	760 €
Nokia N810	2	380 €	760 €
		Total	2320 €

Cuadro A.1: Presupuesto del equipo utilizado en el proyecto

Los honorarios los hemos calculado según los baremos orientativos del Colegio Oficial de Ingenieros de Telecomunicación (COIT). El tiempo invertido en el proyecto fue de aproximadamente seis meses. Este tiempo se ha desglosado en varias tareas, mostrándose en el Cuadro A.2.

Suponiendo que el sueldo de un ingeniero es de aproximadamente 46.000

Concepto	Tiempo (Horas)
Documentación	45
Estudio de Maemo SDK	35
Implementación del protocolo SDM	180
Pruebas de ancho de banda	40
Pruebas de batería y retardo	30
Redacción de la memoria	150
Total	**480**

Cuadro A.2: Horas dedicadas a cada tarea del proyecto

euros brutos anuales y la cantidad de horas trabajadas al año es de 1760, la tarificación de la hora normal de trabajo es por tanto de 26 euros (sin considerar la realización de horas extraordinarias). Con esto, los honorarios de ejecución ascienden a 12480 euros.

A esta cantidad se le debe añadir el trabajo de dirección del tutor, que estimaremos en un 7 % de los honorarios. Además hay que incluir un 16 % de IVA. Los honorarios totales se pueden observar en el Cuadro A.3.

Ejecución	12480.0 €
Dirección	873.5 €
IVA (16 %)	2136.5 €
Total	**15490.0 €**

Cuadro A.3: Parte del presupuesto dedicado a los honorarios

Con lo que el presupuesto total queda reflejado en el Cuadro A.4.

Equipo	2320 €
Honorarios	15490 €
Total	**17810 €**

Cuadro A.4: Presupuesto total del proyecto

El presupuesto total del proyecto asciende a diecisiete mil ochocientos diez euros.

Anexo B

Entorno de desarrollo Maemo

En este anexo se presenta el entorno de desarrollo Maemo y las principales aplicaciones que se utilizan para crear un paquete Debian instalable en el dispositivo Maemo (.deb). En primer lugar se realizará una breve introducción explicando la terminología necesaria para entender el funcionamiento del entorno de desarrollo de software Maemo. Después se explicará los principales pasos para crear un paquete Debian instalable en todos los dispositivos Maemo y por último la ejecución tanto del demonio olsrd como del plugin SDM en el terminal del dispositivo Maemo.

B.1. Introducción

El kernel de los dispositivos móviles Maemo está basado en la arquitectura ARM, diferente de la utilizada en los ordenadores convencionales, que utilizan una arquitectura x86. Por lo tanto para poder crear una aplicación para la plataforma Maemo a partir del código fuente se debe realizar lo que se denomina una compilación cruzada, pues las arquitecturas origen y destino son diferentes. Esto puede llegar a crear problemas al existir la posibilidad de que la ejecución de las aplicaciones Maemo en el ordenador sea diferente a la ejecución en el dispositivo móvil. Aún así, este tipo de problemas se dan en muy contadas ocasiones, pudiendo también emular en el ordenador la arquitectura ARM a través del entorno ARMEL proporcionado por el kit de desarrollo Maemo SDK. De todas maneras, la comprobación definitiva de que la aplicación funciona correctamente debe ser realizada en el propio dispositivo móvil Maemo.

Al estar la plataforma Maemo basada en Linux, el entorno de desarrollo de aplicaciones (*Maemo SDK*) sólo puede ser ejecutado en distribuciones

GNU/Linux tales como Ubuntu, Debian, Fedora, Mandriva, etc. Si algún usuario quiere utilizar dicho entorno de desarrollo en otro sistema operativo como Windows o Mac debe utilizar una máquina virtual que proporcione una imagen de un entorno de trabajo Linux. En nuestro caso, aunque se podía utilizar perfectamente un sistema Linux (Ubuntu [66]) para instalar y ejecutar el entorno de desarrollo Maemo, se ha preferido ejecutar en una máquina virtual (VMware [65]) una imagen con todas las herramientas ya instaladas para la creación, depuración y ejecución de las aplicaciones Maemo [67], ahorrándose así bastante tiempo en instalar dichas herramientas en nuestro sistema Linux. La imagen cargada en la máquina virtual puede verse en la Figura B.1.

Figura B.1: Entorno de desarrollo Maemo SDK

El software de desarrollo de Maemo crea un entorno especial dentro del sistema denominado *Sandbox* donde se testean los programas considerados inestables para que si dichos programas realizan alguna acción imprevista no afecte al resto del sistema pudiéndole ocasionar daños irreversibles. La herramienta que crea este entorno se denomina *Scratchbox*, intentando que el proceso de desarrollo sea muy similar al de cualquier aplicación de un sistema GNU/Linux, haciendo transparente para el programador la compilación cruzada del código fuente. Como *Scratchbox* es prácticamente un sistema GNU/Linux completo, incluye las típicas herramientas de desarrollo para sistemas GNU/Linux como el compilador *gcc*, el depurador *gdb*, el gestor de memoria *valgrind* o el

monitor de llamadas a sistema *ltrace*. Para aquellas aplicaciones que poseen interfaz gráfica, el entorno de desarrollo Maemo utiliza un sistema de ventanas X11 denominado *Xephyr* para dotar de una ventana al sistema donde pueda ejecutarse la interfaz gráfica de la aplicación Maemo.

