

*UNIVERSIDAD DE MURCIA*

FACULTAD DE INFORMÁTICA  
DEPT. DE INFORMÁTICA ELECTRÓNICA E INTELIGENCIA  
ARTIFICIAL



PROYECTO FIN DE CARRERA

# LA CARA OCULTA DE MBONE: Una arquitectura para el control de sesiones en MBone

Soluciones al problema del control de acceso

PEDRO MIGUEL RUIZ MARTÍNEZ  
Murcia, mayo de 1999

# Índice General

<b>1</b>	<b>Introducción y referencias históricas</b>	<b>3</b>
<b>2</b>	<b>La tecnología que subyace a MBone</b>	<b>6</b>
2.1	Introducción . . . . .	6
2.2	El direccionamiento multicast . . . . .	7
2.3	Unicast frente a multicast . . . . .	8
2.4	Direccionamiento multicast a nivel MAC . . . . .	9
2.5	El protocolo IGMP . . . . .	9
2.6	Enrutamiento Multicast . . . . .	12
2.7	El ámbito multicast . . . . .	13
2.8	Resumen . . . . .	14
<b>3</b>	<b>La integración del usuario y la tecnología</b>	<b>16</b>
3.1	Introducción . . . . .	16
3.2	Session Description Protocol (SDP) . . . . .	17
3.2.1	La sintaxis de SDP . . . . .	18
3.2.2	La actualidad de SDP . . . . .	18
3.3	Session Announcement Protocol (SAP) . . . . .	19
3.4	Session Initiation Protocol . . . . .	20
<b>4</b>	<b>El control de los emisores multicast</b>	<b>21</b>
4.1	El enrutamiento multicast y su relación con el kernel . . . . .	22
4.1.1	Soporte del sistema operativo al enrutamiento multicast . . . . .	22
4.1.2	Problema que nos plantea este tipo de funcionamiento . . . . .	23
4.2	Modificaciones al Kernel del sistema operativo . . . . .	24
4.3	Detalle de las modificaciones al kernel . . . . .	27
4.4	Una primera solución . . . . .	32
4.4.1	Ventajas e inconvenientes . . . . .	34
4.5	Una solución mejorada . . . . .	34
4.5.1	Protocolo de comunicación entre <i>mauthclient</i> y <i>musersd</i> (Mcontrol) . . . . .	35
4.5.2	Ventajas e inconvenientes . . . . .	37
4.6	Solución definitiva . . . . .	38
4.6.1	Comunicación entre el <i>musersd</i> y el <i>mauthserver</i> . . . . .	39
4.6.2	Comunicación entre el <i>mauthserver</i> y el <i>mauthclient</i> . . . . .	39
4.6.3	Creación de nuestro directorio de sesiones . . . . .	40
4.6.4	Ventajas e inconvenientes . . . . .	41
4.7	Otro enfoque alternativo . . . . .	43
4.7.1	Una primera alternativa descartada . . . . .	43
4.7.2	Una alternativa basada en una extensión de IGMP . . . . .	43
4.8	Implantación de la nuestra solución en el marco de la Universidad de Murcia . . . . .	46
4.8.1	Hacer nuestras modificaciones a los fuentes del sistema operativo . . . . .	46
4.8.2	Recompilación del kernel del MRouter . . . . .	47
4.8.3	Instalación de mouted 3.9 beta 3 en Linux . . . . .	48

4.8.4	Configuración del demonio de enrutamiento . . . . .	48
4.8.5	Instalación del resto de componentes . . . . .	49
<b>5</b>	<b>Conclusiones y vías futuras</b>	<b>50</b>
<b>A</b>	<b>Estudio de los algoritmos de enrutamiento multicast</b>	<b>55</b>
A.1	Algoritmos basados en inundación y poda . . . . .	55
A.2	MOSPF . . . . .	56
A.3	Center-based Trees . . . . .	56
A.4	Core-Based Trees . . . . .	57
A.5	Sparse-Mode PIM . . . . .	58
A.6	Border Gateway Multicast Protocol . . . . .	59
<b>B</b>	<b>Estructura del entorno de pruebas</b>	<b>61</b>
<b>C</b>	<b>Fichero in.h necesario para compilar mrouterd 3.9 beta 3</b>	<b>64</b>
<b>D</b>	<b>Modificaciones a <i>mrouterd 3.9 beta 3</i> para permitir la el control de emisores multicast</b>	<b>67</b>
<b>E</b>	<b>Fichero de configuracion del núcleo de Linux para actuar como MRrouter</b>	<b>71</b>

# Índice de Figuras

1.1	La pila de protocolos de Internet para Multimedia . . . . .	4
2.1	Jerarquización de las direcciones IP . . . . .	7
2.2	Enrutamiento unicast frente a enrutamiento multicast . . . . .	8
2.3	Mapping de direcciones multicast en direcciones Ethernet . . . . .	9
2.4	Formato de los mensajes IGMP . . . . .	10
2.5	Solapamiento entre zonas similar al que permite el administrative scoping . . . . .	14
3.1	Formato básico de una descripción de sesión realizada con SDP . . . . .	17
3.2	Formato del paquete SAP . . . . .	19
3.3	Llamada completa empleando SIP . . . . .	20
4.1	Estructura de la pila de gestión de red en Linux . . . . .	22
4.2	Arquitectura general del kernel modificado . . . . .	26
4.3	Esquema del funcionamiento del kernel modificado . . . . .	32
4.4	Arquitectura con la primera solución . . . . .	33
4.5	El applet de gestión de usuarios . . . . .	35
4.6	Protocolo de comunicación Mcontrol entre el <i>mauthclient</i> y el <i>musersd</i> . . . . .	35
4.7	Formato de los mensajes de solicitud y respuesta de Mcontrol . . . . .	36
4.8	Esquema de esta solución con administración distribuida . . . . .	37
4.9	Esquema general de la solución final . . . . .	39
4.10	Applet de administración de la solución final . . . . .	42
4.11	Arquitectura en base a <i>ingress</i> y <i>egress</i> routers . . . . .	44
A.1	Formación del árbol de enrutamiento multicast con CBT . . . . .	57
A.2	Formación de los árboles de distribución en PIM-SM . . . . .	59
A.3	Formación de los árboles de distribución compartidos en BGMP . . . . .	60
B.1	Estructura del entorno de pruebas . . . . .	61
B.2	Arquitectura actual de los túneles multicast a nivel nacional . . . . .	63

# Índice de Tablas

2.1	Valores para el campo Type del protocolo IGMP . . . . .	11
2.2	Asociación típica entre TTL's y ámbitos de sesiones . . . . .	14
2.3	Asociación entre TTL's y ámbitos en RedIRIS . . . . .	14
4.1	Parámetros de las llamadas al sistema para enrutamiento multicast . . . . .	23
4.2	Nuestras llamadas al sistema para la gestión de emisores multicast . . . . .	25
4.3	Límites de ancho de banda para los anuncios por TTL . . . . .	41
4.4	Nuevos valores para el campo Type . . . . .	45

# Agradecimientos

En primer lugar quiero agradecer a mi tutor de proyecto todo su apoyo así como el haberme dado la posibilidad de trabajar en un proyecto tan ambicioso como este y haber confiado en mí más que yo mismo.

En segundo lugar me gustaría agradecer a Norihiro Ishikawa del NTT Information and Communication Systems Laboratories su ayuda al resolvernros algunas cuestiones relativas al draft que ha elaborado su grupo de trabajo así como por el hardcopy de su sistema experimental sobre FreeBSD.

Además, me gustaría extender el agradecimiento a Francisco Cruz, Jesús Gonzalez Barahona y su grupo de trabajo dentro de la Universidad Carlos III de Madrid, por el interés mostrado en el tema así como por su participación en lsa reuniones virtuales que se celebraron dentro del grupo de trabajo IRIS-MBone.

Tampoco quiero dejar fuera de mi agradecimiento a RedIRIS por la coordinación del grupo de trabajo IRIS-Mbone y en especial a Angel L.Mateo Martínez que fue quién me ayudo en mis inicios con MBone y con el que siempre he podido contar.

Por último, agradezco a la Comisión Interministerial de Ciencia y Tecnología CICYT por soportar parcialmente todo este trabajo a través del proyecto TEL97-2001-E.

# Abstract

MBone[MB94] es una tecnología relativamente actual que se está asentando con el discurrir del tiempo. Sin embargo, durante los últimos años, los esfuerzos se han centrado sobre todo en el desarrollo de herramientas multimedia interactivas basadas en IP multicast[Dee89a] y han quedado al descubierto otra serie de aspectos que desde los comienzos han estado en un segundo plano.

Si bien Mbone permite la emisión de audio y vídeo sobre Internet con una calidad aceptable, normalmente sólo lo emplean investigadores en áreas relacionadas con esta tecnología debido a que tiene una serie de carencias tales como desaprovechamiento de ancho de banda, ausencia de control sobre las uniones a las sesiones o incluso imposibilidad de controlar el TTL de los paquetes enviados por los emisores multicast.

Todo este tipo de problemas hace que los proveedores de Internet (ISP) no aprovechen todas las posibilidades que ofrece esta tecnología debido sobre todo, a que ven a Mbone aún como una tecnología inmadura y no pueden tener un control eficaz de sus clientes.

Esta misma problemática se plantea en el entorno tanto de la Universidad de Murcia como en otras universidades a nivel nacional. Será nuestro objetivo pues, dotar a la comunidad universitaria de un mecanismo que permita la implantación de aulas de acceso libre a Mbone asegurando en todo momento que ninguno de los usuarios de dichas aulas pueda causar efectos dañinos en las sesiones que estén empleando otros usuarios.

Del mismo modo que es de vital importancia para el crecimiento de Mbone el desarrollo de nuevos algoritmos de compresión, nuevos algoritmos de transporte para tiempo real o incluso el empleo de nuevas herramientas que abran los horizontes y hagan más atractiva esta tecnología, no es menos importante la resolución de estos problemas que acabamos de plantear ya que hasta que Mbone no pueda ofrecerse como un servicio serio y tarificable nunca conseguirá un desarrollo y asentamiento pleno. Es por esto que el desarrollo del proyecto se va a enmarcar en la resolución del problema más importante y que quizá sea el que más está frenando el asentamiento de Mbone: la autenticación de emisores multicast y el control de acceso en entornos multicast. Para conseguir nuestro objetivo se modificarán los esquemas de enrutamiento multicast a nivel de sistema operativo.

# Capítulo 1

## Introducción y referencias históricas

Con el fin de conseguir la transmisión de contenidos multimedia sobre la arquitectura de Internet, se fue desarrollando toda una tecnología basada en IP multicast. Dado que en un principio eran muy pocos los routers capaces de entender el direccionamiento multicast, se estableció un esquema de túneles en torno a un backbone central al que se llamó MBone y que cada vez integra a más y más organizaciones.

En los últimos 7 años, las dos tecnologías más famosas y potentes que han aparecido son el World Wide Web (WWW) y el MBone[Kum96]. El Web es una tecnología muy simple basada en el modelo cliente/servidor típico de cualquier base de datos actual. Sin embargo, cuando hablamos de MBone, estamos hablando de una tecnología mucho más compleja y avanzada tecnológicamente. De hecho, introduce multitud de conceptos e ideas nuevas que permiten el traslado de las tecnologías multimedia a Internet.

Internet se desarrolló originalmente para el intercambio de información entre ordenadores, pero la definición de información se ha ido ampliando y haciéndose más flexible. Mientras que antes hablábamos de texto, ahora hablamos de intercambio de audio, vídeo, edición compartida y otros muchos nuevos servicios que surgen gracias a las mejoras en las tecnologías de red y los protocolos.

Si bien, la tecnología multimedia se encuentra ya desde hace varios años en nuestra vida diaria, su transmisión e implantación en las redes de ordenadores es totalmente novedosa y requiere de la introducción de bastantes cambios tanto en las tecnologías de red existentes como la definición e implementación de nuevos protocolos tales como RTP[SCFJ96], IGMP[Fen97], etc. La relación entre todos estos protocolos se puede apreciar en la figura 1.1.

Pese a lo actual que es el MBone, ya pueden apreciarse claramente dos aspectos que rodean esta nueva tecnología: el contenido y la distribución. Cuando hablamos del contenido nos estamos refiriendo a los datos o información que se intercambian (audio, vídeo,...). Esta es sin duda alguna la cara más atractiva y que más llama la atención del usuario. Sin embargo, hay otra cara oculta que normalmente pasa desapercibida para el usuario pero que es sin duda alguna la más importante y sin la que ninguno de estos efectos tan espectaculares como la transmisión de vídeo tendría sentido. Nos referimos a la distribución de los datos (algoritmos de enrutamiento multicast, establecimiento de túneles, etc.).

Dentro de la parte más relacionada con las aplicaciones de videoconferencia multimedia, existen actualmente dos enfoques distintos. Por un lado soluciones muy conocidas tales como NetMeeting y CU-SeeMe que se basan en la idea de los *reflectores*[Tho96] o nodos de la red que actúan como repetidores de información: todo lo que envía cada participante de la videoconferencia se reenvía al resto de participantes.

Frente a esta solución, aparece otro grupo de aplicaciones basadas en el IP multicast

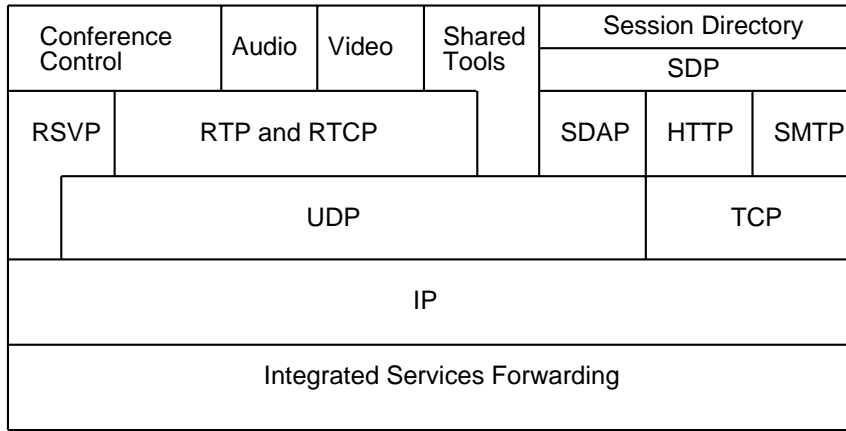


Figura 1.1: La pila de protocolos de Internet para Multimedia

como medio de transmisión. Con este tipo de transmisiones se pretende disminuir el ancho de banda que se consume en los enlaces así como aumentar la escalabilidad de las aplicaciones desarrolladas. Este beneficio se obtiene gracias a que el emisor siempre enviará un único paquete por la red. Este paquete solo se replicará en los nodos en los que sea necesario. De este modo, se obtiene un aprovechamiento óptimo del ancho de banda de los enlaces.

El IP multicast [Dee89b] se basa en el empleo de direcciones de grupo en lugar de las direcciones IP pertenecientes a un sólo equipo. Así, un determinado equipo recibirá los paquetes dirigidos tanto a su dirección IP como a la correspondiente a alguno de los grupo a los que se haya unido.

Como contrapunto, este enfoque requiere unos enrutadores especiales llamados MRouters que son los que nos permiten trabajar con este tipo de direcciones de grupo. De este modo, la complejidad recae en las estructuras de red necesarias. Estos nuevos elementos llamados MRouters (Multicast Routers), actúan de un modo similar a los routers tradicionales con la única salvedad de que emplean un algoritmo de enrutamiento multicast. Es decir, cuando les llega un paquete dirigido a un determinado grupo multicast por un determinado interfaz virtual, determinan a partir de su tabla de enrutamiento multicast, el interfaz o interfaces de salida por los que deben reenviar este paquete.

Básicamente, se pueden distinguir dos tipos de MRouters: los routers típicos que implementan ya algoritmos de enrutamiento multicast y los hosts que ejecutan algún demonio de enrutamiento multicast.

Como no todos los routers que existen actualmente en Internet tienen soporte de IP multicast, los equipos que hacen enrutamiento multicast se comunican mediante el establecimiento de túneles por los que se encapsulan los paquetes multicast. En concreto, esos paquetes multicast se encapsulan dentro de paquetes unicast dirigidos al MRrouter situado al otro extremo del túnel. A toda esta infraestructura de túneles para soporte multicast es a lo que se le ha llamado Multicast Backbone (Mbone).

Durante los últimos años, los esfuerzos y las inversiones más importantes se han dirigido hacia la creación de nuevas aplicaciones, protocolos de tiempo real, etc. Sin embargo, hay otra serie de aspectos que han quedado descuidados que si bien no son tan vistosos, si que son igual o más importantes y necesarios para hacer del IP multicast un servicio serio y maduro capaz de ser ofertado por los IPS's. Estos aspectos tienen que ver con el control de los usuarios que hay en un determinado grupo, el ámbito o Time To Live (TTL) con el que están emitiendo, etc. Todo esto convierte al IP multicast en un servicio que los proveedores de servicios, ya sean estos proveedores comerciales, las universidades o cualquier otro tipo de proveedor de conexión a Internet, ven complejo, difícil de gestionar y de utilizar. Esto provoca que este servicio se encuentre totalmente

infrautilizado.

Afortunadamente, los algoritmos y sistemas de enrutamiento han alcanzado un grado bastante alto de estabilidad, es decir, aunque el algoritmo de enrutamiento en sí no es el más eficiente que se puede utilizar, lo cierto es que una vez levantados correctamente los túneles necesarios, el sistema logra un alto grado de estabilidad. En este sentido, los problemas aparecen cuando surge algún problema en la red multicast que hay que depurar. Esto se debe sobre todo a la falta de aplicaciones de depuración tales como las que puede haber para el IP unicast.

Sin embargo, aunque la gestión de la red una vez que ésta se ha puesto en funcionamiento no requiere un gran esfuerzo, los proveedores continúan sin ofrecer masivamente el servicio, sobre todo debido a que no existe ningún sistema que permita controlar quién accede a MBone y cómo. Así, los proveedores comerciales no lo pueden ofrecer como servicio de valor añadido porque o lo ofrecen a todos sus clientes o no lo ofrecen. Del mismo modo, las universidades si ofrecen el servicio a todos sus usuarios sufren el riesgo de ver considerablemente aumentado el tráfico en sus enlaces.

Así pues, para que el MBone alcance el grado de popularidad necesario para que se aproveche toda la potencia que proporciona, es necesario definir un esquema que proporcione un cierto control sobre el uso que se hace de él. Con esto, se podría ofrecer el servicio a los clientes de los proveedores y alcanzaría un cierto grado de popularidad, que además provocaría el desarrollo de nuevas aplicaciones más amigables y vistosas de cara al usuario final.

Por lo tanto, el problema en el que nos centramos es el control de acceso de los usuarios a MBone. Este sistema debería proporcionar un control sobre qué usuario recibe o envía sobre determinado grupo multicast. Para abordarlo nosotros hemos optado por una solución intermedia que permita la autenticación de los emisores en los grupos multicast, pero que no hace ningún cambio en el sistema de recepción de información.

Este problema que pretendemos resolver, es sin duda alguna totalmente innovador. De hecho, son muy pocas las entidades que se han planteado este problema y aún menos las que han intentado resolverlo. Una de las entidades que si que lleva un par de años intentando buscar una solución ha sido la *Red Investigadora y Académica (RedIRIS)* pero hasta el momento de finalización de este proyecto siguen sin conseguirlo. De hecho, somos la primera universidad europea en conseguir implementar y poner en marcha un mecanismo de este tipo.

A nivel mundial, sabemos de la existencia de una implementación sobre FreeBSD de un sistema similar en cuanto a filosofía pero un tanto diferente en cuanto a funcionamiento. Este sistema se basa en la extensión del protocolo IGMPv2 para dar soporte a la autenticación de usuarios. De hecho, esta filosofía aparece reflejada en el draft [IYT98] elaborado por Norihiro Ishikawa del NTT Information and Communications Center Laboratories.

## Capítulo 2

# La tecnología que subyace a MBone

En el apartado anterior, hemos visto una breve introducción a la tendencia actual dentro del mundo de MBone así como al problema que pretendemos resolver. En este apartado, nos centraremos más en dar una visión del funcionamiento del MBone así como de explicar algunos de los elementos tecnológicos más interesantes en los que se apoya MBone. Haremos especial hincapié en todos los nuevos protocolos que han aparecido y que se están modificando o actualizando a las nuevas tendencias.

### 2.1 Introducción

Cuando necesitamos enviar información a muchos receptores de forma simultánea, podemos recurrir a dos métodos tradicionales: replicación y broadcast.

Entendemos por replicación al envío de varios paquetes idénticos uno para cada receptor. Para el envío de tráfico multimedia esto sólo puede funcionar a costa de un coste muy elevado para el emisor y las tecnologías de transmisión subyacentes.

Del mismo modo, si empleamos broadcast, el coste de los canales de comunicación sería muy elevado y se necesitaría un ancho de banda desmesurado.

Para este tipo de tráfico que tiene unos requerimientos tan extremos surge una solución alternativa: IP multicast. Esta nueva tecnología convierte a Internet en un canal de broadcast pero imponiendo una serie de condiciones que permiten que cualquiera pueda enviar multimedia sin necesidad de un alto consumo de ancho de banda.

Internet es una red basada en datagramas[Ste94]. Es decir, los paquetes pueden llegar a su destino por cualquier ruta y en cualquier orden. El emisor se limita a enviar el paquete pero en ningún momento interviene en el cálculo de rutas ni se preocupa por el camino que seguirá dicho datagrama. Es la subred subyacente la que se encarga de la entrega (en modo best-effort) de esos datagramas.

El añadir multicast a la Internet actual no altera la filosofía de este modelo básico. Los emisores siguen enviando simplemente el datagrama a una dirección IP, pero con la única salvedad de que las direcciones destino van a presentar un formato un tanto especial que explicaremos más adelante. Donde aparece realmente el cambio de concepción es en la recepción de datagramas multicast. Al contrario que en el caso tradicional del unicast, los hosts se van suscribiendo de forma dinámica a los grupos multicast que les interesen y de este modo hacen que les empiece a llegar el tráfico multicast asociado a ese grupo. Este modelo puede resumirse en:

- Los emisores envían los datagramas con una dirección multicast como destino.
- Los receptores se suscriben a los grupos multicast que les interesen.

- Los routers se encargan de hacer que el tráfico multicast llegue desde los emisores hasta los receptores.

Como acabamos de ver, el envío de tráfico multicast no es muy diferente de lo que es el envío tradicional de datagramas en Internet. La única diferencia es que la dirección IP de destino pertenece a un grupo multicast. Sin embargo, para la recepción de tráfico multicast, los hosts que estén interesados deben de indicar a su MRrouter local cuales son los grupos en los que están interesados. Para ello se utiliza el *Internet Group Management Protocol*[Fen97] (IGMP) que también comentaremos más adelante.

Si bien la idea general es muy simple, aparecen algunas cuestiones que trataremos más adelante como son la gestión de los grupos multicast, el cómo decidir sobre que grupo emitir, etc. Lo que está claro es que los MRouters deben de ser capaces de construir una árbol de distribución que parta de los emisores y llegue a todos los receptores. Sin embargo, no cuentan con ninguna información por parte de los hosts ya que:

- Los emisores simplemente envían a un grupo multicast, pero no conocen a los receptores.
- Los receptores se suscriben a determinados grupos multicast pero no tienen porqué conocer en ningún momento la dirección IP del emisor.

Pero bueno, estas cuestiones las vamos a ir viendo más en detalle a continuación.

## 2.2 El direccionamiento multicast

Para poder comprender el direccionamiento multicast en Internet, resulta interesante comenzar por estudiar el direccionamiento unicast. Como todos sabemos, el direccionamiento unicast en Internet se realiza mediante la asignación de un identificador o identificadores únicos a cada host de Internet. A este identificador se le suele denominar comúnmente la *dirección IP* del host precisamente porque estas direcciones se asocian al nivel de red del host. En IPv4, estas direcciones constan de 32 bits mientras que en IPv6 se amplía el rango de direccionamiento hasta 128 bits.

Normalmente estas direcciones (en el caso de IPv4) suelen denotarse mediante 4 valores entre 0 y 255 separados por puntos, por ejemplo, 155.54.95.100. Estas direcciones se clasifican de un modo jerárquico mediante la división de esas direcciones en un identificador de red y un identificador de host. A su vez, cualquier organización puede dividir su rango de direccionamiento interno en diferentes subredes mediante la configuración de la llamada *máscara de red*. Podemos apreciar las diferentes clases de direcciones IP en la figura 2.1.

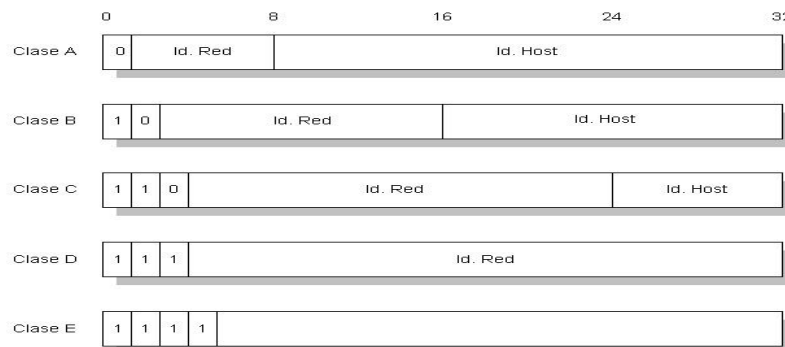


Figura 2.1: Jerarquización de las direcciones IP

Como se aprecia en la figura 2.1, el primer byte de la dirección es el que determina el tipo de dirección que se está utilizando. Pues bien, dentro del rango de direcciones de

clase D, se reservan las que van del rango 224.0.0.0 al 239.255.255.255 para el Mbone. Son las llamadas *direcciones de grupo o direcciones multicast*.

De este rango de direcciones, se reserva el rango 224.2.\*.\* para la realización de conferencias multimedia. La autoridad encargada de reservar estas direcciones de Mbone es el *Internet Address Number Authority* IANA. Además de este rango que se reserva, hay otra serie de direcciones con significado específico que se emplean para cuestiones de enrutamiento y control. Por ejemplo, la dirección 224.0.0.1 se asocia a todos los equipos de la red con capacidad multicast. Es decir, cualquier host con capacidad multicast deberá de atender cualquier datagrama que le llegue dirigido a este grupo multicast.

## 2.3 Unicast frente a multicast

Una vez hemos visto cuales son las direcciones que se asignan para tráfico multicast y cómo se asignan, nos vamos a centrar en este apartado en ver como se modifica la distribución de los paquetes en este tipo de entornos multicast así como las ventajas que aporta este nuevo enfoque frente al enfoque tradicional a la hora de trabajar con contenidos multimedia.

