
# TREBALL DE FI DE CARRERA

**TÍTOL DEL TFC: Redes celulares de 4G basadas en IPv6: transporte de paging con multicast**

**TITULACIÓ: Enginyeria Tècnica de Telecomunicació, especialitat Telemàtica**

**AUTORS: Fernando López Muñoz**
**Jessica Reyes Barbancho**

**DIRECTOR: Rafael Vidal Ferré**

**DATA: 21 de gener d 2005**

**Títol: Redes celulares** de 4G basadas en IPv6: transporte de paging con multicast

**AUTORS:** Fernando López Muñoz
Jessica Reyes Barbancho

**DIRECTOR:** Rafael Vidal Ferré

**DATA:** 21 de gener de 2005

**Resum**

Todas las redes celulares, GSM, GPRS, UMTS, etc. ofrecen las funciones de traspaso, localización y paging. Entendemos por traspaso el cambio de punto de acceso del terminal móvil sin perdida de las comunicaciones. La localización en cambio es el proceso en el que el nodo móvil informa a la red de su posición para poder recibir información (un SMS, una llamada,..). Cuando se da este caso, hace falta avisar al nodo móvil a partir de la información obtenida en el proceso de localización. Esto es lo que se conoce como paging.

Ante la llegada de la cuarta generación de redes celulares (4G) IP se está intentando que actúe como elemento concentrador de las diferentes tecnologías radio. IP se encargará de nuevas funciones entre ellas la de gestionar la movilidad, para ello el IETF ha empezado por el protocolo Mobile IP. Sin embargo este protocolo no implementa todas las funcionalidades necesarias, como es el paging y a día de hoy se desconoce cómo se le añadirá.

La elección de IPv6 viene determinada por el conjunto de mejoras que supone respecto a la versión actual de IP (IPv4) en temas como el direccionamiento, seguridad o soporte a la movilidad. Remarcar que definitivamente, IPv6 pasará a utilizarse en redes comerciales en un futuro muy próximo como demuestra el compromiso del Departamento de Defensa de los EEUU para migar a IPv6 y que el 3GPP lo incluya como obligatorio en próximas versiones de la red UMTS.

Como fruto del trabajo realizado en un proyecto anterior [1] se dispone de una primera implementación del protocolo Geopaging [4]. Se trata de un protocolo multicast especialmente diseñado para realizar un transporte eficiente de los mensajes de paging sobre una red celular basada en IPv6. Esta implementación de Geopaging se conoce como Mcast.

A partir de esta primera implementación se ha mejorado el código para posteriormente analizar su rendimiento en un escenario que simula un ejemplo de red real. Además de este análisis se ha llegado a implementar una arquitectura de paging para unirla con Mcast. Finalmente este trabajo une Mcast y la arquitectura de paging realizando diferentes pruebas para comprobar su funcionamiento.

**Title:** Redes celulares de 4G basadas en IPv6: transporte de paging con multicast

**Authors:** Fernando López Muñoz
Jessica Reyes Barbancho

**Director:** Rafael Vidal Ferré

**Date:** January, 21th 2005

**Overview**

Nowadays all cellular networks, GSM, GPRS, UMTS, etc. provide handover, location and paging functions. A handover is a change of access point by a mobile terminal, without loss of connection. In the other hand, location is the process where mobile node informs the networks about its position, so it could receive information (SMS, calls…). In this case, mobile node must be notified using the information received in the location process.

Focusing in the fourth cellular network generation (4G) IP will eventually act as a concentrator element in the different radio technologies. IP is going to work in new functions as a mobility management. This is why IETF has developed the standard protocol Mobile IP. However, Mobile IP does not implement all the needed functions like paging. By now, is unknown how to add paging functions to Mobile IP.

The choice of IPv6 has been determined by the amount of improvement that supposes this version respect IPV4, in topics like addressing, security or mobility support. IPv6 will widely used in commercials networks in a closed future like is demonstrated by the agreement of the USA Defense Department, to migrate IPv6.

This project comes from another one made before [1] where was first implemented the Geopaging protocol [4]. That refers a multicast protocol specially developed to realize an efficient transport of paging messages over a cellular network based to IPv6. This implementation is known as Mcast.

This implementation improves Mcast code. After that, we analyze its performance in a scenario that represents an example of a real network. Furthermore, we have achieved an implementation of paging architecture. Finally this project links Mcast and the paging architecture, making different tests to check the correctness of their implementation

# DEDICATORIA

Este proyecto está dedicado a todas aquellas personas que nos han apoyado y animado en su realización. Especialmente a nuestro director Rafael Vidal, quien confió en nosotros para su desarrollo.También no olvidamos a Marcos García quien fue el pionero del proyecto.
Finalmente, agradecer a nuestras familias el apoyarnos y estar ahí en las horas bajas.

ÍNDICE

# INTRODUCCIÓN

Todas las redes celulares, GSM, GPRS, UMTS, etc. ofrecen las funciones de traspaso, localización y paging. Entendemos por traspaso el cambio de punto de acceso del terminal móvil sin perdida de las comunicaciones. La localización en cambio es el proceso en el que el nodo móvil informa a la red de su posición para poder recibir información (un SMS, una llamada,..). Cuando se da este caso, hace falta avisar al nodo móvil a partir de la información obtenida en el proceso de localización. Esto es lo que se conoce como paging.

Ante la llegada de la cuarta generación de redes celulares (4G) IP se está intentando que actúe como elemento concentrador de las diferentes tecnologías radio. IP se encargará de nuevas funciones entre ellas la de gestionar la movilidad, para ello el IETF ha empezado por el protocolo Mobile IP. Sin embargo este protocolo no implementa todas las funcionalidades necesarias, como es el paging y a día de hoy se desconoce cómo se le añadirá.

La elección de IPv6 viene determinada por el conjunto de mejoras que supone respecto a la versión actual de IP (IPv4) en temas como el direccionamiento, seguridad o soporte a la movilidad. Remarcar que definitivamente, IPv6 pasará a utilizarse en redes comerciales en un futuro muy próximo como demuestra el compromiso del Departamento de Defensa de los EEUU para migar a IPv6 y que el 3GPP lo incluya como obligatorio en próximas versiones de la red UMTS.

Como fruto del trabajo realizado en un proyecto anterior [1] se dispone de una primera implementación del protocolo Geopaging [4]. Se trata de un protocolo multicast especialmente diseñado para realizar un transporte eficiente de los mensajes de paging sobre una red celular basada en IPv6.

En el presente proyecto, una primera fase del trabajo girará en torno a la posible mejora de esta implementación así como a la calibración de su rendimiento. A partir de aquí, en una segunda fase se pretende implementar la arquitectura que permite la gestión del paging a nivel IP (IP paging) descrita en [7]. En este RFC se describen elementos, sus funcionalidades y sus interfaces de comunicación pero en ningún momento se especifican ni que protocolos se deben utilizar ni el número ni el formato de los mensajes intercambiados. Esto llevará a una toma de decisiones de diseño previas a la programación de esta arquitectura.

Así pues, los objetivos de este trabajo pueden resumirse en cuatro. Primero comprender y modificar la implementación del protocolo de transporte de paging IP existente. El segundo probar y determinar el rendimiento de dicha implementación buscando sus cuellos de botella. El tercero, implementar la arquitectura de paging IP comentada anteriormente. Y finalmente, el cuarto consistirá en probar de manera conjunta el protocolo y la arquitectura de paging IP para comprobar su correcto funcionamiento. Todo el trabajo realizado para cubrir estos objetivos se ha documentado a lo largo de los capítulos y los

anexos de esta memoria pensando siempre en facilitar una posterior continuación de este TFC.

La primera parte de la memoria presenta los fundamentos adquiridos necesarios para entender el protocolo Geopaging, así como modificar y caracterizar su implementación. De esta manera en el capitulo 1 se explicará a fondo el protocolo Geopaging: el sistema de identificación de celdas, su traducción a direcciones IPv6, el sistema de paging dinámico basado en distancia, la construcción de tablas de encaminamiento y la posterior construcción del árbol multicast. En el segundo capítulo se especifican los primeros cambios realizados en el código existente para mejorar su rendimiento, y se plantea como conseguir un escenario más realista que el empleado en [1]. Es en el capítulo 3 donde se detallan todas las pruebas realizadas para comprobar el correcto funcionamiento de los cambios realizados, así como detectar los cuellos de botella en el código existente.

En los siguientes capítulos, se entra en el detalle de la nueva parte del código agregada correspondiente a la arquitectura de paging IP, así como también las pruebas de su correcto funcionamiento. El capítulo 4, explica en su primera parte la base teórica para el diseño de los agentes, así como los mensajes que se intercambiarán en los posibles casos de paging o localización. Finalmente se describen las funciones de cada uno de los agentes y del nodo móvil. En el capítulo 5 se explican las pruebas realizadas con los nuevos agentes y el nodo móvil para comprobar que realmente la arquitectura implementada funciona y los mensajes implementados se envían y reciben correctamente.

Cierra la memoria las conclusiones donde se resumen los aspectos más destacados de este TFC, se realiza una valoración crítica de los objetivos planteados al inicio del trabajo, y establecen las líneas para una continuación de este proyecto. Se adjuntan también una serie de anexos que complementarán la información detallada en la memoria. El primero explicará, paso por paso, la instalación del software del protocolo ya implementado. En el segundo se amplía la descripción del código y de su funcionamiento de cada uno de los agentes diseñados además del nodo móvil realizada en el capítulo 4.

# CAPÍTULO 1. GEOPAGING: PROTOCOLO E IMPLEMENTACIÓN EXISTENTE

## 1.1. Introducción

Como ya se ha comentado en la introducción, en este TFC se ha trabajado con una implementación del protocolo Geopaging fruto de un TFC anterior [1]. En este capítulo, a modo de recordatorio, se explica el protocolo Geopaging, empezando por la problemática de la identificación de las celdas, seguido de cómo a partir de estos identificadores se construyen direcciones IPv6, y finalmente se muestra el proceso de generación de un mensaje de paging y su enrutado a través de la red. Para terminar, se describen las principales características de la implementación existente de este protocolo, haciendo especial hincapié en aquellas partes que quedaron abiertas y que serán objeto de trabajo en el presente TFC.

## 1.2. Direccionamiento

El funcionamiento del protocolo se basa en un esquema creado para identificar celdas mediante direcciones IPv6. Esto es posible gracias a la escalabilidad que permite el amplio espacio de direcciones de IPv6. Estas direcciones se forman a partir de unas etiquetas que identifican a cada celda. La principal característica de estas etiquetas es que si un terminal móvil conoce las dos etiquetas correspondientes a dos celdas, puede calcular fácilmente y de manera exacta su distancia en celdas.
A continuación se describe el mecanismo de construcción de estas etiquetas y el de las direcciones IPv6 asociadas a cada una de ellas.

### 1.2.1. Construcción de las etiquetas

El primer paso para la construcción de las etiquetas es aproximar la cobertura de una celda con un hexágono. El conjunto de siete hexágonos (clúster), dispuestos según muestra la figura 1.1, forman una celda de nivel superior. Si se agrupan siete clusters del mismo nivel tenemos otro clúster de nivel superior. La identificación de cada celda se realiza usando un eje de coordenadas. Identificar una celda mediante su coordenada (i,j) es equivalente a identificarla mediante un número de 0 a 6 resultado de sumar 2i + j, en la figura 1.2 se observa un ejemplo.

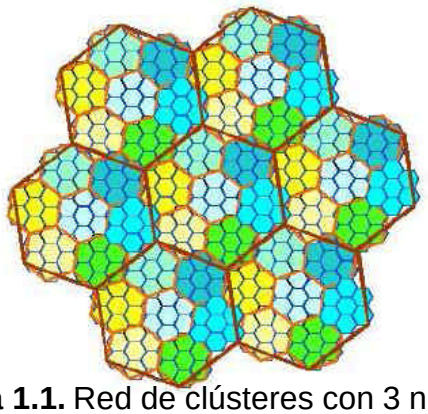

**Figura 1.1.** Red de clústeres con 3 niveles

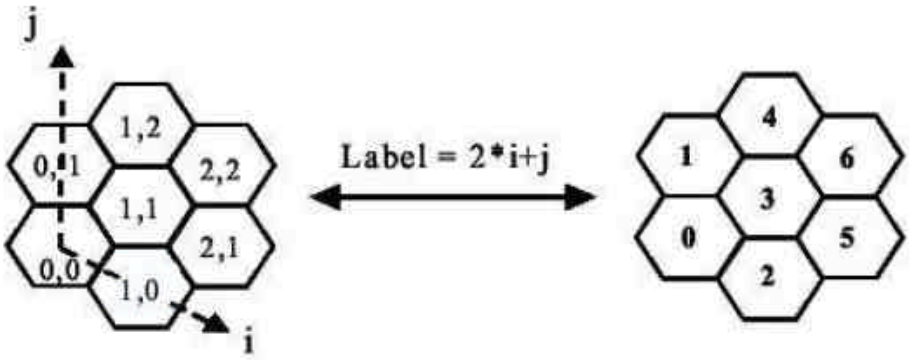


**Figura 1.2.** Sistema de coordenadas y de numeración de las celdas

Utilizar una figura de 7 celdas permite usar simplemente 3 bits para identificarlas (del 0 a 6). En la figura 1.3 se puede observar un ejemplo de jerarquía de clusters. Los identificadores de celda 7 se entienden como clústeres que no contienen celdas de jerarquía inferior (en la figura, la celda 5.7).

Por ejemplo, si consideramos sistemas con clusters de hasta 3 niveles, podemos tener la celda 1.3.4, la 0.2.5, la 6.2.4, etc. Para indicar que el clúster de nivel 3 con identificador 5 no tiene ningún cluster de nivel inferior (nivel 2), se usa el número 7, es decir, será el cluster 5.7.0.

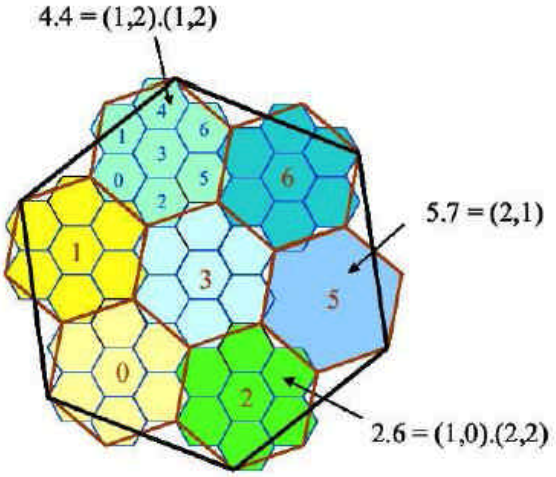


**Figura 1.3**. Ejemplo de jerarquía de tres niveles

### 1.2.2. Construcción de las direcciones

Una vez que tenemos los ID de celda podemos obtener dos direcciones IPv6. Una de ellas es una dirección unicast virtual, asociada con cada Access Point (AP) quien traducirá el mensaje de paging al protocolo de paging usado en el medio radio. Es virtual porque el AP es un dispositivo de nivel 2. La dirección unicast se consigue a partir de la etiqueta de la celda: cada número es representado en binario con 3 bits, y se ponen de tal manera que el ID de más bajo nivel se coloca al final de la dirección IPv6. Por ejemplo el AP con ID 0.4.4 es en hexadecimal 000 100 100, y si lo representamos con números hexadecimales es 0x024. Este número será el ID de interfaz de la dirección IPv6 del AP. Si usamos un ID de subred site-local, por ejemplo FEC0:0:0:1::/64, el AP tendrá una dirección virtual FEC0:0:0:1::024 (ver figura 1.4).

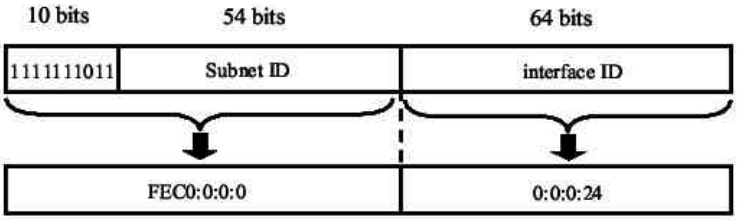


**Figura 1.4**. Dirección unicast asociada a la celda

La otra dirección es la multicast, utilizada como IP destino de la petición de paging. La celda identificada por la porción de ID de grupo de la IP multicast será el centro del área paging. La parte de la dirección correspondiente al ID de subred será por ejemplo una dirección multicast permanente del tipo site-local (FF08::/8). También se puede utilizar un identificador para diferenciar nuestras direcciones de otras reservadas para otros usos, por ejemplo se podría usar el ID 6666:0:6666 resultando así un prefijo de subred FF08:6666:0:6666/64 (figura 1.7).

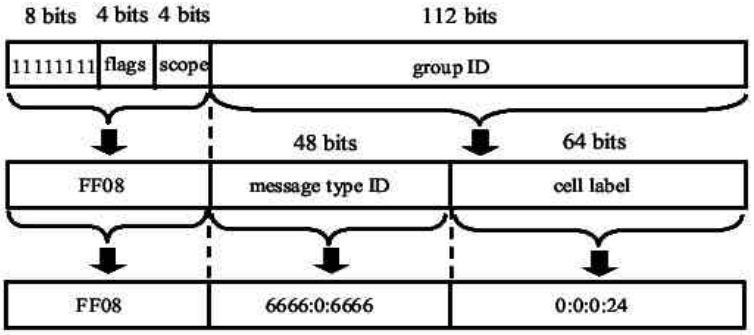


**Figura 1.5**. Dirección multicast asociada a la celda

De esta manera en cada router tendremos tablas de enrutado unicast con direcciones del rango FEC0:0:0:1::/64, y encaminarán paquetes con IP destino FF08:6666:0:6666/64.

## 1.3. Paging mediante Geopaging

El *Paging Agent* (PA) que quiera enviar una petición de paging a un agente móvil, deberá construir un paquete IPv6 multicast. Esa dirección la obtendrá a partir de la etiqueta de celda donde el agente móvil anunció su última posición (mediante un mensaje de *location update*). El margen de distancia que el PA especificará lo obtendrá según varios factores. Si el dispositivo siempre ha tenido mucha movilidad (por ejemplo, un móvil en un coche), el valor de distancia será mucho mayor que el de un agente más sedentario (por ejemplo un móvil en una oficina). Esa distancia la incluirá en la *Hop By Hop Option Extension Header* del paquete IPv6 multicast. También pondrá el ID del agente móvil, ya sea su dirección IPv6 de Mobile IP, su IMSI (*Internacional Mobile Subscriber Identity*) o su NAI (*Network Acces Identifier*), dentro de una *Destination Option Extension Header*.

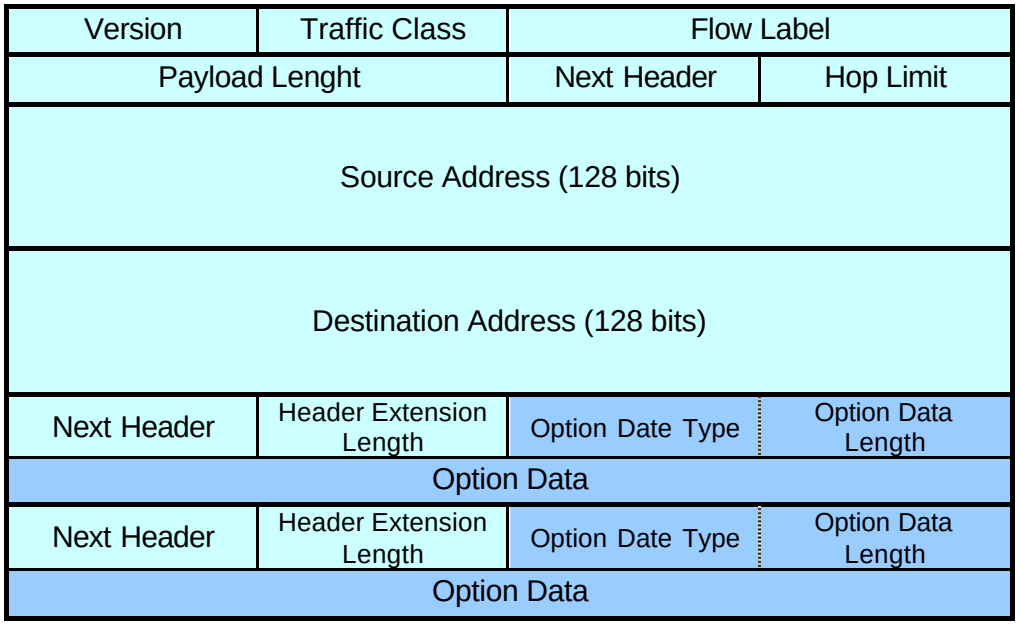


**Figura 1.6.** Estructura de un mensaje de paging

Los routers y en último término los *Access Routers* (AR) son los encargados de encaminar estos paquetes de paging en dirección a los AP, que son dispositivos de nivel 2 que comunican con los agentes móviles. El árbol multicast acaba en los AR que comunican directamente con los AP.

En la figura 1.8 podemos ver un ejemplo de paging a un dispositivo que envió su última actualización (*Location Update*, LU) desde la celda negra. En este caso, la distancia de paging es 1, por lo tanto el router sólo enviará el paquete al AR de la izquierda, el cual hará paging en aquellas celdas que controla y que cumplen la distancia de paging. En la figura 1.9 la distancia de paging es 2, con lo que el router reenvía el paquete a ambos AR.

En la figura, el árbol multicast se puede ver como los segmentos donde pasa el paquete de paging (las flechas representan un paquete de paging). Este proceso se explica en detalle en el apartado 1.5

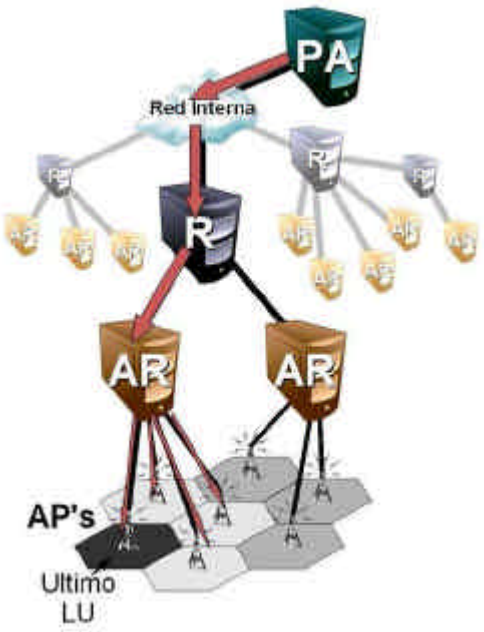


**Figura 1.7**: Paging con distancia 1

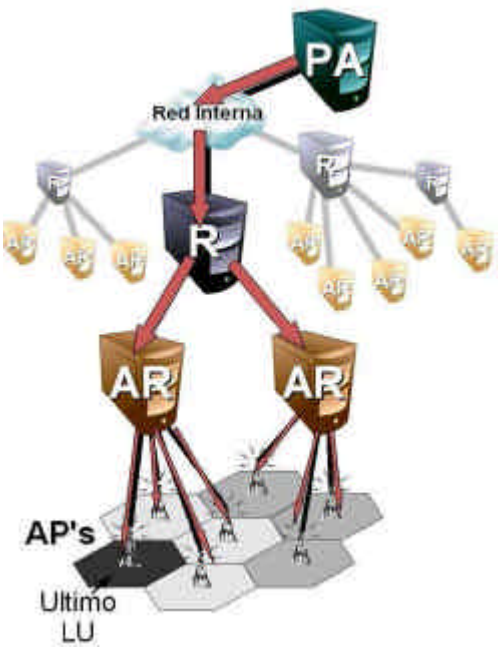


**Figura 1.8**: Paging con distancia

## 1.4. Construcción de tablas de enrutado

Es importante destacar que las áreas de paging del protocolo Geopaging son diferentes e independientes de las subredes IP. Cuando se desea que un AR controle un grupo de celdas (cada una con un AP), se le añaden las rutas con las direcciones unicast virtuales de las celdas de esa área. El protocolo de encaminamiento unicast utilizado se encargará de dar a conocer esa área de paging al resto de AR.
Un posible problema es que tenemos una ruta por cada AP, habiendo miles de éstos en una red celular tendríamos tablas de enrutado muy grandes. La solución está en la agregación de rutas, pues un AR controlará varios AP que probablemente estén cerca geográficamente, entonces se pueden suprimir algunas entradas. Por ejemplo si un AR controla las celdas 0.2 y 0.3, en binario tenemos 000010 y 000011. El router entonces deberá agregar ambas y tener una sola ruta a la dirección IP que acaba (en binario) en 000001x, es decir, acortamos la máscara de subred 1 bit. Contra más AP se añadan, mayor será la cantidad de rutas agregadas que se podrán realizar

## 1.5. Algoritmo de construcción del árbol multicast

En el momento que el Paging Agent envía el mensaje de paging, todos los AR son candidatos a recibirlo. El software debe ser capaz de entregar correctamente el mensaje usando solamente las tablas de enrutado y la información contenida en el mensaje de paging.

Cuando un router recibe un mensaje de paging, comprueba cada entrada en su tabla de enrutado para ver si hay alguna celda que pertenezca al área de paging solicitada. En el caso correcto, se marca la interfaz de salida como

usada. El router entonces sigue buscando en su tabla de enrutado, sin mirar ahora en aquellas entradas que pertenezcan a alguna interfaz marcada como usada. Esto ahorra mucho tiempo de proceso, vital para la rápida entrega del mensaje de paging. Este proceso se repite en todos los AR intermedios. El cálculo para saber si una dirección IPv6 está dentro del área de paging es un sencillo problema de geometría que se resume en contar las coronas que hay entre una celda y otra. Entendemos por corona al conjunto de celdas colindantes a una en concreto. Por ejemplo, con una distancia de uno, el área de paging se limita a las 6 celdas alrededor de la celda destino (ésta también se incluye en el área). Si la distancia es 2, el área la forman la primera corona (6 más la celda destino) y los 12 colindantes a ésta. Con distancia 3, tenemos los anteriores más los 18 colindantes a la segunda corona, y así sucesivamente.
De este modo, si el dispositivo móvil envió su última localización desde la celda negra de la figura, un paging con distancia 3 se enviaría a todas las celdas dentro de la tercera corona.

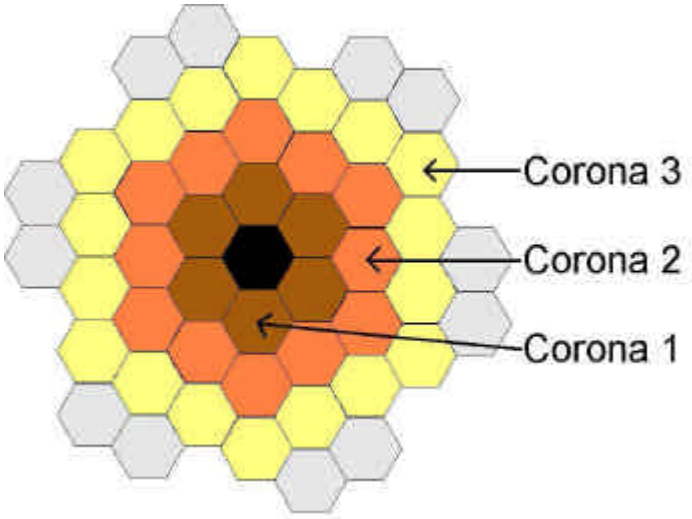


**Figura 1.9**: Aumento de la cantidad de celdas receptoras en función de la distancia

Si tuviéramos dos routers que controlasen dos áreas (ver figura 1.11), el árbol multicast se crearía en función de la distancia de paging solicitada. El router 1 tiene un área de radio R1 (4) centrada en la celda C1; y el router 2 tiene un área en C2 de radio R2 (1). Si el dispositivo móvil envió su última localización (Location Update, LU) desde la celda de color negro, entonces el Paging Agent enviará un paquete de paging con una distancia de paging D (2) a la dirección multicast de la última celda donde se recibió el LU. En el caso de que llegue ese paquete a ambos routers, sólo el primero reenviará el paquete porque el área de paging cae dentro del área que él controla. La condición para reenviar el paquete multicast es la siguiente (Fórmula 1.1):

$$\text{Distancia}[C1, \text{Celda\_LU}] - R1 <= D \quad (3.1)$$

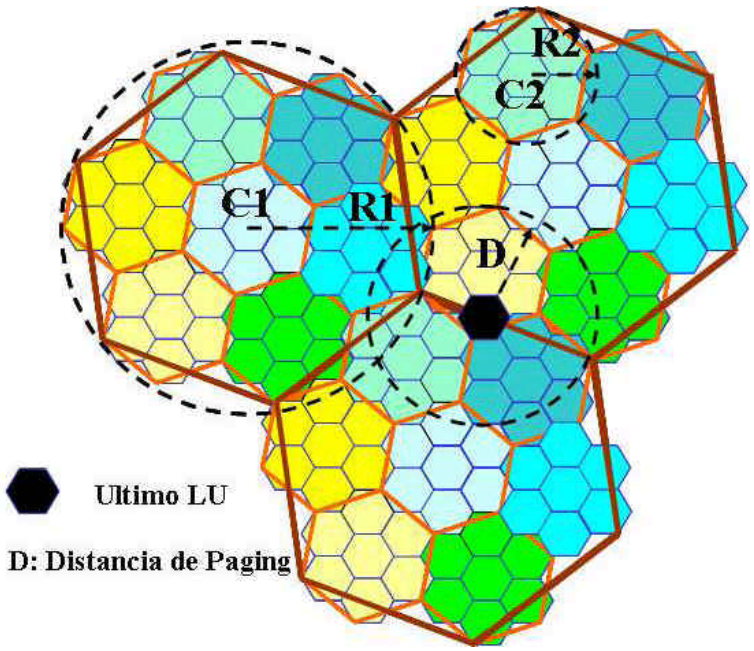


**Figura 1.10:** Propagación multicast basándose en la distancia de paging

Es decir, tanto el router 1 como el router 2 tienen que aplicar la fórmula 1.1 para cada una de sus rutas. Para simplificar, en el ejemplo de la figura 1.11 suponemos que sólo tienen una ruta configurada. El router 1 tiene una ruta a un cluster de nivel 3 (49 APs) centrado en C1. El router 2 tiene una ruta a un cluster de nivel 2 (7 APs) centrado en C2. Cuando les llega una petición de paging de distancia D, centrada en la celda del último LU (celda negra), ambos aplican la fórmula. Los resultados se ven en la tabla 1.1.

