TREBALL DE FI DE CARRERA

TÍTOL: Programación de un protocolo multicast geográfico sobre IPv6

AUTOR: Marcos García Martí

DIRECTOR: Rafael Vidal Ferré

DATA: 6 de Setembre de 2004

Títol: Programación de un protocolo multicast geográfico sobre IPv6

Autor: Marcos García Martí

Titulació: Ingeniería Técnica de Telecomunicaciones

Especialitat: Telemática

Pla: 2000

Director: Rafael Vidal Ferré

Departament: ENTEL

Vist i plau,

Director del TFC

Registre:

Títol: Programación de un protocolo multicast geográfico sobre IPv6

Autor: Marcos García Martí

Director: Rafael Vidal Ferré

Data: 6 de Setembre de 2004

Resum

Parece claro que la denominada cuarta generación de redes celulares (4G) tendrá como ejes estratégicos el empleo de múltiples interfaces radio y la utilización de IP como protocolo de transporte de datos y señalización. De hecho IP ya es el protocolo de interconexión universal, y las redes están evolucionando hacia "todo IP" (All-IP). La adopción de IPv6 como protocolo de red en lugar de IPv4 pondrá fin a la actual escasez de direcciones, además de aportar mejoras significativas en aspectos como la seguridad y el soporte de la movilidad. Es por todo ello que se espera que IPv6 sea el protocolo de red de la 4G de redes celulares.

Los protocolos existentes de comunicaciones IP inalámbricas no dan soporte a la movilidad de los llamados nodos durmientes. Éstos son nodos sin comunicaciones en curso que pasan a un estado donde reducen la monitorización del medio radio para ahorrar baterías. Por eso en ese estado el nodo móvil en lugar de informar a la red de cada cambio de celda que realiza se limita a informar sólo cuando cambia de un grupo predefinido de celdas denominado a otro. Estos grupos se denominan áreas de localización (*Location Areas*, LAs). Esto conlleva que la red tenga un método de búsqueda llamado paging que permite averiguar exactamente en qué celda del LA se encuentra un nodo durmiente para poder avisarle, por ejemplo, cuando tiene una llamada.

Este proyecto pretende implementar una nueva propuesta de protocolo de paging a nivel IP llamada Geopaging. Se trata de un protocolo de multicast para IPv6 que permite el transporte de los mensajes paging desde su generador, el agente de paging (*Paging Agent*, PA) hasta las celdas que forman parte de la LA a la que pertenece el terminal mòvil..

Los objetivos de este trabajo son principalmente tres. El primero es entender el protocolo y desarrollar un programa que lo implemente. El segundo es hacer pruebas y depurar el programa para que funcione adecuadamente. Y finalmente el último objetivo es dejar una documentación que sirva para mejorar la implementación en un posterior TFC y conseguir un entorno funcional de paging IP basado en GeoPaging.

Summary

Nowadays seems like next generation of cellular networks (4G) will use multiple radio interfaces and IP as network protocol for transmission of data and signaling. Actually IP is already the universal interconnexion protocol, so all the networks are evolving into an All-IP environment. The adoption of IPv6 instead of IPv4 as the main network protocol will end the lack of directions, besides improving some factors like security and mobility support. Therefore IPv6 is expected to become the protocol of 4G networks.

The current IP wireless network protocols do not support dormant-node mobility. They are nodes that deactivate their unused radio interface because of power saving. This is the reason the mobile node only notifies the network when they trespass their Location Area (LA), a predefined group of cells, instead of notifying every cell trespassing. A LA change activates a mechanism called paging, which its main purpose is to find the mobile node when it has incoming calls or data.

This project expects to implement a novel proposal of IP paging protocol, called GeoPaging. It is a multicast IPv6 protocol that transports paging messages from their generator the Paging Agent (PA) to the cells in the LA where the mobile node is.

The main goals of this Project are three. The first of them is the understanding the paging protocol and development of a program to implement it. The second one is to make tests and debug the program to correct it. Finally the last objective is to document everything to improve the program in a future project, making possible the deployment of a functional GeoPaging based IP Paging environment.

DEDICATORIA

Me gustaria dedicar este TFC a mi familia, por apoyarme durante este tiempo y ayudarme en todo momento. También se lo dedico a mis amigos por respetar mis periodos de trabajo intenso, y a mis compañeros de laboratorio, tan receptivos y colaboradores con mi proyecto.

Y sobretodo dedicárselo a mi director de proyecto, Rafael Vidal, padre de este trabajo y de las ideas expuestas en él, por haberme ayudado tanto y por su paciencia ante mis tropiezos.

ÍNDICE

INTRODUCCION

Parece claro que la denominada cuarta generación de redes celulares (4G) tendrá como ejes estratégicos el empleo de múltiples interfaces radio y la utilización de IP como protocolo de transporte de datos y señalización. De hecho IP ya es el protocolo de interconexión universal, y las redes están evolucionando hacia "todo IP" (All-IP). La adopción de IPv6 como protocolo de red en lugar de IPv4 pondrá fin a la actual escasez de direcciones, además de aportar mejoras significativas en aspectos como la seguridad y el soporte de la movilidad. Es por todo ello que se espera que IPv6 sea el protocolo de red de la 4G de redes celulares.

Con este escenario, múltiples tecnologías celulares y una sola de red, parece lógico que algunas funciones de soporte de la movilidad comunes a todas las redes celulares se realicen a nivel IP. En este sentido el IETF estandarizó el protocolo Mobile IP (MIP) para permitir las comunicaciones IP, aún habiendo cambios de interfaces radio, de manera transparente al usuario y manteniendo sus conexiones activas.

Pero MIP no da soporte a la movilidad de los llamados nodos durmientes. Éstos son nodos sin comunicaciones en curso que pasan a un estado donde reducen la monitorización del medio radio para ahorrar baterías. Por eso en ese estado el nodo móvil en lugar de informar a la red de cada cambio de celda que realiza se limita a informar sólo cuando cambia de un grupo predefinido de celdas denominado a otro. Estos grupos se denominan áreas de localización (*Location Areas*, LAs). Esto conlleva que la red tenga un método de búsqueda llamado paging que permite averiguar exactamente en qué celda del LA se encuentra un nodo durmiente para poder avisarle, por ejemplo, cuando tiene una llamada.

Este proyecto pretende implementar una nueva propuesta de protocolo de paging a nivel IP llamada Geopaging [1]. Se trata de un protocolo de multicast para IPv6 que permite el transporte de los mensajes paging desde su generador, el agente de paging (*Paging Agent*, PA) hasta las celdas que forman parte de la LA a la que pertenece el terminal mòvil..

La ventaja que ofrece GeoPaging contra otras alternativas de paging IP es que permite el uso de técnicas de paging dinámico basadas en distancia. Se denominan dinámicas porque se define una LA para cada terminal en función de sus patrones de movilidad y llamadas, y basadas en distancia porque el parámetro que determina el envío de un mensaje de cambio de LA es la distancia en celdas. Estas técnicas han demostrado ser las más eficientes que las estrategias basadas en áreas de paging estáticas, y requieren mucha menos señalización, tal y como se explica en [2]. Sin embargo presentan un importante problema: el cálculo por parte del móvil de la distancia en celdas. Geopaging supera este problema mediante la utilización de un mecanismo de etiquetaje de las celdas [3]. A partir de ahí se generan direcciones IPv6 unicast y multicast que permiten la creación de las tablas de encaminamiento, y con ellas ya se puede enrutar los mensajes de paging.

Este protocolo utiliza multicast de una forma distinta a la habitual. Como se comentará en esta memoria, los protocolos existentes de IP multicast intercambian información sobre qué cantidad de usuarios hay en cada grupo multicast, y crean un árbol de distribución de paquetes para cada grupo basándose en sus usuarios activos. Este proceso es inviable para paging pues si identificamos una celda con un grupo, donde los usuarios están constantemente entrando y saliendo de la celda, se necesitaría mucho intercambio de información relativa a gestión de usuarios. En cambio en GeoPaging la creación del árbol

multicast se basa en la información de celda implícita en la dirección IPv6 de destino y en la distancia de paging indicada en una extensión de la cabecera. Por eso no se necesita intercambiar información, proporcionando así los beneficios de multicast pero evitando sus inconvenientes.

Los objetivos de este trabajo son principalmente tres. El primero es entender el protocolo y desarrollar un programa que lo implemente. El segundo es hacer pruebas y depurar el programa para que funcione adecuadamente. Y finalmente el último objetivo es dejar una documentación que sirva para mejorar la implementación en un posterior TFC y conseguir un entorno funcional de paging IP basado en GeoPaging.

La primera parte de esta memoria presenta los fundamentos teóricos utilizados en el TFC. El primer capítulo habla de IPv6, de su cabecera y sus extensiones, de las implicaciones que tiene en antiguos protocolos de nivel superior, y se explica a fondo el esquema de direccionamiento implantado. El segundo trata de IP multicast, empezando por su direccionamiento, la gestión de grupos y demás conceptos de enrutado, para seguir con los diferentes algoritmos y protocolos que hay actualmente, incidiendo especialmente en aquellos que puedan servir para entender el sistema multicast de GeoPaging. Será en el capítulo 3 donde se explica a fondo este protocolo: 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.

Los capítulos siguientes se centran en la parte práctica del trabajo. En el cuarto se explica cómo se ha llevado a término dicho protocolo mediante la programación en C sobre una maqueta de pruebas formada por PCs con sistema operativo Linux. Primero se comenta el algoritmo seguido para enrutar los paquetes de paging y después se referencia a las diferentes funciones y variables implicadas para poder entender el código fuente incluido en el CD adjunto. En el capítulo 5 se detallan las pruebas realizadas sobre la la maqueta con el fin de observar el comportamiento del programa y valorar si es válido o no.Tras mostrar los pasos realizados y las medidas tomadas, se podrá comprobar que el software desarrollado sigue correctamente las directrices del protocolo GeoPaging.

Finalmente en las conclusiones se resumen los aspectos más destacados del TFC y se realiza una valoración crítica sobre el cumplimiento de los objetivos planteados al inicio del trabajo. También se incluyen algunas líneas futuras de trabajo para próximos desarrollos. Pensando especialmente en estos se adjuntan cinco anexos. El primero describe el estado del software para Linux relacionado con IPv6 y multicast. El segundo contiene los detalles de la maqueta, así como los pasos que hay que seguir para configurarla. El tercero es un resumen de las tecnologías que se han investigado para programar en Linux. De la misma forma se documenta en el cuarto anexo los medios de obtener información del kernel de Linux. Finalmente, en el quinto se detallan algunas notas adicionales para la mejora del programa. Además se adjunta un CD con el código fuente y las herramientas de configuración del escenario.

CAPÍTULO 1
Protocolo de Internet v6 (IPv6)

1.1 Introducción

El protocolo de Internet de nueva generación, o IPv6, nace como una evolución del protocolo estándar para las comunicaciones de la Internet actual, o IPv4. El origen de IPv4 se remonta a finales de los años 70, y no es hasta los 90 cuando se puede hablar de Internet a nivel mundial. Es entonces cuando se observan una serie de problemas que impedirán el propio crecimiento de Internet si el protocolo utilizado no se adapta a las nuevas necesidades.

El proceso de estandarización de dichos protocolos sigue el procedimiento habitual de publicación de drafts del IETF (*Internet Engineering Task Force*), que posteriormente se aceptan como RFC (*Request For Comments*). El primero que habla del Protocolo de Internet (IP), evolución de los estándares de DARPA, es el RFC 760. Fue revisado en 1981 como RFC 791, definiendo el estándar IPv4, Posteriormente han habido muchos más RFC, añadiendo opciones o mejorando el protocolo con añadidos como ICMP, IGMP, enrutamiento, etc. IPv6 también ha ido evolucionando: el primer RFC relacionado es el 1550. Por entonces se hablaba de *IP Next Generation* (IPng), y apareció tras muchos intentos de extender las limitaciones de IPv4. Actualmente se toma como referencia el RFC 2460, que es el más actual (año 98).

Las características principales de este nuevo protocolo son:

- **Mayor capacidad de Direccionamiento**: Con IPv6 tenemos 128 bits para asignar direcciones, mientras que con IPv4 teníamos 32 bits. El espacio disponible se puede agotar (máximo de 4.294.967.296 direcciones), pues actualmente se estima que se ha ocupado dos terceras partes de éste y algunos países en expansión ven peligrar su infraestructura de Internet debido a dicha escasez. Con IPv6 tenemos hasta $3{,}4 \times 10^{38}$; se estiman más de 1500 direcciones por metro cuadrado de la Tierra. También se mejoran algunos aspectos técnicos del direccionamiento, como multicast mejorado y el nuevo tipo de direcciones anycast.

- **Simplificación de la cabecera**: algunas de las antiguas opciones se han eliminado o convertido en opcionales, reduciendo así el coste de procesado por paquete.

- **Soporte mejorado para extensiones y opciones**: el uso de un mecanismo flexible como es el campo Next Header (siguiente cabecera) hacen de IPv6 un protocolo flexible pero eficiente.

- **Capacidad de identificar flujos**: la presencia de un campo Flow Label permite tratamientos especiales de los paquetes, como mayor prioridad o incluso servicios en tiempo real.

- **Capacidad de Autenticación y Privacidad**: hay especificadas extensiones de cabecera para autenticar y asegurar integridad y confidencialidad de los datos.

1.2 Protocolo

1.2.1 Cabecera básica

La cabecera básica ocupa 40 bytes, repartidos de la siguiente forma:

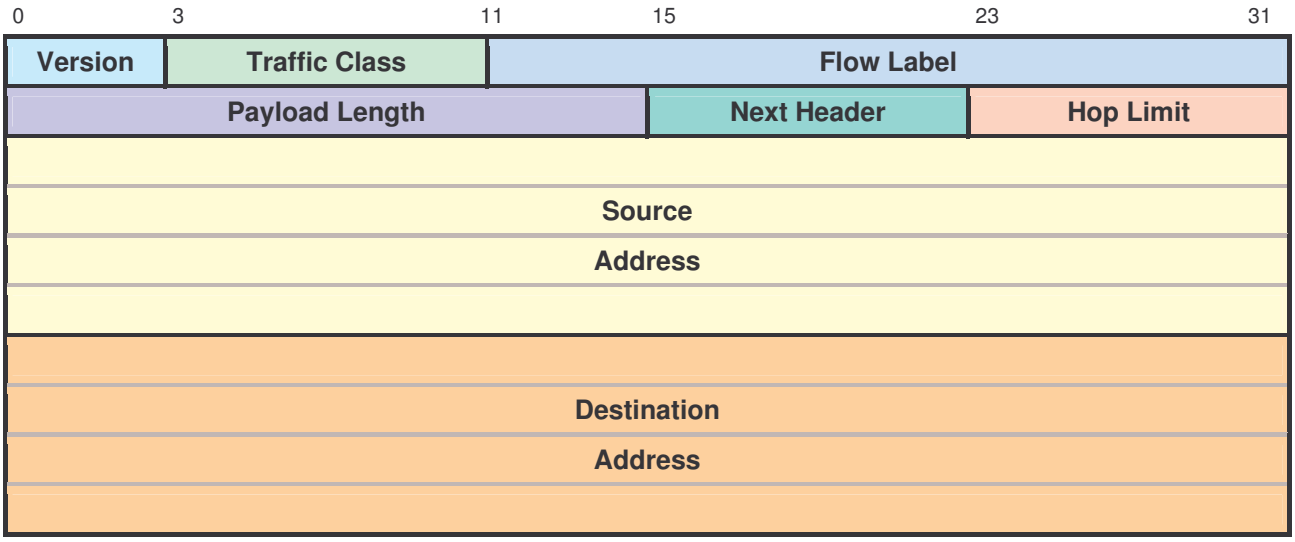

Figura 1.1 Cabecera IPv6

Los campos están organizados pensando en la eficiencia del algoritmo de enrutado, de manera que los primeros campos identifican la forma en la que se debe tratar el paquete (control de flujo, clase de tráfico), en definitiva, la QoS a aplicar.

El campo *Versión* son 4 bits, y es como el de IPv4 sólo que en este caso contendrá el número 6. *Traffic Class* son 8 bits que sirven junto con *Flow Label* para determinar el trato que debe recibir el paquete. *Payload Length* son 16 bits que indican la longitud del payload de IPv6, considerando payload todo aquello que no es cabecera básica (incluyendo las Extension Headers). El siguiente campo, *Next Header* (8 bits), contiene los mismos valores estandarizados para los payloads de IPv4 (p.ej. TCP=0x06, UDP=0x11, etc.), pero definiendo otros valores para dar cabida a las Extension Headers (ver siguiente capítulo). *Hop Limit* (8 bits) equivale al antiguo campo TTL. Por último, tanto *Source* como *Destination Address* tienen 128 bits y son las direcciones origen y destino del paquete.

1.2.2 Extensiones a la cabecera

Como hemos visto, hay un campo *Next Header* (NH) que sirve para indicar qué viene a continuación de la cabecera básica de IPv6 de 40 bytes. Veamos un ejemplo de como se aplican las Extension Headers.

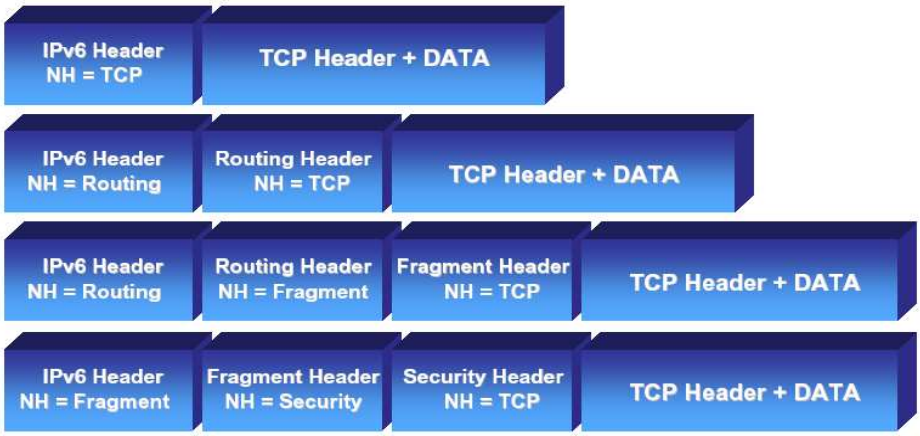

Figura 1.2 Ejemplos de extensiones a la cabecera IPv6

Como se puede observar, el funcionamiento es bien sencillo: dada una cabecera básica, podemos hacer que NH apunte directamente a un contenido de nivel superior (4, transporte TCP) o bien a una extensión de la cabecera de nivel 3 (red, IPv6) como por ejemplo información de enrutamiento, de fragmentación o de seguridad. Algunas de estas extensiones eran opciones que IPv4 tenía integradas en la cabecera, como las de fragmentación.

La lista de Extension Headers que de momento está definida es la mostrada en la Tabla 1.1. Hay que tener en cuenta que se aconseja el siguiente orden de cabeceras, y sólo se permite repetir la de Destination Options Header.

Tabla 1.1 Extension Headers

<IPv6 Header (básica)>
Hop-by-Hop Options
Destination Options *
Routing
Fragment
Authentication
Encapsulation Security Payload
Destination Options **
<Header de protocolo superior>

* a procesar por todos los elementos presentes en la extensión *Routing*
** a procesar sólo por el destinatario final

1.2.3 Formato de las Opciones en las cabeceras

Dos de las cabeceras definidas, la Hop-by-hop y Destination contienen opciones del estilo TLV (*Type-Length-Value*). Primero hay un campo Option type, que ocupa 8 bits y

identifica el tipo de opción. Después hay un campo Option Data Length de 8 bits, que determina el tamaño en bits del siguiente campo, Option Data. Hay una peculiaridad, y es que el primer campo de 8 bits (Type) ha de cumplir la siguiente norma para aquellos nodos que desconozcan el código que contiene el campo Type:

- Si los dos primeros bits son 00, el nodo deberá ignorar esa opción y seguir procesando
- Si son 01, se debe descartar el paquete.
- Si es 10, se descarta el paquete pero se avisa con ICMP Parameter Problem, code 2 al equipo que originó el paquete y especificando la Option Type que se desconoce.
- Si es 11, se actua como en el caso anterior sólo en el caso que la dirección destino NO sea multicast.

Hay otra norma, y es que el tercer bit indica si los datos del campo Option Data pueden cambiar mientras el paquete viaja por la red (valor 1), o si no pueden cambiar (valor 0).

De esta manera, cuando se defina uno de los 255 posibles tipos de opción, se deberá tener en cuenta el número a escoger en función de los criterios anteriores. En el apéndice B del RFC 2460 se especifican cómo construir nuevas opciones, y también como se formatean los *paddings* (rellenos con bits igual a 0) para alinear correctamente las opciones con el resto de cabecera. De momento sólo están definidas estas dos opciones, ambas relacionadas con el padding.

1.2.4 Hop-by-Hop Header

Esta cabecera cuyo código es el 0 se procesará en cada nodo intermedio en una comunicación IP. Tiene la siguiente forma: 8 bits de *Next Header*, 8 bits de *Header Extended Length* que indican la longitud total del header (descontando los primeros 8 bits del NH) y a continuación N opciones TLV. Como las opciones TLV son de longitud variable, se necesita la existencia de opciones especiales para el padding, que se definen en el RFC 2460

1.2.5 Routing Header

Un host puede generar un paquete con esta extensión para listar aquellos nodos intermedios por los cuales desea que pase el paquete. Su código es el 43, y tiene la siguiente estructura: 4 bytes consecutivos (*Next Header, Header Extension Length, Routing Type* y *Segments Left*), seguidos de un campo de tamaño variable con datos específicos del tipo de enrutado.

El comportamiento que tendrá un nodo ante una extensión de este tipo que contenga un campo Routing Type desconocido dependerá del valor de Segments Left. Si es 0, debe ignorar esta cabecera y seguir procesando las siguientes. Si no es 0, deberá descartar el paquete e informar al origen vía *ICMP Parameter Problem*, código 0.

El Tipo 0 de Routing Type, el único especificado en el RFC 2460, consta de una lista de direcciones, de manera que cuando llega el paquete a un salto (router), éste procesa la cabecera y pone la siguiente dirección de la lista en el campo Destination Header del paquete y lo reenvía, cambiando el puntero que representa el siguiente salto (Segments

Left). Con esto conseguimos ir de host en host por el camino deseado. La única limitación es que no se permiten direcciones multicast.

Si un equipo quiere usar esta cabecera, enviará el paquete a uno de los routers por donde quiere que viaje la información, el cual lo reenviará al siguiente router, y éste al siguiente, y finalmente le llegará al destino final. Éste verá un paquete normal, con el la dirección origen del equipo emisor, pero con una lista donde se especifican los routers por donde el emisor ha querido que viajase el paquete.

En resumen, y como ejemplo, la Routing Header tipo 0 permite al nodo A comunicarse con B pasando por una ruta conocida, que A quiere que sea X,Y y Z por ser una ruta rápida o fiable, por ejemplo. Así pues, los paquetes viajarían de A a X, X procesaría la cabecera y generaría un paquete hacia Y con dirección origen de A, y éste generaría otro paquete hacia Z, y posteriormente habría otro hacia B. Así pues, B vería que le llega un paquete con dirección origen A y que en la cabecera de enrutado saldría en orden la lista de las IPs de X,Y y Z.

1.2.6 Fragment Header

IPv6 requiere una MTU de mínimo 1280 bytes, y en el caso en que el enlace no pueda contener un paquete IP tan grande, deberá ser la capa de enlace la que fragmente. En el resto de casos, cuando queramos enviar un paquete de un tamaño mayor que la MTU del medio, será necesaria la fragmentación. Por eso se define esta cabecera de código 44. En IPv6 los campos de fragmentación se han sacado de la cabecera básica debido a que normalmente no se fragmentará pues la implementación de IPv6 debería realizar un *MTU Discovery* para averiguar la MTU a utilizar antes de enviar un paquete. Además a diferencia de IPv4 los routers intermedios no deberán fragmentar, sino enviar mensajes ICMP informando al emisor.

1.2.7 Destination Options Header

Tiene el código 60 y sirve para transportar información opcional que se debe examinar sólo en el nodo destino (a diferencia de la cabecera Hop-by-Hop). Como ya se ha comentado en el apartado 1.2.3, se codifica mediante opciones tipo TLV, y en el RFC 2460 sólo se especifican dos de ellas (Pad1 y PadN), requeridas para el padding.

Por último, hay una cabecera, **No next header** (código 59) que sirve para indicar que no hay ninguna cabecera a continuación de la actual, con lo que se considerarán payload los bytes que aparezcan a continuación de la cabecera.

1.3 Soporte para QoS

1.3.1 Flow Label

Con este campo de la cabecera, de 20 bits, un nodo puede enviar una serie de paquetes etiquetados para que reciba un tratamiento especial, aspirando incluso a un servicio real-time. Por lo tanto se requiere que un determinado flujo implique que todos los paquetes tengan la misma IP destino y origen y el mismo ID de flujo. El establecimiento del flujo y del tratamiento deseado se hará mediante un protocolo de control entre routers (p.ej. RSVP) o bien usando las extensiones Hop-by-Hop de IPv6. De momento ésta capacidad

se considera experimental pues hoy en día todavía se discute qué cantidad de tipos de servicio se deben ofrecer en Internet. Mientras tanto, los equipos que generen tráfico IPv6 deberán inicializar este campo a 0, e ignorarlo cuando reciban un paquete.

1.3.2 Traffic Class

Este campo de 8 bits está disponible para que los nodos que originan paquetes puedan marcar algunos de ellos. Permite que haya distintas clases o prioridades para tratar los paquetes en los routers intermedios, sin la necesidad y complejidad de establecer un flujo, permitiendo por ejemplo el envío puntual de un paquete IP con mayor fiabilidad y menor retardo. Igual que en el caso anterior, aún no se han determinado exactamente los diferentes tipos de servicio. Eso sí, se intenta que sea compatible con el campo Type of Service y Precedence de IPv4.