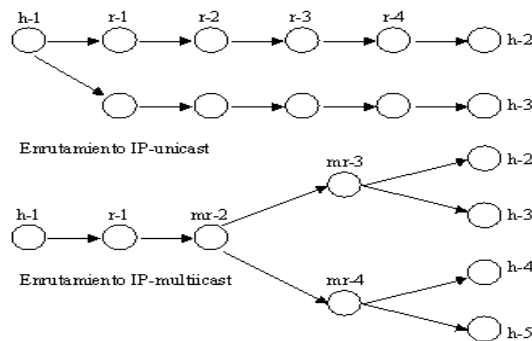


Figura 2.2: Enrutamiento unicast frente a enrutamiento multicast

En la figura 2.2 podemos apreciar claramente la diferencia de enfoque entre el encaminamiento unicast tradicional y el encaminamiento multicast. Con el encaminamiento unicast tradicional, la sobrecarga de la red de comunicaciones y la potencia de procesamiento de los hosts aumenta considerablemente frente al otro enfoque. Esto se debe principalmente al hecho de que el host origen de las comunicaciones debe de enviar un datagrama a la dirección de destino de cada uno de los receptores. Es por esto, que la arquitectura tradicional basada en IP unicast tiene serios problemas con la escalabilidad que la hacen totalmente inadecuada para el envío de datagramas en tiempo real con comunicaciones muchos a muchos.

Para resolver todas estas carencias que presenta IP unicast aparece una nueva tecnología: IP multicast. Como también se aprecia en la figura 2.2, la filosofía de IP multicast se centra en la idea de que la replicación de datagramas sólo se va a realizar en los MRouters y nunca en los hosts. Además, esta replicación sólo se va a producir en los nodos en los que sea realmente necesario. Esto permite que las exigencias en cuanto a la capacidad de cálculo de los hosts sean menores. De hecho, el emisor sólo tiene que enviar un único datagrama con la dirección IP destino de un determinado grupo multicast independientemente del número de receptores. Además, este enfoque añade una ventaja adicional: el emisor en ningún momento tiene porqué saber las direcciones de los destinatarios mientras que en el enfoque anterior, el emisor debe conocer cada uno de sus receptores para poder enviarles los datagramas correspondientes dirigidos a su dirección de nivel de red.



de unión entre estos dos componentes lo proporciona el direccionamiento multicast que acabamos de explicar y el protocolo IGMP para controlar las suscripciones de hosts a grupos multicast junto con la transmisión y recepción tradicional de paquetes IP. En esta sección vamos a tratar la parte correspondiente al host, es decir, IGMP y en las siguientes secciones trataremos las cuestiones relativas a la red.

En lo relativo a los hosts, lo primero que debemos exigirles es que sean capaces de manejar direcciones multicast. Para que un host pueda soportar multicast, su interfaz de servicios IP debe extenderse en tres aspectos principales:

1. Un host debe de poder unirse a un determinado grupo multicast. Esto significa que debe ser capaz de reprogramar su nivel de red y posiblemente sus niveles inferiores para ser capaz de recibir paquetes con una dirección multicast como destino.
2. Una aplicación que se encuentra suscrita a un determinado grupo multicast y comienza a emitir a ese grupo, debe de ser capaz de indicar si desea que el host reenvíe hacia su interfaz de loopback los paquetes que ella misma envía de forma que ésta reciba sus propios paquetes.
3. Un host debería ser capaz de limitar el ámbito o *Time To Live* TTL con el que se envían los mensajes. El protocolo IP contiene un campo denominado *time to live* que originalmente se usaba para controlar el tiempo de vida de los paquetes y evitar así los bucles en las rutas. Este mismo campo se usa en multicast en combinación con los MRouters para controlar como de lejos puede llegar un datagrama multicast.

Cuando una aplicación le dice al software de red del host que desea unirse a un grupo multicast, dicho software comprueba que el host no se encuentra ya suscrito al mismo grupo. Si no lo había realizado, entonces anota el hecho y envía un mensaje IGMP de tipo *join*. Al mismo tiempo, se transforma esta dirección multicast a nivel IP a una dirección multicast a nivel MAC y se programa la interfaz de red para que acepte paquetes que le lleguen dirigidos a esa dirección MAC.

Existen diferentes versiones de IGMP que van desde IGMPv1 [Dee89a] hasta IGMPv3 [Fen97]. Actualmente la más utilizada es IGMPv2 que se define en el RFC 2236 mientras que IGMPv3 está siendo desarrollado actualmente y aún no está totalmente especificado.

Como ya hemos comentado anteriormente, este protocolo sirve para comunicar los hosts de una determinada subred con su MRouter más cercano y permite que los hosts indiquen al MRouter los grupos multicast en los que están interesados.

Físicamente, los mensajes IGMP viajan encapsulados en paquetes IP (con un campo de *número de protocolo* de 0x02), y presentan el formato que se muestra en la figura 2.4.

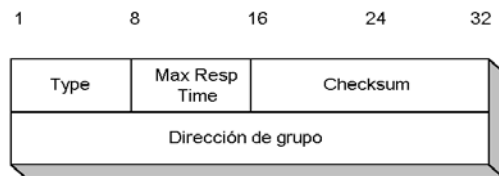


Figura 2.4: Formato de los mensajes IGMP

Mediante el intercambio de este tipo de mensajes, los MRouters pueden aprender qué grupos multicast tienen miembros en cada una de sus interfaces físicas. Cada router multicast, almacenará una lista para cada interfaz física con los grupos por los que al menos un host de la subred que define esa interfaz ha mostrado interés.

De todos los valores que puede tomar el campo *Type* los más destacados son los que se muestran en la tabla 2.1. Es también importante destacar el hecho de que dentro del mensaje *Membership Query* pueden darse dos tipos de mensajes dependiendo del valor que tome el campo *Group Address*:

Valores del campo <i>Type</i> de IGMP	
Valor	Significado
0x11	Membership Query
0x16	Version 2 Membership Report
0x17	Leave Group
0x12	Version 1 Membership Report (por compatibilidad con IGMPv1)

Tabla 2.1: Valores para el campo *Type* del protocolo IGMP

- Cuando dicho campo contiene el valor 0.0.0.0, significa que el MRouter asume el rol de *Querier* y está preguntando por los grupos que tienen miembros en la red directamente conectada. A este tipo de consulta la denominaremos *General Query*
- Cuando el valor de ese campo es un grupo multicast en concreto, entonces el MRouter está asumiendo el rol de *non-Querier* y está consultando si ese grupo multicast tiene algún miembro en la red directamente conectada. A este tip de consulta la denominaremos *Group-Specific Query*

Al inicializarse, todos los MRouters actúan como *Querier*. Es decir, van a enviar solicitudes dirigidas a todos los hosts con soporte multicast para que les respondan con los grupos en los que están interesados. Si más adelante detectan otro MRouter en esa subred que también esté en modo *Querier* y con dirección IP menor entonces pasan a actuar en modo *non-Querier*. Es decir, que sólo van a poder preguntar por la pertenencia de hosts a determinados grupos multicast en concreto y nunca en general. Esta decisión de elegir como MRouter del tipo *Querier* el de menor dirección IP es simplemente para asegurar que sólo pueda haber un Mrouter en modo *Querier* en cada tramo de red.

Los MRouters que asuman el rol de *Querier* enviarán de forma periódica cada cierto periodo de tiempo preestablecido (que denominaremos *Query interval*), un mensaje IGMP del tipo *Membership Query* y más concretamente del tipo *General Query*. Los MRouters que asuman el rol de *non-Querier* sólo enviaran mensajes *Membership Query* del tipo *Group-Specific Query*.

Este tipo de mensajes se envían a la dirección multicast de todos los hosts con soporte multicast (224.0.0.1) con un TTL de 1 para evitar que estos mensajes puedan salir fuera de la red local. (En realidad, aunque se envíen con un TTL superior a 1 no salen de la subred porque el MRouter reconoce el grupo multicast 224.0.0.1 y no retransmite ninguno de los datagramas dirigidos a ese grupo).

Cuando un host recibe un mensaje de *General Query*, establece un temporizador para cada grupo al que se encuentre suscrito (excepto para el 224.0.0.1) inicializado a un valor aleatorio diferente y en el rango (0, *MaxResponseTime*). Siendo *MaxResponseTime* el valor para este parámetro que viene en el mensaje de *General Query*. Cuando un host recibe un mensaje de *Group-Specific Query*, establece un temporizador del mismo modo que antes pero sólo para el grupo que aparezca en el mensaje de *Group-Specific Query*. En el caso de que ya hubiese un temporizador corriendo para alguno de los grupos, sólo se reinicializará el temporizador cuando el nuevo *MaxResponseTime* sea menor que el anterior.

Cuando expira alguno de los temporizadores, entonces el host envía un mensaje del tipo *Version 2 Membership Report* al grupo multicast en cuestión con un TTL de 1.

Para evitar duplicación de informes de respuesta, cuando a un host le llega un mensaje *Membership Report* de otro host mientras él tiene el temporizador asociado a ese grupo aún corriendo, el primer host ya no envía ningún *Membership Report* a ese grupo.

Pero surgen en este momento una serie de preguntas a las que debemos de dar respuesta:

1. ¿Porqué no es necesario que el host que ve el *Membership Report* de otro host envíe

su propio *Membership Report*?. Pues básicamente porque al MRouter le basta con que haya un host interesado en la subred para reenviar los paquetes dirigidos a ese grupo por el interfaz correspondiente. Y como todos los hosts interesados escuchan ese grupo, seguirán recibiendo los paquetes correspondientes.

2. ¿Porqué el host envía el *Membership Report* al grupo multicast que corresponda cuando en realidad el grupo multicast del *Membership Report* va en el campo *Group address*?. Esto es así por dos motivos principalmente. El primero es que de este modo conseguimos que los otros host interesados al grupo puedan ver el mensaje del primero en contestar ya que si se han unido al grupo multicast, recibirán todos los datagramas dirigidos a ese grupo multicast y en concreto el *Membership Report* de otros hosts interesados en el mismo grupo. El segundo es que así no hacemos que hosts que no tienen ningún interés en ese grupo tengan que procesar paquetes adicionales. Es decir, si la respuesta en vez de ir al grupo multicast que corresponda fuese al grupo 224.0.0.1 (todos los hosts multicast), este mensaje llegaría a todos los hosts de la subred incluidos los que no están interesados. Obviamente, el hecho de que se envíe esta respuesta con un TTL de 1 es para que no salga de ese tramo de red.

Una vez que el MRouter recibe un mensaje *Membership Report*, añade el grupo a la lista de grupos con receptores correspondiente al interfaz por la que recibió el mensaje. En este momento, se refresca el temporizador asociado a ese grupo. Si en algún momento el temporizador asociado a algún grupo multicast expirase, el MRouter asumiría que no quedan hosts interesados en ese grupo y dejaría de reenviar los datagramas dirigidos a ese grupo por ese interfaz.

Cuando un host se une a un grupo multicast, lo primero que debe de hacer es actuar como si le hubiese llegado un *Group-Specific Query* para ese grupo multicast y enviar varios *Membership Report* consecutivos para evitar que se pudiese perder el primer mensaje y sea éste el primer host de la subred interesado en ese grupo multicast.

Del mismo modo, cuando un host abandona un grupo multicast, si él era el último que contestó a un mensaje de *Query* con un *Membership Report* para ese grupo, éste debería de enviar un mensaje *Leave Group* al grupo multicast de todos los routers (224.0.0.2). Si él no fue el último en contestar al mensaje de *Query*, esto significa que hay algún otro host interesado y no es necesario que se envíe el mensaje *Leave Group*. Sin embargo, y en caso de duda lo mejor es enviar el mensaje de *Leave Group* al abandonar un grupo multicast.

Cuando un MRouter recibe un mensaje de *Leave Group*, pregunta en el interfaz por el que le ha llegado el mensaje para comprobar que no quedan hosts interesados en el grupo. En caso de que no haya respuesta entonces asume que no hay ningún host interesado y automáticamente deja de reenviar los datagramas dirigidos a esa dirección multicast por el interfaz correspondiente.

La principal diferencia ente IGMPv1 e IGMPv2 es la introducción del mensaje de *Leave Group* para mejorar la latencia a la hora de abandonar grupos multicast. Por último mencionar que IGMPv3, está intentando implantar un esquema conocido como *source-specific joining*. Este esquema se basa en el hecho de que un host pueda suscribirse o rechazar el tráfico proveniente de un determinado emisor más que tener que hacerlo con el grupo entero. Lógicamente, este nuevo esquema aumenta la complejidad y el número de estados que deben de contemplarse en los MRouters.

## 2.6 Enrutamiento Multicast

El principal problema que se plantea con el enrutamiento multicast es: dado que los emisores y los receptores no se conocen unos a otros, ¿Como consiguen los MRouters que todo el tráfico multicast fluya de los emisores a los receptores?.

Pues bien, supuesto que tenemos como tecnología subyacente al protocolo TCP/IP que nos permite el envío de información entre hosts por canales unicast, se pueden plantear un gran espectro de soluciones posibles. En un extremo, podemos considerar la posibilidad de inundar datos desde el emisor a todos los posibles receptores y permitir a los MRouters la posibilidad de podar (prunning) algunas ramas del árbol de distribución cuando no haya hosts interesados en dichas ramas. En el otro extremo, podemos enviar información en un algoritmo de enrutamiento multicast que informe de la localización de todos los receptores a los MRouters en todos los caminos hasta todos los emisores.

Lógicamente ninguno de estos dos enfoques anteriormente planteados son buenos a escala global. Normalmente se emplean soluciones de compromiso situadas entre estos dos extremos.

Básicamente, los algoritmos más empleados en este sentido son los siguientes:

- Algoritmos basados en inundación y poda: DVMRP (Distance Vector Multicast Routing Protocol) [WPD] y DM-PIM (Dense Mode Protocol Independent Multicast).
- MOSPF (Multicast Open Shortest Path First) [Moy94].
- Basados en Center-Based Trees: CBT (Core-based Trees) [Bal97], SM-PIM (Sparse Mode Protocol Independent Multicast) [EFH<sup>+</sup>97] y BGMP (Border Gateway Multicast Protocol).

Para entender mejor los límites prácticos del empleo del multicast resulta interesante hacer un análisis de los algoritmos de enrutamiento más comunes con sus pros y sus contras. Este análisis lo podemos ver en el apéndice A.

## 2.7 El ámbito multicast

Cuando las aplicaciones trabajan en todo el MBone, queda claro que no todos los grupos multicast tienen porqué tener un ámbito global. Si esto fuese así, algunos algoritmos tales como los basados en inundación y poda tendrían rendimientos más que pobres y además se impediría la reutilización de diferentes direcciones multicast. Si limitamos el ámbito de una sesión podemos permitir que la misma dirección multicast se emplee en varias localizaciones siempre y cuando los ámbitos de esas sesiones no se solapen.

Actualmente se proponen dos esquemas de control del ámbito: *TTL scoping* y *administrative scoping*

### TTL scoping

Esta solución consiste en emplear el campo *Time To Live* de la cabecera de los datagramas IP para especificar un valor entre 0 y 255 que mida el ámbito dentro del MBone por el que puede circular el datagrama. Este mecanismo se complementa con el establecimiento de unos thresholds de entrada en los túneles multicast y enlaces multicast. El funcionamiento es muy sencillo. Cuando llega un paquete multicast al MRRouter, este decrementa el TTL y si el resultado es menor que el threshold del túnel el paquete se ignora.

Todos estos thresholds deben elegirse de un modo consistente de forma que se pueda asociar un TTL para sesiones locales, nacionales, internacionales, etc.

Los valores típicos de TTL que se suelen asociar a cada ámbito aparecen reflejados en la tabla 2.2.

La asociación entre el TTL y el alcance que rige en España, lo impone RedIRIS y es el mostrado en la tabla 2.3

Asociación entre TTL y ámbito	
TTL	Ámbito
1	Localhost
15	Local
31	Regional
63	Nacional
127	Mundial

Tabla 2.2: Asociación típica entre TTL's y ámbitos de sesiones

Asociación entre TTL y ámbito en el entorno RedIRIS	
TTL	Ámbito
1	Localhost
32	Regionales
48	Nacionales
64	Europeas
128	Internacionales

Tabla 2.3: Asociación entre TTL's y ámbitos en RedIRIS

## Administrative Scoping

En algunas situaciones en las que resulta muy complicado establecer los TTL adecuados para conseguir solapamiento de ámbitos y cosas así puede emplearse el Administrative Scoping. De este modo se pueden conseguir esquemas solapados como el de la figura 2.5.

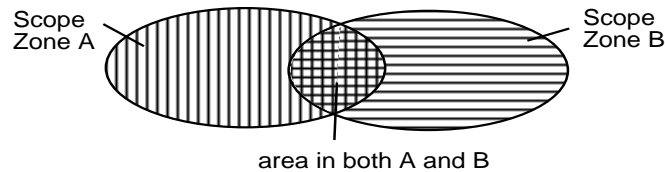


Figura 2.5: Solapamiento entre zonas similar al que permite el administrative scoping

El administrative scoping permite la configuración de una frontera especificando un rango de direcciones multicast que no serán reenviadas a través de esa frontera en cualquier dirección.

Una vez hemos visto los dos enfoques, la pregunta es, ¿Cual es la tendencia actual?. Sin duda alguna el enfoque más utilizado es el basado en TTL ya que es el más sencillo de utilizar. El *administrative scoping*, si bien es mucho más flexible, también es cierto que tiene una serie de inconvenientes que están haciendo que no se emplee demasiado. De hecho, es imposible saber a partir de una dirección multicast donde irá el paquete. Es por esto que la tendencia actual es emplear el *TTL scoping* y sólo recurrir a la otra alternativa para configurar aspectos muy concretos de nuestro entorno multicast.

## 2.8 Resumen

Hemos visto a lo largo de este capítulo cual es la tecnología y los protocolos que permiten tanto la emisión de contenidos multimedia por una red del tipo best-effort como es Internet, como los protocolos y técnicas que han tenido que diseñarse para permitir la entrega de estos contenidos.

Hemos visto la forma en la que se produce la comunicación, como se envían los contenidos a grupos multicast, el empleo del protocolo IGMP para que los receptores puedan mostrar su interés por los contenidos y recibirlos, etc. Sin embargo, quedaría por explicar como se hace la unión entre el usuario y estos protocolos. Es decir, un usuario tiene que mostrar interés por un contenido pero, ¿Cómo sabe donde está ese contenido?, ¿Cómo sabe que grupo multicast corresponde a ese contenido?. Es más, ¿Cómo sabe siquiera de la existencia de dicho contenido?.

Pues bien, todo este tipo de cuestiones son las que vamos a tratar en el siguiente capítulo.

## Capítulo 3

# La integración del usuario y la tecnología

En este capítulo, veremos como se manejan las sesiones multimedia. Realmente no existe ningún mecanismo formal para manejarlas, sin embargo, en la práctica necesitamos ofrecer a los usuarios que desean participar en la comunicación algún mecanismo para que puedan averiguar la dirección multicast a emplear, los codecs con los que transmitir o recibir, e incluso para descubrir la propia existencia de la sesión.

Si bien, al principio cabe pensar en el empleo del correo electrónico, realmente hoy por hoy existen unas técnicas específicas más integradas y sencillas de manejar para los usuarios.

Existen muchos modos de iniciar las sesiones dependiendo principalmente del objetivo de la reunión. Sin embargo, se pueden dividir básicamente en dos tipos diferentes: invitación y anuncio.

### 3.1 Introducción

Actualmente, existen dos tendencias o familias de protocolos a la hora de establecer sesiones multimedia: la familia de protocolos del IETF y los protocolos del ITU. La familia de protocolos del ITU está más enfocada a videoconferencias basadas en RDSI sobre la red telefónica mientras que los protocolos del IETF están más enfocados hacia una arquitectura global sobre Internet similar a Mbone.

En un futuro no muy lejano se preve que estas tecnologías converjan y podamos por ejemplo iniciar una conferencia basada en el protocolo ITU-H.323[Tho96] usando para ello el protocolo SIP del IETF. Sin embargo, y mientras todo esto llega, nosotros nos vamos a centrar en los protocolos del IETF que son los que realmente se adaptan a la arquitectura sobre la que estamos creando unos mecanismos de valor añadido. En concreto, los protocolos que propone el IETF para el inicio de las sesiones son el *Session Description Protocol (SDP)*[HJ98], *Session Announcement Protocol (SAP)*[Han96] y *Session Invitation Protocol (SIP)*[HSSR99]. Todos estos protocolos los veremos con mayor profundidad en los apartados siguientes.

Como ya hemos comentado antes, el IETF se plantea básicamente dos modos de crear una sesión. Sin embargo, también se admiten esquemas híbridos tales como el hecho de anunciar una sesión con un determinado TTL y al mismo tiempo invitar a algunos usuarios en concreto para que participen.

Para anunciar una sesión a todos los posibles participantes se emplea el protocolo SAP (*Session Announcement Protocol*). Éste envía información anunciando la sesión a todos los que estén escuchando a una determinada dirección multicast por la que se reciben los anuncios.

Para invitar a alguien se usa el protocolo SIP (*Session Invitation Protocol*). Este protocolo se encarga de enviar la información necesaria para poder participar en la sesión a un usuario concreto.

Ambos protocolos transportan una descripción de la sesión que sigue un formato muy concreto especificado en el protocolo SDP (*Session Description Protocol*). Realmente aunque en su nombre pone protocolo, el SDP no es realmente un protocolo, es más bien una especificación del formato que debe de presentar el campo de descripción de la sesión de los otros protocolos que se emplean para el anuncio de las sesiones.

### 3.2 Session Description Protocol (SDP)

Este protocolo aparece reflejado en el RFC 2327[HJ98] y está en estado de *Proposed Standard del IETF*.

La descripción de una sesión expresada en SDP es una pequeña descripción en modo texto del nombre, el propósito, los codecs a emplear, las direcciones multicast empleadas, información de la temporización y el transporte, etc.

SDP es simplemente el formato con el que se describe la sesión y no dice nada de cómo debe hacerse el transporte de esta sesión. De hecho, para difundir la descripción de la sesión pueden emplearse mecanismos tan diferentes como: SAP, SIP, Real-Time Streaming Protocol (RTSP), correo electrónico usando las extensiones de MIME o incluso el protocolo HTTP.

Aunque SDP no está pensado para que lo lean los humanos, presenta un formato que puede entenderse tal y como muestra la figura 3.1.

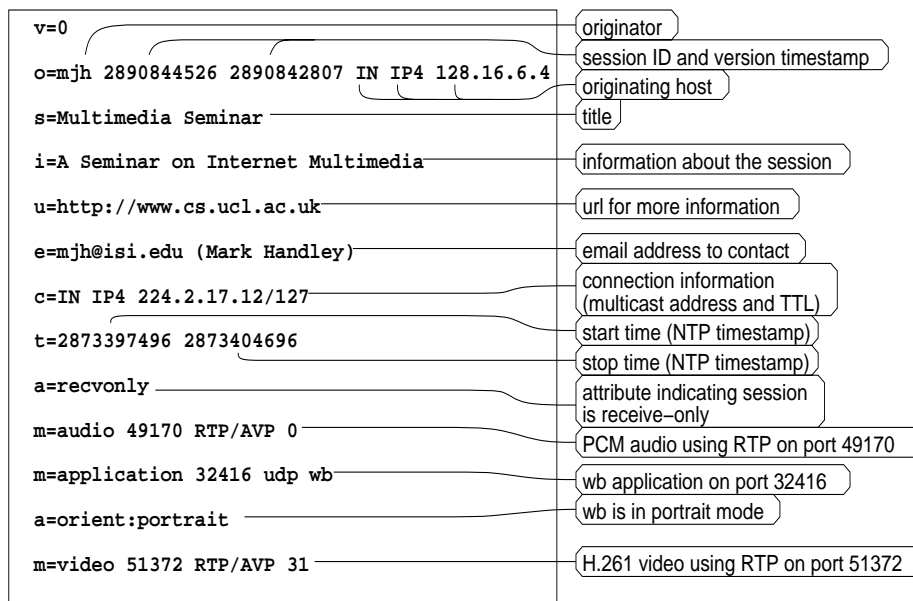


Figura 3.1: Formato básico de una descripción de sesión realizada con SDP

Por lo tanto, tal y como se puede apreciar en la figura 3.1, SDP además de incluir el nombre de la sesión y una descripción de su propósito, define también otros aspectos tan interesantes como la duración, los codecs a utilizar para poder participar en la sesión (direcciones, puertos, formatos, etc.). Además, para permitir la creación de sesiones privadas permite también el envío de información sobre la clave a utilizar en la sesión.

### 3.2.1 La sintaxis de SDP

La sintaxis básica es algo de la forma:

`<type>=<value>`

`<type>` va a ser siempre un sólo carácter y además va a ser sensible a mayúsculas o minúsculas. `<value>` va a ser o un número de argumentos separados por espacios o una cadena con formato libre.

La descripción de una sesión consiste en una descripción a nivel sesión (detalles aplicables a la sesión y a cada uno de los *media* a emplear. Después, puede aparecer de forma opcional varias descripciones a nivel de *media*.

La parte de descripción de la sesión siempre comienza con una línea 'v=' y llega hasta la primera línea de descripción de medios. La descripción de medios comienza con una línea del tipo 'm=' y llega hasta la siguiente descripción de medios o el final de la descripción de la sesión.

El significado de los campos se muestra a continuación. Los campos con '\*' son opcionales.

v= Versión del protocolo  
o= El propietario o creador y el identificador de la sesión  
s= Nombre de la sesión  
i=\* Información de la sesión  
u=\* URI de la descripción  
e=\* dirección de correo electrónico  
p=\* número de teléfono  
c=\* Información de la conexión  
b=\* Información del ancho de banda  
*Una o más descripciones de la duración*  
z=\* Ajustes debidos al Time Zone  
k=\* Clave de cifrado  
a=\* *Cero o más* atributos de sesión  
*Cero o más descripciones de medios*

Cada descripción de tiempo se compone de un campo 't=' seguido opcionalmente de un campo 'r='. La 't=' indica el tiempo que la sesión está activa y la 'r=' indica el número de veces que se va a repetir.

Además, cada descripción de los medios a utilizar se compone de un campo 'm=' seguido de otros campos opcionales que proporcionan información adicional:

m= Nombre del medio y dirección de transporte  
i=\* Título del medio  
c=\* Información para la conexión  
b=\* Información del ancho de banda  
k=\* clave de cifrado  
a=\* *cero o más* atributos de medios

### 3.2.2 La actualidad de SDP

SDP está evolucionando bastante para permitir la introducción de nuevas funcionalidades a la descripción de sesiones. De hecho, esta evolución se ve directamente plasmada en la aplicación que más trabaja con este protocolo (*sdr*).