**Tabla 1.1**: Resultado de aplicar la fórmula 1.1 en el escenario de la figura 1.10

| **R1** | **D** | **D[C1, LU]** | **Formula 3.1** | **Reenvía?** |
|---|---|---|---|---|
| 4 | 2 | 6 | 6 – 4 <= 2? | SI |
| **R2** | **D** | **D[C2,LU]** | **Formula 3.1** | **Reenvía?** |
| 1 | 2 | 6 | 6-2<=2? | NO |

El cálculo a la hora de enrutar es diferente pues en la tabla de encaminamiento los routers no tienen un parámetro que especifique el área de paging que controlan. Simplemente tienen una lista de los AP bajo su dominio, es decir, la lista de direcciones IPv6 que hay en la tabla de encaminamiento. Cada una de esas direcciones representa un ID de celda. El cálculo de distancias se realiza con esos ID de celda. En este ejemplo, el router 1 controla la celda C1, la cual representa un cluster de nivel 3 (por ejemplo, con identificador 3). Como ya se explicó en el apartado 1.2.1, si un cluster no contiene clusters de nivel inferior, se usa el número 7 como ID inferior. Así pues, el router 1 controla el cluster 3.7.0, y el router 2 controla el cluster 5.4.7 porque C2 es el cluster 4 de nivel 2 dentro del cluster 5 de nivel 3. Entonces el cálculo para saber la distancia entre dos celdas, sabiendo solamente sus ID, es un problema de geometría que se explica a fondo en [3].

## 1.6. Implementación existente

### 1.6.1. Características de la implementación

Para implementar el protocolo Geopaging se desarrolló un software llamado Mcast, este software encamina los paquetes de paging generados por una pequeña aplicación llamada sendip, hacia la máquina destino. De este programa destacamos las siguientes características:

- El SO utilizado es la distribución GNU/Linux Gentoo con kernel 2.6.
- Para la programación del protocolo se utilizó el lenguaje de programación C.
- El programa accede a datos externos como es la tabla de enrutado, la lista de interfaces del ordenador y las direcciones IPv6 configuradas en los routers. Para obtener las rutas y las interfaces del sistema se lee de ficheros de texto de la interfaz /proc usando llamadas del tipo IOCTL para acceder a datos como son la dirección MAC de una interfaz.
- Utiliza librerías de bajo nivel pues el software lee, modifica e inyecta paquetes IPv6 personalizados que deben encaminarse por las interfaces indicadas. De estas destacan libpcap y libnet, librerías externas utilizadas para capturar e inyectar paquetes respectivamente. La utilización de estas librerías se debe a que las librerías internas que trae el propio kernel no soportaban todas las opciones de IPv6 necesarias para el programa.

A parte de estas características generales para solucionar los requerimientos del protocolo se programaron tres conceptos que en un principio no especificaba el protocolo.

1. El agente de paging puede estar en cualquier lugar de la red. No se predefine su posición y por lo tanto el código está preparado para recibir una petición de paging de cualquier interfaz y desde cualquier nodo y enrutarla según el árbol multicast correspondiente.
2. El protocolo crea un árbol multicast no basándose en información intercambiada sobre miembros del grupo (como utilizan los protocolos DVMRP, Distance Vector Multicast Routing Protocol [1] o MOSPF, Multicast Open Shortest Path First, [1]), sino en cálculos basados en distancias de direcciones IPv6.
3. Se utiliza RPB, Reverse-Path Broadcasting [1], para transmitir únicamente los paquetes que llegan procedentes de la interfaz con el camino más corto hacia la dirección IP del origen del paquete.

### 1.6.2. Puntos abiertos a mejorar en Mcast

En la anterior implementación del protocolo se dejaron varios temas abiertos, a continuación los listaremos y mencionaremos brevemente sus posibles soluciones.

- Limitación en el número de rutas e interfaces con las que puede trabajar Mcast, una posible solución a este problema seria el paso a memoria dinámica abriendo la posibilidad de trabajar con el número de rutas e interfaces realmente necesarias para la aplicación.

- Los scripts de configuración de la red son poco escalables, tan sólo permiten agregar un número de rutas muy reducido que queda muy por debajo del número que encontraríamos en un escenario real. Una posible solución sería generar un nuevo método de generación de rutas basado en los scripts.

- Mcast implementa el protocolo Geopaging encaminando los paquetes correctamente como se demostró en [1] pero un aspecto no tenido en cuenta en el TFC anterior es que no contempla el formato de los mensajes de paging enviados. La herramienta utilizada por Mcast para enviar el paging es un programa que envía paquetes IPv6 a una determinada dirección multicast con una determinada distancia de paging sin tener en cuenta todas las Extensions headers que forman un paquete IPv6. Otro tema abierto para mejorar es pulir el formato de estos mensajes enviados de tal manera que sean paquetes IPv6 bien construidos con las Extensions Headers necesarias para poder trabajar con ellas en futuras implementaciones, de estos mensajes se hablará más a fondo en el capítulo 4.

- Eliminación del código de la librería libnetlink no utilizada y que reduce el tamaño del ejecutable. En un principio esta librería iba a utilizarse para obtener información sobre las interfaces, sobre las direcciones configuradas y sobre la tabla de enrutado, pero la programación en estas librerías era más difícil que con routing sockets, y el soporte de IPv6 es mejor en estos últimos que en libnetlink. Esta librería estaba incluida junto con el resto del código aunque ya no es necesaria debido a las pruebas iniciales que se realizó para la programación de Mcast. Por tanto su eliminación es recomendable pues reduciremos el tamaño del ejecutable sin que ninguna de sus funcionalidades quede afectada.

- Posible migración del código a FreeBSD según los resultados obtenidos en las pruebas. Una vez realizadas las pruebas de rendimiento sobre Mcast se decidirá el paso a FreeBSD o mantendremos la distribución actual.

- Estudiar la utilización de librerías como bs packets sockets o los BSD sockets. La utilización de los packets sockets aumenta la complejidad de la programación ya que trabaja a más bajo nivel y los routing sockets se tendrían en cuenta con la migración del código a FreeBSD.

Finalmente comentar que en el momento de comenzar este TFC no se disponía de ningún manual de instalación de Mcast. Este aspecto fue el primero en ser solventado y queda recogido en el anexo I.

# CAPÍTULO 2. MEJORAS REALIZADAS EN EL CÓDIGO MCAST

## 2.1. Introducción

En el capítulo anterior se detallaron los aspectos más significativos a mejorar de la implementación existente de Geopaging. En este capítulo se explicarán las mejoras realizadas sobre el código existente para optimizar su rendimiento

## 2.2. Agregación de rutas

### 2.2.1. Scripts de configuración

Uno de los objetivos finales de la implementación de este protocolo es que se pueda utilizar en redes reales, con un número de celdas indefinido, en la mayoría de los casos este número puede llegar a ser de millones de ellas.

El escenario planteado en [1] simula una jerarquía de celdas donde cada una de ellas tiene asignada una dirección IP diferente. Estas celdas se representan mediante las interfaces dummy, para acceder a cada celda necesitaríamos que cada una de estas interfaces tuviera una ruta con la dirección de las celdas. Entonces si imaginamos un escenario con un número reducido de celdas no habría problema en agregar a mano la ruta correspondiente a cada una de ellas, pero si queremos crear un escenario con un número de celdas mayor, que se acerque más a la realidad necesitamos crear un método automático para que agregue todas las rutas sin necesidad de introducirlas una a una.
En definitiva aumentar el número de celdas hace que necesitemos crear un método que agregue nuevas rutas de forma automática y a su vez este número de rutas sea fácilmente ampliable en caso de necesitar un número de celdas mayor.
Para facilitar esta pequeña mejora al código de Mcast se han creado nuevos scripts de configuración de rutas para las interfaces dummy0. Se ha implementado un script por cada máquina que interviene en la maqueta.

Estos nuevos scripts se basan en agrupar las celdas. A continuación explicaremos el proceso seguido para la comprensión del comportamiento de los routers a la hora de agregar las rutas. Este método de creación de los nuevos scripts se deduce del funcionamiento de las celdas en Geopaging y su traducción a dirección IPv6 mencionado en [2]. Partiremos de un ejemplo práctico con el que facilitaremos la comprensión del método. En este ejemplo, nos hemos basado en una red con 5 niveles de clúster, con este número de niveles es posible la creación de 15625 rutas, este valor surge de elevar 75, 7 celdas que forman un cluster y 5 niveles, consideramos que éste es un número bastante elevado y más aproximado a la realidad que el número utilizado en [1] aunque este punto está abierto a posibles mejoras de implementación.

Sabemos, que a cada ID de celda le corresponde una dirección IPv6, el proceso a seguir para asociar una ID de celda a una dirección IPv6 es el siguiente:

1. Cada número de ID de celda se transforma en uno binario de 3 dígitos. Por ejemplo, la ID 1.1.1.1.0 en binario sería: 001.001.001.001.000.
2. Una vez realizada la transformación se agrupan los bits en grupos de cuatro, empezando por la derecha. Si continuamos el ejemplo anterior, la ID 1.1.1.1.0 en binario y en grupos de cuatro seria: 0001.0010.0100.1000.
3. Finalmente, transformamos a hexadecimal las agrupaciones quedando el ejemplo anterior como: 1248. Una posible IP para esta celda sería por ejemplo FEC0::1248/64.

Una vez conocemos el proceso para asociar los ID de celdas a determinadas direcciones IP debemos proponernos averiguar un agregado de rutas, para ello, consideramos que cada router puede tener uno o varios clusters de nivel 5 y que por ejemplo, el router1 contiene 2 de estos clusters, el clúster 0 y el clúster 1.
Para descubrir un agregado de rutas óptimo, hemos de fijarnos en la conversión de la primera y de la última celda del clúster 1, que en este caso corresponderían a las ID de celda: 1.0.0.0.0 y 1.6.6.6.6 respectivamente. Siguiendo los pasos de la conversión anterior, la IP de la primera celda sería FEC0::1000/64 y la de la última FEC0:1EB6/64. Del mismo modo, la primera y la última celda del clúster 0 serían FEC0::/64 y FEC0::0EB6/64.
Los resultados muestran como únicamente cambian los 3 últimos dígitos, es decir, los 12 últimos bits, en direcciones IP del mismo clúster.
Asimismo, la máscara de subred en IPv6 se compone de 128 bits a los que restamos los 12 últimos bits que varían y obtendremos la máscara de subred necesaria para agregar todas las rutas. A continuación mostramos el código de los scripts utilizados:

```
#! /bin/bash
ifconfig dummy0 up
ifconfig dummy0 add fede::0:2/112
route –A inet6 add FEC0::1000/116 dev dummy0
route –A inet6 add FEC0::2000/116 dev dummy0
```

**Figura 2.1.** Fichero de configuración configura_router1.sh

```
#! /bin/bash
ifconfig dummy0 up
ifconfig dummy0 add fede::0:2/112
route –A inet6 add FEC0::3000/116 dev dummy0
route –A inet6 add FEC0::4000/116 dev dummy0
```

**Figura 2.2.** Fichero de configuración configura_router2.sh

```
#! /bin/bash
ifconfig dummy0 up
ifconfig dummy0 add fede::0:2/112
route –A inet6 add FEC0::5000/116 dev dummy0
route –A inet6 add FEC0::6000/116 dev dummy0
route –A inet6 add FEC0::7000/116 dev dummy0
```

**Figura 2.3.** Fichero de configuración configura_router3.sh

Con este nuevo código permitimos la creación y agregación de un número de rutas que supera las 15000 rutas.

Recordamos que para que estos archivos sean ejecutables, se les debe dar permiso de ejecución. Con la consola y el comando: “chmod +x” seguido del nombre del archivo que se quieran cambiar sus permisos.

Como podemos observar en los scripts anteriores, la interfaz dummy0 se configura con direcciones IPv6 y las rutas se agregan a las direcciones de las celdas mediante esa misma interfaz.

Finalmente Cabe recordar que dummy es la denominación que le da el kernel de Linux a la interfaz virtual y cuyo soporte se activa con la opción CONFIG_DUMMY=Y de cada kernel. Otra característica a tener en cuenta es que el tráfico enviado hacía las interfaces dummy0 no se transfiere por ninguna interfaz real. En el escenario representado, dummy0 es la interfaz que comunica los routers con la red de acceso celular nivel 2, ethernet, frame relay, o tecnologías equivalentes, que dan conectividad a los puntos de acceso (APs).

## 2.3. Paso a memoria dinámica

### 2.3.1. Eliminación de limitaciones

Otra de las modificaciones realizadas en el código Mcast ha sido la eliminación de la limitación que tenía referente al número máximo de interfaces y de rutas que podía implementar el programa.

Hasta el momento la implementación del programa estaba limitada a la activación de un máximo de 20 interfaces y de 200 rutas que se podían agregar. Trabajar con memoria estática supone, por un lado, una reserva de memoria fija, desde el momento en que el programa se ejecuta, se necesita un espacio de memoria suficiente para el número de interfaces y rutas configurado, una de las características de este tipo de memoria es que puede darse el caso que muchas veces se inicialice un espacio de memoria que no se llegue a utilizar nunca, por otro lado si miramos de llevar el programa a un escenario real, el número de rutas que permite agregar es bastante pequeño

comparado con la realidad. Tenemos que tener en cuenta, que tampoco se podía configurar inicialmente un número muy grande de rutas para agregar, ya que como se ha comentado anteriormente esto significa la reserva de un espacio de memoria determinado y a priori desconocemos si el sistema realmente dispone de todo este espacio.

Para solventar este problema, hemos pasado a trabajar con memoria dinámica, creando vectores indefinidos. Esta manera de trabajar permite no definir el número de interfaces y de rutas que se crearán, este número queda sin determinar, de tal manera que podemos tener 5 interfaces y 50 rutas para agregar o 50 interfaces y 1000 rutas para agregar. Por lo tanto, al no saber el número exacto de interfaces y rutas necesarias, el espacio necesario para esta operación también se desconoce y es aquí uno de los principales problemas que surgen al trabajar con memoria dinámica y que se nos plantean a la hora de realizar este cambio en el código.

Con este cambio liberamos espacio de memoria y permitimos la creación de un número superior de rutas para agregar. Para llevar a cabo todos estos cambios se ha tenido que cambiar parte del código existente. A continuación, exponemos cada uno de los cambios realizados así como una pequeña explicación de los efectos que suponen dichos cambios.

Empezaremos explicando el primer cambio que se ha realizado: la modificación del archivo *defs.h*., Es en este archivo donde se declaran todas las estructuras que intervienen en el programa así como la declaración de todas las librerías necesarias.

### *2.3.1.1. Modificaciones realizadas en defs.h*

En defs.h se han cambiado las estructuras *interficies_ip* y *rutas_ip*, estas estructuras han quedado de la siguiente manera:

```
struct interficies_ip
{     int cantidad;
      IFS_interfaz_t *interfaz;};
```

**Figura 2.4.** Estructura interficies_ip

Creando este puntero de interfaces se crea un vector dinámico de interfaces, de tal manera que el número de interfaces queda indeterminado, permitiendo que se activen tantas interfaces como sean necesarias. Otra característica que permite la creación de este puntero es que ahora el espacio de memoria necesario es desconocido, y a la hora de trabajar con él podremos crear y reservar el espacio en memoria que realmente sea necesario. La otra estructura modificada ha sido *rutas_ip*, ésta ha quedado de la siguiente manera:

```
struct rutas_ip
{     int cantidad;
      RT_ruta_t *ruta;};
```

**Figura 2.5.** Estructura rutas_ip

Antes de seguir detallando cada uno de los cambios realizados en las diferentes funciones, y poder entenderlos, primero vamos a explicar brevemente las funciones implicadas en el paso a memoria dinámica y su implicación en el proceso.

Cuando ejecutamos el programa, éste inicializa una estructura tipo *IFS_t* y *RT_T* en *main.c*.

En el paso anterior hemos explicado la modificación realizada en las estructuras, *interficies_ip* y *rutas_ip*. Cuando ejecutábamos el programa con la definición de las estructuras antiguas, *IFS_t* contenía 20 elementos del tipo *IFS_interfaz_t*, y *RT_t* inicializaba 200 elementos del tipo *RT_ruta_t*, ahora éstos no inicializan ningún vector de x elementos, inicializan una estructura que apunta a un vector indeterminado. Ambas estructuras siguen la política de tener un vector y un contador de elementos del vector (o cantidad de elementos), este pequeño contador será muy útil a la hora de reservar únicamente el espacio necesario.
El único código que modifica *RT_t* es *RT_parsea_from_proc(),* así que todos los cambios para pasar a memoria dinámica las rutas utilizadas afectarán a esa función, de manera similar a la anterior función, el código que se debe modificar en el paso a memoria dinámica de las interfaces es *IFS_parsea_from_proc().* Estas funciones se encuentran en los archivos *RT.c* y *IFS.c* respectivamente.

Siguiendo la estructura modular del código existente hemos creado varias funciones independientes para inicializar y reservar espacio suficiente en memoria para las estructuras necesarias en la ejecución del programa. A continuación mostramos el código de estas funciones.

```
void UTIL_init_interfaz_mem (IFS_t *ifs, int mem)
{
      ifs->interfaz=malloc(sizeof(IFS_interfaz_t));
      ifs->interfaz=realloc(ifs->interfaz,
mem*sizeof(IFS_interfaz_t));
}
```

**Figura 2.6.** Función inicialización y reserva de memoria para las interfaces

*Comentarios a la función:*

Las funciones creadas, pertenecerán a la familia de las funciones UTILS, ya que como se explica en [1] éstas recogen un conjunto de funciones generales. El primer paso que realiza es una inicialización del espacio en memoria con la ayuda de un malloc y posteriormente reserva espacio para tantas interfaces como se hayan creado.

```
 void UTIL_init_ruta_mem (RT_t *rts)
{
      rts->ruta=malloc(sizeof(RT_t));
}
```

**Figura 2.7.** Función inicialización de memoria para las rutas

```
 void UTIL_create_more_mem(RT_t *rts, int mem)
{
      rts->ruta=realloc(rts->ruta, mem*sizeof(RT_ruta_t));
}
```

**Figura 2.8.** Función reserva de memoria para las rutas

A diferencia del caso de interfaces, aquí creamos dos funciones diferentes para inicializar y reservar memoria, hemos decidido crear dos funciones diferentes ya que en el caso de las rutas se puede crear un número muy superior al de interfaces y a la hora de llamar a las funciones en el código, primero inicializaremos el espacio y posteriormente iremos creando según las necesidades de forma progresiva.

A continuación explicaremos los cambios realizados en el código fuente de Mcast

*2.3.1.2. Modificaciones realizadas en el código fuente Mcast*

*2.3.1.2.1. GEOPAG.c*

En este archivo se han cambiado todas las llamadas a la función ***`memset`*** que inicializaba el espacio en memoria necesario para la estructura ***`IFS_t`*** anterior.

Exactamente, se han extraído del código de la función ***`GEOPAG_router`***, las siguientes llamadas a la función ***`memset`***:

```
memset(&child,0,sizeof(IFS_t));
memset(&interfaces_downlink,0,sizeof(IFS_t));
```

Estas llamadas corresponden a las líneas 231 i 232 del código de ***`GEOPAG.c`***

Estas líneas se han eliminado, ya que ahora ***`IFS_t`*** contiene un entero y un

puntero que apunta a una estructura cuyo tamaño desconocemos, por lo tanto no podemos inicializar un espacio desconocido.

*2.3.1.2.2. IFS.c*

Del mismo modo que anteriormente en ***GEOPAG.c*** hemos eliminado todas las llamadas a la función ***memset*** que inicializaban la estructura ***IFS_t***, en ***IFS.c*** también debemos realizarlo. Esta eliminación se realiza en las líneas: 21, 71, 92, 132.

Anteriormente se ha comentado que las funciones ***IFS_*** son las que deben inicializar el espacio y reservarlo. En ***IFS_parsea_from_proc*** hemos añadido después de leer las direcciones IP del proc la llamada a la función que inicializa y reserva espacio para las interfaces, línea 32 de ***IFS.c.***

`UTIL_init_interfaz_mem(ifs, contador);` a esta función le pasamos una estructura ***IFS_t*** y el número de IP's que ha leído del archivo.

Otros cambios que tenemos que realizar son que en las dos funciones que sacan únicamente las interfaces solositelocal y sololinklocal, hemos de inicializar y reservar espacio para esas interfaces, ya que les pasamos otras estructuras diferentes a las inicializadas en ***IFS_parsea_from_proc.***

*2.3.1.2.3. RT.c*

Como en los casos anteriores, aquí también hemos suprimido la inicialización del espacio de memoria con la llamada a la función memset, ahora esta inicialización no es necesaria, ya que únicamente creamos espacio para las interfaces y rutas utilizadas y por otro lado, ahora trabajamos con estructuras que contienen punteros que apuntan a otras estructuras cuyo espacio no conocemos.

La función en la que se ha creado espacio, ha sido ***RT_parsea_from_proc***, en esta función, primero inicializamos el espacio y reservamos únicamente espacio para las rutas que lee del proc. En este caso, hablamos de dos funciones separadas porque el número de rutas que se van creando es muy superior al de interfaces que se puedan activar y como no es trivial el trabajo con la memoria dinámica que realizan las funciones ***malloc y realloc***, decidimos por seguridad separar las funciones.

*2.3.1.3. Liberación del espacio utilizado*

De la misma forma en que inicializamos y reservamos espacio en memoria para trabajar con las estructuras explicadas en los puntos anteriores, debemos una vez finalizado el proceso liberar el espacio utilizado.

Esta liberación de espacio es necesaria siempre que se trabaje con las

funciones *malloc() y realloc(),* ya que éstas van cogiendo espacio de la memoria de la máquina y reservándolo para su utilización y si éste no se libera cuando se acaban todos sus procesos éste espacio queda inutilizado todo el tiempo que éste corriendo el código en cualquier máquina. Esto puede carecer de importancia a primera vista, pero si se mira desde la perspectiva de que Mcast se implementa en una máquina determinada pasado x tiempo puede haber consumido x memoria del sistema y esta memoria se irá incrementando todo el tiempo que Mcast esté corriendo en la máquina, en cambio si se libera el espacio una vez acabadas todas las tareas esta memoria se libera y el sistema puede utilizarla para cualquier otra cosa. Este detalle puede carecer de importancia en máquinas potentes pero en máquinas menos potentes, el incremento de memoria que estas funciones van reservando es bastante significativo, una prueba de ello se ha realizado con Mcast en el escenario del laboratorio y se muestran algunos de los resultados obtenidos más adelante.

Para liberar la memoria utilizada se utiliza la función *free(),* esta función libera las estructuras cuyo espacio se ha inicializado y reservado anteriormente con la función *malloc().*

Las funciones creadas para liberar el espacio en memoria utilizado por las rutas e interfaces son:

```
void UTIL_libera_mem_rutas(RT_t *rts)
{
      free(rts);
}
```

**Figura 2.9.** Función para liberar el espacio utilizado por las rutas

```
void UTIL_libera_mem_int(IFS_t *ifs)
{
      free(ifs);
}
```

**Figura 2.10.** Función para liberar el espacio utilizado por las interfaces

# CAPÍTULO 3: RENDIMIENTO DE MCAST

## 3.1. Introducción

Una vez vistas las mejoras realizadas en la implementación de Geopaging, en este capítulo se explicará el estudio de rendimiento realizado sobre Mcast, para probar el rendimiento del protocolo y poder compararlo con otros protocolos de encaminamiento existentes. Para poder realizar este análisis, se ha diseñado un escenario diferente al utilizado en [1] que se describe en este mismo capítulo.
En este apartado se estudia el comportamiento de Mcast y sobretodo se analiza su rendimiento realizando diferentes pruebas a fin de observar y poder determinar en qué partes del código la ejecución de Mcast es más lento y puede presentar un cuello de botella.

El capítulo empieza con la descripción del nuevo escenario, seguida de la descripción de cada una de las pruebas realizadas. La primera prueba que se explicará será la realizada bajo un protocolo unicast estándar, con el fin de disponer de una marca temporal de referencia con la que valorar los tiempos obtenidos posteriormente con las pruebas de Mcast. Después se exponen las pruebas realizadas sobre Mcast con un capturador de trazas y finalmente se realizan otras pruebas de rendimiento de Mcast pero esta vez con marcas temporales introducidas en el propio código fuente del programa. También se incluye una prueba enfocada a determinar el consumo de memoria del programa y finalmente se reflexiona sobre los resultados obtenidos.

## 3.2. Objetivos de las pruebas

Los objetivos de las pruebas son los siguientes:

- Comprobar que las modificaciones del código no afectan al correcto funcionamiento de Mcast.
- Ver si es necesario disponer de un Hardware mínimo para correr Mcast
- Contrastar el rendimiento de Mcast con un protocolo estándar en un mismo escenario
- Determinar los cuellos de botella del código Mcast

## 3.3. Descripción del nuevo escenario

En la figura 3.1 se muestran los equipos y la interconexión entre ellos que se utilizaron para el análisis. Las interfaces de red se llaman ethx y las interfaces virtuales se han llamado dummy0, en la página 13 de este mismo documento se describe que es una interfaz dummy0. Cada segmento de red se ha nombrado Red1, Red2, Red3.

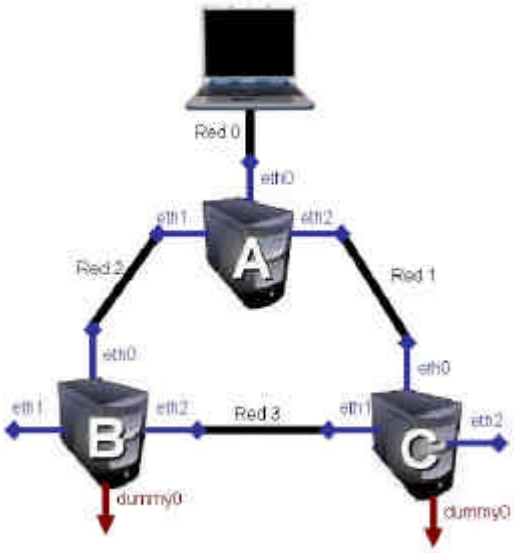


**Figura 3.1** Esquema de elementos del escenario de pruebas

Los componentes del escenario son tres PCs que actúan de router, cada uno tiene tres interfaces de red ethernet, de 100Mbits cada una, activas. Además el inyector de tráfico será un portátil, éste actúa de Paging Agent, recordemos que el agente de paging es el encargado de enviar el paging. Las especificaciones de hardware, configuraciones de red, velocidades de transmisión, etc. se detallarán en el siguiente apartado.

