Universidad Carlos III de Madrid
Escuela Politécnica Superior

Ingeniería Informática

Implantación de IPv6: red, mecanismos de transición IPv4/IPv6, servicios de aplicación y migración de aplicaciones

Autor: Juan Fco. Rodríguez Hervella
Tutor: Alberto García Martínez

Abril de 2003

PROYECTO FIN DE CARRERA

Departamento de Ingeniería Telemática

Universidad Carlos III de Madrid

Título: Implantación de IPv6: red, mecanismos de transición IPv4/IPv6, servicios de aplicación y migración de aplicaciones

Autor: Juan Francisco Rodríguez Hervella

Tutor: Alberto García Martinez

La defensa del presente proyecto fin de carrera se realizó el día 10 de Abril de 2003 estando presente como tribunal las siguientes personas:

Presidente: Marisol García Valls.

Vocal: Marcelo Bagnulo Braun.

Secretario: Javier Fernández Muñoz.

Habiendo obtenido la siguiente calificación:

Presidente **Vocal** **Secretario**

A mis padres,
Lola y Francisco.

Y a mis hermanos,
Avelino y Conchita.

Agradecimientos.

En primer lugar quiero dar las gracias a mis padres, para quienes va dedicado principalmente este trabajo. En segundo lugar, quiero dar las gracias a todos mis compañeros de laboratorio y deseo dejar constancia en este apartado de los buenos momentos que he pasado junto a ellos. Dedico este trabajo a todos ellos. Tampoco quiero olvidarme de los técnicos, me refiero al grupúsculo de personas que se encuentran detrás de tres o cuatro monitores, a los que la gente sólo busca cuando hay problemas. Ellos realizan un trabajo poco agradecido (nos acordamos de ellos cuando las cosas no funcionan), por lo que quiero desde este pequeño apartado agradecerles su paciencia y su buen hacer. Sin ellos, este trabajo no habría sido posible.

En último lugar, y no por ello el menos importante, quiero destacar la labor realizada por mi tutor Alberto García Martínez. Sin sus correcciones y sin sus sugerencias, este trabajo no habría visto la luz de la biblioteca.

También es mi deber agradecer a la Universidad Carlos III de Madrid y en particular al departamento de Ingeniería Telemática el haber podido aprender e investigar disponiendo de todos los medios habidos y por haber. Puedo afirmar, con conocimiento de causa, que los licenciados e ingenieros de la Universidad Carlos III de Madrid , son, a día de hoy, los mejor preparados de España.

Y como de bien nacidos es ser agradecidos, rápidamente quiero nombrar aquí a todas las personas que han participado de una u otra forma en la elaboración de este trabajo. Gracias a Ricardo Romeral, por sus conocimientos sobre LaTeX, la herramienta con la que he desarrollado este documento. Mi agradecimiento también para Marcelo Bagnulo, por sus conocimientos, que en general, abarcan cualquier ámbito científico o social, y que para mí resultan inabarcables. Debo dar las gracias a Raquel Panadero Arroyo, Manolo Urueña Pascual, Carlos Jesús Bernardos, Gregorio Corral Torres, Rafael Sánchez Varas y muchos más. Sin ellos, todo hubiera sido mucho más aburrido.

A estas alturas de los agradecimientos, alguien podría sentirse decepcionado por no encontrar su nombre escrito. Por consiguiente, es mi deber realizar un agradecimiento "general" (algo semejante a la ofrenda al soldado desconocido), puesto que no resulta factible citar aquí a todas las personas que han influido de alguna u otra manera en este trabajo.

Introducción.

El presente proyecto fin de carrera trata los aspectos más importantes de la gestión y migración de las redes IP hacia el nuevo protocolo de red IPv6 y pone gran énfasis en experiencias de implantación de los mismos. A lo largo de este proyecto se profundiza sobre los distintos mecanismos de transición IPv4/IPv6 existentes, concretamente se describen y se prueban (en entornos *Linux* y *FreeBSD*) los siguientes:

- Túneles 6to4, automáticos, configurados.
- Mecanismo DSTM, TRT, ISATAP, NAT-PT.

También se presentan y se implantan en un entorno operativo servicios de red ofrecidos a través de la nueva plataforma IPv6, entre los que podemos citar:

- DNS.
- Servidor web.
- Red de servidores de IRC.
- Servicio de NFS.

Y se demuestra su funcionamiento en entornos mixtos IPv4/IPv6, diseñando escenarios que hacen uso de los mecanismos de traducción o encapsulación explicados en el presente proyecto fin de carrera.

También se discuten y se implantan los protocolos de encaminamiento RIP y BGP en sus modalidades para IPv6 y se comenta la participación de nuestra red IPv6 de pruebas dentro de la red de pruebas multicast para IPv6 a nivel mundial, denominada m6bone.

Se finaliza el presente trabajo con la migración al nuevo protocolo de la herramienta generadora de paquetes IP denominada MGEN, junto con la migración de las siguientes aplicaciones:

- Servidor (FICS) y cliente de ajedrez (*xboard*).
- Implementación de una simulación de ping encapsulado utlizando el mecanismo de transición "6to4".

A lo largo de todo el proyecto se presentan recomendaciones para la implantación del nuevo protocolo, las dificultades que ralentizan su implantación y los beneficios que se pretenden obtener, así como las decisiones más importantes tomadas hasta la fecha.

De una forma más concreta, el presente proyecto fin de carrera describe el trabajo realizado dentro del Departamento de Ingeniería Telemática, relacionado con el nuevo protocolo de red IPv6. El marco dentro del cual se ha desarrollado este trabajo es el proyecto europeo LONG[**?**], proyecto IST.

El proyecto LONG trata de preveer y solucionar los problemas relacionados con el diseño, configuración y desarrollo de redes de telecomunicaciones de nueva generación, especialmente cuando se pretende proporcionar servicios a través de ellas. El protocolo IPv6 llegará a ser una parte integral de dichas redes. La proliferación de tecnologías de acceso de gran ancho de banda, como ADSL y 802.11, constituirán una parte esencial de las mismas. Por consiguiente, las aplicaciones deben ser conscientes de los servicios avanzados proporcionados por la nueva capa de red, y también se debe tener en cuenta el impacto de la red en dichas aplicaciones. Como resultado del proyecto LONG y del presente proyecto fin de carrera, se ha conseguido desarrollar una red IPv6 de pruebas, migrar determinadas aplicaciones a IPv6 y desarrollar guías para la gestión y migración de redes hacia el nuevo protocolo.

A lo largo de la realización del presente proyecto fin de carrera se ha trabajado con varias implementaciones de IPv6, sobre todo para los sistemos operativos *FreeBSD* y *Linux*. Durante la vida de este proyecto se han ido probando las distintas implementaciones y se ha colaborado activamente en las respectivas listas de distribución reportando *"bugs"* y respondiendo determinadas preguntas. El proyecto japonés denominado *"KAME"* realiza un activo desarrollo del nuevo protocolo, siendo líderes en en la implementación de IPv6 para las distintas distribuciones *BSD*. De forma semejante, el proyecto también japonés, denominado *"USAGI"*, realiza lo mismo para el núcleo de *Linux*. Nótese que las ramas oficiales de *BSD* y *Linux* ya poseen soporte IPv6, pero los proyectos japoneses realizan implementaciones de mecanismos avanzados que todavía se encuentran en fase experimental, como por ejemplo el soporte de IPSec, movilidad y determinados mecanismos de transición, por lo que se ha hecho uso de dichas implementaciones durante la realización del presente trabajo.

El proyecto fin de carrera que se presenta en los subsiguientes capítulos trata algunos puntos que se han desarrollado dentro del proyecto europeo LONG, junto con los temas que se presentan desglosados con más detalle a continuación:

Montaje de la red: se describe la topología de la red de pruebas utilizada durante la realización del presente proyecto fin de carrera. Montar una red consiste en diseñar la interconexión de varias máquinas de tal forma que se puedan comunicar a nivel de red entre ellas. Se realiza una distribución de las distintas subredes y se configuran adecuadamente los routers que actúan como pasarelas entre ellas.

Servicios de red: por servicios de red se entiende aquellos servicios que se ofrecen de cara a la propia red, para mejorar o posibilitar su correcto funcionamiento. Detro de esta categoría nos encontramos con los mecanismos de encaminamiento, el servicio de DNS, NFS, y servicios de transición/pasarela.

Servicios de aplicacion: como parte del projecto LONG se han implantado distintos servicios de aplicación dentro de nuestra red, como por ejemplo un servidor web, una red de servidores de IRC y varios juegos en red, como un cliente/servidor de ajedrez, de tetris y de *"Quake2"*, por citar sólamente los más importantes.

Migración de aplicaciones: en lo que atañe al presente proyecto fin de carrera, se ha migrado la aplicación MGEN. MGEN son las iniciales de *"Multi-Generator Toolset"*. Esta

herramienta ha sido desarrollada por *"The Naval Research Laboratory"* [**?**] y proporciona flujos de paquetes UDP/IP unicast y multicast. Las aplicaciones deben migrarse debido a que los protocolos IPv4 e IPv6 son incompatibles, de tal forma que, cuando IPv6 reemplace al protocolo IPv4, aquellas aplicaciones de red que hagan uso de *"sockets"* IPv4 no podrán comunicarse entre sí, al menos directamente, salvo si dichas aplicaciones se han reprogramado para tener "consciencia" del nuevo protocolo de red. Además, para que IPv6 se haga realidad en la futura Internet de nueva generación, necesita de aplicaciones que hagan uso de dicho protocolo de red, puesto que sin aplicaciones no hay usuarios, y sin usuarios el nuevo protocolo está condenado a su extinción. Por supuesto, las empresas de telecomunicaciones y de telefonía móvil serán claves en la introducción del nuevo protocolo en los próximos años, al ofrecer servios IP, para los cuales necesitarán la capacidad de direccionamiento que ofrece IPv6, pero el mercado de juegos en red y de la actual Internet debe aprovechar las nuevas capacidades del nuevo protocolo y anticiparse al cambio que, por otro lado, resulta inevitable.

Servicios de transición IPv4/IPv6: uno de los objetivos claves de este projecto consiste en probar la coexistencia de ambos protocolos de red. La introducción de IPv6 se está realizando de forma gradual, por consiguiente este aspecto resulta de vital importancia. En este trabajo se han evaluado las distintas posibilidades existentes, se realizan recomendaciones y se describen los pasos necesarios para su implantación. Como efecto colateral de este trabajo, la red del laboratorio proporciona algunos mecanismos de transición de una forma estable, como por ejemplo NAT-PT[**?**].

Desarrollo de aplicaciones: se ha desarrollado un programa denominado "ping6to4", que puede utilizarse para probar el correcto funcionamiento del mecanismo de transicion denominado 6to4[**?**], en el extremo remoto, sin que la máquina origen tenga implementado dicho mecanismo. Evidentemente, si ambos hosts tienen implementado el mencanismo de transición 6to4, un simple "ping" sería suficiente. El desarrollo de esta aplicación en las primeras etapas de la realización de este proyecto nos ha permitido conocer el nuevo interfaz de comunicaciones desarrollado para IPv6, y nos ha proporcionado una clara idea del esfuerzo y problemas asociados con la migración de aplicaciones, a la vez que nos ha preparado para afrontar retos mucho más complicados como la migración y ampliación de capacidades de la herramienta MGEN.

Mecanismos de transición. La descripción teórica de los mecanismos de transición constituye una gran parte del presente proyecto fin de carrera. Se introducen a nivel teórico los variopintos mecanismos de transición que el administrador de red tiene a su disposición para mantener la comunicación entre máquinas IPv4 e IPv6, y viceversa. Se prueban los mecanismo de transición más importantes, se presentan reglas para su correcta implantación y se realizan algunas pruebas para comprobar su correcto funcionamiento. También se describen los mecanismos que permiten obtener una dirección IPv6 a partir de una dirección IPv4, en el caso de que no se disponga de ningún proveedor de IPv6.

En el primer capítulo se describen brevemente las características principales del nuevo protocolo IPv6. Concretamente, se describen los motivos que han llevado a la creación de un nuevo protocolo incompatible con el actual IPv4. Se esclarecen y desmienten algunos equívocos que IPv6 parece llevar consigo desde el momento de su nacimiento, y finalmente se comenta la especificación del protocolo de forma resumida.

El siguiente capítulo está dedicado a los mecanismos de transición. Después de comentar su necesidad y las herramientas de las cuales hacen uso la mayoría de ellos, se realiza una descripción teórica de los mismos y se clasifican atendiendo a diferentes criterios, tales como el ámbito de aplicación, requisitos de direcciones IPv4 e IPv6, etc.

A continuación, se describe la configuración de cada uno de los mecanismos que han sido probados en el laboratorio. Sólamente se han probado aquellos mecanismos que disponían de una implementación para *Linux*, *Windows* o *FreeBSD* en el momento de realizar este trabajo. También se describen las pruebas realizadas en cada uno de ellos.

El siguiente capítulo, capítulo cuatro, está dedicado a la implantación de la red IPv6 dentro del laboratorio, utilizando las máquinas dedicadas al proyecto LONG. En este capítulo se describe la topología de la red de pruebas del laboratorio, el rango de direcciones utilizado, cómo se ha realizado la conexión al exterior y con otras entidades externas, y finalmente aspectos relacionados con el encaminamiento dinámico (fundamentalmente la configuración de los demonios de "RIPng" y "BGP4+").

A continuación se describen los servicios ofrecidos por IPv6 dentro de la red de pruebas. Fundamentalmente se han probado "nfs", "ftp", "ircd", "apache" y "named". Para ofrecer estos servicios se ha desarrollado uan arquitectura de implantación flexible, haciendo uso de mecanismos de traducción e inyectando determinadas rutas para atraer el tráfico hacia los traductores. Todo el software utilizado compila tanto en sistemas *Linux* como en sistemas *FreeBSD*, y pueden ser ofrecidos desde cualquiera de las máquinas dedicadas a la realización de este proyecto. Los servicios anteriormente citados pueden utilizarse desde fuera de la red universitaria por cualquier internauta con soporte IPv6, colaborando de este modo a la introducción de IPv6 en la actual internet. Por otro lado, otros muchos servicios han sido utilizados diariamente de una forma transparente por los distintos usuarios del laboratorio, como por ejemplo "ssh" o el servicio de web, junto con nombres de máquinas pertenecientes al dominio "ipv6.it.uc3m.es". Se prevé que en un futuro la red IPv6 del laboratorio ofrezca más servicios de aplicación de una forma estable.

El último capítulo está dedicado a la migración de aplicaciones. En él se describe la migración de la herramienta MGEN[**?**], del servidor de ajedrez FICS[**?**] y el desarrollo de la herramienta "ping6to4", esta última utlizada para poder efectuar "pings" a una interfaz que implementa el mecanismo de transición denominado "6to4" sin necesidad de tener implementado dicho mecanismo en la propia máquina. Finalmente se termina el presente proyecto fin de carrera con un capítulo de conclusiones y trabajos futuros.

Índice General

Índice de Figuras

Capítulo 1

Protocolo IP versión 6.

IPv6, el protocolo de Internet de nueva generación, fue aprobado por el IETF el 17 de Noviembre de 1994 como un *"Proposed Standard"*. Desde entonces, un gran número de organizaciones han estado trabajando en la especificación y prueba de las implementaciones de IPv6. Varios grupos de trabajo del IETF han trabajado para completar IPv6 , incluyendo la especificación de aspectos del nuevo protocolo, arquitecturas de direccionamiento, DNS, seguridad, mecanismos de transición e ICMP. Los principales fabricantes de routers han añadido IPv6 en sus productos, entre los cuales podemos citar a Nortel Networks, Hitachi, Ericsson Telebit y Cisco Systems. Fabricantes de ordenadores personales como Digital Equipment Corporation, Apple, Hewlett-Packard, Novell, y Sun Microsystems poseen ya equipos con IPv6[**?**]. Muchas organizaciones están trabajando para implementar IPv6 en sistemas operativos de código fuente libre (*Linux*, **BSD*, etc). Vendedores de aplicaciones de red han anunciado la migración a IPv6 de sus productos. Por si fuera poco, existe una red de pruebas denominada *"6bone"* que interconecta un gran número de dispositivos con IPv6 funcionando en todo el mundo, utilizando como capa de enlace a IPv4.

En este capítulo se va a comenzar por una descripción de la especificación del protocolo IPv6, cuáles son sus diferencias y similitudes con el anterior protocolo de Internet (IPv4), por qué resulta necesario y cuáles son sus ventajas e inconvenientes. En los siguientes capítulos se profundizará en los distintos elementos que forman parte del nuevo protocolo de red. Se hará incapié en los distintos mecanismos de transición que han sido definidos hasta el presente y se comentarán aspectos de diseño de red y de migración de aplicaciones. Otros elementos constitutivos del nuevo protocol, como movilidad, calidad de servicio, *multihoming* o seguridad no son objeto de estudio de este proyecto fin de carrera.

1.1 Necesidad de un nuevo protocolo.

La población mundial no ha cesado de aumentar, y sean cuales sean las condiciones económicas de la época, se debe pensar y tener en cuenta la capacidad de acceso potencial a la red de todos los habitantes del planeta (recordemos que el objetivo primigenio de Internet no es otro que el construir una gran red de interconexión global). Es decir, no se considera aceptable producir una tecnología que no escale para todos los posibles usuarios, bajo condiciones económicas apropiadas.

Más aún, la conexión de distintos dispositivos (teléfonos móviles, electrodomésticos, consolas de videojuegos, etc) a la red supondrá la existencia de varios "elementos IP" por persona. Si se echan cuentas vemos que el máximo de 4 billones de direcciones públicas permitidas por la actual versión 4, incluso con el uso de técnicas (que presentan ciertos incovenientes) como NATs, serán insuficientes en un futuro próximo. Aparte del espacio de direccionamiento, IPv6 también tiene otros beneficios, como la provisión de autoconfiguración, o mecanismos *"plug and play"*, que promete reducir la complejidad de administración de redes IP. Pero más aún, el principal beneficio de IPv6 es que al tener direcciones suficientes, se podrá restaurar el modelo *end-to-end* sobre el cual Internet fue construido.