De hecho, han aparecido muchas versiones de *sdr* últimamente e incluso ya existen parches para soportar el anuncio de sesiones que a su vez contienen una nueva lista de sesiones.

SDP está siendo muy utilizado con el protocolo SAP, SIP, H.232 y RTSP e incluso se ha propuesto para el uso en el contexto de televisión avanzada.

### 3.3 Session Announcement Protocol (SAP)

Para anunciar una sesión multicast, el creador de la sesión simplemente envía por multicast el anuncio de la sesión a la dirección 224.2.127.254 por el puerto UDP 9875. Los anuncios de sesión deben de enviarse con el mismo TTL con el que posteriormente se emitirá la conferencia. Cada paquete SAP llevará como carga útil un paquete SDP de descripción de la sesión que se está anunciando. El formato del mensaje SAP para IPv4 lo mostramos en la figura 3.2.

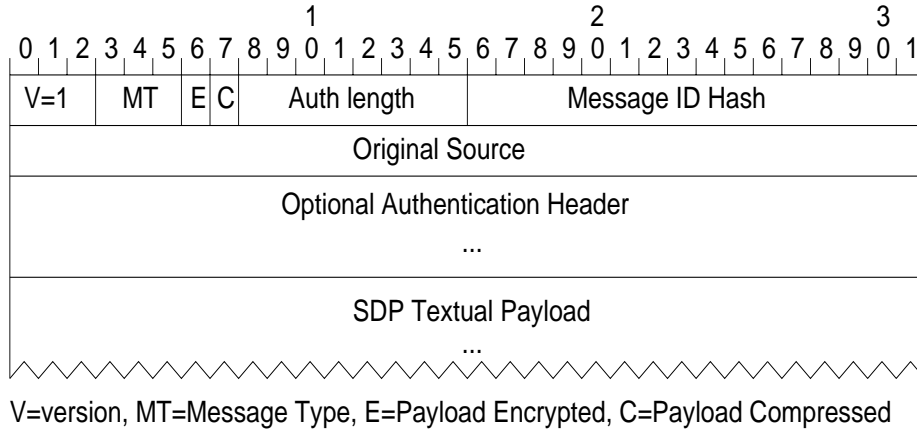


Figura 3.2: Formato del paquete SAP

El campo del tipo de mensaje (MT) indica si el paquete es para anunciar, borrar o modificar la sesión a la que hace referencia. Hay un bit (E) que indica si la carga útil del mensaje va cifrada o no. El bit (C) indica si la carga útil va comprimida o no. Las combinación de los campos *message ID hash* y *original source* permite tener un identificador único de sesión que sirve para identificar esta versión particular de esta sesión.

Los anuncios SAP pueden ser autenticados incluyendo una firma digital de la carga útil en el campo *optional authentication header*. El protocolo SAP soporta actualmente tanto las firmas digitales realizadas con PGP como con PKCS#7 aunque actualmente casi ningún anuncio se autentica. Lógicamente tan pronto como MBone se empiece a emplear con propósitos más serios esta filosofía cambiará.

Del mismo modo, también se pueden enviar los anuncios cifrados, pero esto no significa que esta sea la forma idónea de tener sesiones privadas con pocos participantes pero muy dispersos. Para eso es mejor emplear SIP. Este tipo de anuncios cifrados están pensados para sesiones a gran escala en las que los miembros por ejemplo tuviesen que pagar por participar. De todos modos estas cuestiones de seguridad aún no las tiene totalmente resueltas el protocolo y se están planteando enfoques tales como la integración de tarjetas inteligentes para controlar este tipo de cuestiones.

Debe quedar muy claro que aunque SAP escalará bien para cualquier número de receptores, no lo hará muy bien para un número muy elevado de sesiones. El IETF ya sabe que deberá de buscar alguna alternativa a este protocolo pero mientras tanto, conseguimos que de un resultado bueno incluso con miles de sesiones gracias a la introducción de un sistema para mantener una caché de anuncios últimamente recibidos.

Este protocolo debe emplearse para anunciar sesiones cuyos participantes no conocemos de antemano. Si los conocemos de antemano lo mejor es utilizar SIP (siempre y cuando el número potencial de receptores no sea muy elevado).

### 3.4 Session Initiation Protocol

La idea general en la que se basa SIP es en la localización de determinados usuarios para establecer comunicaciones basándose en consultas a proxies de SIP. Cuando estos proxies reciben una solicitud SIP para un determinado usuario se encargan a nivel local de buscarlo y pasarle la solicitud.

Para ello cada usuario del servicio SIP, además de estar registrado en el proxy de SIP para su organización, debe de disponer de lo que se conoce como *SIP URL* que es muy parecida a la dirección de correo.

La ventaja que tiene este enfoque es que el proxy es capaz de informar al que invita (en caso de que el usuario se encuentre en otro lugar) del nuevo proxy al que tiene que hacer la consulta para contactar con él.

Un diagrama con la secuencia completa de la llamada con SIP se muestra en la figura 3.3.

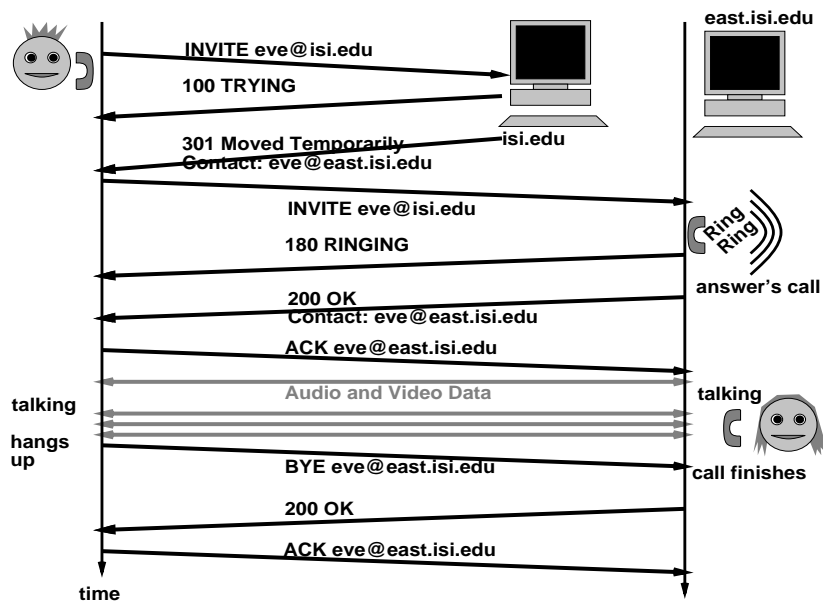


Figura 3.3: Llamada completa empleando SIP

Notar que la aplicabilidad de esta técnica en Mbone es para invitación a sesiones sin embargo, es un planteamiento que se está intentando implantar en otros entornos como por ejemplo telefonía.

## Capítulo 4

# El control de los emisores multicast

En los capítulos anteriores hemos mostrado los aspectos tecnológicos en los que se basa Mbone y sin los que no sería posible entender realmente el contenido y la justificación de las decisiones que se van a hacer a lo largo de este capítulo.

En este capítulo explicaremos en detalle todas las cuestiones relativas a la implementación de la infraestructura de control de emisores multicast que es objeto de este proyecto. Además haremos un repaso a las diferentes soluciones que se plantean así como un estudio de las ventajas e inconvenientes de cada una de las soluciones.

Lo importante es que no debemos olvidar nuestro objetivo en ningún momento: queremos controlar quién emite y con que TTL en el entorno multicast que es la arquitectura de Mbone de la Universidad de Murcia. Del mismo modo, tenemos que tener presentes todas las implicaciones y las tareas que esto conlleva. En concreto:

1. Modificación del soporte para enrutamiento multicast de los kernels 2.0.X y 2.2.X de Linux para permitir el filtrado de datagramas IP multicast.
2. Adición de nuevas llamadas al sistema en el kernel de Linux para permitir la gestión dinámica de los emisores multicast.
3. Modificación del código fuente de *mrouterd*. Como veremos a continuación, una de las soluciones que proponemos pasa por añadir un nuevo fichero de configuración a *mrouterd* para que pueda configurar los emisores multicast en el kernel. Por lo tanto una de las primeras tareas ha sido modificar el código fuente del demonio de enrutamiento multicast más usual: *mrouterd 3.9 beta 3*.
4. Creación de un algoritmo de establecimiento de políticas de control a nivel del kernel del sistema operativo del equipo que haga de MRouter.
5. Creación de un intérprete de SAP/SDP para poder controlar todos los anuncios de sesiones que se producen.
6. Creación de applets de administración remota de todo el sistema.
7. Estudio y puesta en marcha de controles de acceso al administrador en base a certificados digitales u ACLs.

A continuación veremos más en detalle como se implementan estas acciones.

## 4.1 El enrutamiento multicast y su relación con el kernel

Ya en el capítulo 2 vimos el funcionamiento de diversos algoritmos de enrutamiento a nivel de operación, mantenimiento del estado, intercambio de mensajes y construcción de árboles de distribución. Sin embargo, cuando entramos más en detalle a examinar el código fuente de alguno de estos demonios de enrutamiento, observamos que en realidad todo el tráfico no pasa por el demonio. El demonio simplemente escucha los mensajes IGMP y a partir de ahí utiliza las rutinas de *forwarding* del núcleo del sistema operativo. Es por esto que cualquier control o filtrado del tráfico multicast que deseemos realizar va a estar muy ligado al soporte que ofrezca al sistema operativo más que al propio algoritmo de enrutamiento. Es por esto que vamos a tener que trabajar a nivel de las estructuras que muestra la figura 4.1 en la que podemos apreciar la estructura modular de la implementación de la pila de protocolos TCP/IP en el kernel de Linux.

Además, utilizando las posibilidades de forwarding del núcleo del sistema operativo se obtiene la ventaja adicional de que los paquetes se distribuyen de un modo más rápido al necesitar menos tiempo de proceso en la máquina que hace de MRouter.

Pero, ¿Cómo es realmente el soporte que ofrece el sistema operativo a los demonios de enrutamiento multicast?.

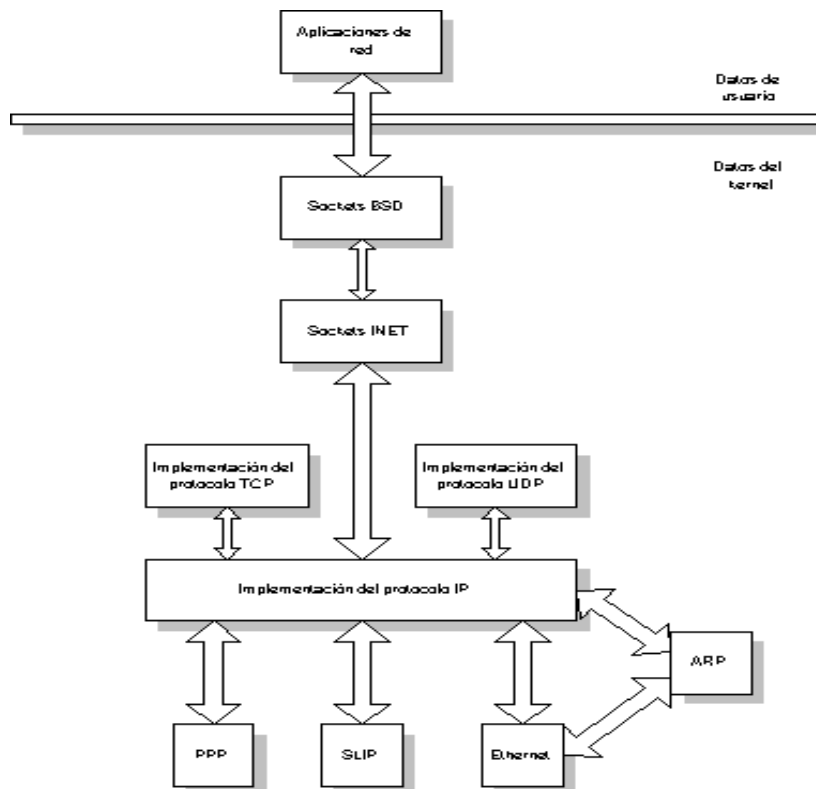


Figura 4.1: Estructura de la pila de gestión de red en Linux

### 4.1.1 Soporte del sistema operativo al enrutamiento multicast

Pues el soporte que ofrece el sistema operativo es en base a llamadas al sistema. Es decir, el sistema operativo ofrece una serie de parámetros extra para la llamada al sistema *setsockopt* que se emplea con sockets multicast.

Los principales parámetros para estas llamadas aparecen reflejados en la tabla 4.1.

System Calls para multicast routing	
Parámetro	Significado
MRT_INIT	Inicialización de la tablas de enrutamiento multicast
MRT_DONE	Finaliza el soporte para enrutamiento multicast
MRT_ADD_VIF	Añade un nuevo interfaz virtual
MRT_DEL_VIF	Elimina un nuevo interfaz virtual
MRT_ADD_MFC	Añade una nueva entrada a la tabla de forwarding
MRT_DEL_MFC	Elimina una nueva entrada de la tabla de forwarding
MRT_PIM	Mensajes de gestión del algoritmo de enrutamiento PIM

Tabla 4.1: Parámetros de las llamadas al sistema para enrutamiento multicast

Notar que todos estos parámetros cambian entre versión y versión del sistema operativo. Los que nosotros mostramos aquí corresponden al Kernel 2.2.2 del sistema operativo Linux.

El algoritmo de enrutamiento multicast en detalle sigue los pasos que se van a comentar a continuación:

1. Abre un socket en modo *raw* (de aquí que sólo se pueda arrancar el demonio con permisos de root) para recibir todos los mensajes IGMP. Hay que tener en cuenta que son estos mensajes los que mantienen informado al demonio de todos los grupos para los que tiene que reenviar tráfico y para los que no.
2. Inicializa las estructuras de enrutamiento y distribución multicast del kernel mediante la llamada al sistema `setsockopt(...MRT_INIT...)`.
3. Lee la configuración desde un fichero y va inicializando las interfaces multicast y los túneles en el kernel mediante llamadas a `setsockopt(...MRT_ADD_VIF...)`.
4. Cada vez que va recibiendo un mensaje *IGMP join* de algún host se actualiza la lista de distribución de tráfico multicast del kernel mediante la llamada del sistema `setsockopt(...MRT_ADD_MFC...)`.
5. Cuando se da cuenta de que deja de haber alguien interesado en un determinado grupo multicast, elimina ese grupo de la lista de distribución multicast del kernel mediante la llamada al sistema `setsockopt(...MRT_DEL_MFC...)`.
6. Cuando se para el demonio, va eliminando todas la interfaces para enrutamiento multicast (VIFS) del kernel mediante la llamada `setsockopt(...MRT_DEL_VIF...)`.
7. Por último, se eliminan todas las estructuras de enrutamiento internas mediante la llamada a `setsockopt(...MRT_DONE...)`.

#### 4.1.2 Problema que nos plantea este tipo de funcionamiento

El problema que aparece es que no existe ningún control sobre los emisores y por lo tanto cualquier usuario que tenga accesible un MRrouter puede emitir el tráfico multicast ya que el MRrouter simplemente retransmite los paquetes por los interfaces virtuales que corresponda. Es precisamente nuestra tarea el establecer un control sobre los emisores multicast de modo que sólo las personas autorizadas puedan generar tráfico de salida y limitando además el ámbito (TTL) con el que puede transmitir un usuario. De este modo por ejemplo, podemos limitar el TTL de los usuarios noveles o poco experimentados a 15 y de este modo podrían recibir todas las sesiones mundiales pero en ningún caso podrían enviar tráfico multicast fuera del ámbito de su organización por lo que no podría saturar los enlaces externos ni interferir en otras conferencias realizadas por otras personas de fuera de su ámbito organizativo.

Sin embargo, esta forma que emplea el MRrouter para realizar los reenvíos de paquetes nos obliga a tener que bajar a un nivel inferior al demonio de enrutamiento multicast para poder tener cierto control sobre los paquetes multicast y poder comprobar realmente que el tráfico que genera el usuario cumple las características que se le exigen en cuanto a TTL y demás. El único control que podemos llevar a cabo a nivel de demonio de enrutamiento multicast, es el decidir si vamos a permitir que un determinado tráfico multicast que llegue por un determinado interfaz virtual va a ser reenviado o no. En ningún caso podemos diferenciar entre usuarios o equipos. Es decir, o habilitamos a todos los host y usuarios de esa subred a emitir a un determinado grupo o no habilitamos a ninguno. Además, caso de habilitar a alguno, cualquiera de los hosts accesibles desde ese interfaz virtual puede emitir con el TTL que les apetezca sin que podamos hacer nada para evitarlo ya que ese control sólo puede llevarse a cabo con el buffer de paquetes IP que mantiene internamente el sistema operativo.

Este enfoque basado en hacer el control a nivel de demonio de enrutamiento multicast tiene la ventaja de que es muy general y podría aplicarse a cualquier equipo que tenga ejecutándose cualquier demonio de enrutamiento multicast independientemente del sistema operativo que tenga. Sin embargo, tiene el inconveniente de que es muy poco elástico y prácticamente inservible de cara al administrador que pretende hacer este control de usuarios multicast dentro de su propia red.

Necesitamos pues mejores soluciones que indudablemente van a pasar por equipos basados en sistemas operativos cuyo código fuente del kernel sea totalmente abierto tales como Linux o FreeBSD.

Actualmente la gestión de los emisores multicast es un tema totalmente novedoso y por estudiar. De hecho, sólo existe una propuesta (que se ha convertido en Internet-draft) que describe una serie de extensiones al protocolo IGMPv2 para soportar la autenticación de emisores multicast [IYT98]. En concreto, hemos realizado un estudio en profundidad de este draft que aparece en la sección 4.7.2

Para resolver el problema descrito, pueden plantearse diferentes soluciones cada una de ellas con sus ventajas y sus inconvenientes. Todas estas soluciones tienen un denominador común que nos viene impuesto por la forma en la que se realiza el reenvío de los paquetes: la necesidad de modificar el soporte que da el sistema operativo a los demonios de enrutamiento multicast. Básicamente, se trata de controlar cuando un determinado paquete multicast debe de ser reenviado o no. En concreto, un datagrama multicast sólo se reenviará cuando cumpla una serie de condiciones dependientes del usuario o host que lo envía.

## 4.2 Modificaciones al Kernel del sistema operativo

Como acabamos de ver, para poder realmente controlar los paquetes multicast que están circulando por nuestra red y que están atravesando nuestro MRrouter, no tenemos otra opción que controlar los datagramas IP a nivel del kernel del sistema operativo del host que hace de MRrouter.

De este modo, y más concretamente, mediante el manejo de buffer de datagramas IP que implementa el sistema operativo Linux, podemos comprobar si un determinado datagrama debe enviarse o no. Para decidir esto, la opción más factible pasa por comprobar los datos que incluya la cabecera IP de dicho datagrama. Una versión antigua de las estructuras de este buffer puede encontrarse en [Rus98].

Nuestro entorno de trabajo en cuanto a la modificación realizada en el kernel ha sido un PC con la distribución RedHat 5.2 del sistema operativo Linux. Esta distribución de Linux, ofrece la versión 2.0.36 del Kernel que es precisamente sobre la que se comenzó a trabajar. Sin embargo, a la hora de estudiar, analizar y modificar el esquema de enrutamiento multicast del kernel descubrimos varios errores. Tras consultar las nuevas versiones del kernel, observamos que estos problemas ya se habían resuelto con la aparición del kernel 2.2.2 y por lo tanto decidimos migrar a esta última versión.

El problema que nos aparece con el cambio de kernel es que las modificaciones que se estaban planteando para 2.0.36 no son directamente aplicables al kernel 2.2.2 sobre todo debido a que a partir de la versión 2.2.0 cambia el modo en el que se realiza la copia de datos entre el espacio de datos del usuario y el espacio de datos del kernel.

En concreto, en la versión 2.0.36 del kernel, esta copia se hace con la ayuda de dos llamadas al sistema: primero se hace un *verify\_area()* y a continuación si esta llamada no devuelve ningún error se hace un *memcpy\_fromfs(...)*.

Sin embargo, a partir de la versión 2.2.0 del kernel, una sólo llamada al sistema es más que suficiente para hacer dicha copia. Consiste en hacer un *copy\_from\_user(...)*.

Nuestra modificación al kernel del sistema operativo consiste en la adición de unas estructuras de datos que permitan el almacenamiento de los parámetros asociados a cada uno de los emisores multicast para posteriormente poder hacer las comprobaciones necesarias que aseguren que los datagramas que se reenvían en el MRrouter se ajustan a dichos parámetros.

De este modo, se va a mantener en todo momento un registro de los emisores multicast autorizados junto a los grupos por los que están autorizados a emitir e incluso el TTL máximo con el que van a poder emitir. Y para complementar y hacer uso de estas estructuras se han definido también una serie de funciones que se encargan de hacer el filtrado de tráfico generado por los emisores multicast así como permitir la reconfiguración dinámica tanto de los emisores multicast registrados como de sus parámetros asociados.

Las nuevas llamadas al sistema que hemos implementado para la realización de la reconfiguración dinámica son las que se muestran en la tabla 4.2

Nuestras llamadas al sistema	
Parámetro	Significado
MRT_SENDER_INIT	Inicialización de las estructuras de emisores multicast
MRT_SENDER_ADD	Registro de un nuevo emisor multicast en el kernel
MRT_SENDER_DEL	Eliminación de un emisor multicast del kernel

Tabla 4.2: Nuestras llamadas al sistema para la gestión de emisores multicast

Independientemente de la versión del kernel que se utilice, debemos de definir una serie de estructuras internas de manejo de datos que nos permitan gestionar qué usuarios o hosts pueden enviar a qué grupos multicast y con qué TTL máximo. En concreto se ha modificado el módulo de definición de estructuras internas del kernel para enrutamiento multicast añadiendo la definición de la nueva estructura de gestión de emisores multicast. Esta estructura, que hemos denominado *m sender\_ctl*, se encarga de almacenar los parámetros necesarios para controlar el tráfico que pueda generar un determinado emisor multicast. El formato y la información que almacena esta estructura aparece reflejado a continuación:

```
struct msender_ctl {
    __u32 src; /* Dirección IP del emisor*/
    __u32 dst; /* Grupo multicast destino*/
    unsigned char ttl;
};
```

El siguiente paso una vez que hemos realizado las modificaciones necesarias al soporte multicast del sistema operativo y hemos creado nuestras propias llamadas al sistema para disponer de una configuración dinámica de los emisores multicast, es el comprobar para cada datagrama multicast si realmente cumple los parámetros que tiene asignado el emisor en la estructura *m sender\_ctl*. Para ello, lo primero que debemos de hacer es definir una política de control.

La política de control que hemos implementado consiste en descartar todo el tráfico multicast excepto el que provenga de un origen registrado en el kernel y que no viole los parámetros definidos para ese origen en la estructura *msender\_ctl* que le corresponde en el kernel del sistema operativo del MRrouter.

Esta política nos asegura que ningún usuario de la subred multicast destinada a uso público va a poder emitir tráfico a algún grupo multicast al que no le esté permitido. De este modo se evita que unos usuarios puedan emitir a sesiones a las que no tienen permiso y por lo tanto no puedan molestar o interferir esas otras sesiones de forma malintencionada. Es decir, con este enfoque sólo los usuarios que nosotros autorizamos de forma explícita para emitir a un determinado grupo multicast y con un determinado TTL podrán hacerlo. Y además, esto nos da una cierta confianza ya que sólo permitiremos el empleo de TTL superior a 15 a las personas que consideremos experimentados o de confianza mientras que a los más inexpertos o de los que podamos desconfiar no podrán emitir fuera de la red local porque se les va a asignar un TTL de 15 o menor.

Sobre todo el aspecto del control del TTL es uno de los más importantes. De hecho, con esta política nos aseguramos de que a cualquier emisor que intente emitir datagramas multicast con un TTL superior al permitido se le descarten los paquetes.

Una vez definida la política de gestión de datagramas y emisores multicast, lo que realmente hace falta es implementarla y ponerla en marcha. De hecho, para esto se definió una rutina interna al kernel que, si bien es muy sencilla en cuanto a la descripción verbal de su funcionamiento, también es bastante compleja en cuanto a su implementación. Más concretamente, esta rutina se encarga de ir al buffer de datagramas IP que almacena el soporte para TCP/IP del kernel de linux (*sk\_buff*), tomar el datagrama multicast, comprobar su origen, buscar ese origen en nuestra estructura interna de almacenamiento de parámetros de emisores multicast, comprobar que realmente el datagrama cumple con los parámetros allí definidos para dicho usuario multicast y en caso de que así sea, reenviar el paquete. Lógicamente, si alguna de las comprobaciones anteriores falla, el datagrama se elimina de la cola.

Todo este proceso se podría resumir con el esquema que muestra la figura 4.2.

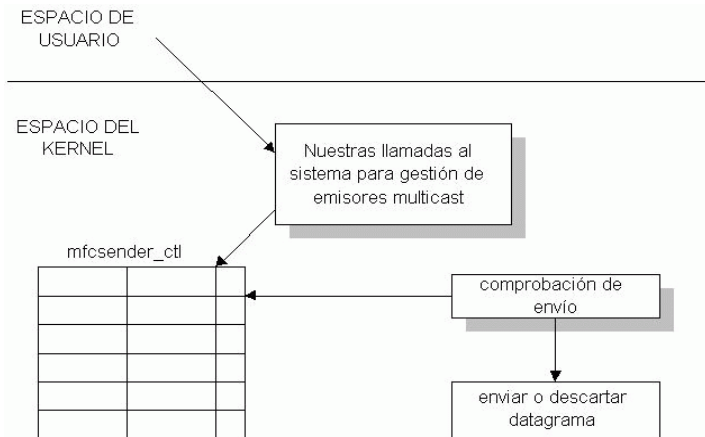


Figura 4.2: Arquitectura general del kernel modificado

La pregunta que surge ahora es, ¿Como ayudan estas modificaciones al kernel a la consecución de nuestro objetivo?.

Pues más que ayudar a conseguir nuestro objetivo es que prácticamente lo consiguen. De hecho, simplemente con estas modificaciones al esquema de enrutamiento multicast del kernel ya disponemos de los mecanismos básicos para controlar de un modo exhaustivo a los emisores multicast. Una vez tenemos este soporte básico o este valor añadido que ofrecemos desde el kernel, ya podemos asegurar que tenemos resuelto el problema a bajo nivel, es decir, podemos controlar el tráfico multicast. Ahora lo que se plantea es cual

debe ser la estructura idónea que debemos de ofrecer al administrador del sistema para que pueda gestionar a los emisores multicast de un modo sencillo y flexible.

Ahora nos falta ofrecer una serie de soluciones al problema que se basen en este soporte que nos acabamos de crear en el kernel. La forma más sencilla de hacer esto es evidentemente mediante el uso de las llamadas al sistema que nos hemos creado.