La topología mostrada en la figura 3.1 representa el escenario de la figura 3.2 :

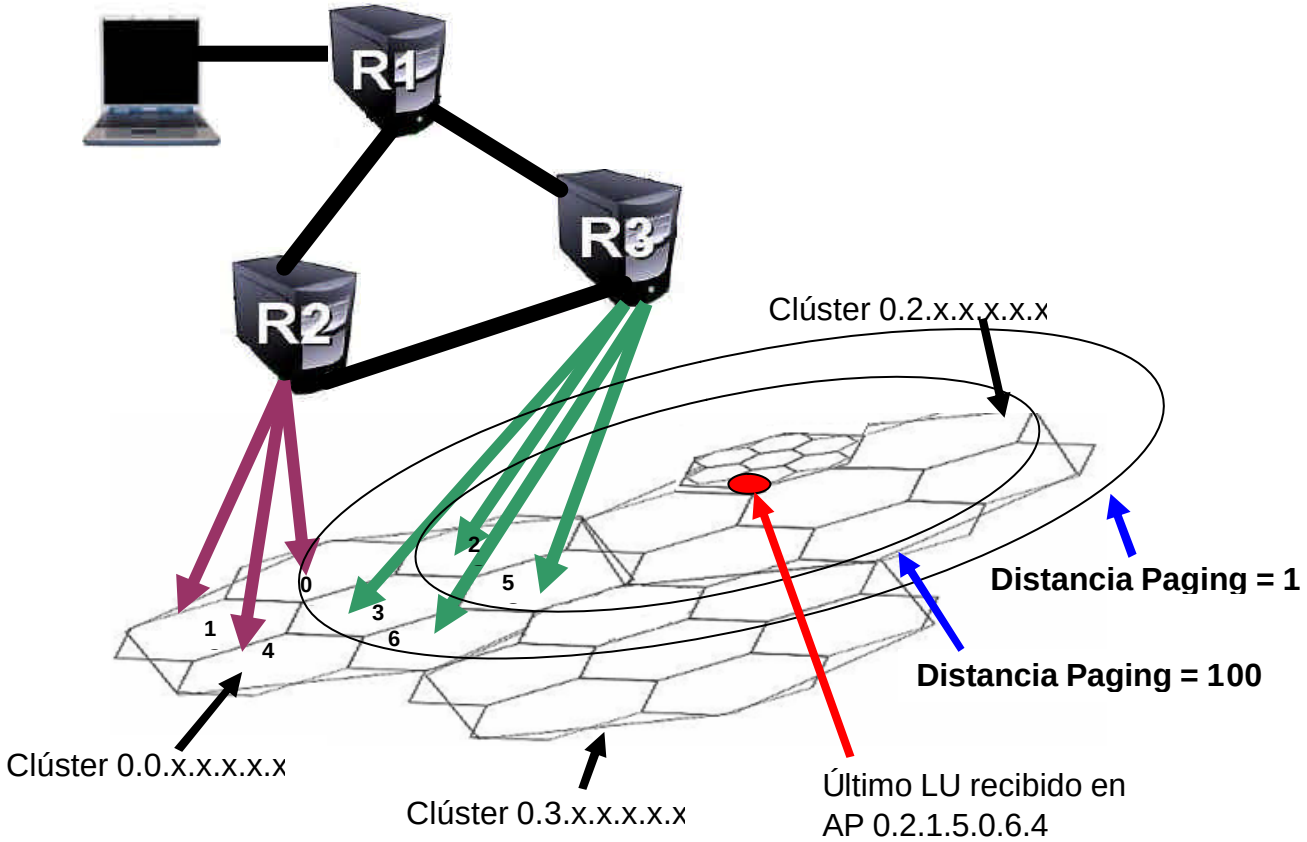


**Figura 3.2.:** Representación del escenario virtual de pruebas

Los routers R1, R2 y R3 son las máquinas A, B y C del laboratorio respectivamente. Las flechas verdes representan las direcciones virtuales de los APs agregadas a la tabla de encaminamiento del router 3 y las flechas moradas representan las direcciones virtuales de los APs agregadas a la tabla de encaminamiento del router 2. La celda que contiene el punto rojo es la celda donde se recibió el último LU, por lo que será el destino de la petición de paging enviada por el portátil, éste actúa de PA. Los círculos marcados representan las áreas de paging establecidas en caso que el parámetro de Distancia de Paging fuese 100 o 150 de radio de las celdas.
A continuación se muestra una tabla resumen de la configuración de los equipos (tabla 3.1).

**Tabla 3.1.** Tabla resumen de la configuración R1, R2, R3 y portátil

| ID de equipo | Interfaz | Ips configuradas | Rutas de acceso |
|---|---|---|---|
| R1 | eth0 | FEDE::A:1/112 | |
| | | FE80::A:1/112 | |
| | eth1 | FEDE::2:1/112 | |
| | | FE80::2:1/112 | |
| | eth2 | FEDE::1:1/112 | |
| | | FE80::1:1/112 | |
| **ID de equipo** | **Interfaz** | **Ips configuradas** | **Rutas de acceso** |
| R2 | eth0 | FEDE::B:2/112 | |
| | | FE80::B:2/112 | |
| | eth1 | FEDE::2:2/112 | |
| | | FE80::2:2/112 | |
| | eth2 | FEDE::3:2/112 | |
| | | FE80::3:2/112 | |
| | Dummy0 | FEDE::8:2/112 | FEC0::/116 |
| | | | FEC0::1000/116 |
| | | | FEC0::2000/116 |
| **ID de equipo** | **Interfaz** | **Ips configuradas** | **Rutas de acceso** |
| R3 | eth0 | FEDE::C:3/112 | |
| | | FE80::C:3/112 | |
| | eth1 | FEDE::1:3/112 | |
| | | FE80::1:3/112 | |
| | eth2 | FEDE::2:3/112 | |
| | | FE80::2:3/112 | |
| | Dummy0 | FEDE::9:3/112 | FEC0::3000/116 |
| | | | FEC0::4000/116 |
| | | | FEC0::5000/116 |
| | | | FEC0::6000/116 |
| **ID de equipo** | **Interfaz** | **Ips configuradas** | **Rutas de acceso** |
| Portátil (PA) | Eth0 | FEDE::A:E/112 | |

Las rutas accesibles son las direcciones virtuales de los APs, creadas a partir de los ID de celda.

## 3.4. Descripción del hardware utilizado

Este apartado explica los componentes que intervienen en el montaje del escenario donde se han realizado todas las pruebas de Mcast. Detallaremos el hardware utilizado, para facilitar la comprensión de algunos resultados que se expondrán más adelante.

El hardware utilizado ha sido:

- 4 PCs, uno de ellos portátil.
- 1 hub ethernet 10/100 de 8 puertos
- 9 tarjetas de red

A continuación mostramos una tabla que recoge las especificaciones técnicas de los pcs utilizados como routers en la maqueta, es importante diferenciar estas características, ya que son influyentes en las pruebas realizadas.

**Tabla 3.2.** Especificaciones

| | R1 | R2 | R3 | Portátil |
|---|---|---|---|---|
| CPU | 1700 GHz | 233 Mhz | 3000 GHZ | 1'6GHz |
| RAM | 256MB | 192 MB | 512 MB | 512MB |
| Tarjetas: | | | | |
| T0 | 3COM 3C905C-TX Tornado / 100MB | 3COM 3C905B-Cyclone / 100Base Tx | Realtek Semiconductor RTL-8139/ 100MB | Realtek RTL 8139/810x Family Fast Ethernet NIC / 1G |
| T1 | 3COM 3C905C-TX Tornado /100 MB | 3COM 3C905C-TX Tornado /100 MB | Realtek Semiconductor RTL-8139/ 100MB | |
| T2 | 3COM 3C905C-TX Tornado/ 100MB | Realtek Semiconductor RTL-8139/ 100MB | Realtek Semiconductor RTL-8139/ 100MB | |

## 3.5. Pruebas realizadas

### 3.5.1. Introducción: Descripción del entorno de trabajo

1. El modo de trabajo de cada uno de los routers será en línea de comandos.

2. Toda la información sobre el estado de los paquetes y el proceso seguido por los paquetes en su enrutamiento se direccionará hacia un fichero, esto reduce el consumo de recursos del sistema en más de la mitad.

3. Únicamente se imprimirá por pantalla el tiempo de proceso de la prueba que se esté realizando.

4. Los procesos que están corriendo en los ordenadores son los mínimos necesarios, es de especial importancia que tengamos muy presente este punto en los ordenadores menos potentes.

   - Al arrancar los ordenadores tenemos corriendo 30 procesos en los diferentes routers.
   - Arrancamos *ripngd* y *zebra* con esto, los procesos se incrementan en dos. Es importante fijarse en que remarcamos los procesos ejecutados en cada máquina a fin de minimizar el consumo de recursos provocados por fuentes innecesarias.

### 3.5.2. Prueba 1: Comprobación del funcionamiento Mcast con memoria dinámica

#### *3.5.2.1. Descripción*

En esta prueba comprobaremos que realmente los cambios realizados en el código al paso a memoria dinámica no han afectado al funcionamiento de Mcast. Para esto, enviaremos un paquete con distancia 100 y otro con distancia 150 y observaremos como se reenvía por las interfaces que realmente toca.

#### *3.5.2.2. Esquema de la prueba 1*

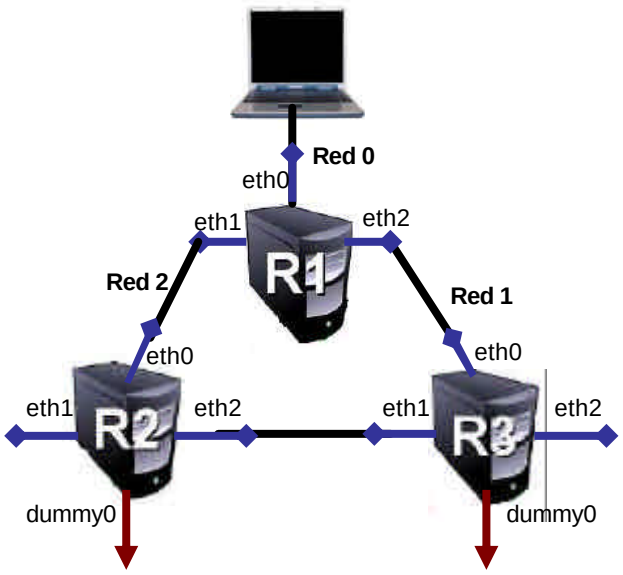


**Figura 3.3.** Esquema de la prueba 1

El router 1 es la puerta de salida hacia la red 0 para el router R1 y R2. A continuación mostramos las rutas de los routers del escenario. Recordamos

que las rutas link-local aparecen como próximo salto como consecuencia del uso del protocolo RIPng.

**Tabla 3.3.** Tabla de rutas del router R1

```
fec0::/116 via fe80::2:2 dev eth1 proto zebra metric 1024 mtu 1500 advmss 1440
fec0::1000/119 via fe80::2:2 dev eth1 proto zebra metric 1024 mtu 1500 advmss 1440
fec0::2000/119 via fe80::2:2 dev eth1 proto zebra metric 1024 mtu 1500 advmss 1440
fec0::3000/119 via fe80::1:3 dev eth2 proto zebra metric 1024 mtu 1500 advmss 1440
fec0::4000/119 via fe80::1:3 dev eth2 proto zebra metric 1024 mtu 1500 advmss 1440
fec0::5000/119 via fe80::1:3 dev eth2 proto zebra metric 1024 mtu 1500 advmss 1440
fec0::6000/119 via fe80::1:3 dev eth2 proto zebra metric 1024 mtu 1500 advmss 1440
fede::1:0/112 dev eth2 metric 256 mtu 1500 advmss 1440
fede::2:0/112 dev eth1 metric 256 mtu 1500 advmss 1440
fede::3:0/112 via fe80::2:2 dev eth1 proto zebra metric 1024 mtu 1500 advmss 1440
fede::8:0/112 via fe80::2:2 dev eth1 proto zebra metric 1024 mtu 1500 advmss 1440
fede::9:0/112 via fe80::1:3 dev eth2 proto zebra metric 1024 mtu 1500 advmss 1440
fede::a:0/112 dev eth0 metric 256 mtu 1500 advmss 1440
fede::b:0/112 via fe80::2:2 dev eth1 proto zebra metric 1024 mtu 1500 advmss 1440
fede::c:0/112 via fe80::1:3 dev eth2 proto zebra metric 1024 mtu 1500 advmss 1440
```

**Tabla 3.4.** Tabla de rutas del router R2

```
fec0::/116 via fe80::2:2 dev eth1 proto zebra metric 1024 mtu 1500 advmss 1440
fec0::1000/119 dev dummy0 proto zebra metric 1 mtu 1500 advmss 1440
fec0::2000/119 dev dummy0 proto zebra metric 1 mtu 1500 advmss 1440
fec0::3000/119 via fe80::2:3 dev eth2 proto zebra metric 1024 mtu 1500 advmss 1440
fec0::4000/119 via fe80::2:3 dev eth2 proto zebra metric 1024 mtu 1500 advmss 1440
fec0::5000/119 via fe80::2:3 dev eth2 proto zebra metric 1024 mtu 1500 advmss 1440
fec0::6000/119 via fe80::2:3 dev eth2 proto zebra metric 1024 mtu 1500 advmss 1440
fede::1:0/112  via fe80::2:1 dev eth1 proto zebra metric 1024 mtu 1500 advmss 1440
fede::2:0/112 dev eth1 metric 256 mtu 1500 advmss 1440
fede::3:0/112 dev eth2 metric 256 mtu 1500 advmss 1440
fede::8:0/112 dev dummy0 proto zebra metric 1 mtu 1500 advmss 1440
fede::9:0/112 via fe80::2:3 dev eth2 proto zebra metric 1024 mtu 1500 advmss 1440
fede::a:0/112 via fe80::2:1 dev eth1 proto zebra metric 1024 mtu 1500 advmss 1440
fede::b:0/112 dev eth0 metric 256 mtu 1500 advmss 1440
fede::c:0/112 via fe80::2:3 dev eth2 proto zebra metric 1024 mtu 1500 advmss 1440
```

**Tabla 3.5.** Tabla de rutas del router R3

```
fec0::/116 via fe80::3:2 dev eth2 proto zebra metric 1024 mtu 1500 advmss 1440
fec0::1000/119 via fe80::3:2 dev eth2 proto zebra metric 1024 mtu 1500 advmss 1440
fec0::2000/119 via fe80::3:2 dev eth2 proto zebra metric 1024 mtu 1500 advmss 1440
fec0::3000/119 dev dummy0 metric 1 mtu 1500 advmss 1440
fec0::4000/119 dev dummy0 metric 1 mtu 1500 advmss 1440
fec0::5000/119 dev dummy0 metric 1 mtu 1500 advmss 1440
fec0::6000/119 dev dummy0 metric 1 mtu 1500 advmss 1440
fede::1:0/112 dev eth1 metric 256 mtu 1500 advmss 1440
fede::2:0/112 dev eth2 metric 256 mtu 1500 advmss 1440
fede::3:0/112 via fe80::1:1 dev eth1 proto zebra metric 1024 mtu 1500 advmss 1440
fede::8:0/112  via  fe80::3:2  dev  eth2  proto  zebra  metric  1024  mtu  1500  advmss  1440
fede::9:0/112 dev dummy0 metric 256 mtu 1500 advmss 1440
fede::a:0/112 via fe80::1:1 dev eth1 proto zebra metric 1024 mtu 1500 advmss 1440
fede::b:0/112 via fe80::3:2 dev eth2 proto zebra metric 1024 mtu 1500 advmss 1440
fede::c:0/112 dev eth0 metric 256 mtu 1500 advmss 1440
```

### *3.5.2.3. Comportamiento teórico*

En la siguiente imagen mostramos el comportamiento que esperamos obtener al inyectar un paquete de paging desde el PA según la información de rutas que muestran las tablas de encaminamiento de los routers.
Los paquetes de paging para realizar estas pruebas se han enviado con distancias de 100 y 150.

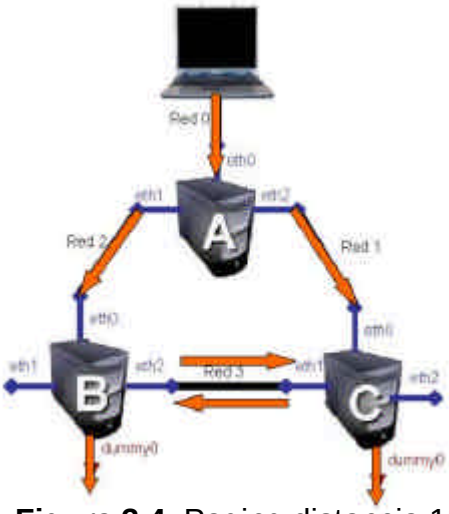


**Figura 3.4.** Paging distancia 150

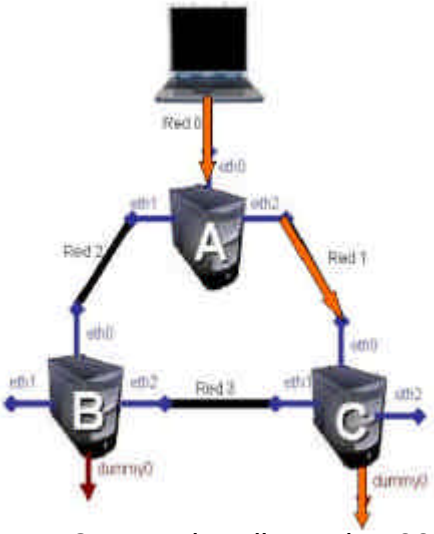


**Figura 3.5.** Paging distancia 100

### *3.5.2.4. Resultado*

En el caso de enviar un paquete de paging con D=150 observamos que el paquete es reenviado por las dos interfaces del R1 que están conectadas a los otros dos routers.
En el caso de R2 y R3 estos reenvían a sus interfaces dummy0 puesto que estas están dentro del área de paging, asimismo estos routers reenviaran el paquete por la interfaz 2, puesto que tanto el uno como el otro verán como válidos sus AP's aunque el software Geopaging lo descartará al no llegar por la interfaz parent.

Para el caso de paging con D=100 observamos como el R1 reenvía únicamente por su interfaz eth2 ya que él ve como válidas únicamente las rutas que anuncia el R3.

Este resultado era el esperado y con esto se comprueba que el funcionamiento del código después de las modificaciones hechas sobre la memoria dinámica es correcto.

### 3.5.3. Prueba 2: Análisis del consumo de memoria

#### *3.5.3.1. Descripción*

Esta prueba consiste en mirar cómo y en cuánto aumenta el consumo de memoria de las diferentes máquinas al ejecutar Mcast no liberando el espacio de memoria previamente reservado con las funciones malloc() y realloc(), enviando paquetes de paging y después analizar el consumo de memoria de Mcast en ejecución durante 30 minutos.

#### *3.5.3.2. Esquema del escenario*

Esta prueba no requiere de ninguna topología de red especial ni diferente a la mostrada anteriormente en la figura 3.4, en este análisis tomamos las máquinas individualmente.

#### *3.5.3.3. Resultado*

Al poner en marcha los ordenadores el primer comando que ejecutamos para observar los procesos que están corriendo en el sistema así como el consumo de cada uno de ellos es el comando: #top, con este comando observaremos el estado del sistema, cantidad de procesos corriendo, consumo de memoria, consumo de cpu de cada uno de ellos y el consumo total entre otras características menos importante para éste análisis.

A continuación mostramos una tabla con información sobre el consumo de cpu y de memoria de Mcast, estos datos son al iniciarse el programa, es importante fijarse que a medida que Mcast lleva funcionando durante más tiempo el consumo de memoria aumenta, este dato se muestra en las tablas 3.6 y 3.7„ esto se debe a que cada vez que Mcast ejecuta las funciones malloc() y realloc() reserva memoria del sistema y por lo tanto el consumo de cpu necesaria aumenta, esto no es tan perjudicial en la máquina más potente y este aumento es casi despreciable, pero en la máquina menos potente de la maqueta, esto se nota bastante y es un hecho que se tiene que tener en cuenta.

**Tabla 3.6.** Consumo de memoria Mcast al iniciar el sistema

| Router | % cpu usuario | % cpu sistema | Nº procesos total | % MEM Mcast | % CPU Mcast (inicio programa) |
|---|---|---|---|---|---|
| *Sin ejecutar Mcast con ripng zebra* | | | | | |
| R1 | 0.0 – 0.3 | 0.3-0.7 | 33 | | |
| R2 | 0.0 – 0.3 | 0.3-0.7 | 32 | | |
| R3 | 0.0-0.3 | 0.3-0.7 | 36 | | |
| *Mcast en ejecución* | | | | | |
| R1 | 0.0-0.3 | 0.3-0.7 | 34 | 0.4 | |
| R2 | 0.3-1.3 | 0.7-1.6 | 33 | 1.0 | 4% |
| R3 | 0.0-0,2 | 0.0-0.2 | 37 | 0.2 | |

Después de una hora monitorizando con top en R2, debido a no liberar el espacio de memoria reservado continuamente, éste aumenta hasta un 7% de memoria consumida.

**Tabla 3.7**. Monitorización R1

| Tiempo [minutos] | %MEM Mcast |
|---|---|
| 0 | 0.3 |
| 5 | 0.4 |
| 8 | 0.5 |
| 11 | 0.6 |
| 14 | 0.7 |
| 17 | 0.8 |
| 20 | 0.9 |
| 24 | 1 |
| 27 | 1.1 |
| 31 | 1.2 |
| 36 | 1.3 |

**Tabla 3.8.** Monitorización R3

| Tiempo [minutos] | %MEM Mcast |
|---|---|
| 0 | 0.2 |
| 2 | 0.3 |
| 8 | 0.4 |
| 16 | 0.5 |
| 19 | 0.6 |

Únicamente mostramos algunos datos tomados en las máquinas con menor y mayor capacidad para observar las diferencias y ver cómo puede afectar el hecho de no liberar el espacio en memoria. Estas mismas pruebas se han realizado añadiendo al programa las funciones `free()` comentadas en el capitulo anterior y los resultados obtenidos han sido que esta utilización incremental de memoria por parte de Mcast no se daba.

Finalmente después de analizar todos los resultados anteriores podemos decir que el consumo de cpu y memoria necesarios para correr Mcast en cualquier tipo de máquina son muy pequeños y aún no liberando el espacio reservado previamente con las funciones `malloc()` y `realloc()` esta utilización continua siendo despreciable, por lo que Mcast no está limitado a ejecutarse en máquinas con características muy potentes, sino que en máquinas como puede ser el R2, que es una máquina no muy potente Mcast no consume exageradamente y puede ejecutarse sin problemas.

### 3.5.4. Prueba 3: Enrutado de paquetes con un protocolo unicast: RIPng

#### *3.5.4.1. Descripción*

El objetivo de esta prueba es comparar el tiempo de proceso y transmisión obtenido en cada router utilizando un protocolo de enrutado unicast, como es RIPng[1], con el protocolo multicast, Mcast, implementado a lo largo del trabajo.

Las pruebas realizadas han sido envíos de pings entre los routers que componen la topología de la red, éstos medirán la diferencia de tiempo entre la entrada de los paquetes por una interfaz y la salida por la otra interfaz. En las pruebas de enrutado multicast se inyectará un paquete dentro de la red y se medirá el tiempo de proceso de los distintos routers, estas pruebas se describen con más detalle en apartados posteriores.
Para efectuar las medidas usaremos el programa tcpdump, como se explica en [1], tcpdump es el programa oficial de la librería libpcap para la captura de cualquier tipo de paquete interpretándolo de tal manera que imprime por pantalla los datos más importantes según el protocolo del paquete, que en nuestro caso será IPv6. La página oficial de desarrollo de proyecto es: http://www.tcpdump.org

#### *3.5.4.2. Esquema de la prueba 3*

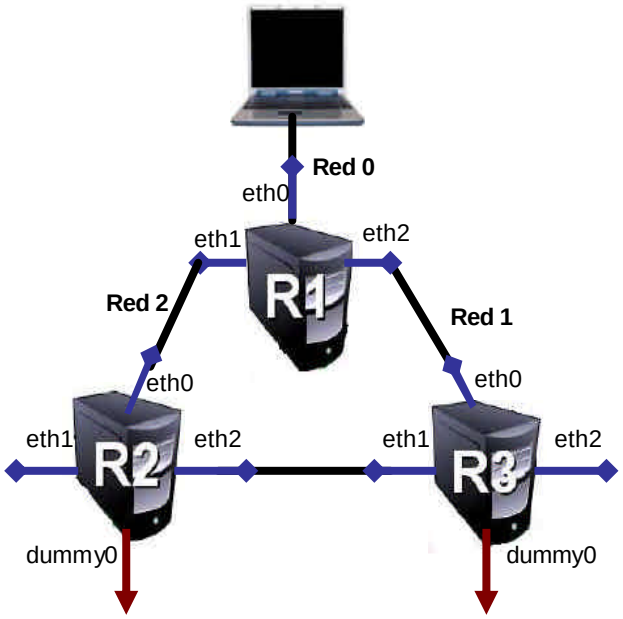


**Figura 3.6.** Esquema de la prueba 3

[1] En el anexo I, Instalación del código, se explica detalladamente cuál es la versión de RIPng utilizada.

#### *3.5.4.3. Resultados*

La siguiente tabla resume el tiempo medio obtenido en el envío de los pings. El número total de repeticiones ha sido de 20 por cada una de las máquinas.

**Tabla 3.8.** Media del tiempo de enrutado unicast

| | R1 | R2 | R3 |
|---|---|---|---|
| **Enrutado Unicast** | 15,45±5 µs | 54,25±6 µs | 6,3±3 µs |

En puntos anteriores se han detallado las características de las máquinas utilizadas en la maqueta del escenario, estas características son fundamentales para comprender estos tiempos ya que el tiempo de proceso obtenido depende de la capacidad de los routers, de ahí que se observen tiempos tan diferentes con variaciones incluso en un orden de 10 veces más, algo a tener en cuenta para más adelante comprender también las diferencias temporales obtenidas dependiendo de los routers donde se toman las medidas. Estos resultados servirán para tener un baremo con los que poder contrastar los tiempos obtenidos en el enrutado multicast.

Como resumen de esta prueba podemos decir que con en un proceso tan simple como es el envío de un ping se obtienen medidas temporales del orden de microsegundos, tiempos realmente muy pequeños pero que son lógicos ya que un ping se procesa directamente en el kernel y este tipo de procesos son más rápidos que otros que deban ser procesados por el sistema.

### 3.5.5. Prueba 4: Enrutado de paquetes con un protocolo multicast: Mcast

#### *3.5.5.1. Descripción*

Esta prueba pretende medir el tiempo de proceso y transmisión de un paquete, es decir, medir el tiempo que transcurre desde que un paquete entra por una interfaz hasta que sale por otra, con la aplicación tcpdump siguiendo el mismo esquema que se ha realizado en la prueba 3. Una vez se tengan los resultados se podrán comprar con los tiempos obtenidos utilizando la introducción de trazas en el código.

#### *3.5.5.2. Esquema del escenario*

El esquema del escenario utilizado es exactamente el mismo que para realizar la prueba anterior, tal y como podemos observar en la figura 3.6.

#### *3.5.5.3. Resultado*

La prueba se ha realizado mediante 10 repeticiones y los resultados obtenidos son los siguientes:

**Tabla 3.9.** Tiempo de proceso medido con tcpdump

| | R1 | R2 | R3 |
|---|---|---|---|
| **Enrutado Multicast** | 435.2±37 µs | 249±54 ms | 287.25±12 µs |

Fijénse que en el router dos se han obtenido tiempos del orden de milisegundos mientras que en los otros dos este orden es de microsegundos.

Comparando estos resultados con los de la prueba anterior podemos decir que Mcast funciona peor que un protocolo de enrutado unicast, en nuestro caso RIPng, es decir Mcast tarda mucho más en encaminar los paquetes que un protocolo unicast, este hecho hace que nos preguntemos y busquemos el porqué de estos resultados, qué hace que obtengamos con Mcast estas medidas tan superiores a las pruebas anteriores. Para poder resolver está duda necesitamos desglosar el tiempo total de proceso y transmisión de los paquetes en trazas y observar donde se producen los cuellos de botella del programa que lo ralentizan.

Cabe recalcar que como se observará en las siguientes pruebas los resultados de tiempo de proceso medidos con tcpdump y los medidos con trazas difieren. El tiempo de proceso medido con tcpdump es menor que el tiempo de proceso obtenido mediante trazas en ordenadores potentes mientras que por el contrario si se utiliza en ordenadores poco potentes el tiempo proporcionado por tcpdump es bastante mayor al proporcionado por las trazas. Aunque no se han determinado las causas exactas de este comportamiento creemos que puede ser debido a por un lado, un mayor consumo de memoria por parte de tcpdump que haría que se ralentizase toda la maquina en conjunto estas no son muy potentes; y por otro los problemas de exactitud de la librería empleada en las trazas, aspecto este último que será comentado más adelante Comentamos esta observación aquí para que a continuación cuando vean los resultados obtenidos con trazas y no coincidan con exactitud los resultados sepan a qué es debido.