Aunque es posible utilizar todas estas herramientas por separado (un editor de textos para escribir el código fuente, el entorno *Scratchbox* para compilar dicho código, *Xephyr* para ejecutar la interfaz gráfica de la aplicación, etc.), el entorno de desarrollo de aplicaciones Eclipse [64] posee un plugin denominado ESbox que sirve para aunar en una sola aplicación todas las herramientas necesarias para la creación y depuración de aplicaciones Maemo, ahorrando de esta manera muchísimo tiempo y haciendo el desarrollo de estas aplicaciones más cómodo y sencillo. Una instantánea del programa Eclipse con el plugin ESbox puede verse en la Figura B.2.

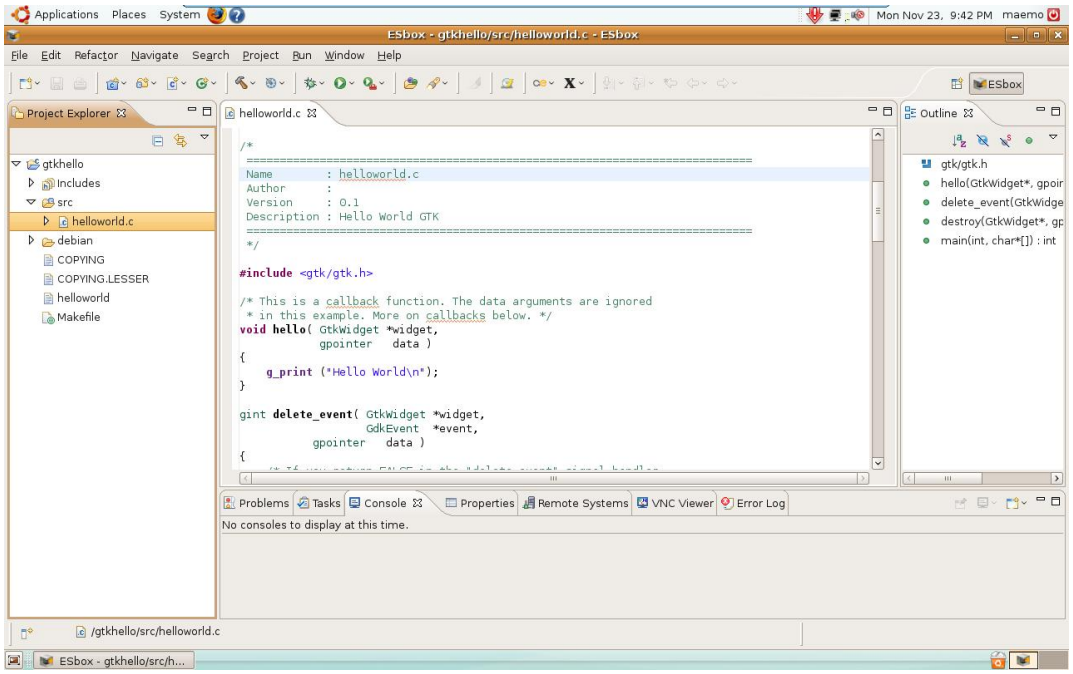

Figura B.2: Entorno de desarrollo Eclipse con plugin ESbox

Este plugin permite muchas formas de ejecutar las aplicaciones tanto en el ordenador como en el dispositivo Maemo para poder depurar el código. Entre las más importantes se encuentran (de menor a mayor parecido con la realidad):

- Ejecución de la aplicación en el ordenador con su arquitectura x86 (como si fuera un programa dirigido al sistema GNU/Linux del ordenador).

- Ejecución de la aplicación en el ordenador con la arquitectura ARM del dispositivo Maemo (a través del emulador ARMEL de dicha arquitectura).

- Ejecución de la aplicación en el dispositivo Maemo a través del cable USB, Wifi o Bluetooth en tiempo real (posibilidad de depurar el código viendo las variables del programa en el entorno Eclipse del ordenador)

- Creación de un paquete Debian (.deb) para su instalación en el dispositivo Maemo, ejecutándose así de forma independiente al ordenador.

B.2. Creación de un paquete Debian (.deb)

A continuación se muestran los pasos a seguir para, a partir del código fuente, crear un paquete Debian (.deb) que pueda ser instalado en el dispositivo Maemo a través del gestor de aplicaciones.

1. Una vez iniciado el entorno de desarrollo Eclipse con el plugin ESbox, se crea un proyecto nuevo (Ir a ***File*** > ***New*** > ***C Maemo Project***).

2. Una vez creado el proyecto, se importan todos los ficheros del código fuente de la aplicación que queremos instalar en el dispositivo Maemo (Con el botón derecho del ratón pulsar el proyecto creado e ir a ***Import*** > ***File System***).

3. El próximo paso es crear la estructura del paquete .deb (Con el botón derecho del ratón pulsar el proyecto creado e ir a ***Debian Package*** > ***Create Debian Structure***).

4. Una vez creada la estructura del paquete .deb hay que escribir toda la información relativa a la aplicación y al autor en el fichero ***control*** dentro de la carpeta ***debian*** en el proyecto creado. Es importante que en el apartado *Section* del fichero ***control*** se cambie el valor *Unknown* por otro, por ejemplo *user/other*, para que el paquete pueda ser instalado correctamente en el dispositivo Maemo.

5. Por último se compila y crea el paquete Debian (.deb) para que pueda ser instalado en el dispositivo Maemo a través del gestor de aplicaciones (Con el botón derecho del ratón pulsar el proyecto creado e ir a ***Debian Package*** > ***Build Debian Package***).