Sin duda alguna, el éxito del proyecto se debe en gran medida a que ya desde un principio se tuvieron muy claros los requerimientos que eran necesarios por parte del kernel y desde el principio se tomó la modificación del kernel de un modo ambicioso. Esto nos ha permitido crearnos unas llamadas al sistema que aportan muchas ventajas además de dotar de una gran flexibilidad a cualquier otro mecanismo de nivel superior que haga uso de ellas. De hecho, la gran ventaja que nos proporcionan estas llamadas es que nos permiten una configuración dinámica de los parámetros de los usuarios sin necesidad de rearrancar el demonio de enrutamiento.

### 4.3 Detalle de las modificaciones al kernel

En la sección anterior vimos de un modo algo detallado las modificaciones que se han realizado al kernel del sistema operativo. Con el contenido del apartado anterior, el lector tiene conocimiento suficiente como poder obviar esta sección. Sin embargo, hemos creído conveniente mostrar las modificaciones totalmente detalladas para que todo el trabajo realizado pueda ayudar a alguien que en un futuro tenga la necesidad de hacer algo similar.

Si la documentación existente sobre alguno de los kernels más o menos actuales de Linux es escasa, ni que decir tiene que la información sobre la implementación que hace el kernel del soporte para enrutamiento multicast es nula. Es por esto, que todo lo que hemos aprendido sobre el enrutamiento multicast del kernel ha sido a base de leer código fuente del kernel.

Lo primero era averiguar los ficheros del kernel que implementan todo el soporte multicast. Tras un tiempo de búsqueda intensiva entre los fuentes de kernel, averiguamos que los ficheros implicados eran:

```
/usr/src/linux/include/linux/mroute.h
/usr/src/linux/net/ipv4/ipmr.c
```

La siguiente fase era realizar un estudio detallado de estos ficheros para comprender el funcionamiento de todas las funciones y definiciones que incluyen. El primero de los dos ficheros anteriores contiene todas las definiciones de las estructuras de datos y macros involucradas con el enrutamiento multicast. En concreto aparecen las definiciones de la tabla de rutas multicast, lista para control de los VIFS, definición de los parámetros asociados a un VIF, y una de las partes más importantes que es la definición de los parámetros a emplear por las aplicaciones del espacio de usuario (como por ejemplo el demonio de enrutamiento multicast) a la hora de acceder a los servicios proporcionados por esta parte del kernel.

En concreto, la lista de definiciones estándar es la siguiente:

```
#define MRT_BASE          200
#define MRT_INIT          (MRT_BASE)      /* Activate the kernel mroute code */
#define MRT_DONE          (MRT_BASE+1)    /* Shutdown the kernel mroute */
#define MRT_ADD_VIF       (MRT_BASE+2)    /* Add a virtual interface */
#define MRT_DEL_VIF       (MRT_BASE+3)    /* Delete a virtual interface */
#define MRT_ADD_MFC        (MRT_BASE+4)    /* Add a multicast forwarding entry */
#define MRT_DEL_MFC        (MRT_BASE+5)    /* Delete a multicast forwarding entry */
```

```

#define MRT_VERSION      (MRT_BASE+6)    /* Get the kernel multicast version */
#define MRT_ASSERT      (MRT_BASE+7)    /* Activate PIM assert mode */

```

A estos parámetros, nosotros tuvimos que añadir los nuestros propios. Para lo cual, lo primero que tuvimos que hacer fue ir buscando valores libres para nuestras llamadas. Tal y como se aprecia a continuación, al final nos quedamos con las que van desde el 207 hasta el 210. Notar que en FreeBSD estas definiciones comienzan a partir del 100 por cuestiones internas de implementación. Sin embargo, lo importante más que el valor, es nombre que se les da. En este sentido, todos los sistemas operativos han de emplear el mismo para que el soporte al algoritmo de enrutamiento multicast sea homogéneo.

```

#define MRT_SENDER_INIT (MRT_BASE+8)    /* Inicializar emisores */
#define MRT_SENDER_DEL  (MRT_BASE+9)    /* Añadir una entrada de emisor */
#define MRT_SENDER_ADD  (MRT_BASE+10)   /* Borrar una entrada de emisor */

```

Como ya hemos explicado en el apartado anterior, estas llamadas se encargan de llevar el control de la lista de emisores multicast y permiten que de forma dinámica la aplicación del espacio de usuario (en nuestro caso *musersd*) pueda ir añadiendo y eliminando emisores multicast.

Con esto, simplemente estamos dando opción a que se pasen estos parámetros en llamadas a *setsockopt*. Más adelante veremos como tenemos que programar la parte que se encarga de atender a esas llamadas.

Otra estructura interesante que se define en este primer fichero bastante, es la encargada de almacenar las lista actual de *forwards* multicast que está haciendo el kernel. En concreto la estructura se llama *mfc\_cache* y se muestra a continuación:

```

struct mfc_cache
{
    struct mfc_cache *next;           /* Next entry on cache line */
    __u32 mfc_mcastgrp;              /* Group the entry belongs to */
    __u32 mfc_origin;                /* Source of packet */
    vifi_t mfc_parent;               /* Source interface */
    struct timer_list mfc_timer;     /* Expiry timer */
    int mfc_flags;                   /* Flags on line */
    struct sk_buff_head mfc_unresolved; /* Unresolved buffers */
    int mfc_queuelen;                /* Unresolved buffer counter */
    unsigned long mfc_last_assert;
    int mfc_minvif;
    int mfc_maxvif;
    unsigned long mfc_bytes;
    unsigned long mfc_pkt;
    unsigned long mfc_wrong_if;
    unsigned char mfc_ttls[MAXVIFS]; /* TTL thresholds */
};

```

Además de todas las estructuras que aparecen en este fichero, tuvimos que definir nuestra propia estructura de gestión de emisores multicast. Como ya hemos comentado en el apartado anterior, hemos denominado a esta estructura *msernder\_ctl* y presenta los siguientes campos:

```

struct msender_ctl {
    __u32          src;      /* Direccion IP del emisor */
    __u32          grp;      /* Grupo multicast destino */
    unsigned char  ttl;      /* TTL asociado a ese emisor */
};

```

Esta estructura nos permite almacenar los parámetros que se emplean en la versión actual para limitar la capacidad emisora de un usuario en concreto.

Si bien, este fichero es muy importante, donde realmente se han realizado más modificaciones es en el segundo de los anteriormente mencionados (*ipmr.c*).

El primer problema que se nos planteó en un principio en cuanto a la modificación del kernel, fue a la hora de que el kernel recibiese información desde el espacio de usuario para poder asociar los parámetros a un usuario multicast y posteriormente filtrar su tráfico.

Debido al entorno de programación tan limitado que es el kernel en el que no disponemos de muchas funcionalidades de alto nivel, como abrir un fichero pasando como argumento un string ni nada de eso, estuvimos evaluando diferentes modos de conseguir ese paso de información.

Al final, por homogeneidad con lo ya existente, decidimos pasar los emisores multicast al kernel del mismo modo que por ejemplo se pasan los reenvíos a realizar. Es por esto que nos creamos las tres llamadas que hemos comentado antes: *MRT\_SENDER\_INIT*, *MRT\_SENDER\_ADD* y *MRT\_SENDER\_DEL*.

Antes hemos visto la parte de definición de las llamadas en el fichero *mroute.h*. Sin embargo, ahora vamos a tratar la implementación que debemos de hacer en el kernel para dar soporte a estas nuevas llamadas. En concreto, debemos de añadir funcionalidad a la llamada *ip\_mroute\_setsockopt(...)* que hay en el fichero *ipmr.c*. Esta funcionalidad que añadimos se encargará de tomar los parámetros del espacio de usuario y copiarlos a espacio del kernel para a continuación poder trabajar con estos parámetros. El fragmento de código que muestra esta copia entre espacio de usuario y kernel se muestra a continuación:

```

.....
        case MRT_SENDER_INIT: {
            return (ip_mr_sender_init());
        }

        case MRT_SENDER_DEL:
        case MRT_SENDER_ADD:
        {
            printk("Uno de los dos es");
            if (optlen!=sizeof(ctl)) {
                printk("Tamaño distinto\n");
                return -EINVAL;
            }
            if (copy_from_user(&ctl,optval, sizeof(ctl))) {
                printk ("Error copy_from_user\n");
                return -EFAULT;
            }
            return (ip_mr_trata_sender(optname,&ctl));
        }
.....

```

Como vemos en los últimos dos casos que son los realmente interesantes, lo que hacemos es emplear la función que describimos en la sección anterior: *copy\_from\_user*.

De este modo, en este trozo de código se toma el registro del tipo *msender\_ctl* que envía la aplicación de nivel de usuario y se copian esos campos en variables declaradas en la zona del kernel como en este caso la variable *ctl*. Una vez hecha la copia, llamamos a una función que nos hemos creado (*ip\_mr\_trata\_sender*) pasándole estos parámetros y esta función se encarga de añadir el nuevo emisor multicast o eliminarlo dependiendo del campo *optname* que puede ser *MRT\_SENDER\_ADD* o *MRT\_SENDER\_DEL*.

El código de esta función también lo mostramos a continuación:

```
int ip_mr_trata_sender(int optname, struct msender_ctl *mp)
{
    if (optname==MRT_SENDER_DEL) {
        return ip_mr_sender_del(mp);
    }
    else {
        return ip_mr_sender_add(mp);
    }
}
```

Las funciones *ip\_mr\_sender\_del* e *ip\_mr\_sender\_add* las hemos construido para gestionar la lista de emisores multicast autorizados. Como ejemplo de alguna de estas funciones a continuación mostramos el código correspondiente a la función *ip\_mr\_sender\_del(...)*:

```
int ip_mr_sender_del(struct msender_ctl *mp)
{
    int i,j;

    for (i=0; i<mfc_num_msender; i++) {
        if ((mfc_msender_tbl[i].src == mp->src) &&
            (mfc_msender_tbl[i].grp == mp->grp) &&
            (mfc_msender_tbl[i].ttl == mp->ttl)) break;
    }

    if (i == mfc_num_msender) return (ENOENT); /* No encontrado */

    for (j=i; j<mfc_num_msender-1;j++) {
        mfc_msender_tbl[j].src = mfc_msender_tbl[j+1].src;
        mfc_msender_tbl[j].grp = mfc_msender_tbl[j+1].grp;
        mfc_msender_tbl[j].ttl = mfc_msender_tbl[j+1].ttl;
    }
    mfc_num_msender--;

    return(0);
}
```

Como vemos, primero va recorriendo la lista de emisores multicast llamada *mfc\_msender\_tbl* hasta encontrarlo. Si no lo encuentra devuelve un error. En caso contrario lo elimina y devuelve como resultado un 0.

La otra función es muy parecida. Primero busca para ver que no existe ya la entrada y la inserta. Si la entrada ya existía entonces sale sin añadir la nueva entrada.

Otra función muy interesante dentro toda esta serie de funciones que estamos añadiendo al kernel y la que realmente nos permite resolver todo el problema que se plantea, es la encargada de comprobar para cada datagrama multicast si debe de reenviarse o no.

Para tomar esta decisión comprobamos si ese datagrama concreto cumple los parámetros que le estamos exigiendo a todo el tráfico que provenga de ese emisor.

En realidad lo que vamos a hacer es construir una función que dado un puntero a la posición que ocupa el datagrama IP dentro de la lista de datagramas multicast a reenviar que guarda el kernel, decida, contrastando los parámetros del datagrama con los de la lista de emisores multicast, si ese datagrama debe enviarse o no.

Para hacer esta función tan simple en cuanto a código, lo primero que necesitamos es conocer la estructura del buffer de datagramas IP que emplea el kernel de linux y después ya podemos hacer la comparación.

La declaración del buffer aparece en el fichero

```
/usr/src/linux/include/linux/sk_buff.h
```

Una vez ya hemos visto la estructura del buffer de datagramas IP, ya podemos hacer la función que comprueba si el paquete debe enviarse o no. El código se muestra a continuación:

```
int check_ttl(struct sk_buff *skb) {

    int i;
    u_char auxttl;
    __u32 auxsrc;
    __u32 auxgrp;

    auxttl = skb->nh.iph->ttl;    /* Acceso al datagrama IP */
    auxsrc = skb->nh.iph->saddr;
    auxgrp = skb->nh.iph->daddr;

    for (i=0; i<mfc_num_msender; i++) {
        if ((mfc_msender_tbl[i].src == auxsrc) &&
            (mfc_msender_tbl[i].ttl >= auxttl)) return(FW_OK);
    }
    return(FW_NG); /* Forward no permitido (Not Granted) */
}
```

Esta función se llama desde la función *ip\_mr\_forward(...)* que es la que realmente se encarga de llamar a la función *ipmr\_queue\_xmit(...)* que a su vez es la que se encarga realmente de enviar el datagrama multicast. El fragmento de código que controla el reenvío se muestra a continuación:

```
int ip_mr_forward(struct sk_buff *skb, struct mfc_cache *cache, int local)
{
    ....
    for (ct = cache->mfc_maxvif-1; ct >= cache->mfc_minvif; ct--) {
        if (check_ttl(skb)==FW_NG) goto dont_forward;
        if (skb->nh.iph->ttl > cache->mfc_ttls[ct]) {
            if (psend != -1)
                ipmr_queue_xmit(skb, cache, psend, 0);
            psend=ct;
        }
    }
    if (check_ttl(skb)==FW_NG) goto dont_forward;
    if (psend != -1)
```

```

        ipmr_queue_xmit(skb, cache, psend, !local);

dont_forward:
    if (!local)
        kfree_skb(skb);
    return 0;
}

```

Como vemos, es justo antes de copiar el datagrama a la cola de datagramas a transmitir comprobamos si realmente debe de enviarse llamando a *check\_ttl*.

En la figura 4.3 puede apreciarse un esquema global de los elementos que intervienen en los cambios que hemos hecho al kernel.

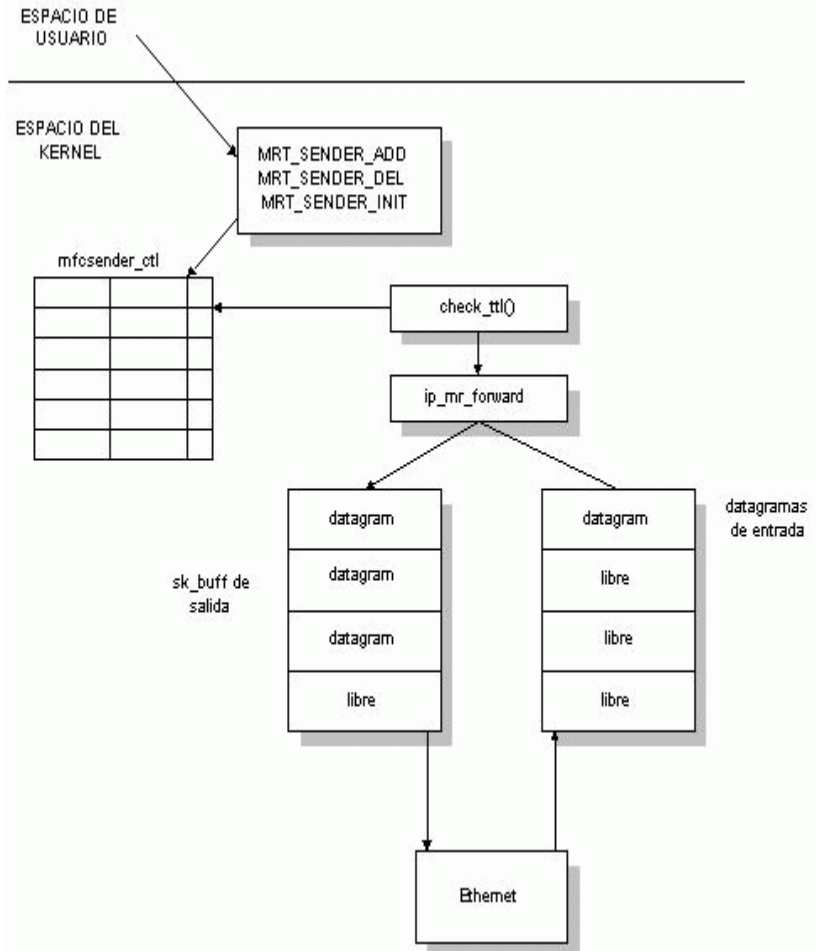


Figura 4.3: Esquema del funcionamiento del kernel modificado

## 4.4 Una primera solución

Una vez que ya tenemos construido el soporte básico para controlar el cumplimiento de la política que acabamos de definir, la siguiente fase del proyecto pasa por integrar todo este mecanismo en un sistema real y ponerlo a prueba. Se trata pues de montar un verdadero esquema que permita ofrecer aulas de acceso libre a Mbone. Es decir, una primera solución totalmente implantable.

Esta primera solución de la que estamos hablando consiste en realizar la modificación del código fuente del demonio de enrutamiento multicast (que en nuestro caso es *mrouterd 3.9 beta 3*) para que lea de un fichero de configuración los detalles de los usuarios a los que debe de autorizar. De este modo será el demonio de enrutamiento el que emplee nuestras llamadas al sistema de la tabla 4.1 para introducir en las estructuras del kernel los usuarios que lea del fichero de configuración. Del mismo modo, al finalizar el demonio vuelve a dejar las estructuras de enrutamiento multicast tal y como estaban.

¿Como se produce en este caso la actualización de la lista de emisores?. Pues la solución es modificar el fichero de configuración (que en nuestro caso se llama *mauth.conf*, rearrancar el demonio de enrutamiento multicast para que vuelva a leer el fichero y esperar a que comience a enrutar de nuevo. Es decir, tan simple como un *kill -HUP*.

El formato del fichero de configuración *mauth.conf* es muy sencillo. Se trata de un fichero de texto en el que vamos a ir introduciendo una línea por cada emisor multicast y separando cada columna por un tabulador. Un ejemplo de configuración es el que mostramos a continuación:

```
155.54.95.102 224.2.2.2      15
155.54.95.101 224.232.221.122 255
155.54.95.103 226.184.123.71  64
```

Como vemos, con esta configuración estamos diciendo que el equipo con dirección IP 155.54.95.102 sólo pueda emitir al grupo multicast 224.2.2.2 con un TTL máximo de 15. Cualquier otro datagrama que venga de este host y vaya dirigido a cualquier otro grupo multicast o bien lleve un TTL superior a 15 será descartado. La configuración que se establece para los otros dos equipos se explica de un modo similar. Los detalles de las modificaciones al demonio de enrutamiento multicast *mrouterd* pueden apreciarse en el apéndice D

De este modo, la puesta en marcha de esta solución presenta un aspecto similar al mostrado en la figura 4.4.

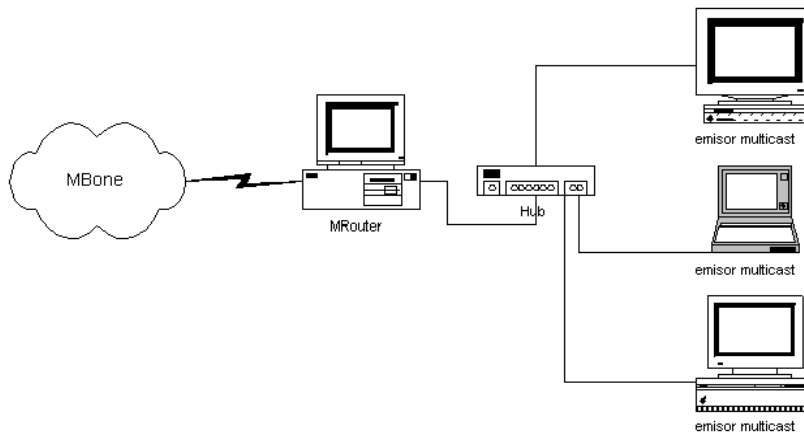


Figura 4.4: Arquitectura con la primera solución

Como podemos ver, el MRouter local se encuentra dando servicio a toda la subred donde se sitúan los usuarios sobre los que queremos realizar el control de tráfico multicast. De este modo cualquier datagrama multicast que vaya a salir de esa red pasará por el MRouter que será el que hará dicho control en base a los parámetros leídos del fichero de configuración *mauth.conf*<sup>1</sup>

<sup>1</sup>Es importante destacar el hecho de que este es el fichero de configuración para el control de usuarios. Además de este que hemos creado nosotros, *mrouterd* incorpora su propio fichero de configuración para especificar la configuración de los túneles, interfaces, etc que se llama *mrouterd.conf*

### 4.4.1 Ventajas e inconvenientes

La principal ventaja que aporta esta solución es muy clara: simplifica mucho el proceso de gestión. Se trata simplemente de editar un fichero de texto, cuya sintaxis además es muy sencilla, y reentrancar el demonio de enrutamiento con un simple comando.

Lógicamente, este enfoque presenta varios inconvenientes que podríamos resumir en que no aprovecha las ventajas que nos aporta el haber realizado las modificaciones en el núcleo del sistema operativo. Tenemos unas llamadas al sistema que soportan una configuración dinámica pero sin embargo aquí la configuración se hace de un modo totalmente estático.

## 4.5 Una solución mejorada

Los problemas que aparecen en la solución anterior se deben a la ligadura que se establece entre el demonio de enrutamiento multicast y el proceso de definición de los usuarios y sus parámetros. Para tratar de evitar estos problemas en la medida de lo posible vamos a introducir un enfoque distribuido que nos permita aislar la parte de administración de los emisores multicast de la parte de enrutamiento multicast propiamente dicho. Además, como ventaja adicional, este enfoque nos va a permitir la posibilidad de hacer que nuestra arquitectura sea totalmente independiente del demonio de enrutamiento multicast que se utilice.

Esta independencia se consigue gracias a las ventajas que aportan las llamadas al sistema que hemos creado. Estas ventajas podrían concretarse como:

1. Nuestras llamadas al sistema nos permiten insertar y eliminar emisores multicast sin necesidad de reentrancar nada y sin embargo con el enfoque anterior se nos obliga a reentrancar el demonio de enrutamiento multicast para que tenga efecto cada modificación de los parámetros asociados a los usuarios.
2. Nuestras llamadas al sistema nos proporcionan independencia del demonio de enrutamiento multicast que se emplee y nos evitan la tarea de ir modificando el código fuente de ningún demonio de enrutamiento multicast concreto para que vaya leer los usuarios y sus parámetros de un fichero de configuración.

Para conseguir la arquitectura independiente del demonio de enrutamiento multicast de la que estamos hablando, vamos a crearnos nuestro propio demonio de gestión de usuarios multicast frente al kernel. A este demonio le llamaremos *musersd* (Multicast users daemon).

La función del *musersd* va a ser la de permanecer indefinidamente escuchando a los comandos que le vayan llegando dirigidos a un determinado puerto UDP de la máquina y posteriormente procederá a ejecutarlos.

Lógicamente, los comandos que puede recibir son añadir un nuevo emisor multicast junto con sus parámetros al kernel o eliminar un emisor multicast del kernel. Notar que no aparece el comando de inicialización de las estructuras internas del kernel ya que este proceso lo realizará de forma automática el *musersd* al arrancar.

Pero, ¿quién envía los comandos al *musersd*?. Pues en el otro extremo de la comunicación con el *musersd* vamos a situar un applet de administración que va a ser la interfaz entre el administrador y el kernel de MRouter. El administrador del sistema de control de usuarios será el encargado de autenticar o autorizar a cada uno de los emisores multicast frente al kernel del MRouter usando para ello el interfaz amigable que le proporciona el applet de administración. La interfaz que ofrece este applet se puede observar en la figura 4.5.

A esta parte del applet es a la que vamos a llamar *mauthclient*. Acabamos de mencionar que vamos a tener dos entidades llamadas *musersd* y *mauthclient* comunicadas por un socket pero no hemos dicho nada del protocolo que van a emplear para comunicarse.

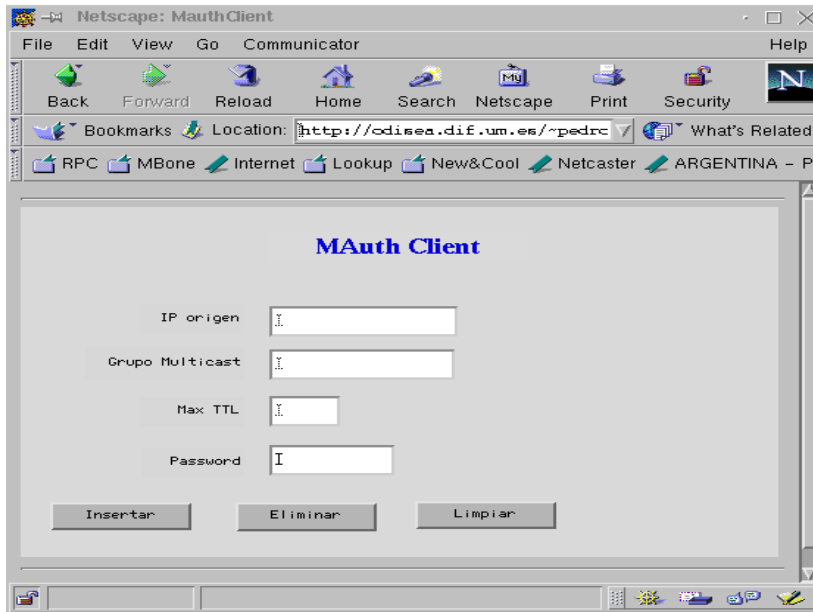


Figura 4.5: El applet de gestión de usuarios

Es por esto que lo primero que vamos a hacer es definir el protocolo de comunicación para estas dos entidades.

#### 4.5.1 Protocolo de comunicación entre *mauthclient* y *musersd*(Mcontrol)

Dado que podemos interpretar como que el *musersd* hace de servidor para que el *mauthclient* pueda registrar a los emisores multicast, proponemos un protocolo del tipo *call/response* en el que el *mauthclient* envía las solicitudes y el *musersd* le responde bien sea con un asentimiento o con un error indicando que no se ha podido llevar a cabo con éxito el comando solicitado. El esquema básico que corresponde a este tipo de comunicación aparece en la figura 4.6.