El resto de pruebas que se muestran a partir de este punto tienen como principal objetivo encontrar el cuello de botella de Mcast que relentiza el software.

### 3.5.6. Prueba 5: Medida del tiempo de filtrado de un paquete

#### *3.5.6.1. Descripción*

Para la realización de esta prueba se han introducido marcas temporales en el código para monitorizar con mayor exactitud qué tiempo de filtrado consume. Estas marcas temporales se han realizado utilizando la librería `time.h` que viene de serie en el kernel de Linux. Para obtener más información acerca de esta librería pueden acceder a la dirección: http://www.opengroup.org/onlinepubs/007908799/xsh/time.h.html.
Un aspecto a tener en cuenta es la resolución de la librería `time.h`, esta es del orden de microsegundos y a partir de las centenas de microsegundos los resultados obtenidos pueden ser no muy exactos, esto se debe a que esta

librería se basa en el reloj interno de los SO y debido a que estos tienen desviaciones temporales hacen menos fiable la resolución de *time.h*

El tiempo de filtrado de un paquete se mide desde que un nodo recibe un paquete, lo analiza y verifica que cumpla las condiciones para que sea enrutado por GEOPAG.

Para realizar esta medida se ha introducido una marca temporal en la primitiva *PCAP_recibe_msg* y otra justo antes de llamar a la primitiva *GEOPAG_router*. Con estas marcas obtenemos el tiempo de proceso de cada paquete de PAGING enviado.

Para realizar esta prueba se ha enviado un paquete de distancia de paging 150 a la dirección multicast ff05::1:1a34. Se ha escogido esta distancia debido a que ahora la topología de la red representada tiene clústeres de nivel superior y necesitamos enviar paging de distancia superior a 125 para que todos los routers procesen los paquetes y de esta manera poder observar el efecto en todos ellos.

La estimación de esta prueba se basa en muestras de 20 repeticiones, consideramos 20 repeticiones un número suficiente para analizar este comportamiento.

### *3.5.6.2. Esquema de la prueba 5*

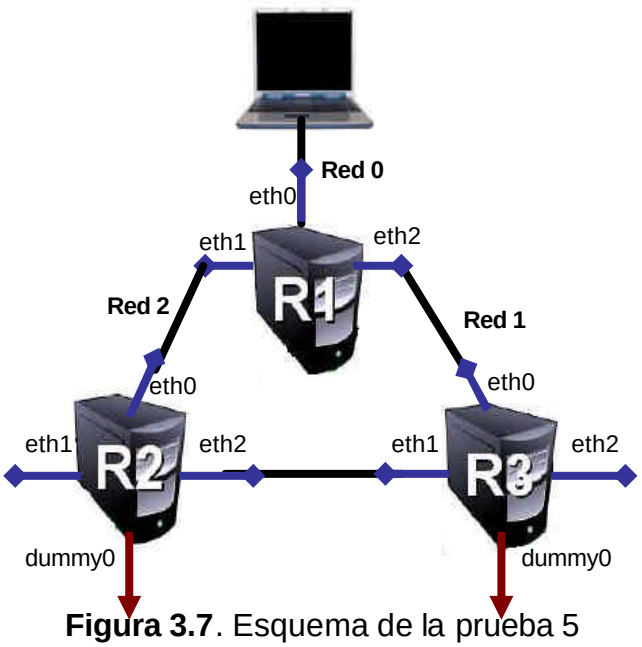


**Figura 3.7**. Esquema de la prueba 5

### *3.5.6.3. Comportamiento teórico*

En este análisis analizamos el tiempo de filtrado individual de cada una de las máquinas, para ello se envían paquetes de paging de distancia 150, de esta manera aseguramos que el paquete sea procesado por todos los routers y comprobamos cuánto tarda cada uno de ellos en analizar el paquete y decidir que se trata de un paquete de paging válido.

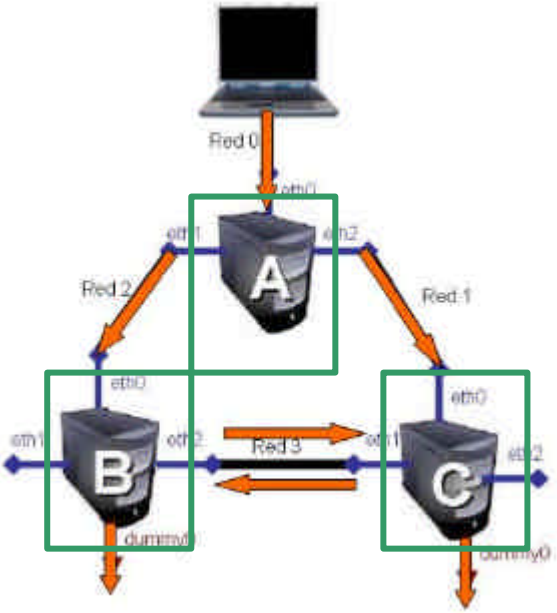


**Figura 3.8.** Esquema de la prueba 5

### *3.5.6.4. Resultados*

A continuación mostramos la tabla con las medidas realizadas sobre el tiempo de filtrado de cada nodo y el gráfico representativo de estas medidas. Para obtener más información sobre otras medidas realizadas con diferentes muestras y observar las diferencias obtenidas ver el archivo pruebas_*Mcast.xls*

**Tabla 3.10.** Tiempo de filtrado de un paquete (µs)

| | R1 | R2 | R3 |
|---|---|---|---|
| Tiempo Medio | 165.7±31 | 553.7±115 | 94.9±27 |

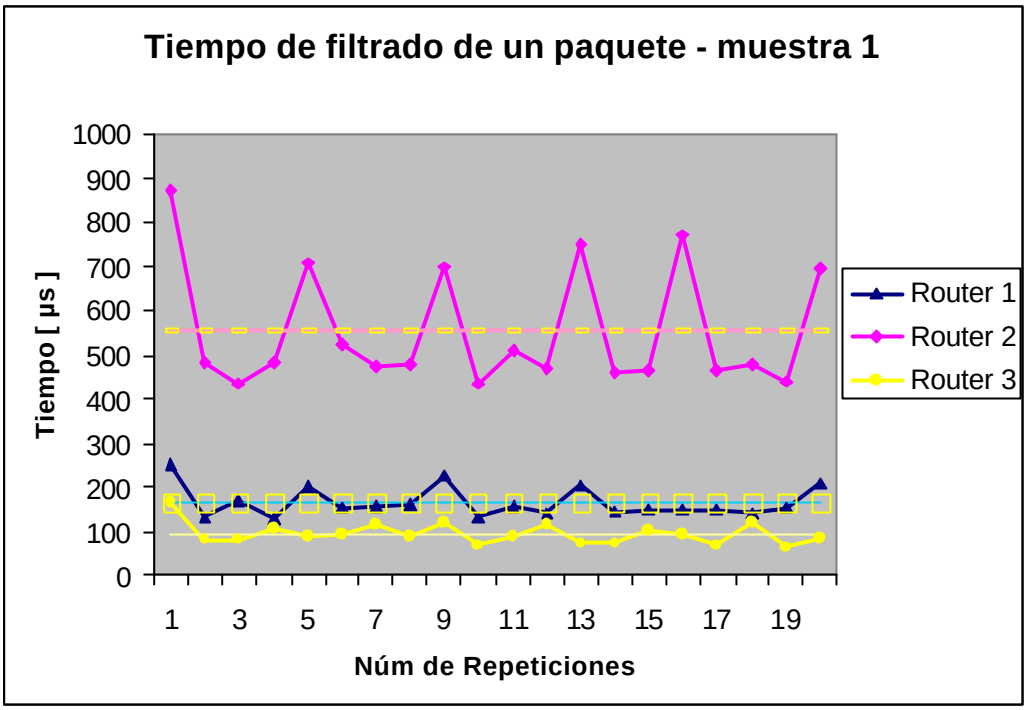


**Figura 3.9.** Tiempo de filtrado de un paquete

El gráfico muestra los resultados obtenidos en esta prueba, las líneas señaladas con un rombo, pertenecen a las medidas tomadas en el router 2, las líneas más oscuras las medidas tomadas en el router 1 y las líneas más claras representan al router 3. Se observan dos tipos de líneas de cada tipo, una con picos y otras líneas rectas, éstas últimas representan el valor medio que toma el tiempo de filtrado en cada router.

Analizando los resultados obtenidos podemos apreciar como estos tiempos son bastante reducidos, del orden de microsegundos, comparados con los tiempos obtenidos en los pings de la prueba en el enrutado unicast hay diferencias bastante notables pero que tienen su lógica. Anteriormente se ha comentado como un ping es procesado directamente por el kernel por lo que es de entender que el tiempo obtenido sea mucho menor que el obtenido en esta prueba. Los paquetes capturados en ésta prueba se procesan por diferentes librerías del sistema, no es el kernel el que los procesa directamente por lo que forzosamente las medidas temporales han de ser mayores, pero que hayan salido resultados del orden de microsegundos es un buen indicador de Mcast para demostrar como su rendimiento se puede considerar muy positivo.

Si analizamos estos resultados desde la perspectiva de los componentes hardware del sistema, las especificaciones técnicas de las máquinas del laboratorio son muy diferentes en cuanto a capacidades y por lo tanto esperábamos encontrarnos frente a resultados muy variados entre los routers. Estos resultados corresponden perfectamente con las especificaciones técnicas de cada máquina, es decir, por ejemplo los tiempos obtenidos por el router 2 son los dobles que los obtenidos con el router3. La máquina del laboratorio que corresponde al router 3 es un PENTIUM IV a 3000GHz en cambio el router 2 es un PENTIUM IV a 1700 GHz, por lo tanto es totalmente

lógico y entra dentro de la normalidad que estos tiempos de filtrado sean el doble de un ordenador a otro.

Esta particularidad de diferencias de resultados entre las máquinas, se darán en la mayoría de las pruebas en las que intervenga la capacidad de los sistemas para procesar los paquetes.

### 3.5.7. Prueba 6: Medida del tiempo de routing y forwarding de un paquete

#### *3.5.7.1. Descripción*

Para la realización de estas medidas se ha seguido el mismo proceso de introducción de marcas temporales que para la prueba anterior. Para poder medir el tiempo de routing y forwarding se han introducido marcas temporales en *GEOPAG_router.*

Primero tenemos que identificar que se entiende por tiempo de routing y forwarding de un paquete. Consideramos los tiempos de routing y forwarding el tiempo desde que un router mira que el paquete haya llegado por la interfaz parent y si tiene rutas disponibles, si se cumplen las condiciones anteriores, entonces reenvía el paquete.

El código de *GEOPAG_router* se divide en los cuatro procesos explicados anteriormente, en cada uno de ellos hemos introducido una marca temporal diferente que los identificará en la ejecución del código y de esta manera poder medir el tiempo necesario en cada una de estas partes.

Exactamente hemos introducido marcas temporales en los siguientes pasos del código de *GEOPAG.c:*

PASO 1: Comprobación del TTL del paquete
PASO 2: QUE EL PAQUETE HAYA LLEGADO POR LA INTERFAZ PARENT
PASO 3: MIRA LAS RUTAS DISPONIBLES PARA EL ENVÍO DEL PAQUETE
PASO 4: FORWARDING DEL PAQUETE

Es importante comentar que la última marca temporal se ha introducido fuera de un bucle que depende del número de interfaces por las que tenga que reenviar el paquete, el tiempo obtenido en esta marca se multiplicará. Es decir, las medidas que mostraremos a continuación se han realizado para un caso particular, enviado un paquete que únicamente los routers tuvieran que inyectar por una única interfaz, en caso de tener que inyectarse por ejemplo, por 2 interfaces, este tiempo se multiplicaría por el doble.

### *3.5.7.2. Esquema de la prueba 6*

El siguiente esquema muestra la medida temporal realizada en cada nodo.

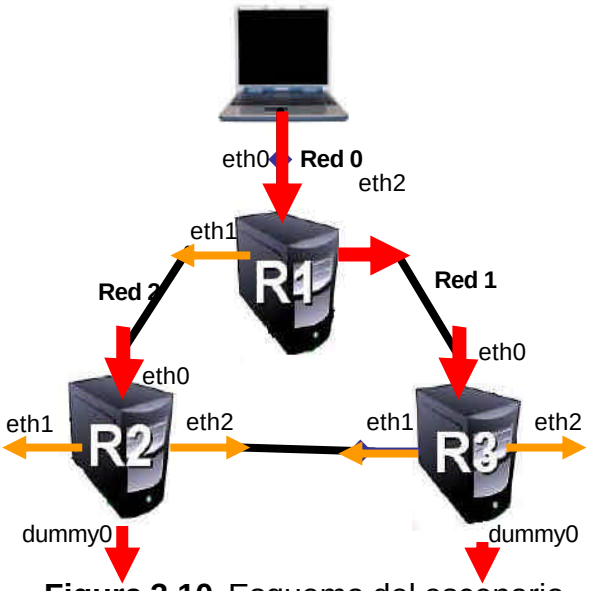


**Figura 3.10.** Esquema del escenario

Las flechas naranjas, más finas, representan que la medida de tiempo se ha realizado de forma individual, es decir, se ha medido lo que tarda en entrar un paquete por una interfaz e inyectarse por otra interfaz.

### *3.5.7.3. Resultados*

A continuación mostramos el gráfico que representa el total de medidas realizadas sobre los diferentes routers, en este caso, únicamente mostramos el gráfico, para obtener los datos precisos sobre los diferentes tiempos obtenidos ver el archivo pruebas_Mcast.xsl, hoja de “ROUTING i FORWARDING”.

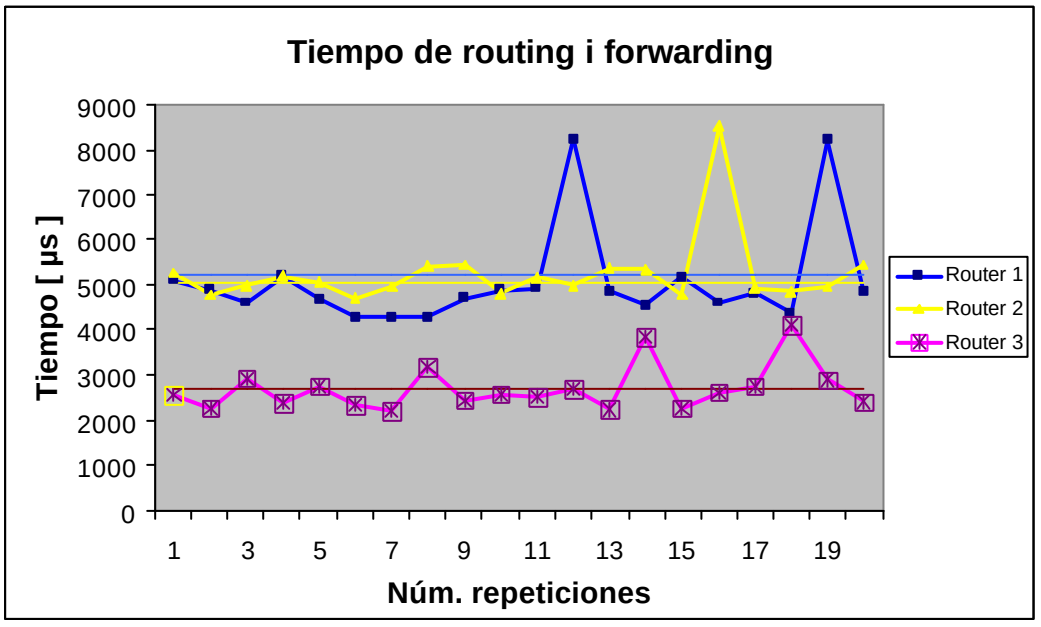


**Figura 3.11.** Tiempo total de routing i forwarding

El gráfico representa la suma de los tiempos obtenidos en los cuatro pasos explicados anteriormente. Como en el gráfico anterior las líneas horizontales representan los tiempos medios de cada uno de los routers. A continuación se desglosan los tiempos medios obtenidos en cada uno de los pasos para poder apreciar donde realmente Mcast tarda más en realizar las funciones y de esta manera poder determinar posibles mejoras y cambios posteriores.

**Tabla 3.11.** Tiempos medios de los routers: desglose de pasos

| Tiempos medios [microsegundos] | | | |
|---|---|---|---|
| **PASO** | **R1** | **R2** | **R3** |
| **1** | 2,55 | 7,3 | 1,55 |
| **2** | 12,35 | 54,8 | 8,15 |
| **3** | 112,4 | 268,95 | 63,2 |
| **4** | 4944,25 | 4915,7 | 2624,6 |

En el primer gráfico (figura 3.12) que aparece a continuación se representa el desglose de marcas temporales en *GEOPAG.c*. No se incorpora el tiempo invertido en el paso 1, ya que el tiempo dedicado en este proceso es muy reducido y así se pueden apreciar mejor el resto de valores, que son más importantes en cuanto a magnitud. En el segundo (figura 3.13) se representa el porcentaje de tiempo dedicadazo a cada una de las trazas introducidas en GEOPAG.

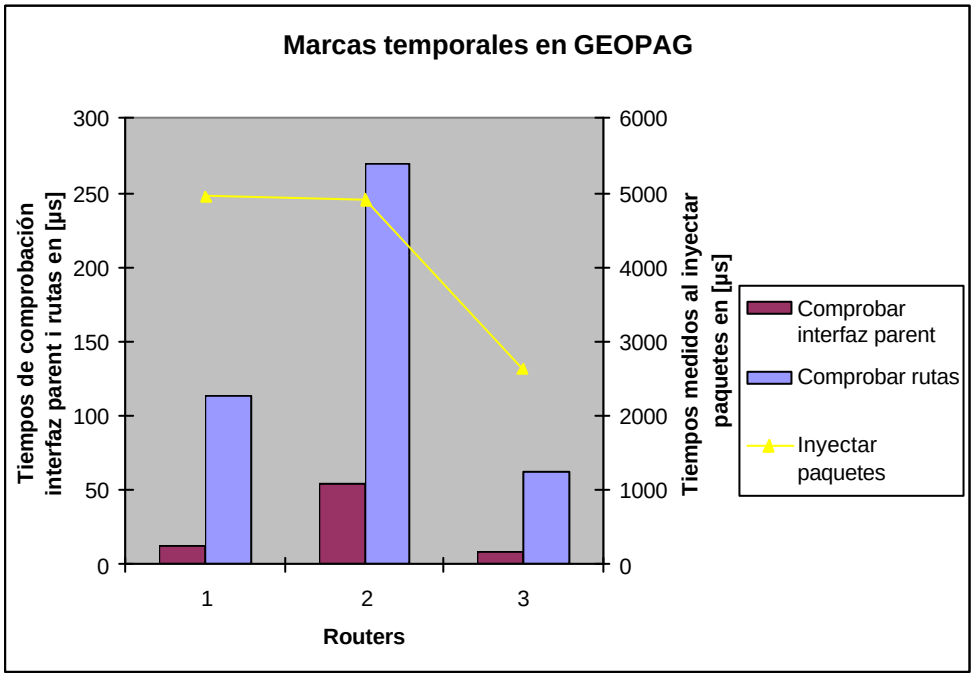


**Figura 3.12**. Gráfico desglose de marcas temporales en GEOPAG

En la figura 3.12 se observan dos ejes diferentes de tiempo, cada uno de ellos representa una escala temporal diferente, esto se ha realizado de este modo para que se vea perfectamente el desglose de tiempos invertidos en los pasos 2, 3 y 4. El eje vertical de la izquierda corresponde a la escala de tiempos de la

comprobación de la interfaz parent y de las rutas, el eje de la derecha corresponde al tiempo dedicado al inyectar paquetes.

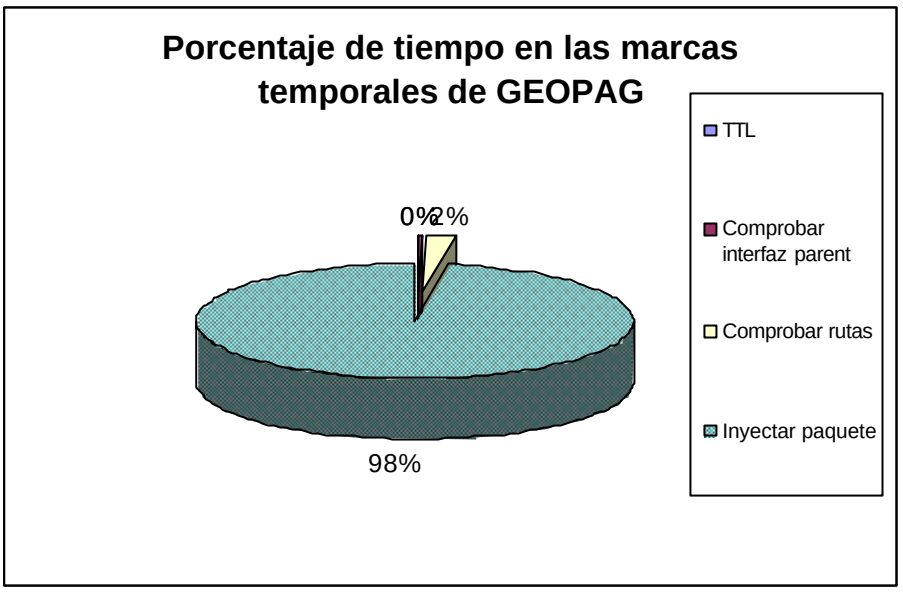


**Figura 3.13**. Gráfico porcentaje de tiempo en el desglose de las marcas temporales en *GEOPAG.c*

Como en el caso de la prueba anterior se observa la importancia del hardware en los pasos 1,2 y 3. En el paso 4 esta importancia no es tan evidente puesto que tanto R1 como R2 obtienen resultados muy similares que representan aproximadamente el doble de los obtenidos por R3. Estas diferencias podrían ser motivadas por la interacción con el driver de las tarjetas de red o al propio hardware de estas. Como se observa en Tabla 3.2, las tarjetas son prácticamente iguales para R1 y R2, del mismo fabricante y 3 de ellas del mismo modelo, y de un fabricante diferente para el R3. También esta similitud en esta prueba entre los R2 y R1 se puede deber a que en los tiempos medidos aquí la CPU de los sistemas no es un factor determinante en los resultados, ya que en esta parte los paquetes ya se han leído y volcado en memoria que son los procesos donde la CPU interviene más, en esta parte del programa intervienen otros factores externos dependientes de las librerías utilizadas.

En los gráficos anteriores se puede observar que el paso donde el protocolo invierte más tiempo, es precisamente el 4, llegando a consumir el 98% del tiempo total.
Hay que señalar que estos resultados son correctos en términos relativos pues nos permiten determinar proporcionalmente que pasos son los más costosos en tiempo pero que en términos absolutos no lo son tanto, puesto que la librería que utilizamos para la medida de las trazas es la *time.h,* bastante imprecisa a causa del kernel del SO que es bastante impreciso cuando ha de medir por debajo de las centenas de microsegundos.

Estos resultados son muy significativos ya que permiten identificar a un proceso mucho más lento que los demás y con marcas de tiempo dentro de la resolución de time.h por lo que queda perfectamente determinado el cuello de

botella de Mcast. Este punto crítico coincide con la utilización de las librerías libnet, por lo tanto podemos decir que éstas son el auténtico cuello de botella del código Mcast.

Que sea este punto el cuello de botella del programa es interesante porque abre la posibilidad de solucionar el problema de una manera relativamente sencilla, gracias a la implementación en forma modular del código Mcast, con solo cambiar el método de inyección de los paquetes por otra opción, por ejemplo que trabaje a más bajo nivel, como son los raw sockets, los routing sockets o utilizar los packet socket de la librería *packet.h*. Con estas opciones se podrían obtener tiempos más reducidos que podrían aproximarse al mínimo necesario para el envío de un paquete de paging situado en lo 8 µs, ya que como se ha comentado todos estos trabajan a más bajo nivel que libnet, directamente en el kernel, aspecto que complicaría la programación de una posible solución basada en estas opciones.

Dejando un lado el impacto negativo de libnet, podemos fijarnos en los resultados para los demás pasos. .De estos se puede decir que las medidas de tiempo obtenidas no son desorbitadas pudiéndose considerar buenas. Como se comentó anteriormente, si en un ping se obtienen tiempos del orden de decenas de microsegundos, y es un proceso que recae exclusivamente en el kernel, obtener tiempos del orden de centenas de microsegundos en procesos en los que intervienen diferentes librerías y para un protocolo más complejo es un dato positivo respecto al rendimiento de Mcast y pensando en su utilización en una red real.

### 3.5.8. Resultados generales prueba 5 y prueba 6

En este punto analizaremos el tiempo total de proceso de un paquete, para ello necesitamos la información recopilada en las pruebas 5 y 6. En ellas hemos calculado el tiempo en unas determinadas partes del proceso, a continuación se mostrará la suma total de cada una de ellas para poder observar realmente el tiempo de proceso de un paquete de paging.

El gráfico resultante de la suma del tiempo de filtrado, más el tiempo total de routing y forwarding es el siguiente:

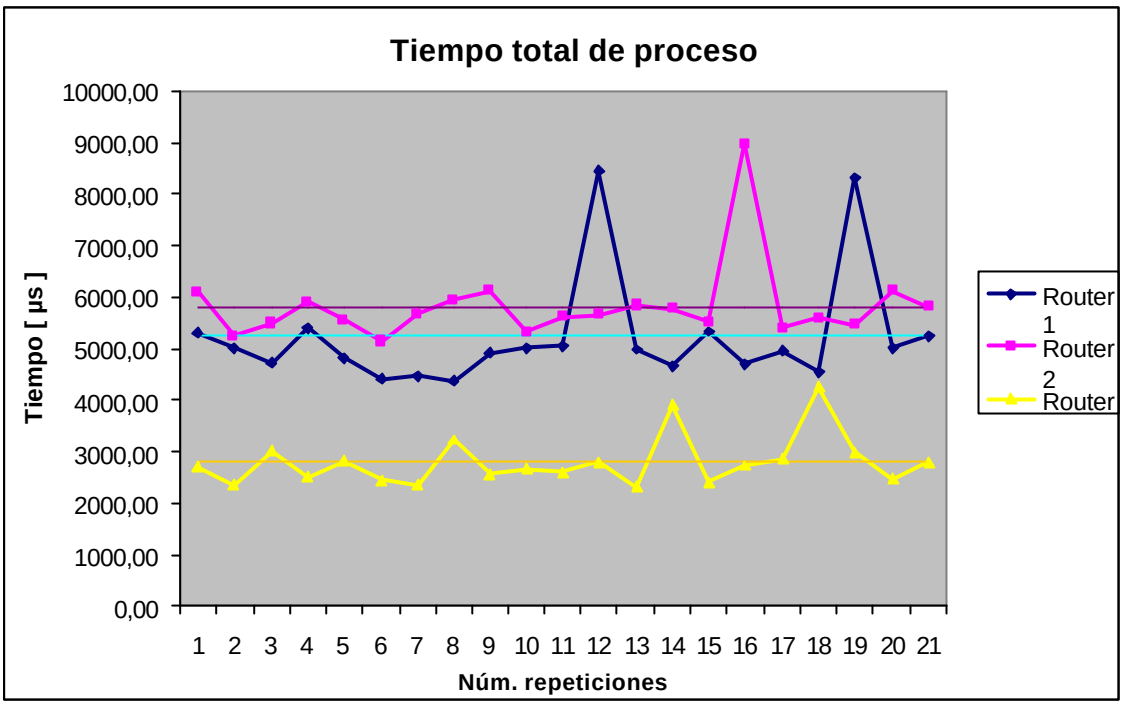


**Figura 3.13.** Tiempo total de proceso de los tres routers

En la gráfica las líneas horizontales representan los tiempos medios de cada router, en la siguiente tabla mostramos el valor exacto de estos tiempos. :

**Tabla 3.12.** Tiempo de proceso medio

| | R1 | R2 | R3 |
|---|---|---|---|
| Tiempo medio total | 5224,25 µs | 5821,85 µs | 2802,05 µs |

Con estos datos podemos calcular la cadencia máxima de inyección de paquetes soportada por los routers en el sistema, para posteriormente realizar las pruebas de eficiencia.

Si analizamos el sistema, podemos ver fácilmente que el router que está limitando el sistema es el router 2, por lo tanto será suficiente con calcular su cadencia máxima de inyección de paquetes soportada para analizar de este modo la eficiencia y capacidad del sistema.