B.3. Ejecución de olsrd y SDM en Maemo

Por último este anexo indicará la forma de ejecutar correctamente el demonio olsrd y el plugin SDM en el dispositivo Maemo. Suponiendo que el paquete Debian (.deb) del demonio olsrd junto con el plugin SDM ha sido instalado correctamente en el dispositivo Maemo, lo primero que hay que hacer es crear una red ad-hoc o conectarse a una ya establecida. Para ello hay que instalar el programa *PC-Connectiviy-Manager* utilizando el repositorio *Maemo extras-devel* y desde el panel de control activar el modo WLAN ad-hoc.

Una vez conectado a una red ad-hoc y antes de ejecutar el demonio olsrd hay que instalar el paquete *rootsh* desde el gestor de aplicaciones para que se pueda acceder como administrador del sistema a través del terminal del dispositivo Maemo, ya que para ejecutar el demonio olsrd necesitas tener permisos de administrador. Una vez instalado el paquete *rootsh*, ejecutamos el terminal del dispositivo Maemo (Pulsas el icono de aplicaciones y vas a ***Utilidades*** > ***Xterm***). En la ventana del terminal tecleamos ***root*** para acceder al sistema como administrador y ejecutamos el demonio olsrd tecleando ***olsrd***. En ese momento el demonio leerá el archivo de configuración almacenado en */etc/olsrd.conf* y cargará los plugins y parámetros establecidos en dicho archivo.

Si queremos utilizar el plugin SDM para anunciar nuestros propios servicios, abrimos una nueva ventana del terminal y tecleamos ***telnet localhost 5555*** para conectarse al plugin SDM a través del puerto 5555 (En este caso no es necesario acceder como administrador del sistema). Los comandos para anunciar, eliminar o mirar el contenido de las cachés de servicios fueron descritos en el apartado 7.10.

Anexo C

Tabla de medidas obtenidas

En este anexo se recogen todos los valores medidos experimentalmente en el laboratorio con los dispositivos Nokia N800 y N810, los relacionados con el ancho de banda consumido y el retardo del protocolo SDM para las diferentes configuraciones de red y diferentes valores de los parámetros más característicos y significativos. A partir de las tablas que se ilustran a continuación se han obtenido los valores finales con los que, analizados convenientemente en el capítulo 8, se han llegado a lograr conclusiones importantes sobre las características, el funcionamiento y el rendimiento del protocolo SDM en dispositivos reales.

C.1. Ancho de banda consumido

En este apartado aparecen los valores obtenidos en la prueba sobre el ancho de banda consumido por el protocolo SDM en función del número de servicios anunciados y del intervalo entre mensajes (`SDM Interval`) que aparece en el apartado 8.6. En las tablas aparecen 3 tandas de medidas para cada configuración de red diferente, siendo el valor final la media de estos 3 valores, obteniendo a su vez la desviación estándar y el intervalo de confianza al 95 %, del que se muestra el extremo inferior (-) y el extremo superior (+) de dicho intervalo.

C.1.1. En función del número de servicios anunciados

Ahora se mostrarán las tablas de valores sobre el ancho de banda consumido por el protocolo SDM en función del número de servicios anunciados (1, 5, 10, 15 y 20 servicios), manteniendo el intervalo de tiempo entre mensajes SDM

constante e igual a 20 segundos y por cada configuración de red posible (1, 2, 3 y 4 nodos).

En el Cuadro C.1 aparecen los valores del ancho de banda consumido cuando sólo hay 1 nodo en la red. En los Cuadros C.2 (por cada nodo) y C.3 (media) cuando hay 2 nodos en la red. En los Cuadros C.4 (por cada nodo) y C.5 (media) cuando hay 3 nodos en la red. Finalmente, en los Cuadros C.6 (por cada nodo) y C.7 (media) cuando hay 4 nodos en la red.

	1 servicio	5 servicios	10 servicios	15 servicios	20 servicios
1ª Medida	237.12	251.20	265.92	284.48	295.36
2ªMedida	233.28	245.65	267.20	281.92	298.35
3ª Medida	234.13	249.07	266.35	280.64	296.64
Media	**234.84**	**248.64**	**266.49**	**282.35**	**296.78**
Desviación estándar	**2.02**	**2.80**	**0.65**	**1.96**	**1.50**
Intervalo de Confianza (-)	**232,56**	**245,47**	**265,75**	**280,13**	**295,09**
Intervalo de Confianza (+)	**237,13**	**251,81**	**267,23**	**284,56**	**298,48**

Cuadro C.1: Ancho de banda consumido con 1 nodo en la red

		1 servicio	5 servicios	10 servicios	15 servicios	20 servicios
1ª Medida	Nodo 1	351.89	370.24	384.21	401.17	413.44
	Nodo 2	354.45	381.55	394.03	400.53	418.35
2ª Medida	Nodo 1	356.80	373.76	383.89	405.65	419.41
	Nodo 2	351.47	370.45	388.16	397.65	415.89
3ª Medida	Nodo 1	359.36	367.79	386.56	408.32	420.16
	Nodo 2	360.00	368.85	387.41	413.55	420.27

Cuadro C.2: Ancho de banda consumido por cada nodo (Red de 2 nodos)

C.1.2. En función del intervalo entre mensajes

Ahora se mostrarán las tablas de valores sobre el ancho de banda consumido por el protocolo SDM en función del intervalo de tiempo entre mensajes SDM