1.4 Implicaciones de IPv6 sobre otros protocolos

1.4.1 Checksums

Como ya se ha comentado, IPv6 no tiene CRC pues confía en el resto de capas (superiores e inferiores) para que se aseguren de tener integridad. Pero esas capas superiores a veces usan campos de la cabecera IP, como es el caso de TCP y UDP que usan una pseudocabecera para realizar el chequeo. Es necesario cambiar la implementación de dichos protocolos para que admitan los cambios de la cabecera IPv6 respecto de IPv4, como son las direcciones de 128 bits (en vez de 32).

Otro ejemplo de los problemas que existen es por ejemplo el caso de comunicaciones extremo a extremo usando cabeceras Routing de IPv6. Como el CRC de las capas superiores se calcula usando la dirección destino, al llegar al receptor la cabecera IP ha cambiado, debido al funcionamiento de la Routing Header, con lo que fallará. Para solucionarlo, el origen debería coger la última direccion que aparece en la Routing Header para hacer el cálculo.

Protocolos como por ejemplo ICMP que antes no tenían CRC deberán ahora implementarlo (ya introducido en ICMPv6) pues IPv6 ya no realiza ese chequeo y podríamos tener mensajes ICMP con errores.

1.4.2 Tiempo de vida

Este campo se decrementa en 1 cada vez que el paquete es enrutado de una interfaz a otra, como un contador, y que cuando llega a 0 el router que lo recibe, descarta el paquete. En IPv4 se obligaba a inicializar a un valor considerado normal el campo TTL (en IPv6 ahora se llama Hop Limit) para evitar paquetes itinerantes por la red de forma prolongada. En cambio IPv6 no obliga a ello. Pero en la realidad, pocas implementaciones de IPv4 controlan el valor del campo TTL y dejan uno por defecto (128, 255, etc). Por eso en IPv6 no se obliga.

1.4.3 Máximo tamaño de Payload

En las capas superiores, como TCP, se calcula el payload máximo que pueden alojar usando tamaños fijos de cabecera IP. En IPv4 se usaba 20 bytes pues es el tamaño mínimo que tendrá la cabecera. Este valor será de 40 bytes en el caso de paquetes IPv6 sin Extension Headers.

1.4.4 Respuesta a Paquetes con Routing Header

Imaginemos que nos llega un paquete TCP de una conexión Telnet ya establecida pero con cabecera Routing Header. Como sólo nos fiamos del extremo que está conectado por telnet, no podemos confiar en que la lista de IPs que han enrutado el paquete sea fiable, o verídica, pues podría ser que un atacante quisiera espiarnos (ataque de “hombre-en-medio”). Así pues se desaconseja responder a paquetes con cabecera de enrutado usando la misma lista de “routers”, es decir, no se podrá responder con la misma cabecera de enrutado. Siempre se podrá hacer de forma normal, sin cabecera de enrutado, o con una diferente al anterior, creado en este caso por el servidor de telnet.

Sólo se podrá responder con la misma cabecera de enrutado (se entiende que se ha de invertir el orden para seguir la misma ruta con la que llegó el paquete) si, y sólo si, se ha verificado la integridad y autenticidad de las direcciones IP tanto del origen como de los enrutadores.

1.5 Arquitectura del direccionamiento IPv6

Como ya se ha comentado, las direcciones de IPv6 tienen 128 bits. La forma más habitual de representarlas es mediante 8 bloques de 16 bits representados por 4 caracteres hexadecimales, separados por “:”.

Un ejemplo de dirección es 41BC:0:0:0:5:DDE1:8006:2334. Como se puede observar, en cada bloque de 4 números hexadecimales se pueden obviar los ceros a la izquierda y no ponerlos, como por ejemplo 5 (en vez de 0005). También se podría poner de la siguiente forma 41BC::5:DDE1:8006:2334 donde “::” quiere decir que los campos que faltan son todo 0 (0:0:0), siempre y cuando no resulte ambiguo.

En IPv6 no existe el concepto clásico de “clases” de direcciones (clase A,B,C,D,E en Ipv4). Sin embargo, se reservan unos conjuntos de direcciones para usos especiales. La asignación cambia con el tiempo a medida que el protocolo evoluciona. El primer RFC fue el 1884, luego el 2373 lo deprecó, y el válido actualmente es el 3513, al que ya se le prepara su sustituto, el draft-ietf-ipv6-addr-arch-v4-00.

De momento ya se han reservado varias porciones del espacio de direcciones, mediante la reserva de los primeros bits de la dirección (prefijo), para los siguientes usos:

Tabla 1.2 Asignación actual de direcciones IPv6

Allocation	Prefix (binary)	Fraction of Address Space
Unassigned	0000 0000	1/256
Unassigned	0000 0001	1/256
Reserved for NSAP Allocation	0000 001	1/128
Unassigned	0000 01	1/64

Unassigned	0000 1	1/32
Unassigned	0001	1/16
Global Unicast	001	1/8
Unassigned	010	1/8
Unassigned	011	1/8
Unassigned	100	1/8
Unassigned	101	1/8
Unassigned	110	1/8
Unassigned	1110	1/16
Unassigned	1111 0	1/32
Unassigned	1111 10	1/64
Unassigned	1111 110	1/128
Unassigned	1111 1110 0	1/512
Link-Local Unicast Addresses	1111 1110 10	1/1024
Site-Local Unicast Addresses	1111 1110 11	1/1024
Multicast Addresses	1111 1111	1/256

Tenemos tres tipos de direcciones:

Unicast: asignada a una sola interfaz IPv6.

Anycast: identifica a un conjunto de interfaces de manera que un paquete destinado a una dirección de este tipo se entregará a la interfaz más cercana (a la que primero llegue, con menor número de saltos). El administrador será quien configure los equipos para que compartan una dirección anycast.

Multicast: identifica a un conjunto de interfaces, definidas por una aplicación (router, software cliente multicast, etc) de manera que un paquete con destino multicast se entrega a todos los equipos identificados por dicha dirección.

Hay que destacar la ausencia de direcciones broadcast. Se considera un caso especial de multicast, pero que implica mayor coste pues todos los nodos deben procesar el paquete broadcast aun a pesar de que probablemente no sea necesario. Por ejemplo una petición DHCP era escuchada por todos los nodos, sean o no servidores, usando broadcast; pero si usamos una dirección multicast especial para DHCP (o para todos los servidores de una red local) evitamos procesado inútil en todas las interfaces de la red.

1.5.1 Segmentación del espacio de direcciones

Como en IPv4, las topologías de red se organizan según un criterio de asignación de direcciones jerárquicas, basadas en máscaras de subredes. Se indican mediante un prefijo que indica la longitud de la máscara, al igual que en CIDR[1] de IPv4. Así pues, una red 12AB:0:0:CD30::/60 permite alojar en su interior el conjunto de direcciones desde 12AB:0:0:CD30:: (todo 0) hasta la 12AB:0:0:CD3F:FFFF:FFFF:FFFF:FFFF. A mayor índice de subred (p.ej /120), menor cantidad de hosts en la subred cabrán (para /120, un total de 2^8, es decir 256).

Así pues, una dirección IPv6 tiene la siguiente forma:

Tabla 1.3 Identificadores en dirección IPv6

n bits	*128 - n bits*
ID de Subnet	**ID de Interfaz**

[1] Más información en http://public.pacbell.net/dedicated/cidr.html

La **ID de Interfaz** se requiere que sea de 64 bits para direcciones unicast, y que se construya con el formato EUI-64. Éste formato garantiza direcciones únicas. Por ejemplo se pueden construir tomando como base las direcciones MAC de 802.3 (también universales, pero de 48 bits) y modificándolas según lo especificado para que ocupen 64 bits, rellenando con 0 y con dos bits especiales.

Hay una serie de direcciones especiales, entre ellas la de loopback o 0:0:0:0:0:0:0:1, y la unspecified o 0:0:0:0:0:0:0:0.

También se reservan rangos para tener compatibilidad con IPv4 (siempre y cuando la dirección IPv4 sea global, no privada), que se explican en el siguiente cuadro:

Tabla 1.4 Direcciones IPv6 para compatibilidad con IPv4

Direcciones IPv6 que son compatibles con IPv4, pues los routers intermedios harán un tunel IPv6 sobre IPv4 de forma automática

80 bits	16	32 bits
0000.....0000	0000	IPv4 address

Direcciones IPv4 mapeadas dentro de una IPv6, sirven para representar nodos IPv4 en redes IPv6.

80 bits	16	32 bits
0000.....0000	FFFF	IPv4 address

NOTA: se permite mezclar notación hexadecimal con decimal para estos casos, es decir, 0:0:0:0:0:FFFF:192.168.1.2.

Finalmente, las direcciones globales de IPv6 tendrán la siguiente forma:

Tabla 1.5 Direcciones IPv6 globales

n bits	m bits	128-n-m bits
global routing prefix	subnet ID	interface ID

El prefijo global de enrutado se asigna según unos criterios de manera que permite un enrutado eficiente. La ID de interfaz debe ser de 64 bits (excepto cuando el prefijo global empieza por 000, entonces no hay limitación). Esta propuesta hace que el direccionamiento sea mejor: en vez de asignar subredes de 64 bits, se segmenta ese espacio en, por ejemplo, subredes de proveedores de servicios, subredes militares, subredes académicas, etc, mediante la asignación de prefijos de enrutado globales diferentes a cada tipo.

1.5.2 Direcciones privadas

Las direcciones locales se dividen en de ámbito relativo al enlace (link-local) o de ámbito relativo a la organización (site-local).

Tabla 1.6 Direccionamiento privado

	10 bits	54 bits	64 bits
Link-Local	1111111010	0	interface ID
Site-Local	1111111011	Subnet ID	interface ID

Las direcciones link-local sirven para asignar direcciones en un enlace para su autoconfiguración, normalmente en la fase de descubrimiento de router, o bien para enlaces sin salida al exterior.

Las de site-local son aptas para un direccionamiento privado automático en una red corporativa con varias subredes. Permite tener la red bien segmentada, con IDs de subredes, y facilita la migración a direcciones globales simplemente cambiando los primeros 10 bits.

En cuanto a las direcciones anycast, sólo hay una de obligatoria, y es la de “cualquier router de la subred”. Dicha dirección consta de la ID de subred seguido de todo 0. Todos los routers deben tenerla configurada. Esto permite que un nodo se pueda comunicar con cualquiera de los routers de la subred.

1.5.3 Direcciones Multicast

Las direcciones multicast tienen este estilo:

Tabla 1.7 Direcciones IPv6 multicast

8	4	4	112 bits
11111111	flgs	scop	group ID

Se usa un campo flags, que de momento sólo puede tener dos valores, para indicar si es una dirección multicast conocida y asignada por la IANA (valor 0000) o bien si es una dirección permanente (valor 0001). También hay un campo scope que sirve para limitar el alcance del tráfico multicast, y que puede tener los siguientes valores:

0 reserved	**4** admin-local scope	**8** organization-local scope	**C** (unassigned)
1 interface-local scope	**5** site-local scope	**9** (unassigned)	**D** (unassigned)
2 link-local scope	**6** (unassigned)	**A** (unassigned)	**E** global scope
3 reserved	**7** (unassigned)	**B** (unassigned)	**F** reserved

Figura 1.3 Campo *scope* en direcciones multicast

Cuando se asigna un ID de grupo para una aplicación, entonces se usa el scope para identificar un conjunto más o menos amplio de miembros, ya sea relativos al enlace, a la organización, global, etc. Por ejemplo, para el caso de servidores NTP (cuyo ID de grupo es 101) tenemos los siguientes casos:

- FF01:0:0:0:0:0:0:101 son todos los servidores NTP de la misma interfaz (mismo nodo) que el cliente.
- FF02:0:0:0:0:0:0:101 son todos los servidores NTP del mismo enlace local.
- FF05:0:0:0:0:0:0:101 son todos los servidores NTP del mismo sitio (corporación).
- FF0E:0:0:0:0:0:0:101 son todos los servidores NTP de Internet.

Además, hay una serie de direcciones multicast ya asignadas, como la de “todos los nodos del enlace” (FF01:0:0:0:0:0:0:1 y FF02:0:0:0:0:0:0:1), o la de “todos los routers” (FF01:0:0:0:0:0:0:2, FF02:0:0:0:0:0:0:2 y FF05:0:0:0:0:0:0:2). Los rangos o scope en estos casos son 1 (interface-local), 2 (link-local), y 5 (site-local)

1.6 Direcciones asociadas a un nodo

Así pues, un nodo debe reconocer una serie de IPs, y responder a ellas:

- La dirección Link-Local para cada una de sus interfaces.
- Cualquier dirección anycast o unicast que se haya configurado.
- La dirección de loopback.
- La dirección de multicast referente a “todos los nodos”.
- La dirección especial solicited node, descrita en la sección 2.7.1 del RFC 2460.
- Cualquier dirección multicast que se haya configurado.

Un router deberá reconocer, además de las anteriores, las siguientes direcciones:

- La dirección anycast referente a “cualquier router de la subred”.
- La dirección multicast referente a “todos los routers”.

CAPÍTULO 2
Enrutado Multicast

2.1 Concepto

El concepto de Multicast significa entrega a algunos (punto-multipunto). En una red de datos, esto implica comunicación entre un origen y varios receptores. A diferencia de Broadcast, que significa entrega a todos, multicast aprovecha al máximo los recursos de la red y no implica que todos los nodos tengan que procesar datos que no son suyos.

Cuando aplicamos este concepto a IP y pasamos a hablar de Multicast en IP, podemos considerar los siguientes términos y problemáticas:

- **Grupo**: conjunto de receptores de un flujo de la misma información. Se puede ser miembro de ninguno, de uno o de varios grupos a la vez.

- **Direcciones de grupo**: una dirección multicast demarca unívocamente a un grupo. Hay direcciones permanentes, asignadas por la IANA, y direcciones temporales, asignables para su uso en cualquier aplicación arbitraria.

- **Cantidad de grupos**: el límite no está en el espacio de direcciones, pues tanto IPv4 como IPv6 reservan una buena cantidad para este uso, sino que el límite es el tamaño de las tablas de enrutado, que como se explicará en apartado 2.4 pueden llegar a tener un gran tamaño.

- **Los miembros de un grupo cambian dinámicamente**: no solo la variación es rápida, sino que un nodo puede ser miembro de un número arbitrario de grupos.

- **Uso de hardware multicast**: si IP sabe que las capas inferiores lo soportan, lo utilizan (por ejemplo, Ethernet tiene direcciones especiales para multicast); en caso contrario se usa broadcast o envío múltiple unicast, según el caso.

- **Envío entre diferentes redes**: se necesita la existencia de routers multicast para poder reenviar los paquetes entre miembros de un grupo pertenecientes a diferentes redes.

- **Transmisión entre diferentes miembros**: cualquier nodo, miembro o no de un grupo, puede enviar paquetes a un grupo arbitrario. El hecho de ser miembro solo condiciona la recepción.

En resumen, podemos decir que todo sistema multicast necesita cuatro elementos:

1. Un esquema de direccionamiento para identificar los grupos multicast
2. Un sistema eficiente de notificación y gestión de grupos de receptores (comunicación host-router multicast)
3. Un mecanismo eficiente para el reenvío entre diferentes redes (comunicación entre routers)
4. Una aplicación cliente/servidor capaz de enviar/recibir información vía multicast.

A parte de los elementos necesarios que debe tener un sistema multicast, el protocolo utilizado debe tener en cuenta los siguientes factores:

Los grupos son dinámicos, así que un cambio en ellos debe ser perfectamente manejado por el protocolo (ver apartado 2.3) y no afectar al resto de usuarios. Este dinamismo puede afectar a las aplicaciones sensibles al retardo y a las pérdidas de paquetes (debido a que IP es best-effort), como vídeo, audio, noticias en tiempo real, etc. También hay que tener en cuenta la inseguridad de las comunicaciones IP, que se ve agraviada en redes multicast, donde el intercambio de claves de cifrado es más complicado.

La escalabilidad del sistema es otro de los factores más críticos. Por su naturaleza, un protocolo multicast tiende a sufrir problemas de escalabilidad debido a que un número arbitrario de clientes puede pertenecer a un número arbitrario de grupos. Esto dificulta mecanismos como el control de flujo y de errores, como los ACKs de TCP, que si se implementan sobre redes multidifusión provocarían una inundación de mensajes de control en dirección a la fuente. Además hay otro elemento a considerar, y es que el uso de multicast en la red global es inviable sin la segmentación por sistemas autónomos (ver apartado 2.6.3)

2.2 Direccionamiento

Un grupo multicast se identifica mediante una dirección multicast. En una red de difusión, como ethernet, todos los nodos conectados a un mismo concentrador reciben el mismo tráfico, pero sólo interpretan aquél cuya dirección destino coincide con la configurada en el equipo. Cuando se envía un paquete multicast en una red de difusión, todos los equipos reciben el paquete, pero sólo aquellos que sean miembros lo interpretarán. Es decir, no importa cuántos receptores tenga un grupo multicast en una red local, sólo se envía un paquete.

Las direcciones multicast en IPv4 son aquellas comprendidas en el rango 224.0.0.0 hasta 239.255.255.255 (espacio de direcciones de clase D), pero en IPv6 las direcciones son todas aquellas direcciones del segmento FF00::/8, es decir, todas aquellas cuyos 8 primeros bits son 11111111. De todo ese espacio de direcciones multicast se reservan subconjuntos para fines específicos, tal y como se especifica en el apartado 1.5.3.

En redes como Ethernet, IP utiliza el mecanismo de multicast de nivel 2 mediante el cual consigue minimizar más aún el gasto de CPU de los nodos de la red, pues entonces es la propia tarjeta de red la que descarta el tráfico que no corresponde al nodo. El mapeo de la dirección IPv4 multicast a la dirección MAC a utilizar se realiza mediante el espacio de direcciones MAC reservado para ello: desde 01-00-5E-00-00-00 hasta 01-00-5E-FF-FF-FF. Los últimos 23 bits de estas direcciones se sacan de los últimos 23 bits de la dirección IPv4 multicast[2]. Para IPv6 se utiliza otro prefijo reservado (prefijo IPv6 Neighbour Discovery), de manera que quedan direcciones MAC 33-33-XX-XX-XX-XX donde los últimos 32 bits corresponden con los últimos 32 bits de la dirección IPv6 multicast (ver sección 7 del RFC 2464).

En redes que a diferencia de Ethernet no dispongan de direcciones de nivel 2 que soporten mapeo de IP multicast, entonces se escogerá envío broadcast de nivel 2 o bien multienvío unicast, dejando la elección al criterio del emisor del paquete (normalmente el router de la red)

[2] Más información en http://www.enterasys.com/products/whitepapers/deploy-ip/

Cualquier equipo puede enviar paquetes a una dirección multicast, pero si la red no tiene constancia de que ese equipo envía tráfico multicast, no llegará a ningún destino fuera de su red de área local. Igualmente, cualquier equipo se puede unir a un grupo multicast, pero si la red (el router) no lo sabe, al equipo no le llegará tráfico. Para ello es necesario utilizar una serie de mecanismos que se explican en el siguiente apartado.

2.3 Gestión de grupos

Hay dos maneras de plantear el problema de quién controla la unión de clientes a grupos multicast. Se puede controlar desde la fuente, de una forma centralizada, o bien desde los receptores, de forma distribuida. En IP se usa habitualmente esta estrategia, donde la fuente ignora quién recibe sus paquetes.

Si queremos usar **multicast basado en la fuente,** es necesario que cada cliente comunique a la fuente su intención de recibir el tráfico. De esta manera, es la fuente quien duplica los paquetes y los envía de forma unicast a todos sus clientes. Esto es muy poco escalable, pero funciona bien en sistemas pequeños, con gran variedad de grupos, y en sistemas IP tradicionales (usamos envío unicast). Protocolos basados en este punto de vista son ST-II (*Stream Protocol v.II*), XTP (*Xpress Transport Protocol*) y MTP (*Multicast Transport Protocol*). Recientemente ha aparecido un protocolo llamado XCAST (*eXplicit Multicast*), que funciona sobre IPv6, que también sigue este modelo.

En cambio, si hablamos de **multicast basado en el receptor,** aparecen los grupos de receptores, que en IP se identifican mediante direcciones multicast. Así pues, tenemos una fuente que genera tráfico con IP destino la dirección del grupo (multicast), tenemos también una serie de routers interconectados entre sí, y tenemos subredes (conectadas a esos routers) con posibles clientes. En cuanto un cliente desea recibir el flujo de información destinada al grupo, envía una petición a su router local, el cual hace lo mismo con los otros routers de la red de manera que empieza a recibir el tráfico multicast. El conjunto de routers que intercambian y duplican el tráfico multicast se llama árbol de routers multicast. En cuanto ese router que ya no hay nadie en su subred interesado en recibir esa información, envía una petición al resto de routers para que dejen de enviarle esa información. Este mecanismo se llama *prunning* ya que es como podar un extremo del árbol multicast.

En todo ese proceso, la fuente no sabe si su flujo es recibido o no por algún cliente. Simplemente envía su información sin duplicar con destino a una IP multicast. Si durante el trayecto los routers intermedios duplican los paquetes, o bien no propagan la información porque no hay clientes, la fuente no tiene constancia de ello. Este proceso de gestión de grupos lo controla el protocolo IGMP (RFC 2236) en IPv4, y MLD (*Multicast Listener Discovery*, integrado en ICMPv6, RFC 2710 y 2463) en IPv6.

2.4 Enrutado de Multicast

2.4.1 Conceptos de enrutado

El mecanismo básico de enrutado en IP consiste en decidir qué ruta debe tomar un paquete en su viaje. Un router sólo tiene información de sus redes locales, pero la decisión debe ser global y por eso es necesario el intercambio de información entre ellos. La información no es por host, sino por subred, pues se entiende que tenemos siempre al menos un router que nos lleva hacia esa subred.

Todo protocolo de enrutado, sea unicast o multicast, debe intentar seguir unas normas de diseño:

- Minimizar las tablas de enrutado, pues eso implica aprovechar los recursos (RAM) del router. Si dichos recursos se agotan porque las tablas son demasiado grandes, el protocolo entonces fallará porque no podrá recoger más información de la red.
- Minimizar los mensajes de control entre routers, ahorrando recursos de red.
- El protocolo debe ser robusto, evitando pérdidas de paquetes o bucles.
- El enrutado debe elegir siempre la ruta más óptima, sea cual sea el criterio (retardo, velocidad...)

A modo de resumen, se podría simplificar el enrutado en redes de paquetes como IP en dos categorías: basados en **vector-distancia** o en **estado-enlace**. En ambos se supone que cada router conoce qué redes tiene conectadas, qué routers hay en ellas, y qué coste hay para llegar a esos routers. El coste se puede estimar en velocidad del enlace, retardo, porcentaje de pérdidas, etc. Así pues, en un algoritmo de vector-distancia, cada router comunica a sus vecinos las distancias que tiene registradas para llegar a todos las redes, es decir, les envía su tabla de enrutado completa. En cambio en un algoritmo estado-enlace los routers se comunican entre sí solamente las distancias respecto a las redes vecinas.

Los basados en vector-distancia funcionan muy bien en redes pequeñas o bien poco variantes, pues tienen mucho tiempo de convergencia ante cualquier cambio. Además cuando la red crece, el intercambio de mensajes también (mayores tablas de enrutado a intercambiar).

En cambio los basados en estado-enlace siguen la filosofía de comunicar la topología de la mediante el intercambio de mensajes con los costes que tiene cada router para llegar a su vecino, y así cada router hará los cálculos de mejores rutas de forma independiente. En otras palabras, cada router informa de una parte de la red al resto, y así todos juntan las partes y calculan las mejores rutas de independientemente de los cálculos de los demás. Si el sistema de cálculo del coste del enlace es común, no solo se garantiza la elección de la mejor ruta, sino que también se evitan bucles, lo cual es más complicado en vector-distancia.

Los mecanismos de enrutado unicast resuelven eficientemente el problema de encaminar un paquete a través de la red, pero no saben cómo hacer que un paquete multicast llegue a todos los receptores.. Por lo tanto, se discuten varios protocolos (apartado 2.6) basados en diferentes algoritmos (apartado 2.5), con el objetivo de poder decidir cuándo es conveniente duplicar un paquete por una interfaz. Para poder funcionar correctamente estos protocolos necesitan tablas de enrutado unicast, además de un protocolo de gestión de grupos para saber hasta dónde debe llegar el árbol multicast.

Los routers multicast también tendrán que intercambiar información entre ellos para aprender la topología de la red y decidir en función de ella, pero esta topología es mucho más problemática que la unicast, debido al dinamismo con que los hosts se unen o abandonan un grupo multicast. Además, cada grupo multicast tendrá una topología de distribución de los clientes diferente, con lo que el cálculo que deberán realizar los routers y la memoria necesaria para almacenar esa información es mayor que en unicast.

En definitiva, un protocolo de enrutado multicast deberá centrarse en factores como la construcción de los árboles (ver siguiente apartado 2.4.2) multicast y su mantenimiento, la

distribución de usuarios en cada grupo y las latencias asociadas a los cambios en los usuarios de esos grupos. Sólo cuando se solucionan adecuadamente esos factores se puede pensar en la escalabilidad del sistema global, aspecto que hoy en día todavía no se ha solucionado de forma estándar. Mecanismos como el *pruning* (explicado en apartado 2.3) ya se implementan y evitan que se envíe tráfico a routers sin clientes, lo cual sería inútil y desaprovecha ancho de banda; consiguiendo así que la red pueda crecer y soportar mayores cantidades de tráfico

2.4.2 Mecanismos para la construcción de los árboles

En el enrutado unicast se habla de tablas de encaminamiento, pero en multicast se utiliza teoría de grafos y se habla de árboles, raíces (la fuente), ramas (los routers) y hojas (los clientes). La mejor manera es calcular un árbol para el encaminamiento del tráfico, de manera que sea imposible pasar dos veces por el mismo sitio (sin bucles).

Hay dos tipos básicos de árboles multicast: árboles basados en la fuente (**source trees**) y árboles compartidos (**shared trees**).