La cadencia del router 2, calculada a partir de los tiempos medios anteriores es de: 171,8 paquetes por segundo. Esto quiere decir que el sistema empezará a perder paquetes a partir de que se inyecten paquetes a esa tasa. En la siguiente prueba insistiremos en este punto y comprobaremos realmente como reacciona el sistema ante la inyección de paquetes a tasas muy elevadas.

### 3.5.9. Prueba 7: Medida de la capacidad de proceso de los sistemas

#### *3.5.9.1. Descripción*

La prueba 7 consiste en medir la capacidad de proceso de los diferentes routers en el envío de mensajes de paging a diferentes velocidades, para de esta manera poder analizar la cantidad de mensajes que los routers pueden llegar a procesar sin tener pérdidas y analizar a partir de qué velocidad los routers empiezan a padecer las consecuencias de las elevadas tasas.
Para poder realizar esta prueba, se ha creado un pequeño programa generador de tráfico, al que le indicamos el periodo de cadencia en el que debe enviar los paquetes.

Se ha calculado de forma manual la cantidad de paquetes que representa cada tasa y de esta manera se ha introducido en el programa.

El escenario a representar es el mismo que en las pruebas anteriores, el PA envía paquetes de distancia de paging 150, para que de esta forma sea procesado por todos los routers, la diferencia con los casos anteriores es que ahora los paquetes se envían de golpe, ya que en esta prueba lo que importa es la capacidad de los sistemas y por lo tanto no tenemos que ir midiendo tiempos sino comprobando resultados.

#### *3.5.9.2. Resultados*

A continuación se muestra la tabla de eficiencias obtenidas enviando paquetes a tasas entre 1kbps y 1Mbps. La elección de este rango esta determinada por la cadencia máxima del sistema, en este caso determinada por el router 2. En el apartado anterior hemos calculado dicha cadencia, esta es aproximadamente de 171,8, por lo tanto tenemos que comprobar como reacciona el sistema hasta llegar a esta tasa y una vez superada.

**Tabla 3.13.** Eficiencia routers

| Velocidad [kbps] | Velocidad [pqts/s] | R1 | R2 | R3 |
|---|---|---|---|---|
| 1 | 1,33 | 100,00% | 100,00% | 100,00% |
| 2 | 2,66 | 100,00% | 100,00% | 100,00% |
| 3 | 3,99 | 100,00% | 100,00% | 100,00% |
| 4 | 5,32 | 100,00% | 100,00% | 100,00% |
| 5 | 6,65 | 100,00% | 100,00% | 100,00% |
| 10 | 13,30 | 100,00% | 100,00% | 100,00% |
| 20 | 26,60 | 100,00% | 100,00% | 100,00% |
| 30 | 39,89 | 100,00% | 100,00% | 100,00% |
| 40 | 53,19 | 100,00% | 100,00% | 100,00% |
| 50 | 66,49 | 100,00% | 100,00% | 100,00% |
| 60 | 79,79 | 100,00% | 100,00% | 100,00% |
| 70 | 93,09 | 100,00% | 100,00% | 100,00% |
| 80 | 106,38 | 100,00% | 100,00% | 100,00% |
| 90 | 119,68 | 100,00% | 100,00% | 100,00% |
| 100 | 132,98 | 100,00% | 100,00% | 100,00% |
| 110 | 146,28 | 100,00% | 100,00% | 100,00% |
| 120 | 159,57 | 100,00% | 91,50% | 100,00% |
| 130 | 172,87 | 100,00% | 91,30% | 100,00% |
| 140 | 186,17 | 100,00% | 90,60% | 100,00% |
| 150 | 199,47 | 100,00% | 90,60% | 100,00% |
| 200 | 265,96 | 100,00% | 90,60% | 100,00% |
| 300 | 398,94 | 96,10% | 88,40% | 100,00% |
| 400 | 531,91 | 86,70% | 82,20% | 100,00% |
| 500 | 664,89 | 86,60% | 81,50% | 100,00% |
| 1000 | 1329,79 | 76,70% | 75,00% | 100,00% |

Era de esperar que el sistema no empiece a perder paquetes hasta que no nos aproximemos a la cadencia de inyección máxima soportada por el router 2, en la prueba comprobamos como realmente el router 2 no es capaz de procesar todos los paquetes que le llegan a partir de una tasa aproximada de 160 paquetes por segundo, teóricamente esperábamos que las perdidas empezaran a partir de los 170 paquetes por segundo aquí observamos como el resultado obtenido no se distancia tanto del comportamiento teórico, esta diferencia entra totalmente dentro del rango esperado.
A continuación mostramos la gráfica referente a estos resultados.

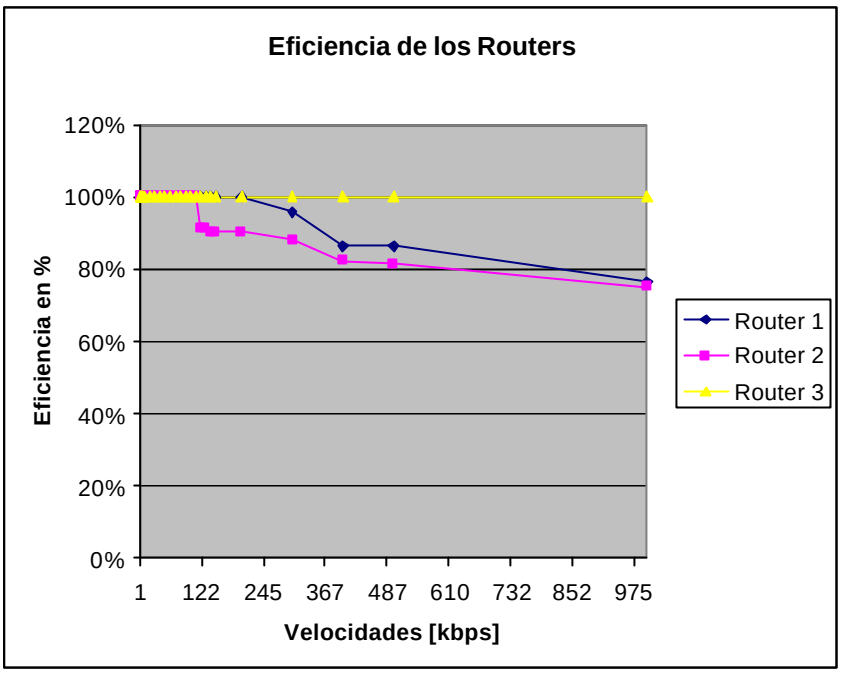


**Figura 3.14.** Eficiencia sistema

# CAPÍTULO 4. ARQUITECTURA E IMPLEMENTACIÓN DE PAGING IP

## 4.1 Introducción

El objetivo de este capítulo es explicar la arquitectura de paging que se ha implementado en el escenario de pruebas del laboratorio. Esta arquitectura está definida en [7]. El capítulo empieza con la definición de los conceptos previos para comprender la arquitectura, después se describen los principales agentes involucrados en el proceso de paging, seguido de las interfaces definidas entre ellos, y los mensajes enviados entre cada uno de ellos, más adelante se expondrá el escenario implementado en este trabajo para probar esta arquitectura así como en un último capítulo se expondrán las diferentes pruebas de funcionamiento realizadas.

## 4.2 Conceptos previos

A continuación se definen aquellos conceptos necesarios para comprender la arquitectura de paging. Las definiciones aquí expuestas están extraídas de [6].

**Estado Inactivo***:* estado en el que el terminal móvil limita su capacidad para recibir tráfico IP normal reduciendo la monitorización de los canales radio. Esta limitación permite ahorrar energía (batería) y reducir la señalización enviada en la red.

**Períodos de tiempo en modo inactivo**: Implementación del modo inactivo en el que el móvil alterna periodos de escuchas de tráfico con otros en los que no escucha ningún tipo de tráfico. Los períodos de tiempo en modo inactivo se sincronizan con la red, esta sincronización aporta retrasos de tráfico al móvil durante los períodos de escucha. Además, el móvil puede estar limitado a escuchar determinados canales de señalización que en la práctica no son canales típicos por los que se transporte tráfico IP.

**Paging:** El paging es la consecuencia del envío de un paquete a un determinado móvil que en ese momento se encuentre en estado inactivo. Utilizamos el paging para despertar al terminal y que anuncie el área de localización donde se encuentra para poder enviar el paquete que se había enviado para él.

**Área de paging:** Grupo de puntos de acceso radio que están señalados para localizar un nodo móvil en estado inactivo. Un área de paging no tiene que corresponder necesariamente con una subred IP. Un móvil en estado inactivo debe enviar una señal a la red cuando cruza el límite de un área de paging para que la red no tenga información de localización equivocada.

**Canal de paging:** canal radio dedicado a la señalización de móviles en estado inactivo para las funciones de paging. En la práctica, el protocolo utilizado en el

canal de paging es usado normalmente por el protocolo radio de la capa de enlace, aunque algunos protocolos de paging tienen previsión de llevar arbitrariamente tráfico (y éstos pueden utilizarse para llevar tráfico IP).

**Canal de tráfico:** canal radio por donde se envía normalmente el tráfico IP dirigido hacia un móvil activo. Este canal lo utilizan los móviles activos enviando y recibiendo tráfico IP y no está continuamente activo en un móvil en estado inactivo. Para algunos protocolos de enlace radio, éste puede ser el único canal utilizado.

**Registros de áreas de paging**: Corresponde a la información de localización de los terminales móviles. Los nodos móviles en estado inactivo envían información hacía la red cuando cambian de área de paging para informar de su nueva localización y de esta manera establecer la presencia del nodo en la nueva área de paging.

## 4.3 Arquitectura de paging

La arquitectura de paging implementada se basa en la arquitectura descrita en [7]. Este apartado describe los agentes funcionales que intervienen en el paging IP y las interfaces utilizadas entre ellos.

### 4.3.1 Agentes involucrados en la arquitectura

En la arquitectura de paging intervienen los siguientes elementos:

**Host (H):** El host móvil, es una máquina IP estándar, en [8], encontramos una definición exacta de máquina. Ésta puede conectarse por una red IP backbone cableada, por un enlace inalámbrico en el que se intercambian datagramas IP (modelo usado en móviles), o puede conectarse directamente a una red cableada IP, de forma irregular (modelo usado en arquitecturas nómadas) o de forma constante (modelo usado en arquitecturas cableadas). El host puede soportar algunos protocolos de movilidad IP (por ejemplo, Mobile IP). El Host debe anunciar cuando pasa a estado inactivo y sobre cualquier cambio de área de paging mientras se encuentre en este estado. Otra característica que puede tener este elemento es que puede soportar el protocolo utilizado en la red para despertarle de su inactividad cuando un paquete va dirigido hacia él. Este protocolo puede estar especializado en la capa dos, L2, del canal de paging, para el tráfico IP o puede ser un período de tiempo en estado inactivo en el que la máquina periódicamente se despierta y escucha la capa dos para ver si hay tráfico IP para ella.
Un móvil en estado inactivo también es responsable de determinar cuando ha cambiado de área de paging y debe responder a los cambios de área de paging directa o indirectamente informando al Tracking Agent sobre su localización.

**Paging Agent (PA):** Este agente es el responsable de alertar a la máquina o terminal cuando llega un paquete y la máquina está en estado inactivo. El PA alerta al terminal a través del protocolo característico de la capa dos de enlace

y de la implementación de la máquina en estado inactivo, este agente supone que la capa dos soporta IP. Además, debe mantener actualizada la información sobre las áreas de paging difundiendo la información a través de los routers hasta los puntos de acceso, usando broadcast o multicast, de manera que los Host puedan identificar en qué área de paging se encuentran. La información de área de paging puede enviarse en la capa dos o puede enviarse mediante IP. Una característica importante es que cada información de área de paging es enviada por un único agente de paging.

**Tracking Agent (TA):** Éste es el responsable de acceder a la base de datos de todos los clientes que están bajo el control de un Paging Agent. El Tracking Agent accederá a la base de datos para poder cambiar el estado de los clientes, ya sea para el paso a estado inactivo o para la activación de los terminales, así como para actualizar cualquier otro registro referente a cada uno de los clientes que contenga la base de datos. El Tracking Agent recibe mensajes de actualización de los hosts cuando éstos cambian de área de paging. Cuando un paquete llega para un host bajo el control de un Dormant Monitoring Agent, el Tracking Agent es el responsable de notificar al Dormant Monitoring Agent, sobre qué Paging Agent es el último que ha mandado la información de área de paging de la máquina.

**Dormant Monitoring Agent (DMA)**: Éste agente detecta el retraso de los paquetes que van hacia una máquina que está inactiva. Es el responsable de preguntar al Tracking Agent el último agente de paging conocido de la máquina e informar al Paging Agent que envíe el paging al terminal. Una vez que el agente de paging ha enviado que existe una conexión enrutable hacía Internet desde la máquina, el Dormant Monitoring Agent fija el retraso del paquete hacía la máquina. Además, la máquina o su Tracking Agent pueden seleccionar el Dormant Monitoring Agent cuando la máquina entra en estado inactivo y periódicamente cada vez que la máquina cambia de área de paging.

### 4.3.2 Descripción de las interfaces

La comunicación entre los elementos descritos se realiza mediante diferentes interfaces por las que se intercambian los mensajes implicados en el diálogo entre un terminal inactivo o activo y los agentes de la arquitectura o el tráfico generado entre los agentes.

A continuación describiremos las interfaces que existen y los mensajes que circulan por éstas interfaces, se explicarán los mensajes que hemos implementado en el escenario de pruebas del laboratorio.

En la arquitectura de paging expuesta existen cinco interfaces por las que circula el tráfico procedente de la máquina hacia cada uno de los agentes y de los agentes hacia la máquina o el tráfico generado entre agentes, a continuación definimos estas interfaces y los mensajes seleccionados para la implementación del escenario.

**Host-Paging Agent (H-PA):** esta interfaz corresponde al tráfico que circula entre la máquina y el agente de paging. El mensaje implementado ha sido:

1. Mensajes del agente de paging alertando al terminal cuando el Dormant Monitoring Agent le ha anunciado que ha llegado un mensaje para la máquina, éstos son los mensajes de paging.

**Host- Tracking Agent (H-TA):** interfaz por la que circulan los mensajes que se intercambian el terminal y el Tracking Agent. En esta interfaz hemos implementado:

2. Información de la máquina hacia el Tracking Agent cuando ésta ha cambiado de área de paging.

**Dormant Monitoring Agent-Tracking Agent (DMA-TA):** esta interfaz corresponde al tráfico generado por el DMA hacia el TA y del TA al DMA. Aquí destacan y a la vez se han implementado:

3. Información del Dormant Monitoring Agent hacia el Tracking Agent diciéndole que un paquete ha llegado para una máquina inactiva cuya ruta es inaccesible.
4. Mensajes del Dormant Monitoring Agent hacia el Tracking Agent informando de que debe actualizar los registros de los clientes cuando se produce cualquier cambio, paso de estado activo a inactivo, cambio de área de paging, etc.
5. Mensajes del Tracking Agent al Dormant Monitoring Agent informándole del Agente de Paging de cada máquina: este mensaje no se ha implementado ya que suponemos en nuestra arquitectura que únicamente tenemos un agente de paging y que por lo tanto todas las máquinas tendrán el mismo agente de paging.
6. Información del Tracking Agent hacia el Dormant Monitoring Agent cuando una máquina ha entrado en estado inactivo si la máquina no le ha informado previamente.

**Dormant Monitoring Agent- Paging Agent (DMA-PA):** interfaz por donde se comunican el agente de paging y el Dormant Monitoring Agent. Los mensajes desarrollados han sido:

7. Mensaje del Dormant Monitoring Agent hacia el Agente de Paging informando que debe enviar mensajes de paging a una determinada máquina.
8. Respuesta negativa del agente de paging si la máquina no ha mandado respuesta a los mensajes de paging.
9. Respuesta positiva del agente de paging si la máquina ha enviado una respuesta al paging.

**Host- Dormant Monitoring Agent (H-DMA):** esta interfaz comunica cada máquina con su Dormant Monitoring Agent. Los mensajes implementados han sido:

10. Registros de la máquina hacia el agente cuando ésta entra en estado inactivo.
11. Mensajes de la máquina informando al Dormant Monitoring Agent que ha cambiado de área de paging.

Para facilitar la comprensión de las interfaces y elementos anteriormente descritos y que se pueda visualizar un escenario esquemático exponemos un sencillo diagrama en el que se aprecian todos los elementos citados.

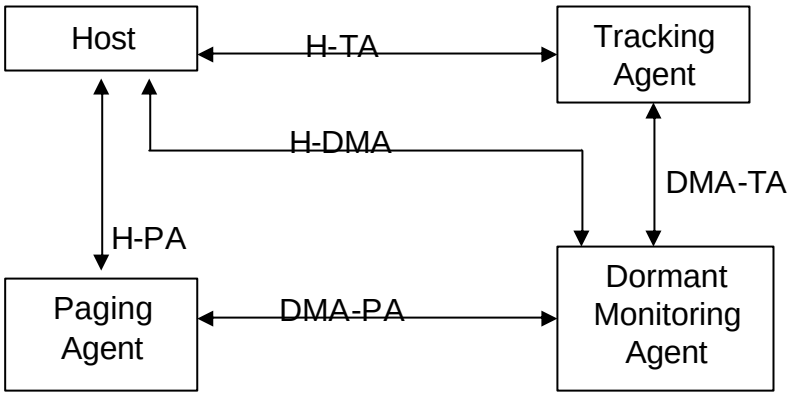


**Figura 4.1**. Diagrama funcional de la arquitectura

## 4.4 Implementación

Una vez explicados todos los elementos que intervienen en la arquitectura de paging así como los diferentes mensajes que circulan por las interfaces, explicaremos la implementación en el escenario del laboratorio que hemos desarrollado. Primero introduciremos brevemente el método de desarrollo para posteriormente empezar con la descripción del escenario, seguida de un breve repaso a los elementos así como a los mensajes para finalmente comprobar su correcto funcionamiento con una serie de pruebas.

### 4.4.1 Objetivos y requisitos

Los objetivos que debe cubrir esta implementación son:

- Programar los elementos que se explica en la arquitectura de paging tal y cómo la especifica el RFC 3154.
- Definir y programar los mensajes descritos en las diferentes interfaces explicadas en 4.3.2.
  - Que el contendido de los mensaje sea el que establezca el RFC, por cada interfaz circulan mensajes con información diferente.
  - Que cumplan con los protocolos definidos en la arquitectura de paging.
- Asegurar la correcta interacción con los protocolos de routing, en especial Mcast. .

A continuación listamos los principales requisitos que deben respetar los diferentes programas ha desarrollar:

- El código fuente debe ser **flexible**, se ha continuado la filosofía heredada de [1], un diseño que permite con un esfuerzo mínimo el cambio de librerías sin problemas.
- Debe ser un código **portable**, es decir, que se pueda utilizar en sistemas diferentes a Linux. Por ejemplo, FreeBSD es una buena plataforma en cuanto al desarrollo y testeo de aplicaciones IPv6.
- El programa debe estar **optimizado** al máximo. En la arquitectura de paging implementada intervienen un gran número de elementos que pueden hacer que haya pérdidas de paquetes y conflictos entre ellos, los diferentes programas deben ser capaz de aceptar de forma independiente diferentes paquetes a la vez y cada uno de los elementos trabajar de forma independiente de forma que no interfieran los unos con los otros y puedan trabajar paralelamente.
- De forma parecida a Mcast todo el software desarrollado debe utilizar librerías a bajo nivel, ya que tienen que leer, modificar e inyectar paquetes IPv6 e ICMPv6 con unas características que hemos determinado previamente.

### 4.4.2 Descripción del entorno de trabajo

La plataforma de desarrollo elegida ha sido GNU / Linux, con la distribución Gentoo Linux, usando el kernel 2.6, esta elección está basada en que la implementación del software Mcast para probar Geopaging se realizó bajo esta misma plataforma y debido a que los resultados obtenidos han sido satisfactorios hemos decidido continuar y no migrar a FreeBSD posibilidad que fue comentada en [1].

Como lenguaje de programación ha sido utilizado el lenguaje C ya que es habitualmente instalado en una distribución GNU/Linux de serie, junto a su herramienta de compilación llamada gcc. Por cada agente se ha creado un Makefile para facilitar la compilación del código, ya que el tiempo de compilación es mucho menor que utilizando la herramienta de compilación que trae el Anjuta. Anjuta es el entorno gráfico de programación por el que optamos a la hora de trabajar más fácil de visualizar errores a la hora de escribir el código que creando los archivos desde un terminal de consola. En el anexo I se encuentra más información acerca de este entorno gráfico de trabajo.

También como en [1] los procesos desarrollados se ejecutarán con permisos de root, esto se debe a que en Linux un programa reside en un espacio de memoria llamado espacio de usuario, si este programa quiere acceder a datos propios del kernel, realiza lo que llamamos una llamada de sistema. Dependiendo del tipo de datos, se debe ser superusuario (root) y también es necesario ser root para poder realizar cierto tipo de operaciones, por ejemplo las relacionadas con redes. Un usuario normal sólo tiene derecho a establecer sockets de alto nivel (TCP o UDP). No puede inyectar ni recoger paquetes a

más bajo nivel. De esta manera decidimos trabajar en modo root ya que los agentes que intervienen en la arquitectura deben inyectar y leer paquetes a más bajo nivel.
Para la programación de los agentes que deben capturar y escribir paquetes IPv6, se probó la interfaz estándar de Linux, que cumple con los RFC 3493 y 3542, pero como se comentaba anteriormente en [1] en la utilización de esta interfaz se encontraron dos problemas: uno era que algunas funciones no se comportaban según lo esperado, y otra era que en IPv6 el comportamiento de los RAW sockets cambia, hemos comprobado con la realización de diferentes pruebas programando con RAW sockets, que éstos aún no son capaces de capturar paquetes IPv6, aunque las especificaciones dicen que sí.
La manipulación de la cabecera IPv6 necesaria para introducir Extensions Headers se ha realizado mediante dos librerías: *libpcap* para escuchar paquetes y *libnet* para inyectarlos.
Finalmente, para poder enrutar los paquetes que generaremos en todas las máquinas que configuran la topología del escenario es necesario ejecutar *zebra y ripngd.*

### 4.4.3 Esquema del escenario

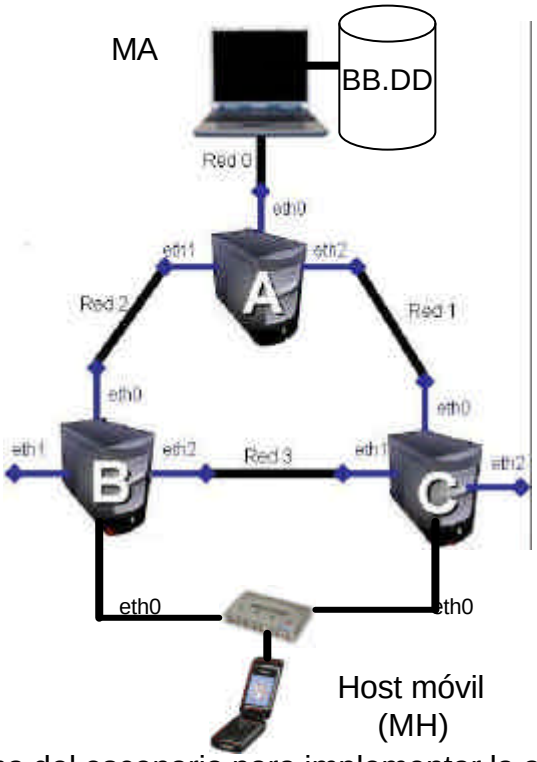


**Figura 4.2.** Esquema del escenario para implementar la arquitectura de paging

La maqueta montada sigue el modelo de topología implementado en capítulos anteriores cuando realizamos las pruebas de Mcast, la diferencia está en que hemos conectado dos de los routers a un hub que a su vez permitirá tener conectado un nuevo elemento, un host móvil, e interactuar con los agentes situados en el portátil conectado al router A. Se ha instalado un hub para poder simular que el host movil cambia de celda y así le puedan llegar las peticiones de paging pasen por un router o por otro. En el siguiente punto explicamos el funcionamiento de esta implementación.

### 4.4.4 Definición de los elementos

El equipo MA contiene toda la arquitectura del paging IP. Este elemento actúa, como Tracking Agent, Dormant Monitoring Agent y Paging Agent. Para ello, tendrá instalado y ejecutándose la implementación realizada de los tres agentes. Además, esta máquina incluirá una pequeña base de datos necesaria para cualquier arquitectura de paging. Se trata de un fichero de texto que en nuestro caso contiene los siguientes campos:

- un campo con la dirección de la ultima celda actualizada en un LU,
- distancia de paging,
- el estado de los clientes,
- la home address y la política de paging.

La finalidad de este fichero es poder probar toda la arquitectura implementada con una base de datos que contenga la máxima información real posible.

Las máquinas A, B y C son los routers que formaban el escenario del capítulo 3, la función principal de estas máquinas es encaminar los paquetes hacia y desde el receptor móvil. Formarán nuestra red celular IP.

El MH representa el receptor móvil, este nuevo elemento será la máquina a quién van dirigidos los paquetes y ésta cambia de estado, si está activo recibirá su paquete sin problemas y por lo tanto no entrarán en juego el resto de elementos de la arquitectura de paging, pero si está en estado inactivo, entrarán en juego los elementos que han de despertarle enviándole mensajes de paging. Este nuevo elemento como su propio nombre indica puede tener movilidad y por lo tanto cambiar de área de paging, cuando esto sucede debe avisar a la red que ha cambiado de área de paging.

El hub nos permite emular toda la red de acceso radio formada por un gran número de celdas conectadas a los nodos B o C. Los movimientos del MH por esta red de acceso y sus cambios de estado serán también emulados por un código residente en el propio MH. En caso de que el MH pasa de una celda controlada por el nodo B a otra controlada por él o viceversa y la situación lo requiera (por ejemplo, el MH esté activo) el MH cambiará su configuración IP para adaptarse al cambio de red.

### 4.4.5 Código desarrollado

Cada uno de los agentes involucrados en el escenario se ha implementado de forma independiente al resto y con una serie de funciones también independientes entre ellas para continuar la filosofía de realizar un código modular que facilite su posterior mejora.

A continuación listamos los archivos de código que forman cada uno de los agentes.

- Archivos que forman el TA:
  - *tclientdma2.c*
  - *tmain.c*
  - *track.c*
  - *tserverdma1.c*
  - *tserverdma2.c*
  - *tutils.c*
- Archivos que forman el DMA 1:
  - *dormant-client.c*
  - *dormant-main.c*
  - *dormant-utils.c*
- Archivos que forman el DMA 2:
  - *dclient_paging.c*
  - *dmain.c*
  - *dpcap.c*
  - *dserver.c*
  - *dutils.c*
  - *dclient.c*
- Archivos que forman el PA:
  - *main.c*
- Archivos que forman el MH:
  - *IFS.c*
  - *NET.c*
  - *PCAP.c*
  - *RAW.c*
  - *UTILS.c*
  - *Main.c*

Cada uno de estos archivos está formado por un conjunto de funciones independientes pero que siguen el mismo criterio de implementación, algunas de las funciones se han reutilizados para varios agentes.
De forma general para que entiendan el tipo de funciones que constituyen el código explicaremos el significado global de cada una de ellas. Las funciones que constituyen los archivos *xutils.c*, son funciones reutilizadas por todos los agentes que realizan funciones de carácter general, por ejemplo son las funciones para transformar números de string a hexadecimal y a la inversa o las funciones para inicializar espacio en memoria de los vectores dinámicos que se utilizan en el código.
Los archivos *xclient.c* son archivos constituidos por dos funciones, en una se crea el socket TCP para la implementación de la parte del cliente y en la otra se envía la información al servidor que estará escuchando en su puerto correspondiente, más adelante especificaremos cada uno de los puertos utilizados.
De la misma manera que los clientes, los archivos *tserverdma1.c, tserverdma2.c, dserver.c* son los archivos que implementan los diferentes servidores TCP que se crean, estos también están formados por dos funciones principales, en una se crea el socket y en la otra se escucha del cliente.

Los archivos *xpcap.c* contienen todas las funciones necesarias para la captura de los paquetes. El resto de funciones se explicaran con más detalle en el annexo II de este mismo documento. Por otra parte en el CD que adjuntamos con este documento encontraran todos los archivos que forman el código, estos se han dividido en carpetas según el elemento implementado.

Los diferentes agentes se ejecutan en el MA, esta máquina como se ha comentado más arriba debe tener la base de datos de clientes y deben estar ejecutándose el programa de enrutado zebra y ringd, más adelante insistiremos en este aspecto.