El nacimiento de IPv6 se ha visto acompañado por una gran cantidad de promesas sin fundamento y de malentendidos sobre el nuevo protocolo. Pero después de la fase de optimismo desbordado, conviene examinar detenidamente las implicaciones y las mejoras que IPv6 nos proporciona.

Básicamente, IPv6 se introducirá en Internet gracias a la escasez de direcciones IPv4. Debe quedar claro que si IPv4 tuviera un rango de direcciones mayor, nunca se reemplazaría. Por consiguiente, la única característica realmente importante y que juega a favor de IPv6 es su amplio rango de direccionamiento.

Aunque pueda resultar un poco decepcionante, IPv6 no constituye el clímax de las redes telemáticas, pero sin duda es superior a su predecesor en cuanto a diseño, extensibilidad, direccionamiento y movilidad. Además, su implantación resultará irremediable a corto o medio plazo, como ya se ha comentado.

1.2 Características y especificación del protocolo IPv6.

A continuación se va a comentar brevemente la especificación del protocolo IPv6. Este resumen se basa en [**?**].

El protocolo IP versión 6 (IPv6) es una nueva versión del protocolo de Internet, diseñada con el objetivo de reemplazar al exhausto protocolo IPv4 [**?**]. Los cambios realizados respecto al protocolo IPv4 se pueden resumir en las siguientes áreas:

- Expansión del espacio de direcciones.
- Simplificación del formato de la cabecera.
- Soporte mejorado para extensiones y opciones.
- Capacidad de etiquetado de flujos.
- Capacidad de autenticación y privacidad.

En la figura 1.1 de la página 17 se puede observar el formato de la cabecera de IPv6:

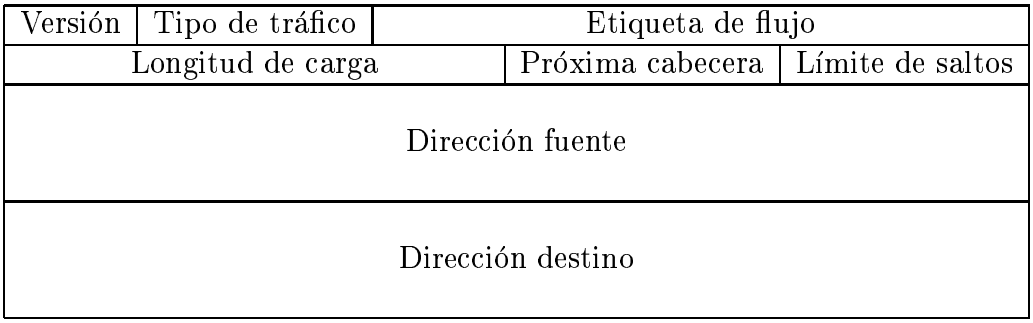

Figura 1.1: Formato de la cabecera IPv6.

La arquitectura de cabeceras de extensión de IPv6 reemplaza el campo de opciones de IPv4 y también impacta en el campo "tipo de protocolo", que es utilizado en IPv4 para indicar el tipo de protocolo dentro del *"payload"* (por ejemplo, TCP o UDP). IPv6 reemplaza el campo "tipo de protocolo" con un campo de "siguiente cabecera" que indica el protocolo almacenado en la siguiente cabecera de extensión o *"payload"*.

1.2.1 Cabeceras de extensión.

A continuación se describen las cabeceras de extensión definidas dentro del protocolo IPv6.

Hop-by-hop options header.

La cabecera de extensión *hop-by-hop* presenta el formato que se puede observar en la figura 1.2. Este formato es el mismo que se utiliza para la definición de las opciones de destino, como se explicará más adelante:

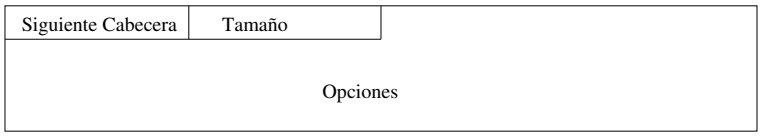

Figura 1.2: Formato de la cabecera de *"hop-by-hop"*.

Cuando está presente, esta cabecera lleva opciones que son examinadas por los nodos intermedios a lo largo de la ruta que sigue el paquete. Como esta cabecera se lee por todos los routers del camino, es útil para transmitir información de gestión de red o comandos de depuración de routers. Un tipo de opción *hop-by-hop* definida en la especificación del protocolo IPv6 es la *"jumbo payload option"*, utilizada para extender el tamaño del campo de datos o *payload*, que por defecto se encuentra limitado a 65535 bytes. Existen otras opciones *hop-by-hop*, como por ejemplo la opción de *"router alert"*, la cual informa a los routers de que el paquete debería ser procesado en profundidad antes de ser reenviado al siguiente salto. Un ejemplo de tal paquete puede ser, por ejemplo, un mensaje de reserva de recursos de RSVP. Cada opción posee un tamaño variable, pero todas se definen de acuerdo con el formato TLV o *"Type-Length-Value"*, como se muestra en la figura 1.3 de la página 18:

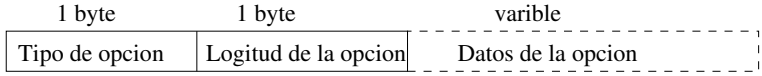

Figura 1.3: Formato de todas las opciones.

Destination option header.

Esta cabecera posee exactamente el mismo formato que la cabecera *hop-by-hop*, y las opciones que transporta se codifican utilizando el mismo formato TLV explicado anteriormente. Las cabeceras de destino están pensadas para añadir nueva funcionalidad al protocolo IPv6. Otra forma de añadir funcionalidad consiste en definir una nueva cabecera de extensión, pero dicha opción presenta los siguientes problemas:

- Requiere la asignación de números para el campo "tipo de cabecera", que se encuentran limitados a 256 y por tanto son un recurso escaso.
- Requiere que las máquinas origen y destino comprendan la opción, si su procesamiento no está implementado, el comportamiento por defecto ante una cabecera de extensión desconocida consiste en descartar el paquete IPv6 por completo.

Tanto las cabeceras de destino como las *hop-by-hop* transportan opciones de tamaño variable, pero se debe recalcar que el conjunto formado por la cabecera IPv6 más las cabeceras de extensión debe estar alineado a 64 bits, por lo que se definen dos opciones que se utilizan para rellenar los posibles huecos. El formato de estas opciones se muestra en la figura 1.4:

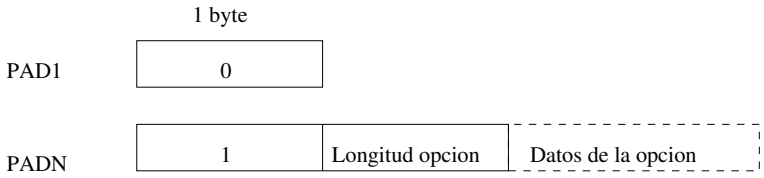

Figura 1.4: opciones PAD1 y PADN.

La cabecera de destino puede aparecer en dos posiciones diferentes dentro de la cadena de cabeceras de extensión que forman el paquete IP. En la primera posición se utiliza para enviar información a todos los nodos intermedios listados en la cabecera de *"source routing"* o para enviar información al destino final, si no existe cabecera de *"source routing"*. En la segunda posición se utiliza para transmitir información opcional que es leida sólo por el destino final.

En la primera posibilidad se coloca hacia el frente de la cadena de cabeceras, justo despues de la cabecera *hop-by-hop* (si existe), mientras que en la segunda posibilidad se relega a una posición hacia el final de la cadena de cabeceras IPv6, siendo la última extensión justo antes de la cabecera de transporte y el *payload*.

Source routing header.

La función de esta cabecera de extensión es semejante a la función de encaminamiento basado en fuente soportada actualmente por IPv4. Esta cabecera de extensión permite a un nodo

origen especificar una lista de direcciones IP que dictan qué camino va a seguir un determinado paquete, como se observa en la figura 1.5:

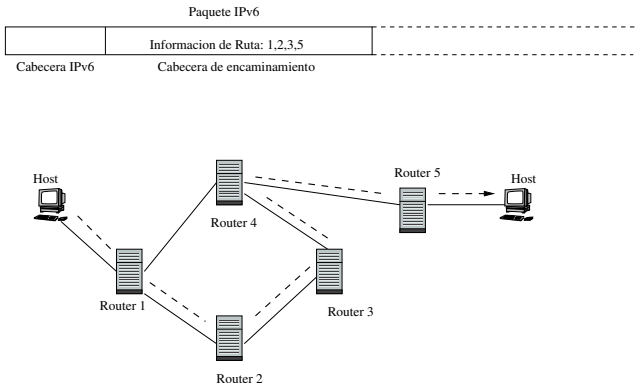

Figura 1.5: Funcionamiento de la cabecera de *"routing header"*.

En [**?**] se define una version de esta cabecera de extensión denominada *"type 0"*, con el formato que se muestra en la figura 1.6:

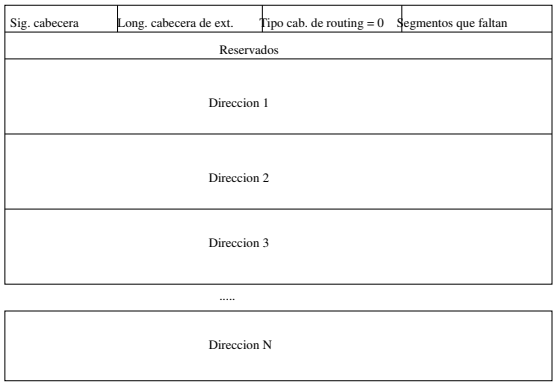

Figura 1.6: Formato de la cabecera de *"routing header"* de tipo 0.

El campo de "siguiente cabecera" identifica el tipo de cabecera que se encuentra a continuación de la de *"routing header"*. El campo de "longitud de la cabecera de extensión" indica la longitud de dicha cabecera en unidades de ocho octetos, sin incluir los primeros ocho octetos. Para el caso de la cabecera de tipo 0, este campo resulta ser igual a dos veces el número de direcciones que contiene la cabecera. Por último, el campo de "segmentos que faltan" indica el número de nodos intermendios que todavía necesitan ser visitados.

Cuando se utiliza la cabecera *"type 0"*, la cabecera IPv6 inicial contiene como dirección destino la dirección del primer router en el camino, no contiene el destino final. En cada salto, los nodos intermedios reemplazan la dirección destino con la dirección del siguiente nodo del camino y el campo *"segments left"* se decrementa.

Fragmentation header.

IPv4 tiene la habilidad de fragmentar paquetes en cualquier punto del camino, dependiendo de las capacidades de transmisión de los enlaces involucrados. Esta característica ha sido

deshabilitada en favor de una fragmentación/reensamblado extremo a extremo, que se ejecuta sólo en los nodos origen y destino. La fragmentación de paquetes no se permite en nodos intermedios. La eliminación del campo de fragmentación de la cabecera IPv6 permite enviar paquetes serializados y consigue un mejor rendimiento de los routers cuando no se requiere fragmentación, lo cual ocurre en la mayoría de los casos. Las redes de hoy en día generalmente soportan tamaños de *"frames"* suficientemente grandes como para transmitir paquetes IP sin fragmentación. En el caso de que se requiera fragmentación, IPv6 proporciona una cabecera de extensión opcional que se utiliza en el nodo fuente para dividir el paquete en un número arbitrario de paquetes más pequeños. La cabecera IPv6 de framentación contiene campos que identifican un determinado grupo de fragmentos como pertenecientes a un único paquete y les asigna números de secuencia.

El formato de dicha cabecera se muestra en la figura 1.7:

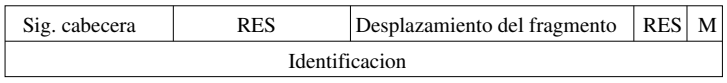

Figura 1.7: Formato de la cabecera de fragmentación.

El campo "M" indica si hay más fragmentos. el resto de campos son equivalentes a los presentes en el protocolo IPv4, salvo pequeñas variaciones en el tamaño.

Debido a que los routers IPv6 no realizan fragmentación, la responsabilidad de enviar un paquete con un tamaño apropiado recae en los nodos origen, los cuales necesitan determinar la MTU de los enlaces que se atraviesan en el camino desde el nodo origen al nodo destino, o utilizar la MTU mínima definida en 1280 bytes. Los nodos pueden determinar la MTU más pequeña del camino con el proceso denominado "MTU discovery"[**?**]. Típicamente, con esta técnica, el nodo origen envía un paquete con una MTU tan grande como la interfaz local pueda soportar. Si esta MTU es demasiado grande para algún enlace a lo largo del camino, un mensaje ICMP de "Packet too big" será recibido. Este mensaje contendrá el MTU del link afectado. El nodo origen puede entonces ajustar el tamaño del paquete y retransmitir otro paquete de prueba. Este proceso se repite hasta que un paquete alcance al nodo destino. La MTU "descubierta" se utiliza a partir de este momento para fragmentar los paquetes. Aunque la fragmentación basada en fuente está totalmente soportada en IPv6, se recomienda que las aplicaciones de red ajusten el tamaño del paquete para acomodarse a la MTU más pequeña del camino. Esto evitará la sobrecarga asociada con la fragmentación/reensamblaje en los nodos origen y destino. Dado que puede ser necesario transmitir la información de MTU a las aplicaciones, se han implementado opciones de socket apropiadas[**?**].

Authentication header.

La actual carencia de un esquema de seguridad estandarizado a nivel de red es una de las mayores deficiencias de la actual Internet IPv4. El estándar de IPv6 soluciona esta situación con dos importantes cabeceras de extensión, una que habilita la autenticación del tráfico IP y otra que total o parcialmente encripta paquetes IP. Se hace notar que la implementación de la seguridad a nivel IP puede beneficiar tanto a aplicaciones "seguras" como a aplicaciones "ignorantes" o "no-seguras" que no traten de forma específica la seguridad.

La cabecera de extensión de autenticación, también denominada AH, siglas de *"authentication header"*, da a las aplicaciones de red una garantía de que el paquete ha llegado de una fuente auténtica. Este mecanismo combate la cada día mayor afición de *"hackers"* de configurar la dirección IP de una máquina para suplantar a otra y ganar así acceso a recursos en teoría seguros (*"spoofing"*). Utilizando las cabeceras de autenticación de IPv6, los nodos establecen una asociación estandar de seguridad que se basa en el intercambio de claves privadas (por ejemplo, MD5). El formato de la cabecera de autenticación es el que se muestra a continuación, figura 1.8:

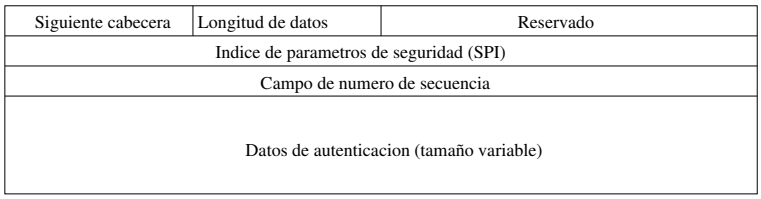

Figura 1.8: Formato de la cabecera de autenticación.

En una sesión cliente/servidor, por ejemplo, tanto el cliente como el servidor necesitan tener conocimiento de la clave. Antes de enviar cada paquete, la autenticación de IPv6 crea un *"checksum"* basado en la clave y combinado con el contenido completo del paquete. Este *"checksum"* se recalcula en el extremo receptor y se compara el resultado. Esta aproximación proporciona autenticación del emisor y grantiza que los datos dentro del paquete no han sido modificados por un tercero. La autenticación puede tener lugar entre clientes y servidores o entre los mismos clientes.

Encryption header.

Las cabeceras de autenticación eliminan los ataques de *"spoofing"* y ataques de modificación de paquetes, pero no proporcionan seguridad respecto a la lectura no-contaminante (*"sniffing"*, *"snooping"*) del contenido del paquete. Esta área se trata por el servicio ESP de IPv6. Esta cabecera de extensión proporciona varios niveles de privacidad e integridad, algo que no se encuentra disponible en la actual internet salvo para determinadas aplicaciones "seguras" (correo electrónico y servidores webs seguros). ESP proporciona encriptación a nivel de red para todas las aplicaciones de una forma altamente estandarizada. Se puede ver un esquema en la figura 1.9 de la página 22.

Concretamente, ESP se utiliza para encriptar la cabecera de transporte y el *payload*, o para encriptar el detagrama IP en su totalidad. Ambos métodos se realizan con una cabecera de extensión que lleva los parametros de encriptación y las claves extremo a extremo. Cuando sólo el *payload* va a ser encriptado, la cabecera ESP se inserta en el paquete IP justo antes de la cabecera de transporte. En este caso, las cabeceras anteriores a la ESP no están encriptadas y las cabeceras y el *payload* a continuación del ESP si lo están. A este tipo de encriptación se la denomina *"transport-mode encryption"*. Si se desea encriptar el datagrama IP por completo, una nueva cabecera IPv6 junto con una cabecera ESP se adjuntan alrededor de todo el paquete. Este tipo de encriptación algunas veces se denomina *"tunnel-mode encryption"* porque los contenidos del datagrama únicamente son visibles en los extremos del túnel. Se considera que éste último tipo de encriptación es más seguro porque ningún campo del paquete IP está disponible para ser analizado.

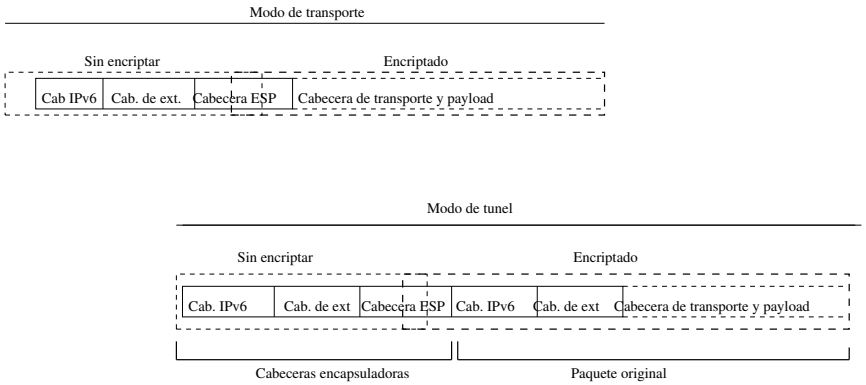

Figura 1.9: *"Tunnel Mode"* y *"Transport Mode"* de la cabecera de encriptación de IPv6.