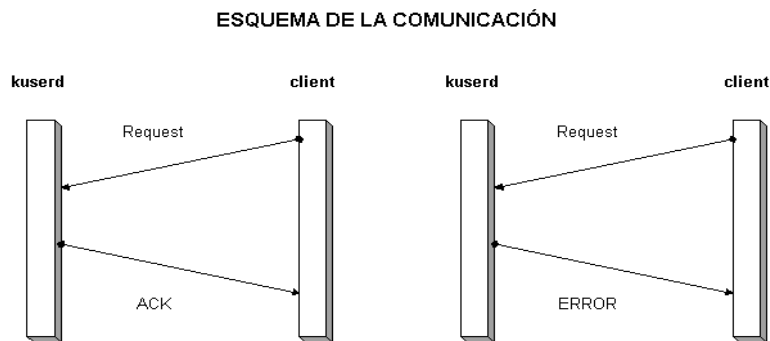


Figura 4.6: Protocolo de comunicación Mcontrol entre el *mauthclient* y el *musersd*

Debido al carácter eminentemente no orientado a la conexión que tiene la comunicación, se optó por trabajar con UDP pese a que resulta un poco más complicada la tarea de programación. De hecho, para evitar comportamientos anómalos en el funcionamiento del protocolo se han introducido algunos campos extra en cada mensaje que permitan el control de las pérdidas de paquetes, pérdidas de los propios asentimientos así como el establecimiento de temporizadores para la gestión de los timeouts. En concreto, para

evitar los problemas que acarrearán las pérdidas de paquetes se emplea por un lado un *número de secuencia* asociado a cada mensaje para poder detectar la falta de algún paquete intermedio y por otro lado se emplean temporizadores para reenviar los mensajes en caso de no recibir respuesta en un determinado lapso de tiempo.

Es importante destacar el matiz eminentemente integrador que presenta aquí el protocolo de comunicaciones definido. No hay que olvidar que estamos comunicando un servidor llamado *musersd* que está totalmente programado en lenguaje C con un cliente llamado *mauthclient* que se encuentra totalmente programado en Java. Necesitamos por lo tanto tener en cuenta mecanismos de conversión entre representaciones, tipos de datos, etc.

Por ejemplo, mientras que en C solemos trabajar con las direcciones IP simplemente usando los 4 bytes que las componen<sup>2</sup>, en Java se trabaja con instancias de la clase *InetAddress* y es por esto que es necesario hacer una transformación de los tipos en la parte Java a una ristra de bytes que van a componer el paquete UDP tanto a la hora de enviar datos como al recibirlos. De hecho, incluso las instrucciones de creación del socket UDP son totalmente diferentes.

Otro aspecto importante que ha habido que tener en cuenta ha sido precisamente la implementación de una semántica de llamada al más puro estilo RPC (*Remote Procedure Call*). La semántica que nos interesa implementar en nuestro caso corresponde a lo que en RPC se llamaría semántica *at most once*. Es decir, debemos de controlar que el hecho de que llegue varias veces una misma solicitud al servidor, no suponga más de una ejecución en el servidor para evitar efectos laterales.

Para evitar este efecto indeseado, se ha recurrido a la típica ventana deslizante que en nuestro caso va a ser de tamaño uno. Es decir, el *musersd* va a mantener en memoria la última solicitud que le llegó así como la respuesta o el resultado asociado a la ejecución del comando que venía en esa respuesta. De este modo puede responder directamente a esa solicitud sin necesidad de ejecutar el comando de nuevo. El formato de los mensajes de solicitud y respuesta empleados en esta comunicación aparece en la figura 4.7.

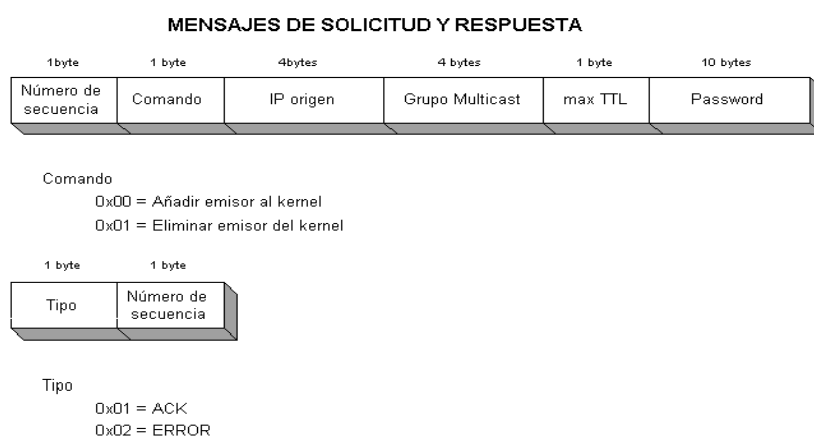


Figura 4.7: Formato de los mensajes de solicitud y respuesta de Mcontrol

Como vemos, hay un campo dentro del mensaje de solicitud que es un password que nos permite autenticar al usuario. Esto se hace para que el *musersd* no acepte cualquier mensaje dirigido a ese puerto sin que lleve un determinado password. Además, se pueden proponer diferentes esquemas para proteger el password basados en técnicas de cifrado cuya implantación sería muy simple.

Además, también se hemos conseguido hacer la autenticación en base a certificados X.509 y tarjetas inteligentes. Para ello bastaría con emplear un servidor Web seguro

<sup>2</sup>Recordar que nos estamos limitando a suponer IPv4

basado en SSL (*Secure Socket Layer*), definirle una ACL (*Access Control List*) para acceso a la página Web donde está el applet de administración y de este modo controlar que sólo pueda conectarse a esa página Web la persona que se identifique con su carnet inteligente. De este modo se consigue una autenticación fuerte basada en la posesión (la tarjeta) y el conocimiento (PIN) en lugar de la típica autenticación débil basada sólo en el conocimiento (password). De hecho, todo esto se controla con PKCS#12.

Por lo tanto, estamos haciendo que además de ser una administración sencilla, incorpore mecanismos de seguridad que la hagan fiable y operativa en un entorno real.

Mostramos un esquema de esta arquitectura en la figura 4.8. Como vemos el esquema es muy similar al anterior. Sin embargo, mientras que en el esquema anterior el administrador tenía que ir modificando un fichero de texto en el MRRouter, aquí la administración se hace totalmente distribuida desde cualquier ordenador que disponga de un *browser* con soporte para Java.

Además, también es interesante destacar que este ordenador encargado de llevar la gestión puede estar en cualquier localización. No tiene incluso porqué estar dentro de la misma red sobre la que estamos haciendo la gestión de los emisores multicast.

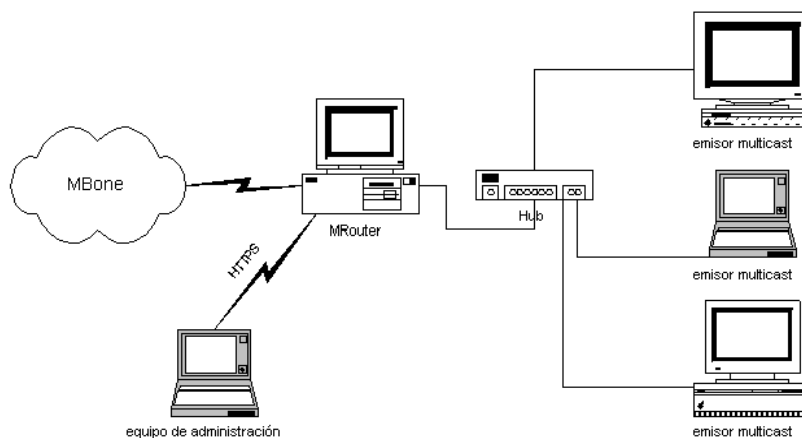


Figura 4.8: Esquema de esta solución con administración distribuida

## 4.5.2 Ventajas e inconvenientes

Evidentemente, este enfoque plantea una serie de ventajas sobre la alternativa anterior. Las principales ventajas son las que se describen a continuación:

1. Independencia total del demonio de enrutamiento multicast. Como hemos comentado antes, este enfoque nos permite insertar y eliminar emisores multicast sin necesidad de rearrancar la máquina y además permite que toda la parte del algoritmo de enrutamiento no se vea afectada por las modificaciones insertadas en el kernel para conseguir la gestión del tráfico multicast. De hecho, la comprobación de que unos determinados datagramas cumplan ciertas restricciones es obviamente independiente del interfaz por el que deba de salir, que en el fondo es lo único que va a decidir el demonio de enrutamiento.
2. Ofrece una gestión totalmente distribuida y basada en Web con la consiguiente comodidad y facilidad de uso que supone para el administrador. Del mismo modo, al ser la aplicación de gestión un applet nos asegura la suficiente independencia de la plataforma como para disponer de una herramienta de gestión fácil de instalar e integrar en cualquier entorno.

3. Es una solución que impone muy pocas restricciones al administrador de la red. Sólo se impone que la máquina que hace de MRouter se ejecute sobre un sistema operativo del que se disponga del código fuente del kernel. El resto de máquinas que participen en las sesiones multicast e incluso la del administrador pueden correr sobre cualquier plataforma.

Lógicamente, este enfoque también tiene algún inconveniente. El manejo puede resultar poco intuitivo. Además, el administrador tendría que estar atento a los grupos multicast que existen actualmente empleando alguna de las herramientas típicas de Mbone tales como *SDR* (Session DiRectory) o *MAnnouncer* (Multicast Announcer) y a partir de esa información añadir las autorizaciones al kernel del demonio de enrutamiento en base a la información obtenida sobre las sesiones activas.

## 4.6 Solución definitiva

Como hemos comentado antes, el principal problema de la solución anterior tenía que ver con el hecho de que el administrador necesitaba alguna herramienta adicional de gestión de sesiones para poder identificar los grupos multicast que corresponden a cada una de las sesiones. Es por esto, que nuestro siguiente objetivo va a consistir en el desarrollo de una herramienta global de gestión de usuarios multicast que permita facilitar la gestión de estos usuarios.

Para conseguir esto, la herramienta debe de permitir que el administrador desde una sólo ventana sea capaz de controlar las sesiones que van creándose o eliminándose, así como los emisores multicast autorizados a participar en cada una de esas sesiones junto con los parámetros asociados a esa participación.

Como ya hemos visto, la conjunción del protocolo SAP junto con el SDP para la gestión de las sesiones multicast, es una herramienta muy potente a la vez que complicada ya que es un protocolo en el que existen multitud de estados, tipos de paquetes, anuncios que pueden llegar comprimidos o cifrados, etc. Sin embargo, y aunque la tarea sea muy difícil, vamos a necesitar construir un nuevo elemento que se encargue de implementar el protocolo SAP/SDP al más puro estilo de cualquier herramienta de gestión de sesiones para Mbone. Este nuevo elemento que vamos a llamar *mauthserver* se va a colocar como un elemento intermedio entre el *musersd* y el *mauthclient* y va a tener varias funciones:

1. Escuchar anuncios de sesiones multicast. Para ello, tal y como hemos mencionado, el *mauthserver* va a implementar tanto el SAP (Session Announcement Protocol) como el SDP (Session Description Protocol). Así, este elemento se va a dedicar a analizar todos y cada uno de los anuncios que reciba y entonces enviará al *mauthclient* los eventos correspondientes de añadir o eliminar sesiones. El *mauthclient* al recibir estos eventos podrá presentar en la ventana de administración todas las sesiones que se encuentren activas en un instante dado.
2. Dar una cierta tolerancia a fallos al sistema. La posición estratégica que ocupa el *mauthserver* le permite controlar todo el flujo de información entre el kernel del sistema operativo del equipo que está haciendo de MRouter y el *mauthclient*. Es por esto que el *mauthserver* va a ser el elemento homogeneizador que va a permitir la recuperación del sistema ante cualquier caída que pudiera producirse en cualquiera de sus extremos. Para ello, el *mauthserver* mantendrá la información tanto de las sesiones como de los usuarios autorizados para emitir a esas sesiones en disco. En el instante en el que se produzca cualquier caída, a partir de los datos que se encuentran en el disco se puede conseguir la información suficiente como para para reestablecer la situación normal de funcionamiento.

El esquema general que representa esta solución puede apreciarse en la figura 4.9.

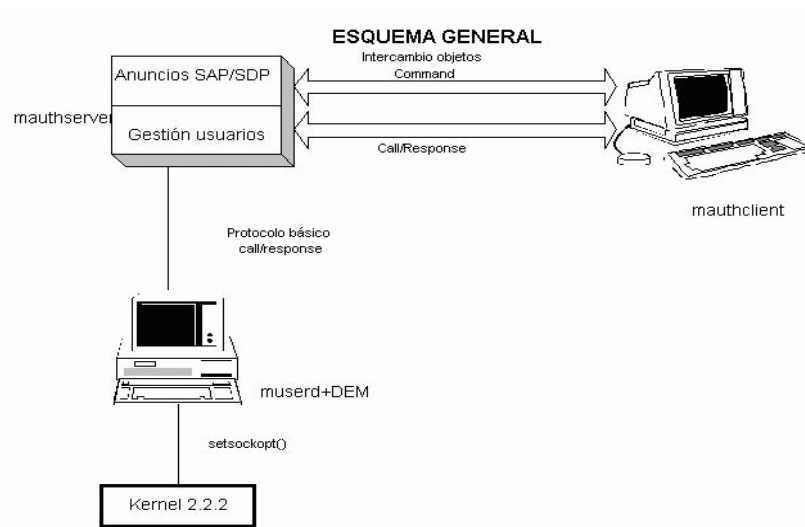


Figura 4.9: Esquema general de la solución final

Como vemos en la figura 4.9, se plantea un enfoque totalmente descentralizado en el que aparecen una serie de entidades que se comunican pero sin establecer en ningún caso restricciones sobre la localización de ninguna de esas entidades.

También se puede apreciar como aparecen una serie de esquemas de comunicación entre las diferentes entidades que componen la arquitectura global, que deben de definirse.

#### 4.6.1 Comunicación entre el *musersd* y el *mauthserver*

Para la comunicación entre estas dos entidades parece evidente que no debemos de crearnos ningún protocolo nuevo. De hecho, la comunicación es similar a la que se producía en la solución anterior entre el *mauthclient* y el *musersd*. Pues bien, aquí vamos a seguir empleando exactamente el mismo protocolo del tipo *call/response* que se empleaba en su momento.

Es decir, vamos a seguir teniendo un cliente y un servidor. El servidor va a seguir siendo el *musersd* pero sin embargo, el rol de cliente lo va a tomar en este caso el *mauthserver* que es el que va a hacer de intermediario entre el *mauthclient* y el *musersd*.

El *mauthserver* enviará solicitudes que a su vez recibe del *mauthclient*, y recibirá las respuestas a esos comandos por parte del *musersd*. En función de estas respuestas se actualizará el nuevo estado del sistema en disco y se devolverá la respuesta que corresponda al *mauthclient*.

El protocolo y los formatos de los mensajes van a ser por tanto los mismos que se mostraron anteriormente en la figura 4.7. Por supuesto, se siguen realizando las solicitudes por UDP y por lo tanto se siguen teniendo en cuenta las posibles pérdidas de paquetes, retransmisiones y demás mecanismos de control de flujo.

#### 4.6.2 Comunicación entre el *mauthserver* y el *mauthclient*

Para comunicar el *mauthserver* y el *mauthclient* se necesitan dos esquemas de comunicación: por un lado necesitamos que el *mauthserver* comunique al *mauthclient* los eventos relacionados con la creación o desaparición de las sesiones. Por otro lado, necesitamos que el *mauthserver* acepte las solicitudes que haga el administrador desde el *mauthclient* para registrar o eliminar emisores multicast y las transmita al *musersd* para que las haga efectivas.

Vemos por lo tanto la necesidad de dos esquemas de comunicación entre *mauthserver* y *mauthclient*. El primero de ellos encargado de la comunicación de eventos de creación

y expiración de las sesiones, se definirá a continuación. El segundo, que corresponde a la comunicación de los comandos para gestión de emisores multicast, se va a resolver empleando el protocolo Mcontrol anteriormente definido.

En este caso, dado que tanto uno como otro están implementados en Java, la comunicación se realiza de un modo más sencillo que en el caso anterior gracias sobre todo a que ya no tenemos que implementar ningún mecanismo de traducción de tipos de datos tal y como sucedía en el caso anterior. Lógicamente lo que sí que se ha tenido que programar ha sido la parte del servidor en Java que es el encargado de escuchar la llegada de comandos de gestión de usuarios, registrarlos, transmitir esos comandos al *musersd* y devolver la respuesta de nuevo al *mauthclient*.

Para el otro esquema de comunicación referente a la comunicación de los eventos de creación o desaparición de sesiones sí que hemos empleado un esquema de comunicación totalmente diferente a los anteriores. Más concretamente, hemos aprovechado el hecho de que tanto el *mauthserver* como el *mauthclient* se encuentren implementados en Java para utilizar las posibilidades de serialización de objetos que ofrece este lenguaje[Eck97]. De este modo, cada vez que el *mauthserver* descubra el anuncio de una nueva sesión, pasará a través de un socket que conecte ambos elementos un objeto del tipo *Comando* que será recibido en el otro extremo.

Para utilizar este mecanismo de envío y recepción de objetos a través de la red nos debemos de crear primero un flujo de datos de tipo objeto sobre un socket existente. Es decir, primero abrimos un socket y sobre ese objeto socket creamos un *ObjectOutputStream* o un *ObjectInputStream* según corresponda. Una vez nos hemos creado los streams correspondientes para enviar objetos se emplea el método *writeObject(objeto)* y para recibirlos se emplea el *readObject()*. Además, es importante tener en cuenta que una vez se recibe el objeto debemos de hacer el casting correspondiente para transformarlo a objeto de la clase que corresponda.

Básicamente, un objeto de la clase *Comando* va a contener un atributo *tipo* que va a identificar si se trata de añadir, modificar o eliminar alguna de las sesiones ya existentes. Además, también contendrá otro atributo de la clase *Session*. En este atributo es precisamente donde se almacena toda la información referente a la sesión multicast correspondiente (grupos multicast que intervienen, los tipos de codificación, el título, etc.).

De esta forma, el *mauthclient*, puede estar permanentemente informado de las sesiones que se vayan creando eliminando o modificando.

### 4.6.3 Creación de nuestro directorio de sesiones

Como hemos mencionado antes, un componente muy importante dentro de toda esta arquitectura es el encargado de implementar los protocolos SAP/SDP para poder llevar un control de las sesiones activas en cualquier instante.

Lo cierto es que la implementación de estos dos algoritmos no es ni mucho menos una tarea sencilla debido principalmente a que son protocolos pensados para funcionar en entornos muy diferentes y por lo tanto tienen un modo de funcionamiento muy peculiar para diferentes estados.

Como ejemplo de este modo de funcionamiento que debe adaptarse a diferentes condiciones de entornos, condiciones de tráfico y que siempre debe de escalar, podemos destacar el proceso que se lleva a cabo a la hora de olvidar las sesiones. El tiempo que debe de esperarse para eliminar una sesión de la que no se ha recibido refresco del anuncio se calcula como una función bastante complicada que depende de:

- El TTL de la sesión
- El número de sesiones actualmente anunciadas desde cualquier localización de ese ámbito.

Básicamente lo que se hace es limitar el ancho de banda que pueden consumir los anuncios según el ámbito de la sesión. Esta limitación aparece en la tabla 4.3.

Asociación entre TTL y ámbito <i>Type</i> de IGMP	
TTL	ancho de banda máximo
1-15	2 Kbps
16-63	1 Kbps
64-127	1 Kbps
128-255	200bps

Tabla 4.3: Límites de ancho de banda para los anuncios por TTL

Por lo tanto, todos los que han anunciado sesiones en la misma franja de TTLs, cuando van recibiendo nuevos anuncios de otros emisores a esta misma franja de TTLs tienen que reducir el tiempo de retransmisión de anuncios como corresponda. De hecho, la forma de calcularlo es como se muestra a continuación.

Dado un *limite* en bits por segundo y un tamaño del anuncio (*tam\_an*), el intervalo de retransmisión en segundos se calcula como:

$$interval = \text{MAX}(300, (8 * num\_anuncios * tam\_an) / limite)$$

Otro aspecto a tener en cuenta es la dificultad de tratamiento especial que entrañan los anuncios que vienen cifrados o comprimidos.

Una vez que se estudiaron todas estas dificultades, se comenzó a construir esta parte del proyecto. Para ello, comenzamos por estudiar e implementar el uso de sockets multicast con Java ya que tal y como vimos en el capítulo 2, los anuncios de sesiones se realizan mediante paquetes SAP/SDP dirigidos al grupo multicast 224.2.127.254 y al puerto UDP 9875. Por lo tanto, la parte de escuchar las sesiones consiste en unirnos a este grupo multicast y recibir los datagramas de los anuncios que correspondan. Además, debe de llevarse un control exhaustivo para evitar anuncios duplicados, detectar también modificaciones y borrados de sesiones, etc.

Por otra parte, también se tuvo que implementar un analizador del protocolo SDP de descripción de sesiones para poder extraer de cada paquete referido a una sesión toda la información que a nosotros nos interesa gestionar y mostrar en la interfaz de usuario.

En cuanto a la comunicación de los eventos de anuncio de una nueva sesión, modificación de una existente o expiración de sesiones, se emplea el mecanismo anteriormente mencionado de serialización de objetos.

En la figura 4.10 se puede observar el applet de administración definitivo. En la parte izquierda del applet se aprecia las sesiones que aparecen en la lista y como se pueden seleccionar. Al seleccionarse, aparecen los parámetros correspondientes en el resto de campos de la pantalla. En la parte final del applet se aprecian todos los campos destinados a la gestión de los usuarios multicast.

#### 4.6.4 Ventajas e inconvenientes

Como ahora veremos, este enfoque presenta ya bastantes ventajas. Las principales son las que mostramos a continuación:

1. Ofrece una solución realmente operativa y manejable. De hecho, estamos ofreciendo una herramienta integral que engloba toda la gestión de los emisores multicast y que permite que el administrador pueda establecer de un modo sencillo, dinámico y vía Web los parámetros que le corresponden a cada uno de sus emisores multicast. De este modo, podrá controlar totalmente el tráfico multicast que sale de su red

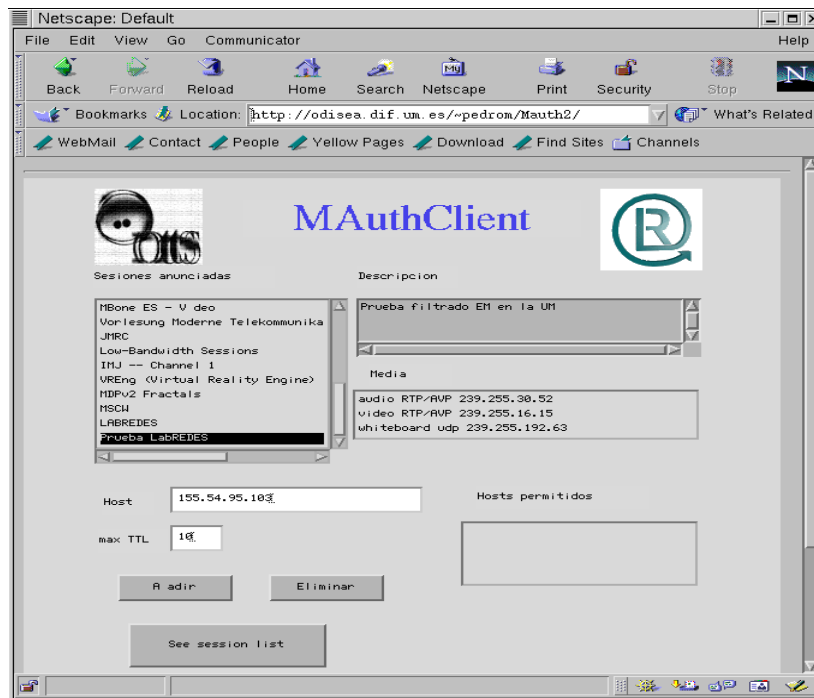


Figura 4.10: Applet de administración de la solución final

y evitará así todos los problemas que nos planteábamos al comienzo del desarrollo de este proyecto.

2. Propone un enfoque totalmente flexible y distribuido a la vez que fácil de integrar y utilizar en diferentes entornos informáticos.
3. El empleo del lenguaje de programación Java para la realización de todos los elementos que integran esta arquitectura de gestión de emisores multicast<sup>3</sup>, nos aporta una independencia total de la plataforma que nos ayuda aún más a conseguir una mayor flexibilidad. De hecho, el único elemento del sistema que no es multiplataforma es el MRrouter al que sólo se le impone que el sistema operativo sobre el que se ejecuta ofrezca los fuentes del kernel.
4. Facilita la tarea del administrador ya que el *mauthclient* incorpora una interfaz totalmente amigable, intuitiva y sencilla de utilizar. Esta interfaz se puede apreciar en la figura 4.10.
5. Todo el control se produce de forma transparente al usuario final. Es decir, en la parte del usuario final, todas las aplicaciones típicas de Mbone siguen pudiendo emplearse del mismo modo que siempre y sin cambiar la apariencia ni el código.
6. Por supuesto, sigue siendo independiente del demonio de enrutamiento multicast que utilice el MRrouter. De este modo, es una solución fácilmente integrable en multitud de entornos multicast y situaciones diferentes.

En cuanto a los inconvenientes, destacar la necesidad de la figura del administrador que debe de establecer los parámetros que van a regir el tráfico de salida de cada emisor multicast. Como ya se verá en las vías futuras, entra dentro de nuestros planteamientos tratar de evitar la presencia de un administrador y automatizar en la medida de lo posible el proceso.

---

<sup>3</sup>excepto el *musersd*

## 4.7 Otro enfoque alternativo

Hemos dedicado gran parte de este capítulo a explicar y justificar la solución que hemos adoptado pero sin embargo se ha hablado bien poco de las alternativas que se han descartado ni de otros posibles enfoques que existan.

Lo cierto es que las soluciones alternativas son muy escasas debido principalmente a que es un problema muy poco tratado en el MBone de hoy día.

### 4.7.1 Una primera alternativa descartada

Gracias a la labor mediadora de RedIRIS, tuve la oportunidad de participar en las reuniones mensuales del Grupo de Trabajo IRIS-MBONE e intercambiar impresiones con otra serie de personas interesadas en tratar temas relacionados con MBone e intercambiar experiencias.

Tras plantear el problema que pretendíamos resolver en este Grupo de Trabajo, un grupo de personas de la Universidad Carlos III de Madrid planteó su visión de como debería de resolverse el problema.

Su planteamiento estaba dirigido hacia modificar el código fuente del demonio de enrutamiento multicast (en concreto ellos querían hacerlo con mrouted) para que desechase todo aquel tráfico multicast cuyos paquetes RTP no incluyesen una determinada clave o secuencia de bytes. Los encargados de hacer que los paquetes multicast llevasen esos bytes de clave iban a ser las típicas aplicaciones de MBone.

Ya desde un principio, el hecho de tener que modificar el código fuente de las aplicaciones de MBone para que al generar el tráfico multicast se generen también estos bytes de clave pareció muy poco flexible. Sin embargo, como posible solución lo cierto es que ahí estaba.

Sin embargo, al estudiar en profundidad el funcionamiento del enrutamiento multicast en el sistema operativo Linux, ya se vió di cuenta de que esta alternativa nunca funcionaría.