La comunicación entre todos los elementos definidos en la arquitectura de paging requería que la implementación fuese concurrente y paralela entre los elementos. Para poder desarrollar esta concurrencia fue necesaria la implementación del código mediante threads, cabe decir también que la comunicación entre los diferentes agentes se ha realizado a través de sockets TCP/IPv6. A continuación detallaremos la función de todos y cada uno de los threads y sockets utilizados.

El TA lo constituyen dos threads. En un thread se crean dos sockets, los dos esperando una comunicación provinente del DMA. El primero de ellos servirá para recibir una petición del DMA para consultar en la BBDD una determinada IP, esta comunicación se efectuará por el puerto 8005.
El segundo servirá para actualizar en la BBDD una petición del DMA, este socket escucha en el puerto 9000. Estos dos sockets se encuentran en un mismo thread ya que el segundo es totalmente dependiente del primero, puesto que si se ha realizado una petición de paging y si el MH contesta adecuadamente, habrá una actualización en la BBDD.
En el otro thread se espera a que el DMA le pida que busque a un cliente en la base de datos y le envíe toda la información que tenga sobre él, este DMA es independiente al citado anteriormente, ya que éste es al que le llega un paquete para un MH y tiene que saber si se encuentra en su base de datos.

El DMA, está dividido en dos programas independientes entre si. El primero, DMA 1, será el que ejecute la petición de búsqueda de cliente hacia el TA, así, este programa creará un socket en el puerto 8000 para enviarle al TA la home address del MH solicitado.

El segundo programa, DMA 2, estará formado por dos threads, el primero como ya se ha dicho anteriormente comunicará el DMA con el TA por el puerto 8005. Por este socket el DMA 2, recibirá la información de estado del MH solicitado y si se encuentra en estado inactivo, este le enviará al PA toda la información del MH al que hay que hacer un paging por el puerto 8010.
El segundo thread se encargará de capturar los mensajes recibidos del MH como contestación a un paging, un cambio de estado o un cambio de área de localización. Esto se realizará mediante un filtro pcap que capturará todos los paquetes que sean IPv6. Este mismo filtro detectará que tipo de mensaje es y si es alguno de los enviados por un MH, pasará la información al TA por el puerto 9000 para que este la actualice oportunamente.

Finalmente, la implementación del PA no ha necesitado ningún thread, uya que éste únicamente escucha por el puerto 8010 para recibir una petición para hacer paging, si se recibe alguna petición, éste ejecuta el mecanismo para formar una petición cuyo formato se explicará más adelante. Además, cabe comentar que en el Anexo II, se explicará profundamente el codigo programado.

### 4.4.6 Funcionamiento de la implementación

La arquitectura de paging realizada responde a todas las funcionalidades explicadas anteriormente como hemos visto éstas dependen del estado del terminal móvil. A continuación describiremos el funcionamiento de la arquitectura en el escenario implementado mostrando un diagrama donde se observan los mensajes enviados entre los diferentes agentes.

Antes de entrar a describir este funcionamiento debemos comentar algunos aspectos a tener en cuenta. Para que todo funcione correctamente debemos iniciar previamente en todas las máquinas el zebra y el ripngd para que encaminen los paquetes que se enviarán, una vez inicializados se ejecutan todos los elementos que componen el MA, el Tracking Agent, el Dormant Monitoring Agent y el Paging Agent, paralelamente debemos ejecutar el programa que simulará el MH en otra máquina. Una vez ejecutándose todos los programas podemos ver cómo funciona la implementación.

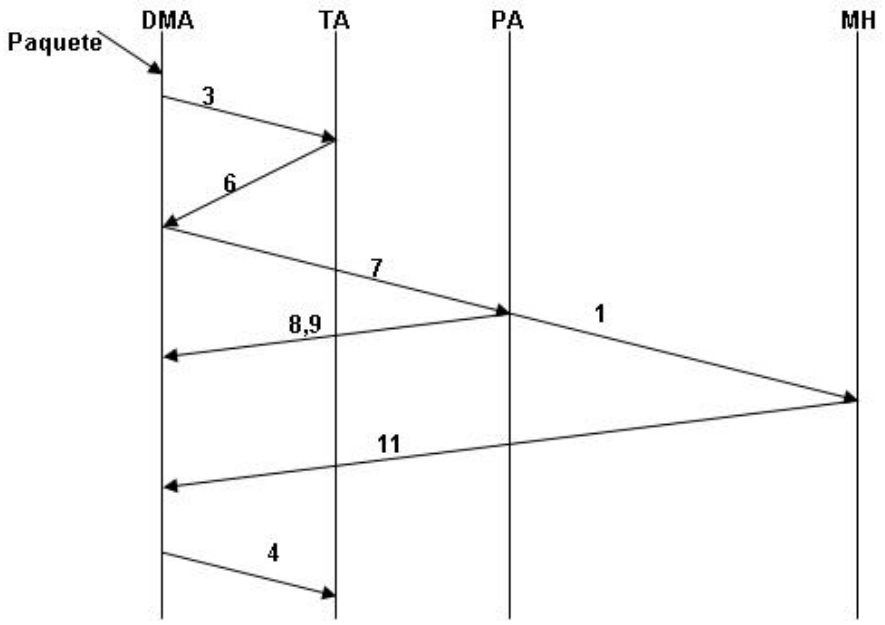


**Figura 4.3.** Proceso de paging (falta modificarlo para meter el 9 y10)

La figura 4.3 muestra el proceso seguido en la arquitectura de paging, los números corresponden a los tipos de mensajes enviados entre los agentes explicados en el apartado 4.3.2 Descripción de las interfaces. Cuando el DMA recibe un paquete para un MH éste envía un mensaje tipo 3, donde le envía al TA la dirección del MH que debe buscar en la base de datos. La contestación a

este mensaje corresponde al número 6, aquí el TA le dice si el cliente se encuentra o no en la base de datos, y además si el cliente se encuentra en la base de datos y su estado es inactivo, le envía toda la información referente a ese cliente. En el mensaje número 7 el DMA le envía toda la información referente al cliente que hay que despertar al PA. Si PA no puede enviar el paging al cliente envía el mensaje número 8, si en cambio, si que ha podido realizar el paging correctamente envía el mensaje número 9. El mensaje número 1, es el mensaje de paging enviado hacia el MH para despertarle. El mensaje 11 es la contestación del MH hacía el DMA al Paging Agent, este mensaje podrá ser un paquete ICMPv6 o un paquete IPv6 con Extensions Headers. Finalmente en el gráfico observamos el mensaje número 4, en este mensaje el DMA le dice al TA que actualice la información del cliente en la base de datos, para ellos el DMA se la envía.
El proceso representado en la figura anterior se simplifica de manera radical cuando el MH se encuentra en estado activo, es decir, cuando el terminal está activo el DMA recibe un paquete para él, éste envía hacia el TA el mensaje número 3 y en el mensaje número 6 el TA le comunica que el terminal está activo y por lo tanto puede enviarle el paquete al MH.

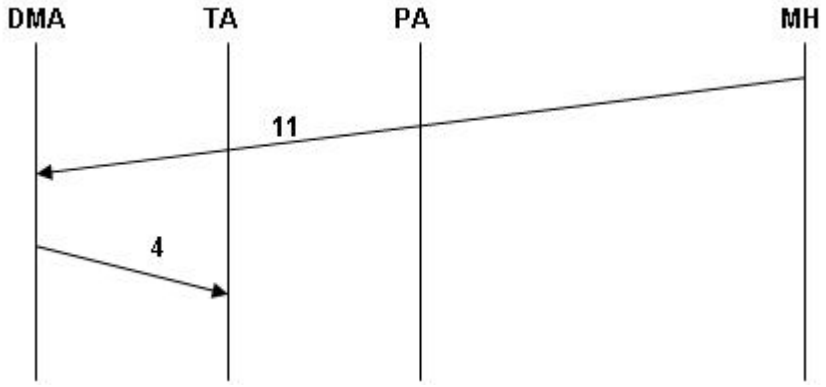


**Figura 4.4.** Proceso de cambio de estado o celda

En la figura 4.4 se observa el proceso de cambio de estado o celda de un MH. Cuando el MH cambia de estado o celda, en nuestra implementación cada 30 y 55 segundos respectivamente, envía el mensaje número 11, notificándole su cambio. Cuando el DMA recibe este paquete envía el mensaje número 4 al TA para que actualice la nueva información del MH en la base de datos.

### 4.4.7 Descripción de los mensajes

En este punto detallaremos el formato de los mensajes comentados en el apartado anterior.
Estos mensajes han sido previamente extraídos del documento [7]. Cabe decir que se han hecho dos tipos nuevos y otro más se ha modificado para tener compatibilidad con el receptor móvil.

El primer mensaje creado ha sido el de notificación por parte del host móvil que ha recibido un mensaje de paging y que pasará a estado activo. Para esto, y tal y como dice el draft referenciado en [7] se ha usado un mensaje ICMPv6 con un tipo de mensaje que no está en uso actualmente y que identificaremos como

una respuesta a un mensaje de paging, en este caso el tipo de mensaje será identificado con el número 43. Finalmente y como datos del mensaje, este contendrá primero su home address y su identificador de celda. En la figura X.1 podemos ver la estructura general del mensaje.

| Type (1 byte) | Code (1 byte) | Checksum (2bytes) |
|---|---|---|
| Identifier (2 bytes) | | Sequence Number (2 bytes) |
| Data … | | |

**Figura 4.5** Estructura de un mensaje ICMPv6

Se pensó primeramente implementar este mensaje con RAW Sockets, pero viendo la incompatibilidad que en según que maquinas funcionaba y en según cuales maquinas no, se decidió implementar el mensaje ICMPv6 con la aplicación sendip.
El segundo mensaje creado cumple la función de indicar por parte del host móvil al Dormant Monitoring Agent cualquier cambio de estado que no sea producido por una petición de paging y también cuando se produzca un cambio de celda en la que se encuentre (Location Update). Ésta esta formado por una Destination Option Header que contiene como datos, el estado, la home adress y el identificador de celda en el que se encuentra. Podemos ver su estructura en la figura 4.6.

| Versión | Traffic Class | Flow Label | |
|---|---|---|---|
| Payload Lenght | | Next Header | Hop Limit |
| Source Address (128 bits) | | | |
| Destination Address (128 bits) | | | |
| Next Header | Header Extension Length | Option Date Type | Option Data Length |
| Option Data | | | |

**Figura 4.6.** Estructura de un mensaje IPv6 con una Destination Option Header

A este mensaje también se le llama desde una aplicación externa llamada *dormantpacket* que a su vez llama a la aplicación *sendip* utilizada para la inyección de diferentes tipos de paquetes en una red. Cabe decir que a la aplicación externa dormanpacket se le pasará por parámetro el estado, la home address y la celda en la que se encuentra el host móvil, en este orden.

```
./sendip -p ipv6 -6s fede::c:5 -d \
0xbb085b4e6f64655f5374617465 5d3d 30303030 5b486f6d655f416464726573735d3d 66
6564653a3a33303030 3a373030305b43656c6c5f4964656e5d3d 00000000000000000000
00000 31 -6n 0 fede::a:e
```

Finalmente, el mensaje modificado corresponde a la petición de paging que envía el Paging Agent al host móvil, la modificación ha consistido en introducir como datos de la Hop by Hop Extensión Header la home address del host móvil a parte de la distancia de paging antes ya introducida. En la figura 4.7 se muestra como seria tal mensaje.

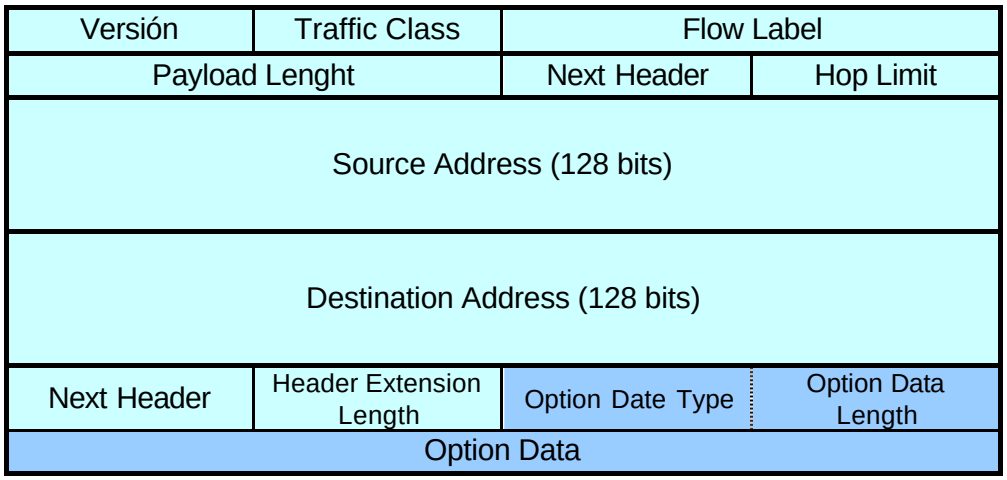


**Figura 4.7**. Estructura de un mensaje IPv6 con Hop by Hop Extensión Header

# CAPÍTULO 5. TEST DE LA IMPLEMENTACIÓN DE PAGING IP

## 5.1 Introducción

En este capítulo se describirán las pruebas realizadas para comprobar el correcto funcionamiento de la implementación de paging IP explicada en el capítulo anterior. Para comprobar que los diferentes programas hacían realmente lo que les corresponde hemos diseñado un conjunto de sencillas pruebas donde mostraremos capturas de las tramas enviadas entre los agentes para que se pueda observar como la comunicación se desarrolla tal y como se explicó en el capítulo 4.

## 5.2 Descripción del escenario

Para llevar a cabo estas pruebas se ha diseñado un escenario mínimo que permite emular toda una red, con todos los agentes que forman la arquitectura de paging ubicados en un solo nodo el MA (Mobility Agent) y un solo nodo móvil (MH) para probar que realmente responde a las especificaciones del RFC.

El escenario montado para realizar las pruebas de funcionamiento corresponde a la topología mostrada en el capítulo 4. Recordemos que este escenario está formado por 3 máquinas que simulan los routers de la red por donde se envían los paquetes, una cuarta máquina que representa el receptor móvil al que van dirigidos los paquetes, y finalmente otro ordenador portátil que representa la conexión de los tres agentes implicados en el proceso de paging. También no podemos dejar de mencionar que el receptor móvil se une a la red gracias a un hub que interconecta las máquinas del escenario. Las especificaciones de todas las máquinas están explicadas en el capítulo 2, donde se realizaron las pruebas de rendimiento de Mcast.

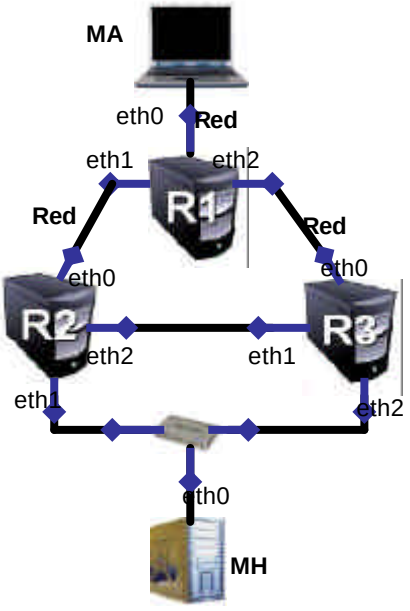


**Figura 5.1.** Esquema del escenario de pruebas

La configuración de los equipos es la misma que la utilizada en el pruebas del capítulo 2, el MH con el que hemos realizado las pruebas tiene configurada la IP ff00::a000:1234. Para probar el funcionamiento podíamos haber escogido cualquier otra IP que tuviera el MH en su base de datos. Para comprobar la funcionalidad de la implementación hemos ejecutado los agentes en el MA y enviado paquetes al MH configurado en la red.
Una característica del funcionamiento de la red es que la comunicación entre el MA y el MH se realizará pasando por diferentes routers dependiendo del identificador de celda asociado al MH al que se envían los paquetes, es decir, si el MH al que enviamos los paquetes tiene asociado el identificador de celda ff05::1:1a34 el paging pasará por los routers A y C, pero si el identificador de celda asociado al MH es el ff05::1:1234 el paging enviado por el PA pasará por todos los routers, para comprender el porqué de este encaminamiento de los paquetes pueden consultar las tablas de encaminamiento mostradas en el capítulo 2 y las distancias de paging configuradas en los routers para la simulación. .

## 5.3 Objetivos de las pruebas

El objetivo de estas pruebas es comprobar el correcto funcionamiento de la implementación realizada y comprobar que la comunicación entre los agentes se realiza como se especifica en [7].

## 5.4 Prueba 1: Envío de un paquete a un MH inactivo

### 5.4.1 Descripción

En esta prueba comprobaremos que la implementación realizada responde correctamente cuando se le quiere enviar un mensaje a un MH que está en el dominio del PA del área y este MH se encuentra inactivo. El software, primero busca en la base de datos al cliente al que se le tiene que enviar un paquete, comprueba el estado y si su estado es inactivo se le envía el paging para despertarle. Para ello, ejecutamos el DMA 1, pasándole por parámetro la IP del cliente al que supuestamente le ha llegado un paquete, el DMA 1 se tiene que comunicar con el TA, quien lo buscará en su base de datos y comprobará su estado, una vez comprobados estos campos, le responderá con la información referente a ese cliente al DMA2 , quien se comunicará para que envíe el paging al MH que se ha de despertar.

### 5.4.2 Esquema de la prueba 1

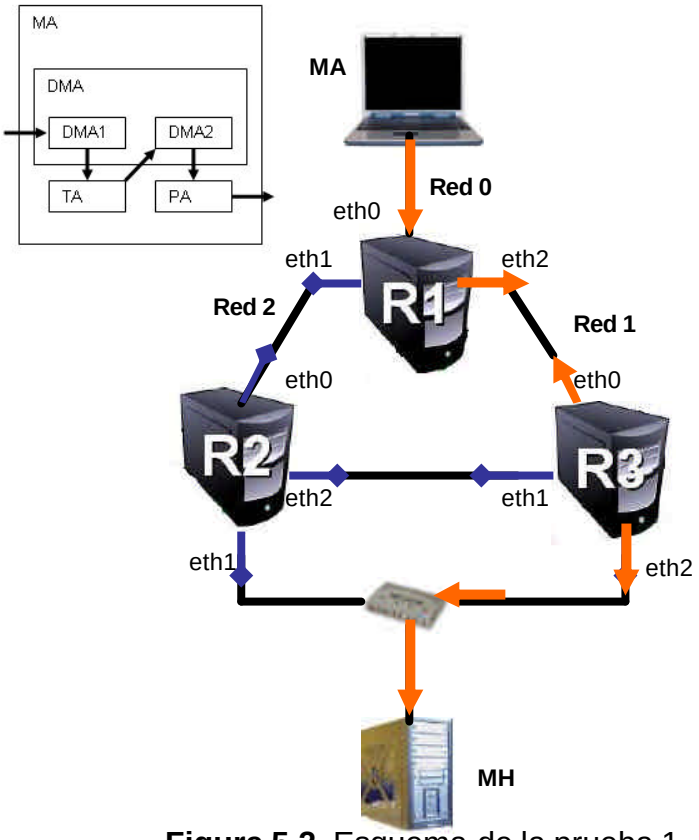


**Figura 5.2.** Esquema de la prueba 1

### 5.4.3 Comportamiento teórico

En esta primera prueba cuando ejecutamos el DMA 1 en el MA, el identificador de celda del MH es ff05::1:1a34, con este identificador el paquete de paging se pasará por R1 y R3. De esta manera el comportamiento esperado es que ejecutemos el DMA1 pasándole por parámetro la IP del cliente a enviar un paquete y este se comunique con el TA quien una vez comprobado el estado inactivo del MH en la base de datos le enviará toda la información al DMA2, el DMA 2 le pasará al PA los datos del cliente a quién debe enviarle el paging y si este puede enviar el paging correctamente al MH contestará con un mensaje de envío de paging afirmativo. El estado del terminal se actualiza en la base de datos, por lo que cuando miremos la base de datos, el estado del MH tiene que haber cambiado.

### 5.4.4 Resultado

Cuando le pasamos por parámetro al DMA 1 la dirección del cliente al que tenemos que despertar, observamos como se crea la primera conexión con el TA, el DMA1 le envía la dirección IP que será la home address del MH que el TA debe buscar en su base de datos. En la siguiente figura se observa esta primera conexión creada por el puerto 8000, recordemos que para esta comunicación se crea un socket TCP.

```
Frame 17 (1112 bytes on wire, 1112 bytes captured)
Linux cooked capture
Internet Protocol Version 6
Version: 6
Traffic class: 0x00
Flowlabel: 0x00000
Payload length: 1056
Next header: TCP (0x06)
Hop limit: 64
Source address: ::1 (::1)
Destination address: ::1 (::1)
Transmission Control Protocol, Src Port: 32889 (32889), Dst Port: 8000
(8000), Seq: 1, Ack: 1, Len: 1024
Source port: 32889 (32889)
Destination port: 8000 (8000)
Sequence number: 1    (relative sequence number)
Next sequence number: 1025    (relative sequence number)
Acknowledgement number: 1    (relative ack number)
Header length: 32 bytes
Flags: 0x0018 (PSH, ACK)
Window size: 32768 (scaled)
Checksum: 0x0cd9
Options: (12 bytes)
Data (1024 bytes)
```

**Figura 5.3.** Petición por parte del DMA de búsqueda de una home address en la BBDD del TA

Una vez el TA ha buscado al cliente en la base de datos, este contesta al DMA 2, para ello crea una conexión TCP por el puerto 8005, donde le envía toda la información referente al MH buscado. En la figura 5.4 se observa esta conexión, hay que fijarse que esta conexión se crea únicamente si el cliente buscado se encuentra en la base de datos y el estado del terminal es inactivo.

```
Frame 22 (1112 bytes on wire, 1112 bytes captured)
Linux cooked capture
Internet Protocol Version 6
    Version: 6
    Traffic class: 0x00
    Flowlabel: 0x00000
    Payload length: 1056
    Next header: TCP (0x06)
    Hop limit: 64
    Source address: ::1 (::1)
    Destination address: ::1 (::1)
Transmission Control Protocol, Src Port: 32890 (32890), Dst Port: 8005
(8005), Seq: 1, Ack: 1, Len: 1024
    Source port: 32890 (32890)
    Destination port: 8005 (8005)
    Sequence number: 1    (relative sequence number)
    Next sequence number: 1025    (relative sequence number)
    Acknowledgement number: 1    (relative ack number)
    Header length: 32 bytes
    Flags: 0x0018 (PSH, ACK)
    Window size: 32768 (scaled)
    Checksum: 0x9fe9
    Options: (12 bytes)
Data (1024 bytes)
```

**Figura 5.4.** Contestación del TA a la petición del DMA

Cuando el DMA recibe la respuesta del TA, éste crea la conexión con el PA, a quien debe enviarle los datos del cliente al que se le debe enviar el paging.

```
Frame 27 (1112 bytes on wire, 1112 bytes captured)
Linux cooked capture
Internet Protocol Version 6
    Version: 6
    Traffic class: 0x00
    Flowlabel: 0x00000
    Payload length: 1056
    Next header: TCP (0x06)
    Hop limit: 64
    Source address: ::1 (::1)
    Destination address: ::1 (::1)
Transmission Control Protocol, Src Port: 32891 (32891), Dst Port: 8010
(8010), Seq: 1, Ack: 1, Len: 1024
    Source port: 32891 (32891)
    Destination port: 8010 (8010)
    Sequence number: 1    (relative sequence number)
    Next sequence number: 1025    (relative sequence number)
    Acknowledgement number: 1    (relative ack number)
    Header length: 32 bytes
    Flags: 0x0018 (PSH, ACK)
    Window size: 32768 (scaled)
    Checksum: 0xef51
    Options: (12 bytes)
Data (1024 bytes)
```

**Figura 5.5.** Petición de paging enviada del DMA al PA

Una vez el PA recibe todos los datos del cliente, envía el paging al MH que le ha pasado el DMA. En la figura 5.6 se observa el paquete de paging enviado por PA.


**Figura 5.6** Mensaje de Paging enviado por el PA

Una vez el PA ha enviado correctamente el paquete de paging al MH, envía una respuesta afirmativa  al DMA y a su vez el DMA se comunica con el TA para que actualice el estado del MH en su base de datos. En la figura 5.7. se muestra la respuesta del paging enviada por el MH hacia el DMA y en la figura

5.8. se observa la conexión que establece el DMA con el TA para que actualice el estado ante la llegada de una respuesta de paging.

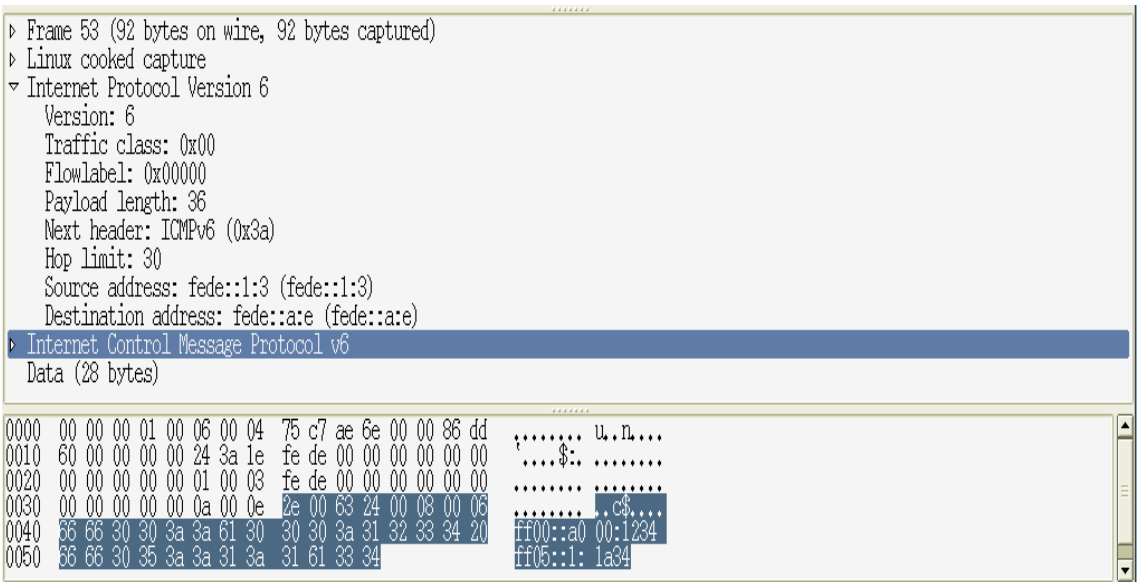


**Figura 5.7. Mensaje de Paging Response enviado por el MH hacia el DMA**

```
Frame 57 (1112 bytes on wire, 1112 bytes captured)
Linux cooked capture
Internet Protocol Version 6
    Version: 6
    Traffic class: 0x00
    Flowlabel: 0x00000
    Payload length: 1056
    Next header: TCP (0x06)
    Hop limit: 64
    Source address: ::1 (::1)
    Destination address: ::1 (::1)
Transmission Control Protocol, Src Port: 32892 (32892), Dst Port: 9000
(9000), Seq: 1, Ack: 1, Len: 1024
    Source port: 32892 (32892)
    Destination port: 9000 (9000)
    Sequence number: 1    (relative sequence number)
    Next sequence number: 1025    (relative sequence number)
    Acknowledgement number: 1    (relative ack number)
    Header length: 32 bytes
    Flags: 0x0018 (PSH, ACK)
    Window size: 32768 (scaled)
    Checksum: 0x3d85
    Options: (12 bytes)
Data (1024 bytes)
```

**Figura 5.8.** Petición de actualización en BBDD como consecuencia de la llegada de un paging response, un cambio de estado o un cambio de área de localización.

Las imágenes mostradas demuestran el correcto funcionamiento de la comunicación entre los agentes ante la llegada de un paquete para un MH que se encuentre inactivo. Finalmente para comprobar que la base de datos se ha actualizado correctamente abrimos el archivo y miramos que el estado se haya actualizado. En este documento no mostramos las líneas de la base de datos, ya que estas continuamente están actualizándose debido al mecanismo creado de cambio de estado y de LU automático.

## 5.5. Prueba 2: Envío de un paquete a un MH activo

### 5.5.1 Descripción

Esta segunda prueba es muy similar a la anterior y es para corroborar que realmente cuando un móvil se encuentra activo no se envía el paging. El TA al comprobar su estado en la base de datos se lo comunica al DMA y el paquete se podría enviar sin problemas, sin necesidad de recurrir al PA para que despierte al MH.

### 5.5.2 Esquema

El esquema de esta prueba es el mismo que representa la figura 5.2.