Los servicios de encriptación y autentificación de IPv6 trabajan conjutamente para crear distintas y potentes soluciones al tema de la seguridad. En algunos casos la cabecera de autenticación se utilizará dentro de un paquete totalmente encriptado, porporcionando una capa adicional de integridad de datos y verificación de la identidad del emisor. En otros casos, la cabecera de autenticación se puede situar al principio del paquete encriptado utilizando el *"transport-mode"*. Esto es deseable cuando la autenticación tiene lugar antes de que el extremo receptor descifre el paquete. De esta forma, la autenticación y la encriptación de IPv6 proporcionan una arquitectura de seguridad que se espera que juegue un papel crítico en la continua expansión del comercio electrónico y de las operaciones corporativas en la actual y futura red.

Orden de las cabeceras de extensión.

A diferencia de la cabecera IPv4, la cabecera IPv6 siempre ocupa 40 bytes. Si se quieren añadir opciones, se añaden nuevas cabeceras encadenándose unas con otras. Este esquema proporciona flexibilidad y adaptabilidad al nuevo protocolo, al mismo tiempo que rapidez de procesamiento pues no todos los nodos intermedios tienen que leer todas las cabeceras. El orden de las cabeceras es el que se muestra en la figura 1.10:

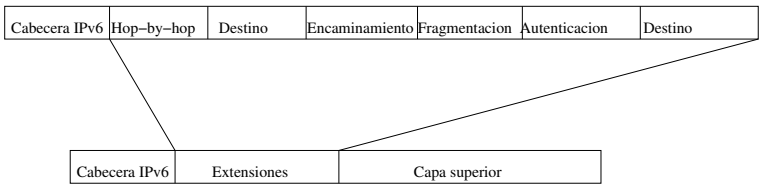

Figura 1.10: Orden de las cabeceras de extensión.

Concretamente, el orden es el siguiente:

1. Cabecera IPv6.
2. Cabecera de opciones *"hop-by-hop"*.
3. Cabecera de opciones de destino. (nota 1)
4. Cabecera de encaminamiento.
5. Cabecera de fragmentación.
6. Cabecera de autenticación. (nota 2)
7. Cabecera ESP. (nota 2)
8. Cabecera de opciones de destino. (nota 3)
9. Cabecera de la capa superior (UDP, TCP, etc).

nota 1: para opciones que van a ser procesadas por el primer destino que aparece en el campo destino más todos los destinos listados en la *"routing header"*.

nota 2: la especificación de IPsec propone recomendaciones adicionales sobre el orden relativo a las cabeceras IPsec.

nota 3: para opciones que van a ser procesadas sólamente por el destino final del paquete.

Para una descripción más detallada de las cabeceras y su formato se remite al lector a [**?**].

Resulta interesante hacer notar, llegados a este punto, que la actual especificación de IPv6 define el orden anterior para las cabeceras de extensión pero no es muy clara respecto al orden entre las diferentes *"destination option headers"* que puede transportar un determinado paquete IPv6. La única restricción al esquema anteriormente mostrado es que la *"hop-by-hop option header"* sólo puede ocurrir una vez y justo después de la cabecera IPv6. Para más información sobre el orden de las cabeceras y los problemas asociados se remite al lector a [**?**].

1.2.2 Modelo de direccionamiento.

En este apartado se va a comentar el modelo de direccionamiento en IPv6, cómo se representan en texto las direcciones y cuáles son las direcciones que un nodo IPv6 necesita manejar obligatoriamente. También se comentan los distintos tipos de direcciones IPv6 definidas y cómo se contruyen y utilizan. Este apartado se encuetra basado principalmente en [**?**].

Representación de direcciones.

Existen dos formas básicas de representación de direcciones:

- Formato estándar: se contruye utilizando números hexadecimales agrupados de cuatro en cuatro y separados por ":", por ejemplo:
  - 3452:fe78:a375:0000:0000:002c:79e0:0040

  Dentro de cada grupo, los ceros a la izquiera pueden obviarse, por consiguiente, el ejemplo anterior podría escribirse tambien así:
  - 3452:fe78:a375:0:0:2c:79e0:40

  Además de esta forma de representación, también existe la notación corta o abreviada, que se puede utilizar cuando existen varios grupos de ceros en cuyo caso se sustitituyen todos los ceros por "::". Esto sólo se puede realizar una vez, para no dar lugar a equívocos. En este caso, nuestro ejemplo quedaría así:
  - 3452:fe78:a375::2c:79e0:40
- Formato compatible con IPv4: se construye escribiendo los últimos 32 bits (o los últimos dos grupos hexadecimales) utilizando la notación de IPv4, separando cada byte por ".". Retomando nuestro ejemplo, también podríamos haber escrito:
  - 3452:fe78:a375::2c:121.224.0.64

  Donde 121.224.0.64 se corresponde con "79e0:40" en hexadecimal.
- Si se quiere especificar el prefijo, este se separa de la dirección mediante "/", por ejemplo:
  - 3452:fe78:a375::2c:121.224.0.64/64
- Si sólamente se quiere escribir el prefijo, el resto de bits se ponen a cero, y utilizando la notación corta, se puede escribir, por ejemplo:
  - 3452:fe78:a375::/64
- Por último, la representación de direcciones junto con el puerto se define en [**?**] para URLs, y consiste en introducir la dirección IP entre corchetes y separar el puerto mediante ":", como se observa en el siguiente ejemplo:
  - [3452:fe78:a375::2c:121.224.0.64]:80

Tipos de direcciones.

Las direcciones IPv6 constituyen identificadores de 128 bits. Las direcciones se asignan a interfaces, no a nodos. Una dirección unicast se refiere a una única interfaz. Como cada interfaz pertenece sólamente a un determinado nodo, cualquiera de las direcciones unicast de los interfaces de un nodo puede ser utilizada como identificador para dicho nodo.

Atendiendo a la función de identificador, las direcciones IPv6 se clasifican en los siguientes grupos:

- Unicast: se trata de un identificador para una única interfaz. Un paquete enviado a una dirección unicast se encamina a la interfaz identificada por dicha dirección.
- Anycast: se trata de un identificador para un conjunto de interfaces (normalmente pertenecientes a distintos nodos). Un paquete enviado a una dirección anycast se encamina a uno de los interfaces identificados por dicha dirección (concretamente, hacia la interfaz "más cercana", según la medida de la distancia utilizada por el protocolo de encaminamiento).
- Multicast: se trata de un identificador para un conjunto de interfaces (normalmente pertenecientes a distintos nodos, como en el caso anterior). En este caso, los paquetes enviados a una dirección multicast son recibidos por todos los interfaces identificados por dicha dirección. No existen direcciones de broadcast en IPv6, su función ha sido reemplazada por estas direcciones.

Las siguientes direcciones son consideradas como "direcciones unicast especiales":

- dirección sin especificar: esta dirección se corresponde con "0:0:0:0:0:0:0:0" o en formato corto con "::". No se puede emplear como dirección destino de un paquete IPv6, ni en la cebecera de encaminamiento. Esta dirección resulta útil para utilizarla como origen cuando el nodo está inicializándose y todavía no sabe cuál es su dirección IPv6.
- dirección de loopback: se corresponde con "0:0:0:0:0:0:0:1" o en formato corto con "::1". No se puede asignar a ninguna interfaz física, puesto que identifica a la propia máquina. Un paquete IPv6 que lleve la dirección de loopback como origen o destino no puede ser enviado fuera de la propia máquina.

El tipo de dirección IPv6 viene determinado por los primeros bits de ésta. En el siguiente extracto se pueden observar las asignaciones actualmente registradas, obtenidas de [**?**]:

```
Address type         Binary prefix        IPv6 notation
------------         -------------        -------------
Unspecified          00...0  (128 bits)   ::/128
Loopback             00...1  (128 bits)   ::1/128
Multicast            11111111             FF00::/8
Link-local unicast   1111111010           FE80::/10
Site-local unicast   1111111011           FEC0::/10
Global unicast       (everything else)
```

Las direcciones anycast se toman del espacio reservado para direcciones unicast (de cualquier ámbito) y son sintácticamente indistinguibles.

Se han definido algunos subtipos específicos dentro de las direcciones unicast globales, sobre todo para el propósito de mecanismos de transición IPv6-IPv4, como por ejemplo las direcciones ISATAP, las direcciones 6to4, las direcciones Teredo o las direcciones traducidas, por citar sólo algunos ejemplos. Estas direcciones serán comentadas en profundidad cuando se expliquen los mecanismos de transición que hacen uso de ellas. Además de estas direcciones, existen dos tipos que merece la pena comentar más detenidamente. Son las siguientes:

- Direcciones IPv6 compatibles con IPv4: también conocidas como "direcciones compatibles", se utilizan en el mecanismo de transición denominado "túneles automáticos" para realizar una encapsulación automática en IPv4. Estas direcciones se construyen utilizando una dirección IPv4 y prefijando noventa y seis ceros, por ejemplo: "::163.117.140.166".

- Direcciones IPv4 mapeadas a IPv6: también conocidas como "direcciones mapeadas", se utilizan para representar en los *"sockets"* IPv6 la recepción o el envío de paquetes IPv4. Se construyen utilizando el prefijo "0:0:0:0:0:ffff::/96" y añadiendo a continuación la dirección IPv4. Por ejemplo: "::ffff:163.117.140.166".

Futuras especificaciones puede redefinir uno o más subrangos, pero hasta que dichas definiciones se aprueben, las implementaciones deben tratar todas las direcciones que no comienzan por los prefijos listados en la tabla anterior como direcciones unicast globales.

Direcciones *link-local* y *site-local.*

Atendiendo al ámbito o alcance, las direcciones IPv6 unicast se subdividen en *link-local*, *site-local* y *global*. Las direcciones *link-local* y la direcciones *site-local* se corresponden con direcciones locales a la subred y a la organización, respectivamente. En las siguientes figuras, figuras 1.11 y 1.12 se observa el formato de cada una de ellas:

Figura 1.11: Formato de direcciones link-local.

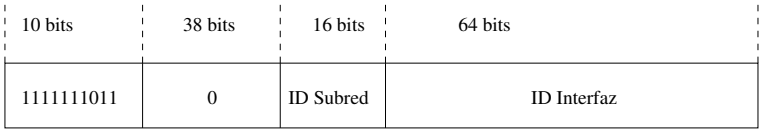

Figura 1.12: Formato de direcciones site-local.

Ambas direcciones no pueden utilizarse en Internet, es decir, los routers no deben permitir el paso de paquetes con direcciones *link-local* o *site-local*. El propósito consiste en brindar

conectividad IP a una organización o a una red local. Cuando una máquina se autoconfigura mediante el mecanismo de autoconfiguración sin estado, adquiere automáticamente una dirección *link-local*.

Las direcciones *site* y *link-local* están disponibles para uso privado interno, y no necesitan ser asignadas por autoridades públicas de registro o terceras partes. Las direcciones *site-local* constituyen un mecanismo flexible para que las redes de una misma empresa u organización con un ámbito bien definido se comuniquen entre sí. Si se cambia de ISP, por ejemplo, el direccionamiento *site-local* permanece exactamente igual porque no está directamente relacionado con el mundo exterior.

Las direcciones *link-local* pueden utilizarse para aplicaciones que están limitadas a un determinado enlace, y también para el proceso de arranque a través de la red antes de que las máquinas reciban una dirección identificativa global.

Direcciones multicast.

Las direcciones multicast identifican a un conjunto de interfaces. El formato es el de la figura 1.13:

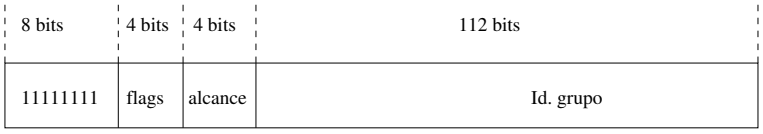

Figura 1.13: Formato de direcciones multicast.

Las direcciones multicast no pueden ser utilizadas como direcciones fuente ni aparecer en cabeceras de encaminamiento. Además, los routers no deben reenviar ningún paquete multicast más alla del ámbito o alcance indicado por el campo "ámbito" en la propia dirección multicast.

Los campos que aparecen en la figura se describen a continuación:

Flags: los tres primeros bits se encuentran siempre inicializados a cero, mientras que el último nos indica si la dirección es una asignación temporal (uno) o permanente (cero).

Alcance: campo que delimita el radio de acción de los grupos multicast. Los valores asignados actualmente se pueden consultar en [**?**].

Id. de grupo: como su propio nombre indica, es el identificador del grupo multicast.

Existen un conjunto de direcciones multicast "bien conocidas", que se enumeran a continuación:

- Direcciones multicast reservadas:
  - ff0x:0:0:0:0:0:0:0 (con x = 0..9)
- Direcciones multicast *"all-nodes"*: estas direcciones identifican el grupo de todos los nodos IPv6, con ámbito local a la interfaz (1) o ámbito *link-local* (2).

  - ff01:0:0:0:0:0:0:1
  - ff02:0:0:0:0:0:0:1

- Direcciones multicast *"all-routers"*: estas direcciones identifican el grupo de todos los router IPv6, con ámbito local a la interfaz (1), ámbito *link-local* (2) o ámbito *site-local* (5).
  - ff01:0:0:0:0:0:0:2
  - ff02:0:0:0:0:0:0:2
  - ff05:0:0:0:0:0:0:2
- dirección multicast *"solicited-node"*: estas direcciones se calculan como una función de las direcciones unicast y anycast. Una dirección multicast *"solicited-node"* se contruye tomando los 24 bits de menor orden de la dirección (unicast o anycast) y añadiendo dichos bits al prefijo "ff02:0:0:0:0:1:ff00::/104". Por ejemplo, la dirección *"solicited-node"* correspondiente a la dirección IPv6 "4037::01:800:200e:8c6c" es "ff02::1:ff0e:8c6c".

Direcciones requeridas por un nodo IPv6.

Las siguientes direcciones deben ser reconocidas por todas las máquinas como pertenecientes a sí mismas:

- Una dirección *link-local* para cada interfaz: es obligatorio que cada interfaz IPv6 posea al menos una dirección *link-local*.
- Cualquier dirección adicional que haya sido configurada en los interfaces.
- La dirección de *loopback*.
- La dirección multicast *"all-nodes"* ff02::1.
- La dirección multicast *"solicited-node"* para cada una de las direcciones unicast y anycast de la máquina.
- las direcciones multicast de todos aquellos grupos a los cuales se haya podido unir la máquina.

Un router está obligado a reconocer las mismas direcciones que una máquina reconoce como propias, junto con las siguientes direcciones:

- La dirección multicast *"all-routers"*.
- La direcciones anycast *"subnet-router"* para todos los interfaces para los cuales el router actúa como tal.
- Cualquier otra dirección anycast que haya sido configurada para el router.

1.2.3 Política de distribución de direcciones.

En este apartado se introducen los distintos modelos propuestos hasta la fecha para asignar o delegar[1] el espacio de direcciones de IPv6.

En primer lugar se comentan los objetivos que cualquier política de distribución de direcciones debe intentar alcanzar. Teniendo en cuenta el enorme espacio de direccionamiento que IPv6 pone a nuestra disposición, el objetivo más importante del modelo de asignación de direcciones consiste en mantener en la medida de lo posible el crecimiento de las tablas de rutas bajo control.

Después de una breve introducción a los problemas de encaminamiento surgidos con la delegación de direcciones en IPv4, se continúa describiendo el modelo de asignación incial para el nuevo protocolo IPv6, modelo inicial que aunque ya obsoleto se describe en este apartado por su interés técnico e histórico. Para terminar se comenta de forma resumida el nuevo modelo de jerarquización de direcciones que parece imponerse en la actual Internet IPv6, un modelo menos rígido y más flexible. No está claro que este modelo vaya a ser el definitivo, puesto que se han producido numerosos cambios en los últimos años y todavía existe discusión en ambientes relacionados con la toma de decisiones a este respecto, lo cual es bastante lógico, puesto que IPv6 se encuentra en sus primeras fases de introducción y las delegaciones y asignaciones de direcciones deben ser mecanismos adaptables a lo largo del tiempo, en función de la demanda y los recursos disponibles en cada momento. A este respecto, se presupone que IPv6 debe poseer al menos la misma capacidad de evolución que IPv4 ha demostrado a lo largo de sus más de veinte años de existencia.

Objetivos del sistema de registro de Internet.

Los objetivos que se van a describir a continuación han sido formulados por la comunidad de Internet de forma explícita para el espacio de direcciones de IPv6. Tratan de reflejar el interés de todos los miembros de la comunidad por extender la funcionalidad y el crecimiento de Internet lo máximo posible. Es responsabilidad de cada entidad de registro asegurar que todas las asignaciones y delegaciones del espacio de direcciones IPv6 sean consistentes con los siguientes objetivos:

- Unicidad: toda dirección global IPv6 unicast debe ser única. Se trata de un requisito imprescindible para garantizar que toda máquina de Internet pueda ser identificada.
- Agregación: las direcciones IPv6 deben ser distribuidas de una forma jerárquica, permitiendo de esta forma la agregación de información de encaminamiento y limitando el número de rutas anunciadas hacia Internet. Este requisito resulta necesario para asegurar un funcionamiento correcto del sistema de encaminamiento y para asegurar un crecimiento controlado de las tablas de rutas. En IPv6, la agregación de rutas externas es el principal y más importante objetivo que se debe asegurar y cumplir.

[1]Nótese que se trata de dos conceptos distintos. Delegar significa distribuir el espacio de direcciones a determinadas entidades de registro con el propósito de ser distribuido otra vez por ellas, mientra que por asignar entendemos la delegación del espacio de direcciones a ISPs o usuarios finales para un uso determinado dentro de la infraestructura de Internet en la cual dichas entidades operan. Las asignaciones deben realizarse sólo para propósitos específicos claramente documentados y no pueden ser subasignadas a terceras partes.

- Utilización eficiente del espacio de direcciones: aunque el espacio de direcciones de IPv6 puede considerarse casi infinito, dicho espacio no debe ser desperdiciado. De forma específica, aunque la conservación del espacio de direcciones de IPv6 no es "por sí mismo" un requisito, las entidades de registro deben implementar políticas y guías que impidan a las organizaciones derrochar direcciones. Para ello, las entidades de registro deben ubicar espacio de direcciones sólamente en función de necesidades que puedan ser demostradas.