	1 servicio	5 servicios	10 servicios	15 servicios	20 servicios
1ª Medida	944.585	404.96	375.90	361.81	355.47
2ªMedida	940.745	403.36	372.11	363.79	355.31
3ª Medida	944.375	397.335	368.32	362.35	355.47
Media	**943.24**	**401.89**	**372.11**	**362.65**	**355.41**
Desviación estándar	**2.16**	**4.02**	**3.79**	**1.02**	**0.09**
Intervalo de Confianza (-)	**940.79**	**397.34**	**367.82**	**361.49**	**355.31**
Intervalo de Confianza (+)	**945.68**	**406.43**	**376.39**	**363.80**	**355.51**

Cuadro C.3: Ancho de banda medio consumido con 2 nodos en la red

		1 servicio	5 servicios	10 servicios	15 servicios	20 servicios
1ª Medida	**Nodo 1**	363.52	368.00	377.92	400.21	413.55
	Nodo 2	549.76	593.17	624.43	697.92	702.40
	Nodo 3	355.95	364.69	380.91	395.63	415.25
2ª Medida	**Nodo 1**	360.43	368.64	383.89	403.31	419.31
	Nodo 2	568.85	586.13	633.17	682.99	729.71
	Nodo 3	358.61	369.71	387.20	400.75	424.32
3ª Medida	**Nodo 1**	357.12	370.56	391.57	395.63	424.43
	Nodo 2	564.05	583.89	631.25	682.56	725.33
	Nodo 3	358.61	371.95	373.87	406.08	413.76

Cuadro C.4: Ancho de banda consumido por cada nodo (Red de 3 nodos)

	1 servicio	5 servicios	10 servicios	15 servicios	20 servicios
1ª Medida	423.08	441.95	461.09	497.92	510.40
2ªMedida	429.30	441.49	468.09	495.68	524.45
3ª Medida	426.59	442.13	465.56	494.76	521.17
Media	**426.32**	**441.86**	**464.91**	**496.12**	**518.67**
Desviación estándar	**3.12**	**0.33**	**3.55**	**1.63**	**7.35**
Intervalo de Confianza (-)	**422.79**	**441.49**	**460.90**	**494.28**	**510.36**
Intervalo de Confianza (+)	**429.85**	**442.23**	**468.92**	**497.96**	**526.99**

Cuadro C.5: Ancho de banda medio consumido con 3 nodos en la red

		1 servicio	5 servicios	10 servicios	15 servicios	20 servicios
1ª Medida	**Nodo 1**	363.52	362.13	386.67	397.44	422.61
	Nodo 2	651.52	687.57	766.08	806.83	882.99
	Nodo 3	637.01	687.47	759.68	832.11	884.48
	Nodo 4	354.45	367.15	388.48	404.37	424.21
2ª Medida	**Nodo 1**	350.08	369.39	395.20	394.45	420.05
	Nodo 2	642.67	694.19	766.51	821.33	886.08
	Nodo 3	639.57	687.36	752.43	822.61	850.88
	Nodo 4	351.79	371.63	384.96	406.72	414.19
3ª Medida	**Nodo 1**	356.59	370.67	386.56	399.68	419.95
	Nodo 2	650.88	698.45	763.31	819.41	887.04
	Nodo 3	642.77	696.11	753.71	812.59	901.87
	Nodo 4	353.49	369.60	386.45	399.79	416.32

Cuadro C.6: Ancho de banda consumido por cada nodo (Red de 4 nodos)

	1 servicio	5 servicios	10 servicios	15 servicios	20 servicios
1ª Medida	501.63	526.08	575.23	610.19	653.57
2ªMedida	496.03	530.64	574.78	611.28	642.80
3ª Medida	500.93	533.71	572.51	607.87	656.30
Media	**499.53**	**530.14**	**574.17**	**609.78**	**650.89**
Desviación estándar	**3.05**	**3.84**	**1.46**	**1.74**	**7.14**
Intervalo de Confianza (-)	**496.08**	**525.80**	**572.52**	**607.81**	**642.81**
Intervalo de Confianza (+)	**502.98**	**534.49**	**575.82**	**611.75**	**658.96**

Cuadro C.7: Ancho de banda medio consumido con 4 nodos en la red

(1, 10, 20, 30 y 40 segundos), manteniendo el número de servicios anunciados por dispositivo constante e igual a servicios y por cada configuración de red posible (1, 2, 3 y 4 nodos).

En el Cuadro C.8 aparecen los valores del ancho de banda consumido cuando sólo hay 1 nodo en la red. En los Cuadros C.9 (por cada nodo) y C.10 (media) cuando hay 2 nodos en la red. En los Cuadros C.11 (por cada nodo) y C.12 (media) cuando hay 3 nodos en la red. Finalmente, en los Cuadros C.13 (por cada nodo) y C.14 (media) cuando hay 4 nodos en la red.

	1 seg.	10 seg.	20 seg.	30 seg.	40 seg.
1ª Medida	860.80	280.96	251.20	239.57	233.28
2ªMedida	861.23	281.39	245.65	240.00	235.52
3ª Medida	862.93	283.52	249.07	240.00	235.52
Media	**861.65**	**281.96**	**248.64**	**239.86**	**234.77**
Desviación estándar	**1.13**	**1.37**	**2.80**	**0.25**	**1.29**
Intervalo de Confianza (-)	**860.38**	**280.41**	**245.47**	**239.58**	**233.31**
Intervalo de Confianza (+)	**862.93**	**283.51**	**251.81**	**240.14**	**236.24**