### 5.5.3 Comportamiento teórico

El comportamiento teórico esperado en esta prueba es que el TA busque en la base de datos al MH que le pasa el DMA, y éste a comprobar que se encuentra activo se lo comunica a este mismo DMA y no se debe enviar ningún paquete de paging.

### 5.5.4 Resultados

Al ejecutar el DMA1, como en la prueba anterior se crea la misma conexión que muestra la figura 5.3, donde se comunica con el TA para que busque al MH al que se quiere enviar un paquete. El TA busca en su base de datos al cliente y si lo encuentra comprueba su estado, al comprobar que el que el MH está activo no abre ninguna conexión con el DMA 2. Para comprobar que el TA ha buscado bien en la base de datos, la abrimos y corroboramos el estado del MH.

## 5.6 Prueba 3: Actualización de estado y de LU de un MH

### 5.6.1 Descripción

En esta última prueba mostraremos como el código implementado para que actualice el estado del MH cada 30 segundos y el LU cada 55 segundos funciona como se ha especificado en el capítulo 4, y además una vez enviados estos mensajes de actualización el DMA y el TA se comunican para que este último actualice la información del cliente en la base de datos.

### 5.6.2 Esquema

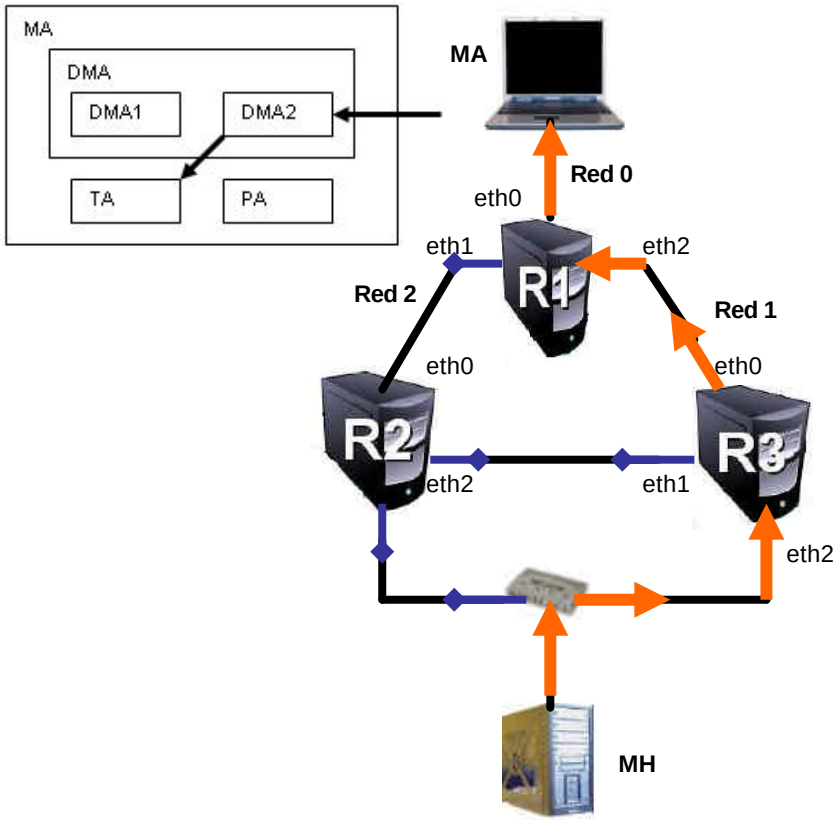


Figura 5.9. Envío de mensajes de actualización de estado y LU

En la figura 5.9, se puede observar el recorrido que seguirán los mensajes de actualización enviados por el MH. Para entender este recorrido pueden ver en el capítulo 2, las tablas de encaminamiento de los routers.

### 5.6.3 Comportamiento teórico

El comportamiento teórico esperado es que los paquetes de actualización generador por el MH se envíen cada 30 y 55 segundos respectivamente por la eth0 del MH y lleguen al MA, una vez en el MA, el DMA recibirá estos paquetes

y al identificar que son paquetes IPv6 con Destination Option Header se establecerá una conexión con el TA para que actualice estos parámetros en la base de datos.

### 5.6.4 Resultados

Los resultados obtenidos son como los esperados. El DMA 2 que se encuentra constantemente escuchando a la espera de algún tipo de paquete, al capturar paquetes IPv6 analiza su cabecera y encuentra un Destination Option Header del que lee los campos de estado, identificador de cela y home address, estos campos son los que el TA tiene que actualizar en la base de datos.
A continuación mostramos los paquetes enviados por el MH cuando actualiza su estado y cuando cambia de LU.

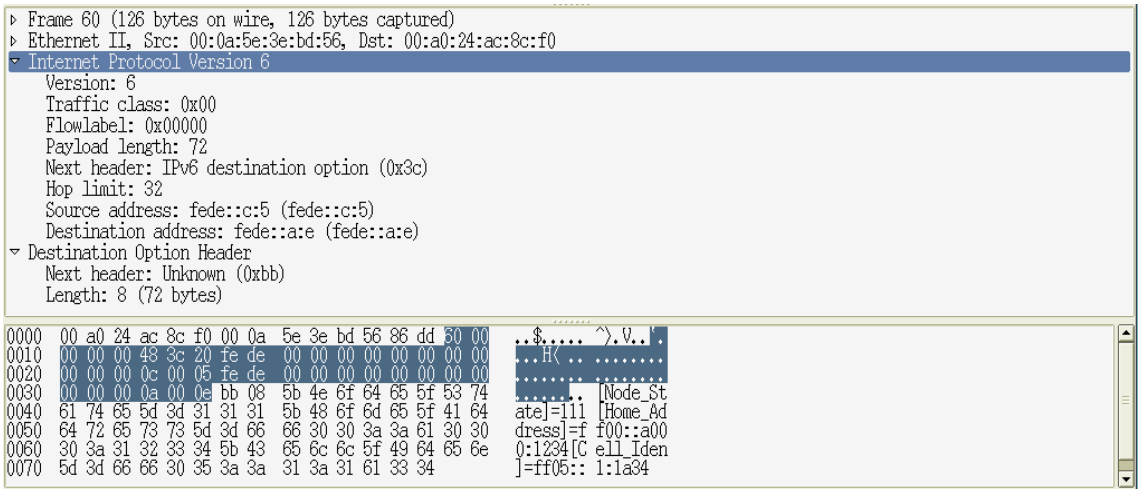


**Figura 5.10.** Cambio a estado activo (111)

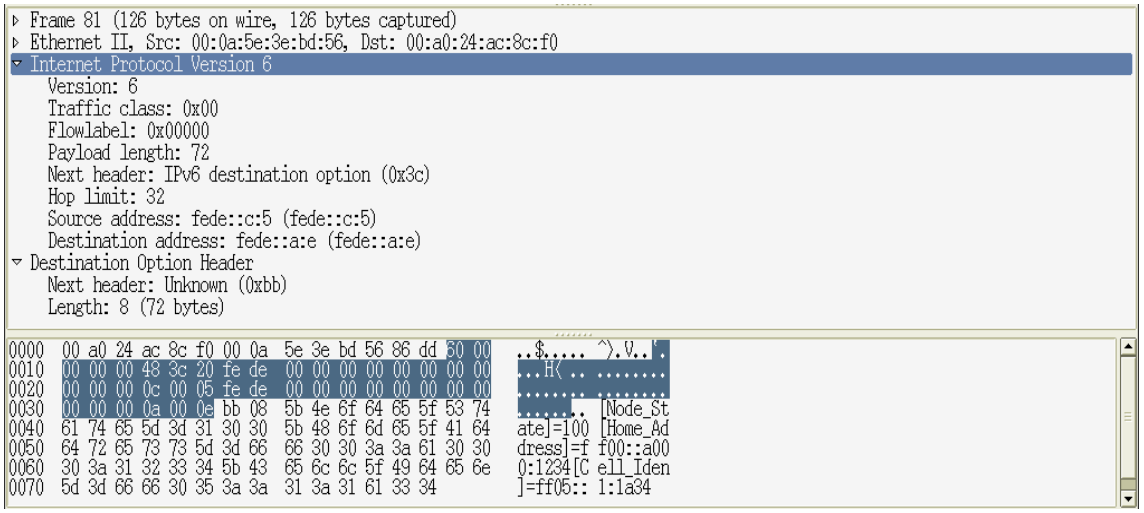


**Figura 5.11.** Cambio a estado inactivo (100)

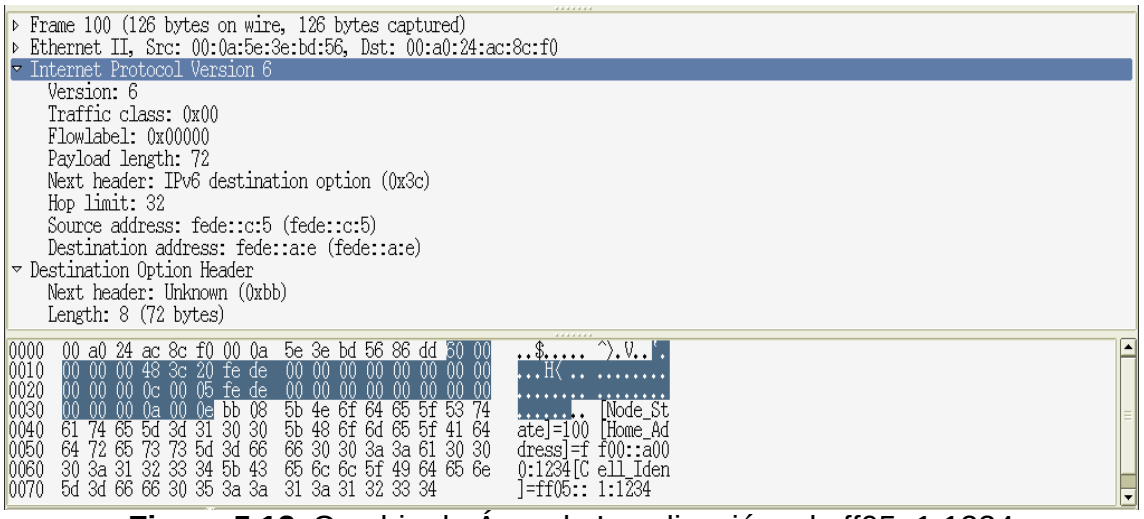


**Figura 5.12.** Cambio de Área de Localización a la ff05::1:1234

En todas las figuras anteriores se puede observar la construcción del tipo de paquete en el que se envía la información de actualización y cambio de LU. Estos paquetes son IPv6 con Destination Option Header dirigidos a la IPv6 del MA.
Cuando el DMA 2 recibe alguno de estos mensajes crea una conexión con el TA para que actualice los campos del MH en la base de datos. El DMA lee de estos paquetes, los campos de Node_State, Home_Address y Cell_Iden para pasárselos al TA. La figura 5.13 muestra la conexión que se crea para la actualiación de los campos.

```
Frame 57 (1112 bytes on wire, 1112 bytes captured)
Linux cooked capture
Internet Protocol Version 6
    Version: 6
    Traffic class: 0x00
    Flowlabel: 0x00000
    Payload length: 1056
    Next header: TCP (0x06)
    Hop limit: 64
    Source address: ::1 (::1)
    Destination address: ::1 (::1)
Transmission Control Protocol, Src Port: 32892 (32892), Dst Port: 9000 (9000),
Seq: 1, Ack: 1, Len: 1024
    Source port: 32892 (32892)
    Destination port: 9000 (9000)
    Sequence number: 1    (relative sequence number)
    Next sequence number: 1025    (relative sequence number)
    Acknowledgement number: 1    (relative ack number)
    Header length: 32 bytes
    Flags: 0x0018 (PSH, ACK)
    Window size: 32768 (scaled)
    Checksum: 0x3d85
    Options: (12 bytes)
Data (1024 bytes)
```

**Figura 5.13.** Petición de actualización en BBDD como consecuencia de la llegada de un paging response, cambio de estado o cambio de área de localización.

# Conclusiones

Los objetivos de este TFC fueron 4. El primero era comprender y modificar la implementación del protocolo de transporte de paging IP existente. El segundo probar y determinar el rendimiento de dicha implementación buscando sus cuellos de botella. El tercero, implementar la arquitectura de paging IP que se especifica en [7]. Y finalmente, el cuarto era probar de manera conjunta el protocolo y la arquitectura de paging IP para comprobar su correcto funcionamiento.

Para poder realizar estos objetivos debimos empezar por entender el protocolo GeoPaging y para ello fue necesario adquirir nuevos conocimientos teóricos sobre IPv6, sobre el funcionamiento de los protocolos de multicast, unicast y de las redes celulares, entender los mecanismos de localización utilizados por estas redes, en definitiva todos aquellos conceptos descritos en el anterior TFC necesarios para poder continuar el nuevo trabajo aquí presentado.

Un paso importante en todo el proceso era asimilar todo el software que constituye Mcast para analizarlo posteriormente con diferentes pruebas. Entender y asimilar este software no es un proceso sencillo, ya que el código desarrollado en [1], es de unas dimensiones considerables, como puede verse en el CD anexo. Además, requiere del conocimiento de diversas librerías externas que utiliza y presenta elementos de programación complejos, como son por ejemplo, la programación concurrente con threads. Esta dificultad inicial para entender el código disminuyó gracias a encontrarnos con un código modular, es decir, dividido en funciones que facilitan el entender cada una de las partes del programa, y muy bien documentado que a su vez facilitó el encontrar referencias que nos ayudaran.

Paralelamente al aprendizaje de todos los conceptos se fueron realizando pequeñas pruebas de funcionamiento sobre Mcast para entrar en contacto con la implementación, ya que la mejor forma para entender el enrutado del protocolo era jugando con su implementación.

Una vez adquirimos los conceptos teóricos necesarios para ponerse a trabajar, empezamos por el primer objetivo del TFC, discutiendo sobre todos los temas que se dejaron abiertos, de manera que se pudiera empezar a mejorar el programa y cuando se tuviera mejorado empezar con las pruebas de funcionamiento y rendimiento.
Entre los planteamientos de mejora surgidos sobre Mcast, destacamos el método de agregación de rutas, el paso a memoria dinámica y la creación de un programa que enviara paquetes de paging con el formato que les corresponde a los mensajes de paging. Para mejorar cada uno de estos tres aspectos, previamente se discutió si realmente merecía la pena optimizarlos o directamente pasar al análisis de Mcast. El primer punto que se mejoró, el método para la agregación de rutas, fue un tema interesante de mejora, ya que con una pequeña modificación del script de configuración de las interfaces dummy se llegó a agregar un número de rutas muy superior al utilizado anteriormente. El otro aspecto modificado fue el paso a memoria dinámica, este

proceso fue más complicado que el anterior y nos demoró más tiempo analizar cómo realizarlo, ya que esto significaba trabajar con punteros que apuntaban a estructuras de un tamaño desconocido cuyo espacio se tenía que reservar. Este hecho,que a primera vista puede parecer trivial, no lo es, ya que en [1], se diseñó el software para que trabajara con memoria estática, es decir, con estructuras que contenían un número de elementos definidos y por lo tanto no había ningún problema al trabajar con su espacio de memoria ya que se creaba previamente porque se conocía. Esta característica limitaba el número de interfaces y rutas que se podían utilizar en el escenario representado, con el paso a memoria dinámica se mejoró esta limitación aunque esto implicó una dificultad a la hora de programar y un conjunto de nuevas pruebas de funcionamiento con las que obtuvimos los primeros resultados satisfactorios del análisis.

Una vez mejorado el software y probadas las modificaciones, se analizó su rendimiento sobre un escenario que ejemplificaba una posible red real. Los principales objetivos de estas pruebas eran valorar el rendimiento de Mcast, comparando los resultados con un protocolo de enrutado unicast, ver en qué manera afecta el hardware de las máquinas para su rendimiento y una vez analizado su rendimiento intentar descubrir los procesos que generan los posibles cuellos de botella en el código.

La principal dificultad en esta parte fue realizar el planteamiento de pruebas que se podían realizar para analizar el código, cómo determinar los puntos críticos del protocolo y el consumo de memoria de Mcast.

Las pruebas realizadas para el análisis de Mcast se dividen en tres tipos; todas ellas basadas en el esquema de la maqueta utilizado en [1]. Las primeras son pruebas de funcionamiento del software con las mejoras introducidas ya comentadas, ,donde los resultados demostraron que Mcast funcionaba como debía con las modificaciones, el segundo tipo son las pruebas de rendimiento de Mcast, para ello primero se hicieron pruebas con un protocolo de enrutado unicast con el que tener unos resultados de referencia y poderlos contrastar con los de Mcast. En esta prueba se enviaron pings y se capturaron los tiempos de proceso y transmisión con tcpdump, después se realizó esta misma prueba con Mcast, con lo que se pudieron ver las diferencias entre ambos protocolos.
Los resultados obtenidos entre los dos protocolos son bastante diferentes, obteniendo tiempos mucho mayores en el enrutado con Mcast. Fueron estos resultados los que no llevaron a preguntarnos en qué parte del código Mcast dedicaba más tiempo y por tanto generaba el cuello de botella. Aquí surge una nueva complejidad, cómo averiguar el cuello de botella del código. Para poder descubrir qué parte era la causante del mayor tiempo, pensamos en capturar diferentes trazas del código, para ello se utilizó la librería *time.h* .

La manera que se nos ocurrió para analizar el código por partes fue introducir diferentes marcas temporales según el código realizara unas funciones u otras, es decir, introducimos dentro del código marcas que dividían el programa en el tiempo de filtrado, el tiempo para comprobar el TTL, el tiempo para comprobar la interfaz parent, el dedicado para la comprobación de rutas y el tiempo que

tardaba en inyectar un paquete. Obteniendo que el 98% del tiempo total consumido recae en inyectar los paquetes a la red, esto se debe a la utilización de la librería libnet en este proceso. Descubrir este punto crítico abre la posibilidad de mejorar este proceso de inyección utilizando. Otros métodos alternativos, como son la utilización de los divert socket, los routing sockets, o la librería packet.h., Estas alternativas presentan como desventaja un aumento en la complejidad de la programación, ya que todas ellas trabajan a bajo nivel.

Finalmente con todos los resultados obtenidos se puede decir que el hardware es determinante para optimizar los tiempos de Mcast. La RAM y la CPU de las máquinas determinan los tiempos obtenidos en el filtrado y en el proceso del paquete pero no tanto en la inyección de éste en la red. Este hecho se debe a que Mcast utiliza para capturar los paquetes libpcap, una librería externa al SO, una vez los paquetes son leídos, vuelca los mensajes en memoria donde se procesan, por lo que la CPU de las máquinas son importantes en esta parte pero no al inyectar donde ya están procesados los mensajes, aquí son más determinantes otros aspectos, como son la RAM de las máquinas, influyente para coger los paquetes de memoria y enviarlos hacia las interfaces, y también son importantes los drivers de las tarjetas Ethernet, en los resultados se puede apreciar como los R1 y R2 que tienen las mismas tarjetas de red obtienen tiempos muy similares.

Otra característica a destacar en estos resultados, son las diferencias obtenidas entre las pruebas realizadas con tcpdump y las realizadas con las marcas temporales. Estas diferencias se debe a que aunque la resolución de la librería *time.h* es del orden de microsegundos, esta está limitada por la propía resolución del kernel de Linux cuya resolución es del orden de milisegundos.

Una vez analizado mcast empezamos a desarrollar la segunda parte de este TFC que recogió los dos últimos objetivos, no menos importantes que los anteriores. Estos objetivos recordemos que eran implementar una arquitectura real de paging y finalmente unir en un mismo escenario el protocolo Geopaging y la arquitectura. Para poder desarrollar esta arquitectura, se tuvieron que aprender nuevos conceptos, todos ellos relacionados con la arquitectura de paging que especifica [7]. Estos conceptos fueron aprender los diferentes elementos que formaban una arquitectura de paging, completamente desconocidos hasta el momento, el Tracking Agent y el Dormant Monitoring Agent, y aprender más sobre el Paging Agent y el Receptor Móvil, además de conocer todas las interfaces que unían estos elementos para poder implementar los mensajes enviados en ellas.

Para implementar la arquitectura se desarrollaron todos estos elementos según especifica el protocolo, usando el lenguaje de programación C, consiguiendo implementar una arquitectura real donde la comunicación entre los agentes fuera continúa y se enviaran cada uno de los mensajes que corresponde a cada interfaz. La complicación en esta implementación fue realizar un software capaz de establecer una comunicación continua donde los paquetes enviados por el Receptor Móvil hacia la red y los Mensajes de Paging fueran mensajes

IPv6 e ICMPv6 bien construidos, es decir, enviar mensajes de paging siguiendo el formato de un paquete IPv6 con Extensions Headers.

El desarrollo de los agentes fue paralelo a la implementación de los paquetes y a medida que se fueron creando, el código se fue testeando para comprobar su funcionalidad.

El resultado final de esta implementación es totalmente satisfactorio cumpliendo los requisitos del protocolo, prueba de ello son las pruebas de funcionamiento mostradas en el capítulo 5, donde se corrobora y se demuestra que la arquitectura funciona.

Finalmente podemos decir que en este trabajo hemos cumplido con todos los objetivos que se plantearon inicialmente, dejando como heredamos nosotros, una buena base para continuar trabajando en la arquitectura de paging e innovando con nuevos aspectos de ésta.

Para futuras líneas de desarrollo queda la optimización del código generado para implementar la arquitectura de paging, así como la creación de nuevas características de la arquitectura para las que el código está totalmente preparado. Algunas de estas nuevas funciones podrían ser la creación de diferentes políticas de paging según el tipo de MH o de tráfico que deba recibir. Actualmente el código está diseñado contemplado este aspecto, pero no se utiliza, esta mejora sólo debería utilizar un campo ya creado.

Actualmente el PA que tiene que enviar el paging no interpreta correctamente la home address que le envía el DMA, por lo tanto el paquete de paging que envía no contiene la Extensión Header con la home address del cliente. La implementación está preparada para coger este campo e introducirlo en el paquete como una Extensión Header por lo que no es un problema de creación del paquete de paging sinó de interpretación de la dirección ipv6 enviada por el DMA.

Paralelamente a estos aspectos en futuras líneas de desarrollo también se podría tener en cuenta el diseñar mensajes de carácter opcional definidos en el RFC, como pueden los que tengan que ver con la comunicación directa entre el MH y el TA, para que actualice la base de datos.

Una vez se dispone de una arquitectura de paging IP se podría estudiar su ingración con Mobile IPv6, el protocolo estandard del IETF para el sosporte de la movilidad, que carece de soporte de paging. Esta integración permitiria disponer de una red IPv6 que soportaria todas las funciones de una red celular de soporte de la moviladad ya comentadas en la introducción: traspaso, localización y paging.

A continuación detallaremos el impacto ambiental que representa este trabajo, para ello primero se describen los materiales utilizados y se relacionan con el consumo energético que ha supuesto la implementación.

El número de equipos que se han utilizado para implementar el escenario final de pruebas han sido 4 máquinas y un hub, además para cada una de las máquinas que representan los routers fueron necesarias tres targetas de red y sus respectivos cables para interconectarlos. Con todo esto queremos decir que la cantidad de material para desarrollar el proyecto fue elevada comparada con otros TFCs. A pesar de todo ello, la mayoría de los materiales se han reutilizado.

Dos de los ordenadores que representan los routers de nuestro escenario fueron ya reutilizados en el anterior TFC, así como el hub necesario para conectar el MH. Pero para implementar todo el escenario fue necesaria la adquisición de una màquina totalmente nueva, por lo que este hecho si que repercute negativamente en el medio ambiente. Esta compra fue necesaria ya que la màquina que hacía de router anteriormente estaba en muy malas condiciones y no nos era útil para realizar las funciones que debía, además en el laboratorio no había ningun otro equipo libre que cumpliera los requisitos necesarios.

Hemos de decir que la màquina antigua no se desechó completamente, ya que la utilizamos para realizar diferentes pruebas de envíos de paging. Asimismo la nueva màquina y el resto de la maqueta se continuarà utilizando en futuros proyectos.

Desde el punto de vista energético, la implementación y el uso de la maqueta utilizada en el laboratorio no representa un consumo de energía eléctrica demasiado elevado comparado con el ahorro de energía de los terminales móviles. Esto es debido a que con la implementación de una arquitectura de paging, los terminales pueden pasar a un estado inactivo en el que ahorran baterías y ser despertado únicamente cuando sea realmente necesario.

# Bibliografía y enlaces web

## REFERENCIAS

[1] Marcos Garcia Martí, TFC: “Programación de un protocolo multicast geográfico sobre IPv6”, 2004.

[2] R. Vidal, J. Paradells, "Geopaging: a novel multicast approach to delivery ip paging", PIMRC 2004 (15th IEEE International Symposium on Personal, Indoor and Mobile Radio Communications), Barcelona (Spain), 05–08 September 2004.

[3] A. Bar-Noy, I. Kesler and M. Sidi, “Mobile Users: To Update or not to Update?”, in *Proceedings of Annual Joint Conf. of the IEEE Computer and Communications Societies (INFOCOM)*, Toronto, Canada, pp. 570-576, June 1994.

[4] R. Vidal, J. Paradells and J. Casademont, “Labelling Mechanism to Support Distancebased Dynamic Location updating in cellular networks”, *IEE Electronics Letters*, vol. 39, no. 20, pp. 1471-1472, October 2003.

[5] W. Richard Stevens, “*UNIX Network Programming, Vol 1 2nd Edition*”, Prentice-Hall PTD, Upper Saddler River (New Jersey, EEUU), 1998.

[6] RFC 3132: Dormant Mode Host Alerting ("IP Paging") Problem Statement

[7] RFC 3154: Requirements and Functional Architecture for an IP Host Alerting Protocol

[8] Braden, R., "Requirements for Internet Hosts Communication Layers", STD 3, RFC 1122, October 1989

## BIBLIOGRAFIA

· Warren W. Gay, “*Linux Socket Programming*”, Ed. Que, Indianápolis (Indiana, EEUU), 2000.
· W. Richard Stevens, “*Advanced Programming in the UNIX environment*”, Ed. Addison-Wesley, Indianápolis (Indiana, EEUU), 1993, última edición 2002.
· W. Richard Stevens, “*UNIX Network Programming, Vol 1 2nd Edition*”, Prentice-Hall
PTD, Upper Saddler River (New Jersey, EEUU), 1998.
· Dave Kosiur, “*IP multicasting*”, Ed. Wiley, New Cork (EEUU), 1998.
· Douglas E. Comer “*Internetworking with TCP/IP, Vol 1 4th Edition*”, Prentice Hall,
Upper Saddler River (New Jersey), 1999

## ENLACES WEB

RFC Estándar IPv4
http://www.ietf.org/rfc/rfc0791.txt
RFC Estándar IPv6
http://www.ietf.org/rfc/rfc2460.txt
RFC Basic Socket Interface Extensions for IPv6
http://www.ietf.org/rfc/rfc3493.txt
RFC Estándar IPv6
http://www.ietf.org/rfc/rfc3542.txt
IGMPv2
http://www.ietf.org/rfc/rfc2236.txt
ICMPv6
http://www.ietf.org/rfc/rfc2460.txt
Multicast Listener Discovery
http://www.ietf.org/rfc/rfc2710.txt
Especificación Básica de XCAST
http://www.ietf.org/internet-drafts/draft-ooms-xcast-basic-spec-05.txt
Información sobre IPv4
http://www.networksorcery.com/enp/protocol/ip.htm
Información sobre IP Multicast
http://www.cisco.com/univercd/cc/td/doc/cisintwk/ito_doc/ipmulti.htm
Multicast Backbone
http://www-itg.lbl.gov/mbone/
Internet con IPv6
http://www.6bone.net/
Proyecto de MBONE sobre IPv6
http://www.m6bone.net/
Encaminando de un paquete a través del kernel
http://www.auto.ucl.ac.be/~guffens/doc/path_packet.pdf
Kernel de Linux
http://www.kernel.org
Gentoo Linux
http://www.gentoo.org
FreeBSD
http://www.freebsd.org
Libpcap
http://www.tcpdump.org
Libnet
http://www.packetfactory.net/Projects/Libnet/
Anjuta (IDE de programación utilizado)
http://anjuta.sourceforge.net/
Programming in C: UNIX System Calls and Subroutines using C
http://www.cs.cf.ac.uk/Dave/C/CE.html

Escola Politècnica Superior
de Castelldefels
UNIVERSITAT POLITÈCNICA DE CATALUNYA

# ANNEXOS

## INDICE DE ANEXOS

# ANEXO I

# Instalación de Mcast en un PC

## I.1. Instalación

El código del programa ha sido instalado desde la distribución Gentoo Linux, este anexo explica la instalación del código en cualquier distribución linux, ya sea Gentoo o cualquier otra para poder ejecutar Mcast.