- Registro: cada asignación y delegación del espacio de direcciones IPv6 debe ser registrado en una base de datos públicamente accesible. Esto resulta necesario para asegurar la unicidad y para proporcionar información. Cada entidad de registro, denominada por sus siglas inglesas RIR, mantendrá su propia base de datos con las ubicaciones de direcciones realizadas.

- Equidad y simplicidad: todas las políticas y reglas relacionadas con el uso del espacio público de direcciones se deben aplicar de una forma sencilla, clara y equitativa a todos los miembros actuales y futuros de la comunidad de Internet, sin importar su localización, nacionalidad, tamaño o cualquier otro factor.

- Minimizar la sobrecarga de trabajo: es deseable minimizar la sobrecarga de trabajo asociada con la obtención del espacio de direcciones. Hablamos de la sobrecarga relacionada con la necesidad de solicitar a los RIRs espacio de direccionamiento adicional de una forma frecuente, o de la sobrecarga asociada con la gestión de un espacio de direcciones que crece a través de un gran número de pequeñas expansiones sucesivas en lugar de a través de pocas pero grandes expansiones, por poner sólo unos ejemplos.

Política de distribución de direcciones en IPv4.

La especificación inicial del protocolo IPv4[**?**] define un esquema de direcciones basado en "clases", el cual divide los bits de una dirección en bits de red y bits de máquina o *host*. Este esquema, junto con una asignación de direcciones sin control y la utilización de mecanismos de *multihoming*, ha provocado un aumento peligroso en el número de rutas que deben soportar los routers del *"backbone"* de Internet.

Estas limitaciones del direccionamiento de IPv4 impactan en la red a nivel global y a nivel local. Para combatir las deficiencias de IPv4 a nivel de área local, la técnica de *"subnetting"* se ha desarrollado para crear una división más granular de redes grandes. Utilizando *"subnetting"*, una única dirección de red puede soportar varias subredes físicas, aprovechando más el espacio de direccionamiento obtenido.

Al nivel del encaminamiento global del *"backbone"* de Internet, las direcciones IPv4 pueden ser agregadas de una forma más eficiente utilizando *"supernetting"* para aprovechar un reparto de direcciones jerárquico. Con el *"supernetting"*, los routers del *"backbone"* almacenan una única dirección que representa el camino a un determinado número de redes de más bajo nivel. Esto puede reducir considerablemente el tamaño de las tablas de rutas de los routers centrales, lo que incrementa el rendimiento y disminuye además la cantidad de memoria que dichos routers necesitan. *"Subnetting"* y *"supernetting"* han sido particularmente útiles para extender la usabilidad de las direcciones IPv4 de tipo "C". Estas técnicas son posibles gracias al emparejamiento de direcciones con máscaras de bits que indican qué bits en la dirección son válidos en los diferentes niveles de la jerarquía.

Este procedimiento de construcción de una jerarquía de encaminamiento adaptable para IPv4 ha sido formalizado en CIDR[**?**]. Este mecanismo ha extendido la vida de las direcciones IPv4 y ha ayudado a Internet a escalar hasta alcanzar su tamaño actual, pero no ha podido ser implementado de una forma eficiente debido a una mala política de distribución de direcciones, que ha impedido un mayor nivel de agregación.

Política *inicial* de distribución de direcciones en IPv6.

En respuesta directa a la experiencia obtenida de IPv4, para IPv6 se propuso inicialmente un espacio de direcciones altamente escalable que pudiera ser particionado obteniendo una jerarquía global de direccionamiento flexible y a la vez eficiente. Lo que se va a contar a continuación esta basado en [**?**].

Este documento describe un sistema de registro para distribuir globalmente el espacio de direcciones IPv6 de una forma jerárquica. Este espacio se encontraría gestionado por la IANA. Esta organización posee autoridad sobre todo el espacio de direcciones IP, incluyendo IPv6. La IANA delega partes del espacio de direcciones IPv6 a *Regional Internet Registries* o "Entidades de Registro Internacionales" , normalmente abreviado como "RIRs", de acuerdo con sus necesidades. Los RIRs posteriormente delegan subrangos a otras organizaciones.

Según este modelo, en lo más alto de esta jerarquía, varias RIRs[2] asignarían bloques de direcciones a entidades denominadas TLAs. Estas TLAs serían esencialmente, aunque no necesariamente, "puntos de transito públicos" (*"exchanges"*) donde grandes proveedores y compañías de telecomunicación establecerían conexiones entre ellas. En la figura 1.14 se muestra gráficamente este esquema:

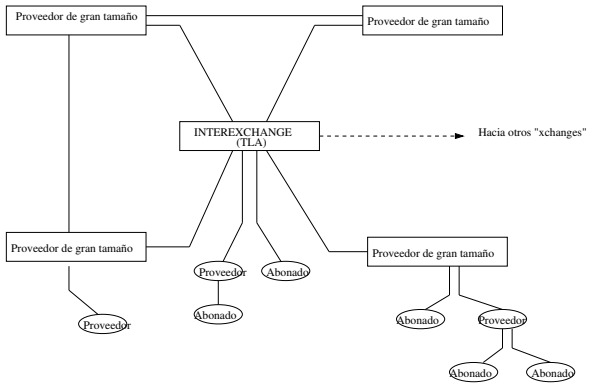

Figura 1.14: Estructuras de asignación basadas en agregación (*"exchanges"*).

Las TLAs asignarían bloques de direcciones a *"Next Level Aggregators"* o NLAs, que serían grandes proveedores y redes corporativas de ámbito global. Según este esquema, Cuando el NLA es un proveedor, puede asignar sus direcciones a sus distintos abonados. El encaminamiento de esta forma resulta eficiente debido a que las NLAs que se encuentran bajo el mismo TLA tendrán direcciones con un prefijo TLA común. De igual forma, los abonados pertenecientes al mismo proveedor tienen direcciones IP con un prefijo NLA común.

[2]Actualmente sólo existen tres entidades: RIPE NCC, ARIN y APNIC. ARIN gestina el continente americano y la mitad inferior de Africa. RIPE NCC gestiona Europa (occidental y oriental), la parte superior del continente africano y Groenlandia. APNIC gestiona Asia, Oceanía y Madagascar.

El esquema que acabamos de comentar se basa en [?], donde se explicita que las direcciones agregables se organizan formando una jerarquía topológica, consistente en una topología pública, una topología de sitio y el identificador de la interfaz, como se observa en el dibujo 1.15:

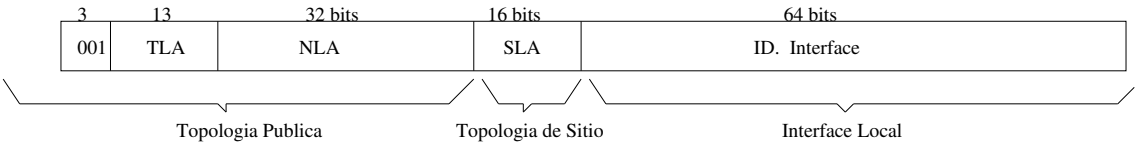

Figura 1.15: Direcciones IPv6 basadas en la agregación.

En [?] se describe una distribución ligeramente distinta a la anterior para la ubicación inicial de direcciones IPv6, que es la que se muestra a continuación en el dibujo 1.16:

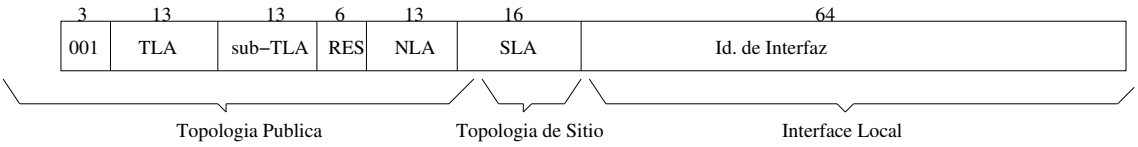

Figura 1.16: Direcciones IPv6 basadas en la agregación y *"slow start"*.

Esta modificación se realiza para que las RIRs puedan utilizar un mecanismo de delegación controlado y lento. Según este mecanismo, la delegación inicial permitirá utilizar a la entidad solicitante (denominada en este caso sub-TLA) hasta 13 bits de NLA. Este modificación respecto al esquema de direcciones inicial resulta beneficiosa por los siguiente motivos:

- Permite a las RIRs especificar criterios de entrada relativamente asequibles para las organizaciones que desean una delegación de sub-TLA. Todo el mundo recibe la misma cantidad, que puede considerarse como relativamente pequeña.
- Permite a las RIRs mantener contacto con entidades TLA según se van desarrollando, proporcionando ayuda y soporte para que las políticas y prácticas se implementen de una forma consistente. Sin este mecanismo, entidades TLA que recibieran grandes delegaciones iniciales podrían no tener un contacto formal con las RIRs durante varios años. De esta forma, se puede monitorizar a las sub-TLAs para conocer si se están aplicando correctamente determinadas políticas.

Retomando el esquema mostrado inicialmente, sin *"slow start"*, las entidades TLAs podrían dividir el campo "NLA" para crear su propia jerarquía de una forma adecuada a la actual organización de los ISPs, en donde ISPs pequeños se conectarían a ISPs más grandes. Esto se consigue realizando sucesivas subdivisiones del campo NLA como se observa en la figura 1.17 de la página 33.

A continuación del campo NLA se encuentraría el campo denominado *"site-level-aggregator"* o SLA, que junto con el identificador de interfaz serviría para describir la red del abonado o subscriptor del servicio. Normalmente, los ISPs delegan a sus abonados bloques de direcciones contiguas, que son usadas por organizaciones individuales para crear su propia jerarquía de

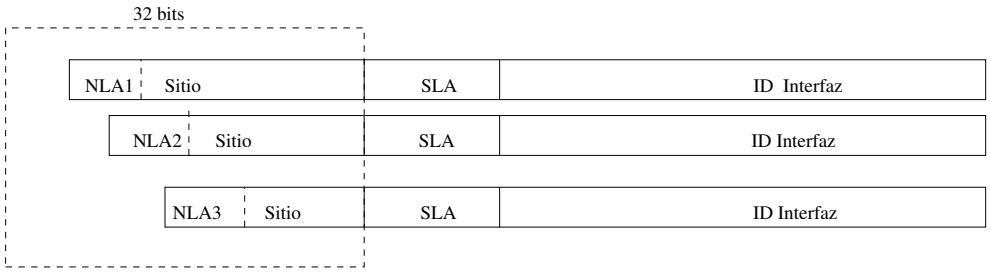

Figura 1.17: Subdivisión del campo NLA.

direccionamiento local. Los 16 bits del campo SLA permitirían definir hasta 65.535 subredes inviduales. El identificador de interfaz que se utiliza para identificar (valga la redundancia) una interfaz dentro de un enlace, se define a partir de la dirección MAC de dicha interfaz. Por consiguiente, según este esquema, un abonado recibiría un prefijo /48.

Hoy en dia, los routers del *"backbone"* de Internet tienen que mantener alrededor de 120.000 rutas (80.000 del DFZ y 40.000 propias)[**?**]. Con la jerarquía de direcciones que se ha explicado, todos los segmentos de red de un determinado abonado podrían ser alcanzados utilizando uno o más puntos de agregación de alto nivel. Esto permitiría a los routers del *"backbone"* resumir de forma eficiente las rutas a la red del cliente utilizando prefijos de TLAs.

Resumiendo, nos encontramos con un modelo de asignación de direcciones altamente jerárquico que trata de combinar las ventajas de ubicación por proveedor y las ventajas de ubicación geográfica. La ubicación por proveedores divide la jerarquía a lo largo de grandes proveedores de servicio, sin importar su ubicación espacial. Por otro lado, la ubicación geográfica divide o delega el árbol de direcciones estrictamente en base a la localización de proveedores/abonados de forma semejante a los códigos telefónicos de países y provincias. Ambas aproximaciones tienen sus deficiencias si se toman de una forma aislada, debido a que las grandes redes a menudo no se establecen estrictamente en los límites de un proveedor o de una localizacion geográfica determinada. Algunas redes, por ejemplo, pueden conectarse a varios ISPs, y otras redes pueden atravesar numerosos países y regiones del globo.

Por lo tanto, según este esquema, IPv6 se encontraría basado en la existencia de un número limitado de *"exchanges"*, donde grandes proveedores de servicio y compañías de telecomunicaciones se interconectarían. La utilización de puntos de intercambio para dividir el espacio de direcciones de IPv6 tiene una componente geográfica porque los *"exchanges"* se distribuyen por paìses alrededor del mundo. Pero también posee una orientación hacia el proveedor de servicio debido a que todos los grandes proveedores están conectados a uno (o más) de estos puntos de intercambio.

Para finalizar con este apartado, debe quedar claro que las direcciones agregables unicast son sólamente una parte del total del espacio de direcciones que ha sido definido en IPv6. Otros rangos de direcciones han sido asignados para multicast y para nodos que sólo requieren direccionamiento dentro de un área determinada (*site-local* y *link-local*) y, por supuesto, se dispone de mucho espacio todavía sin asignar.

La nueva política de distribución de direcciones para IPv6.

Todo lo que se va a contar en este apartado se puede encontrar en [**?**]. Dado que nos encontramos en un período evolutivo inicial del nuevo protocol IPv6, es posible que esta política sea suplantada o modificada a lo largo del tiempo, pero en el momento de escribir el presente proyecto fin de carrera, dicho reglamento es el que se encuentra actualmente en funcionamiento.

En la actualidad, la responsabilidad de la gestión del espacio de direcciones IPv6 se distribuye globalmente de acuerdo con la estructura jerárquica que se muestra en 1.18:

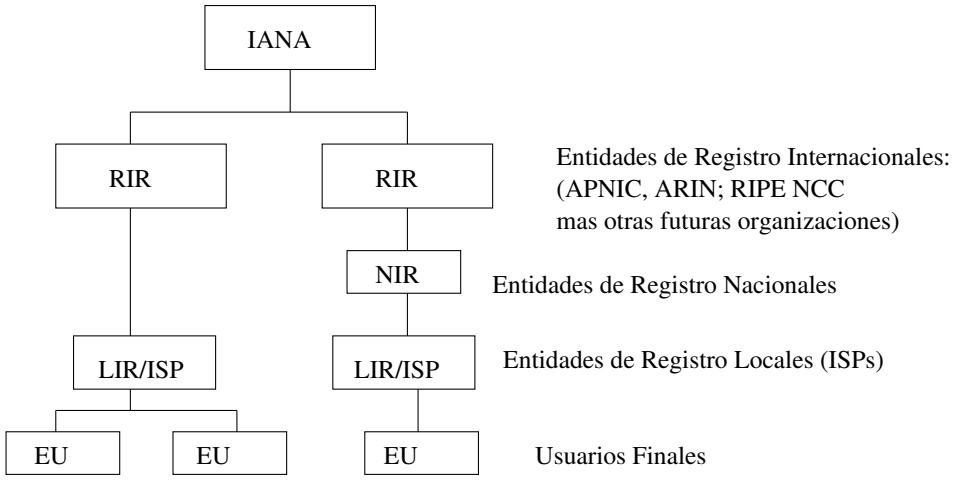

Figura 1.18: Gestión distribuida del espacio de direcciones IPv6.

Para alcanzar los objetivos que las entidades de registro se han propuesto, se deben seguir los siguientes principios básicos:

- El espacio de direcciones no se considera propietario: la política de distribución de direcciones se basa en el reconocimiento de que el espacio de direcciones se alquila, mediante una licencia de uso, renovándose cada cierto tiempo. Va en contra de la comunidad de Internet, vista como un todo, que el espacio de direcciones se considere propietario.

- Encaminamiento no garantizado: las entidades de registro no garantizan que todas las asignaciones o delegaciones de direcciones sean globalmente enrutables. No es su trabajo. No obstante, las entidades de registro internacionales o RIRs deben aplicar procedimientos enfocados a reducir la posibilidad de acabar teniendo un espacio de direcciones fragmentado.

- Delegación mínima: las RIRs aplicarán un tamaño mínimo para las delegaciones de direcciones IPv6, para facilitar un filtrado basado en el prefijo. Este tamaño se encuentra actualmente definido en /32. El prefijo /29 queda reservado para poder realizar agragaciones futuras. También resulta conveniente hacer notar que durante la fase de introducción inicial de IPv6 se especificada realizar agragaciones de tamaño /35.

- Consideración de la infraestructura IPv4: cuando un proveedor de servicio IPv4 solicite espacio de direccionamiento IPv6 para una eventual transición de servicios al nuevo protocolo, el número de actuales clientes IPv4 puede ser utilizado para justificar una delegación más grande de la que podría justificarse basándose sólamente en la infraestructura IPv6 que se pretende utilizar.

Respecto a la organización interna de las direcciones IPv6, la antigua compartimentalización del espacio se considera obsoleta ya que mezcla límites tecnológicos con límites o imposiciones relacionadas con la gestión del espacio de direcciones. Por tecnología se entiende en este caso límites que vienen impuestos por la definición o estructura del protocolo IPv6, por ejemplo los últimos 64 bits de la dirección IPv6, que se encuentran definidos por la especificación del propio protocolo como el identificador de la máquina. De esta forma, se pasa a tener el siguiente formato para las direcciones IPv6 globales unicast, donde existen por un lado recomendaciones y por otro imposiciones tecnológicas, como se puede observar en la figura 1.19:

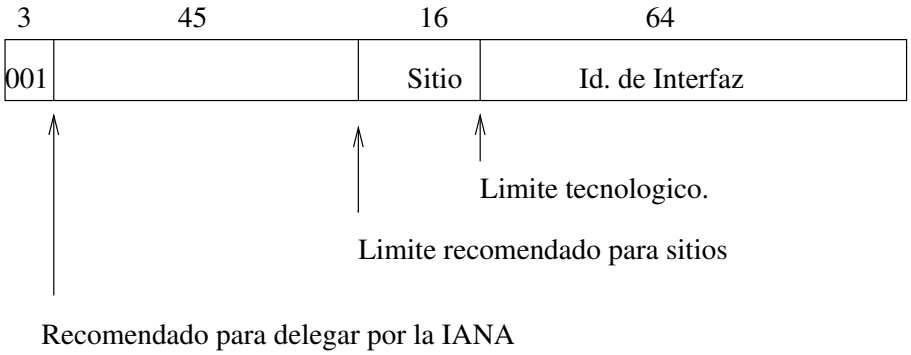

Figura 1.19: Nueva dirección IPv6 unicast.