Cuadro C.8: Ancho de banda consumido con 1 nodo en la red

		1 seg.	10 seg.	20 seg.	30 seg.	40 seg.
1ª Medida	Nodo 1	948.69	412.37	370.24	364.37	356.37
	Nodo 2	940.48	397.55	381.55	359.25	354.56
2ª Medida	Nodo 1	944.53	406.93	373.76	364.16	353.39
	Nodo 2	936.96	399.79	370.45	363.41	357.23
3ª Medida	Nodo 1	947.95	394.35	367.79	364.80	351.04
	Nodo 2	940.80	400.32	368.85	359.89	359.89

Cuadro C.9: Ancho de banda consumido por cada nodo (Red de 2 nodos)

	1 seg.	10 seg.	20 seg.	30 seg.	40 seg.
1ª Medida	944.585	404.96	375.90	361.81	355.47
2ªMedida	940.745	403.36	372.11	363.79	355.31
3ª Medida	944.375	397.335	368.32	362.35	355.47
Media	**943.24**	**401.89**	**372.11**	**362.65**	**355.41**
Desviación estándar	**2.16**	**4.02**	**3.79**	**1.02**	**0.09**
Intervalo de Confianza (-)	**940.79**	**397.34**	**367.82**	**361.49**	**355.31**
Intervalo de Confianza (+)	**945.68**	**406.43**	**376.39**	**363.80**	**355.51**

Cuadro C.10: Ancho de banda medio consumido con 2 nodos en la red

		1 seg.	10 seg.	20 seg.	30 seg.	40 seg.
1ª Medida	Nodo 1	947.95	398.83	368.00	362.13	355.09
	Nodo 2	1868.91	669.01	593.17	559.57	556.80
	Nodo 3	950.72	402.56	364.69	362.67	358.61
2ª Medida	Nodo 1	943.47	394.56	368.64	362.67	355.52
	Nodo 2	1871.68	663.15	586.13	568.96	556.05
	Nodo 3	942.83	399.89	369.71	357.65	352.64
3ª Medida	Nodo 1	951.15	403.20	370.56	359.57	353.49
	Nodo 2	1832.53	668.48	583.89	560.53	546.99
	Nodo 3	953.17	409.81	371.95	358.19	352.53

Cuadro C.11: Ancho de banda consumido por cada nodo (Red de 3 nodos)

	1 seg.	10 seg.	20 seg.	30 seg.	40 seg.
1ª Medida	1255.86	490.13	441.95	428.12	423.50
2ªMedida	1252.66	485.87	441.49	429.76	421.40
3ª Medida	1245.62	493.83	442.13	426.10	417.67
Media	**1251.38**	**489.94**	**441.86**	**427.99**	**420.86**
Desviación estándar	**5.24**	**3.99**	**0.33**	**1.84**	**2.95**
Intervalo de Confianza (-)	**1245.45**	**485.43**	**441.49**	**425.92**	**417.52**
Intervalo de Confianza (+)	**1257.31**	**494.45**	**442.23**	**430.07**	**424.20**

Cuadro C.12: Ancho de banda medio consumido con 3 nodos en la red

		1 seg.	10 seg.	20 seg.	30 seg.	40 seg.
1ª Medida	Nodo 1	957.33	395.84	362.13	364.27	363.41
	Nodo 2	2341.44	780.80	687.57	667.52	652.80
	Nodo 3	2373.23	778.45	687.47	675.41	653.23
	Nodo 4	937.60	403.95	367.15	360.11	355.84
2ª Medida	Nodo 1	928.96	401.39	369.39	366.93	354.35
	Nodo 2	2342.83	793.49	694.19	671.36	640.32
	Nodo 3	2345.39	766.51	687.36	664.96	647.47
	Nodo 4	948.91	405.01	371.63	364.16	352.96
3ª Medida	Nodo 1	943.36	400.85	370.67	355.31	357.55
	Nodo 2	2337.28	804.27	698.45	659.52	655.15
	Nodo 3	2328.96	769.07	696.11	665.81	649.81
	Nodo 4	929.28	398.40	369.60	361.71	358.19

Cuadro C.13: Ancho de banda consumido por cada nodo (Red de 4 nodos)

	1 seg.	10 seg.	20 seg.	30 seg.	40 seg.
1ª Medida	1652.40	589.76	526.08	516.83	506.32
2ªMedida	1641.52	591.60	530.64	516.85	498.78
3ª Medida	1634.72	593.15	533.71	510.59	505.18
Media	**1642.88**	**591.50**	**530.14**	**514.76**	**503.42**
Desviación estándar	**8.92**	**1.70**	**3.84**	**3.61**	**4.07**
Intervalo de Confianza (-)	**1632.79**	**589.58**	**525.80**	**510.67**	**498.82**
Intervalo de Confianza (+)	**1652.97**	**593.42**	**534.49**	**518.84**	**508.02**

Cuadro C.14: Ancho de banda medio consumido con 4 nodos en la red

C.1.3. En función del número de servicios solicitados por minuto (protocolo Mercury)

A continuación se mostrarán las tablas de valores sobre el ancho de banda consumido por el protocolo Mercury en función del número de servicios solicitados por minuto (1, 5, 10 y 15 servicios) para una configuración de red de 4 nodos.

En los Cuadros C.15 (por cada nodo) y C.16 (media) aparecen los valores del ancho de banda consumido por Mercury para una configuración de red de 4 nodos.