El problema es básicamente el que se ha comentado al comienzo de este capítulo: no podemos tener ningún control sobre el tráfico multicast a nivel de demonio de enrutamiento multicast por lo que el esfuerzo de modificar las aplicaciones de MBone sería un esfuerzo en vano.

### 4.7.2 Una alternativa basada en una extensión de IGMP

Otra alternativa que nos pareció muy interesante en su momento y que actualmente pensamos que sería un punto de referencia indispensable, es la que se describe en el *draft-ishikawa-igmp-auth-01.txt* [Y98]. Este draft cuyos autores son Norihiro Ishikawa del NTT, Nagatsugu Yamanouchi de IBM y Osamu Takahashi también del NTT y cuyo título es *IGMP Extension for Authentication of IP Multicast Senders and Receivers*, explica una posible extensión al protocolo IGMPv2 para conseguir la autenticación de los emisores multicast y de este modo poder controlar quién envía y como envía.

En el enfoque que aquí se propone, cada emisor o receptor multicast debe de autenticarse frente a su MRrouter local. En el momento en el que esa autenticación falla, el MRrouter comienza a descartar todos los datagramas IP multicast que provengan de ese emisor o de cualquier otro no autenticado.

Para el proceso de autenticación, si bien se comenta que se están estudiando diferentes alternativas, en el draft proponen un enfoque similar al del protocolo CHAP basado en el mecanismo Challenge/Response. Es decir, el elemento a autenticar comienza enviando un mensaje indicando que va a comenzar a operar. A este mensaje de comienzo, el elemento autenticador responde con un mensaje de *Challenge*. Este mensaje básicamente sirve para que el elemento a autenticar sepa por un lado que tiene que autenticarse y por otro frente a quien tiene que autenticarse que no es otro que el origen del mensaje de *Challenge*. En el momento en el que el elemento a autenticar recibe el mensaje de *Challenge* debe de

responder con el mensaje de *Response* que es realmente el que contiene todos los datos necesarios para que se produzca la autenticación. Estos datos, dependerán del protocolo o mecanismo de autenticación concreto que se haya definido en cada caso.

Otro enfoque interesante en lo referente a la autenticación tiene que ver con el empleo de un servidor de RADIUS que ofrezca la posibilidad a los MRouters de comprobar la autenticidad de los emisores.

En su planteamiento proponen una arquitectura basada en *ingress routers* y *egress routers* similar a la de la figura 4.11.

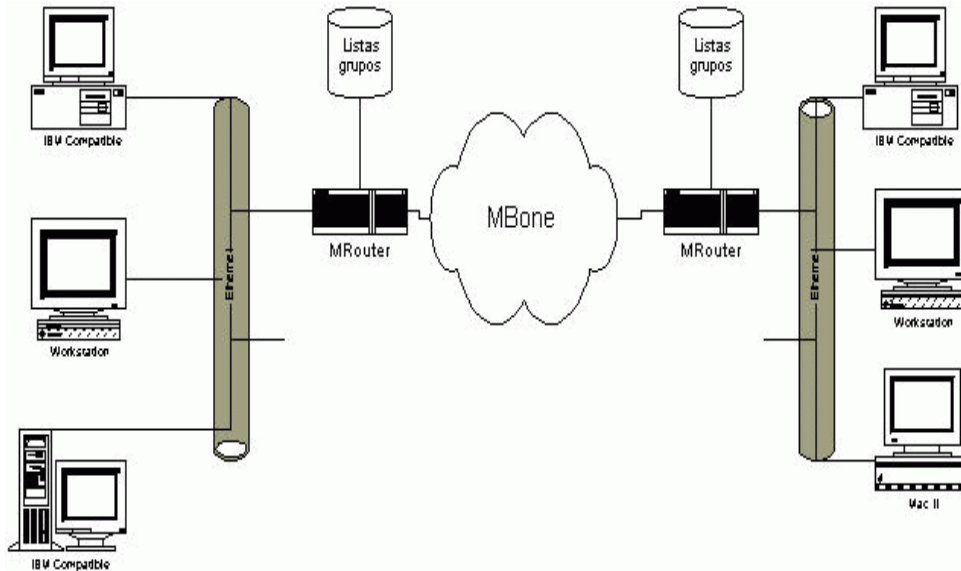


Figura 4.11: Arquitectura en base a *ingress* y *egress* routers

Como vemos en la figura 4.11 se trata de que tanto los emisores multicast como los receptores finales tengan que autenticarse frente a sus routers locales antes de poder participar en las sesiones multicast. A estos routers locales con capacidad de autenticación los llamamos *ingress* y *egress* routers. Los primeros se encargan de autenticar a los emisores mientras que los segundos se encargarán de autenticar a los receptores.

¿Como sucede entonces la autenticación de los emisores multicast?.

Para la autenticación de los emisores multicast, tal y como hemos dicho antes, se sigue un esquema *Challenge/Response*, es decir, el emisor multicast que desea comenzar a emitir datagramas a un determinado grupo multicast lo primero que hace es enviar un mensaje del tipo *Sender Start* al grupo multicast 224.0.0.2 (todos los routers multicast). Con este mensaje, el emisor le está indicando a los MRouters sólo su deseo de enviar a ese grupo multicast.

A este mensaje, responderá el MRouter que corresponda con un mensaje de *Challenge*. El mensaje de *Challenge* va dirigido al emisor multicast que envió el mensaje de *Sender Start* y básicamente sirve para que dicho emisor sepa de la existencia y localización del MRouter que va a encargarse de su autenticación.

Tras recibir el mensaje de *Challenge*, el emisor multicast deberá responder con un mensaje de *Response* que contenga la información que permita que el MRouter le autentique.

Cuando el mensaje de *Response* llega al MRouter, realizará la autenticación correspondiente del modo que se establezca. Destacar que no se impone ningún modo de autenticación que se deba de seguir estrictamente. De hecho los mismos autores proponen varios mecanismos entre los que parece destacar principalmente el de emplear un servidor de RADIUS. Si la autenticación ha sido correcta, entonces el MRouter enviará

de vuelta al emisor multicast un mensaje de *Success*. Sin embargo si la autenticación no es correcta le envía un mensaje de *Failure*.

Lógicamente, todo el tráfico multicast que envía el emisor multicast a este grupo antes de que se termine de realizar la autenticación será descartado por lo que no tiene ningún sentido que comience su emisión antes de estar totalmente autenticado.

Destacar también el hecho de que cada cierto tiempo el MRrouter puede requerir la reautenticación de cualquiera de sus emisores multicast. De hecho, el mensaje de *Success* lleva un campo llamado *Validity Period* que indica el tiempo que debe transcurrir entre reautenticaciones.

Para controlar posteriormente que todo el tráfico que salga del MRrouter corresponda sólo al proveniente de los emisores que se hayan autenticado con antelación, proponen que el MRrouter maneje una lista de direcciones IP correspondientes a los emisores multicast junto con un determinado grupo multicast y un puerto de destino. Cuando se autentica un nuevo emisor multicast se van añadiendo entradas a la lista y conforme vayan desapareciendo o se les vaya expirando el *Validity period* sin reautenticarse, se irán eliminando las entradas.

La autenticación de los receptores multicast va a ser un tanto diferente debido principalmente a la diferencia de funcionamiento del IGMPv2 original con respecto a los emisores y a los receptores.

Si recordamos el funcionamiento de IGMPv2 que explicamos en el capítulo 2, los MRrouters emitían periódicamente una serie de consultas tanto generales como específicas a un grupo multicast en concreto. Pues bien, el enfoque aquí propuesto consiste en añadir una serie de campos a los mensajes IGMP que permitan incorporar la autenticación a los mensajes de *Query*, *Group-Specific Query* y *Membership Report* para que los receptores multicast puedan autenticarse frente a su *egress* router usando los típicos mensajes de IGMPv2 pero modificados.

Del mismo modo que sucedía con los emisores, puede darse el caso de que se requiera una reautenticación pasado un cierto *Validity Period*. Para conseguir esto el MRrouter enviará un mensaje de *Group-Specific Query* con un campo nuevo llamado *parameter* conteniendo la cadena 'reason=reauthentication is required'. A este mensaje el host responderá con un mensaje *Membership Report* que contenga un parámetro de autenticación.

Los nuevos mensajes que incorpora esta propuesta a IGMPv2 son los que se pueden apreciar en la tabla 4.4.

Nuevos valores del campo <i>Type</i>	
Valor	Significado
0x21	Sender Start
0x22	Challenge
0x23	Response
0x24	Success
0x25	Failure

Tabla 4.4: Nuevos valores para el campo *Type*

## Ventajas e Inconvenientes

La principal ventaja que plantea este enfoque respecto al nuestro es el hecho de que todo el proceso de gestión de emisores multicast se realiza de un modo automático.

Al integrarse todo el proceso de autenticación en el protocolo IGMP, todo el proceso se hace transparente al usuario. Sin embargo, esto también plantea algún que otro inconveniente actualmente.

Con nuestro enfoque, sólo necesitábamos modificar el kernel del sistema operativo del equipo que hacía de MRouter. Aquí, al modificarse el protocolo IGMP estamos obligados a que todos los equipos que participen en sesiones multicast incorporen estas adiciones al protocolo IGMP por lo que los participantes sólo pueden ser equipos que tengan algún sistema operativo de dominio público. Es por lo tanto una solución menos flexible hoy por hoy. Sin embargo, en un futuro cercano, si llegase a imponerse este draft, incluso los sistemas operativos propietarios como Solaris, IRIX, SCO-Uinx, etc, incluirían ya estas modificaciones. En ese momento, estaríamos hablando de una solución flexible.

Otro pequeño inconveniente que se plantea es el hecho de que aunque se pretenda la autenticación de receptores multicast (sobre todo con vistas a que puedan existir en un futuro sesiones de pago), de momento y con la arquitectura actual del multicast, en redes de medio compartido este enfoque no tiene sentido. En el momento que se autentique cualquier receptor multicast de la subred ya cualquiera de los otros podrán recibir el mismo grupo multicast sin necesidad de autenticarse frente el MRouter.

El único modo de solventar estos problemas pasa por el empleo de criptografía y/o algoritmos de cifrado para la transmisión de datagramas multicast. Sin embargo, hoy por hoy no existe ningún algoritmo bueno de gestión de claves en entornos multicast ni entre MRouters. Es por esto que la única solución hoy por hoy pasaría por usar diferentes flujos de datos en la red de los receptores finales con lo que se perdería bastante escalabilidad que es precisamente la gran ventaja de Mbone frente a las propuestas de la ITU.

Por último, comentar una carencia en este draft y que tendrá que ser subsanada en versiones futuras. Con el esquema que plantean los autores, en ningún caso se controla el TTL con el que envían los emisores multicast. Esto puede suponer un problema tan pronto como dos personas lleguen a un acuerdo y después de autenticarse comiencen uno a emitir y otro a recibir con un TTL muy grande. Todo este flujo de tráfico iría circulando por todos los enlaces intermedios entre ambos extremos. Si el ancho de banda es grande y los usuarios se encuentran muy distantes podrían incluso provocar la saturación de líneas internacionales.

Para evitar esto, deberían haber adoptado un esquema de control del TTL similar al nuestro. En su caso bastaría con hacer que el MRouter además de comprobar la autenticación tal cual la proponen actualmente, vigile el TTL de esos datagramas para que no exceda el permitido para ese usuario.

## **4.8 Implantación de la nuestra solución en el marco de la Universidad de Murcia**

En este apartado vamos a explicar en detalle los pasos y acciones que habría que llevar a cabo para poder instalar una aula de libre acceso para Mbone en la Universidad de Murcia usando nuestro sistema de control de sesiones.

Lo primero que vamos a necesitar va a ser un PC con el sistema operativo Linux instalado y que emplee un Kernel 2.2.0 o superior.

En este sentido, y con la aparición tan reciente de la distribución RedHat 6.0, lo más sencillo es instalar esta distribución debido principalmente a la facilidad de instalación y a que ya incorpora el Kernel 2.2.5.

### **4.8.1 Hacer nuestras modificaciones a los fuentes del sistema operativo**

Una vez tenemos el sistema operativo ya instalado y funcionando, lo siguiente es realizar las modificaciones al kernel para que ofrezca las llamadas al sistema para registro de emisores multicast y permita realizar un control exhaustivo de los datagramas multicast que van a circular por nuestra red.

Para esto, lo más sencillo es aplicar un parche para el kernel que ofrecemos. En concreto lo que hemos hecho ha sido crearnos un fichero de diferencias (.diff) entre nuestro kernel una vez ya modificado y los fuentes del kernel sin modificar. Luego para modificar el kernel automáticamente lo que se hace es aplicar el parche mediante el comando *patch* de Linux.

Una vez que aplicamos el parche, tenemos que proceder a la recompilación del kernel del sistema operativo para que las modificaciones introducidas por el parche tomen efecto.

## 4.8.2 Recompilación del kernel del MRouter

El proceso de recompilación del kernel de Linux aparece recogido y explicado en multitud de documentos y HOWTOS, sin embargo, por comodidad para el lector no experimentado se comenta a continuación de un modo muy breve el modo de hacerlo. De todos modos no debe olvidarse que esta máquina no va a actuar como cualquier host normal de nuestra red sino que se le requiere que realice las funciones de enrutador multicast por lo que vamos a tener que habilitar bastantes funciones del kernel que normalmente no suelen tenerse en cuenta.

Lo primero como siempre es situarnos en el directorio de los fuentes de linux que por defecto suelen estar en */usr/src/linux*.

Lo siguiente, como en cualquier recompilación del kernel, va a ser configurar las opciones que queremos habilitar o deshabilitar del kernel y las que queremos que se soporten como módulos. Para ello se puede emplear cualquiera de los siguientes tres comandos: *make config*, *make menuconfig* o *make xconfig*. Si disponemos del entorno gráfico X, lo más recomendable es la última ya que nos permite la opción de consultar la ayuda sobre las opciones que desconozcamos.

Ahora en el menú de configuración de las opciones del kernel, es de vital importancia para que el host funcione realmente como enrutador multicast que se habiliten una serie de opciones del kernel en la parte de red que vamos a enumerar a continuación:

- `CONFIG_PACKET=y`
- `CONFIG_NETLINK=y`
- `CONFIG_UNIX=y`
- `CONFIG_INET=y`
- `CONFIG_IP_MULTICAST=y`
- `CONFIG_IP_ROUTER=y`
- `CONFIG_NET_IPIP=y`
- `CONFIG_IP_MROUTE=y`

Estas opciones son las indispensables y son las que nos van a permitir que nuestro equipo pueda dar el soporte al algoritmo de enrutamiento multicast, entender el direccionamiento de los grupos multicast y además permitir la creación de túneles para el envío y recepción de paquetes encapsulados.

El siguiente paso ahora es la compilación propiamente dicha del kernel. Para ello hacemos desde el mismo directorio un *'make clean'* & *'make dep'*. Esto hace que se borren todos los ficheros que no sirvan de la compilación anterior y que se cree el fichero de dependencias.

A continuación tecleamos *'make zilo'* y automáticamente comenzará a recompilarse todo el código fuente y al final se nos creará un fichero llamado */vmlinux*. Ahora ya sólo falta actualizar las entradas en el fichero */etc/lilo.conf*.

Por último hacemos *'make modules'* & *'make modules\_install'* y de este modo ya tenemos el sistema listo para arrancar con el nuevo kernel. Esto es precisamente lo que tenemos que hacer a continuación.

### 4.8.3 Instalación de mrouterd 3.9 beta 3 en Linux

El siguiente paso una vez que hemos recompilado el kernel es instalar el demonio de enrutamiento multicast.

Este paso también presenta una serie de consideraciones a destacar y por eso vamos a comentar aquí como se realiza el proceso.

Todas estas consideraciones vienen porque esta versión de mrouterd está preparada para instalarse en FreeBSD. El problema es que el soporte multicast que da el kernel de FreeBSD, si bien es muy parecido, es diferente en cuanto a la definición y el manejo de algunas estructuras de datos asociadas al enrutamiento multicast.

Lo primero que tenemos que hacer es conseguirnos los fuentes de este algoritmo de enrutamiento multicast. Estos fuentes pueden encontrarse por ejemplo en el servidor de ftp de RedIRIS.

Tras descomprimirlos, lo primero que tenemos que cambiar es el fichero Makefile que se emplea para obtener los ejecutables. Resulta que las opciones de compilación para Linux están mal puestas. Las opciones de CFLAGS que necesita Linux son: -DIOCTL\_OK\_ON\_RAW\_SOCKET, -DRAW\_INPUT\_IS\_RAW y -D\_BSD\_SOURCE.

Ahora para poder emplear la cabeceras que realmente espera *mrouterd* durante la compilación, la mejor solución consiste en hacer que emplee las de algún programa que incluya sus propios ficheros de cabecera para los paquetes IP. En ese sentido el más idóneo es *tcpdump*. Para ello, nos bajamos el código fuente del programa *tcpdump v.3.4* y lo descomprimimos.

Ahora tenemos que cambiar el fichero *Makefile* del *mrouterd* para que la variable MCAST\_INCLUDE tenga el valor *../tcpdump-3.4/linux-include*.

Ahora debemos de crear el fichero *../tcpdump-3.4/linux-include/netinet/ip\_mroute.h* que contenga simplemente la línea *#include<linux/mroute.h>*.

Además crearemos un fichero *../tcpdump-3.4/linux-include/netinet/in.h*. Este fichero se encuentra disponible vía Web y se muestra en el apéndice C.

Por último, en el fichero *defs.h* de la distribución del *mrouterd*, debemos eliminar la línea que hace referencia a *sys\_errlist*.

Una vez realizados todos estos cambios, recompilamos *mrouterd* y ya dispondremos de una versión estable y operativa del demonio de enrutamiento.

### 4.8.4 Configuración del demonio de enrutamiento

Para la configuración del demonio de enrutamiento, debemos de modificar el fichero *mrouterd.conf* con las opciones que correspondan. En concreto, el fichero de configuración que empleamos en nuestro entorno de pruebas es el que se muestra a continuación.

```
rexmit_prunes off
```

Esta línea se introdujo porque al principio no se realizaba el pruning correctamente.

```
phyint eth0 netmask 255.255.255.0 force_leaf
```

Esta línea simplemente declara la interfaz física a emplear.

```
tunnel eth0 130.206.208.2 metric 1 threshold 4 rate_limit 768
```

Y por último esta línea declara el túnel IPIP con el MRrouter principal de la Universidad de Murcia. Lógicamente, en el otro extremo del túnel debe configurarse el mismo túnel con los mismos parámetros.

### 4.8.5 Instalación del resto de componentes

La instalación del resto de componentes es muy sencilla. De hecho, el `musersd` se instala como un ejecutable ya en linux, y el resto de componentes son simplemente páginas HTML que contienen el applet de administración o programas en Java por lo que no tienen ningún problema para adaptarse a la plataforma. Consiste básicamente en ir colocando los elementos hasta conseguir la arquitectura que se describió en los apartados anteriores.

En cuanto a la instalación de los equipos clientes, dado que esta arquitectura es totalmente transparente al usuario final, bastaría simplemente con realizar la instalación de las aplicaciones de videoconferencia para MBone que vayan a emplearse. Por supuesto, las mínimas necesarias serían:

SDR  
VAT o RAT  
VIC  
WB o DLB  
mWEB

Para otras cuestiones sobre esta parte como las aplicaciones disponibles o las modificaciones a VIC para que soporte el estándar video4linux he ido manteniendo un WEB a lo largo del año en el que se pueden consultar muchas cuestiones de esta índole.

Además, sobre la instalación y el estudio de las aplicaciones de MBone existe un proyecto fin de carrera elaborado por Angel L. Mateo que aborda todas estas cuestiones que, si bien son interesantes, caen fuera del ámbito de este proyecto.

## Capítulo 5

# Conclusiones y vías futuras

El rápido desarrollo del IP multicast sobre Internet, a resultado en la consolidación de una arquitectura de red basada en túneles sobre los routers usuales denominada MBone. Pero, MBone sigue siendo una red experimental basada en IP multicast sobre Internet. Es precisamente este carácter experimental el que está frenando su implantación definitiva.

Para hacer de MBone un servicio comercial que pueda ser ofertado por los proveedores de servicios de Internet (ISP's), hay que introducir muchas mejoras que tienen que ver sobre todo con la seguridad, control de conexiones, calidad de servicio y asignación de direcciones multicast.

De todas estas mejoras, sin duda alguna la más importante y necesaria es la mejora en la seguridad incluyendo autenticación de emisores multicast, control sobre las sesiones para que usuarios malintencionados no puedan perjudicar las sesiones existentes, cifrado de las conexiones, gestión de claves, etc. De todas estas, la más básica e importante tiene que ver con la autenticación de los usuarios y es por esto que nosotros no hemos centrado en este aspecto.

Como hemos visto a lo largo del documento, cabe plantear diferentes alternativas para resolver el problema desde una solución a nivel de aplicación que impida a los usuarios arrancar las herramientas hasta que no se autentique hasta una solución basada en la modificación del soporte IGMP del kernel del sistema operativo para que incluya campos de autenticación.

El enfoque que proponemos podría situarse a mitad de camino entre los dos extremos anteriores aunque siempre tendiendo más hacia el segundo extremo. Más concretamente, hemos tratado de establecer el control sobre los usuarios en el MRouter más que en las aplicaciones. Para ello, hemos ofrecido e implementado tres soluciones al problema. Todas ellas tienen un denominador común: controlan el tráfico proveniente de los emisores multicast en el MRouter. En concreto, se ha conseguido:

- Modificación del soporte para enrutamiento multicast de los kernels 2.0.X y 2.2.X de Linux para permitir el filtrado de datagramas IP multicast.
- Adición de nuevas llamadas al sistema en el kernel de Linux para permitir la gestión dinámica de los emisores multicast.
- Modificación del código fuente de *mrouterd*. Como veremos a continuación, una de las soluciones que se han propuesto pasa por añadir un nuevo fichero de configuración a *mrouterd* para que pueda configurar los emisores multicast en el kernel. Por lo tanto una de las primeras tareas ha sido modificar el código fuente del demonio de enrutamiento multicast más usual: *mrouterd 3.9 beta 3*.
- Creación de un algoritmo de establecimiento de políticas de control a nivel del kernel del sistema operativo del equipo que haga de MRouter.

- Creación de un intérprete de SAP/SDP para poder monitorizar todos los anuncios de sesiones que se producen.
- Creación de applets de administración remota de todo el sistema que permitan simplificar las tareas del administrador.
- Estudio y puesta en marcha de controles de acceso al administrador en base a certificados digitales u ACLs.
- Definición de protocolos de comunicación entre las diferentes elementos que integran el sistema final.

De hecho, tenemos implementado y funcionando un esquema cuya migración al segundo extremo sería bastante sencilla. Esto nos permitiría disponer de una de las primeras implementaciones de las extensiones a IGMP si al final el draft de Ishikawa llegase a convertirse en RFC.

Lo realmente importante es que disponemos de un mecanismo que, del mismo modo que actualmente lo es para la Universidad de Murcia, puede ser de gran interés para el resto de universidades o proveedores de Internet a muy corto plazo. De hecho, disponemos de la primera implementación conocida a nivel Europeo de un esquema de este tipo. Las únicas implementaciones a nivel mundial ahora mismo son la realizada por Norihiro Ishikawa que funciona sobre FreeBSD y la nuestra que funciona sobre Linux.

Sin duda alguna, tan pronto como pasen unos años y los proveedores de Internet quieran ofrecer Mbone como un servicio más, van a necesitar un esquema de este tipo para poder limitar el ancho de banda consumido e incluso poder tarificar.

Además, el mecanismo que ofrecemos es lo suficientemente general como para que pueda ponerse en marcha en gran multitud de entornos de un modo rápido, sencillo y eficaz.

No obstante, aunque se trata de un sistema totalmente operativo, esto no significa que no pueda ser objeto de nuevas mejoras, algunas de las cuales se están intentando introducir desde ya mismo. En concreto algunas de las mejoras que se pueden introducir son:

- Puesto que es necesaria una comunicación entre distintos equipos de una red, es necesario establecer mecanismos de autenticación que nos permitan identificar los equipos que están participando en las comunicaciones y poder comprobar que no haya ningún host suplantando a ningún otro y que cada uno de los participantes es quién dice ser. En este sentido, se están considerando varias posibilidades como pueden ser la utilización de SSL(Secure Socket Layer) o bien la firma digital mediante PGP(Pretty Good Privacy).
- Actualmente, la autorización se hace mediante la figura del administrador, el cual, tiene que definir qué usuarios pueden enviar información y con qué TTL. Sin embargo, sería interesante integrar todo el sistema en un esquema de RADIUS para liberar de estas tareas al administrador.
- El sistema se centra fundamentalmente en prohibir a determinados usuarios el envío de información a determinados grupos y con determinados TTL para evitar que puedan con ello perjudicar a otros usuarios en cualquier otro lugar del mundo o mejor dicho de Internet. Sin embargo, no se ha tratado el tema de la recepción de información. En un siguiente paso, junto con la integración con IGMP, se puede realizar también autenticación de usuario a nivel de recepción, lo cual implica la introducción de algún esquema de cifrado de la información.
- Una vez hemos desarrollado ya todo el trabajo más difícil, no nos resultaría demasiado complicada la realización de un proxy de Mbone que permitiese incluso la traducción de direcciones. De este modo, los equipos que formen parte de la subred

podrían tener direcciones no enrutables y de ese modo no se emplean dirección IP válidas que tan escasas están actualmente. Sin duda alguna esta es otra nueva idea que surge, de la que a buen seguro no existe ninguna otra implementación conocida y que puede acabar en el desarrollo de una herramienta muy interesante.