De acuerdo con la nueva política de distribución de direcciones, se especifica que las entidades de registro locales o LIRs deben realizar asignaciones de direcciones IPv6 de acuerdo con las siguientes reglas:

- /48 en el caso general, salvo para abonados muy grandes.
- /64 cuando se sepa que sólo se necesita una única subred.
- /128 cuando se sepa con certeza que sólo se va a conectar un dispositivo, este caso se contempla en el reglamento pero no se prevé su utilización a gran escala.

Las entidades de registro internacionales o RIRs no se ven afectadas por el tamaño de las direcciones que asignan las LIRs.

Las RIRs sólo delegan (actualmente) subrangos de direcciones IPv6 cuando se cumple la siguiente condición:

"Se mantiene comunicación con 3 o más redes IPv6 (3 o más sistemas autónomos) "Y" existe un plan para proporcionar servicios IPv6 en un plazo no superior a 12 meses "Y" se posee 40 o más clientes. "O", por otro lado, cuando se quiere realizar pruebas dentro de la red IPv6 de pruebas denominada 6bone."

Como ya se ha comentado con anterioridad, las RIRs están delegando prefijos de tamaño /35, reservando el prefijo /29 para futuras agregaciones. En Abril de 2002, APNIC había delegado 49 prefijos, 24 ARIN y 51 RIPE NCC[3]

Resumiendo, esta forma de realizar la asignación resulta más flexible, sencilla y con menos restricciones sobre el formato de las direcciones. En el momento de escribir este trabajo el documento [**?**] acaba de entrar en vigor.

1.2.4 Mecanismos de autoconfiguración.

La configuración de direcciones IP es una constante en la vida de los administradores de redes corporativas. Las direcciones IP necesitan ser configuradas para máquinas nuevas y para máquinas conectadas a redes físicas que reciben nuevos subrangos de direcciones (por ejemplo un cambio en el prefijo de red debido a un cambio de proveedor o de localización geográfica de la empresa). De esta forma, la autoconfiguración de direcciones se posiciona como un componente imprescindible en el nuevo protocolo IP, con el objetivo de facilitar la administración y la renumeración automática de grandes poblaciones de máquinas.

Existen tres mecanismos de autoconfiguración en IPv6 claramente diferenciados:

- Sin estado:
  - Utilizando sólamente una dirección *link-local*.
  - Utilizando *"router advertisements"* para adquirir un prefijo de red.
- Con estado:
  - DHCPv6.

Utilizando mensajes de *"router advertisement"*, el router puede controlar si las máquinas utilizarán autoconfiguración con estado o sin estado. En el caso de la autoconfiguración con estado ("stateful configuration"), la máquina contactará con un servidor de direcciones DHCP o similar, el cual asignará una dirección de un conjuto de direcciones administradas manualmente. Utilizando autoconfiguración sin estado ("stateless autonconfiguration"), una máquina puede automáticamente configurar su propia dirección IPv6, utilizando direcciones *link-local* en ausencia de routers en la subred, lo que se conoce con el mecanismo *"plug and play"* característico de IPv6, o bien puede configurarse con un prefijo de mayor alcance gracias al envío de *"router advertisements"* por parte de uno o varios routers IPv6 establecidos en la subred en la que se encuentre dicha máquina.

A modo de ejemplo, se va a comentar brevemente el procedimiento típico de autoconfiguración de una máquina utilizando el mecanismo sin estado, cuando existen routers IPv6 en la subred. Para una información más detallada sobre el proceso de autoconfiguración en IPv6 y sobre los protocolos involucrados, se remite al lector a [**?**].

El proceso de autoconfiguración en IPv6 comienza con el protocolo *"neigbor discovery"*. Este procolo combina el servico proporcionado por ARP e ICMP en IPv4. En un escenario típico, una máquina comienza el proceso de autoconfiguración utilizando de manera temporal una dirección *link-local*. Esta dirección se construye añadiendo al prefijo *link-local* un *"token"*

[3]datos extraidos de http://www.ripe.net/cgi-bin/ipv6allocs.

generado a partir de la dirección MAC de la interfaz. Una vez que se ha construido esta dirección, la máquina envía un mensaje de *"neigbour discovery"* hacia dicha dirección, para asegurarse de que es única dentro de ese enlace. Si no se recibe ningun mensaje ICMP, la dirección es única, en caso contrario, se genera otro identificador de interfaz utilizando por ejemplo números aleatorios.

Una vez que la máquina dispone de una dirección *link-local* autogenerada con la seguridad de que resulta ser única en el enlace, dicha máquina envía un paquete de *"router solicitation"* utilizando como dirección origen dicha dirección *link-local*. Este paquete se envía a una dirección multicast bien conocida (IPv6 define actualmente un conjunto permanente de grupos multicast para encontrar determinados recursos en el enlace, incluyendo un grupo *"all-routers"*, un grupo *"all-hosts"* y un grupo para servidores de DHCP, como se ha comentado en anteriores apartados). Los routers responden al mensaje con un mensaje unicast de *"router advertisement"*, que contiene, entre otras cosas, el prefijo de la subred que la máquina debe utilizar. Los routers también envían estos anuncios periódicamente a la dirección multicast *"all-nodes"*, independientemente de su petición por parte de la máquina.

Una vez que la máquina ha recibido el paquete de *"router advertisements"*, puede autoconfigurarse con una dirección IPv6 global además de la dirección *link-local* anteriormente adquirida.

Resulta importante resaltar que IPv6 soporta además múltiples direcciones por interfaz, y que las direcciones IPv6 asignadas o autoconfiguradas en una interfaz se clasifican en "válidas", "caducadas" o "no-válidas"[4] dependiendo del estado en que se encuentren. Por ejemplo, durante un proceso típico de renumeración de subredes, una prefijo determinado caducará y un nuevo prefijo IPv6 será automáticamente asignado. Durante un determinado período de tiempo, las direcciones así caducadas que se encuentren en uso podrán seguir utilizandose, pero no se iniciará ninguna nueva conexión con direcciones o prefijos caducados. Esto permite que las conexiones en curso finalicen de una forma no-disruptiva, permitiendo que la renumeración de la red ocurra de una forma dinámica y transparente para el usuario final. Señalar que también existen otros estados dentro de una dirección IPv6, como el estado "tentativo" que se produce cuando se configura una dirección IPv6 en la interfaz y se mantiene mientras el proceso de descubrimiento de duplicados no ha finalizado.

Para finalizar esta apartado, comentar que el formato de los paquetes involucrados en el proceso de autoconfiguración de direcciones así como los detalles del mecanismo de descubrimiento de duplicados, también denominado DAD por sus siglas inglesas, resulta demasiado extenso para ser comentado en el presente proyecto fin de carrera. El lector debe hacerse una idea de la existencia de tales mecanismos, pero para una descripción más detallada se debe consultar [**?**].

1.2.5 Cambios en el DNS para soportar IPv6.

En [**?**] se definen los cambios necesarios para que el sistema de nombres de dominio, más conocido como DNS, soporte máquinas IPv6. Los cambios incluyen un nuevo tipo de *"resource record"* para almacenar una dirección IPv6 y actualizaciones en los tipos de consulta. Estas extensiones se han definido de tal forma que resultan compatibles con las aplicaciones cliente y servidor existentes en la actualidad. A continuación se va a comentar muy brevemente los

[4] En inglés, respectivamente, *"valid"*, *"deprecated"* y *"invalid"*.

nuevos registros que IPv6 introduce en el DNS, su formato de representación y los problemas y decisiones que se han tomado en este ámbito.

Nuevos *"resource records"*. AAAA y A6.

En [?] y [?] se definen varios tipos de registros relacionados con IPv6. De ellos, los más importantes son los registros AAAA y los registro A6 que realizan el mapeo de un nombre a una dirección IPv6.

Los registros de tipo AAAA se utilizan para almacenar direcciones IPv6. Una máquina con más de una dirección IPv6 podrá tener más de un registro de tipo AAAA. La forma de introducir estos registros en la base de datos del DNS se puede observar en el siguiente ejemplo:

```
mira.ipv6.it.uc3m.es IN AAAA 2001:720:410:ffff::1
```

En [?] se define además un nuevo tipo de registro denominado A6. A continuación se muestra el siguiente ejemplo para ilustrar las capacidades de los registros A6:

Supongamos que nuestro ISP mantiene el dominio "ispx.tld", y gestiona el prefijo 1234:5600::/24. Por otro lado, nuestro sitio, denominado "sitex.tld", obtiene el prefijo 1234:5678:9abc::/48 de dicho proveedor. A continuación se muestra la parte relevante de la zona del ISP utilizando registros A6:

```
$ORIGIN ispx.tld
ipv6    IN A6   0 1234:5600::
sitex   IN A6   24 0:0:78:9abc:: ipv6
```

Y esta es la parte correspondiente a nuestro sitio:

```
$ORIGIN sitex.tld
ipv6            IN A6   48 0::0 sitex.ispx.tld.
subnet1 IN A6   48 0:0:0:1:: ipv6               ;
host1           IN A6   64 ::2a0:80ff:fed5:55a9 subnet1
```

Mediante este ejemplo se observan el formato y las características principales de este nuevo tipo de registro. El formato obedece al siguiente esquema:

```
<domain>        IN A6   <prefixlen> <suffix> <prefix-name>
```

Donde <prefixlen> es la longitud del prefijo especificado en la parte <prefix-name>. El campo <suffix> se añade por el final a la dirección IPv6 especificada en <prefix-name> para formar la dirección IPv6 del dominio en cuestión.

Siguiendo el ejemplo anterior, se asignarían las siguientes direcciones:

```
ipv6.ispx.tld =1234:5600::
sitex.ispx.tld = 1234:5678:9abc::
subnet1.sitex.tld = 1234:5678:9abc:1::
host1.sitex.tld = 1234:5678:9abc:1:2a0:80ff:fed5:55a9
```

Nótese que el registro A6, cuando se representa con un <prefixlen> de cero, resulta ser semánticamente equivalente al registro AAAA (con un octeto de más desperdiciado en su codificación), de tal forma que cualquier cosa que puede hacerse con los registros AAAA puede realizarse también con los registros A6.

Árbol de resolución inverso.

Se define además el dominio IP6.ARPA para buscar un nombre dada una dirección IPv6. Inicialmente se definió una rama para el árbol de resolución inverso bajo IP6.INT, pero en la actualidad dicho rama ha sido deprecada, aunque puede que permanezca vigente durante algunos años por motivos de compatibilidad con antiguas implementaciones de servidores de nombre que consultan directamente contra dicha rama del árbol.

Una dirección IPv6 se representa como un nombre en el dominio IP6.ARPA utilizando una secuencia de *"bitstrings"* separados por puntos y añadiendo el sufijo ".IP6.ARPA" al final de la cadena. La secuencia de *"bitstrings"* se codifica en orden natural, es decir, tal como se escribe normalmente la dirección, a diferencia del formato *"nibble"* donde los dígitos se escriben en orden inverso. La utilización de *"bitstrings"* posee la ventaja de que el límite de terminación se define por bits, no por dígitos hexadecimales. Por ejemplo, el nombre de dominio inverso para la siguiente dirección:

```
2345:00c1:ca11:0001:1234:5678:9abc:def0
```

utilizando el sistema de representación *"bitstrings"* es el siguiente:

```
\[x234500c1cA110001123456789abcdef0/128].IP6.ARPA.
```

Por otro lado, el formato *"nibble"* se utiliza para la rama del árbol IP6.INT. Cada *"nibble"* se mediante un dígito hexadecimal. Por ejemplo, el nombre de dominio inverso correspondiente a la dirección:

```
4321:0:1:2:3:4:567:89ab
```

Es el que se va a mostrar a continuación. Por supuesto, escribir tales cadenas resulta tedioso, por lo que se necesitan herramientas que hagan la conversión y la consulta de una forma sencilla para un ser humano:

```
b.a.9.8.7.6.5.0.4.0.0.0.3.0.0.0.2.0.0.0.1.0.0.0.0.0.0.0.1.2.3.4.IP6.INT.
```

Consideraciones finales.

Con la publicación de [**?**] y [**?**], el IETF ha estandarizado dos formas diferentes de almacenar direcciones IPv6 dentro del DNS. Este hecho ha provocado confusión sobre cuál de ambas conviene implantar e implementar.

La RFC 2874[**?**] define el registro A6, que consiste en el almacenamiento de la dirección IPv6 como una cadena de registros, en vez de en un único registro. Los registros A6 fueron

diseñados para simplificar la gestión del DNS ante eventos de renumeración de sitios. Además, dicha RFC define un registro inverso, denominado DNAME, que podría ser utilizado para gestionar el árbol inverso.

Las discusiones alrededor de la utilización de los registros A6, *"bit-string"* y registros DNAME han sido largas y opuestas. Mientras que unos consideran que las extensiones al DNS constituyen una herramienta útil para gestionar la renumeración de sitios, otros sienten que los beneficios de estas extensiones no han sido suficientemente demostrados, y que su complejidad y peligros potenciales no justifican un uso amplio de las mismas. Se ha alcanzado consenso dentro de la comunidad de Internet, en el sentido de que dichas extensiones no deberían ser ampliamente implantadas durante la primera fase de introducción de IPv6, y que IPv6 debe continuar utilizando la misma lógica para el DNS que la que se utiliza para el protocolo IPv4.

Respecto a los peligros asociados a los registros A6, se puede encontrar un ejemplo en [**?**], aunque el problema allí mencionado se puede minimizar si se sigue la regla *"bounded work per query"*[5]. También se recomienda la lectura de [**?**].

Recientemente, se ha procedido a reclasificar la RFC 2673[**?**] y la RFC 2874[**?**] como experimentales. Con esta acción, el IETF está recomendando que el mecanismo de DNS para dar soporte a IPv6 permanezca esencialmente igual que el que ya se encuentra definido para IPv4.

Por último, el IESG ha aprobado los documentos [**?**] y [**?**], de carácter informacional, que ponen de manifiesto las acciones llevadas a cabo como consecuencia de dichas discusiones y los problemas encontrados en las citadas extensiones al DNS, respectivamente. Para el lector ávido de conocimientos, se recomienda encarecidamente su lectura.

[5]Limitar la cantidad de trabajo (número de paquetes, procesos lanzados en paralelo, etc) de tal forma que una consulta no pueda entrar en un bucle infinito o provocar una reacción en cadena de consultas incluso si alguien ha configurado alguna zona de forma incorrecta. Definida en [**?**], página 35.

Capítulo 2

Descripción teórica de los mecanismos de transición.

En este capítulo se describen algunos de los mecanismos de transición más importantes que el IETF ha definido en el momento de realizar este trabajo. Algunos de estos mecanismos han sido probados y la configuración junto con los resultados obtenidos se presentan en un capítulo posterior. La referencia obligada para obtener información sobre cualesquiera de ellos es el grupo de trabajo *"ngtrans"* del IETF[1]

Después de explicar por qué se necesitan mecanismos de transición, se examinan los distintos mecanismos clasificados en dos grandes grupos. Por un lado los mecanismos que se utilizan para interconectar islas IPv6 y por otro lado los mecanismos que permiten establecer comunicación entre IPv6 e IPv4.

2.1 Necesidad de utilizar mecanismos de transición.

Un de los aspectos más importantes durante el desarrollo de IPv6 ha sido cómo hacer la transición desde IPv4 hacia IPv6. Se espera que IPv4 e IPv6 coexistan durante varios años durante esta etapa de transición. Dicha migración se está basando en parte en la utilización de la red IPv4 como un enlace virtual punto a punto que permite, gracias a la encapsulación de IPv6 sobre IPv4 (*"IPv6-over-IPv4 encapsulation"*), comunicar redes IPv6 que de otro modo estarían completamente aisladas. Estos túneles suponen un incremento de la complejidad administrativa tanto para las organizaciones finales, como para las redes que proporcionan el servicio de encapsulación, fundamentalmente en la creación, gestión y mantenimiento de dichos túneles.

Cuando se intenta introducir IPv6 en Internet, a primera vista se pueden diferenciar dos grandes conjuntos de problemas. El primero está relacionado con el hecho de tener comunicación IPv6 entre dos o más "islas" IPv6 dentro de una Internet que es IPv4. El segundo conjunto de problemas está relacionado con el establecimiento de algún tipo de comunicación entre el mundo IPv4 existente y el nuevo mundo IPv6.

En el primer conjunto de problemas, las soluciones se basan generalmente en routers *"dual stack"* y en túneles "IPv6 en IPv4". Por otra lado, los mecanismos necesarios para resolver el

[1]http://www.ietf.org/html.charters/ngtrans-charter.html. Ver también http://www.6bone.net

segundo conjunto de problemas hacen uso de técnicas de *"dual stack"*, traductores a nivel de aplicación, tecnología de NAT o ubicación temporal de direcciones IPv4 y de encapsulación "IPv4 en IPv6".

Para solventar estos problemas, el grupo de trabajo *"ngtrans"* del IETF continúa desarrollando nuevos mecanismos de transición, y continuará ofreciendo nuevas formas de realizar dicha transición hasta que quede absolutamente claro que la migración hacia el nuevo protocolo se pueda realizar en todas las organizaciones y empresas de una forma satisfactoria y sencilla. La migración al nuevo protocolo debe resultar transparente para el usuario final. Por consiguiente, el grupo de trabajo *"ngtrans"* se ha fijado los siguientes objetivos.

- Especificar las herramientas y mecanismos que pueden ser utilizados para transitar a IPv6.
- Escribir documentos que muestren cómo las distintas herramientas y mecanismos pueden aplicarse a distintos escenarios de red.
- Coordinarse con la red de pruebas del *6bone*, que se encuentra operando bajo las direcciones IPv6 reservadas para pruebas, para aplicar los mecanismos de transición al desarrollo, a la experimentación y a la implantación de IPv6 por todo el mundo.