		1 servicio	5 servicios	10 servicios	15 servicios
1ª Medida	Nodo 1	359.20	385.33	381.33	400.00
	Nodo 2	620.80	631.47	660.80	697.07
	Nodo 3	604.27	636.27	671.47	693.33
	Nodo 4	353.07	387.47	408.53	443.47
2ª Medida	Nodo 1	363.47	373.07	402.13	403.73
	Nodo 2	612.27	635.20	678.93	708.27
	Nodo 3	617.07	666.67	691.20	716.27
	Nodo 4	362.40	379.47	385.87	436.80
3ª Medida	Nodo 1	349.07	378.13	389.60	402.13
	Nodo 2	599.47	643.73	662.40	704.53
	Nodo 3	612.27	640.53	682.13	701.33
	Nodo 4	353.33	374.67	398.40	434.93

Cuadro C.15: Ancho de banda consumido por cada nodo (Mercury)

C.2. Retardo

En este apartado aparecen los valores obtenidos en la prueba sobre el retardo producido al anunciar y notificar algún servicio eliminado por el protocolo SDM entre los extremos de redes compuestos por 2, 3 y 4 nodos que se muestra en la sección 8.7. También aparecen los valores del retardo que se produce al solicitar un servicio con el protocolo Mercury y recibir la respuesta a dicha solicitud, documentado en el apartado 8.8.4.

	1 servicio	5 servicios	10 servicios	15 servicios
1ª Medida	484.34	510.14	530.53	558.47
2ªMedida	488.80	513.60	539.53	566.27
3ª Medida	478.54	509.27	533.13	560.73
Media	**483.89**	**511.00**	**534.40**	**561.82**
Desviación estándar	**5.15**	**2.29**	**4.63**	**4.01**
Intervalo de Confianza (-)	**478.07**	**508.40**	**529.16**	**557.28**
Intervalo de Confianza (+)	**489.72**	**513.60**	**539.64**	**566.36**

Cuadro C.16: Ancho de banda medio consumido por Mercury

En las tablas aparecen 5 tandas de medidas para cada configuración de red diferente, siendo el valor final la media de estos 5 valores, obteniendo a su vez la desviación estándar y el intervalo de confianza al 95 %, del que se muestra el extremo inferior (-) y el extremo superior (+) de dicho intervalo.

C.2.1. Anuncio de servicios en SDM

En el Cuadro C.17 aparecen los valores sobre el el retardo producido al anunciar servicios para una configuración de red de 2, 3 y 4 nodos.

	2 Nodos	3 Nodos	4 Nodos
1ª Medida	0.0105	0.9971	1.4802
2ªMedida	0.0358	1.1088	1.8553
3ª Medida	0.0630	1.0168	1.5508
4ª Medida	0.0369	0.7985	1.9812
5ª Medida	0.0324	1.0096	1.8431
Media	**0.0357**	**0.9862**	**1.7421**
Desviación estándar	**0.0187**	**0.1139**	**0.2153**
Intervalo de Confianza (-)	**0.0194**	**0.8864**	**1.5534**
Intervalo de Confianza (+)	**0.0521**	**1.0860**	**1.9308**

Cuadro C.17: Retardo producido en el anuncio de servicios

C.2.2. Eliminación de servicios en SDM

En el Cuadro C.18 aparecen los valores sobre el el retardo producido al notificar la eliminación de algún servicio para una configuración de red de 2, 3 y 4 nodos.

	2 Nodos	3 Nodos	4 Nodos
1ª Medida	0.0356	0.6950	1.1301
2ªMedida	0.0315	0.8429	1.8897
3ª Medida	0.0361	1.1146	1.5751
4ª Medida	0.0353	0.9198	1.2647
5ª Medida	0.0336	0.6809	1.1468
Media	**0.0344**	**0.8506**	**1.4013**
Desviación estándar	**0.0019**	**0.1786**	**0.3262**
Intervalo de Confianza (-)	**0.0328**	**0.6941**	**1.1153**
Intervalo de Confianza (+)	**0.0361**	**1.0072**	**1.6872**

Cuadro C.18: Retardo producido en la notificación de eliminación de un servicio

C.2.3. Petición de servicios en Mercury

En el Cuadro C.19 aparecen los valores sobre el el retardo que tarda en enviarse una solicitud de un servicio y en recibir una respuesta en el protocolo Mercury para una configuración de red de 2, 3 y 4 nodos.

	2 Nodos	3 Nodos	4 Nodos
1ª Medida	0.1084	1.7653	2.7071
2ªMedida	0.1154	2.1091	2.7349
3ª Medida	0.2183	1.4028	2.5637
4ª Medida	0.1095	1.7707	2.9705
5ª Medida	0.1091	1.6169	2.7905
Media	**0.1321**	**1.7330**	**2.7533**
Desviación estándar	**0.0482**	**0.2581**	**0.1475**
Intervalo de Confianza (-)	**0.0899**	**1.5067**	**2.6241**
Intervalo de Confianza (+)	**0.1744**	**1.9592**	**2.8826**

Cuadro C.19: Retardo producido en la petición de servicios (Mercury)

Bibliografía

[1] T.Clausen, P.Jacquet. *RFC 3626: Optimized Link State Routing Protocol (OLSR)*. Proyect Hipercom. INRIA. October 2003.

[2] Maemo: Home of the Maemo community. *http://maemo.org/*. Accedido en 2009.

[3] L. Viennot. *Complexity results on election of multipoint relays in wireless networks*. Project Hipercom, Report RR-3584. INRIA, 1998.

[4] A. Qayyum et L. Viennot A. Laouiti. *Multipoint relaying: An efficient technique for flooding in mobile wireless networks*. Technical report. INRIA. 2002.