Los algoritmos **basados en la fuente** crean árboles donde la raíz es el equipo emisor el tráfico multicast, de manera que para enrutar se sigue la ruta más corta a través del árbol. Por eso se les conoce como algoritmos **SPT** (shortest path tree). Se construye un árbol diferente para cada identificador (S,G), donde S es la fuente (IP unicast de la subred de la fuente[3]) y G es el grupo (IP multicast). La ventaja es que las rutas son óptimas, pero la gran desventaja es que se debe tener un árbol diferente para cada fuente-grupo.

Los **basados en árbol compartido** (llamados **SDT,** shared distribution tree) parten de la base de que se ha configurado la red para tener una estructura jerárquica, donde existe un router designado para que sea la raíz del árbol compartido para ese grupo multicast. De esta manera todos los routers multicast saben que todo el tráfico debe ser dirigido hacia la raíz, quien decidirá el camino a seguir por el tráfico. La ventaja es que se pueden definir diferentes grupos multicast para cada router (teniendo varios routers raíz), pero no se necesitan tablas de encaminamiento para cada fuente de un mismo grupo. Así pues, las tablas en SDT se identifican como (*.G), pues una tabla sirve para todas las fuentes.

Ambos sistemas evitan bucles, uno a costa de sacrificar memoria en los routers y el otro a costa de hacer algo más ineficiente el encaminamiento.

2.4.3 Dense mode y Sparse mode

Con el tiempo se ha visto que no se puede aplicar el mismo funcionamiento a entornos con alta densidad (dense) de usuarios que a entornos con baja densidad (spare). Los mecanismos que presuponen que casi todas las subredes tienen al menos un cliente de ese grupo multicast, se les considera protocolos de enrutado **dense mode**, mientras que los que tienen pocos usuarios se les considera **sparse mode**.

La principal diferencia está en que en sparse mode se actúa por un mecanismo de inundación, esperando a que los routers que no necesitan recibir tráfico multicast avisen que quieren eliminarse del árbol multicast. A este mecanismo se le llama *flood and prune*, o bien *data driven* pues primero se envían los datos y luego se espera a que sean

[3] Se usa la IP de la subred del emisor en vez de la IP de la fuente: es equivalente y minimiza la tabla de enrutado.

rechazados. En cambio en dense mode primero se espera a que los routers soliciten el tráfico (porque tienen usuarios), enviando peticiones en dirección a la fuente, y acto seguido empiezan a recibir información. Esto se considera un proceso iniciado por el receptor, o *receiver-initiated*.

Así pues, si usamos dense mode, gastaremos inicialmente muchos recursos de red, pero aseguramos una entrega de la información rápida y fiable; mientras que en sparse mode se hace más compleja la estructura de la red, pues normalmente hay que jerarquizar en forma de árbol distribuido (SDT), aumentando el tiempo para entrar en el grupo, pero aprovechando mejor los recursos de red. Como es lógico, el modo esparcido escala mucho mejor cuando hay muchos grupos de pocos usuarios, pero el modo denso funciona mejor cuando hay pocos grupos de muchos usuarios.

2.5 Algoritmos de enrutado multicast

Una vez entendidos los conceptos básicos de multicast (enrutado, árboles de distribución, sparse o dense mode) pasamos a explicar las diferentes alternativas que han surgido para solucionar la problemática de multicast. Primero se exponen los algoritmos más simples, acabando por los más complejos, que recogen las ideas expuestas por los algoritmos sencillos.

Podemos considerar cuatro categorías, según su complejidad y prestaciones: mecanismos simples, basados en SPT, basados en SDT o mecanismos para intercomunicar sistemas autónomos.

2.5.1 Mecanismos simples

Entre los mecanismos simples podemos considerar el simple **flooding**, o inundación, por el cual al llegar un paquete por una interfaz, se reenvía por todas las demás interfaces. La única diferencia con el **broadcasting** es que se controla si se repite un paquete, en cuyo caso no se reenvía (una manera simple de intentar evitar bucles). Tiene muchas limitaciones evidentes, como el mal uso de ancho de banda o la imposibilidad de llevar un registro preciso de todos los paquetes enviados. Otro mecanismo, no tan simple, es el llamado **spanning tree**, donde a partir de una red de routers multicast interconectados arbitrariamente se procede a “recortar” o deshabilitar algunas interfaces. Se pretende seguir un modelo de árbol, sin bucles, imponiendo la restricción de cada router debe tener como máximo dos interfaces de salida y una de entrada, tomando un punto de referencia como raíz. El cálculo del árbol más óptimo es complejo, además es muy complicado de desplegar en redes medianamente extensas. A partir de ese árbol, simplemente usando el mecanismo de inundación, se obtiene un sistema multicast de forma bastante sencilla.

2.5.2 Basados en SPT

Los algoritmos basados en fuente, y en árbol más corto (SPT) tienen en común que utilizan la tabla de encaminamiento unicast como medio para conocer la topología de la red, y según esa información construir un árbol para cada grupo multicast. Se crean árboles diferentes para cada pareja de direcciones IP multicast fuente – grupo (F,G).

Los algoritmos han ido evolucionando con el tiempo, y cada uno aprovecha lo mejor del anterior. Estudiaremos desde el más básico hasta el más complejo.

2.5.2.1 Reverse-Path Broadcasting (RPB)

El clásico sistema de propagación por *broadcast* se basa en recibir paquetes por una interfaz cualquiera y duplicarlos por el resto de interfaces. Puede haber bucles, llamados tormentas de broadcast. El único límite que se le puede imponer es el decremento del campo TTL (Hop Limit en IPv6) del paquete cada vez que es propagado por una interfaz.

RPB, también llamado *Reverse-Path Forwarding* (RPF) propone un chequeo antes de retransmitir los paquetes: sólo se transmitirá si la interfaz por la que ha llegado el paquete es el camino mas corto hacia la dirección IP del origen del paquete. Para ello consulta la tabla de encaminamiento unicast y comprueba cuál es la ruta más rápida hacia la fuente. Se llama interfaz padre (*parent*) a esa interfaz que lleva hacia la fuente del tráfico multicast. Al resto de interfaces se les llama hijos (*child*).

2.5.2.2 RPB mejorado

Se añade otro chequeo adicional antes de transmitir según RPB: el router mira si los routers accesibles por las interfaces *child* lo son también en el encaminamiento unicast. Es decir, si el router A hace de gateway para el router B en su camino hacia la fuente, entonces B recibirá tráfico multicast desde A. En cambio, si tanto A como B conectan con la fuente, A no propagará tráfico multicast por la interfaz que comunica A con B porque sabe que B, al aplicar RPB, descartará los paquetes porque no han llegado por su interfaz *parent*. Esto evita desperdiciar ancho de banda.

2.5.2.3 Truncated Reverse Path Broadcasting (TRPB)

Basándonos en RPB, se añade un chequeo que comprueba si una subred tiene receptores multicast activos. Se usa la información que proporciona IGMP sobre los miembros de una red que pertenecen a un grupo multicast. Por eso se dice que el router trunca el reenvío de paquetes multicast si en esa subred no hay receptores activos. También se habla de podar una rama al hecho de truncar el reenvío por una subred o rama del árbol de distribución.

2.5.2.4 Reverse Path Multicasting (RPM)

La idea de RPM mejora la del TRPB extendiendo la comprobación no sólo a receptores multicast, sino también a routers intermedios del árbol de distribución. RPM proporciona mecanismos para saber si en una subred hay routers intermedios en dirección hacia los receptores multicast, además de usar IGMP como lo hace TRPB.

RPM primero da por supuesto que una red tiene routers multicast, así que construye un árbol de distribución similar al que haría TRPB. Pero si un router trunca una interfaz porque comunica con una subred donde no hay receptores para ese grupo, no se limita a recibir y descartar paquetes como haría TRPB, sino que comunica a su router superior (el llamado *parent link*) que desea dejar de recibir información. Para ello utiliza un *Prune message* o “Mensaje de Poda”. Este mecanismo se llama *flood and prune* y hace que periódicamente los routers superiores en un árbol de distribución vuelvan a enviar tráfico multicast hacia los routers inferiores, hasta que éstos comuniquen otra vez su intención de dejar de recibir tráfico.

El árbol de distribución creado es más eficiente que el de TRPB pues todas sus ramas llevan a receptores activos.

2.5.3 Basados en SDT

La principal diferencia con los basados en SPT es que no se construye un árbol de distribución diferente para cada pareja fuente-grupo, sino que se construye un único árbol común para todos los miembros de un grupo. El tráfico de un grupo es distribuido siempre por el mismo árbol independientemente de la fuente. Sin embargo los miembros deben unirse explícitamente al árbol pues por defecto no se supone que una subred tiene miembros, tal como se hacen en algunos algoritmos SPT.

Mediante SPT tenemos tres sustanciales ventajas. Primero, minimizamos el gasto de recursos de los routers al no crear nuevos árboles para cada fuente. Segundo, minimizamos el tráfico pues no se inunda periódicamente la red en búsqueda de posibles receptores. Tercero, como los miembros deben explícitamente unirse al árbol (mensajes *join*) se consigue que el tráfico multicast sólo vaya por caminos que lleven a clientes.

Sin embargo, un protocolo basado en SDT debe evitar congestiones de tráfico en los routers del árbol y optimizar dicho árbol para que sea igual de eficiente para cualquier fuente y para todos los receptores.

2.6 Protocolos

Los algoritmos presentados en el apartado anterior aparecen implementados en una serie de protocolos; algunos de ellos con poco éxito, otros implementados en redes como MBONE[4]. Pero con el algoritmo de enrutado no es suficiente. En las redes multicast globales se necesitan 3 tipos de protocolos para poder cumplir los requisitos descritos a lo largo de este capítulo:

1. Un protocolo para administrar los grupos multicast (gestión de miembros)
2. Un protocolo de enrutado multicast, para comunicar los cambios en los grupos entre los routers y permitir el intercambio de datagramas entre ellos.
3. Un protocolo para permitir la comunicación entre sistemas autónomos a larga distancia.

En la tabla 2.1 se muestra un breve resumen de los conceptos aparecidos en el apartado anterior, junto con los protocolos descritos a lo largo de este apartado.

Tabla 2.1 Descripción de algoritmos y protocolos multicast

Mecanismo	Nombre	Algoritmo que usa
Simples	Flooding	Broadcast con protección anti-bucles
	Spanning Tree	Construcción manual del árbol multicast, flooding
Basados en SPT	RPF/RPB	Comprobación interfaz parent, flooding,
	TRPF/RPB	Comprob. si subred tiene receptores multicast, RPF/RPB
	RPM	Comprob. si subred tiene routers activos receptores de multicast, TRPF/TRPB.
	DVMRP	Rápida recuperación de ramas podadas (grafting), RPM
	PIM-DM	DVMRP

[4] Multicast Backbone, http://www.live.com/mbone/

Basados en SDT	**PIM-SM**	Shortest Path Tree, SDT
	MOSPF	Calcula árbol mediante estado-enlace, SDT
	CBT	División en regiones, SDT
Inter-AS	**HDVMRP**	DVMRP con niveles jerárquicos
	HPIM	Lista de candidatos a RP
	MBGP	Extensión a BGP
	GUM	Segmentación de espacio de direcciones multicast

Nota Tabla 2.1: Los nombres en negrita son protocolos, los demás son algoritmos.

2.6.1 Gestión de Miembros

Como ya se ha enunciado previamente, tenemos dos protocolos que realizan esta función: IGMP para IPv4 y MLD para IPv6. El primero es un protocolo estándar que encapsula su información directamente sobre la cabecera IP. El segundo es un subconjunto de comandos de ICMPv6 que realizan la misma función que los mensajes IGMP.

En ambos casos el objetivo es ofrecer una herramienta para comunicar interés en recibir tráfico multicast de un grupo determinado. La primera versión de IGMP no permitía comunicar el deseo de dejar de recibir tráfico, pero en IGMPv2 se creó un mensage llamado *Leave Group*. Los routers de redes con clientes multicast sólo necesitan saber si hay algún cliente. En cuanto reciben respuesta positiva de alguno de ellos informando de su intención de unirse a un grupo multicast, el router entonces informa a los routers vecinos, de las redes superiores, de su intención de recibir tráfico (para ello usa también IGMP). En el caso de que un router se quede sin nodos receptores en su subred, informará a la red superior de que quiere dejar de recibir tráfico multicast. A esto se le llama *prune*, e implica que cuando un router está en el extremo del árbol (router “hoja”), se le puede podar si se queda sin clientes. Entonces si un router superior se queda sin routers inferiores, también se poda del árbol. La reconstrucción del árbol, en el caso de que vuelvan a haber clientes, se llama *grafting*, y usando solamente IGMP tiene una gran latencia pues los routers interrogan a sus nodos hijos cada 125 segundos. Los protocolos de enrutado multicast aceleran y optimizan este proceso evitando problemas como los bucles o el desgaste de ancho de banda.

En todo este proceso de unión o salida del grupo multicast, tanto los clientes como los routers intermedios usan direcciones IP multicast (224.0.0.1 para todos los nodos y 224.0.0.2 para todos los routers). También se aprovecha la capacidad de multicast del hardware de red para optimizar la mensajería entre clientes y el router, sin molestar a nodos que no participan en ninguna sesión multicast.

2.6.2 Enrutado en un sistema autónomo

2.6.2.1 Distance Vector Multicast Routing Protocol (DVMRP)

DVMRP (RFC 1075) es uno de los primeros protocolos en utilizarse, y el más usado en la red MBONE. Está basado en el concepto de RIP usando el algoritmo vector-distancia. El protocolo permite a los routers comunicarse los grupos multicast activos y los costes de transferencia de datagramas. Con esa información, los routers construyen un árbol de distribución (del tipo SPT, el más rápido) diferente para cada pareja (S,G).

Se definen extensiones al protocolo IGMP para la comunicación entre los routers, añadiendo mensajes adicionales que permiten anunciar los grupos activos y los que se desea abandonar (*Leave Group*), y también permite interrogar a otros routers además de poder intercambiar tablas de enrutado con métricas de coste incluidas.

Su mecanismo de funcionamiento se basa en RPM, donde primero se inunda la red con el tráfico multicast, y se deja de enviar por una red si el tráfico es rechazado. Este estado se resetea cada cierto tiempo, volviendo a inundar la red y esperando mensajes *prune*. Además la información de miembros de grupos se propaga lentamente debido al algoritmo vector-distancia. Por eso DVMRP se considera el peor protocolo en cuanto a rendimiento, aunque es el más sencillo de implementar y desplegar.

Su mecanismo de *grafting*, o de recuperación de ramas podadas, es rápido y deja el árbol en su estado anterior a la poda. Además soporta redundancia de rutas hacia la fuente, con lo que para elegir una ruta única para evitar duplicación, elige la ruta con menor coste o IP de router menor (a éste se le llama router dominante), En caso de fallar, conmuta hacia la ruta redundante, ofreciendo así un poco de fiabilidad en el enrutado multicast.

2.6.2.2 Multicast extensions to OSPF (MOSPF)

El estándar MOSPF (RFC 1584) utiliza el sistema del protocolo de enrutado unicast OSPF basado en estado-enlace para calcular las rutas del árbol multicast. Las ventajas inmediatas de este algoritmo es que todos los routers MOSPF tienen el mismo árbol “mas corto” hacia la fuente, pues lo comparten ya que es un algoritmo basado en SDT. Además, el sistema gestiona la unión a los grupos multicast basándose en la demanda, es decir, se recibe tráfico multicast sólo si explícitamente se solicita, no como DVMRP que implica recibir tráfico siempre y dejar de recibir sólo si se rechaza.

El modo de funcionamiento de MOSPF es el mismo de OSPF: se necesita un *designated router* y un *backup router* para que lleven un control de los miembros activos de los grupos multicast. El árbol más corto lo calcula solamente el DR. La información se envía usando unos tipos especiales de LSA (Link State Advertisement). Esto permite el flujo normal de LSAs de OSPF, con lo que no interfiere con el enrutado unicast. Es más, se utiliza las tablas de enrutado OSPF para construir el árbol inicial basado en SDT más óptimo, según esté localizada la fuente en la red y según esté el designated router.

Cada router MOSPF tiene una tabla llamada *forwarding cache* para cada pareja (S,G). Esa tabla indica por qué interfaz (o router, o subred, o *upstream node*) debería llegar el tráfico multicast de esa pareja (S,G), además de indicar las interfaces, o routers o nodos (*downstream nodes*) por las que debería propagarse el tráfico. La **tabla de *forwarding*** se construye con dos componentes: la **información de miembros locales**, obtenida mediante IGMP en la subred directamente conectada al router; y la **información del árbol mas corto** (SPT), construida bajo demanda, pero obtenida gracias al designated router del área MOSPF. La primera tabla, la local, sirve para entregar datagramas multicast a nodos locales, y la segunda para entregarlos a nodos remotos. Los cambios en la tabla de miembros local son inmediatamente comunicados al resto de nodos mediante los *Group-Membership LSAs*.

El hecho de estar basado en algoritmos de estado-enlace obliga al router a tener un gasto de procesado elevado para calcular el árbol mas corto, cosa del todo imposible si tenemos en cuenta la organización jerárquica de las áreas que implica rutas no óptimas.

2.6.2.3 Core Based Trees (CBT)

Es un sistema basado en árbol compartido (SDT) que evita la propagación innecesaria y permite que todos los routers compartan el mismo árbol, aunque pueden usar diferentes protocolos de enrutado unicast pues CBT es independiente de éste. Está definido en los RFC 2189 y 2201. En CBT sólo se empieza a propagar tráfico multicast si alguien explícitamente lo solicita. El problema está en el tiempo entre que el cliente envía su mensaje IGMP (solicitando la unión a un grupo) y recibe tráfico multicast. Ese tiempo es el necesario para propagar la información de miembros por la red CBT. Para permitir mejorar el rendimiento, CBT propone dividir las redes en regiones, donde cada región tendrá un router central (*core router*) que centralizará las peticiones de unión a grupos. La división en regiones es jerárquica y hay un nivel superior (backbone). Los routers *no-core* de una región descubrirán al core dinámicamente, y le pasarán a él las peticiones de unión a grupos multicast. Si el router central falla, se designa uno nuevo que pueda llegar a las regiones backbone de CBT.

La principal ventaja de CBT es su poco coste computacional y de ancho de banda, pero su mayor desventaja es que la ruta del árbol multicast no es en absoluto la óptima.

2.6.2.4 PIM-DM

El Protocolo Independiente de Multicast, *Dense Mode*, o PIM-DM (RFC 2117) es bastante reciente. El hecho de considerar a éste protocolo de Modo Denso es porque presupone que la distribución de clientes en la red es muy densa, con lo que es muy probable que una red tenga clientes de un grupo multicast.

EL funcionamiento de PIM-DM es muy similar a DVMRP. La principal diferencia está en que como su nombre indica, PIM es independiente del protocolo de enrutado unicast utilizado. Se presupone el uso de un protocolo que compute el camino más corto a un destino, actualice la tabla de enrutado del router y la mantenga convenientemente. Así que no puede usar mecanismos como el *poison-reverse* para saber si un router vecino utiliza a otro router para llegar a la fuente. Por lo tanto, siempre se usará mensajes *Join* o *Prune* para unirse o abandonar un árbol multicast.

2.6.2.5 PIM-SM

PIM *Sparse Mode* (RFC 2362) está diseñado para ofrecer multicast a través de redes WAN con diferentes conexiones, algunas de ellas con poco ancho de banda. Se llama *sparse* porque presupone que los receptores están geográficamente dispersos, con lo que sólo enviará tráfico hacia esas redes que realmente lleven a clientes activos.

PIM-SM está basado en CBT y de la misma forma tiene un *Rendez-Vous Point (RP)* que centraliza la gestión de grupos multicast y que condiciona la creación de un árbol compartido (SDT) entre todos los routers PIM-SM. También es independiente del protocolo unicast. Los routers extremos reciben mensajes IGMP solicitando la unión al grupo, entonces ellos envían mensajes *join* en dirección al RP para que se le incluya en el árbol de distribución compartido.

La principal ventaja de PIM-SM sobre CBT es la posibilidad de cambiar de un modo de árbol compartido a un modo de *Shortest Path Tree* (SP tree). En este modo, el RP sigue siendo el mismo, pero tiene sentido cuando la fuente y el emisor están cerca pero el árbol

construido pasa lejos de esa ruta. El árbol compartido siempre tiene su base en el RP, pero los routers extremos pueden pasar al modo SP tree si ven que pueden conseguir una comunicación fuente-receptores mejor. Hay un límite (*threshold*) que indica cuándo pasar a SP o cuándo permanecer en el SDT. Si aparece un cliente muy alejado de la fuente, quizá conviene volver a utilizar el árbol compartido. PIM-SM permite estos cambios de forma eficiente.

2.6.3 Enrutado entre sistemas autónomos

El objetivo de establecer protocolos de comunicación estándares, interoperables, entre sistemas autónomos es permitir que IP multicast llegue a Internet de forma que pueda ser utilizado a gran escala. La escalabilidad, el minimizar el ancho de banda y los recursos de CPU consumidos son factores que aún no se han podido respetar al 100% con los actuales protocolos de enrutado multicast. Por eso se aplica la idea de sistemas autónomos, heredada del enrutado IP unicast en Internet, aunque hay diferentes alternativas en cuanto al protocolo a utilizar.

El primer intento de comunicar sistemas autónomos es HDVMRP (*Hierarchical DVMRP*) y trata de dividid las redes en diferentes regiones, cada una dominada por dos protocolos. El protocolo de nivel 1 seria el usado en el interior del área, y el de nivel 2 el usado para comunicar las diferentes áreas. Este sistema permite la coexistencia de CBT, PIM, DVMRP, etc. HDVMRP difiere de DVMRP en la información que intercambia, que en este caso está basada en la pareja (region_ID, group) y que permite construir el *inter-regional multicast delivery tree*.

Hay otra propuesta, el HPIM o *Hierarchical PIM*, que propone crear listas de candidatos a RP, cada uno de diferente nivel. Así pues, una vez creadas las listas de candidatos de nivel 1 y 2 (por ejemplo), el RP de nivel 1 se registra con el RP de nivel 2, creando así una via de comunicación entre las listas de nivel 1, a través del RP de nivel 2.

También se ha especificado el MBGP (*Multicast Border Gateway Protocol*), como extensión a BGP. La principal función de MBGP es distribuir y elegir los caminos a los diferentes AS (*autonomous systems*). Para ello se introduce la figura del MBR (*Multicast Border Router*).

Por último, se está estudiando el uso de GUM (*Grant Unified Multicasting*) para unificar el enrutado multicast en un solo protocolo de interconexión de áreas. Es una propuesta muy reciente del IETF. El principal problema en su desarrollo está en asociar los rangos de direccionamiento clase D (de IPv4) para permitir dominios raíz que repartan las direcciones entre los diferentes subdominios.

CAPÍTULO 3
Protocolo GeoPaging

3.1 Introducción

En este capítulo se explica más a fondo el protocolo GeoPaging. Primero se trata la problemática de la identificación de las celdas, luego se explica 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.

3.2 Direccionamiento

Como ya se ha comentado, 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 formar a partir de unas etiquetas que identifican a cada celda. Estas etiquetas tienen como principal propiedad que si un terminal móvil conoce las dos 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 para después proceder con el de la direcciones IPv6 asociadas a ellas.

3.2.1 Construcción de las etiquetas

En primer lugar se aproxima la cobertura de una celda con un hexágono. El conjunto (cluster) de siete hexágonos, dispuestos según la figura 3.1, conforman una celda de nivel superior. Si se agrupan siete clusters del mismo nivel tenemos otro cluster 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 numero de 0 a 6 resultado de sumar **2i + j**, como se observa en la figura 3.2.

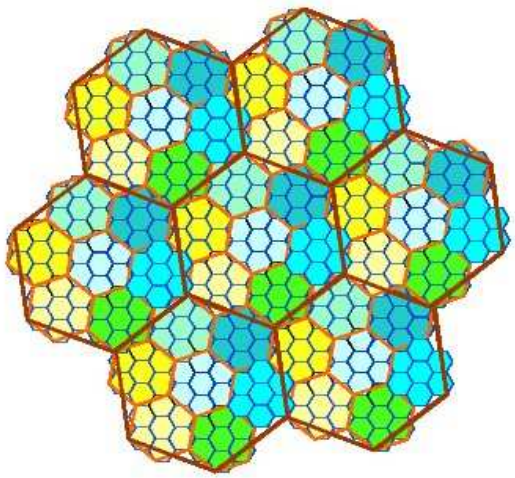

Figura 3.1. Red de clústeres con 3 niveles

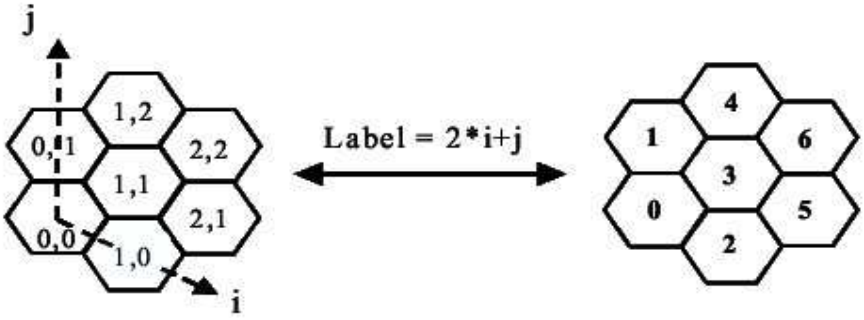

Figura 3.2. Sistema de coordenadas y de numeración de celdas

El hecho de usar una figura de 7 celdas permite utilizar simplemente 3 bits para identificarlos (del 0 a 6). En la figura 3.3 se puede observar un ejemplo de jerarquía en 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 clústeres de hasta 3 niveles, podemos tener la celda 1.3.4, la 0.2.5, la 6.2.4, etc. Para indicar que el cluster 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.