Básicamente, existen los siguientes mecanismos principales para realizar la transición de IPv4 a IPv6 y la mayoría de los mecanismos de transición propuestos hacen uso de alguno de estos mecanismos básicos:

- Máquinas *"dual stack"*: proporcionan soporte completo para IPv4 e IPv6 en routers y máquinas. la denominación *"dual stack"* es por sí misma incorrecta. La mayoría de las implementaciones de IPv6 existentes no ofrecen dos pilas de protocolos TCP/IP completamente separadas, una para IPv4 y otra para IPv6, sino un stack híbrido donde el código se comparte entre las dos pilas de protocolos.
- Encapsulación de IPv6 sobre IPv4: consiste en encapsular paquetes IPv6 dentro de paquetes IPv4 para transportarlos sobre la infraestructura de encaminamiento que proporciona IPv4. Se pueden utilizar túneles para simular una red virtual IPv6 sobre la actual red IPv4, como se realiza actualmente dentro de la red de pruebas del *6bone*.
- Traducción a nivel IP: consiste en traducir el formato del paquete IPv6 a IPv4 y viceversa. Nótese que algunas de las características del nuevo protocolo IP no pueden ser traducidas a IPv4, como por ejemplo algunas cabeceras de extensión.

Como la mayoria de las técnicas que se describen a continuación se basan en la utilización de direcciones IPv4 en ambos extremos del túnel, muchas de estas técnicas no pueden funcionar si se utiliza NAT entre dichos extremos. También, si se utilizan *"firewalls"* se debe habilitar el paso de paquetes que hagan uso del protocolo 41 (*"encapsulated packets"*).

2.2 Herramientas a considerar. Criterios para generar soluciones.

En este apartado se definen algunos criterios genéricos a la hora de comparar las distintas soluciones propuestas[**?**]. También pueden servir de "horizonte mental" a la hora de diseñar un nuevo mecanismo de transición.

Ambito de aplicabilidad: el ámbito de aplicabilidad puede ser de máquina, de dominio o global.

La comprensión del ámbito de los distintos mecanismos es importante cuando uno está tratando de compararlos o de combinarlos juntos. Una solución con un ámbito de dominio generalmente permite a un dominio en su totalidad conectarse, pero no debería tener ningún efecto en el resto de Internet. Una solución con ámbito de máquina permite la conexión de una única máquina. Siguiendo este esquema, dos mecanismos con el mismo ámbito de aplicabilidad no se impactarán el uno al otro cuando se apliquen en lugares diferentes. Por ejemplo, dos sitios distintos, uno implantando "6over4" y otro utilizando DSTM, no deberían experimentar ningún problema de interoperabilidad entre ellos. No obstante, puede no ser fácil ni estar recomendado el uso de ambos mecanismos dentro de un mismo ámbito.

Requisitos de IPv4: qué se necesita en el contexto de IPv4 para que la herramienta propuesta funcione.

Algunos mecanismos pueden obligar a la utilización de multicast sobre IPv4, otros pueden hacer uso de algoritmos de encaminamiento específicos, o incluso modificaciones en los routers, uso de paquetes activos, etc. Especificar correctamente los requisitos exigidos al protocolo IPv4 ayuda a definir la aplicabilidad y la complejidad del mecanismo de transición.

Requisitos de las direcciones IPv4: cuántas direcciones IPv4 se necesitan para implementar la solución propuesta.

Algunos mecanismos pueden requerir el uso de direcciones IPv4 públicas, de más de una dirección, o pueden no funcionar a través de *"firewalls"*. Debe, por consiguiente, analizarse concienzudamente las necesidades de direccionamiento IPv4 que nuestro mecanismo pueda necesitar para su correcto funcionamiento.

Requisitos de IPv6: qué se necesita en el contexto de IPv6 para que la herramienta propuesta funcione.

De forma semejante al apartado referente a IPv4, los mecanismos de transición deben definir qué aspectos de IPv6 necesitan para su correcto funcionamiento, por ejemplo, algún mecanismo de transición podría requerir el uso de IPSec.

Requisitos de las direcciones IPv6: cuántas direcciones IPv6 se necesitan para implementar la solución propuesta.

De forma semejante a IPv4, algunos mecanismos hacen uso de tipos específicos de direcciones IPv6 y de determinados grupos multicast. Es imprescindible definir las direcciones IPv6 que necesita el mecanismo para su correcto funcionamiento.

Requisitos de máquinas: qué se necesita para que las máquinas puedan participar en la solución.

En este apartado se deben especificar los componentes que una determinada máquina necesita para poder participar en el mecanismo de transición propuesto. Algunos mecanismos necesitan aplicar modificaciones a la implementación actual del protocolo IPv6, módulos extra entre la capa de red y la capa de transporte, dispositivos específicos, demonios, NATs, etc.

Requisitos de routers: qué se necesita para que los routers permitan el correcto funcionamiento de la solución propuesta.

Al igual que las máquinas, resulta necesario especificar el impacto del mecanismo de transición en los routers. Un mecanismo de transición puede requerir un tratamiento específico para los paquetes que atraviesan los routers, algunos otros mecanismos pueden necesitar modificaciones en la implementación de los routers, pueden utilizar paquetes activos que deben ser procesados en los mismos, etc. Todos estos requisitos deben conocerse y definirse adecuadamente.

Impacto del NAT: se trata de descubrir si la solución propuesta funcionará si se implanta detrás de un NAT.

Dada la extensión de NATs en la Internet actual, debe considerarse si el mecanismo de transición puede funcionar a través de NATs, y en caso negativo, si existe alguna otra alternativa para solventar el problema del NAT. Los NAT se presentan como una solución en el mundo IPv4, pero constituyen un problema para el desarrollo de IPv6 y de mecanismos de transición IPv4/IPv6.

Otros requisitos: cualquier otra restricción que el nuevo mecanismo necesite para su correcto funcionamiento.

Esta enumeración no trata de ser exhaustiva, sino sólamente de hacer notar las características más importantes que deben considerarse a la hora de desarrollar o de implantar un determinado mecanismo de transición. Por supuesto, cada mecanismo de transición puede tener sus propios requisitos específicos.

Resumiendo, al igual que ocurre con la ingeniería del software, un buen documento de requisitos fija y establece las pautas de lo que podemos conseguir, qué cosas pueden hacerse y qué cosas no pueden (y a qué coste). El "cómo" se hacen, depende del diseño del mecanismo y de su implementación posterior.

A continuación se comentan a grandes rasgos los distintos mecanismos de transición agrupados en función de los problemas que tratan de resolver. Para cada mecanismo, se detallan sus requisitos de acuerdo con los puntos que se han enumerado con anterioridad.

Dividimos los mecanismos de transición en dos grandes grupos:

- Mecanismos de interconexión de islas IPv6.
- Mecanismos de comunicación entre IPv4 e IPv6.

A continuación se describe de forma teórica cada uno de los mecanismos.

2.3 Mecanismos de interconexión de islas IPv6

Los mecanismos descritos a continuación están diseñados para permitir la comunicación entre varias islas IPv6 dentro de un mundo IPv4. Todos ellos hacen uso del mecanismo de encapsulación de paquetes.

2.3.1 Túneles configurados.

En la mayoría de los casos, la infraestructura de IPv6 se construirá utilizando la actual infraestructura de IPv4. Los túneles manualmente configurados pueden ser utilizados para conectar máquinas IPv6 o redes IPv6 a través de dicha infraestructura. Típicamente, los túneles configurados se utilizan entre sitios donde el tráfico será intercambiado regularmente, por lo que se suelen utilizar en conjunción con mecanismos de encaminamiento para IPv6, como por ejemplo BGP4+. Este mecanismo se encuentra descrito en [**?**].

Introducción.

En este escenario, los extremos del túnel se determinan mediante información de configuración obtenida manualmente. Cuando un paquete IPv6 se transmite utilizando este tipo de túneles, la dirección IPv4 del extremo del túnel se extrae a partir de la información suministrada mediante la configuración manual del interfaz del túnel. Existen protocolos y otros mecanismos que permiten una configuración automática de túneles haciendo uso de arquitecturas cliente-servidor [**?**].

Normalmente, Las técnicas encapsulación se clasifican atendiendo al mecanismo por el cual el nodo encapsulador determina la dirección del otro extremo del túnel. Cuando el extremo del túnel es un router, dicho router debe desencapsular el paquete y reenviarlo a su destino final. De esta forma, el extremo del túnel puede no coincidir con el destino del paquete que se está encapsulando. Por consiguiente, las direcciones IPv6 utilizadas en el paquete no pueden proporcionar ninguna información sobre la direccion IPv4 del extremo del túnel, y la configuración manual resulta necesaria.

Por otro lado, cuando el extremo del túnel resulta ser un host, la dirección destino del paquete IPv6 y la dirección destino del paquete IPv4 que se construye identifican a la misma máquina. Este hecho se puede aprovechar para codificar información en la dirección IPv6 que puede ser utilizada para determinar el extremo del túnel IPv4 de una forma automática, como se comentará en sucesivos apartados.

Funcionamiento.

En la figura 2.1 se observar el proceso de encapsulación y desencapsulación de paquetes IPv6 en paquete IPv4:

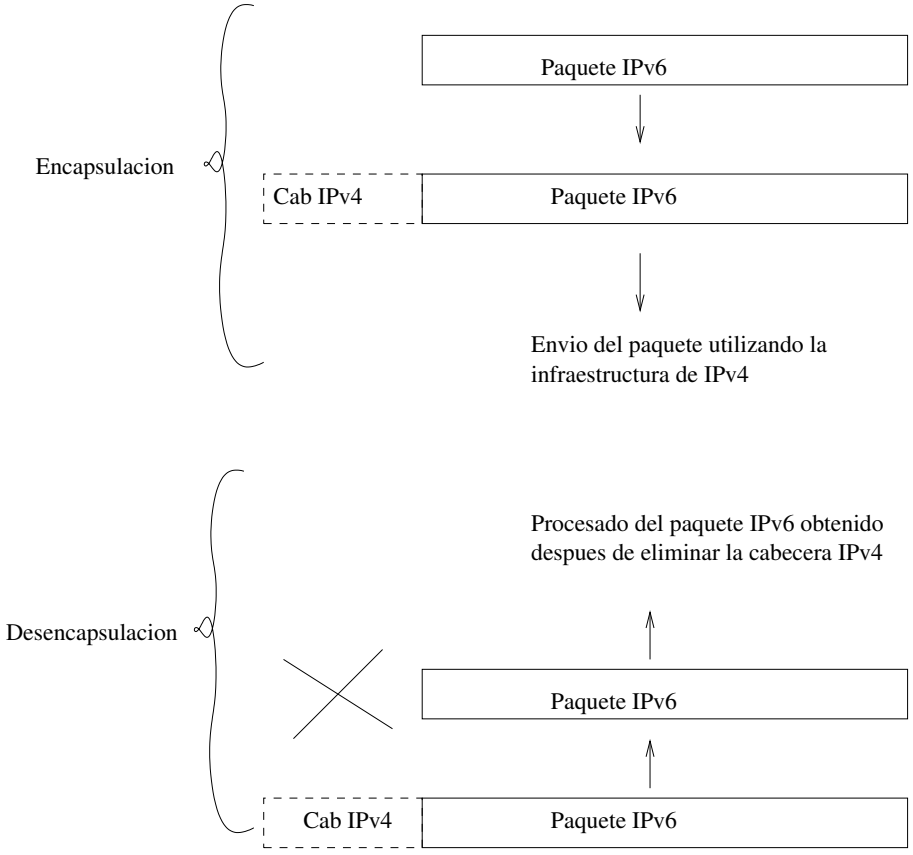

Figura 2.1: Estructura de los mecanismos de túneles.

Esta estructura se aplica a cualquier mecanismo de encapsulación de paquetes IPv6 en paquete IPv4, no sólamente a los túneles configurados. El funcionamiento del mecanismo de túneles configurados es el siguiente:

- El nodo encapsulador crea una cabecera IPv4 y transmite el paquete encapsulado.
- El nodo desencapsulador recibe el paquete encapsulado, reensambla el paquete en caso de ser necesario, elimina la cabecera IPv4 y procesa el paquete IPv6 así obtenido.
- El nodo encapsulador puede mantener información de estado para cada túnel, almacenando información sobre la MTU, por ejemplo.

Clasificación del mecanismo.

- Ambito de aplicación: global.
- Requisitos de IPv4: conectividad IPv4 entre los sitios.
- Requisitos de las direcciones IPv4: >= 1 por sitio.
- Requisitos de IPv6: ninguno.
- Requisitos de las direcciones IPv6: los nodos finales necesitan una dirección IPv6.

- Requisitos de máquina: pila IPv6 o doble pila.
- Requisitos de routers: doble pila.
- Impacto del NAT: no funcionará si el túnel tiene que atravesar un NAT puesto que ambos extremos del túnel deben ser direcciones globalmente enrutables. Puede funcionar si los extremos del túnel se encuentran donde el NAT.
- Otros requisitos: no.

2.3.2 Túneles automáticos.

Este mecanismo se encuentra descrito en [**?**].

Los túneles automáticos se utilizan como los túneles configurados para conectar máquinas o redes IPv6 aisladas unas de otras. Los túneles automáticos se crean cuando se necesitan y se rompen cuando no son utilizados. Típicamente, los túneles automáticos son utilizados entre máquinas individuales o entre redes donde sólo de forma incidental existe una necesidad de intercambio de tráfico. Un prerequisito para la utilización de túneles automáticos es la existencia de direcciones "IPv6 compatibles con IPv4" para las máquinas IPv6 que necesitan establecer la comunicación. Estas direcciones permiten que dichas máquinas puedan obtener las direcciones IPv4 de los extremos del túnel a partir de las direcciones IPv6 de una forma automática.

Introducción.

En el mecanismo de túneles automáticos, las direcciones de los extremos del túnel se determinan utilizando la dirección destino "IPv6 compatible con IPv4" del paquete IPv6 que se está encapsulando. Más concretamente, la dirección "IPv6 compatible con IPv4" posee en su interior la dirección IPv4 de destino, dirección que al mismo tiempo se utiliza para realizar una búsqueda en la tabla de rutas y obtener el interfaz de salida, a partir del cual se construye también la dirección IPv4 fuente del paquete encapsulado.

Por consiguiente, un túnel automático permite que máquinas IPv6/IPv4 puedan comunicarse sobre la infraestructura IPv4 sin configurar manualmente ningún tipo de túneles.

Los nodos comunicantes poseen una dirección "IPv6 compatible con IPv4". Esta dirección se crea automáticamente en el interfaz túnel para cada una de las direcciones IPv4 que pueda poseer el nodo en cuestión. Dicha dirección se identifica mediante un prefijo de 96 bits (todos a cero), y con la dirección IPv4 en los 32 bits más bajos, como se observa gráficamente en el esquema 2.2:

Figura 2.2: Formato de una dirección "IPv6 compatible con IPv4".

Estas direcciones se asignan exclusivamente a nodos que poseen soporte para realizar el mecanismo de túneles automáticos. Una dirección "IPv6 compatible con IPv4" es única de forma global siempre y cuando la dirección IPv4 con la que está construida lo sea.

Funcionamiento.

En el mecanismo de túneles automáticos, la dirección del extremo del túnel se determina a partir del paquete que está siendo encapsulado. Si la dirección IPv6 de destino es compatible con IPv4, entonces el paquete puede ser enviado utilizando este mecanismo. Si la dirección IPv6 de destino es una dirección nativa, el paquete no podrá ser enviado via túnel automático.

Una entrada en la tabla de rutas puede utilizarse para dirigir el mecanismo. Una implementación puede , por ejemplo, tener una ruta estática especial para el prefijo 0:0:0:0:0:0::/96. Los paquetes que "casen" con dicha entrada serán enviados a una pseudo-interfaz encargada de realizar la encapsulación.

Una vez que se encamina adecuadamente, el paquete IPv6 se encapsula utilizando las siguientes direcciones:

- dirección IPv4 destino: los 32 bits más bajos de la dirección IPv6 de destino.
- dirección IPv4 origen: la dirección IPv4 del interfaz por el que se envía el paquete.

Este mecanismo no debe enviar a destinos IPv4 broadcast o multicast. Debe, por consiguiente, tirar todos los paquetes IPv6 que posean en su interior direcciones IPv4 de dicho tipo, incluida la dirección "sin especificar" (0.0.0.0), o la dirección IPv4 de *"loopback"* (127.0.0.1).

Utilización junto con túneles configurados.

Los túneles automáticos se utilizan a menudo en conjunción con la técnica de los túneles configurados. De esta forma, máquinas aisladas IPv4/IPv6[2] se pueden configurar para utilizar túneles automáticos y direcciones "IPv6 compatibles con IPv4", además de poseer al menos un túnel configurado por defecto hacia un router IPv6. Dicho router se puede configurar para realizar también túneles automáticos. Según este esquema, las máquinas aisladas pueden enviar paquetes a direcciones IPv4-compatibles utilizando el túnel automático y también pueden enviar paquetes a destinos IPv6 nativos utilizando el túnel configurado.

Selección de direcciones adecuada.

Cuando una aplicación genera un paquete IPv6, ésta puede fijar la dirección origen de los paquetes. Pero cuando la aplicación no especifica ninguna dirección fuente, es la máquina quien debe seleccionar una dirección entre todas las posibles. Los nodos IPv6/IPv4 que se configuran para realizar túneles automáticos se pueden configurar con una dirección IPv6 nativa además de poseer una dirección IPv6 compatible con IPv4. La elección de qué dirección debe escogerse determinará la forma en que recibiremos los paquetes de respuesta. Si se utiliza una dirección IPv4-compatible, el tráfico de vuelta nos será entregado utilizando un túnel automático, pero si la dirección origen es una dirección IPv6 nativa, el tráfico de retorno no se nos entregará utilizando dicho mecanismo.

De esta forma se observa que la selección adecuada de la dirección fuente para cada caso puede resultar bastante complicada y además está estrechamente relacionada con otros temas

[2]Por aisladas se entiende que no poseen ningún router IPv6 en el enlace.

que escapan al objetivo del presente proyecto fin de carrera, como por ejemplo *"multihoming"*. No obstante, es de referencia obligada para profundizar en el tema [**?**].