# Bibliografía

- [Bal96] A. Ballardie. Scalable multicast key distribution, May 1996. RFC 1949.
- [Bal97] A. Ballardie. Center based trees (cbt) multicast routing architecture, September 1997. RFC 2201.
- [CHW98] J. Crowcroft, M. Hanley, and I. Wakeman. Internetworking multimedia, November 1998. UCL Press.
- [Dee89a] S. Deering. Host extensions for ip multicasting, 1989. RFC 1112.
- [Dee89b] Steve Deering. Ip multicast extensions for 4.3bsd unix and related systems. Technical report, Stanford University, June 1989.
- [Eck97] Bruce Eckel. *Thinking Java*. Prentice Hall, 1997. <http://www.phptr.com>.
- [EFH<sup>+</sup>97] D. Estrin, D. Farinacci, A. Helmy, D. Thaler, S. Deering, M. Handley, V. Jacobson, C. Liu, P. Sharma, and L. Wei. Protocol independent multicast sparse-mode: Protocol specification, June 1997. RFC 2117.
- [Fen97] W. Fenner. Internet group management protocol. version 2, November 1997. RFC 2236.
- [Gar98] J. A. García. Mbone: Arquitectura y aplicaciones. *Boletín de la red nacional de I+D, RedIRIS*, 1998.
- [Han96] Mark Handley. Sap: Session announcement protocol, November 1996. Internet-Draft.
- [HC94] M. Handley and S. Clayman. Specification of the mice conference management and multiplexing centre, version 2.0. [http://www-mice.cs.ucl.ac.uk/mice/cmmc\\_spec/cmmc\\_spec.html](http://www-mice.cs.ucl.ac.uk/mice/cmmc_spec/cmmc_spec.html), 1994.
- [HJ98] M. Handley and V. Jacobson. Sdp: Session description protocol, April 1998. RFC 2327.
- [HSSR99] M. Handley, H. Schulzrinne, E. Schoolet, and J. Rosenberg. Sip: Session initiation protocol, March 1999. RFC 2543.
- [IYT98] N. Ishikawa, N. Yamanouchi, and O. Takahashi. Igmpe extensions for authentication of ip multicast senders and receivers, August 1998. Internet-Draft.
- [KEA98] F. Kuo, W. Effelsberg, and J. J. García Luna Aceves. *Multimedia Communications: Protocols and Applications*. Prentice-Hall, 1998.
- [Kum96] Vinay Kumar. *MBone: Interactive Multimedia on the Internet*. New Riders, 1996.
- [MB94] R. Macedonia and D. P. Brutzman. Mbone provides video and audio across the internet. *IEEE Computer*, 27:30–36, April 1994.

- [Moy94] J. Moy. Multicast extensions to ospf, March 1994. RFC 1584.
- [Rus98] David A. Rusling. *The Linux Kernel*, March 1998. REVIEW. Version 0.8-2.
- [SCFJ96] H. Schulzrinne, S. Casner, R. Frederick, and V. Jacobson. Rtp: a transport protocol for real-time applications, 1996. RFC 1889.
- [SRL96] K. Savetz, N. Randall, and Y. Lepage. *MBone: Multicast Tomorrow's Internet*. IDG Books Worldwide, Inc., 1996. <http://www.savetz.com/mbone>.
- [Ste94] W. R. Stevens. *TCP/IP Illustrated. Volume 1: The Protocols*. Addison-Wesley, 1994.
- [Ste99] Richard W. Stevens. *UNIX Network Programming. Networking APIs: Sockets and XTI*. Prentice Hall, 1999.
- [Tho96] G. A. Thom. H.323: The multimedia communications standard for local area networks. *IEEE Communications Magazine*, December 1996.
- [WHA98] D. M. Wallner, E. J. Harder, and R. C. Agee. Key management for multicast: Issues and architectures, September 1998. `¡draft-wallner-key-arch-01.txt¿`.
- [WPD] D. Waitzman, C. Partridge, and S. Deering. Distance vector multicast routing protocol. RFC 1075.

# Apéndice A

## Estudio de los algoritmos de enrutamiento multicast

A continuación, mostramos un estudio de los pros y los contras de los algoritmos de enrutamiento multicast más conocidos y empleados actualmente.

### A.1 Algoritmos basados en inundación y poda

Este tipo de algoritmos son más conocidos como *algoritmos multicast de camino inverso* (reverse-path multicast algorithms). Los dos algoritmos más conocidos de este tipo son el DVMRP (Distance Vector Multicast Routing Protocol) que se recoge en el RFC 1075 y el DM-PIM (Dense-mode Protocol Independent Multicast) que actualmente se encuentra como *Work in Progress* en el IETF.

Cuando un host comienza por primera vez a emitir, el tráfico se transmite por inundación sobre la red. Un MRouter por lo tanto, puede recibir los datagramas por muchos caminos y por diferentes interfaces. Entonces el MRouter descarta cualquier paquete que le llegue por cualquier otra interfaz diferente a la que él usaría para enviar un paquete unicast al emisor. Después, envía una copia del paquete por cada uno de los otros interfaces diferentes al de vuelta al origen. De este modo cada enlace de toda la red se atraviesa al menos una vez en cada dirección y los datos son recibidos por todos los MRouters de la red.

Pero en el fondo, lo que acabamos de describir no es más que lo que podríamos llamar un broadcast de camino inverso (*reverse-path broadcast*). Empleando sólo este mecanismo habrían zonas de la red que recibirían tráfico multicast aunque no hubiesen host interesados en algunos de esos paquetes. Necesitamos pues introducir un mecanismo adicional que evite en la medida de lo posible este desperdicio de ancho de banda.

Este nuevo mecanismo del que estamos hablando no es otro que el *prunning*. El *prunning* consiste en hacer que cuando los MRouters sepan que no tienen ningún host interesado en algún grupo multicast determinado (mediante el uso de IGMP) envíen por el interfaz de vuelta al emisor mensajes de poda para evitar el flujo innecesario de tráfico. De esta forma, conforme pasa el tiempo, el gran árbol inicial que se crea se va reduciendo hasta un árbol mínimo que llega a todos los receptores. El árbol final de distribución se formaría uniendo los caminos más cortos desde cada receptor a cada emisor. Es por esto que este tipo de árbol de distribución suele llamarse de *shortest-path tree*.<sup>1</sup>

Una vez hemos examinado el comportamiento general de este tipo de algoritmos, pasaremos a centrarnos en explicar las peculiaridades de los dos algoritmos más famosos

---

<sup>1</sup>En realidad las implementaciones actuales hacen que los MRouters no tengan información suficiente como para construirse un verdadero árbol de camino más corto para envío por lo que al final todo queda en un árbol de camino más corto pero inverso.

que siguen este enfoque y que mencionamos antes: DVMRP y DM-PIM.

La principal diferencia entre estos dos algoritmos está en el hecho de que DVMRP calcula su propia tabla de rutas para determinar el mejor camino de vuelta hasta el origen de los datos. Sin embargo, DM-PIM usa la misma tabla de enrutamiento unicast de la que disponga el host (en realidad su sistema operativo). Es por esto que se le llama *Protocol Independent*.

Obviamente, enviar tráfico a todos sitios y esperar a que las distintas localizaciones comiencen a decir que no quieren recibir no es que sea muy buena idea a la hora de la escalabilidad. De hecho, presenta el inconveniente de que incluso los MRouters que no se encuentran el árbol de distribución deben de estar continuamente manteniendo el estado de pruning. Sin embargo, este enfoque, funciona perfectamente en entornos muy sobrecargados de receptores y es por esto que aunque estos algoritmos son una alternativa pobre para esquemas globales, pueden ser muy interesantes para trabajar de forma interna en el seno de algunas organizaciones.

## A.2 MOSPF

Este algoritmo (que aparece recogido en el RFC 1584), más que ser una categoría es simplemente la implementación específica de un protocolo. MOSPF es la extensión a multicast del protocolo OSPF (*Open Shortest Path First*) que es el típico algoritmo de enrutamiento unicast basado en el estado de los enlaces.

Los algoritmos de enrutamiento del tipo *link-state*, trabajan mediante el intercambio entre los routers de listas de vecinos conteniendo como de lejos queda cada uno de esos vecinos. Estos mensajes de enrutamiento se envían por inundación por toda la red y de este modo, cada router puede construirse un mapa de la red que después podrá emplear para construir las tablas de reenvío *forwarding* sin más que usar el algoritmo de Dijkstra.

¿Cómo se extiende esto a un enfoque multicast?. Pues simplemente haciendo que los MRouters dispongan además de una tabla de los grupos en los que hay hosts de la subred interesados. De este modo, dado el mapa de la red y las localizaciones de los receptores, el MRouter puede construir la tabla de *Multicast forwarding* para cada grupo.

Obviamente, este protocolo tiene una muy pobre escalabilidad. Con los protocolos basados en inundar y podar, la llegada de tráfico multicast hace intuir al MRouter donde pueden estar los receptores. Sin embargo, es este enfoque existen una serie de mensajes explícitos que se envían los MRouters para saber donde se encuentran. Sin embargo, y del mismo modo que antes, los MRouters deben conservar el estado de *prunning*. También es cierto que tanto los dos anteriores como este algoritmos construyen árboles de distribución muy eficientes.

## A.3 Center-based Trees

En lugar de inundar los datos a todos sitios o inundar las información de las suscripciones a los grupos a todos sitios, los algoritmos de este tipo, funcionan haciendo un asociación entre la dirección del grupo multicast y la dirección unicast de un MRouter particular. Todos los árboles de distribución se construirán entonces de forma explícita centrados en este MRouter particular.

Lógicamente aparecen tres problemas que hay que resolver:

1. La forma en la que se realizará el mapping de la dirección del grupo multicast y la dirección del MRouter particular.
2. La forma en la que elegir el MRouter particular para que los árboles de distribución sean eficientes.
3. La forma en la que se va a construir el árbol una vez que tenemos la dirección del centro.

Para resolver estos problemas han aparecido diferentes protocolos con diferentes soluciones a cada problema. Los tres protocolos más interesantes a estudiar en este ámbito son: CBT (*Core-Based Trees*), SM-PIM (*Sparse-mode PIM*) y BGMP (*Border Gateway Multicast Protocol*).

## A.4 Core-Based Trees

CBT fue el primer algoritmo del tipo *center-based tree* y, como suele pasar en estos casos, el más simple también. Su descripción puede encontrarse en el RFC 2201.

Cuando un receptor se une a un grupo multicast, su router local CBT anota esa dirección multicast y obtiene la dirección del *Core* router para ese grupo. A continuación envía un mensaje de *Join* para ese grupo hacia el *Core*. Entonces, por cada router que haya en el camino hacia el core, se va actualizando también dicha unión (*forwarding state*). Para asegurar la construcción del árbol, cada router por el que pasa la solicitud va enviando a su router anterior un asentimiento. De este modo, se construye un árbol multicast como se muestra en la figura A.1.

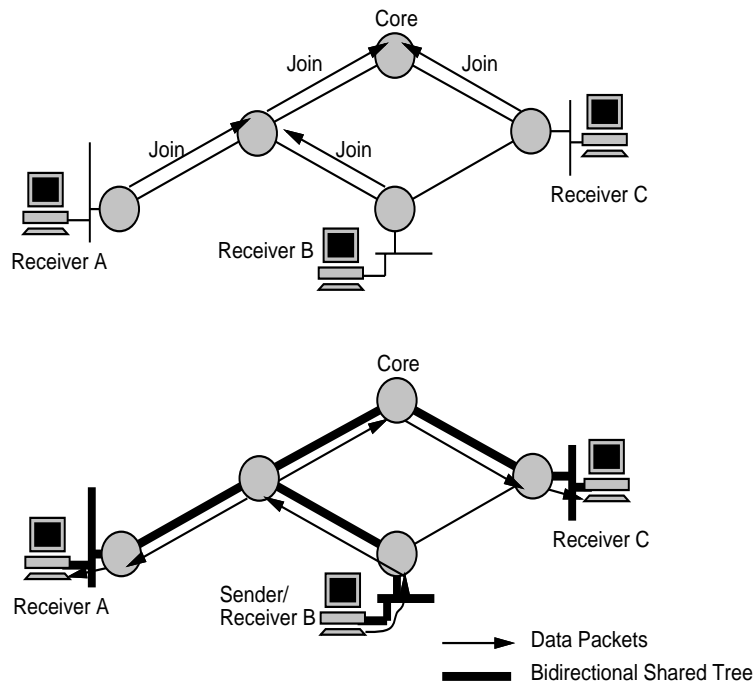


Figura A.1: Formación del árbol de enrutamiento multicast con CBT

Si un emisor es miembro del grupo al que envía, los paquetes llegan a su router local que los envía a alguno de sus vecinos que estén en el árbol de distribución.

Cada router que recibe un paquete lo reenvía por todas sus interfaces que están en el árbol de distribución excepto por la que le llegó. Se consigue por lo tanto un árbol bidireccional de distribución. Los paquetes pueden fluir tanto hacia el core como desde el core dependiendo de la localización del origen.

Cuando un host que no es miembro de un grupo envía paquetes a ese grupo multicast, puede suceder que su router local no pertenezca al árbol de distribución para ese grupo. Es por esto que su router local lo envía al router inmediatamente más cercano por la interfaz mas cercana al core. Procediendo de este modo llegará un momento en el que el paquete llegará a un router del árbol de distribución o llegará al core. En este momento es cuando se puede distribuir el paquete.

En cuanto al problema de hacer el mapping entre la dirección del grupo multicast y la dirección local del core, CBT nunca ha conseguido darle una solución satisfactoria.

Otro problema que presenta este algoritmo es que su eficacia depende en gran medida de la elección del core que se haga. Este proceso es muy complicado y esto hace que este algoritmo no se emplee como algoritmo global de enrutamiento multicast.

Sin embargo, presenta la ventaja de que los routers tienen que almacenar menos estados que con los enfoques anteriores. De hecho, sólo los routers que se encuentren en el árbol de distribución de un determinado grupo deben de mantener información del estado de *forwarding* para ese grupo y por lo tanto este algoritmo tiene una escalabilidad mejor que los protocolos basados en *flood-and-prune*. Esto se aprecia sobre todo en entornos con poca densidad de receptores.

## A.5 Sparse-Mode PIM

Este protocolo aparece definido en el RFC 2117 y está en estado de estándar experimental. Surge como un intento por tratar de resolver las limitaciones que presenta CBT pero manteniendo las buenas propiedades que ofrecen los árboles compartidos.

El equivalente al *core* del algoritmo CBT es lo que aquí se denomina RP (*Rendezvous Point*). De hecho, el propósito que tiene es el mismo.

Cuando un emisor comienza a enviar paquetes multicast a un determinado grupo, éstos son recibidos por su router local independientemente de si el emisor está suscrito al grupo multicast o no. Entonces el router local asocia la dirección del grupo multicast a la dirección del RP y encapsula los paquetes en datagramas IP dirigidos directamente al RP vía unicast.

En el momento en que un receptor se suscribe al grupo, su router local envía un mensaje de *Join* que viaja de router en router hasta llegar al RP y que además hace que cada router por el que pasa el mensaje de *Join* almacene el estado de *forwarding* correspondiente para ese grupo en concreto. Sin embargo, y al contrario que con CBT, aquí este estado es unidireccional. Es decir, sólo sirve para enviar paquetes que van del RP al receptor y no al revés. Los datos provenientes de los emisores se desencapsulan en el RP y se envían siguiendo el árbol de distribución hasta los receptores.

Uno de los avances que incorpora SM-PIM frente a CBT es el darse cuenta de que se puede separar el descubrimiento de los emisores de la construcción de árboles de distribución eficientes entre esos emisores y esos receptores.

Por lo que acabamos de ver, no parece que los árboles de distribución obtenidos sean demasiado buenos pero al menos hacen que comience a llegar el tráfico multicast a los receptores. La cuestión es, ¿Pueden mejorarse los árboles que se obtienen inicialmente?. Pues la respuesta es que sí y ahora veremos como se consigue.

Una vez que los datos están fluyendo, puede suceder que el router local de algún receptor intente cambiar el árbol de distribución compartido que se obtiene inicialmente a un árbol de distribución basado en el camino más corto. Para averiguar ese camino, el router envía un mensaje de *source-specific join* hacia el emisor tal y como se muestra en la figura A.2. Cuando los datos comienzan a llegarle a ese router por el nuevo árbol, ya puede enviarse un mensaje de *prune* por el camino inverso del árbol compartido (o inicial) para evitar que llegue tráfico duplicado.

Otra característica destacable es el hecho de que al igual que sucedía en CBT, sólo es necesario almacenar el estado de *forwarding* para un grupo en los del árbol de camino más corto. (*shortest-path tree*).

Gracias a que PIM-SM puede optimizar el árbol de distribución después de su formación, se consigue disminuir la dependencia de la localización del RP y se subsana pues uno de los problemas que presentaba CBT.

El otro problema que CBT no resolvía bien era el de hacer el mapping de la dirección del grupo multicast a la dirección local del Core. Pues en este caso, PIM-SM propone el empleo de una función especial de hash que permite seleccionar el RP de entre una lista

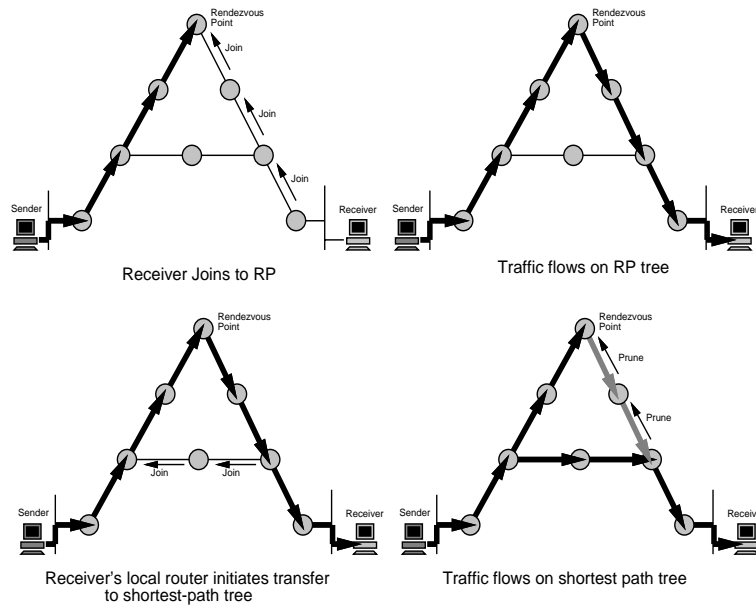


Figura A.2: Formación de los árboles de distribución en PIM-SM

de candidatos que se predistribuye. La función no tiene en cuenta la bondad del árbol obtenido pero esto no es un gran problema ya que PIM-SM permite la optimización del árbol sobre la marcha.

Como inconveniente a este protocolo podríamos plantear el hecho de que la dependencia de la función hash (todos deben conocerla para obtener el mismo RP para el mismo grupo multicast) y el hecho de tener que distribuir la lista de candidatos limita la escalabilidad de este algoritmo.

## A.6 Border Gateway Multicast Protocol

Este algoritmo es tan actual que se encuentra aún en desarrollo por parte del IETF. Sin embargo, esbozaremos las ideas principales que lo rigen al menos en la versión que hasta hoy se conoce.

Este protocolo surge como un intento de buscar un verdadero protocolo de enrutamiento multicast entre dominios con la suficiente escalabilidad como para poder emplearse en toda la Internet. ¿Por qué no pueden emplearse los anteriores?. Pues básicamente por su escalabilidad:

- DVMRP y PIM-DM no se pueden usar porque todos los routers deben de almacenar el estado para cada fuente incluso cuando no pertenecen al árbol de distribución.
- MOSPF tampoco porque si ya OSPF escala mal pues ni que decir tiene que la escalabilidad de MOSPF es muy pobre.
- CBT y PIM-SM tampoco pueden porque están limitados por la función de mapping de la dirección del grupo multicast a la del Core o RP.

Del mismo modo que su hermano gemelo (BGP), este algoritmo se basa en la idea de separar el enrutamiento intradominio del enrutamiento interdominio. De hecho, lo que hace BGMP es construir árboles bidireccionales compartidos similares a los de CBT pero entre dominios en vez de entre routers. A BGMP en ningún momento le importa el algoritmo de enrutamiento multicast que se esté empleando dentro de cada dominio concreto.

Sin embargo, BGMP también puede construir los llamados *source specific branches* que viene a ser algo así como la versión interdominios de los árboles de distribución usados en PIM-SM.

Como hemos visto antes, los problemas de escalabilidad que aparecían con CBT y PIM-SM eran debidos al proceso de mapping de direcciones de grupo a direcciones específicas del Core o del RP. Para evitar caer en este mismo problema, BGMP usa un esquema de asignación de direcciones jerárquico llamado *Multicast Address-Set Claim* (MASC). MASC se encarga de asignar rangos de direcciones multicast a dominios. Estos rangos de direcciones se distribuyen a los routers de frontera del dominio a lo largo de todo el mundo como rutas de grupo empleando el enrutamiento BGP que se usa para enrutamiento unicast entre dominios. Para hacer esto escalable, MASC asigna los rangos de direcciones de forma dinámica y haciendo que el número de rutas de grupos a ser almacenadas en cada router frontera sea relativamente pequeño.

En la figura A.3 se puede apreciar un ejemplo de construcción de un árbol de distribución BGMP. Por simplificar el ejemplo, dentro de los dominios se va a utilizar DM-PIM pero en realidad podemos emplear cualquier otro algoritmo de enrutamiento multicast.

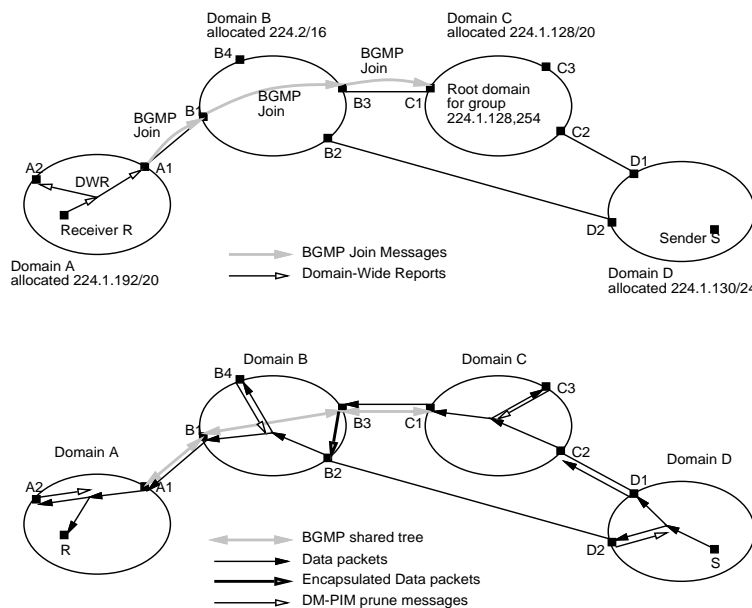


Figura A.3: Formación de los árboles de distribución compartidos en BGMP

La decisión de dónde colocar la raíz para cualquier árbol compartido es un problema difícil. BGMP coloca la raíz en el dominio que ha asignado la dirección multicast. Así, si el que inicia la sesión consigue la dirección de su servidor local de asignación de direcciones multicast, entonces el árbol tendrá como raíz el dominio de los que inicien la sesión. Para muchos usos como por ejemplo los broadcast del tipo TV este enfoque es óptimo. Sin embargo, para otros usos del multicast con muchos emisores podría llegar a ser menos óptimo pero aún una solución muy razonable.

Lo cierto es que hay que reconocer que teniendo el lastre de no conocer por adelantado los receptores es muy difícil hacer algo mejor que esto.

## Apéndice B

# Estructura del entorno de pruebas

Para poder realizar tanto las pruebas de la arquitectura como su desarrollo se tuvo que definir una subred aparte del resto de las subredes de la Universidad de Murcia, para que nuestras pruebas en la gestión de emisores multicast y el filtrado de su tráfico no pudiese perjudicar al resto de tráfico multicast de la Universidad.. En concreto se definió la red 155.54.95/24. Esta red, se conecta con el resto de la Universidad mediante un router Cisco 4500 que se encuentra directamente conectado al router principal de la universidad de Murcia.

El siguiente paso, fue deshabilitar el multicast a este router Cisco para que nuestra máquina de pruebas fuese la que se encargase de la gestión del multicast dentro de esta subred 95.

Por lo tanto, el siguiente paso es definir un túnel multicast entre el router principal de la Universidad de Murcia y nuestro host que va a hacer de enrutador multicast en esta subred.

Esta arquitectura se puede apreciar en la figura B.1.

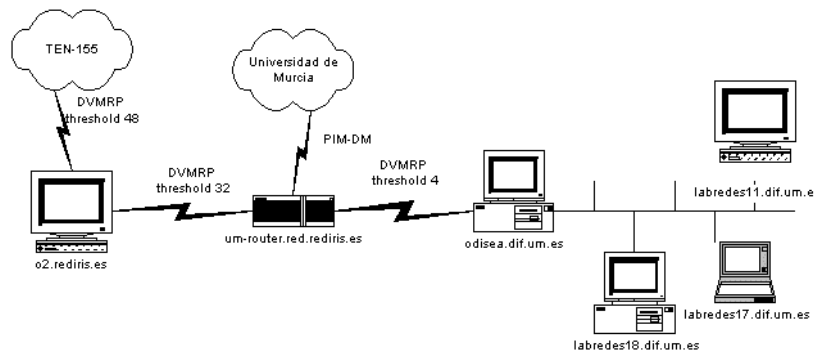


Figura B.1: Estructura del entorno de pruebas

Como vemos en la figura B.1, hemos conectado mediante un túnel DVMRP con threshold 4 a *odisea.dif.um.es* (nuestro MRrouter de la red 95) y a *um-router.red.rediris.es* (Interfaz que conecta al router principal de la Universidad de Murcia con el MRrouter central de RedIRIS en Madrid).