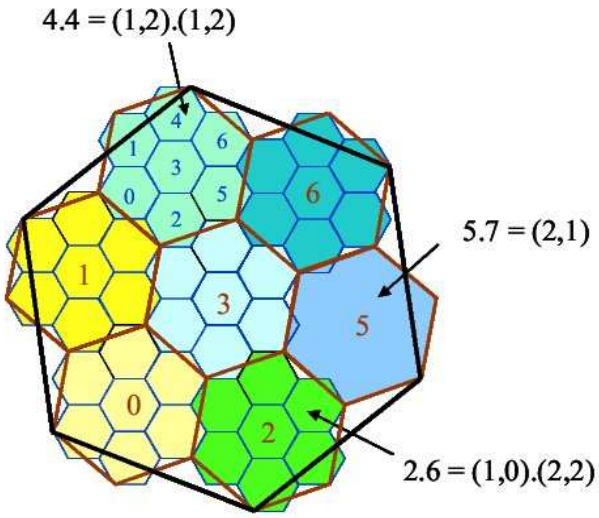

Figura 3.3. Ejemplo de jerarquía de tres niveles

Hay una pequeña dificultad con este sistema y es que al ir subiendo por las jerarquías, nos encontramos con que el eje de coordenadas (que es de 120º) va girando, como se observa en las figuras 3.4 y 3.5. Será cuestión del administrador de la red identificar las celdas en cada jerarquía marcando un criterio al respecto.

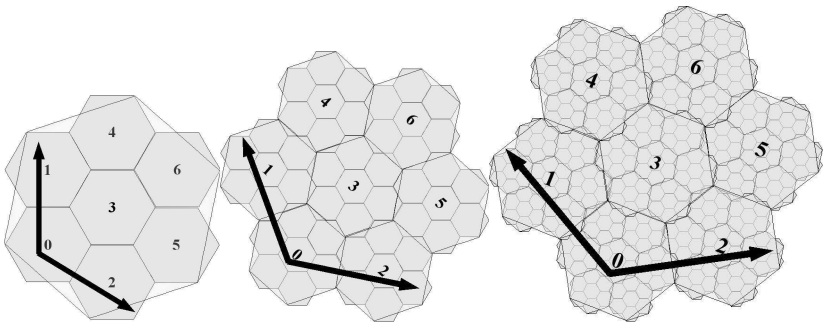

Figura 3.4. Ejemplo de clusteres de nivel 1, 2 y 3

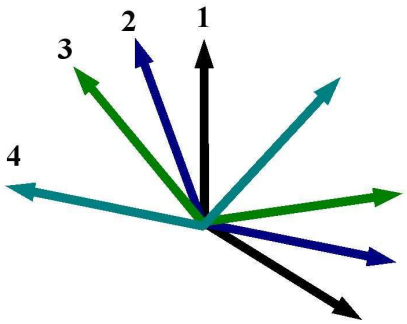

Figura 3.5. Diferencias en los ejes de coordenadas de los diferentes niveles de cluster

3.2.2 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 Acces 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 3.6).

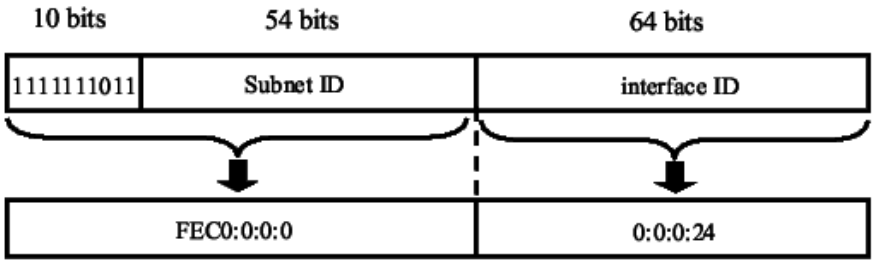

Figura 3.6. Dirección unicast asociada a la celda

La otra dirección es la multicast, usada 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. El trozo de 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 fines, por ejemplo se podría usar el ID 6666:0:6666 resultando así un prefijo de subred FF08:6666:0:6666/64 (figura 3.7).

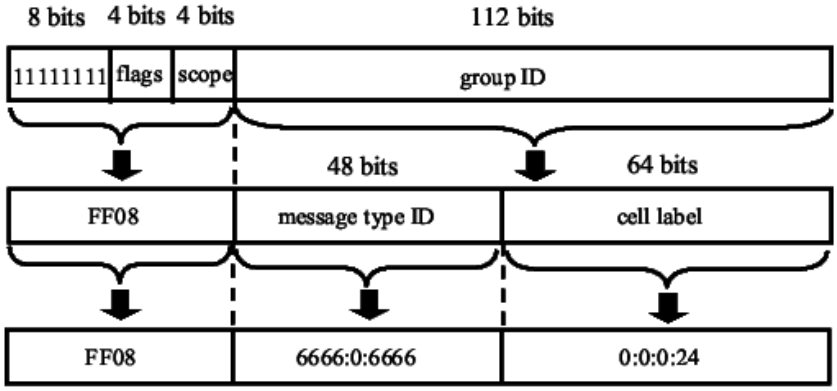

Figura 3.7. Dirección multicast asociada a la celda

Así pues, en los routers 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

3.3 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 mas sedentario (p.ej. 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*.

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 3.8 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 3.9 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 3.5.

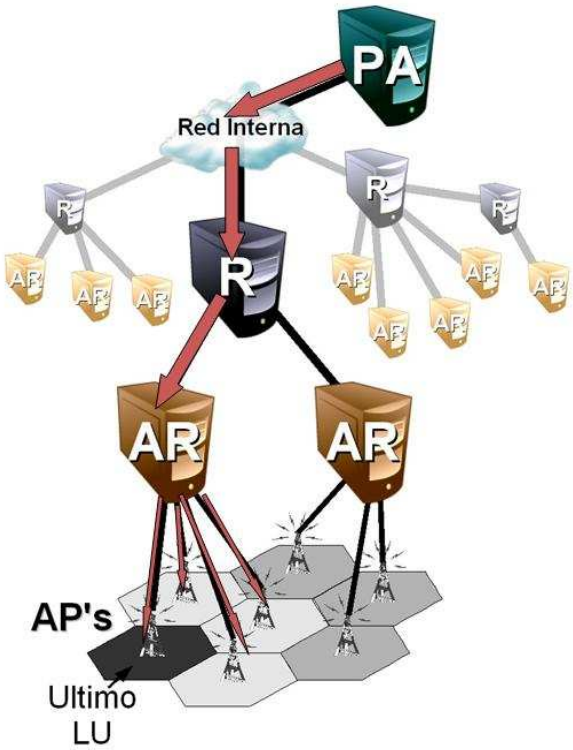

Figura 3.8: Paging con distancia 1

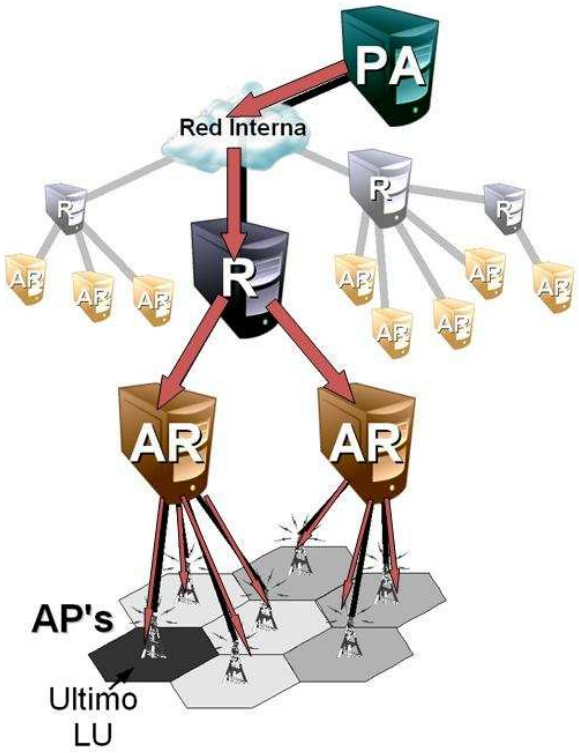

Figura 3.9: Paging con distancia 2

3.4 Construcción de tablas de enrutado

Es importante resaltar que las áreas de paging del protocolo Geopaging son distintas 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 ese área. El protocolo de encaminamiento unicast utilizado se encargará de dar a conocer esa área de paging al resto de AR.

Hay un posible problema, y es que tenemos una ruta por cada AP, habiendo miles de éstos en una red celular, con lo que tendríamos tablas de enrutado enormes. La solución está en la agregación de rutas, pues un AR controlará varios AP que probablemente estén cerca geográficamente, pudiendo entonces 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.

3.5 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 mero problema de geometría que se resume en contar las coronas que hay entre una celda y otra. Una corona no es más que el 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. Así pues, 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.

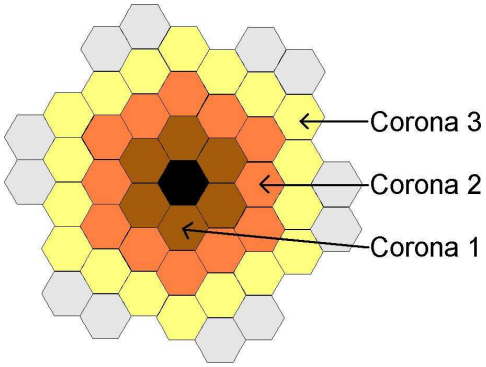

Figura 3.10: Aumento de la cantidad de celdas receptoras en función de la distancia

Si tuviéramos dos routers que controlasen dos áreas (ver figura 3.11), 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 ultima 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 3.1):

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

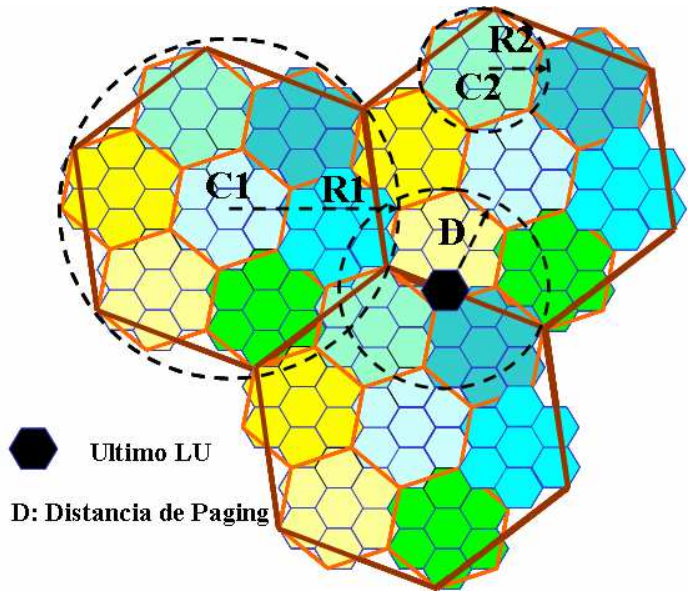

Figura 3.11: Propagación multicast basándose en la distancia de paging

Es decir, tanto el router 1 como el router 2 tienen que aplicar la fórmula 3.1 para cada una de sus rutas. Para simplificar, en el ejemplo de la figura 3.11 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 3.1.

Tabla 3.1: Resultado de aplicar la fórmula 3.1 en escenario de la figura 3.11

R1	**D**	**D [C1,LU]**	**Formula 3.1**	**Reenvia?**
4	2	6	6 – 4 <= 2?	SI
R2	**D**	**D [C2,LU]**	**Formula 3.1**	**Reenvia?**
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 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. Así pues, 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 3.2.1, 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. El cálculo entonces para saber la distancia entre dos celdas, sabiendo solamente sus ID, es un problema de geometría que se explica a fondo en [3].

CAPÍTULO 4
Implementación

4.1 Introducción

En el capítulo anterior se explica el funcionamiento del protocolo Geopaging. En este capítulo se explicará cómo se ha conseguido implementar dicho protocolo. La plataforma de desarrollo elegida ha sido GNU / Linux, con la distribución Gentoo Linux, usando el kernel 2.6

El software debe cumplir una serie de requisitos debido a la propia naturaleza del protocolo y a las limitaciones del entorno de trabajo escogido:

- El código fuente debe ser **flexible**, diseñado de tal manera que con un mínimo esfuerzo se pueda cambiar de librerías. Como se comenta en el anexo 3 y 4, hay varias alternativas a la hora de programar, y las escogidas no siempre son las mejores.

- También 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 lo más **optimizado** posible, pues un software de enrutado debe responder rápido ante un paquete entrante y no colapsarse ni tardar demasiado en aceptar otro paquete, provocando entonces pérdidas de paquetes. Por eso sería idóneo trabajar a bajo nivel de programación, realizando llamadas a sistema y buscando siempre la mayor rapidez de ejecución.

- Las librerías a utilizar deben ser de **bajo nivel** pues el software debe leer, modificar e inyectar paquetes IPv6 totalmente personalizados, y enrutarlos por las interfaces deseadas. Esto implica trabajar contra el sistema de encaminamiento interno del kernel.

- Es necesario que el protocolo sea **escalable**: debe funcionar tanto en redes pequeñas como en redes grandes. El software no debe imponer limitaciones al funcionamiento del protocolo.

- Para el buen funcionamiento del software se necesitan **tablas de enrutado** unicast consistentes. Se necesitará pues un protocolo de enrutado unicast funcionando en paralelo a nuestro programa, o bien mantener las tablas de enrutado de forma estática.

- Como ya se ha explicado, el protocolo no precisa de intercambio de información entre los routers Geopaging. Toda la construcción del árbol multicast se basa en los campos Hop By Hop de IPv6, donde se incluye un parámetro Distancia que condiciona el tamaño y forma del árbol multicast. Así pues el software será **autónomo** pues deberá actuar y decidir de forma independiente, desconociendo incluso cuántos routers vecinos existen.

Siguiendo estos requisitos se probaron una serie de librerías y recursos que ofrece Linux para buscar una solución fiable. El principal problema es que IPv6 es muy reciente, y aunque Linux 2.6 incorpora casi todas las librerías del RFC 3493 y 3542 (*Basic* y *Advanced Socket Interface Extensions for IPv6*), se encontraron ciertos problemas que

forzaron el uso de librerías alternativas. En los anexos 3 y 4 se detallan las diferentes opciones evaluadas y los problemas encontrados con cada una de ellas.

4.2 Descripción del Entorno

Un programa en Linux reside en un espacio de memoria llamado espacio de usuario. Si 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). 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. Así pues desde un principio asumimos que el software de Geopaging debe ser ejecutado como root.

Nuestro software deberá acceder a datos externos como la tabla de enrutado, la lista de interfaces del ordenador y las direcciones IPv6 configuradas en ellas. La interfaz Netlink ofrece una manera unificada de acceder a ellos, pero hubo problemas con el soporte para IPv6. Así pues se optó por la lectura de ficheros de texto de la interfaz Proc para obtener las rutas y las interfaces del sistema, y usamos llamadas del tipo IOCTL para acceder a datos como la dirección MAC de una interfaz. Estas interfaces se explican más a fondo en el anexo 4.

En cuanto a la programación relacionada con la captura y escritura de paquetes IPv6, se probó la interfaz estándar de Linux, que cumple con los RFC 3493 y 3542 anteriormente citados, pero 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 cambió. Se ha introducido un concepto nuevo (*ancillary data*) para dar cabida a la flexibilidad de la cabecera IPv6, que puede tener diferentes elementos y longitudes según las Extension Headers usadas. Este nuevo concepto complica muchísimo la programación necesaria, obligando a usar otra interfaz de más bajo nivel (packet), manejando tramas Ethernet en vez de paquetes IPv6. Así pues se optó por usar la veterana librería libpcap para escuchar paquetes, y libnet para inyectarlos, aunque hay otra opción, gnet pero no soporta tantas opciones de IPv6). Estas librerías se explican en el anexo 3.

4.3 Conceptos

A la hora de programar, se tuvieron que idear tres conceptos para solucionar tres requerimientos del protocolo:

1. El agente de Paging puede estar en cualquier lado de la red, no podemos predecir su posición y por tanto el código debe estar preparado para recibir una petición de paging por cualquier interfaz

2. El protocolo debe crear un árbol multicast no basándose en información intercambiada sobre miembros de un grupo (como DVMRP o MOSPF, ver capítulo 3), sino en cálculos basados en distancias entre direcciones IPv6.

3. El árbol debe ser lo más óptimo posible. Sin intercambio de información no es posible implementar RPB mejorado, pero si que se debe implementar RPB.

Para permitir el primer requerimiento es necesario hablar de **contextos** (ver figura 4.1). Un contexto es un entorno de trabajo donde el software sabe que tiene una sola interfaz, llamada **parent**, y N-1 interfaces llamadas **childs**, en el caso de un equipo con N interfaces de red activas (configuradas adecuadamente con IPv6). Este equipo tendrá funcionando concurrentemente N contextos. Un contexto recibirá paquetes por su interfaz parent (llamado entonces interfaz **uplink**), y los propagará por las interfaces childs si el algoritmo del protocolo lo considera necesario (en ese caso se llaman interfaces **downlink**). Los filtros que el contexto impone a la librería de captura impiden que lleguen paquetes con origen diferente a la interfaz parent. Así pues, venga de donde venga el paging, un solo contexto se activará y aplicará el algoritmo.

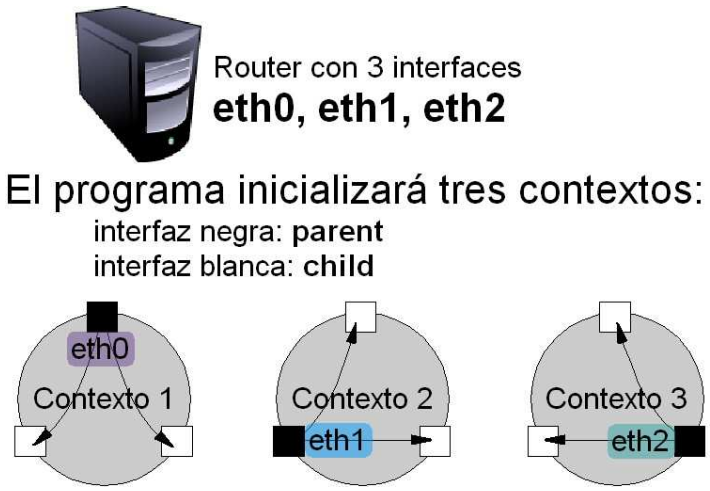

Figura 4.1: Ejemplo de router con 3 contextos activos

El segundo requerimiento se consigue mediante la interpretación de los mensajes contenidos en las Hop by Hop Extension Headers que el Paging Agent introduce en sus paquetes IPv6. En nuestro código consideramos que las Hop by Hop contendrían mensajes con la sintaxis siguiente: "[Paging_Distance]=N" donde N es un número entero. El programa entonces coge su lista de rutas y calcula la distancia que hay entre la dirección destino del paquete y cada una de ellas. Si se cumple la fórmula 3.1 (vista en el apartado 3.5), la interfaz child que lleva a esa ruta sirve para propagar el paquete de paging, considerada entonces una interfaz downlink. Este sistema evita toda transferencia de información entre routers.

Igualmente el tercer requerimiento (optimizar) se consigue con el mecanismo anterior. Sólo se propaga un paquete por donde es realmente necesario, siendo ése el caso en que la distancia calculada entre la ruta y la dirección del paquete sea menor que la distancia especificada en la Hop by Hop Ext. Header (ver fórmula 3.1). La eficiencia del protocolo reside en una buena organización de la red, con unas tablas de encaminamiento unicast consistentes y óptimas.

En [3] se explica cómo calcular la distancia entre dos celdas sabiendo su ID. Pero el ID está basado en jerarquías de clústeres de distinto nivel. Así que se aproxima la distancia entre niveles, tomando como unidad la celda (o cluster de nivel 1). En la tabla 4.1 se muestran los resultados (redondeados al entero) de la función *GEOPAG_inicializa_R()* siguiendo una aproximación basada en el productorio de raíz de 7. En cada iteración se suma 1, para aproximar aún más los resultados. El límite está en 21 niveles, a 3 bits por

cluster son 63 bits que caben en los 64 bits reservados para el direccionamiento en GeoPaging.

Tabla 4.1: Distancias entre clústeres de distinto nivel.

Nivel:	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**
Distancia	3	8	19	59	139	344	908	2402	6353	16808	44468	117649	311270

Nivel:	**15**	**16**	**17**	**18**	**19**	**20**	**21**
Distancia	823543	2178890	5764801	15252229	40353605	106765601	282475233

4.4 Descripción funcional del programa

El programa se ejecuta dando por parámetro el prefijo de las direcciones multicast y unicast virtuales a utilizar. Como ya se indicó en el capítulo 4**,** serán prefijos de red /64.

Lo primero que hace el programa al iniciarse es crear e inicializar una serie de variables en memoria. En concreto tenemos 3 estructuras globales: t*odas_rutas, todas_ifaces* y *ifaces_activas*. El primero guarda todas las rutas leídas del fichero /proc/net/ipv6_route. El segundo, guarda las interfaces de red leídas desde /proc/net/if_inet6. El tercero guarda las interfaces activas, es decir, las que tienen una dirección IPv6 configurada. También hay dos variables globales llamadas *rango_unicast* y *rango_mcast* que como su nombre indica guardan los prefijos IPv6 usados para mapear los ID de celda (ver apartado 3.2.2).

A continuación se crean los contextos, tantos como interfaces haya en la estructura *ifaces_activas*. Utilizamos libpcap, el cual necesita un filtro para capturar los paquetes deseados: los que entren por la interfaz parent. El problema está en que no hay un filtro del tipo “obtener solo paquetes recibidos”, sino que capturamos lo recibido y lo enviado. Así que se recurrió a filtrar por las direcciones MAC origen, eliminando todo el tráfico creado por la máquina y adquiriendo sólo el de otras máquinas, con IP destino el rango multicast especificado por parámetro. El filtro es el mismo para cada contexto, pero se especifica a libpcap la interfaz de trabajo (la parent).

Dicha librería libpcap funciona de la siguiente manera: bloquea el hilo de ejecución del contexto hasta que recibe un paquete. En el caso de ser ethernet, se ejecuta el manejador del protocolo ethernet, definido por el usuario. Lo mismo pasa con IPv6, ICMP, TCP, etc. Así pues en la función que maneja los paquetes realizamos comprobaciones para garantizar que el paquete está bien construido, que contiene una Extension Header con un parámetro *Paging_Distance*, etc.

Finalmente se ejecuta una rutina que hace 4 comprobaciones con el paquete, el cual consta de Cabecera IPv6, Extension Header y Payload. Se ejecutan en orden, y en el momento que falla una, se descarta el paquete.

- Se comprueba el campo Hop Limit, si es menor que 1 se descarta, sino se decrementa. Si ahora se propagase el paquete por todas las interfaces, el router habría hecho una propagación tipo Broadcast.

- Se aplica el algoritmo RPB (visto en apartado 2.5.2.1): averiguamos si la ruta hacia la dirección IPv6 origen del paquete coincide con nuestra interfaz parent. En ese caso se

confirma que el paquete ha llegado por la interfaz uplink. Si no, es que nos ha llegado el paquete por una interfaz que no corresponde, por ejemplo un enlace redundante, y si se reenviase probablemente estaríamos creando un bucle.

- Se aplica el algoritmo del Geopaging (ver capítulo 4). Primero conseguimos la lista de interfaces childs. Segundo se itera por esas interfaces. En cada iteración se filtran las rutas para obtener sólo aquellas accesibles via la interfaz iterada. Tercero se itera por esas rutas calculando la distancia que hay entre la ruta y la dirección IPv6 destino. Si la distancia es igual o menor a la especificada en la Hop by Hop Ext. Header, se agrega la interfaz a la lista de interfaces downlink y se pasa a iterar a la siguiente interfaz child.

- Se procede a reinyectar el paquete por todas las interfaces downlink. La librería usada es libnet, la cual calcula automáticamente los checksums de ethernet (IPv6 no tiene CRC).

El contexto mientras tanto va mostrando por pantalla si el paquete recibido cumple los pasos comentados, y una vez ha acabado de comprobarlos vuelve a su punto inicial, esperando la llegada de otro paquete. Si llegan dos paquetes simultáneos por la misma interfaz, se guardan en memoria gracias a la librería libpcap. Si llegan por diferentes interfaces, al tener diversos contextos concurrentes se procesan los dos paquetes simultáneamente.

Para implementar los contextos de forma concurrente se tuvo que elegir la librería de threads POSIX (pthreads) en vez de la ejecución mediante múltiples procesos (usando fork), pues éste último no comparte la memoria y sería necesario mecanismos de IPC para controlar el acceso a los datos compartidos, como la tabla de enrutado o la de interfaces.

4.5 Descripción del proceso de enrutado de un paquete

En este apartado se describen las variables y funciones utilizadas para recibir, guardar en memoria, procesar y reenviar un paquete a lo largo del programa. Para más información será necesario recurrir a los comentarios del código fuente.

Es fundamental para la comprensión de este proceso entender el mecanismo elegido para modularizar el código. Se han separado los nombres de funciones y variables mediante prefijos que indican su utilidad. Por ejemplo, el prefijo PCAP_ indica captura de paquetes. El prefijo NET_ indica inyección de paquetes en la red, RT_ para la obtención de la tabla de encaminamiento, IFS_ para la tabla de interfaces, GEOPAG_ recoge las funciones relacionadas con el algoritmo de enrutado y finalmente UTIL_ son funciones varias. Este sistema permite de una forma sencilla cambiar las librerías utilizadas por otras, como las estudiadas en los anexos 3 y 4. Por ejemplo, en PCAP se utiliza libpcap, en NET se utiliza libnet, y en RT y IFS se utilizan lecturas al sistema de ficheros Proc.