[5] N. Enneya, A. Baayer, and M. Elkoutbi. *A New Criterion for MPR Selection in the OLSR Protocol*. Proceedings of the 2007 Euro American conference on Telematics and information systems. 2007.

[6] olsrd, an ad-hoc wireless mesh routing daemon. *http://www.olsr.org/*

[7] S. Corson y J. Macker. *RFC2501: Mobile Ad hoc Networking (MANET)*. January 1999.

[8] Official HSPA website. *http://hspa.gsmworld.com/*. Accedido en 2009.

[9] P. Brenner. *A technical Tutorial on the IEEE 802.11 Protocol*. BreezeCOM. 2007.

[10] Bluetooth v3.0. *Bluetooth core system architecture and specification v3.0*. http://www.bluetooth.com/. Accedido en 2009.

[11] C. Eklund, R. Marks, K. Stanwood, S. Wang. *IEEE Standard 802.16: A technical Overview of the WirelessMAN*. IEEE Communications Magazine, pages 98-107. June 2002.

[12] *Long Term Evolution(LTE): A technical overview*. Motorola Technical White Paper. 2007.

[13] IEEE Standard 802.11. *Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications*. IEEE Computer Society. 2007.

[14] C. Hornig. *RFC 894: A standard for the Transmission of IP Datagrams over Ethernet Networks*. April 1984

[15] P. Mockapetris. *RFC 1034: Domain Names - Concepts and Facilities*. November 1987

[16] J. Rosenberg, H. Schulzrinne, G. Camarillo, A. Johnston, J. Peterson, R. Sparks, M. Handley, E. Schooler. *RFC 3261 - SIP: Session Initiation Protocol*. June 2002

[17] R.Droms. *RFC 2131: Dynamic Host Configuration Protocol*. March 1997

[18] P. Hu, P. Hong, J. Li. *Name Resolution in On-demand MANET*. IEEE International Conference on Wireless and Mobile Computing, Networking And Communications 2005. Volume 3 Pages 462-466.

[19] N. Banerjee, A. Acharya, and S. Das. *Enabling SIP-based sessions in ad hoc networks*. Wireless Networks, Volume 13 Issue 4 Pages 461-479. August 2007.

[20] A.Tønnesen, A. Hafslund, P. Engelstad. *IP Address Autoconfiguration for Proactive Mobile Ad Hoc Networks*. Proceedings of IEEE International Conference on Communication (ICC'2004). June 2004.

[21] G. Malkin. *RFC 2435: RIP Version 2*. November 1998

[22] J. Moy. *RFC 2328: OSPF Version 2*. April 1998

[23] D. Welch, S. Lathrop. *Wireless Security Threat Taxonomy*. Proceedings of the 2003 IEEE Workshop on Information Assurance. Pages 76-83. June 2003.

[24] C. Perkins, E. Belding-Royer, S. Das. *RFC 3561: Ad hoc On-Demand Distance Vector (AODV) Routing*. July 2003.

[25] A.Tønnesen. *Implementing and extending the Optimized Link State Routing Protocol*. Master Thesis, Department of Informatics. University of Oslo. August 2004.

[26] D. Johnson, Y. Hu, D. Maltz. *RFC 4728: The Dynamic Source Routing Protocol (DSR) for Mobile Ad Hoc Networks for IPv4*. February 2007.

[27] I. Chakeres, C. Perkins. *Dynamic MANET On-demand (DYMO) Routing*. IETF Internet-Draft Version 17. March 2009.

[28] R. Ogier, F. Templin, M. Lewis. *RFC 3684: Topology Dissemination Based on Reverse-Path Forwarding (TBRPF)*. February 2004.

[29] M. Gerla, X. Hong, G. Pei. *Fisheye State Routing Protocol (FSR) for Ad Hoc Networks*. IETF Internet-Draft Version 3. June 2002.

[30] Z. Haas, M. Pearlman, P. Samar. *The Zone Routing Protocol (ZRP) for Ad Hoc Networks*. IETF Internet-Draft Version 4. July 2002.

[31] E.Ermel, P. Mühlethaler. *Using OLSR Multipoint Relays (MPRs) to estimate node positions in a Wireless Mesh Network*. Research Report. INRIA. December 2006.

[32] Debian: The universal operating system. *http://www.debian.org/*. Accedido en 2009.

[33] GNOME: The Free Software Desktop Project. *http://www.gnome.org/*. Accedido en 2009.

[34] ARM: The Architecture for the Digital World. *http://www.arm.com/*. Accedido en 2009.

[35] E. Gryazin. *Service Discovery in Bluetooth*. Helsinki University of Technology. 2003

[36] Universidad de Murcia. UM-OLSR. *http://masimum.inf.um.es/*. Accedido en 2009.

[37] INRIA. OOLSR. *http://hipercom.inria.fr/OOLSR/*. Accedido en 2009.

[38] University of California and Cornell University. ns-2 Network Simulator. *http://www.isi.edu.nsnam/ns/*. Accedido en 2009.

[39] Naval Research Laboratory. NRL-OLSR. *http://cs.itd.nrl.navy.mil/*. Accedido en 2009.

[40] A. Tønnesen, A. Hafslund, Ø. Kure. *The Unik-OLSR Plugin Library*. In Proceedings of the The first OLSR Interop and Workshop, 2004.

[41] E. Guttman, C. Perkins, J. Veizades, M. Day. *RFC 2608: Service Location Protocol, Version 2*. June 1999.

[42] Y. Goland, T. Cai, P. Leach, Y. Gu. *Simple service discovery protocol/1.0*. IETF Internet-Draft Version 3. October 1999.