No obstante, en [**?**] se recomienda utilizar la siguiente regla para el algoritmo de selección de dirección fuente:

```
Destino = IPv4-compatible
  => Utilizar la dirección IPv6 compatible con IPv4 asociada con la dirección IPv4
     del interfaz de salida.

Destino = IPv6 nativo
  => Utilizar alguna dirección IPv6 nativa del interfaz de salida.
```

Si una máquina IPv4/IPv6 no posee ninguna dirección IPv6 nativa, pero está creando un paquete para un destino IPv6 nativo, podría utilizar la dirección "IPv4-compatible" como su dirección origen.

Clasificación del mecanismo.

- Ambito de aplicación: global.
- Requisitos de IPv4: conectividad IPv4 entre los sitios.
- Requisitos de las direcciones IPv4: 1 por máquina.
- Requisitos de IPv6: ninguno.
- Requisitos de las direcciones IPv6: direcciones IPv6 compatibles con IPv4.
- Requisitos de máquinas: doble pila.
- Requisitos de routers: ninguno.
- Impacto del NAT: no funcionará si el túnel tiene que atravesar un NAT.
- Otros requisitos: no.

Nótese que como esta solución requiere una dirección IPv4 por máquina, su dominio de aplicación se encuetra extremadamente limitado.

2.3.3 *Tunnel Broker.*

Los túneles configurados necesitan normalmente de la coordinación de las dos partes para levantar de forma correcta los extremos del túnel. Los *"tunnel brokers"*[**?**] pueden ayudar a obtener la información necesaria para levantar un túnel. Concretamente, autoconfiguran un extremo y proporcionan al usuario los datos necesarios para configurar su lado. Incluso algunas implementaciones[3] proporcionan scripts "a medida" que el usuario simplemente debe ejecutar para realizar la configuración de su extremo de una forma totalmente automatizada.

[3]

Las implementaciones actuales están basadas en herramientas web que permiten de forma interactiva la configuración de un túnel IPv6 sobre IPv4. Solicitando un túnel, el cliente puede obtener tanto una dirección como un conjunto de direcciones IPv6 del ISP. A continuación se describe brevemente el funcionamiento del mecanismo.

Introducción.

Un *"tunnel broker"* se puede ver como un ISP de IPv6 que ofrece conectividad utilizando túneles IPv6 sobre IPv4. Se basa en el modelo de la figura 2.3:

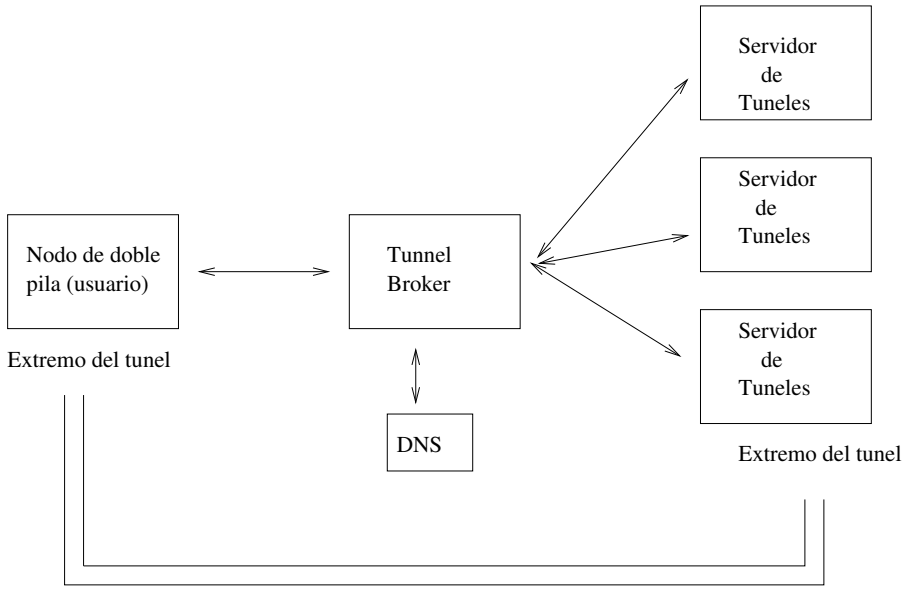

Figura 2.3: Modelo de elementos funcionales del *"tunnel broker"*.

El servidor de túneles es un router *"dual-stack"*. Cuando recibe una orden de configuración proviniente del *"tunnel broker"*, el servidor de túneles crea, modifica o borra el túnel especificado. Normalmente actúa además como recolector de estadísticas.

Funcionamiento.

El cliente del servicio de *"tunnel broker"* debe utilizar una máquina *"dual-stack"* conectada a la Internet IPv4. El *"tunnel broker"* debe solicitar al usuario datos necesarios para identificarle de forma unívoca, y opcionalmente datos para poder facturar por el servicio prestado. De esta forma, el servidor de *"tunnel broker"* se puede ver como un servidor de control de acceso.

Una vez que el cliente ha sido autorizado para utilizar el servicio, se debe proporcionar la siguiente información para poder realizar el túnel:

- La dirección IPv4 del extremo del túnel del cliente.
- Un nombre para utilizarse como entrada del DNS.

- La función que va a realizar el ciente, si va a actuar como máquina o como router. Por ejemplo, si el cliente quiere actuar como proveedor IPv6, el tunnel broker puede requerir más información, como por el ejemplo el número de direcciones IPv6 que el cliente necesita.

Nótese que los túneles consumen recursos en los servidores, en términos de memoria y tiempo de procesamiento. Por consiguiente, se debe mantener el número de túneles sin utilización lo más bajo posible. Es decir, se necesita una gestión de túneles apropiada.

Clasificación del mecanismo.

- Ambito de aplicación: global.
- Requisitos de IPv4: ninguno de forma específica.
- Requisitos de las direcciones IPv4: 1.
- Requisitos de IPv6: ninguno.
- Requisitos de las direcciones IPv6: ninguno en caso de ser un cliente aislado. Necesita la asignación de un prefijo si lo que se quiere conectar es una red.
- Requisitos de máquinas: doble pila, navegador de Internet IPv4.
- Requisitos de routers: ninguno.
- Impacto del NAT: no funcionará si el túnel tiene que atravesar un NAT. Puede funcionar si el túnel que utiliza el cliente está ubicado en la misma máquina que el NAT.
- Otros requisitos: Se necesita mantener una base de datos con los túneles actualmente en funcionamiento. La configuración del tunel en el extremo del servidor se debe realizar de forma automática (mediante un *"cgi"*, por ejemplo) y se deben habilitar mecanimos de *"keep-alive"* para poder detectar la caída de un extremo o para eliminar el túnel cuando no se haya recibido tráfico durante un determinado tiempo. También puede ser interesante realizar de manera automática una actualización segura del DNS.

2.3.4 6to4.

Este mecanismo se encuentra descrito en [**?**].

El mecanismo 6to4 se aplica cuando se necesita la interconexión de dominios IPv6 completos dentro de un mundo que es IPv4. El router de salida del dominio IPv6 crea un túnel con el otro dominio IPv6. Los extremos IPv4 del túnel se identifican dentro del prefijo de la dirección IPv6 de cada uno de los dominios implicados en la comunicación. Este prefijo se contruye a partir de un prefijo único 6to4 de 16 bits (2002) junto con un campo de 32 bits donde se almacena la dirección IPv4 del router de salida del sitio.

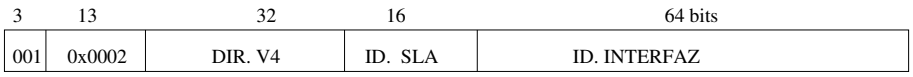

Figura 2.4: Formato de una dirección 6to4.

Introducción.

El mecanismo de túneles 6to4 consiste en utilizar una dirección IPv4 para definir un prefijo IPv6 global. Se utiliza el formato de dirección que se muestra en la figura 2.4:

Como se observa en el dibujo, se definen los siguientes valores:

```
Prefix length: 48 bits
Format prefix: 001
TLA value: 0x0002
NLA value: V4ADDR
```

De esta forma, este prefijo posee exactamente el mismo formato que los prefijos /48 definidos en IPv6. Se puede abreviar como "2002:V4ADDR::/48". Dentro de un determinado sitio puede utilizarse exactamente igual que cualquier otra dirección IPv6.[4]

Para asegurar el correcto funcionamiento del mecanismo 6to4 en topologías complejas, la selección de direcciones fuente y origen de los paquetes se debe implementar de una forma adecuada. Si la máquina IPv6 que está enviando el paquete posee al menos una dirección "2002::", y si el conjunto de direcciones IPv6 que devuelve el DNS para alcanzar el destino contiene al menos una dirección "2002::", entonces la máquina que inicia esta comunicación debe elegir de forma adecuada la dirección origen y destino. Aunque el tema de la selección de direcciones está bajo estudio en el momento de escribir este trabajo[**?**], el siguiente principio general asegura un correcto funcionamiento del mecanismo:

1. Si una máquina posee sólo direcciones 6to4, y el destino posee tanto direcciones 6to4 como otras direcciones IPv6 nativas, la dirección 6to4 debería ser utilizada tanto en el origen como en el destino de los paquetes.

2. Si ambas máquinas poseen una dirección 6to4 junto con una dirección IPv6 nativa, entonces o se usan las direcciones 6to4 tanto en el origen como en el destino, o se usan las direcciones nativas. Esta selección debería ser configurable. La configuración por defecto debería elegir las direcciones IPv6 nativas.

Existen dos tipos de routers 6to4, el denominado *"gateway 6to4"* o "router 6to4", y el *"relay router"*. Ambos términos se describen a continuación:

router 6to4: router IPv6 que soporta una pseudo-interfaz 6to4. Normalmente es un *"border router"* entre un sitio IPv6 y una red IPv4.

Un router 6to4 debe poseer al menos la siguiente información en su tabla de rutas IPv6:

[4] Otro tema más complicado es cómo realizar el mapeo inverso en el DNS.

- "2002::/16" se encamina hacia la pseudo-interfaz 6to4.
- "default" hacia cualquier router IPv6 (posiblemente alcanzable a través de la infraestructura IPv4, al menos en la fase de introducción inicial de IPv6 en internet, pero este hecho resulta irrelevante).

Por otro lado, se define un *"relay router"* de la siguiente forma:

***Relay router*:** un router 6to4 configurado para soportar tránsito de encaminamiento entre direcciones 6to4 y direcciones IPv6 nativas. Un *"relay router"* puede anunciar el prefijo 2002::/16 en el dominio IPv6 nativo.

La diferencia entre un router 6to4 y un *"relay router"* estriba en que mientras que un router 6to4 sólo puede alcanzar a otras islas 6to4 (utilizando la infraestructura IPv4) y hace uso de otros mecanismos de transición para acceder a otras direcciones IPv6, el *"relay router"* puede encaminar direcciones IPv6 nativas además de establecer conexión con cualquier isla 6to4. Todo *"relay router"* es además un router 6to4, pero un router 6to4 puede no ser un *"relay router"*, por ejemplo, en el caso de que sólo encamine direcciones 6to4.

De esta forma, el *"relay router"* al formar parte de la red IPv6, puede anunciar el prefijo 2002::/16 para que otras subredes que no implementan el mecanismo puedan acceder a dichas islas IPv4, actuando *"de facto"* como un *"relay"* o pasarela entre el mundo 6to4 y el mundo IPv6 nativo, de ahí el nombre de *"relay router"*.

Funcionamiento.

Los paquetes IPv6 que fluyen desde un sitio 6to4 se encapsulan en paquetes IPv4 cuando abandonan el sitio utilizando una conexión IPv4 interna. Dichos paquetes se transmiten utilizando el valor 41 en el campo de protocolo de la cabecera IPv4. La cabecera IPv4 contiene las direcciones IPv4 origen y destino. Una o ambas de dichas direcciones serán idénticas a la dirección IPv4 que forma parte de la dirección 6to4.

El único cambio a la forma de encaminar paquetes en IPv6 que supone implementar este mecanismo es que cada router 6to4 (y sólamente los routers 6to4) están obligados a implementar las siguientes reglas adicionales para reenviar y desencapsular paquetes.

En la regla utilizada para reenviar paquetes, que se va a comentar a continuación, el *"next hop"* se refiere al siguiente nodo IPv6 al que se pretende enviar el paquete, que puede no ser el destino final, sino el siguiente vecino IPv6 señalado por el mecanismo de encaminamiento. Si el destino final es una dirección 6to4, dicho destino se considerará como el *"next hop"* para el propósito de esta regla, mientras que si el destino final no es una dirección 6to4, y no es una dirección local al router, el *"next hop"* indicado por el mecanismo de encaminamiento o ruteo será la dirección 6to4 de un relay router.

Sin más preámbulos, a continuación se muestra la modificación de la regla de reenvío en IPv6 para soportar el mecanismo 6to4:

```
REGLA DE REENVIO PARA ROUTERS 6TO4
==================================
```

```
SI el siguiente salto IPv6 se equipara
con la entrada 2002::/16,
ENTONCES:
    1. aplicar consideraciones de seguridad;
    2. encapsular el paquete en IPv4:
           - dirección: el valor NLA o ‘‘V4ADDR’’
    3. encolar el paquete para reenviarlo usando IPv4.
```

Una regla del mismo estilo para desencapsular paquetes IPv4 con valor 41 en el campo del protocolo también debe implementarse, como se muestra a continuación:

```
REGLA DE DESENCAPSULACIÓN PARA ROUTERS 6TO4
==========================================

  - aplicar consideraciones de seguridad;
  - eliminar la cabecera IPv4;
  - introducir el paquete en el sistema
        de encaminamiento IPv6.
```

Para terminar comentar que como el mecanismo 6to4 no tiene ningún impacto en el encaminamiento de IPv4, no pueden provocarse bucles a nivel de IPv4. De igual forma, como el prefijo "2002::" puede considerarse exactamente igual que cualquier otro prefijo IPv6 global, el mecanismo tampoco introduce nuevas formas de provocar bucles a nivel de IPv6.

Clasificación del mecanismo.

- Ambito de aplicación: global.
- Requisitos de IPv4: conectividad IPv4.
- Requisitos de las direcciones IPv4: >= 1 por sitio.
- Requisitos de IPv6: prefijo 6to4 único de forma global.
- Requisitos de las direcciones IPv6: ninguno.
- Requisitos de máquinas: doble pila, mínima capacidad de selección de direcciones fuente.
- Requisitos de routers: implementación de reglas de reenvío y desencapsulación especiales.
- Impacto del NAT: no funcionará si el túnel tiene que atravesar un NAT. Puede funcionar si el router 6to4 se ubica donde se encuentre el NAT.
- Otros requisitos: prefijos de tamaño /48. Debe existir algún mecanismo para descubrir al *"relay router"*.

2.3.5 6over4.

El mecanismo 6over4, definido en [**?**], interconecta máquinas IPv6 aisladas dentro de un sitio utilizando la encapsulación de paquetes IPv6 dentro de IPv4 sin la utilización de túneles explícitos. A continuación se describe brevemente el mecanismo.

Introducción.

El objetivo de este mecanismo consiste en crear un enlace virtual utilizando un grupo IPv4 multicast con ámbito local-organizacional. En este punto conviene recordar que el multicast en IPv4 es un mecanismo opcional.

Las direcciones IPv6 multicast se mapean a direcciones IPv4 multicast para que se pueda ejecutar, entre otras cosas, el mecanismo de descubrimiento de vecinos definido en el protocolo IPv6[**?**]. Además, para encaminar entre la Internet IPv6 y el dominio 6over4 basta con configurar un router con este mecanismo en al menos uno de sus interfaces.

Funcionamiento.

El identificador de interfaz definido en [**?**] para una interfaz IPv4 consiste en la dirección IPv4 propiamente dicha, con los octetos en el mismo orden en que aparecerían dentro de un paquete IPv4, completando con ceros hasta alcanzar 64 bits.

De esta forma, la dirección IPv6 *link-local* de un interfaz virtual IPv4 se construye añadiendo el identificador de interfaz al prefijo fe80::/64, como se observa en la figura 2.5:

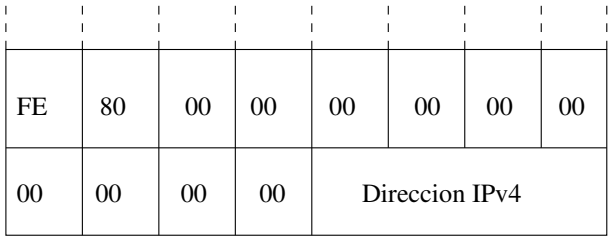

Figura 2.5: Identificador de interfaz para un enlace virtual IPv4.

El procedimiento para descubrir vecinos en IPv6 se encuentra descrito en [**?**]. La opción que almacena la dirección de la capa de enlace tiene la forma de la figura 2.6 cuando dicha capa es IPv4, como ocurre en este mecanismo:

Figura 2.6: Opción que contiene la dirección de la capa de enlace.

Resumiendo, se ha definido el proceso de autoconfiguración y el procedimiento para descubrir vecinos. Sólo falta conocer como se transmiten los paquetes IPv6 multicast dentro de este mecanismo.

Para ello, debe disponerse de multicast en IPv4. Un paquete IPv6 multicast se trasmite a una dirección IPv4 también multicast de ámbito organizacional-local, utilizando el mapeo definido en la figura 2.7:

DST14 y DST15 hacen referencia a los dos últimos bytes de la dirección IPv6 multicast, mientras que OLS es un valor tomado del bloque reservado para el ámbito organizacional-local de direccionamiento multicast IPv4[**?**], por ejemplo, el valor 192.

239	OLS	DST14	DST15

Figura 2.7: Mapeo entre direcciones multicast.

De esta forma, para soportar el protocolo de descubrimiento de vecinos se necesitan los siguientes grupos multicast IPv4. La siguiente lista se obtiene utilizando el mapeo anteriormente definido para derivar la dirección IPv4 a partir de la correspondiente dirección IPv6 multicast:

- dirección *"all-nodes"*: "239.OLS.0.1"
- dirección *"all-routers"*: "239.OLS.0.2"
- dirección *"solicited-node"*: esta dirección multicast se calcula como una función de la dirección IPv4 utilizando los 24 bits de menor orden y prefijando el resultado con "ff02:0:0:0:1:ff00::/104". El resultado final se mapea utilizando el procedimiento descrito anteriormente para generar la correspondiente dirección IPv4. Por ejemplo, si la dirección IPv4 utilizada para formar la dirección IPv6 es "W.X.Y.Z", entonces la dirección *"solicited-node"* es "ff02::1:255.X.Y.Z" y la correspondiente dirección IPv4 multicast es "239.OLS.Y.Z".