Como ya se ha explicado en el apartado anterior, se usan contextos (función **nuevo_contexto*) para trabajar adecuadamente con la librería libpcap. Así pues, tenemos tantos hilos de ejecución (*threads*) como contextos, o lo que es lo mismo, tantos hilos como *interfaces_activas*. También tenemos un thread extra (función *RT_bucle_parsea_from_proc*) que se encarga de mantener actualizada la variable global *todas_rutas*. Sin embargo, la lista de interfaces del sistema sólo se lee una vez, al principio del programa, mediante la función *IFS_parsea_from_proc().*

Imaginemos que llega un mensaje de paging por la interfaz eth0. 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 (con una extension header correcta, ver apartado 4.3), ejecutamos la rutina *GEOPAG_router()* donde se aplican los cuatro pasos vistos en el apartado anterior. En ellos sólo se modifica el campo *hlim* (Hop Limit) de la cabecera IPv6, decrementándolo. El resto del paquete no se modifica. En el último paso se ejecuta en cada interfaz downlink la función *NET_libnet_inject()*, que reinyecta el paquete por esas interfaces.

CAPÍTULO 5
Pruebas

5.1 Introducción

Una vez explicado el funcionamiento del protocolo y descrita la implementación que se ha realizado, mostraremos las pruebas realizadas para ver si funciona tal y como rige el protocolo GeoPaging. El objetivo de este capítulo es demostrar el buen funcionamiento de la implementación del protocolo. Se creó una topología virtual que se simuló en laboratorio mediante el uso de tres ordenadores PC, interconectados mediante redes ethernet. También se usó un equipo auxiliar, un portátil, para que hiciese las funciones de agente de paging.

5.2 Descripción del escenario

En la figura 5.1 se muestra los equipos y la interconexión entre ellos que se utilizaron. Las interfaces de red se llaman ethX, aunque también hay interfaces virtuales como la dummy0 (ver anexo 2, descripción de la maqueta). Cada segmento de red tiene un nombre (p.ej. red 1) para facilitar la comprensión al lector a lo largo del capítulo.

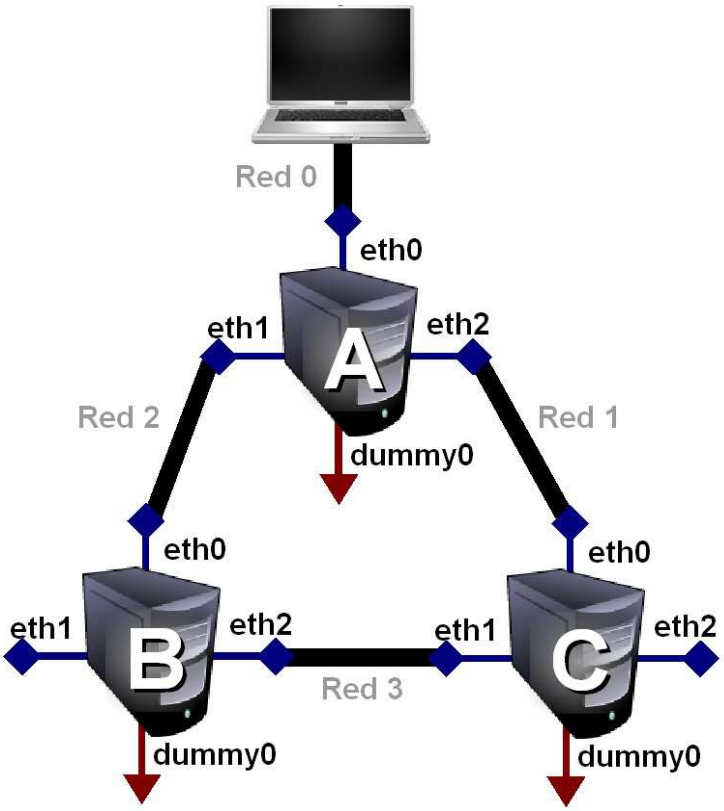

Figura 5.1 Esquema de elementos del escenario de pruebas

Las especificaciones de hardware, configuración de red, velocidades de transmisión, etc se encuentran detalladas en el anexo 2. Tenemos tres PC que hacen de router, cada uno

con tres interfaces de red ethernet activas. También usaremos un un portátil que actuará de PA y será quien inyecte los paquetes de paging a la red.

La topología anterior representa un escenario como el expuesto en esta figura (5.2):

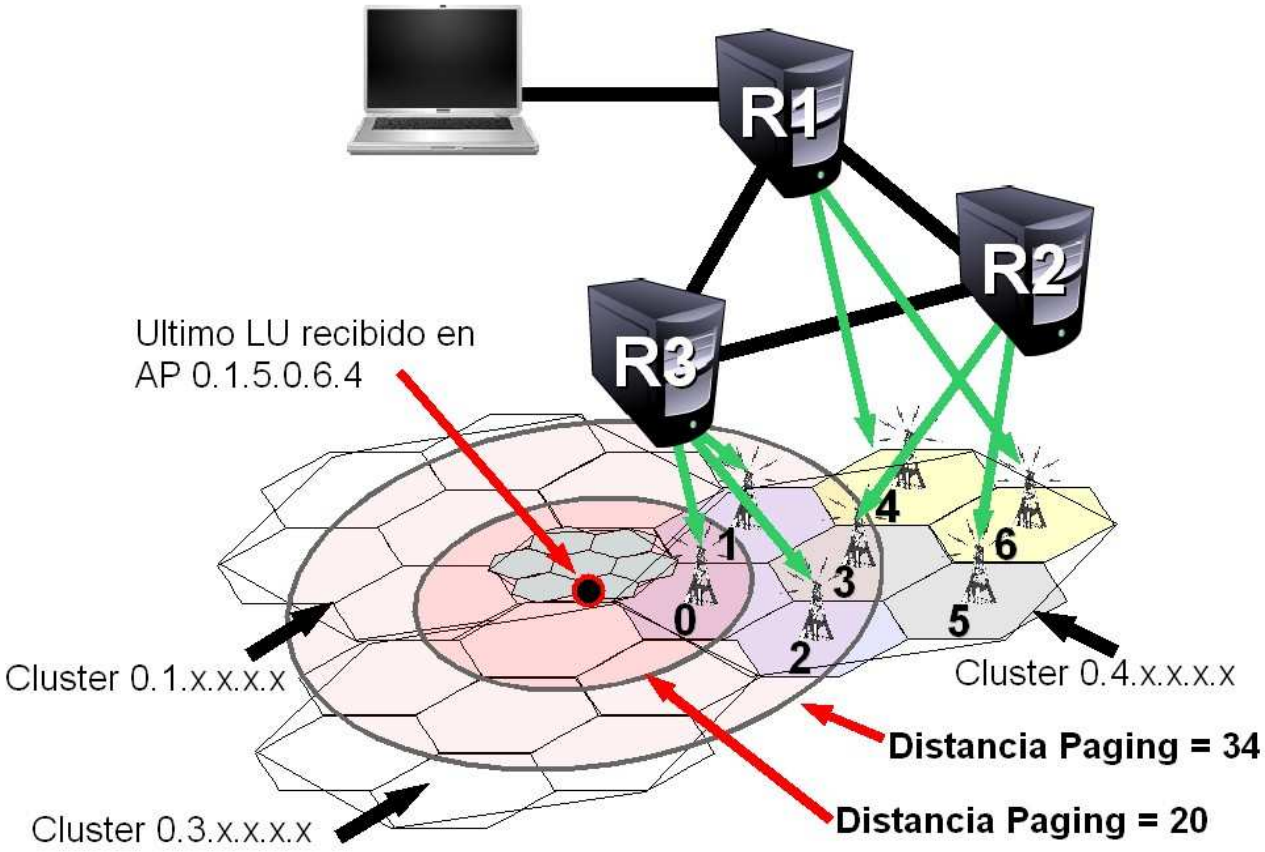

Figura 5.2: Representación del escenario virtual.

Los routers llamados R1, R2 y R3 serán los equipos del laboratorio A, B y C, respectivamente. Las flechas verdes representan las direcciones virtuales de los APs agregadas a la tabla de encaminamiento del router. La celda que contiene un punto negro representa la celda donde se recibió el último LU, por lo tanto será el destino de la petición de paging enviada por el portátil, que hará de PA. Los círculos que rodean dicha celda representan las áreas de paging establecidas si el parámetro Distancia de Paging fuese de 32 o de 20 celdas de radio.

Un resumen de las configuraciones de los equipos se encuentra en la siguiente tabla (5.1)

Tabla 5.1: Resumen de configuraciones

ID de equipo	Interfaz	IPs configuradas	Rutas accesibles
R1	eth0	FEDE::A:1 /112	
		FE80::A:1 /112	
	eth1	FEDE::1:1 /112	
		FE80::1:1 /112	
	eth2	FEDE::2:1 /112	
		FE80::2:1 /112	
	dummy0	FEDE::7:1 /112	FEC0::4800/119
			FEC0::4C00/119
ID de equipo	**Interfaz**	**IPs configuradas**	**Rutas accesibles**
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::4600/119
			FEC0::4A00/119
ID de equipo	**Interfaz**	**IPs configuradas**	**Rutas accesibles**
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::4000/119
			FEC0::4200/119
			FEC0::4400/119
ID de equipo	**Interfaz**	**IPs configuradas**	**Rutas accesibles**
Portátil (PA)	eth0	FEDE::A:E /112	

Las rutas accesibles son las direcciones virtuales de los AP, creadas a partir de los ID de celda. Por ejemplo la celda 0.4.2.0.0.0 es la IP FEC0::4400/119. El rango de direcciones virtuales es FEC0::/64, y el rango de direcciones unicast (site-local) asignados para administrar la red es FEDE::/64. De la misma forma, se define un rango de direcciones link-local (FE80::/64) para dotar de mayor claridad las tablas de enrutado, pues el protocolo de encaminamiento RIPng utiliza este tipo de direcciones. En la topología de la figura 5.1, la red 0 es FEDE::A:0/112, la 1 es FEDE::1:0/112, la 2 es FEDE::2:0/112 y la 3 es FEDE::3:0/112.

5.3 Objetivo de las pruebas

El principal objetivo es comprobar si el programa actúa como el protocolo especifica, es decir, si crea el árbol multicast correctamente y entrega el paquete inyectado por el PA a todos los receptores. Como no tenemos AP receptores solamente comprobaremos que el paquete se propaga por los segmentos de red adecuados. En este caso las interfaces dummy0 se consideran enlaces virtuales a los AP (enlaces que en realidad son sólo a nivel 2). Si un paquete se reenvía por dicha interfaz, consideraremos que el mensaje de

paging se habría entregado a los APs conectados a dicho router. Dicho paquete se describe en la figura 5.3.

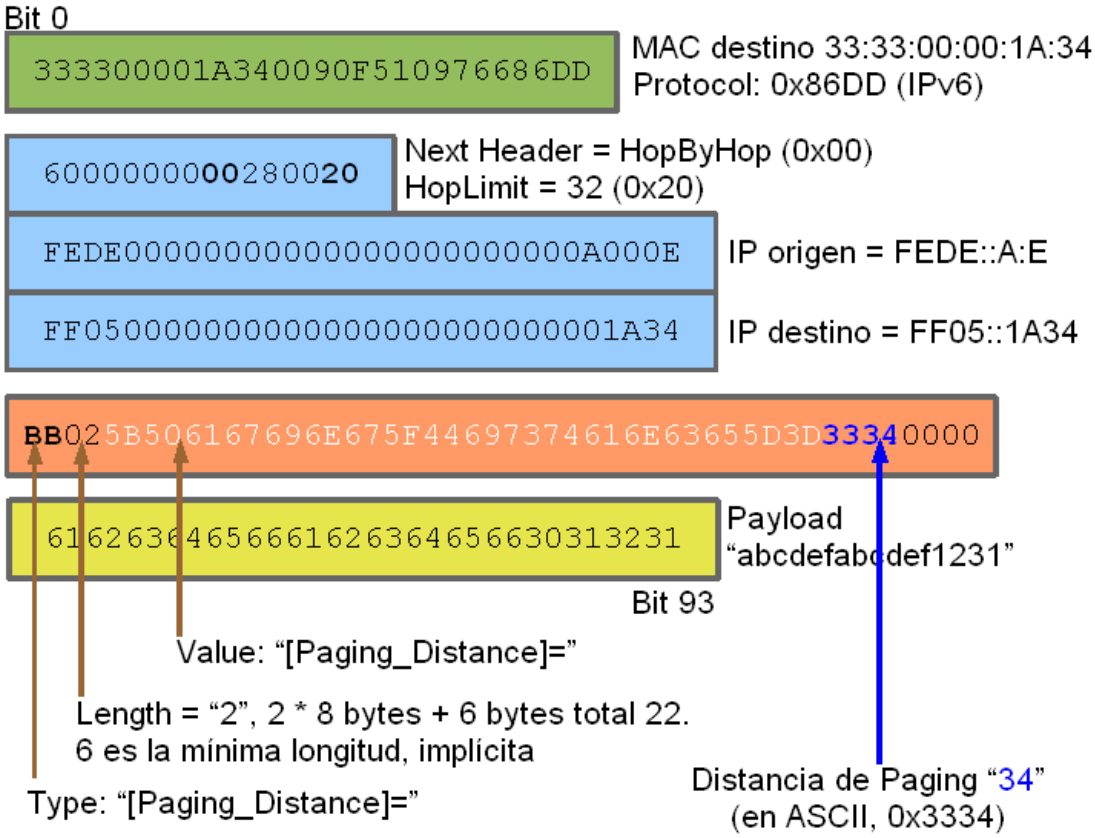

Figura 5.3: Paquete de paging

En la figura se usa el color verde para la cabecera ethernet, el azul para IPv6, el naranja para la Extension Header de IPv6 y en amarillo los datos que contiene el paquete. Como se puede observar, no se incluyen datos que identifiquen al agente móvil destinatario del paging, tal y como se comentó en el apartado 3.3, para simplificar la interpretación del paquete. Se comprobó que el uso de Extensión Headers adicionales a la Hop by Hop (que transporta la distancia de Paging) no afectan al funcionamiento del programa.

Se puede observar que la MAC destino del paquete es del tipo multicast, explicado en el apartado 2.2. El campo Hop Limit es 32 al salir del PA, por eso la dirección IPv6 origen es FEDE::A:E (en el anexo 2 se desglosan todas las IPs configuradas). Es importante notar que la extensión Hop By Hop contiene la cadena de caracteres “[Paging_Distance]=34”, y que dicha cadena sería diferente si se aplica la distancia de 20.

El comportamiento esperado en todos los casos es similar al del algoritmo RPB, solo que los grupos multicast tendrán diferentes receptores según la distancia de paging del paquete. Las pruebas servirán para observar la dinámica con que los routers deciden por qué interfaces deben propagar el paquete, es decir, cuáles forman parte del árbol multicast para el grupo y distancia de ese paquete.

En cada prueba se enviarán dos mensajes de paging diferentes, para observar cómo afecta la distancia de paging al árbol de multicast que se construye. En la tabla 5.2 se

pueden ver los cálculos de distancia que hay desde la dirección de paging destino seleccionada a las rutas del escenario

Tabla 5.2: Resumen de celdas configuradas en cada router

Destino: FEC0::1A34 (AP 0.1.5.0.6.4)			
ID de celda	**IP virtual**	**Distancia al Destino**	**Router a quien pertenece**
0.4.0.0.0.0	FEC0::4000/119	5	R3
0.4.1.0.0.0	FEC0::4200/119	5	R3
0.4.2.0.0.0	FEC0::4400/119	24	R3
0.4.3.0.0.0	FEC0::4600/119	24	R2
0.4.4.0.0.0	FEC0::4800/119	24	R1
0.4.5.0.0.0	FEC0::4A00/119	43	R2
0.4.6.0.0.0	FEC0::4C00/119	43	R1

Es necesario explicar que se usó una ID de celda del tipo 0.4.1.0.0.0 para representar a clústeres de nivel 4 sin niveles inferiores, pero estrictamente hablando deberían tener ID del tipo 0.4.1.7.0.0 según se especifica en el apartado 3.2.1. Pero como añadimos la ruta con una máscara /119 (en vez de /128), se ignoran los bits reservados para los ID de los tres niveles inferiores (9 bits reservados, 3 niveles ignorados) con lo que estamos agregando todas las rutas inferiores en una sola ruta superior (/119). Así que el resultado es el mismo aunque agreguemos las rutas usando la dirección IP de la celda 0.4.1.7.0.0. Para más información, consultar el método de agregación de rutas del apartado 3.4.

5.4 Prueba 1

5.4.1 Descripción

En esta primera prueba comprobaremos que el programa puede reenviar paquetes por todas las interfaces. Para ello enviaremos un paquete con una distancia suficiente como para entrar dentro de todas las áreas configuradas en los tres routers.

5.4.2 Esquema

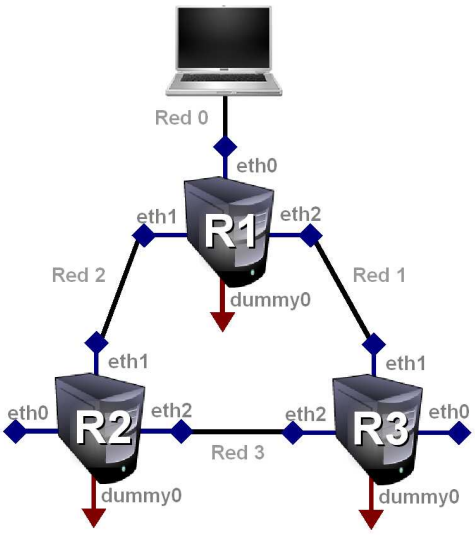

Figura 5.4: Prueba 1

El router 1 es el gateway hacia la red 0 para el R1 y el R2. En las siguientes tablas mostraremos la tabla de enrutado de todos los routers. Las rutas link-local y multicast las omitimos. Es importante resaltar la aparición de las rutas link-local (FE80::/64) como próximo salto en la tabla de encaminamiento; esto se debe al uso de estas direcciones en el protocolo RIPng.

Tabla 5.3: Rutas del R1

```
fec0::4000/119 via fe80::1:3 dev eth2  proto zebra  metric 1024  mtu 1500 advmss 1440
fec0::4200/119 via fe80::1:3 dev eth2  proto zebra  metric 1024  mtu 1500 advmss 1440
fec0::4400/119 via fe80::1:3 dev eth2  proto zebra  metric 1024  mtu 1500 advmss 1440
fec0::4600/119 via fe80::2:2 dev eth1  proto zebra  metric 1024  mtu 1500 advmss 1440
fec0::4800/119 dev dummy0  metric 1  mtu 1500 advmss 1440
fec0::4a00/119 via fe80::2:2 dev eth1  proto zebra  metric 1024  mtu 1500 advmss 1440
fec0::4c00/119 dev dummy0  metric 1  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::1:3 dev eth2  proto zebra  metric 1024  mtu 1500 advmss 1440
fede::7:0/112 dev dummy0  metric 256  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 5.4: Rutas del R2

```
fec0::4000/119 via fe80::3:3 dev eth2  proto zebra  metric 1024  mtu 1500 advmss 1440
fec0::4200/119 via fe80::3:3 dev eth2  proto zebra  metric 1024  mtu 1500 advmss 1440
fec0::4400/119 via fe80::3:3 dev eth2  proto zebra  metric 1024  mtu 1500 advmss 1440
fec0::4600/119 dev dummy0  metric 1  mtu 1500 advmss 1440
fec0::4800/119 via fe80::2:1 dev eth1  proto zebra  metric 1024  mtu 1500 advmss 1440
fec0::4a00/119 dev dummy0  metric 1  mtu 1500 advmss 1440
fec0::4c00/119 via fe80::2:1 dev eth1  proto zebra  metric 1024  mtu 1500 advmss 1440
fede::1:0/112 via fe80::3:3 dev eth2  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::7:0/112 via fe80::2:1 dev eth1  proto zebra  metric 1024  mtu 1500 advmss 1440
fede::8:0/112 dev dummy0  metric 256  mtu 1500 advmss 1440
fede::9:0/112 via fe80::3:3 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::3:3 dev eth2  proto zebra  metric 1024  mtu 1500 advmss 1440
```

Tabla 5.5: Rutas del R3

```
fec0::4000/119 dev dummy0  metric 1  mtu 1500 advmss 1440
fec0::4200/119 dev dummy0  metric 1  mtu 1500 advmss 1440
fec0::4400/119 dev dummy0  metric 1  mtu 1500 advmss 1440
fec0::4600/119 via fe80::3:2 dev eth2  proto zebra  metric 1024  mtu 1500 advmss 1440
fec0::4800/119 via fe80::1:1 dev eth1  proto zebra  metric 1024  mtu 1500 advmss 1440
fec0::4a00/119 via fe80::3:2 dev eth2  proto zebra  metric 1024  mtu 1500 advmss 1440
fec0::4c00/119 via fe80::1:1 dev eth1  proto zebra  metric 1024  mtu 1500 advmss 1440
fede::1:0/112 dev eth1  metric 256  mtu 1500 advmss 1440
fede::2:0/112 via fe80::1:1 dev eth1  proto zebra  metric 1024  mtu 1500 advmss 1440
fede::3:0/112 dev eth2  metric 256  mtu 1500 advmss 1440
fede::7:0/112 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
```

5.4.3 Comportamiento teórico

Viendo la tabla de enrutado, podemos predecir que el paging de distancia 34 hará que se propague el paquete por todas las interfaces de todos los routers. Sin embargo con la Distancia (D) de paging 20, el R1 y el R2 no enviarán paging por sus interfaces dummy0

pues sus APs no están dentro del área de paging. Como el R1 sabe los APs del R2, tampoco le enviará el paquete porque sabe que por el R2 no se llega a ningún AP válido. En la figura 5.5 se observan estos efectos:

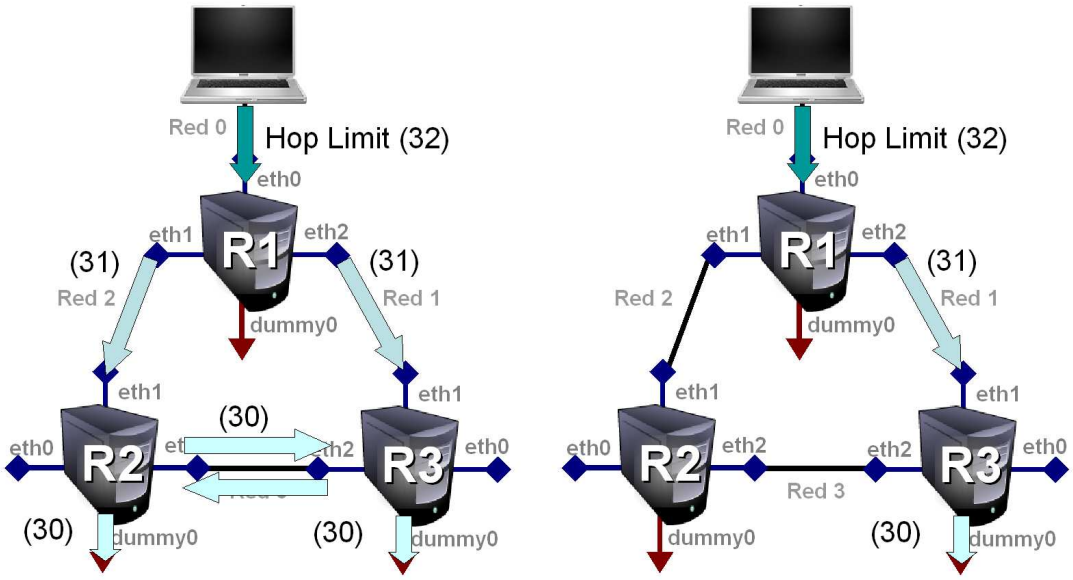

Figura 5.5: Comportamiento Prueba 1 con D=34 y D=20

5.4.4 Resultado

En el caso de paging con D=34 vemos que el paquete se envía en todas las interfaces del router 1, incluída la dummy0 pues sus propios AP forman parte del área de paging. También el router 2 y el 3 reenvían en sus interfaces dummy0 pues sus AP forman parte del área. Pero en la red 2 se transmiten dos paquetes de paging. Esto es una consecuencia del algoritmo RPB, que como se comentó en el apartado 2.5.2, que se evitaría esa duplicación usando TRPB. Esto se debe a que el router 2 observa AP's válidos (del R3) por esa interfaz. Lo mismo le pasa al R3. Es importante resaltar que aunque existen esos paquetes duplicados de la red 3, el software de GeoPaging los descarta porque llegan por una interfaz distinta a la parent (la eth1, que lleva al PA via R1).

En el caso D=20 observamos cómo el R1 envía únicamente por su interfaz eth2 porque en su tabla de enrutado sólo encuentra APs válidos por esa interfaz. Ni su interfaz dummy0 ni las rutas que anuncia el R2 son aptas para el área de paging, con lo que no formarán parte del árbol multicast. Para mayor claridad es conveniente consultar la figura 5.2 donde se ve claramente que la D=20 no abarca ninguna celda controlada por el R1 o R2, pero en cambio la D=34 sí que cubre alguna de sus celdas (aquellas con distancia al destino 24 en la tabla 5.2).

Es importante entender que el campo Hop Limit se va decrementando a lo largo de su paso por los routers. Este efecto se representa en las ilustraciones cambiando el color de las flechas e indicando el valor del Hop Limit entre paréntesis.

5.5 Prueba 2

5.5.1 Descripción

En esta prueba observaremos qué tal se comporta el protocolo ante la caída de un enlace (el de la red 1). No se medirá el tiempo de reacción ante el cambio pues está fuertemente condicionado al tiempo configurado en el protocolo RIPng de enrutado unicast.

5.5.2 Esquema

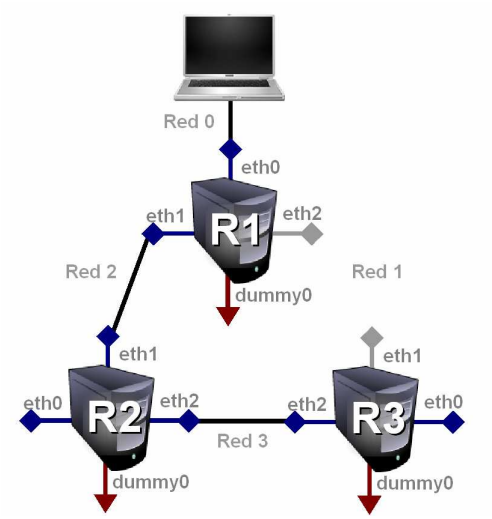

Figura 5.6: Prueba 2

En las tablas siguientes se muestra un pequeño extracto de las tablas de enrutado, para observar las diferencias con el escenario anterior. Tanto el R3 como el R1 pasan siempre por R2 para llegar al resto de APs.

Tabla 5.6: Rutas del R1