[43] S. Cheshire, M. Krochmal. *DNS-Based Service Discovery*. IETF Internet-Draft Version 5. September 2008.

[44] J.Waldo. *Jini Architecture Overview*. Sun Microsystems. 1998.

[45] H. Zimmermann, *OSI Reference Model - The ISO Model of Architecture for Open Systems Interconnection*. IEEE Transactions on Communications. Volume 28, no. 4, Pages 425 - 432. April 1980.

[46] J. A. García-Macías, D. Arias. *Service Discovery in Mobile Ad hoc Networks: Better at the Network Layer?*. Proc. IEEE Intl. Workshop on Wireless and Sensor Networks (WSNET'05). Pages 452-457. Oslo, Norway. June 2005.

[47] C. Campo, C. García-Rubio, A. Marín, F. Almenárez. *PDP: a lightweight discovery protocol for local-scope interactions in wireless ad hoc networks*. Computer Networks. Pages 3264-3283. December 2006.

[48] S. Helal, N. Desai, V. Verma, C. Lee. *Konark - a service discovery and delivery protocol for ad-hoc networks*. Proceedings of the Third IEEE Conference on Wireless Communication Networks (WCNC). New Orleans. 2003.

[49] M. Abou El Saoud, T. Kunz, S. Mahmoud. *SLPManet: service location protocol for MANET*. Proceeding of the 2006 international conference on Communications and mobile computing (IWCMC '06). Pages 701-706. New York (USA). 2006.

[50] A. Obaid, A. Khir, H. Mili. *A Routing Based Service Discovery Protocol for Ad hoc Networks*. In Proceedings of the Third International Conference on Networking and Services (ICNS '07). 2007.

[51] P. Engelstad, Y. Zheng, R. Koodli, C. Perkins. *Service Discovery Architectures for On-Demand Ad Hoc Networks*. International Journal of Ad Hoc and Sensor Wireless Networks. Volume 2, Number 1, Pages 27-58. March 2006.

[52] J. L. Jodra, M. Vara, J. M. Cabero, J. Bagazgoitia. *Service discovery mechanism over OLSR for mobile ad-hoc networks*. Advanced Information Networking and Applications (AINA). Pages 534-542. 2006.

[53] J. Flathagen. *Service Discovery in Mobile Ad-hoc Networks*. Master Thesis. Department of Informatics. University of Oslo. November 2008.

[54] C. Ververidis, G. Polyzos. *Extended ZRP: Performance Evaluation of a Routing Layer Based Service Discovery Protocol for Mobile Ad Hoc Networks*. Proc. 2nd Annual International Conference on Mobile and Ubiquitous Systems: Networking and Services (MobiQuitous '05). San Diego, California. Pages 114-123. July 2005.

[55] J. Su, W. Guo. *A Survey of Service Discovery Protocols for Mobile Ad Hoc Networks*. International Conference on Communications, Circuits and Systems (ICCCAS '08). Pages 398-404. May 2008.

[56] T. Clausen, C. Dearlove, P. Jacquet. *The Optimized Link State Routing Protocol version 2*. IETF Internet-Draft Version 9. July 2009.

[57] T. Clausen, C. Dearlove, J. Dean, C. Adjih. *RFC 5444: Generalized Mobile Ad Hoc Network (MANET) Packet/Message Format*. February 2009.

[58] T. Clausen, C. Dearlove, J. Dean. *MANET Neighborhood Discovery Protocol (NHDP)*. IETF Internet-Draft Version 10. July 2009.

[59] R. Shannon, J. D. Johannes. *Systems simulation: the art and science*. IEEE Transactions on Systems, Man and Cybernetics. Vol. 6. pp. 723-724. 1976

[60] OMNeT++ Community. *http://www.omnetpp.org/*. Accedido en 2009.

[61] K. Plyashkevich. Advanced-Power for Maemo. *https://garage.maemo.org/projects/advanced-power/*. Accedido en 2009.

[62] D. De Couto. *High-Throughput Routing for Multi-Hop Wireless Networks*. Massachusetts Institute of Technology (MIT). April 2004.

[63] ElMundo.es. *Nokia cambiará Symbian por Linux en la serie N a partir de 2012*. 19 de Noviembre de 2009. Accedido en 2009. *http://www.elmundo.es/elmundo/2009/11/19/navegante/1258631774.html*.

[64] Eclipse.org home. *http://www.eclipse.org/*. Accedido en 2009.

[65] VMware: Business Infrastructure Virtualization. Accedido en 2009. *http://www.vmware.com/*.

[66] Ubuntu Home Page. *http://www.ubuntu.com/*. Accedido en 2009.

[67] Maemo SDK Virtual Image. *http://maemovmware.garage.maemo.org/*. Accedido en 2009.

[68] The Symbian Foundation Community Home. *http://www.symbian.org/*. Accedido en 2009.

[69] Android - Meet Android 2.0. *http://www.android.com/*. Accedido en 2009.

[70] Forum Nokia. *Nokia Energy Profiler*. *http://www.forum.nokia.com/Tools_Docs_and_Code/Tools/Plug-ins/Enablers/Nokia_Energy_Profiler/*. Accedido en 2009.

[71] E. García, C. Campo. *Energy consumption of 802.11 on mobile phones at application level*. Trabajo Fin de Máster en Ingeniería Telemática - Universidad Carlos III de Madrid. Septiembre de 2008.

[72] L. M. Feeney, M. Nilsson. *Investigating the Energy Consumption of a Wireless Network Interface in an Ad Hoc Networking Environment*. 2001.