El requisito previo que se debe realizar para la correcta ejecución del programa será la instalación de todas las librerías externas necesarias, la instalación y configuración del programa de enrutado *zebra*, del protocolo de enrutado *ripng* y finalmente proceder a la instalación del código del programa Mcast.

### I.1.1. Instalación de librerías externas

El programa Mcast desarrollado depende de algunas librerías externas que debe tener el sistema si deseamos que el Mcast funcione correctamente. Las librerías externas necesarias para su funcionamiento son *Libpcap y Libnet.* Como se comentó en [1], estas librerías son necesarias para la captura, inyección y creación de paquetes respectivamente.

A continuación detallaremos su instalación en Gentoo y en cualquier otra distribución linux.

Si se trabaja en Gentoo Linux, este paso es tan sencillo como teclear en el terminal: #emerge libpcap y #emerge libnet, pero si se trabaja en otras distribuciones, hemos de bajar el código fuente desde la página de cada proyecto.
La página del proyecto para la librería Libcap es: http://www.tcpdump.org, desde allí se descarga la última versión del código publicado, que en este caso es la versión 0.8.3. Una vez descomprimido, se ejecuta dentro del directorio creado, que por defecto se llamará libpcap-0.8.3, el script nombrado configure. Una vez ejecutado, ejecutamos make y finalmente make install. Finalmente, después de este último paso, la librería Libpcap ya está instalada en nuestro sistema.
Siguiendo el ejemplo de Libpcap, para la instalación de la librería Libnet, se ha de seguir el mismo proceso, descargar la librería de la página del proyecto, http://www.packetfactory.net/Projects/Libnet, actualmente la última versión de *Libnet* es 1.1.2-rc6. Descomprimir el código descargado, por defecto se guardará en una carpeta llamada libnet y de la misma manera que en Libcap, ejecutar desde el terminal el script configure, el make y finalmente make install. Una vez hecho esto, la libreria Libnet estará instalada en el sistema.

### I.1.2. Instalación de zebra y ripng

Una vez finalizada la instalación de las librerías externas, se instala *Zebra. Zebra* es un programa de enrutado que permite la utilización de un gran número de protocolos de enrutado unicast, y que juntamente con *RIPng* proporcionará la configuración IPv6 unicast del escenario del proyecto.

La instalación en Gentoo Linux, es tan sencilla como antes y simplemente es suficiente con escribir en consola: #emerge zebra. En cambio, si se desea instalar *Zebra* en cualquier otra distribución, debemos seguir un proceso similar al explicado en el punto anterior.
El primer paso, como siempre, es descargar el código desde la página del proyecto *Zebra*, este proyecto se puede encontrar en: http://www.zebra.org, la última versión del programa es la 0.94. Una vez descargado, descomprimir el archivo, se creará una carpeta, que por defecto se llamará zebra, ejecutar desde el terminal el script configure, seguido del make y finalmente make install. Una vez realizados estos pasos ya dispondremos de *Zebra* instalado en nuestro ordenador.
La configuración tanto de *Zebra*, como de *RIPng*, necesarios para la ejecución del código, está explicada en el Anexo 1 de [1].

Finalmente, y después de haber realizado todos los pasos descritos, el sistema estará a punto para instalar el código del programa Mcast.

### I.1.3. Instalación del código

La instalación de Mcast se puede realizar de dos formas, instalarlo de forma manual o utilizar cualquier software de programación para linux como puede ser Anjuta, encontramos más información sobre este software en la págna del proyecto: http://www.anjuta.org/.

Antes de explicar el proceso de instalación, es importante saber exactamente cuántos archivos componen el programa. Mcast se compone de 2 librerías (archivos .h), 9 archivos de código fuente (archivos .c) y un archivo Makefile para su compilación. Una vez conocidos el número de archivos, procederemos a la explicación. Si optamos por la instalación manual, el primer paso es descomprimir el archivo *mcast.tar.gz, este es el paquete que compone todos los archivos necesarios para la ejecución de Mcast,* para ello introduciremos en el terminal el siguiente comando: # tar zxvf mcast.tar.gz

El resultado obtenido con este comando es:

```
root(0)localhost:~
## tar zxvf mcast.tar.gz
mcast/
mcast/src/
mcast/src/UTILS.c
mcast/src/utils.c
mcast/src/RT.c
mcast/src/PCAP.c
mcast/src/NET.c
mcast/src/Makefile
mcast/src/main.c
mcast/src/libnetlink.c
mcast/src/IFS.c
mcast/src/GEOPAG.c
mcast/include/
mcast/include/netlink.h
mcast/include/libnetlink.h
mcast/include/GEOPAG.h
mcast/include/defs.h
##
```

**Figura I.1:** Resultado de descomprimir el archivo mcast.tar.gz

Si observamos la salida por consola podemos ver exactamente el nombre de todos los archivos necesarios para la ejecución de mcast. Al descomprimir el archivo mcast.tar.gz, se crea un directorio llamado mcast, éste a su vez se compone de dos carpetas, *include y src,* estas carpetas corresponden a los archivos de librerías y de código fuente necesarios. Con éste comando tendremos todo el código instalado ya en el sistema, a continuación para poder ejecutarlo, primero se ha de compilar con la ayuda del archivo makefile incorporado en el programa.
Para poder compilar el código, nos situamos dentro del directorio donde se encuentra el archivo Makefile, es decir, en la ruta: #/mcast/src, en ese directorio ya se puede ejecutar desde consola #make, con éste se compila el programa, creando un ejecutable, se llamará ./Mcast, ya que es el nombre que se le ha dado en el archivo makefile.

Si lo que deseamos es utilizar un programa de programación como puede ser el Anjuta, debemos crear un proyecto nuevo tipo *Aplicación de consola*, y de forma similar a la instalación manual, añadimos al proyecto todo el código del paquete descomprimido. El requisito para añadir el código al nuevo proyecto creado, es que cada uno de los archivos de la carpeta include descomprimida, se deben incluir al proyecto y los archivos de la carpeta src se deben añadir como archivos de código fuente. De esta forma podemos compilar y construir el programa, generando el mismo ejecutable con el que podremos posteriormente ejecutar Mcast.

# ANEXO II

# Especificaciones del código

## II.1. Introducción

En este anexo se explica más detalladamente lo que ha sido la programación específica de los diferentes agentes y del receptor móvil.
Se describirá en profundidad el funcionamiento de los diferentes elementos junto a las funciones que los forman.

## II.2. Mobile Agent

Como ya se ha comentado antes, el MA es la conjunción de los tres agentes, DMA, TA y PA, que estarán todos juntos trabajando en el PC portátil.

### II.2.1. Dormant Monitoring Agent

El DMA está formado por dos programas independientes, esto se ha implementado así gracias a que uno de los programas recibe un paquete para un MH que será el paso por parámetro de la dirección IPv6 del MH a quién va dirigido el supuesto paquete y este programa únicamente se ejecutará cuando nosotros decidamos enviar paquetes al host. El segundo DMA implementado está siempre escuchando a que llegue algún tipo de paquete, tanto del móvil cambiando de estado o de área de paging, como paquetes de respuesta al paging. Esta necesidad de tener un DMA únicamente ejecutándose cuando queramos buscar a un terminal en la base de datos y otro continuamente escuchando hace que se hayan programado en programas diferentes.

A continuación mostramos una tabla que recoge las funciones que implementan al DMA, muchas de ellas se reutilizan todos los agentes. Más adelante se introducirá un diagrama que representa la comunicación entre los agentes, para ayudar a comprender la comunicación entre éstos.

**Tabla II.1.** Descripción de las funciones que constituyen el DMA (I)

| DMA | | |
|---|---|---|
| DMA 1 | | |
| **Archivo** | **Funciones** | **Descripción** |
| dormant-utils.c | Las funciones que forman el dormant-utils.c, son las mismas que constituyen todos los archivos xutils.c. Estas funciones ejecutan rutinas de carácter general utilizadas por todos los agentes. No todas de ellas se utilizan en cada uno de los elementos. | |
| | `int UTIL_get_hexchar_val(char h);` | Convierte un string a hexadecimal. |
| | `void UTIL_hexstring_to_ipv6(struct in6_addr *ipv6, char *str);` | Convierte un string a una dirección ipv6 bien construida. |
| | `int UTIL_get_number_of_lines(FILE *fd, int buf_len).` | Extrae el número de líneas de un fichero. |
| | `int UTIL_compare_ips(struct in6_addr*direccion_bbdd, struct in6_addr *direccion_cliente);` | Compara dos direcciones ipv6 |
| | `void UTIL_init_client_mem(CLIENT_t *client_t);` | Inicializa el espacio en memoria necesario para un cliente de paging. |
| | `Void UTIL_create_client_more_mem(CLIENTS_t *client_t, int mem);` | Crea espacio en memoria para el número de clientes pasado por referencia. |
| | `void UTIL_liberar_mem(CLIENTS_t *clients_t);` | Libera el espacio en memoria previamente inicializado. |
| dormant-client.c | Estas funciones son las que crean la conexión del cliente del DMA 1 para conectarse con el TA a quién le pasará la home address del cliente que tiene que buscar en su bb.dd. La conexión se crea por el puerto 8000. | |
| | `Int tcpclient();` | Crea el socket del cliente por el puerto definido. |
| | `void str_cli(clients_for_paging *str_cli_pag);` | Envía por el socket al cliente de paging pasado por referencia. |
| dormant-main.c | `int main(int argc, unsigned char **argv);` | Programa principal que recibe por parámetro la dirección del cliente que enviará al TA para que busque en la bbdd. |

**Tabla II.2** Descripción de las funciones que constituyen el DMA (II)

| DMA 2 | | |
|---|---|---|
| **Archivo** | **Funciones** | **Descripción** |
| dclient.c | Las funciones de este archivo siguen el criterio de dormant-client.c Este cliente es el que se conecta con el TA para que busque al cliente que tiene que actualizar en la bbdd. El puerto definido en esta conexión es el 9000. | |
| dclient_paging.c | Las funciones que forman este archivo siguen el criterio de dclient.c, en esta conexión se envían los datos del cliente al PA para que le envíe el paging. El puerto definido en esta conexión es el 8010. | |
| dserver.c | Este archivo crea el servidor TCP que escucha al cliente cuando cambia de estado o actualiza el área de paging. El puerto en el que escucha este servidor es el 8005. | |
| | `void str_echo(int sockfd);` | Escucha al cliente que le pasará la información de cambio de estado o de LU y lo guarda en una estructura. |
| | `Void *dserver(void *args);` | Crea el socket TCP por un puerto especifico. |
| dpcap.c | Las funciones que constituyen este archivo se han implementado para capturar los paquetes por la interfaz que tenga activa el MA, filtrando aquellos que únicamente le interesan para después procesarlos. | |
| | `Void PCAP_receptor_msg(u_char *args, const struct pcap_pkthdr, const u_char *packet);` | Captura paquetes por una determinada interfaz y los filtra. |
| | `int PCAP_handle_udp(u_char *args, const struct pcap_pkthdr, const u_char *packet);` | Estas funciones tratan los paquetes para identificar de qué tipo son. |
| | `int PCAP_handle_icmp6(u_char *args, const struct pcap_pkthdr, const u_char *packet);` | |
| | `int PCAP_handle_ethernet(u_char *args, const struct pcap_pkthdr, const u_char *packet);` | |
| | `int PCAP_handle_ipv6(u_char *args, const struct pcap_pkthdr, const u_char *packet);` | |
| dmain.c | `void *dormant_captura(void *argc);` | Captura los paquetes filtrándolos por una determinada IP y realiza un loop infinito donde llama a la subrutina `PCAP_receptor_msg();` |
| | `void main();` | Programa principal que crea los hilos para escuchar continuamente la red. |

El siguiente diagrama describe los procesos seguidos por el DMA, hemos dividido en dos figuras diferentes para facilitar la comprensión.

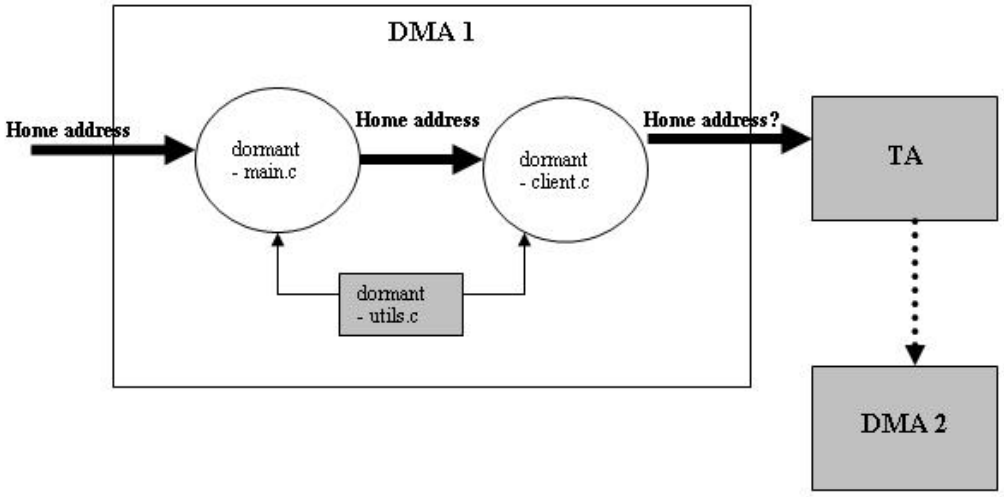


**Figura II.1.** Diagrama funcional del DMA1

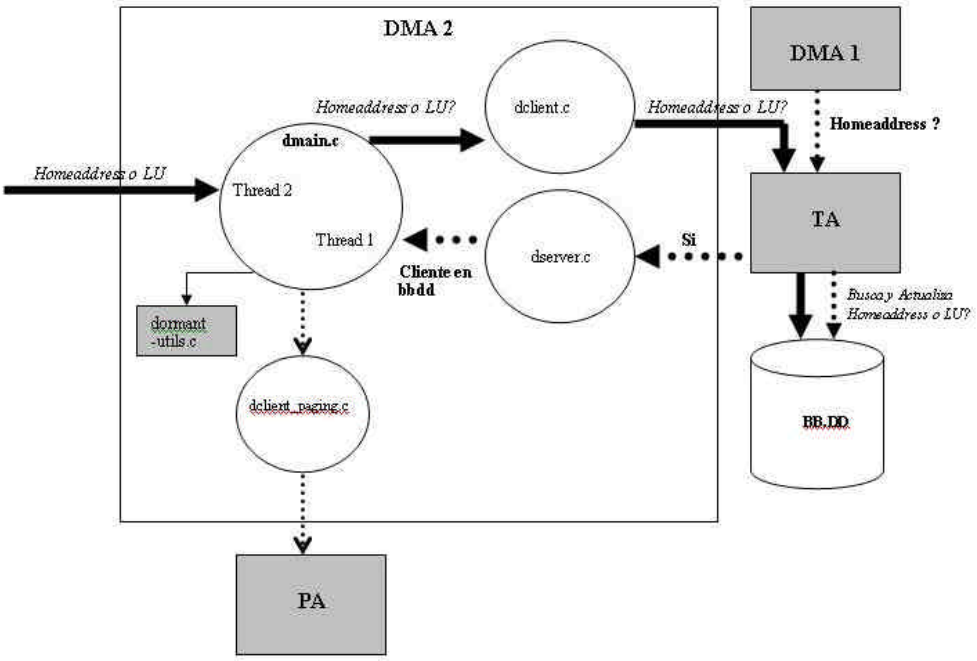


**Figura II.2.** Diagrama funcional del DMA

## II.2.2. Tracking Agent

El TA es el agente encargado de manipular la base de datos, es el único de los elementos que puede actualizarla. Este elemento debe escuchar al DMA ante el envío de los datos de cualquier cliente ya sea para actualizar su información en la base de datos o para buscarlo en ella.
La tabla II.2 muestra las funciones que lo forman.

**Tabla II.3.** Descripción de las funciones que constituyen el TA (I).

| TA | | |
|---|---|---|
| **Archivo** | **Funciones** | **Descripción** |
| `tutils.c` | Las funciones que forman este archivo son las mismas que las mostradas en la tabla II.1, en el archivo dormant-utils.c | |
| `tclitentdma2.c` | Estas funciones son para establer la comunicación con el DMA 2 por el puerto 8005. | |
| | `int tcpclient();` | Crea el socket del cliente por un puerto específico. |
| | `Void str_cli(clients_for_paging *str_cli_pag);` | Envía por el socket el cliente de paging pasado por referencia. |
| `tserverdma1.c` | Este archivo crea el servidor TCP que escucha al cliente del DMA 1 por el puerto 8000. | |
| | `void str_echo(int sockfd);` | Escucha al cliente que le pasará la información del cliente que tiene que buscar en la base de datos. |
| | `void *dserver(void *args);` | Crea el socket TCP por un puerto especifico. |
| `tserverdma2.c` | Este archivo crea el servidor TCP que escucha al cliente del DMA 2 por el puerto 9000. | |
| | `void str_echo(int sockfd);` | Escucha al DMA que le pasará la información del cliente a actualizar en la base de datos. |
| | `void *dserver(void *args);` | Crea el socket TCP por un puerto especifico. |

**Tabla II.4.** Descripción de las funciones que constituyen el TA (II)

| TA | | |
|---|---|---|
| **Archivo** | **Función** | **Descripción** |
| track.c | Este archivo contiene todas las funciones que debe realizar el TA para manipular la base de datos. | |
| | `void TRACK_parsea_from_bbdd(CLIENTS_t *client_t);` | Abre la base de datos y obtiene el número de clientes en ella. |
| | `Void TRACK_BUSCA_CLIENTES_DMA1(CLIENTS_t *client_home, unsigned char *new_client);` | Busca al cliente que le pasa el DMA 1. |
| | `void TRACK_BUSCA_CLIENTES(CLIENTS_t *client_home, unsigned char Ç*new_client, unsigned char *celda, int estat);` | Función que busca al cliente que le envía el DMA 2 para actualizar su estado i el identificador de celda. |
| | `void TRACK_ACTUALIZA_BBDD(CLIENTS_t *bbdd_actualizada, int i);` | Actualiza el estado de un cliente. |
| | `void TRACK_ACTUALIZA_ID_CELDA(unsigned char *celda, int i);` | Actualiza el identificador de celda. |
| | `Int TRACK_COMPRUEBA_ESTADO(clients_for_paging *cliente_en_bbdd);` | Comprueba el estado de los clientes pasados por referencia. |
| Tmain.c | `int main (int argc, unsigned char **argv);` | Programa principal que crea los hilos para establecer una conexión continua con el DMA. |

La siguiente figura muestra los procesos de las subrutinas ejecutadas por el TA. Cada tipo de flecha representa un proceso diferente ejecutado por las diferentes subrutinas del TA.

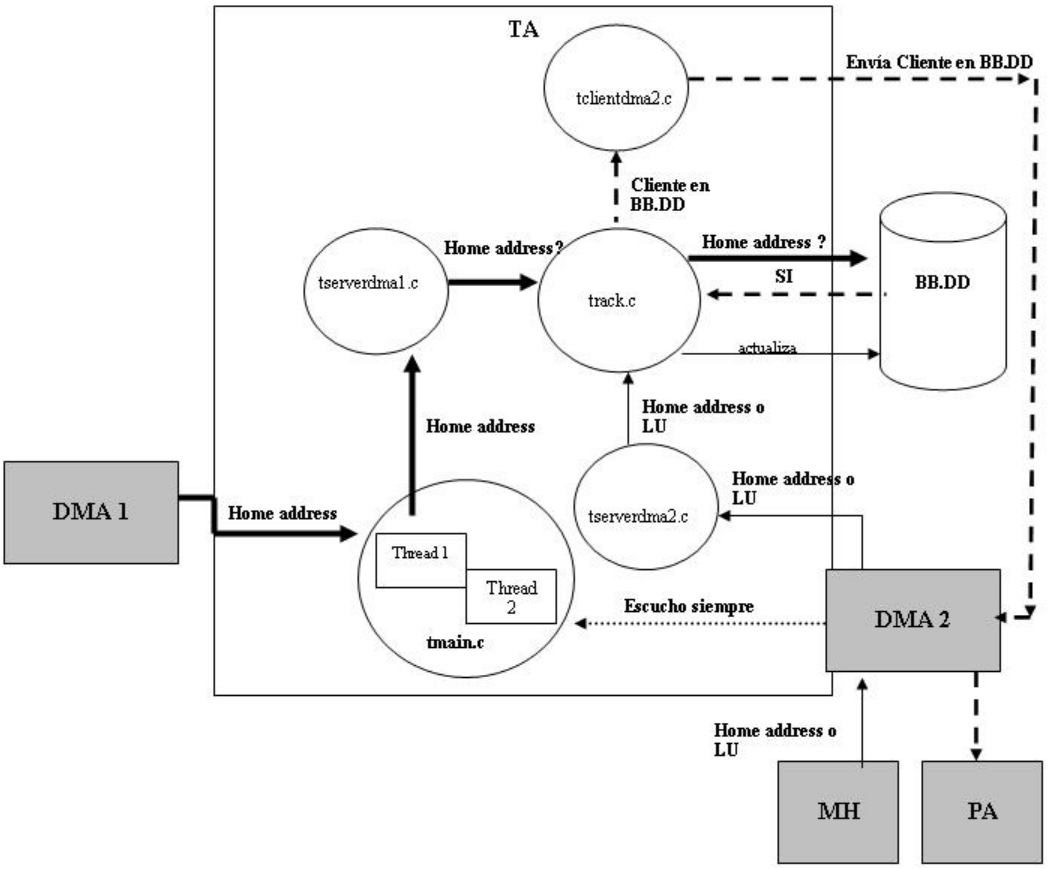


**Figura II.3.** Diagrama funcional del TA

## II.2.3. Paging Agent

El PA es el elemento que escucha al DMA en espera de tener que despertar algún MH. Cuando recibe un cliente del DMA, éste le envía el paging, para ello la implementación realizada es la mostrada en la tabla II.5.

**Tabla II.5.** Descripción de las funciones que constituyen el PA

| PA | | |
|---|---|---|
| **Archivo** | **Funciones** | **Descripción** |
| Main.c | `Void send_paging(clients_for_paging *msg_client);` | Función que envía el paging al cliente pasado por referencia. El paquete contendrá toda la información del cliente (identificador de celda, home address, estado). |
| | `void str_cli(int sockfd);` | Escucha al DMA a la espera de recibir la información del cliente al que tiene que enviar el paging. Esta función llamará a la subrutina send_paging(). |
| | `int main(int argc, char *argv[]);` | Programa principal que crea el socket para escuchar al DMA. Éste llama a str_cli(), para que escuche al cliente. |

El siguiente diagrama representa el proceso que realiza el PA.

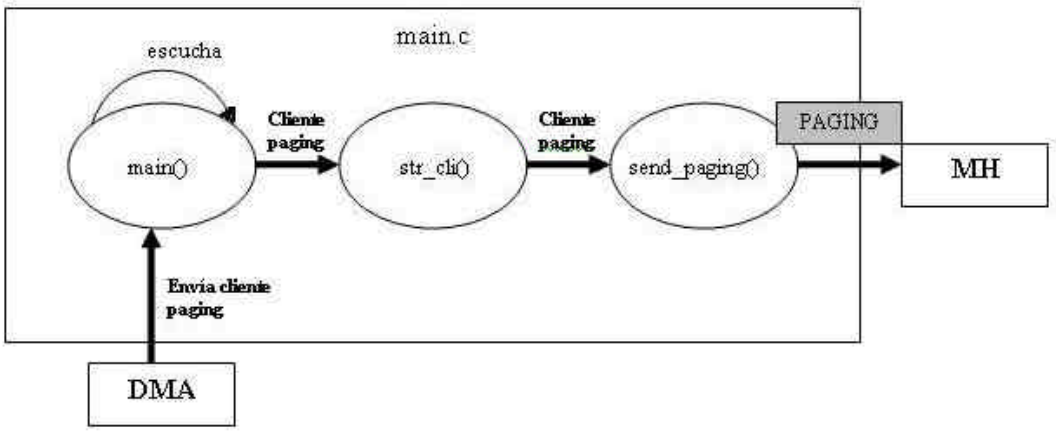


**Figura II.4**. Diagrama funcional del proceso de paging

## II.3. Mobile Host

En este apartado se explican las variables y funciones del MH utilizadas para recibir, guardar en memoria, procesar y reenviar paquetes.
La descripción de este elemento se realizará de una manera diferente a los puntos anteriores ya que para la realización de este código se ha partido de la base del código fuente de Mcast, siendo la parte de captura de un paquete de paging prácticamente igual que el anterior software. Solo decir que se ha prescindido en este caso de los archivos *RT.c*, puesto que el MH en ningún momento necesitará conocer las rutas hacia otras máquinas puesto que no es su función y del archivo *GEOPAG.c* ya que éste es solo un receptor de mensajes de paging.
Las principales diferencias en esta parte con respecto el programa Mcast es la utilización de las funciones RAW_set_home_address (char *home_address) y *RAW_set_cell_address(char *cell_address)* que como sus respectivos nombres indican sirven para inicializar su Home Address y para inicializar el área de Localización donde se encuentra.
En este momento y antes de activar los contextos para la captura de paquetes se crea un thread que llamara a la funcion **RAW_check_time()* que servirá exclusivamente para que el MH cambie cada 30 segundos de estado o cada 55 segundos cambie de area de localización. En la notificación de estos cambios entrará en uso la funcion *RAW_send_to_DMA()* que servirá para crear el paquete que notificará al DMA, un cambio de estado o un cambio en el área de localización.

Ahora bien, si volvemos al código fuente común a Mcast observaremos que la captura de mensajes de paging se ejecutará igual que en este, por ejemplo, imaginemos que llega un mensaje de paging por la interfaz activa del MH. El contexto correspondiente se activará, pues ejecutamos la *función pcap_loop(p,-1, PCAP_receptor_msg, parent)*, lo que significa que se ejecutará un bucle infinito *(parámetro -1)* con la rutina *PCAP_receptor_msg(),* y con el filtro incluido en *p*, pero sólo recibirá por la interfaz *parent*. Dicho filtro sirve para recibir paquetes sólo del rango multicast especificado.
En la rutina *PCAP_receptor_msg()* recibimos un puntero a *packet,* que es el paquete recibido. El resto de información, como los bytes recibidos, la marca de tiempo o la interfaz de llegada, se guarda en la estructura *pcap_pkthdr.*
Así pues, teniendo en un buffer de memoria el contenido del paquete, marcamos las posiciones de las diferentes cabeceras en una serie de variables: *eth_hdr, ip6_hdr, post_ip6_hdr* (justo después de la cabecera mínima de 40 bytes), *hbh_opt_pointer* (si la hay), *dest_opt_pointer* (si la hay), *L4_pointer* (UDP o TCP). También se guarda una copia de la cabecera ipv6 en memoria.
Una vez comprobados que los campos son correctos, es decir, que tenemos un mensaje de paging adecuado (ver figura 4.7 del capitulo 4) se ejecutará la función *RAW_compare_home(char *home_address)* que comparará la home address contenida en el paquete de paging recibido con la que anteriormente le habremos asignado al MH. Si las dos direcciones son iguales, el MH cambiará a estado activo mediante la función *RAW_change_state(1)* y

seguidamente enviará la confirmación del cambio de estado al DMA mediante *RAW_envia_icmp6(in6_addr *src).*

Los archivos *IFS.c, NET.c, PCAP.c y UTILS.c son* los mismos a los utilizados en Mcast, ver [1].

**Tabla II.6**. Funciones del MH

| MH | | |
|---|---|---|
| **Archivo** | **Funciones** | **Descripción** |
| RAW.C | Las funciones RAW_() se utilizan para realizar todas aquellas rutinas definidas en el MH. | |
| | `void RAW_envia_icmp6(struct in6_addr *dst_addr);` | Envía un paquete icmpv6 como respuesta al paging. |
| | `void RAW_change_state(int new_state);` | Cambia el estado del MH cada 30 segundos. |
| | `void_RAW_sendto_DMA();` | Envía al DMA la actualización de estado o de LU. |
| | `void RAW_set_home_address(char *home_str);` | Inicializa una home address. |
| | `void RAW_set_cell_address(char *cell_str);` | Inicializa el identificador de celda de donde se encuentra el MH. |
| Main.c | `void *Nuevo_contexto(void *ifaz_up);` | Crea tantos contextos como interfaces tenga activas el MH. |
| | `int main( int argc, char **argv);` | Este es el programa principal que crea los hilos para inicializar la home address, el identificador de celda y poder actualizarlos cada 30 y 55 segundos respectivamente. |

**Figura II.5.** Diagrama funcional MH