```
fec0::4000/119 via fe80::2:2 dev eth1  proto zebra  metric 1024  mtu 1500 advmss 1440
fec0::4200/119 via fe80::2:2 dev eth1  proto zebra  metric 1024  mtu 1500 advmss 1440
fec0::4400/119 via fe80::2:2 dev eth1  proto zebra  metric 1024  mtu 1500 advmss 1440
fec0::4600/119 via fe80::2:2 dev eth1  proto zebra  metric 1024  mtu 1500 advmss 1440
fec0::4800/119 dev dummy0  metric 1  mtu 1500 advmss 1440
fec0::4a00/119 via fe80::2:2 dev eth1  proto zebra  metric 1024  mtu 1500 advmss 1440
fec0::4c00/119 dev dummy0  metric 1  mtu 1500 advmss 1440
```

Tabla 5.7: Rutas del R2

```
fec0::4000/119 via fe80::3:3 dev eth2  proto zebra  metric 1024  mtu 1500 advmss 1440
fec0::4200/119 via fe80::3:3 dev eth2  proto zebra  metric 1024  mtu 1500 advmss 1440
fec0::4400/119 via fe80::3:3 dev eth2  proto zebra  metric 1024  mtu 1500 advmss 1440
fec0::4600/119 dev dummy0  metric 1  mtu 1500 advmss 1440
fec0::4800/119 via fe80::2:1 dev eth1  proto zebra  metric 1024  mtu 1500 advmss 1440
fec0::4a00/119 dev dummy0  metric 1  mtu 1500 advmss 1440
fec0::4c00/119 via fe80::2:1 dev eth1  proto zebra  metric 1024  mtu 1500 advmss 1440
```

Tabla 5.8: Rutas del R3

```
fec0::4000/119 dev dummy0  metric 1  mtu 1500 advmss 1440
fec0::4200/119 dev dummy0  metric 1  mtu 1500 advmss 1440
fec0::4400/119 dev dummy0  metric 1  mtu 1500 advmss 1440
fec0::4600/119 via fe80::3:2 dev eth2  proto zebra  metric 1024  mtu 1500 advmss 1440
fec0::4800/119 via fe80::3:2 dev eth2  proto zebra  metric 1024  mtu 1500 advmss 1440
fec0::4a00/119 via fe80::3:2 dev eth2  proto zebra  metric 1024  mtu 1500 advmss 1440
fec0::4c00/119 via fe80::3:2 dev eth2  proto zebra  metric 1024  mtu 1500 advmss 1440
```

5.5.3 Comportamiento teórico

A pesar de haber cambiado la red física, la lista de APs sigue siendo la misma en cada router con lo que las interfaces dummy0 deben reenviar paquetes de paging de la misma forma que en el apartado anterior.

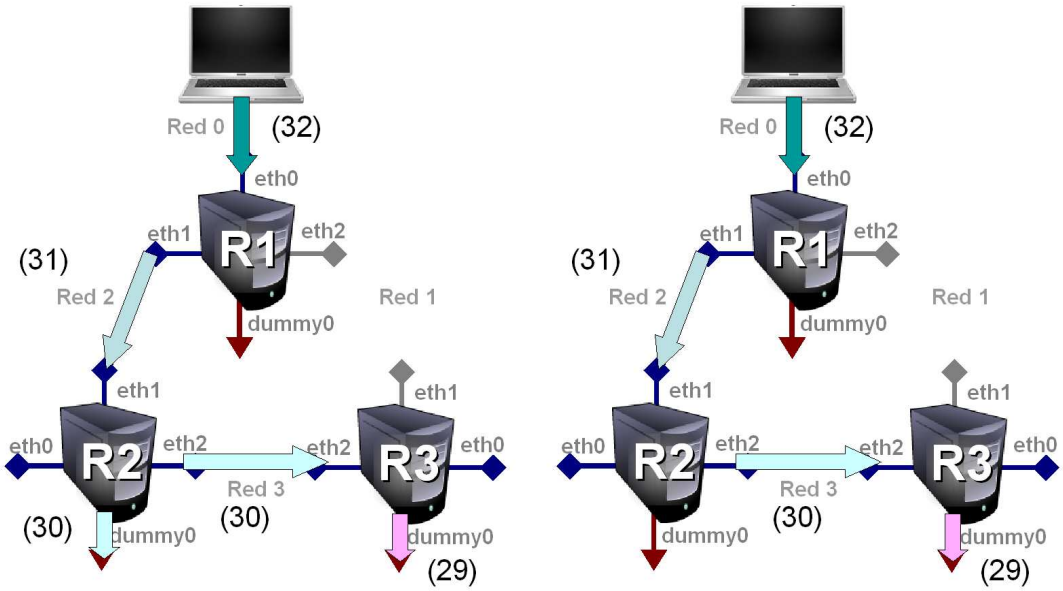

Figura 5.7: Comportamiento Prueba 2 con D=34 y D=20

5.5.4 Resultado

Las pruebas no se realizaron inmediatamente después de romper el enlace de la red 1. Como en el escenario se configuraron actualizaciones RIPng cada 5 segundos, timeout cada 10 y el tiempo de recolección de rutas inválidas (garbage collection) es 6 segundos. Así pues, de media el protocolo unicast tardaba unos 16 segundos en detectar correctamente la caída del enlace. Pasados ese tiempo, cuando las rutas habían convergido, se realizó la prueba.

El comportamiento fue el esperado, el visto en la figura 5.7. Con D=34 los tres routers reenvían el paging por sus interfaces virtuales dummy0. Con D=20, sólo el R3 debe hacerlo así que el resto de routers hacen lo necesario para entregarle el paquete. Y como en ambos casos hay un salto más en el camino entre el PA y el R3, el paquete de paging entregado a los APs es de 29, en vez de 30 como en la prueba 1.

5.6 Prueba 3

5.6.1 Descripción

Es una prueba similar a la anterior, solo que ahora el enlace desactivado es el de la red 2.

5.6.2 Esquema

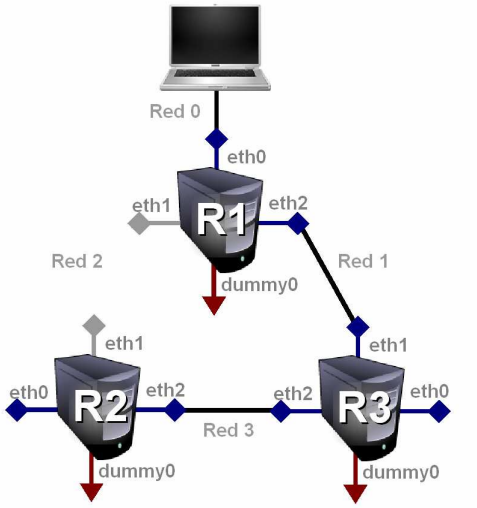

Figura 5.8: Prueba 3

5.6.3 Comportamiento teórico

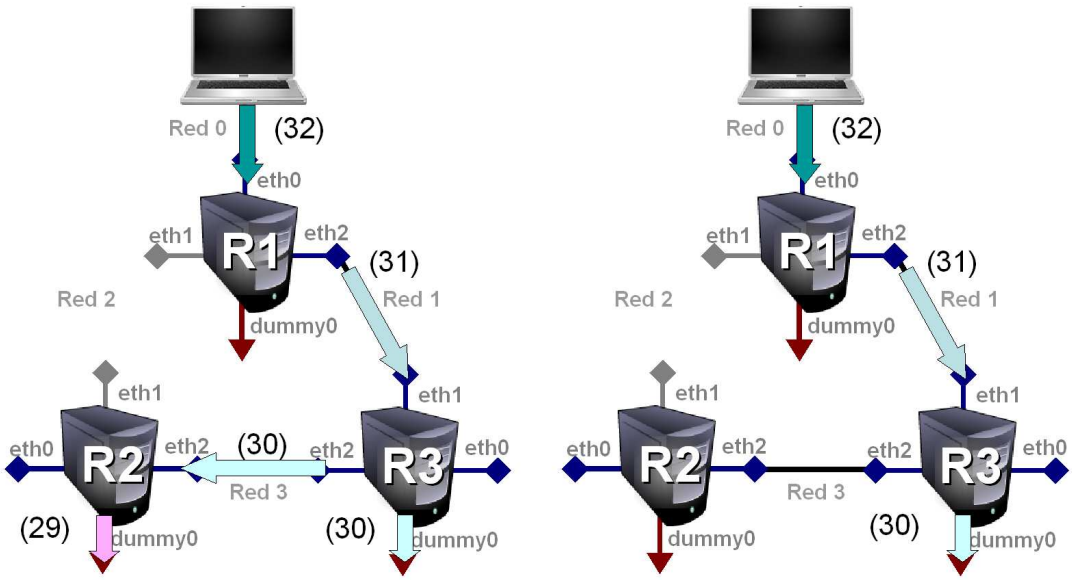

Figura 5.9: Comportamiento Prueba 3 con D=34 y D=20

5.6.4 Resultado

El resultado coincide con las predicciones hechas en la figura 5.9.

5.7 Prueba 4

5.7.1 Descripción

Esta prueba es prácticamente igual a la primera, salvo que quitamos el enlace de la red 3. El objetivo es ver si hay algún fallo en el mecanismo de reenvío que no se haya detectado aún.

5.7.2 Esquema

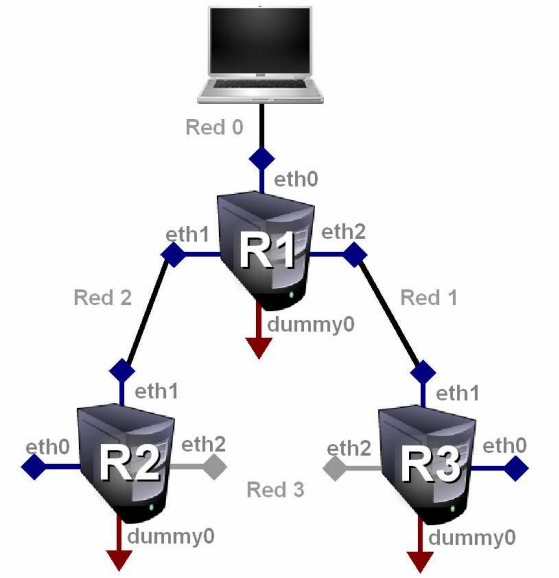

Figura 5.10: Prueba 4

5.7.3 Comportamiento teórico

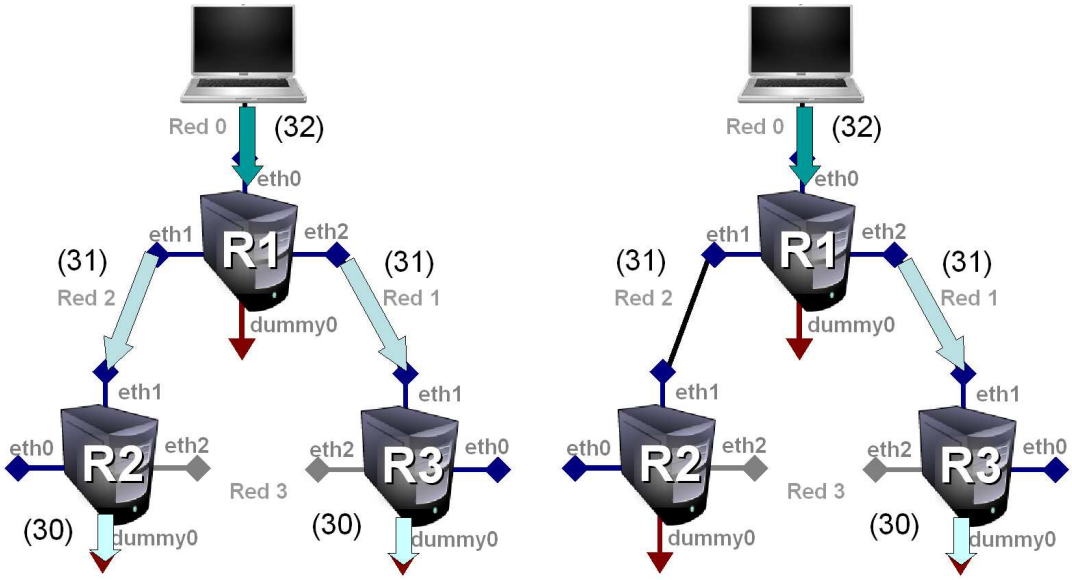

Figura 5.9: Comportamiento Prueba 4 con D=34 y D=20

5.7.4 Resultado

El programa función tal y como se esperaba.

5.8 Prueba 5

5.8.1 Descripción

Comprobaremos cómo se comporta el algoritmo de paging dinámico basado en distancia cuando trabajamos en el límite especificado. Es decir, añadiremos una ruta cuya distancia sea 2, y haremos tres pagings con distancias 1, 2 y 3.

5.8.2 Esquema

Este escenario sólo consta del Portátil y el Router 1, con la misma configuración anterior salvo una ruta añadida en R1:

```
fec0::1a32 dev dummy0  metric 1024  mtu 1500 advmss 1440
```

5.8.3 Comportamiento teórico

El paging con distancia 1 no debería hacer que el R1 propagase el paquete pues con una distancia de 1 la ruta añadida no cumple la fórmula (3.1) pues está a una distancia 2 del destino de paging FEC0::1A34.

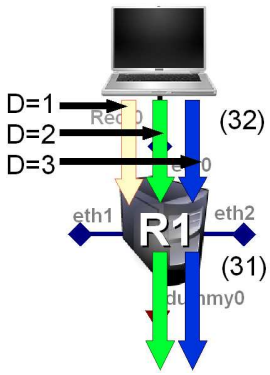

Figura 5.10: Comportamiento Prueba 5 con D=1 ,2 y 3

5.8.4 Resultado

El protocolo funciona tal y como se describe en la figura 5.10. Hay que resaltar que el protocolo reenvía el paging en caso de encontrar un AP con una distancia igual a la deseada. Por lo tanto, reenvía cuando la distancia calculada es mayor o iguala a la solicitada, tal y como se especificaba en la fórmula 3.1.

5.9 Conclusion

Todas las pruebas han demostrado que el programa funciona correctamente en todos los escenarios derivados de la maqueta. Se probaron también otras configuraciones, como por ejemplo situando el PA en otra red diferente a la 0, y se observó que también funcionaba.

CONCLUSIONES

Este TFC tenía como principal propósito implementar el protocolo GeoPaging. Para poder entender este protocolo se necesita una base teórica sobre IPv6 y sobre los algoritmos y protocolos existentes de multicast. También hay que entender el funcionamiento de las redes celulares y los mecanismos de localización utilizados. Así pues, antes de implementar GeoPaging se tuvo que estudiar estos conceptos, resumidos a lo largo de este trabajo.

Una vez adquirida la base teórica se estudió a fondo el protocolo de GeoPaging. De forma paralela se comenzó a preparar la maqueta de pruebas en el laboratorio mediante la instalación de tres ordenadores con Linux, múltiples interfaces de red con IPv6 y la configuración del protocolo de enrutado unicast RIPng. También se experimentó con las librerías analizadas en el anexo 3, lo cual sirvió para practicar la programación en C. Y se estudió a fondo muchas de las utilidades recogidas en el anexo 1, de manera que estudiando el código fuente se pudieron observar las maneras de trabajar contra el kernel de Linux. Un resumen de estas experiencias se encuentra en el anexo 4.

Entendidos el funcionamiento del protocolo y las posibilidades, o limitaciones, de las herramientas de que disponíamos (las librerías y interfaces de sistema), se pasó a diseñar la estructura del código. Desde un principio se siguieron criterios de programación que facilitaban la modularidad del código, permitiendo así una fácil ampliación de éste. La manera como se implementó el protocolo permite una gran flexibilidad a la hora de desplegar una red basada en GeoPaging, gracias al uso de tablas de enrutado unicast y a la activación del protocolo en todas las interfaces mediante el uso de contextos.

El proceso de realización de este TFC ha sido complejo, ya que los conceptos teóricos estudiados (IPv6, multicast, programación en Linux) son nuevos o no se estudian a fondo en la carrera. Además, en el mundo del software libre siempre hay varias soluciones ante un problema, y como ya se ha comentado tuvimos que experimentar con varias librerías de programación, llamadas a sistema, además de consultar código ajeno para aprender a realizar las tareas necesarias. Muchas veces esas horas de trabajo resultaron infructuosas debido a que sólo soportaban IPv4, o bien porque utilizaban recursos de programación demasiado complicados. La mala documentación existente para programar sobre IPv6 obligó a recurrir a la propia experimentación para comprobar si soportaban adecuadamente IPv6.

Una de las tareas más complicadas fue la realización de pruebas para comprobar si el código funcionaba o no. Como el código crecía paso a paso, se tenía que demostrar que funcionaba, pero realmente hasta que no se tuvo mucho código no se pudo testear. Esto introdujo muchos errores en el código, de estructuración y de modularidad. Las pruebas que se realizaban mostraban dónde estaban los errores, así que finalmente el código quedó bien organizado, comentado y separado en módulos funcionales. Para poder realizar estas pruebas se crearon una serie de herramientas (traducción de IDs de celda a direcciones IPv6, inyección paquetes de paging…) que aunque totalmente funcionales son poco flexibles, por ejemplo no admiten parámetros, ya que no hubo tiempo para hacerlas más fáciles de usar.

Otras complicaciones fueron debidas al hardware utilizado en la maqueta. En concreto, algunas de las interfaces de red más antiguas estaban estropeadas dejando de funcionar sin motivo alguno. Costó mucho identificar la fuente del problema que aparentemente era

causado por un mal funcionamiento del programa que realizaba el enrutado unicast, y se tardó semanas en solucionar. Posteriormente al realizar las pruebas finales se descubrió que para reactivar las tarjetas problemáticas había que ponerlas en modo promiscuo mediante el uso de tcpdump (ver anexo 1).

El resultado final ha sido totalmente satisfactorio.

Las pruebas mostradas en el capítulo 5 demuestran cómo el código desarrollado se comporta tal como especifica el protocolo Geopaging, cumpliendo así el objetivo del TFC. Aunque en este trabajo sólo se muestran estas pruebas, otras realizadas con direccionamientos y topologías diferentes obtuvieron también una resultado positivo mostrando la flexibilidad del código realizado.

También se ha cumplido el objetivo de dejar una base firme para futuros trabajos en los que se mejore el código añadiendo funcionalidades hasta crear una plataforma completa de paging IP. El código en sí tiene algunas limitaciones que se han tenido en cuenta en el anexo 5 que impiden el uso del actual código como enrutador de una red real, con miles de AP, pero se ha intentado dejar constancia de cómo arreglarlo. En este sentido, la documentación incluida en los anexos de las librerías estudiadas debería ser suficiente para que otro estudiante pudiese entender el código realizado..

En un futuro se podría mejorar el código en aspectos concretos como la ya comentada limitación de capacidad de enrutado, o dotar de más flexibilidad en cuanto al tráfico que enruta, pues de momento sólo se permiten paquetes de paging con la primera Extension Header del tipo Hop by Hop, que contiene la distancia de paging. Si la primera Extensión Header no es de ese tipo, no se considera un paquete de paging.

Otro objetivo de futuros TFC podría ser buscar mejores implementaciones de IPv6 sobre las que utilizar Geopaging. FreeBSD podría ser una buena opción: a parte de disponer de una mejor documentación y un mayor tiempo de desarrollo, tienen más proyectos basados en IPv6 en comparación con Linux. Como esta opción se valoró desde el primer momento, se programó modularmente para que resultase fácil migrar el código a esa plataforma.

Y finalmente, sería necesario diseñar y realizar una serie de pruebas que permitiesen determinar la eficiencia de ésta y de futuras implementaciones de Geopaging en términos como el consumo de CPU, memoria o tiempo de procesado de un paquete, para poder realizar comparaciones objetivas con otras implementaciones de otros protocolos.

Bibliografía y enlaces web

REFERENCIAS

[1]: 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.

[2] 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.

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

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

ANEXOS

INDICE DE ANEXOS

ANEXO I
IPv6 y Multicast en Linux

I.1 Introducción

Linux es un sistema operativo gratuito y de código abierto. Técnicamente hablando, Linux se sólo un kernel, es decir, un programa que inicializa el hardware del PC y proporciona acceso a él, además de implementar funciones de comunicaciones (protocolo IP por ejemplo). El sistema operativo se llama GNU/Linux pues incorpora el entorno GNU para trabajar, formado por los paquetes gcc, glibc, binutils, coreutils, bash, etc. La versión más reciente del kernel de Linux es la serie 2.6. Al inicio de este TFC estaba disponible la versión 2.6.3 pero al montar la maqueta ya se pudo trabajar con el 2.6.4.

Entre sus características se encuentra un gran soporte de protocolos de red. Entre ellos, el IP multicast. También soporta IPv6, aunque aún faltan características por implementar (debido a que cada día se publican nuevos RFC). En este anexo se explica qué hay que configurar para tener soporte de IPv6 y Multicast en el kernel de Linux. También se comentan las herramientas disponibles en GNU/Linux que soportan IPv6 y Multicast.

I.2 Soporte en el kernel

Linux es un kernel que se desarrolla por ramas, identificadas por un *major version number* y un *minor version number*. En www.kernel.org se almacenan las 4 ramas de Linux tal y como lo conocemos, es decir, del kernel con el 2 como *major version number*: 2.0, 2.2, 2.4 y 2.6. Históricamente hablando, la evolución de una rama a otra pasa por el desarrollo de una rama inestable, de número impar (2.1, 2.3, 2.5), cuyo objetivo es añadir prestaciones al kernel, hasta que se congela el código, se pasa a la siguiente rama con número par y entonces se centran esfuerzos en depurar y corregir errores.

Al inicio de este proyecto se usó Linux-2.6.3 “vainilla”, sin parches tal y como lo publican en www.kernel.org, usando como entorno la distribución *Gentoo Linux.* El soporte para IPv6 existe incluso desde la rama 2.2, pero como el propio protocolo ha cambiado con el tiempo, usamos la rama 2.6 para tener el mejor soporte posible. Sin embargo, un proyecto de investigación llamado USAGI (http://www.linux-ipv6.org) se dedica a mejorar la implementación IPv6 centrándose en que cumpla todos los estándares actuales e incluso las propuestas (Internet-drafts). Los parches existentes en la web de USAGI eran para la rama 2.4 y para la 2.5 de desarrollo. Sin embargo pudimos comprobar que actualmente los desarrolladores de USAGI han sido aceptados en el desarrollo activo del kernel y sus contribuciones están siendo directamente integradas en el kernel 2.6.

En definitiva, podemos decir que el soporte de IPv6 para Linux está en una fase avanzada de su desarrollo, faltando algunos detalles relacionados con cabeceras de autenticación y parámetros de calidad de servicio. Sin embargo, el kernel oficial actualmente incorpora las mejoras que día a día el proyecto USAGI publica. En este enlace (http://www.linux-ipv6.org/tahitest/200401/summary-frame.html) se pueden observar las diferencias de funcionamiento de IPv6 entre el kernel 2.6.1 y el paquete de parches oficial de USAGI a fecha de Enero de 2004.

Aquí podemos ver las opciones disponibles en el kernel 2.6.3 relacionados con IPv6:

CONFIG_IPV6=m
CONFIG_IPV6_PRIVACY=y

CONFIG_INET6_AH=m
CONFIG_INET6_ESP=m
CONFIG_INET6_IPCOMP=m
CONFIG_IPV6_TUNNEL=m

Se activó como módulo por consejo del director del proyecto, ante la duda de si tendríamos que modificar partes del kernel para nuestras pruebas. El funcionamiento es el mismo tanto trabajando con módulos como trabajando con IPv6 nativo en el kernel.

También podemos ver las opciones que el kernel de Linux nos da sobre protocolos de multicast:

CONFIG_IP_MULTICAST=y
....
CONFIG_IP_MROUTE=y
CONFIG_IP_PIMSM_V1=y
CONFIG_IP_PIMSM_V2=y

Con IP_MULTICAST se consigue que esté disponible el protocolo IGMP, permitiendo así que aplicaciones de usuario puedan recibir tráfico multicast. En IPv6 no pasa esto, no es necesario ninguna opción similar pues MLD (el equivalente a IGMP) no es más que un tipo especial de mensaje ICMPv6. Con IP_MROUTE se activa el soporte para DVMRP y MOSPF (antes de esto es necesario activar el enrutado unicast). El protocolo PIM DM no está disponible (necesita que lo implemente una aplicación), pero PIMSM sí, en ambas versiones (v1 y v2).

Este soporte permite que una máquina Linux se conecte a la red **MBONE** (Multicast Backbone, http://www-itg.lbl.gov/mbone), que fue el primer intento de crear una estructura global para la transmisión de información multicast por Internet. Actualmente ya hay esfuerzos para crear algo equivalente, tanto para IPv6 unicast (**6BONE**, http://www.6bone.net) como para IPv6 multicast (**M6BONE**, http://www.m6bone.net). Para conectarse a ambos primero hay que activar las opciones de IPv6 del kernel, y posteriormente conseguir conexión a un proveedor de servicios que esté conectado a dichas redes. También se puede crear un túnel IPv6-sobre-IPv4, utilizando como destino del túnel un equipo directamente conectado a esas redes y atravesando así redes que sólo soportan IPv4.

Actualmente la mayoría de protocolos de enrutado multicast sólo están implementados sobre IPv4. En el momento de la redacción de este documento, no hay ningún protocolo de enrutado multicast portado a IPv6 sobre Linux, sólo para BSD y en estado experimental, gracias al proyecto KAME (ftp://ftp.kame.net/pub/kame/misc) que han implementado PIMv2 como dos programas independientes: pim6dd y pim6sd. Estos protocolos son los utilizados actualmente en la red M6BONE.

Así pues, como conclusión, podemos predecir que el desarrollo de un protocolo de enrutado multicast de IPv6 sobre Linux será problemático debido a la novedad que ello implica y a que no tenemos modelos de referencia válidos, debido a que el software existente está para BSD. Linux no es compatible con BSD al 100%, y los mecanismos de comunicación con el kernel son diferentes.

I.2 Soporte en el software

Después de estudiar IPv6 y Multicast se realizó una búsqueda de proyectos relacionados con esas dos tecnologías, preferiblemente sobre la plataforma Linux o en su defecto BSD. También se incluyen algunos comandos típicos del entorno GNU/Linux, disponibles de serie en cualquier distribución.

Zebra

Página web: http://www.zebra.org
Versión utilizada: 0.93b

Se trata de un programa que permite utilizar una gran variedad de protocolos de enrutado unicast, como RIP, OSPF o BGP, y también para IPv6 (RIPng y OSPF6). Tiene varios años de desarrollo y es estable, aunque los protocolos de enrutado de IPv6 no disponen aún de todas las funcionalidades que tienen sus equivalentes en IPv4.

Quagga

Página web: http://www.quagga.net
Versión utilizada: 0.96.5

Es un fork de Zebra con una mayor comunidad de desarrollo. Por ejemplo ya han implementado el protocolo de enrutado IS-IS. Tiene las mismas prestaciones que Zebra, y los mismos inconvenientes. La versión probada no funcionaba bien: el protocolo RIPng se comportaba de una forma extraña y la interfaz de configuración (vtysh) no iba.

Iproute2

Página web: http://developer.osdl.org/dev/iproute2.
Versión utilizada: 2.4.7.20020116

Este programa es muy conocido entre los administradores de red. Permite configurar las interfaces de red, las rutas, las direcciones IP, etc, todo ello usando el mismo comando, “ip”. Por ejemplo, “ip route” sirve para operaciones relacionadas con la tabla de encaminamiento del sistema. Tiene un soporte parcial de IPv6 (comando “ip –f inet6”). Más información en http://www.linuxgrill.com/iproute2-toc.html.

Mrouted

Página web: http://freshmeat.net/projects/mrouted/
Versión utilizada: 3.9_beta3

Es un programa que se ejecuta en modo demonio que implementa el protocolo de enrutado multicast DVMRP, pudiendo así unirse a la red MBONE. Es un programa algo anticuado, con soporte nativo en el kernel de linux. Sin embargo solo soporta IPv4. Más información en http://www.jukie.net/~bart/multicast/Linux-Mrouted-MiniHOWTO.html.

Iputils

Página web: ftp://ftp.inr.ac.ru/ip-routing
Versión utilizada: 021109-r3

Este programa recoge las herramientas básicas para hacer pruebas en una red: ping, ping6, traceroute, traceroute6 y tracepath. Sólo las herramientas acabadas en 6 soportan IPv6, ya que sirven para generar mensajes ICMPv6 para probar la conectividad entre equipos.

Net-tools

Página web: http://sites.inka.de/sites/lina/linux/NetTools/
Versión utilizada: 1.60

Es un conjunto de herramientas relacionadas con la configuración de red en GNU/Linux, y vienen por defecto en la mayoría de distribuciones. Las aplicaciones proporcionadas son: arp, hostname (domainname, dnsdomainname, nisdomainname), ifconfig, ipmaddr, iptunnel, netstat, rarp, route, plipconfig, slattach y mii-tool.

Se probaron las herramientas arp, hostname, ifconfig, netstat y route, y vimos que todas ellas soportan IPv6 a la perfección. El autor ha creado una página (http://sites.inka.de/lina/linux/ipv6.html) donde se puede obtener más información sobre cómo usar sus herramientas para conectarse a 6BONE.

Glibc (o libc6)

Página web: http://www.gnu.org/software/libc/libc.html
Versión utilizada: 2.3.2 (usando las cabeceras del kernel 2.4.21)

La librería estándar de C, versión GNU. Proporciona la librería <sys/socket.h>, entre otras (todas las librerías <netinet/*.h>), las cuales soportan a la perfección IPv6. Glibc es por así decirlo la interfaz entre el código fuente del usuario y el kernel, quien realiza todo el trabajo relacionado con el uso de IPv6.

SendIP

Página web: http://www.earth.li/projectpurple/progs/sendip.html
Versión utilizada: 2.5

Utilidad muy flexible, utilizada en el TFC, para inyectar todo tipo de paquetes IPv6 o IPv4. Cada campo de la cabecera es personalizable.

Ethereal

Página web: http://www.ethereal.com
Versión utilizada: 0.10.4

Excelente analizador de redes, muy conocido gracias a que es compatible con la mayoría de sistemas operativos. Como está basado en libpcap, que es una librería que captura paquetes a bajo nivel, maneja sin problemas los paquetes IPv6.

Tcpdump

Página web: http://www.tcpdump.org/
Versión utilizada: 3.8.3

El programa oficial de la librería libpcap. Sirve para capturar cualquier tipo de paquete, y lo interpreta por pantalla sacando los datos más importantes según el protocolo del paquete. Por ejemplo, como IPv6 está soportado a la perfección, al capturar un paquete sin ninguna cabecera de nivel 4 muestra las direcciones origen y destino y las Extension Headers (si las hay).

ANEXO II
Descripción de la maqueta

II.1 Introducción

Este anexo explica el montaje de la maqueta donde se ha probado el protocolo multicast sobre una red IPv6. Se trata con detalle el hardware usado así como su distribución en la maqueta. La configuración del software de la maqueta también se incluye para una mejor compresión del funcionamiento del escenario.

II.2 Especificación de la red

Tenemos los siguientes equipos en la maqueta:

1. 4 PCs, uno de ellos portátil.
2. 1 hub ethernet 10/100 de 8 puertos
3. 9 tarjetas de red

En la siguiente tabla mostramos las características de cada equipo, tanto hardware como software:

Tabla II.1 Lista de hardware

Nombre equipo	Procesador (Mhz)	RAM (MB)	S.O.	Version de Kernel	Nº de NICs
Router A	P4 1700	256	Gentoo Linux 2004	Vainilla 2.6.4	4
Router B	P2 200	64	Gentoo Linux 2004	Vainilla 2.6.4	3
Router C	P2 233	96	Gentoo Linux 2004	Vainilla 2.6.4	3
Portátil	P3 1133	256	Gentoo Linux 2004	Vainilla 2.6.4	1

El router A tiene una interfaz deshabilitada. Los enlaces que conectaban con el Router C no funcionaban a 100Mbps pues éste disponía de tarjetas antiguas sólo funcionales a 10Mbps.

II.3 Configuración de los equipos

Esta parte muestra la red creada, especificando las diferentes subredes creadas y sus esquemas de direccionamiento utilizado.

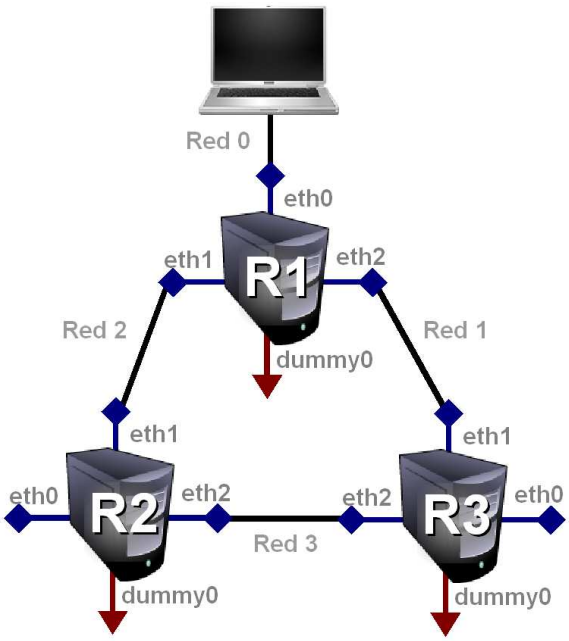

Figura II.1 Escenario de pruebas

En la siguiente tabla se especifica las direcciones IP de cada interfaz.

Tabla II.2: Configuración de red en cada equipo

ID de equipo	Interfaz	IPs configuradas
R1	eth0	FEDE::A:1 /112
		FE80::A:1 /112
	eth1	FEDE::1:1 /112
		FE80::1:1 /112
	eth2	FEDE::2:1 /112
		FE80::2:1 /112
	dummy0	FEDE::7:1 /112
ID de equipo	**Interfaz**	**IPs configuradas**
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
ID de equipo	**Interfaz**	**IPs configuradas**
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
ID de equipo	**Interfaz**	**IPs configuradas**
Portátil (PA)	eth0	FEDE::A:E /112

Las direcciones unicast virtuales de los APs, son del rango FEC0::/64. Se pueden añadir de dos formas: utilizando el software de enrutado Zebra, cuya configuración se muestra en el siguiente apartado, o bien usando el comando “route”. Para mayor comodidad, se utilizó un script en cada router que añadía las rutas. A continuación se muestran los scripts de cada router:

configurar_router1.sh