El establecimiento del túnel podemos consultarlo tanto en *odisea*:

```
[root@odisea /]# mrintfo odisea.dif.um.es
155.54.95.100 (odisea.dif.um.es) [DVMRPv3 compliant]:
  155.54.95.100 -> 0.0.0.0 (local) [1/1/querier/leaf]
```

```
155.54.95.100 -> 130.206.208.2 (um-router.red.rediris.es) [1/4/tunnel]
```

como en *um-router*:

```
[root@odisea /]# mrinto um-router.red.rediris.es
130.206.208.2 (um-router.red.rediris.es) [version 11.2]:
 130.206.208.2 -> 130.206.1.45 (o2.rediris.es) [1/32/tunnel/querier]
 155.54.1.200 -> 155.54.12.225 (router-saba.dif.um.es) [1/0/pim]
 155.54.32.200 -> 0.0.0.0 (local) [1/0/pim/querier/leaf]
 155.54.48.200 -> 0.0.0.0 (local) [1/0/pim/querier/leaf]
 155.54.18.200 -> 0.0.0.0 (local) [1/0/pim/querier/leaf]
 155.54.64.200 -> 0.0.0.0 (local) [1/0/pim/querier/leaf]
 155.54.96.200 -> 0.0.0.0 (local) [1/0/pim/querier/leaf]
 147.84.202.1 -> 0.0.0.0 (local) [1/0/pim/querier/leaf]
 130.206.208.2 -> 155.54.95.100 (odisea.dif.um.es) [1/4/tunnel/querier]
```

Además, el resto de la figura también puede conseguirse consultando los siguientes nodos de los túneles. Por ejemplo, vemos que el túnel de *um-router* va hacia *o2.rediris.es*. Consultando el estado de este host, podemos averiguar los siguientes túneles:

```
[root@odisea /]# mrinto o2.rediris.es
130.206.1.45 (o2.rediris.es) [DVMPv3 compliant]:
 130.206.1.45 -> 130.206.1.6 (A0-0-2.EB-Madrid3.red.rediris.es) [1/1]
 130.206.1.45 -> 212.1.193.225 (atm.ws1.fr.ten-155.net) [1/48/tunnel]
 130.206.1.45 -> 161.111.80.11 (sabina.cti.csic.es) [1/24/tunnel/down/leaf]
 130.206.1.45 -> 193.146.0.2 (indy.rediris.es) [1/16/tunnel]
 130.206.1.45 -> 130.206.5.232 (ccaxp2.unican.es) [1/32/tunnel]
 130.206.1.45 -> 130.206.33.96 (procyon.uib.es) [1/32/tunnel]
 130.206.1.45 -> 130.206.166.106 (gorosti.upna.es) [1/32/tunnel]
 130.206.1.45 -> 130.206.208.2 (um-router.red.rediris.es) [1/32/tunnel]
 130.206.1.45 -> 130.206.211.182 (A0-3.uv-router.red.rediris.es) [1/32/tunnel]
 130.206.1.45 -> 193.145.223.194 (pedraforca-atm.cesca.es) [1/32/tunnel]
 130.206.1.45 -> 150.214.2.82 (titan.cica.es) [1/32/tunnel]
 130.206.1.45 -> 155.210.12.18 (sc.unizar.es) [1/32/tunnel]
 130.206.1.45 -> 156.35.11.205 (rcpd02.net.uniovi.es) [1/32/tunnel/down/leaf]
 130.206.1.45 -> 130.206.224.42 (A3-0-1.EB-Valladolid1.red.rediris.es) [1/32/tunnel/down/leaf]
 130.206.1.45 -> 158.227.170.15 (stsv06.st.ehu.es) [1/32/tunnel]
 130.206.1.45 -> 161.72.69.9 (cisne.ll.iac.es) [1/32/tunnel]
 130.206.1.45 -> 193.144.33.48 (troll.cesga.es) [1/32/tunnel]
 130.206.1.45 -> 193.146.232.35 (tabata.unirioja.es) [1/32/tunnel]
 130.206.1.45 -> 193.146.185.35 (ws4-saba.dit.upm.es) [1/16/tunnel/down/leaf]
 130.206.1.45 -> 130.206.224.66 (A0-0-1.EB-Madrid3.red.rediris.es) [1/24/tunnel/down/leaf]
```

Por ejemplo, apreciamos el túnel de RedIRIS con Europa (TEN155) que está realizado con la máquina *atm.ws1.fr.ten-155.net* y tiene en *threshold* de 48.

Este último resultado, nos lleva al gráfico de la figura B.2 de los túneles de Mbone nacionales:



# Apéndice C

## Fichero in.h necesario para compilar mrouted 3.9 beta 3

```
/* Copyright (C) 1991 Free Software Foundation, Inc.  
This file is part of the GNU C Library.
```

```
The GNU C Library is free software; you can redistribute it and/or modify  
it under the terms of the GNU General Public License as published by  
the Free Software Foundation; either version 1, or (at your option)  
any later version.
```

```
The GNU C Library is distributed in the hope that it will be useful,  
but WITHOUT ANY WARRANTY; without even the implied warranty of  
MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the  
GNU General Public License for more details.
```

```
You should have received a copy of the GNU General Public License  
along with the GNU C Library; see the file COPYING. If not, write to  
the Free Software Foundation, 675 Mass Ave, Cambridge, MA 02139, USA. */
```

```
#ifndef _NETINET_IN_H
```

```
#define _NETINET_IN_H 1  
#include <features.h>
```

```
#include <endian.h>  
#include <sys/socket.h>  
#include <linux/types.h>  
#include <asm/types.h>
```

```
__BEGIN_DECLS
```

```
/* Standard well-known ports. */  
enum
```

```
{  
    IPPORT_ECHO = 7,           /* Echo service. */  
    IPPORT_DISCARD = 9,      /* Discard transmissions service. */  
    IPPORT_SYSTAT = 11,      /* System status service. */
```

```

IPPORT_DAYTIME = 13,      /* Time of day service. */
IPPORT_NETSTAT = 15,     /* Network status service. */
IPPORT_FTP = 21,         /* File Transfer Protocol. */
IPPORT_TELNET = 23,      /* Telnet protocol. */
IPPORT_SMTP = 25,        /* Simple Mail Transfer Protocol. */
IPPORT_TIMESERVER = 37,  /* Timeserver service. */
IPPORT_NAMESERVER = 42, /* Domain Name Service. */
IPPORT_WHOIS = 43,       /* Internet Whois service. */
IPPORT_MTP = 57,

IPPORT_TFTP = 69,        /* Trivial File Transfer Protocol. */
IPPORT_RJE = 77,
IPPORT_FINGER = 79,      /* Finger service. */
IPPORT_TTYLINK = 87,
IPPORT_SUPDUP = 95,      /* SUPDUP protocol. */

IPPORT_EXECSERVER = 512, /* execd service. */
IPPORT_LOGINSERVER = 513, /* rlogind service. */
IPPORT_CMDSERVER = 514,
IPPORT_EFSSERVER = 520,

/* UDP ports. */
IPPORT_BIFFUDP = 512,
IPPORT_WHOSERVER = 513,
IPPORT_ROUTESEVER = 520,

/* Ports less than this value are reserved for privileged processes. */
IPPORT_RESERVED = 1024,

/* Ports greater this value are reserved for (non-privileged) servers. */
IPPORT_USERRESERVED = 5000
};

/* Options for use with 'getsockopt' and 'setsockopt' at the IP level.
The first word in the comment at the right is the data type used;
"bool" means a boolean value stored in an 'int'. */
#define IP_TOS          1    /* int; IP type of service and precedence. */
#define IP_TTL          2    /* int; IP time to live. */
#define IP_HDRINCL      3    /* int; Header is included with data. */
#define IP_OPTIONS      4    /* ip_opts; IP per-packet options. */
#define IP_MULTICAST_IF 32   /* in_addr; set/get IP multicast i/f */
#define IP_MULTICAST_TTL 33  /* u_char; set/get IP multicast ttl */
#define IP_MULTICAST_LOOP 34 /* i_char; set/get IP multicast loopback */
#define IP_ADD_MEMBERSHIP 35 /* ip_mreq; add an IP group membership */
#define IP_DROP_MEMBERSHIP 36 /* ip_mreq; drop an IP group membership */

/* Link numbers. */
#define IMPLINK_IP      155
#define IMPLINK_LOWEXPER 156
#define IMPLINK_HIGHEXPER 158

```

```
/*
 * Many other definitions have been moved to <linux/in.h>,
 * because several parts of the kernel need them. -FvK
 */
#include <linux/in.h>

/*
 * Bind a socket to a privileged IP port
 */
extern int bindresvport __P ((int __sockfd,
                             struct sockaddr_in * __sin));

__END_DECLS

#endif /*netinet/in.h */
```

## Apéndice D

# Modificaciones a *mrouted 3.9 beta 3* para permitir la el control de emisores multicast

Básicamente, la modificación realizada a *mrouted* se centra en la creación de una s estructuras de datos para almacenar los emisores multicast que se leen del fichero de configuración, la creación de nuestras llamadas de inicialización y finalización del control sobre los emisores, y por último, el empleo de nuestras llamadas al sistema para añadir los usuarios a las estructuras de datos del kernel.

La estructura de datos empleada para almacenar los parámetros de los emisores multicast es la que también se emplea en el kernel: *msender\_ctl*.

Los prototipos de las funciones que hemos definido se muestran a continuación:

```
void init_mauth(void);
void finalize_mauth(void);
```

En cuanto a su código lo veremos a continuación:

```
struct msender_ctl ms;

FILE *      fc;
int         n;                /* # of char we have read */
char        s1[19],s2[19];
u_int32     src,grp;
u_char      ttl;
int         auxttl;

if ((fc = fopen("./mauth.conf","r"))==NULL) {
    perror("Error opening configurarion file\n");
    exit(1);
}

if (setsockopt(igmp_socket,IPPROTO_IP,MRT_SENDER_INIT,
              (char *)&ms,sizeof(ms))<0) {
    printf("Error al inicializar emisor...\n");
    fclose(fc);
    exit(1);
}
```

```

\enle ((n = fscanf(fc, "%s\t%s\t%d\n",s1,s2,&auxttl))==3) {
    src=inet_parse(s1,4);
    grp=inet_parse(s2,4);
    ttl=(u_char) auxttl;
    ms.src=src;
    ms.grp=grp;
    ms.ttl=ttl;

    if (setsockopt(igmp_socket,IPPROTO_IP,MRT_SENDER_ADD,
        (char *)&ms,sizeof(ms))<0) {
        printf("Error al añadir emisor...\n");
        fclose(fc);
        exit(1);
    }
    printf("%x, %x, %d\n",ms.src,ms.grp,ms.ttl);
}

if (!(feof(fc))) {
    perror("Incorrect format during configuration file read\n");
    fclose(fc);
}
fclose(fc);
}
end{verbatim}

```

Como vemos, lo primero que se hace es inicializar el control de emisores multicast del kernel mediante

El código para eliminar todos los emisores multicast es similar al anterior. La principal diferencia

```

\begin{verbatim}

```

```

void finalize_mauth()
{
    struct msender_ctl ms;

    FILE *      fc;
    int         n;          /* # of char we have read */
    char        s1[19],s2[19];
    u_int32     src,grp;
    u_char      ttl;
    int         auxttl;

    if ((fc = fopen("./mauth.conf","r"))==NULL) {
        perror("Error opening configuracion file\n");
        exit(1);
    }

    while ((n = fscanf(fc, "%s\t%s\t%d\n",s1,s2,&auxttl))==3) {
        src=inet_parse(s1,4);
        grp=inet_parse(s2,4);
    }
}

```

```

    ttl=(u_char) auxttl;
    ms.src=src;
    ms.grp=grp;
    ms.ttl=ttl;
    if (setsockopt(igmp_socket,IPPROTO_IP,MRT_SENDER_DEL,
        (char *)&ms,sizeof(ms))<0) {
        printf("Error al eliminar emisor...\n");
        fclose(fc);
        exit(1);
    }
    printf("%x, %x, %d\n",ms.src,ms.grp,ms.ttl);
}

if (!(feof(fc))) {
    perror("Incorrect format during configuration file read\n");
    fclose(fc);
}
fclose(fc);
}

```

Por último, sólo nos falta ver donde situar este código de entre las miles de líneas de *mROUTED*. La solución es obvia: buscar en el código fuente donde se inicia y se finaliza el enrutamiento multicast. Al iniciarse colocaremos la función *init\_mauth()*, y al finalizar colocaremos el *finalize\_mauth()*. A continuación mostramos el fragmento de código implicado que se encuentra en el fichero *kern.c* de la distribución de *mROUTED*.

```

void k_init_dvmrp()
{
#ifdef OLD_KERNEL
    if (setsockopt(igmp_socket, IPPROTO_IP, MRT_INIT,
        (char *)NULL, 0) < 0)
#else
    int v=1;

    if (setsockopt(igmp_socket, IPPROTO_IP, MRT_INIT,
        (char *)&v, sizeof(int)) < 0)
#endif
    log(LOG_ERR, errno, "can't enable Multicast routing in kernel");

    init_mauth();          /* Pedrom */
}

void k_stop_dvmrp()
{
    finalize_mauth();     /* Pedrom */

    if (setsockopt(igmp_socket, IPPROTO_IP, MRT_DONE,
        (char *)NULL, 0) < 0)
        log(LOG_WARNING, errno, "can't disable Multicast routing in kernel");
}

```

Estas llamadas son las que se llaman desde el programa principal para inicializar y

finalizar el enrutamiento multicast. De ahí, que al colocar aquí nuestras llamadas se obtenga el efecto deseado.

## Apéndice E

# Fichero de configuracion del núcleo de Linux para actuar como MRouter

```
#
# Automatically generated make config: don't edit
#

#
# Code maturity level options
#
CONFIG_EXPERIMENTAL=y

#
# Processor type and features
#
# CONFIG_M386 is not set
# CONFIG_M486 is not set
# CONFIG_M586 is not set
CONFIG_M586TSC=y
# CONFIG_M686 is not set
CONFIG_X86_WP_WORKS_OK=y
CONFIG_X86_INVLPG=y
CONFIG_X86_BSWAP=y
CONFIG_X86_POPAD_OK=y
CONFIG_X86_TSC=y
# CONFIG_MATH_EMULATION is not set
# CONFIG_MTRR is not set
# CONFIG_SMP is not set

#
# Loadable module support
#
CONFIG_MODULES=y
CONFIG_MODVERSIONS=y
CONFIG_KMOD=y
```

```

#
# General setup
#
CONFIG_NET=y
CONFIG_PCI=y
# CONFIG_PCI_GOBIO is not set
# CONFIG_PCI_GODIRECT is not set
CONFIG_PCI_GOANY=y
CONFIG_PCI_BIOS=y
CONFIG_PCI_DIRECT=y
CONFIG_PCI_QUIRKS=y
# CONFIG_PCI_OPTIMIZE is not set
CONFIG_PCI_OLD_PROC=y
# CONFIG_MCA is not set
# CONFIG_VISWS is not set
CONFIG_SYSVIPC=y
CONFIG_BSD_PROCESS_ACCT=y
CONFIG_SYSCTL=y
CONFIG_BINFMT_AOUT=y
CONFIG_BINFMT_ELF=y
CONFIG_BINFMT_MISC=y
# CONFIG_BINFMT_JAVA is not set
CONFIG_PARPORT=y
CONFIG_PARPORT_PC=y
# CONFIG_PARPORT_OTHER is not set
# CONFIG_APM is not set

#
# Plug and Play support
#
# CONFIG_PNP is not set

#
# Block devices
#
CONFIG_BLK_DEV_FD=y
CONFIG_BLK_DEV_IDE=y

#
# Please see Documentation/ide.txt for help/info on IDE drives
#
# CONFIG_BLK_DEV_HD_IDE is not set
CONFIG_BLK_DEV_IDEDISK=y
CONFIG_BLK_DEV_IDECD=y
# CONFIG_BLK_DEV_IDETAPE is not set
# CONFIG_BLK_DEV_IDEFLOPPY is not set
# CONFIG_BLK_DEV_IDESCSI is not set
CONFIG_BLK_DEV_CMD640=y
# CONFIG_BLK_DEV_CMD640_ENHANCED is not set
CONFIG_BLK_DEV_RZ1000=y
CONFIG_BLK_DEV_IDEPCI=y
CONFIG_BLK_DEV_IDEDMA=y
# CONFIG_BLK_DEV_OFFBOARD is not set
CONFIG_IDEDMA_AUTO=y

```

```
# CONFIG_BLK_DEV_OPTI621 is not set
# CONFIG_BLK_DEV_TRM290 is not set
# CONFIG_BLK_DEV_NS87415 is not set
# CONFIG_BLK_DEV_VIA82C586 is not set
# CONFIG_BLK_DEV_CMD646 is not set
# CONFIG_IDE_CHIPSETS is not set

#
# Additional Block Devices
#
# CONFIG_BLK_DEV_LOOP is not set
# CONFIG_BLK_DEV_NBD is not set
# CONFIG_BLK_DEV_MD is not set
# CONFIG_BLK_DEV_RAM is not set
# CONFIG_BLK_DEV_XD is not set
CONFIG_PARIDE_PARPORT=y
# CONFIG_PARIDE is not set
# CONFIG_BLK_DEV_HD is not set

#
# Networking options
#
CONFIG_PACKET=y
CONFIG_NETLINK=y
CONFIG_RTNETLINK=y
CONFIG_NETLINK_DEV=y
# CONFIG_FIREWALL is not set
CONFIG_NET_ALIAS=y
# CONFIG_FILTER is not set
CONFIG_UNIX=y
CONFIG_INET=y
CONFIG_IP_MULTICAST=y
# CONFIG_IP_ADVANCED_ROUTER is not set
# CONFIG_IP_PNP is not set
CONFIG_IP_ROUTER=y
CONFIG_NET_IPIP=y
# CONFIG_NET_IPGRE is not set
CONFIG_IP_MROUTE=y
# CONFIG_IP_PIMSM_V1 is not set
# CONFIG_IP_PIMSM_V2 is not set
CONFIG_IP_ALIAS=y
# CONFIG_ARPD is not set
# CONFIG_SYN_COOKIES is not set

#
# (it is safe to leave these untouched)
#
# CONFIG_INET_RARP is not set
# CONFIG_IP_NOSR is not set
CONFIG_SKB_LARGE=y
# CONFIG_IPV6 is not set

#
#
```

```

#
# CONFIG_IPX is not set
# CONFIG_ATALK is not set
# CONFIG_X25 is not set
# CONFIG_LAPB is not set
# CONFIG_BRIDGE is not set
# CONFIG_LLC is not set
# CONFIG_ECONET is not set
# CONFIG_WAN_ROUTER is not set
# CONFIG_NET_FASTROUTE is not set
# CONFIG_NET_HW_FLOWCONTROL is not set
# CONFIG_CPU_IS_SLOW is not set

#
# QoS and/or fair queueing
#
# CONFIG_NET_SCHED is not set

#
# SCSI support
#
# CONFIG_SCSI is not set
# CONFIG_SCSI_G_NCR5380_PORT is not set
# CONFIG_SCSI_G_NCR5380_MEM is not set

#
# Network device support
#
CONFIG_NETDEVICES=y
# CONFIG_ARCNET is not set
CONFIG_DUMMY=m
# CONFIG_EQUALIZER is not set
# CONFIG_ETHERTAP is not set
CONFIG_NET_ETHERNET=y
# CONFIG_NET_VENDOR_3COM is not set
# CONFIG_LANCE is not set
# CONFIG_NET_VENDOR_SMC is not set
# CONFIG_NET_VENDOR_RACAL is not set
# CONFIG_RTL8139 is not set
# CONFIG_YELLOWFIN is not set
# CONFIG_ACENIC is not set
# CONFIG_NET_ISA is not set
CONFIG_NET_EISA=y
# CONFIG_PCNET32 is not set
# CONFIG_AC3200 is not set
# CONFIG_APRICOT is not set
# CONFIG_CS89x0 is not set
# CONFIG_DE4X5 is not set
# CONFIG_DEC_ELCP is not set
# CONFIG_DGRS is not set
CONFIG_EEXPRESS_PRO100=m
# CONFIG_LNE390 is not set
# CONFIG_NE3210 is not set
# CONFIG_NE2K_PCI is not set

```

```
# CONFIG_TLAN is not set
# CONFIG_VIA_RHINE is not set
# CONFIG_ES3210 is not set
# CONFIG_EPIC100 is not set
# CONFIG_ZNET is not set
# CONFIG_NET_POCKET is not set
# CONFIG_FDDI is not set
# CONFIG_HIPPI is not set
# CONFIG_DLCI is not set
# CONFIG_PLIP is not set
# CONFIG_PPP is not set
# CONFIG_SLIP is not set
# CONFIG_NET_RADIO is not set
# CONFIG_TR is not set
# CONFIG_SHAPER is not set
# CONFIG_HOSTESS_SV11 is not set
# CONFIG_COSA is not set
# CONFIG_RCPCI is not set

#
# Amateur Radio support
#
# CONFIG_HAMRADIO is not set

#
# IrDA subsystem support
#
# CONFIG_IRDA is not set

#
# ISDN subsystem
#
# CONFIG_ISDN is not set
#
# Old CD-ROM drivers (not SCSI, not IDE)
#
# CONFIG_CD_NO_IDESCSI is not set

#
# Character devices
#
CONFIG_VT=y
CONFIG_VT_CONSOLE=y
CONFIG_SERIAL=y
# CONFIG_SERIAL_CONSOLE is not set
# CONFIG_SERIAL_EXTENDED is not set
# CONFIG_SERIAL_NONSTANDARD is not set
CONFIG_UNIX98_PTYS=y
CONFIG_UNIX98_PTY_COUNT=256
CONFIG_PRINTER=y
# CONFIG_PRINTER_READBACK is not set
CONFIG_MOUSE=y

#
```

```
# Mice
#
# CONFIG_ATIXL_BUSMOUSE is not set
# CONFIG_BUSMOUSE is not set
# CONFIG_MS_BUSMOUSE is not set
CONFIG_PSMOUSE=y
# CONFIG_82C710_MOUSE is not set
# CONFIG_PC110_PAD is not set
# CONFIG_QIC02_TAPE is not set
# CONFIG_WATCHDOG is not set
# CONFIG_NVRAM is not set
# CONFIG_RTC is not set

#
# Video For Linux
#
CONFIG_VIDEO_DEV=y
# CONFIG_RADIO_RTRACK is not set
# CONFIG_RADIO_RTRACK2 is not set
# CONFIG_RADIO_AZTECH is not set
# CONFIG_RADIO_MIROPCM20 is not set
# CONFIG_RADIO_GEMTEK is not set
CONFIG_VIDEO_BT848=m
# CONFIG_VIDEO_BWQCAM is not set
# CONFIG_VIDEO_CQCAM is not set
# CONFIG_VIDEO_PMS is not set
# CONFIG_VIDEO_SAA5249 is not set
# CONFIG_RADIO_SF16FMI is not set
# CONFIG_RADIO_ZOLTRIX is not set

#
# Joystick support
#
# CONFIG_JOYSTICK is not set

#
# Ftape, the floppy tape device driver
#
# CONFIG_FTAPE is not set
# CONFIG_FT_NORMAL_DEBUG is not set
# CONFIG_FT_FULL_DEBUG is not set
# CONFIG_FT_NO_TRACE is not set
# CONFIG_FT_NO_TRACE_AT_ALL is not set
# CONFIG_FT_STD_FDC is not set
# CONFIG_FT_MACH2 is not set
# CONFIG_FT_PROBE_FC10 is not set
# CONFIG_FT_ALT_FDC is not set

#
# Filesystems
#
# CONFIG_QUOTA is not set
CONFIG_AUTOFS_FS=y
# CONFIG_ADFS_FS is not set
```

```

# CONFIG_AFFS_FS is not set
# CONFIG_HFS_FS is not set
CONFIG_FAT_FS=y
CONFIG_MSDOS_FS=y
# CONFIG_UMSDOS_FS is not set
CONFIG_VFAT_FS=y
CONFIG_ISO9660_FS=y
CONFIG_JOLIET=y
# CONFIG_MINIX_FS is not set
# CONFIG_NTFS_FS is not set
# CONFIG_HPFS_FS is not set
CONFIG_PROC_FS=y
CONFIG_DEVPTS_FS=y
# CONFIG_QNX4FS_FS is not set
# CONFIG_ROMFS_FS is not set
CONFIG_EXT2_FS=y
# CONFIG_SYSV_FS is not set
# CONFIG_UFS_FS is not set

#
# Network File Systems
#
# CONFIG_CODA_FS is not set
CONFIG_NFS_FS=y
# CONFIG_NFSD is not set
CONFIG_SUNRPC=y
CONFIG_LOCKD=y
# CONFIG_SMB_FS is not set
# CONFIG_NCP_FS is not set

#
# Partition Types
#
# CONFIG_BSD_DISKLABEL is not set
# CONFIG_MAC_PARTITION is not set
# CONFIG_SMD_DISKLABEL is not set
# CONFIG_SOLARIS_X86_PARTITION is not set
# CONFIG_UNIXWARE_DISKLABEL is not set
CONFIG_NLS=y

#
# Native Language Support
#
CONFIG_NLS_CODEPAGE_437=y
# CONFIG_NLS_CODEPAGE_737 is not set
# CONFIG_NLS_CODEPAGE_775 is not set
CONFIG_NLS_CODEPAGE_850=y
# CONFIG_NLS_CODEPAGE_852 is not set
# CONFIG_NLS_CODEPAGE_855 is not set
# CONFIG_NLS_CODEPAGE_857 is not set
# CONFIG_NLS_CODEPAGE_860 is not set
# CONFIG_NLS_CODEPAGE_861 is not set
# CONFIG_NLS_CODEPAGE_862 is not set
# CONFIG_NLS_CODEPAGE_863 is not set

```

```
# CONFIG_NLS_CODEPAGE_864 is not set
# CONFIG_NLS_CODEPAGE_865 is not set
# CONFIG_NLS_CODEPAGE_866 is not set
# CONFIG_NLS_CODEPAGE_869 is not set
# CONFIG_NLS_CODEPAGE_874 is not set
CONFIG_NLS_ISO8859_1=y
# CONFIG_NLS_ISO8859_2 is not set
# CONFIG_NLS_ISO8859_3 is not set
# CONFIG_NLS_ISO8859_4 is not set
# CONFIG_NLS_ISO8859_5 is not set
# CONFIG_NLS_ISO8859_6 is not set
# CONFIG_NLS_ISO8859_7 is not set
# CONFIG_NLS_ISO8859_8 is not set
# CONFIG_NLS_ISO8859_9 is not set
# CONFIG_NLS_ISO8859_15 is not set
# CONFIG_NLS_KOI8_R is not set

#
# Console drivers
#
CONFIG_VGA_CONSOLE=y
# CONFIG_VIDEO_SELECT is not set
# CONFIG_MDA_CONSOLE is not set
# CONFIG_FB is not set

#
# Sound
#
CONFIG_SOUND=y
# CONFIG_SOUND_ES1370 is not set
# CONFIG_SOUND_ES1371 is not set
# CONFIG_SOUND_SONICVIBES is not set
# CONFIG_SOUND_MSNDCLAS is not set
# CONFIG_SOUND_MSNDPIN is not set
CONFIG_SOUND_OSS=m
# CONFIG_SOUND_PAS is not set
CONFIG_SOUND_SB=m
# CONFIG_SOUND_ADLIB is not set
# CONFIG_SOUND_GUS is not set
# CONFIG_SOUND_MPU401 is not set
# CONFIG_SOUND_PSS is not set
# CONFIG_SOUND_MSS is not set
# CONFIG_SOUND_SSCAPE is not set
# CONFIG_SOUND_TRIX is not set
# CONFIG_SOUND_MAD16 is not set
# CONFIG_SOUND_WAVEFRONT is not set
# CONFIG_SOUND_CS4232 is not set
# CONFIG_SOUND_OPL3SA2 is not set
# CONFIG_SOUND_MAUI is not set
# CONFIG_SOUND_SGALAXY is not set
# CONFIG_SOUND_AD1816 is not set
# CONFIG_SOUND_OPL3SA1 is not set
# CONFIG_SOUND_SOFTOSS is not set
# CONFIG_SOUND_YM3812 is not set
```

```
# CONFIG_SOUND_VMIDI is not set
# CONFIG_SOUND_UART6850 is not set

#
# Additional low level sound drivers
#
CONFIG_LOWLEVEL_SOUND=y
CONFIG_ACI_MIXER=m
CONFIG_AWE32_SYNTH=m

#
# Kernel hacking
#
# CONFIG_MAGIC_SYSRQ is not set
```