```
ifconfig dummy0 up
ifconfig dummy0 add fede::7:1/112
route -A inet6 add FEC0::4800/119 dev dummy0
route -A inet6 add FEC0::4C00/119 dev dummy0
```

configurar_router2.sh

```
ifconfig dummy0 up
ifconfig dummy0 add fede::8:2/112
route -A inet6 add FEC0::4600/119 dev dummy0
route -A inet6 add FEC0::4A00/119 dev dummy0
```

configurar_router3.sh

```
ifconfig dummy0  up
ifconfig dummy0  add fede::9:3/112
route -A inet6 add FEC0::4000/119 dev dummy0
route -A inet6 add FEC0::4200/119 dev dummy0
route -A inet6 add FEC0::4400/119 dev dummy0
```

Hay que recordar que para hacer ejecutable un archivo de texto (un script) hay que ejecutar el comando “chmod +x” y el nombre del archivo a continuación.

Como se ve en los scripts anteriores, se configura la interfaz dummy0 con una dirección IPv6 y se añaden rutas a las direcciones de los AP a través de esa interfaz. Dummy es el nombre que el kernel de Linux le da a la interfaz virtual. Su soporte se activa con la opción CONFIG_DUMMY=Y del kernel. Su funcionamiento se puede entender como una manera cómoda de configurar IPs a la interfaz de localhost. El tráfico enviado a Dummy0 no se transmite por ninguna interfaz real. En este escenario, Dummy0 representa la interfaz que comunica directamente con los AP’s de un dominio, donde normalmente se utilizan tecnologías de nivel 2 como ethernet, frame relay o similares.

II.4 Opciones del software

Como ya se ha comentado, se utilizó Linux 2.6 para la maqueta. Así pues, utilizamos la implementación de IPv6 que viene de serie en el kernel 2.6.4, pero lo cargamos como módulo. De esta manera es más fácil identificar posibles fallos en el sistema. No se configuró nada especial en el kernel, activamos todas las opciones de IPv6 como módulos por si su uso se hacia necesario en el futuro.

En los equipos de pruebas no utilizamos los scripts de inicio para la red que ofrece la distribución de Linux utilizada (Gentoo), sino que tuvimos que utilizar las opciones del programa Zebra para ello. Así pues, este es el contenido del archivo /etc/zebra/zebra.conf para el R1.

Tabla II.3: Configuración de Zebra

```
## cat /etc/zebra/zebra.conf
log file /var/log/zebra/zebra.log
interface lo
!
interface eth0
 ipv6 address fede::A:1/112
 ipv6 address fe80::A:1/112
 ipv6 nd suppress-ra
!
interface eth1
 ipv6 address fede::2:1/112
 ipv6 address fe80::2:1/112
 ipv6 nd suppress-ra
!
interface eth2
 ipv6 address fede::1:1/112
 ipv6 address fe80::1:1/112
 ipv6 nd suppress-ra
!
interface eth3
 ipv6 nd suppress-ra
interface sit0
 ipv6 nd suppress-ra
¡
```

Como se puede ver, la interfaz 3 está sin configurar, al igual que la interfaz “lo” (localhost) y “sti0” (para túneles IPv6 sobre IPv4). La configuración es prácticamente igual para cada router, cambiando las direcciones por las adecuadas (tabla II.1).

El demonio Zebra se encarga de configurar todas las interfaces adecuadamente, de manera que podemos entonces utilizar un protocolo de enrutamiento como el utilizado en la maqueta (RIPng). Su archivo de configuración, situado en /etc/zebra/ripngd.conf, es el siguiente:

Tabla II.4: Configuración de RIPng

```
## cat /etc/zebra/ripngd.conf
hostname ripngd
password zebra
log file /var/log/zebra/ripngd.log
no banner motd
router ripng
 network eth0
 network eth1
 network eth2
 network dummy0
 timers basic 5 10 6
 redistribute connected
 redistribute kernel
 redistribute static
!
```

Lo más relevante de la configuración de RIPng es el comando timers, cuya sintaxis es (*timers basic* update timeout garbage). Por defecto los tiempos son 30 180 120 pero se cambiaron para reducir el tiempo de convergencia del escenario ante cambios, como la ruptura de enlaces que se realizaba en las pruebas.

Para poder activar este demonio de enrutamiento, basta con ejecutar el script de inicio del RIPng, activándose también el demonio del Zebra.

```
## /etc/init.d/ripngd start
* Starting zebra…                                                    [ ok ]
* Starting ripngd…                                                   [ ok ]
```

Para ver las rutas que establece el demonio zebra junto con RIPng, ejecutamos el comando "*ip –f inet6 route"* en el R1, en vez de *route,* por su mayor sencillez y utilidad.

Tabla II.5: Tabla enrutado R1

```
## ip -f inet6 route
fe80::1:0/112 dev eth2  metric 256  mtu 1500 advmss 1440
fe80::2:0/112 dev eth1  metric 256  mtu 1500 advmss 1440
fe80::a:0/112 dev eth0  metric 256  mtu 1500 advmss 1440
fe80::/64 dev eth0  metric 256  mtu 1500 advmss 1440
fe80::/64 dev eth1  metric 256  mtu 1500 advmss 1440
fe80::/64 dev eth2  metric 256  mtu 1500 advmss 1440
fe80::/64 dev dummy0  metric 256  mtu 1500 advmss 1440
fe80::/64 dev eth3  metric 256  mtu 1500 advmss 1440
fec0::4000/119 via fe80::1:3 dev eth2  proto zebra  metric 1024  mtu 1500 advmss 1440
fec0::4200/119 via fe80::1:3 dev eth2  proto zebra  metric 1024  mtu 1500 advmss 1440
fec0::4400/119 via fe80::1:3 dev eth2  proto zebra  metric 1024  mtu 1500 advmss 1440
fec0::4600/119 via fe80::2:2 dev eth1  proto zebra  metric 1024  mtu 1500 advmss 1440
fec0::4800/119 dev dummy0  metric 1  mtu 1500 advmss 1440
fec0::4a00/119 via fe80::2:2 dev eth1  proto zebra  metric 1024  mtu 1500 advmss 1440
fec0::4c00/119 dev dummy0  metric 1  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::1:3 dev eth2  proto zebra  metric 1024  mtu 1500 advmss 1440
fede::7:0/112 dev dummy0  metric 256  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
ff00::/8 dev eth0  metric 256  mtu 1500 advmss 1440
ff00::/8 dev eth1  metric 256  mtu 1500 advmss 1440
ff00::/8 dev eth2  metric 256  mtu 1500 advmss 1440
ff00::/8 dev dummy0  metric 256  mtu 1500 advmss 1440
ff00::/8 dev eth3  metric 256  mtu 1500 advmss 1440
unreachable default dev lo  proto none  metric -1  error -101
```

Vemos que la rutas se aprenden mediante el programa Zebra (ver donde pone "proto zebra"). En los demás routers las rutas son muy parecidas. A continuación mostramos un extracto.

Tabla II.6: Tabla enrutado R2

```
## ip -f inet6 route
fec0::4000/119 via fe80::3:3 dev eth2  proto zebra  metric 1024  mtu 1500 advmss 1440
fec0::4200/119 via fe80::3:3 dev eth2  proto zebra  metric 1024  mtu 1500 advmss 1440
fec0::4400/119 via fe80::3:3 dev eth2  proto zebra  metric 1024  mtu 1500 advmss 1440
fec0::4600/119 dev dummy0  metric 1  mtu 1500 advmss 1440
fec0::4800/119 via fe80::2:1 dev eth1  proto zebra  metric 1024  mtu 1500 advmss 1440
fec0::4a00/119 dev dummy0  metric 1  mtu 1500 advmss 1440
fec0::4c00/119 via fe80::2:1 dev eth1  proto zebra  metric 1024  mtu 1500 advmss 1440
fede::1:0/112 via fe80::3:3 dev eth2  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::7:0/112 via fe80::2:1 dev eth1  proto zebra  metric 1024  mtu 1500 advmss 1440
fede::8:0/112 dev dummy0  metric 256  mtu 1500 advmss 1440
fede::9:0/112 via fe80::3:3 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::3:3 dev eth2  proto zebra  metric 1024  mtu 1500 advmss 1440
```

Tabla II.7: Tabla enrutado R3

```
## ip -f inet6 route
fec0::4000/119 dev dummy0  metric 1  mtu 1500 advmss 1440
fec0::4200/119 dev dummy0  metric 1  mtu 1500 advmss 1440
fec0::4400/119 dev dummy0  metric 1  mtu 1500 advmss 1440
fec0::4600/119 via fe80::3:2 dev eth2  proto zebra  metric 1024  mtu 1500 advmss 1440
fec0::4800/119 via fe80::1:1 dev eth1  proto zebra  metric 1024  mtu 1500 advmss 1440
fec0::4a00/119 via fe80::3:2 dev eth2  proto zebra  metric 1024  mtu 1500 advmss 1440
fec0::4c00/119 via fe80::1:1 dev eth1  proto zebra  metric 1024  mtu 1500 advmss 1440
fede::1:0/112 dev eth1  metric 256  mtu 1500 advmss 1440
fede::2:0/112 via fe80::1:1 dev eth1  proto zebra  metric 1024  mtu 1500 advmss 1440
fede::3:0/112 dev eth2  metric 256  mtu 1500 advmss 1440
fede::7:0/112 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
```

II.5 Herramientas utilizadas

II.5.1 idtohex

Es un programa creado para obtener la ID de interfaz de un AP. La sintaxis es la siguiente:

```
./idtohex .0.3.4.1.2.3.6
```

Y su resultado es éste:

```
bVectorSize es 24
HexVectSize es 3
00000001  11000010  10011110
 : 01 : c29e
```

Los tres primeros son mensajes que sirven para entender el funcionamiento del código conversor de ID a IPv6. Al tener un ID de 7 niveles (.0.3.4.1.2.3.6), necesitamos 7*3 bits, pero 21 bits no se pueden reservar en memoria así que reservamos el múltiplo de 8 más cercano (24 bits, 3 bytes). Estos bytes se representan en binario (*00000001 11000010 10011110*), y se muestra el ID de interfaz de la dirección IPv6 (*: 01 : c29e*).

Así pues, si usamos el rango FEC0::/64 para las direcciones unicast virtuales de los APs, éste AP (.0.3.4.1.2.3.6) tendrá la IP FEC0::01:C29E

II.5.2 GEOPAG_tester

Este programa es una manera de acceder a las funciones GEOPAG_ del código fuente del protocolo GeoPaging. Permite ver la distancia que hay entre dos APs, dándole por parámetro sus direcciones IPv6. Su sintaxis es esta:

```
./GEOPAG_tester FEC0::1A34 FEC0::3425
```

Y su resultado es

```
3, 8, 19, 50, 130, 344, 908, 2402, 6353, 16808, 44468, 117649, 311270, 823543,
2178890, 5764801, 15252229, 40353605, 106765601, 282475233,
Usamos 21 niveles
||Matriz diferencia|| (0,0) (0,0) (1,0) (-1,-1) (0,0) (-1,0) (1,-1) Dv es (29,-
20)
Distancia 49
```

Así pues, la distancia entre las celdas con una ID de interfaz ::1A34 y ::3425 es 49 celdas. Los mensajes que se ven sirven para entender los pasos realizados en el cálculo. Para más información, ver la referencia [3]. La primera línea son las 21 distancias calculadas para clústeres de diferentes niveles (en orden, nivel 2, 3, 4, etc), hasta un máximo de 21 Niveles. Después se muestra la Matriz Diferencia, y posteriormente el vector Distancia, donde sus coordenadas ya están en la misma unidad que Distancia, es decir, la unidad es la celda que controla el AP de más bajo nivel (1).

II.5.3 Sendip

Este programa se utiliza para inyectar los paquetes de paging en la red. Es necesario que la tabla de encaminamiento esté configurada para enrutar los paquetes multicast por la interfaz deseada. En el caso del portátil no hace falta porque sólo hay una interfaz. Pero si queremos especificarla se puede ejecutar "*route add ff00/8 dev ethX metric 1*", donde *ethX* es la interfaz deseada, y donde *metric 1* significa que esta ruta tiene mayor prioridad que cualquier otra ruta similar.

```
./sendip -p ipv6 -6s fede::a:e -d \
0xbb025b506167696e675f44697374616e63655d3d33340000616263646566616263646566303132
31 -6n 0 GEOPAG
```

Los parámetros significan, en orden, el uso de IPv6 justo después de Ethernet, donde la IP de origen (-6s) es FEDE::A:E, y cuyo payload serán la cadena hexadecimal mostrada. También se indica el valor de la next header (-6n), que es 0 y apunta al payload indicando que se interprete como una HopByHop Header. Finalmente, GEOPAG es una entrada en el fichero */etc/hosts* que indica la dirección multicast destino del paging (FF05::1A34).

Dentro del payload es interesante resaltar que se indica una NextHeader con valor 0xBB, que es un valor sin asignar por la IANA. Los datos que hay a continuación de la HopByHop es una cadena de texto arbitrario, "abcdefabcdef0121". También es importante resaltar que se envía la HobByHop con datos en formato ASCII, es decir, "[Paging_Distance]=34". Si se quiere enviar "[Paging_Distance]=20", para hacer las pruebas del capítulo 5, hay que ejecutar el comando así:

```
./sendip -p ipv6 -6s fede::a:e -d \
0xbb025b506167696e675f44697374616e63655d3d32300000616263646566616263646566303132
31 -6n 0 GEOPAG
```

Y si se quiere enviar otra distancia, sin modificar el campo Lenght de la HobByHop, se puede cambiar la zona resaltada de gris por cualquier otro número representado en ASCII. Por ejemplo, distancia 1 se representaría como 31000000, 2 como 32000000, etc.

Para entender mejor este procedimiento, en la siguiente figura mostramos la tabla ASCII. Es conveniente recordar que *sendip* recibe como parámetro números hexadecimales.

Dec	Hx	Oct	Char	Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr
0	0	000	NUL (null)	32	20	040		Space	64	40	100	@	@	96	60	140	`	`
1	1	001	SOH (start of heading)	33	21	041	!	!	65	41	101	A	A	97	61	141	a	a
2	2	002	STX (start of text)	34	22	042	"	"	66	42	102	B	B	98	62	142	b	b
3	3	003	ETX (end of text)	35	23	043	#	#	67	43	103	C	C	99	63	143	c	c
4	4	004	EOT (end of transmission)	36	24	044	$	$	68	44	104	D	D	100	64	144	d	d
5	5	005	ENQ (enquiry)	37	25	045	%	%	69	45	105	E	E	101	65	145	e	e
6	6	006	ACK (acknowledge)	38	26	046	&	&	70	46	106	F	F	102	66	146	f	f
7	7	007	BEL (bell)	39	27	047	'	'	71	47	107	G	G	103	67	147	g	g
8	8	010	BS (backspace)	40	28	050	(	(	72	48	110	H	H	104	68	150	h	h
9	9	011	TAB (horizontal tab)	41	29	051	)	)	73	49	111	I	I	105	69	151	i	i
10	A	012	LF (NL line feed, new line)	42	2A	052	*	*	74	4A	112	J	J	106	6A	152	j	j
11	B	013	VT (vertical tab)	43	2B	053	+	+	75	4B	113	K	K	107	6B	153	k	k
12	C	014	FF (NP form feed, new page)	44	2C	054	,	,	76	4C	114	L	L	108	6C	154	l	l
13	D	015	CR (carriage return)	45	2D	055	-	-	77	4D	115	M	M	109	6D	155	m	m
14	E	016	SO (shift out)	46	2E	056	.	.	78	4E	116	N	N	110	6E	156	n	n
15	F	017	SI (shift in)	47	2F	057	/	/	79	4F	117	O	O	111	6F	157	o	o
16	10	020	DLE (data link escape)	48	30	060	0	0	80	50	120	P	P	112	70	160	p	p
17	11	021	DC1 (device control 1)	49	31	061	1	1	81	51	121	Q	Q	113	71	161	q	q
18	12	022	DC2 (device control 2)	50	32	062	2	2	82	52	122	R	R	114	72	162	r	r
19	13	023	DC3 (device control 3)	51	33	063	3	3	83	53	123	S	S	115	73	163	s	s
20	14	024	DC4 (device control 4)	52	34	064	4	4	84	54	124	T	T	116	74	164	t	t
21	15	025	NAK (negative acknowledge)	53	35	065	5	5	85	55	125	U	U	117	75	165	u	u
22	16	026	SYN (synchronous idle)	54	36	066	6	6	86	56	126	V	V	118	76	166	v	v
23	17	027	ETB (end of trans. block)	55	37	067	7	7	87	57	127	W	W	119	77	167	w	w
24	18	030	CAN (cancel)	56	38	070	8	8	88	58	130	X	X	120	78	170	x	x
25	19	031	EM (end of medium)	57	39	071	9	9	89	59	131	Y	Y	121	79	171	y	y
26	1A	032	SUB (substitute)	58	3A	072	:	:	90	5A	132	Z	Z	122	7A	172	z	z
27	1B	033	ESC (escape)	59	3B	073	;	;	91	5B	133	[	[	123	7B	173	{	{
28	1C	034	FS (file separator)	60	3C	074	<	<	92	5C	134	\	\	124	7C	174			|
29	1D	035	GS (group separator)	61	3D	075	=	=	93	5D	135	]	]	125	7D	175	}	}
30	1E	036	RS (record separator)	62	3E	076	>	>	94	5E	136	^	^	126	7E	176	~	~
31	1F	037	US (unit separator)	63	3F	077	?	?	95	5F	137	_	_	127	7F	177	&#127;	DEL

Source: www.asciitable.com

Figura II.2: Tabla de caracteres ASCII

II.5.4 Mcast

Es el programa que se debe ejecutar en cada ordenador para poder enrutar los paquetes de paging. Su sintaxis es la siguiente:

```
./mcast <subred multicast> <subred unicast>
```

En este escenario se utilizaba un script (*MCAST.sh*) que ejecutaba este programa con los argumentos correctos, es decir, las direcciones usadas para la maqueta:

```
./mcast ff05:: fec0::
```

El funcionamiento de este programa se detalla en el capítulo 4 del TFC.

ANEXO III
Programación de red en Linux

III.1. Introducción

Linux, como todo sistema basado en UNIX, tiene un gran soporte para los programas basados en redes. Las APIs (Application Programming Interface) que ofrece Linux son muy variadas: desde las clásicas basadas en los sistemas BSD antiguos hasta las más modernas diseñadas para trabajar en cualquier SO, incluido Windows.

A lo largo de este TFC se han ido probando varias de las alternativas aquí presentadas, y se han ido descartando algunas según soportasen adecuadamente IPv6 y multicast. No todas cumplían este requisito, aunque está previsto que en un futuro todas lo soporten.

Este anexo se divide en dos partes, presentando primero las APIs que ofrece de serie el kernel de Linux, y luego mostrando algunas de las librerías existentes para trabajar con redes.

III.2 Librerías nativas

Las librerías nativas son aquellas en las que no se necesita ningún software externo para funcionar, a parte del conjunto de herramientas habituales en GNU/Linux como gcc, glibc y el resto de librerías del sistema.

III.2.1 BSD Sockets

Son los usados para crear conexiones de nivel 4, es decir, UDP o TCP. Son considerados de alto nivel pues no permiten modificar ningún campo de la cabecera ni usar direcciones origen diferentes a la propia (sniffing). Cualquier usuario sin privilegios puede utilizarlos.

La documentación disponible en libros y tutoriales es extensa. Más información en:
http://www.ecst.csuchico.edu/~chafey/prog/sockets/sinfo1.html
http://www.ecst.csuchico.edu/~beej/guide/net/html/

III.2.2 RAW Sockets

El uso de estos sockets requiere ser superusuario (root), ya que permiten el sniffing y la manipulación de cualquier campo de la cabecera IPv4 o IPv6. Para usarlos basta crear un socket del domino AF_INET o AF_INET6 con protocolo PF_RAW. El problema está en que IPv4 permitía la lectura de un paquete IPv4, la modificación de un parámetro y su reinyecció en muy pocos pasos y de forma sencilla, pero la existencia de Extensión Headers de IPv6 complica la cosa. Así que no se puede realizar esa operación con IPv6 tan fácilmente.

Más información:

Página del manual
“man 7 raw”

Excelente tutorial de RAW IP sockets
http://mixter.void.ru/rawip.html

III.2.3 Packet Sockets

Estos sockets son la interfaz de más bajo nivel existente en Linux. Reciben cualquier trama de nivel 2, incluso permitiendo el modo promiscuo recibiendo tráfico no destinado a la propia máquina. Por supuesto, requieren ser root para poder abrir un socket del tipo PF_PACKET.

Más información:

Página del manual
“man 7 packet”

Cómo funciona el Packet Filter de Linux
http://www.linuxjournal.com/article.php?sid=4852

III.2.4 Divert Sockets

Los divert sockets (o sistema Frame Diverter) son una potente herramienta para interceptar paquetes al más bajo nivel. Puede convertir un ordenador con dos tarjetas de red en un bridge ethernet. Los requerimientos de los divert sockets son algo complejos: necesitan reglas de firewall configuradas y tener activado en el kernel la opción CONFIG_NET_DIVERT. Al igual que los packet sockets puede capturer tramas de nivel 2 (también require ser root), pero a diferencia de éstos, impide que el subsistema de networking del kernel de Linux reciba esas tramas. Es decir, los sockets RAW y Packet sólo entregan una copia del paquete a la aplicación, pero el paquete se sigue procesando por el kernel. Los divert sockets evitan ese proceso, interceptan las tramas y se las entregan al usuario.

Se pueden usar de la misma forma que los raw sockets, pero siempre requieren una configuración en el firewall del sistema. Al crear un divert socket se debe registrar con una norma de firewall. Esto permite tener varios programas usando divert sockets de forma simultánea ya que será el firewall quien entregue el paquete a la apliación registrada.

Otra gran ventaja de los divert sockets es que puede reinyectar los paquetes como entrantes o como salientes, mientras que los raw sockets (y los packet) solo pueden hacerlo como salientes.

Además se facilita la manipulación de cualquier campo de la cabecera, incluso recalculando los checksums necesarios o los campos relacionados con la longitud.

Más información en

Excelente introducción a divert y comparativa con otros sistemas
http://www.icewalkers.com/Linux/Howto/Divert-Sockets-mini-HOWTO-6.html

III.3 Librerías externas

III.3.1 Libpcap

Una de las librerías de captura de paquetes más famosas. Permite configurar filtros para capturar exactamente el tipo de paquetes deseados especificando de el puerto TCP hasta la MAC origen del paquete. Internamente utiliza la interfaz packet comentada en el apartado anterior, tal y como recoge esta nota (http://www.tcpdump.org/libpcap-changes.txt)

```
Summary for 0.6 release

        New Linux libpcap implementation, which, in 2.2 and later
        kernels, uses PF_PACKET sockets and supports kernel packet
        filtering (if compiled into the kernel), and supports the "any"
        device for capturing on all interfaces.
```

Así pues, internamente libpcap utiliza una librería implementada en el kernel, con lo que se garantiza rapidez de ejecución. Libpcap no proporciona ningún metodo de inyección de paquetes a la red. Tampoco evita que el sistema procese los paquetes, como hacen los divert sockets.

Página del proyecto Tcpdump / libpcap
http://www.tcpdump.org (última version 0.8.3)

III.3.3 Libnet

Se trata de una librería de creación e inyección de paquetes que recientemente ha incorporado el soporte para IPv6. Sin embargo algunas funcionalidades de IPv6 están por desarrollar, pero aún así el soporte es muy bueno. Tiene una excelente documentación y una buena comunidad de desarrolladores.

Página del proyecto
http://www.packetfactory.net/Projects/Libnet/ (ultima versión 1.1.2-rc)

III.3.3 Gnetlib

Librería alternativa a Libnet, con mucho tiempo de desarrollo y excelente documentación. No se utilizó debido al pobre soporte de IPv6, pues no soporta la manipulación de Extension Headers, aunque actualmente están mejorándolo.

http://www.gnetlibrary.org/ (última versión 2.0.5)

III.3.4 Libnwrap

Reciente aparición en el mundo de la programación de red. Pretende ofrecer mecanismos de comunicaciones que sean portable. También pretende implementar desde un principio IPv6, además de funciones criptográficas. Actualmente está en fase de desarrollo, aunque paralizado desde hace tiempo.

Página del proyecto
http://libnwrap.sourceforge.net/ (última version 0.0.1-alpha)

ANEXO IV
Medios de obtener de información del kernel

IV.1 Introducción

A veces a la hora de programar aplicaciones de red en Linux no basta con poder comunicarse con un host remoto. Tampoco es suficiente poder recibir y enviar cualquier tipo de paquete. A veces se requiere información del entorno del sistema, como el número de interfaces activas, las conexiones de red establecidas o la tabla de encaminamiento.

A continuación se presentan tres métodos de obtener información del sistema. También se introduce un método que no es propio de Linux, sino que es propio de sistemas BSD, para que sirva de comparativa entre las dos plataformas. No se explican las llamadas a IOCTL, usadas para modificar parámetros de un conector (en este caso de un socket), pues solamente lo utilizamos para obtener la MAC de una interfaz.

IV.2 Proc

Se trata de una interfaz muy sencilla e intuitiva organizada en carpetas y ficheros de texto. El nombre completo del fichero completo, incluida la ruta de carpetas, sirve para saber qué buscamos. Por ejemplo /proc/net/ipv6_route nos muestra la tabla de encaminamiento del sistema. Para entender este fichero hay que buscar documentación al respecto, normalmente la página del manual (man proc) proporciona un buen punto de partida.

En algunos casos se explica la sintaxis seguida en los ficheros del proc, otras veces la sintaxis viene implícita. Por ejemplo, en /proc/net/arp tenemos lo siguiente:

```
IP address       HW type     Flags       HW address            Mask     Device
192.168.1.3      0x1         0x2         00:0C:6E:DA:ED:EE     *        eth0
192.168.1.1      0x1         0x2         00:04:76:A7:41:1B     *        eth0
```

Los datos anteriores son autodescriptivos. Pero si miramos en otro fichero, por ejemplo en /proc/net/dev_mcast no se puede saber a priori el significado, como se muestra a continuación:

```
2    eth0            1     0     01005e000001
```

También hay archivos que no están pensados para ser leídos por el usuario, como /proc/kcore, que es el fichero del kernel de linux (también conocido como vmlinux) y por lo tanto al abrirlo con un visor de textos no se entiende nada.

Finalmente hay que decir que el proc no sirve sólo para leer información sino que también podemos modificar parámetros del sistema. Por ejemplo si escribimos un 1 en cierto fichero (mediante "*echo 1 > /proc/sys/net/ipv6/conf/all/forwarding*"), el sistema activará las funciones de forwarding de paquetes, es decir, actuará de router.

Para más información:

Manual de proc (man proc)
http://www.die.net/doc/linux/man/man5/proc.5.html

Parámetros configurables via Proc
http://mirrors.bieringer.de/Linux+IPv6-HOWTO/chapter-kernel-settings.html.

IV.3 Sysctl

Se trata de un mecanismo utilizado para obtener y modificar información del sistema, o tal y como se define en la página del manual (man sysctl), sirve para configurar parámetros del kernel en tiempo de ejecución. Los parámetros se llaman MIB, y funcionan igual que las MIB del protocolo de gestión de red SNMP. Se organizan de forma jerárquica, y tienen dos representaciones: una en modo texto (por ejemplo kernel.ostype) y otra en modo numérico (1.1, o lo que es lo mismo, CTL_KERN.KERN_OSTYPE). La lista completa de MIBs está en /usr/include/linux/sysctl.h. Hay diez categorías de primer orden (extraído de sysctl.h):

```
CTL_KERN=1,             /* General kernel info and control */
CTL_VM=2,               /* VM management */
CTL_NET=3,              /* Networking */
CTL_PROC=4,             /* Process info */
CTL_FS=5,               /* Filesystems */
CTL_DEBUG=6,            /* Debugging */
CTL_DEV=7,              /* Devices */
CTL_BUS=8,              /* Busses */
CTL_ABI=9,              /* Binary emulation */
CTL_CPU=10              /* CPU stuff (speed scaling, etc) */
```

Un ejemplo de categorías de segundo es el del subsistema NET (networking):

```
/* /proc/sys/net/ipv6 */
enum {
        NET_IPV6_CONF=16,
        NET_IPV6_NEIGH=17,
        NET_IPV6_ROUTE=18,
        NET_IPV6_ICMP=19,
        NET_IPV6_BINDV6ONLY=20,
        NET_IPV6_IP6FRAG_HIGH_THRESH=21,
        NET_IPV6_IP6FRAG_LOW_THRESH=22,
        NET_IPV6_IP6FRAG_TIME=23,
        NET_IPV6_IP6FRAG_SECRET_INTERVAL=24,
        NET_IPV6_MLD_MAX_MSF=25,
};
```

Es interesante resaltar que en systcl.h además de definir el nombre y el número de MIB, se indica en qué lugar del sistema de ficheros PROC (es decir, en /proc/sys, que es donde están las MIBs de sysctl) se puede obtener la misma información. Por ejemplo si miramos dentro de la carpeta /proc/sys/net/ipv6 veremos ficheros con los nombres anteriores:

```
conf/       neigh/              route/              icmp/       bindv6only
ip6frag_high_thresh             ip6frag_low_thresh              ip6frag_time
ip6frag_secret_interval         mld_max_msf
```

Las carpetas del /proc equivalen a MIBs con niveles inferiores, por ejemplo *net.ipv6.route* tiene las siguientes MIBs definidas:

```
  NET_IPV6_ROUTE_FLUSH=1,
  NET_IPV6_ROUTE_GC_THRESH=2,
  NET_IPV6_ROUTE_MAX_SIZE=3,
  NET_IPV6_ROUTE_GC_MIN_INTERVAL=4,
```

```
NET_IPV6_ROUTE_GC_TIMEOUT=5,
NET_IPV6_ROUTE_GC_INTERVAL=6,
NET_IPV6_ROUTE_GC_ELASTICITY=7,
NET_IPV6_ROUTE_MTU_EXPIRES=8,
NET_IPV6_ROUTE_MIN_ADVMSS=9
```

Para acceder a estos datos se puede usar el comando *sysctl* desde consola directamente. Si ejecutamos "sysctl kernel.ostype" veremos que el resultado es "Linux". No se pueden usar MIBs en formato numérico. También es una alternativa a PROC a la hora de cambiar una MIB, por ejemplo "*echo 1 > /proc/sys/kernel/sysrq*" es equivalente a hacer "*sysctl –w kernel.sysrq="1"*".

Podemos ejecutar llamadas a sysctl desde nuestro código fuente. La función se llama igual, sysctl() (ver man 2 sysctl), pero a diferencia del comando de consola, aquí se deben usar MIBs numéricas, o bien usar los valores definidos en /usr/include/linux/sysctl.h como por ejemplo CTL_KERN o KERN_OSTYPE. La función devuelve 0 si la llamada se ha realizado correctamente, y sino se devuelve -1 y se detalla el error en *errno.* El resultado de la llamada se devuelve en una variable pasada por referencia como un string de caracteres.

Más información en

Proc y Sysctl
http://www.linuxjournal.com/article.php?sid=2365
Sysctl para IPv4
http://ipsysctl-tutorial.frozentux.net/ipsysctl-tutorial.html
Uso de sysctl
http://www.freebsd.org/doc/en_US.ISO8859-1/books/handbook/configtuning-sysctl.html

IV.4 Netlink

Netlink se diseñó como un protocolo de comunicaciones entre un *Forwarding Engine Component* (FEC) y un *Control Plane Component* (CPC), componentes que definen un servicio IP. Los servicios IP se entienden como aquellos componentes que permiten la operación del protocolo IP, como por ejemplo el forwarding[5], la adición o eliminación de rutas, la gestión, etc.

El concepto de separar el servicio IP en dos partes (control y forwarding) fue introducido en BSD 4.4 con los routing sockets. El objetivo era proporcionar un servicio de forwarding para IPv4 controlado por un CPC simple (p. ej. un programa en C). Netlink amplia esta visión y no se limita al mecanismo de forwarding de IPv4.

La interacción entre la capa de control y la de ejecución, es decir entre el CPC y el FEC, se realiza mediante el intercambio de mensajes. Esos mensajes están basados en plantillas y definen un protocolo. Éste actúa como un cable que comunica dos espacios de ejecución en el sistema operativo bien diferentes: espacio de kernel y espacio de usuario. El FEC reside en espacio de kernel pues es el entorno de ejecución, el que realiza las acciones de forwarding, y de eso se encarga el kernel. El CPC reside en espacio de usuario pues es él quien debe controlar el sistema, ya que el CPC no es mas que una

[5] El término "forwarding" se utiliza para referir al mecanismo de reenvío de paquetes que realiza un enrutador

herramienta para modificar y controlar el comportamiento del FEC. Así pues la interfaz netlink es un medio que permite comunicar el CPC con el FEC.

El diseño de Netlink permite que se puedan comunicar todo tipo de servicios, no solo relacionados con IP forwarding. La idea básica es ofrecer un entorno de comunicación entre el kernel y el usuario. Ofrece un cable dedicado a un servicio específico, no mezclando mensajes entre ellos. El cable permite que todos los CPCs registrados con un FEC reciban los paquetes en modo broadcast. Hay otras opciones de comunicación entre FEC y CPC, por ejemplo si ambos residen en espacio del kernel, que no se discuten aquí.

En general podemos decir que la arquitectura de Netlink permite un sistema sencillo para que cualquier software de usuario se comunique con el kernel. La interfaz ofrecida son las clásicas llamadas para establecer sockets, pudiendo elegir el tipo de comunicación (fiable o no), aunque de momento solo se acepta RAW y UDP para enviar los mensajes netlink.

Los mensajes pueden ser para obtener, almacenar o borrar información. De momento hay cuatro tipos de datos que se pueden obtener via netlink: información sobre las interfaces, sobre las direcciones configuradas, sobre la tabla de enrutado o sobre la caché ARP.

La librería rtnetlink ofrece un conjunto de funciones que permiten trabajar con mensajes de netlink relacionados con la tabla de enrutado. El autor del programa *iproute2* (ver anexo1) ha desarrollado unas librerías de alto nivel (fácil uso) para evitar el uso directo de rtnetlink. Esas librerías se llaman libnetlink, y se pueden obtener con iproute2 (ftp://ftp.inr.ac.ru/ip-routing) o bien con el paquete libnlink disponible en Suse 9.1 (http://www.suse.com/us/private/products/suse_linux/prof/packages_professional/libnlink.html).

Sin embargo las pruebas realizadas con esas librerías demostraron dos cosas: es más complicado programar con netlink que con routing sockets, y el soporte de IPv6 es mejor en estos últimos que en netlink. Por ejemplo, en netlink hay dos dominos, PF_ROUTE y PF_ROUTE6. Sólo está implementado el primero, pero sirve para obtener la tabla de enrutado tanto de IPv4 como de IPv6. Pero éste último no funcionaba.

Linux intenta emular la API de routing sockets de BSD para proporcionar portabilidad de código, aunque las pruebas hechas con código existente de BSD no funcionaron en Linux. Este intento de emulación se puede observa en el fichero bits/socket.h

```
#define   PF_ROUTE        PF_NETLINK /* Alias to emulate 4.4BSD.  */
```

En definitiva, se puede decir que netlink es un medio muy bueno para obtener información, pero que falta tiempo para que se implementen todas las funcionalidades deseadas. En cambio los routing sockets, aunque con menos prestaciones, es una API que realmente funciona y que en principio soporta IPv6.

Más información:

Página del manual
“man 7 netlink” y “man 7 rtnetlink”

Software libnetlink
ftp://ftp.inr.ac.ru/ip-routing/iproute2

Introducción a la programación con netlink
http://snafu.freedom.org/linux2.2/docs/netlink-HOWTO.html

ANEXO V
Notas para el futuro desarrollo

V.1 Introducción

El programa está organizado en bloques funcionales, cada uno con un prefijo que indica su utilidad. A continuación describiremos las cuestiones a mejorar en los distintos bloques funcionales, así como algunas indicaciones para añadir prestaciones al programa

Éstos son los ficheros que componen el programa **mcast** (anexo 2):

Makefile	*defs.h*	*main.c*	*PCAP.c*	*RT.c*
IFS.c	*NET.c*	*UTILS.c*	*GEOPAG.c*	

V.2 Comentarios al Makefile

Este fichero es un makefile creado manualmente. No se usó automake ni autoconf. Hay tres variables en tiempo de compilación que afectan al resultado del código. La primera es -DFROMFILE, que en vez de leer el fichero /proc/net/ipv6_route leerá el ./ipv6_route. Esto permite una modificación manual de las rutas en este fichero, para poder hacer pruebas. Lo mismo hará con el fichero /proc/net/if_inet6, que contiene la lista de IPs configuradas en las interfaces del sistema.

La segunda variable es –DIMPRIMELOTODO, que generará multitud de mensajes que mostrarán el cálculo interno que realiza el programa antes de reinyectar un paquete.

La tercera variable es -D DEF_GEOPAG_ROUTER, que sirve para poder compilar el archivo GEOPAG.con todas sus funciones, pues sino sólo sirve para crear el programa de prueba **GEOPAG_tester** (anexo 2).

V.3 Comentarios a defs.h

En todos los ficheros de código fuente (.c) se referencia a un fichero de cabeceras, “defs.h” que incluye todas las librerías necesarias para que funcione el código. Posteriormente en el Makefile se indica al compilador cómo lindar esas librerías al programa *mcast*. El principal problema que hay en este fichero es esto:

```
#define MAX_IFACES 20
#define MAX_RUTAS 200
```

Esto quiere decir que hay un límite en cuanto al número de IPs y de rutas configuradas en el sistema. Esto es así porque se usaron estructuras de tamaño estático en vez de dinámico. Una posible mejora sería cambiar el código para que las escrituras en esas estructuras se hiciesen de forma dinámica (mediante malloc() y realloc()).

V.4 Comentarios a main.c

Hay dos líneas que incluyen las cabeceras *<linux/in_route.h>* y *"ip2hack.h"*. Esto se hizo para probar las funciones de la librería libnetlink (anexo 3) en cuanto a la consulta de la tabla de enrutado del sistema mediante netlink. Mediante esto pudimos demostrar que las funciones propias de consulta de rutas (basadas en la interfaz proc) funcionan igual que las funciones de dicha librería. Si no se utiliza ninguna de estas funciones, sería conveniente eliminar libnetlink del proceso de compilación pues engrosa mucho el tamaño del ejecutable: pasamos de 500k a menos de 100 sólo eliminando esta librería.

V.5 Comentarios a PCAP.c

Hay que mejorar el manejo de paquetes IPv6, pues actualmente sólo se considera un mensaje de paging válido aquel cuya primera extensión header es del tipo Hop by Hop con unos datos formateados como texto ("[Paging_Distance]=xxx"). Según el protocolo GeoPaging (capítulo 3 del TFC) sería conveniente introducir información en una Destination Option Header, aunque actualmente no se controla si el paquete tiene esa cabecera o no.

También hay que estudiar a fondo la función PCAP_handle_ipv6(), usando un buen depurador pues no se sabe con exactitud cómo guarda libpcap los datos más pequeños de 8 bytes en memoria. En concreto hay problemas con la conversión de números de formato máquina a formato de red (ntohl y htonl) debido a que la rutina de inyección de paquetes necesita los datos en formato máquina. El código funciona bien pero en los campos de *traffic class* o *flowlabel* no se controla adecuadamente su tipo de ordenación.

V.6 Comentarios a RT.c

Sería necesario crear una función RT_obten_rutas() que internamente, según la interfaz elegida (PROC, SYSCTL o NETLINK) ejecutase una rutina u otra. La función RT_parsea_from_proc() ya está implementada. La de sysctl se probó y no funcionaba bien con IPv6, y la de Netlink se utilizaron las funciones de libnetlink (ver apartado V.4). Sería adecuado probar de usar todas las opciones posibles y así poder utilizar la más eficiente. Aunque actualmente el uso de un thread paralelo que actualiza las rutas hace que no se note tanto la ineficiencia de la lectura y procesado de ficheros de texto del proc.

Otro problema está en que sólo se consideran direcciones site-local en la tabla de enrutado, pero a lo mejor en un escenario real se utiliza sólo un prefijo limitado, donde sólo unas pocas direcciones site-local estuviesen reservadas para GeoPaging. Seria conveniente pues crear una función que extrayese exactamente las rutas con el prefijo deseado, no todas las que cumplen la condición de ser site-local.

V.7 Comentarios a IFS.c

El mismo comentario que en el apartado anterior: actualmente se usa una función IFS_parsea_from_proc(), pero debería estar enmascarada por otra función tipo IFS_obten_interfaces() la cual ejecutase la rutina correspondiente al método de comunicación con el kernel elegido.

V.8 Comentarios a NET.c

Sólo hay dos funciones realmente útiles, NET_libnet_inject() y NET_get_MAC_address(). El resto son funciones que usan RAW sockets, cuando el código al final usa libnet para inyectar el paquete. Sería conveniente unificarlo todo en una función NET_inyecta_paquete() que llamase a la función adecuada (usando libnet o usando raw socket). Como ya se ha comentado en el anexo 3, se consideraron a los sockets RAW como muy complejos de utilizar, pero son más eficientes en su rendimiento.

V.9 Comentarios a UTILS.c

Son funciones pensadas para ser reutilizadas, pero no están suficientemente documentadas ya que están extraídas de la librería libnetlink. Sería conveniente documentarlas mejor y en castellano (están todas en inglés).

V.10 Comentarios a GEOPAG.c

Sería interesante una función que pasase de IPv6 a su ID de celda, ya que no se realizó pues no fue necesario para el montaje de la maqueta. Tambíen habría que cambiar las llamadas a las funciones comentadas en los apartados V.6 y V.7, en el caso de que se siguieran las indicaciones.

Por último, habría que revisar si el cálculo aproximado de la funcion RT_inicializa_R es adecuado o no. El cálculo es un problema de geometría que se debería de revisar a fondo, y que se describe en la tabla 4.1 del TFC.

V.11 Futuros pasos

En general el código ya funciona adecuadamente, pero se debería medir su rendimiento. Éste se ve muy afectado por factores externos, como la velocidad de impresión por pantalla de mensajes. El programa va mucho más rápido si se redirecciona la salida a un fichero o al dispositivo nulo (p.ej. “programa > /dev/null”). Las medidas deberían considerar el tamaño de paquete, velocidad de transmisión, latencia de generación de paquetes y latencia de reenvío (para ver si coinciden o medir su retardo), etc. Y también sería interesante comparar esos resultados con otros protocolos de enrutado unicast y multicast, tanto IPv4 como IPv6.

Y como se ha explicado, hay librerías que no se han utilizado, como los divert sockets o los routing sockets, que aparentemente son mejores a las usadas. Pero ambas funcionan sobre plataformas diferentes (divert sockets en Linux, y routing sockets en FreeBSD), así que sería adecuado probar el código en ambas plataformas y realizar las modificaciones necesarias para poder experimentar posteriormente las diferencias de rendimiento en ambas. La opción más prometedora es el uso de divert sockets sobre linux, pues soluciona tanto el problema de capturar como el de inyectar paquetes en una sola librería implementada dentro del kernel (gran eficiencia de ejecución).