Clasificación del mecanismo.

- Ambito de aplicación: dominio.
- Requisitos de IPv4: Conectividad IPv4 multicast entre hosts.
- Requisitos de las direcciones IPv4: 1 por host.
- Requisitos de IPv6: ninguno.
- Requisitos de las direcciones IPv6: ninguno.
- Requisitos de hosts: doble pila.
- Requisitos de routers: Configuración del mecanismo 6over4 para encaminar entre diferentes enlaces virtuales y/o enlaces virtuales y la internet IPv6.
- Impacto del NAT: Requerirá un gran esfuerzo para hacerlo funcionar puesto que primeramente se tendrá que habilitar el multicast IPv4 a través de NATs.
- Otros requisitos: Para conectar máquinas IPv6 de diferentes enlaces físicos, el encaminamiento multicast en IPv4 debe estar habilitado en los routers que interconectan dichos enlaces.

2.3.6 ISATAP.

ISATAP son las iniciales de *"Intra-Site Automactic Tunnel Addressing Protocol"*. Definido en [**?**], se trata de un mecanismo de transición que permite un desarrollo incremental de IPv6 utilizando la infraestructura de IPv4 como un enlace *"Non-Broadcast Multiple Access"* o NBMA.

Este mecanismo hace uso de un nuevo formato de identificador de interfaz que contiene en su interior una dirección IPv4 (que puede ser pública o privada), lo que permite la encapsulación automática en IPv4.

Introducción.

Las direcciones ISATAP son direcciones IPv6 unicast con un formato especial para la parte de identificador de interfaz y que utilizan una dirección IPv4 codificada en *"network byte order"*, como se observa en la figura 2.8:

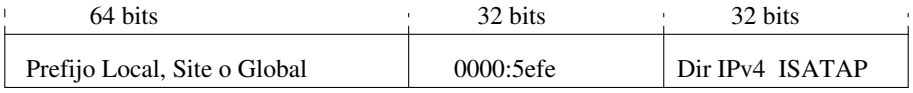

Figura 2.8: Formato de las direcciones ISATAP.

Un nodo puede configurar un enlace ISATAP sobre uno o varios enlaces IPv4. Cada dirección IPv4 se utiliza para configurar una dirección ISATAP *link-local* "fe80::0:5efe:V4ADDR_LINK".

La dirección IPv6 del siguiente salto para paquetes enviados a través de un enlace ISATAP debe ser una dirección ISATAP. Respecto al procedimiento para mapear direcciones unicast en direcciones de la capa de enlace, basta con tratar los últimos cuatro octetos de la dirección ISATAP como una dirección IPv4. En el momento de escribir este trabajo, no se ha definido el mapeo del direcciones multicast.

Dado que el mapeo de direcciones ISATAP a direcciones de la capa de enlace se realiza mediante un cómputo estático de la propia dirección ISATAP, las funciones y las estructuras de datos utilizadas para el mecanismo de descubrimiento de vecinos[**?**] no son necesarias en este caso. La "cache de vecinos", no obstante, se necesita todavía para otros menesteres, como para la detección de vecinos inalcanzables, pero no se utiliza para resolución de direcciones.

Funcionamiento.

La comunicación se realiza encapsulando de forma automática el paquete IPv6 en IPv4. Las direcciones de la cabecera IPv4 se extraen a partir de las propias direcciones ISATAP. Los máquinas con soporte de ISATAP utilizan una nueva estructura de datos denominada "lista de routers potenciales". Esta lista proporciona el contexto para descubrir los vecinos y un modelo básico de confianza para validar los routers existentes en el enlace ISATAP.

Cada entrada de la lista posee una dirección IPv4 y un contador asociado utilizado para probar la conectividad. La dirección IPv4 se utiliza para construir una dirección ISATAP *link-local* para poder comunicarse con el router ISATAP.

Cuando un nodo habilita un enlace ISATAP, debe inicializar su lista de routers potenciales de la siguiente forma:

1. El administrador del dominio mantiene registros de DNS para los distintos interfaces ISATAP de los routers del dominio. Para poder automatizar el proceso, se recomienda usar el nombre "isatap.domain-name".

2. Los nodos intentan descubirir una o más direcciones para la lista preguntando al DNS por los anteriores registros.

3. Despues de la inicialización de la lista de routers potenciales, los nodos periódicamente repiten el punto anterior, cada *"ResolveInterval"* segundos, para actualizar la lista.

Por lo tanto, se define una variable de configuración denominada *"ResolveInterval"*, que indica el tiempo entre consultas contra el servidor de nombres, con un valor sugerido mínimo de una hora.

Gracias a este proceso, un nodo puede descartar cualquier paquete de *"router advertisement"* cuya dirección IPv4 (que como siempre, se encuentra dentro de la propia dirección ISATAP) no se encuentre listada en la lista de routers potenciales. Además, también se debe comprobar que la dirección fuente IPv6 se deriva a partir de la dirección IPv4 encontrada en la lista de routers potenciales.

Las máquinas periódicamente comprueban cada entrada de la tabla de routers potenciales enviando mensajes unicast de *"router solicitation"*. Para soportar este proceso, las máquinas utilizan la variable *"MinRouterSolicitInterval"*, que se define como el tiempo mínimo entre el envío de mensajes de *"router solicitation"* y que posee como valor por defecto y mínimo quince minutos.

Cuando una máquina recibe un paquete de *"router advertisement"* válido, lo procesa de la forma normal establecida en IPv6 y resetea el contador asociado con la entrada de la lista de router potenciales para dicho *"router advertisement"*. El contador se resetea de la siguiente forma:

```
MAX( 0.5 * MIN (router_lifetime, valid_lifetime_pref),
     MinRouterSolicitInterval )
```

Por otra parte, los routers advierten las direcciones ISATAP de sus interfaces exactamente igual que ocurre en IPv6. No obstante, el envío periódico de *"routers advertisements"* no solicitados ya no resulta necesario.

Clasificación del mecanismo.

- Ambito de aplicación: dominio.
- Requisitos de IPv4: ninguno.
- Requisitos de las direcciones IPv4: 1 por máquina.
- Requisitos de IPv6: ninguno.

- Requisitos de las direcciones IPv6: ninguno.
- Requisitos de máquinas: doble pila.
- Requisitos de routers: ninguno.
- Impacto del NAT: no funciona a través de NATs, pero sí que funciona dentro de la parte privada de la red gestionada por el posible NAT. Otros mecanismos de transición se encuentran específicamente diseñados para trabajar a través de NATs, por lo que ISATAP puede utilizarse como un mecanismo complementario.
- Otros requisitos: el procedimiento de descubrimiento de vecinos definido en [**?**] implica que los nodos confían completamente en los *"router advertisements"* recibidos de routers del mismo enlace. Como el campo *"hop-limit"* no se decrementa dentro de paquetes encapsulados, ISATAP requiere un modelo de confianza diferente. En concreto, sólo se confía en los *"router advertisements"* recibidos de miembros de la lista de routers potenciales. Este modelo de confianza se obtiene realizando filtrado de la dirección IPv4 origen.

2.3.7 Teredo.

El mecanismo Teredo, definido en [**?**] no es más que una solución a corto plazo al problema específico de proporcionar servicio IPv6 a nodos localizados por detrás de un NAT. Este mecanismo sólo se debe utilizar cuando no sea posible transformar el NAT IPv4 en un router IPv6.

Introducción.

Los métodos de encapsulación utilizados para transitar hacia IPv6 funcionan utilizando como *payload* IPv4 los paquetes IPv6. Un problema que surge con todos estos métodos es el hecho de que no funcionan cuando el nodo destino se encuentra detrás de un NAT, excepto cuando existe un mapeo inverso en el NAT, lo cual no suele ocurrir salvo configuración manual.

Los NATs, normalmente, no están programados para permitir la transmisión de tipos arbitrarios de *payloads*. Además, en el caso de los túneles 6to4, las direcciones locales o internas no pueden ser utilizadas. El esquema 6to4 funcionará con el NAT sólo si el NAT y el router 6to4 se encuentran en la misma máquina. El mecanismo de transición denominado Teredo[5], descrito en [**?**], trata de resolver el caso relativamente frecuente en que una máquina NAT no puede ser fácilmente actualizada para proporcionar funciones de 6to4 o de router IPv6.

En cuanto se consiga esto, no será necesario utilizar este mecanismo de transición, puesto que transportar paquetes IPv6 utilizando UDP y mediante *"relays"* o terceras partes resulta bastante ineficiente, en comparación con mecanismos "más directos" como los túneles 6to4.

[5] Anteriormente denominado por el propio autor como *"Shipworm"*, este nombre fue desechado por considerarse desagrable.

Funcionamiento.

El mecanismo hace uso de los siguientes elementos:

- Cliente Teredo: se trata de una máquina, normalmente ubicada detrás de un NAT, que adquiere una dirección IPv6 utilizando un servidor Teredo, y se comunica con el destino encapsulando los paquetes IPv6 en paquetes UDP que se envían hacia el servidor Teredo más cercano.
- *"Relay"* Teredo: router encargado, por un lado, de anunciar el prefijo IPv6 de servicio Teredo a la red IPv6, y por otro lado capaz de enviar paquetes UDP sobre IPv4 utilizando como fuente la dirección IPv4 anycast reservada para el servicio Teredo.
- Servidor Teredo: encargado de asignar un prefijo IPv6 específico para el servicio Teredo, mediante el envío de *"router advertisements"* bajo demanda. Se localizan utilizando una dirección IPv4 anycast reservada para el servicio Teredo, y no guardan estado. El único requisito que necesita un servidor Teredo para convertirse en un *"relay"* Teredo es el ser capaz de advertir el prefijo IPv6 de servicio Teredo en la zona IPv6. Esto no tiene por que ser fácil de conseguir, si, por ejemplo, el servidor Teredo adquiere su conectividad IPv6 a través del mecanismo 6to4.

Existen muchos detalles asociados al mecanismo de transición Teredo, por lo que aquí sólo se van a exponer los casos más sencillos, de forma resumida, para tener una idea general de cómo funciona el mecanismo. Para mas detalles se remite al lector a [**?**]

En la figura **??** que se muestra a continuación, se observa el recorrido de los distintos paquetes dentro de la topología Teredo, para el caso de establecimiento de comunicación desde un nodo Teredo hasta un nodo IPv6 y viceversa:

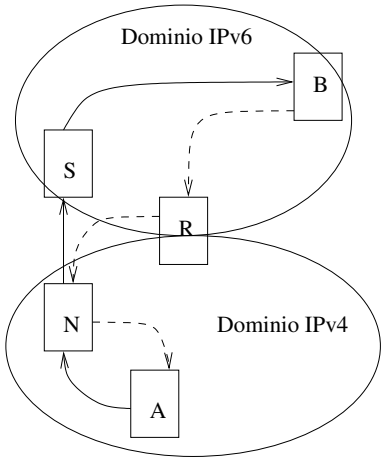

Figura 2.9: Envío y recepción de paquetes usando Teredo.

"A" es el cliente Teredo, situado detrás del NAT "N". "S" es el servidor Teredo más cercano a "A" en la topología de Internet IPv4. "B" es el nodo IPv6 destino y "R" es un *"relay"* Teredo cercano a "B" en la topología IPv6.

Se supone que "A" ha obtenido su dirección IPv6 mediante el intercambio de mensajes de *"router solicitation"* y *"router advertisement"* con el servidor Teredo.

De esta forma, "A" envía un paquete con dirección IPv6 destino la dirección de "B", y con direccion IPv6 fuente la dirección IPv6 recién obtenida, por ejemplo, PTER:0900:0001:1000::A [6]. Este paquete IPv6 es encapsulado por "A" en un datagrama UDP, con dirección origen la dirección IPv4 del nodo, y con dirección destino la dirección IPv4 anycast reservada para el mecanismo Teredo.

El paquete así enviado llega hasta el NAT "N". "N" utiliza el mapeo existente, establecido al solicitar la dirección IPv6, y reemplaza la dirección IPv4 y el puerto origen con sus propios valores ya establecidos para esa conexión, antes de reenviar el paquete.

Dicho paquete es recibido ahora por el servidor Teredo más cercano, en este caso "S", quien reenvía el contenido del paquete sobre IPv6, el cual eventualmente alcanzará el destino "B", y que poseerá como dirección origen PTER:0900:0001:1000::A. El camino seguido por el paquete desde el nodo origen hasta el nodo destino se representa en el dibujo de la figura 2.9 de la página 60 mediante una línea contínua.

Si "B" quiere enviar un paquete de respuesta a "A", se procede ahora del siguiente modo, según se puede observar siguiendo la línea punteada de la figura 2.9 anterior.

Como el paquete de respuesta se envía a la dirección IPv6 de "A" que forma parte del prefijo Teredo, dicho paquete se encamina por IPv6 hasta un *"relay"* Teredo, que es el encargado de advertir el prefijo Teredo en la red IPv6.

El *"relay"* Teredo recibe el paquete, extrae la dirección IPv4 y el puerto que se encuentran dentro de la dirección IPv6 destino y encapsula el paquete IPv6 en un paquete UDP/IPv4. Este paquete se encamina sobre IPv4 y es recibido por el NAT "N". Dado que el NAT ya tiene establecido un mapeo para ese destino y puerto, "N" simplemente utiliza dicha información para reemplazar la dirección IPv4 y el puerto destino por los correspondientes valores del nodo "A". El paquete se reenvía hacia el domino situado bajo el NAT hasta que alcanza al nodo "A", el cual desencapsula el paquete y procesa el paquete IPv6.

Ahora se va a presentar el caso de intercambio de información entre dos nodos Teredos. Este caso resulta muy similar al anterior. Supongamos que existen dos servidores Teredo,"S1" y "S2", y dos NATs, "N1" y "N2", y que cada nodo se encuentra en su respectivo dominio privado.

Cuando "A" quiere enviar a "B", el primer paquete atraviesa "N1", siguiendo el mismo mecanismo comentado en el apartado anterior, hasta que alcanza al nodo "B". Pero al mismo tiempo que "A" envía el paquete a través del servidor Teredo, "A" creará un paquete de prueba y lo enviará hacia "B" utilizando un camino directo, es decir, directamente contra la dirección pública del NAT de "B" (representado en la figura mendiante líneas discontínuas), la cual puede extraerse de la dirección IPv6 destino. Este paquete de prueba se envía a través del NAT "N1", donde crea un mapeo, y se reenvía por IPv4 hasta que alcanza al NAT "N2", donde se descarta, debido a que no existe ningún mapeo establecido entre la dirección de "A" y la dirección de "B". El recorrido se muestra en la figura 2.10 que se muestra a continuación:

De todas formas, cuando "B" responde a "A", el mecanismo es totalmente simétrico y "B" también enviará un paquete de prueba hacia "A", al darse cuenta de que se trata de un destino Teredo. Este paquete de prueba atraviesa el NAT "N2", donde crea un mapeo y

[6]El formato de las direcciones Teredo es el siguiente: PTER corresponde al prefijo reservado por el IANA, los siguientes 32 bits ("0900:0001") se corresponden con la dirección pública IPv4 del NAT, y los siguientes 16 bits ("1000") se corresponden con el puerto externo utilizado por el NAT. Los bits restantes constituyen un identificador del nodo.

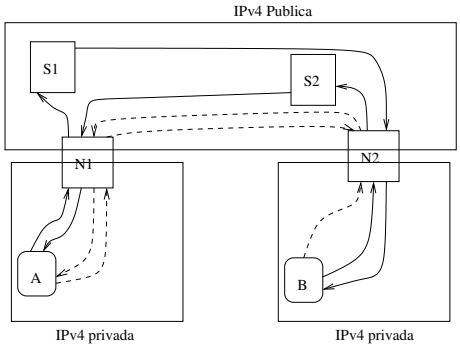

Figura 2.10: Intercambio entre dos nodos Teredo.

después se reenvía hacia el NAT "N1". Como el primer paquete de prueba enviado por "A" ya ha establecido un mapeo en "N1" entre "A" y "B", el paquete de prueba enviado por "B" es aceptado y llega hasta el nodo "A".

En estos momentos, como "A" ha recibido un paquete de prueba del nodo "B", sabe que puede atravesar tanto su NAT como el NAT de "B", y por consiguiente enviará los subsiguientes paquetes utilizando un camino directo. Cuando "B" reciba dichos paquetes, actuará de igual forma.

Por último, se debe considerar el escenario que se presenta cuando se quiere intercambiar información entre dos nodos Teredo que se encuentran en el mismo enlace. Este escenario en concreto se puede optimizar para que la comunicación no tenga que pasar siempre a través del NAT.

La optimización se produce de la siguiente forma. Una vez que ambos nodos han adquirido una dirección IPv6 utilizando un servidor Teredo, para poder establecer una comunicación directa a través del enlace, ambos nodos envían paquetes de prueba hacia la dirección multicast reservada *"Teredo IPv4-Discovery Address"*. Estos paquetes de prueba permiten descubrir la dirección IPv4 local y el puerto local asociados con la dirección Teredo IPv6 de una determinada máquina. Los paquetes de datos pueden a partir de este momento ser enviados sobre UDP a esta dirección, evitando la utilización del camino más largo a través del servidor Teredo.

Clasificación del mecanismo.

- Ambito de aplicación: global o de dominio.
- Requisitos de IPv4: oara ser compatible con el filtrado de entrada, las direcciones IPv4 anycast deben ser topológicamente correctas.
- Requisitos de las direcciones IPv4: 1 dirección anycast para todos los sevidores y *"relays"*.
- Requisitos de IPv6: ninguno.
- Requisitos de las direcciones IPv6: 1 prefijo para el servicio Teredo.
- Requisitos de máquinas: ninguno.

- Requisitos de routers: ninguno.
- Impacto del NAT: el mecanismo funciona a través de la mayoría de NATs.
- Otros requisitos: Teredo debe ser compatible con el filtrado de entrada de los routers.

2.4 Mecanismos de comunicación entre nodos IPv4 y nodos IPv6.

Cuando las islas IPv6 se interconectan utilizando uno o varios de los mecanismos comentados anteriormente, se habilita la comunicación entre nodos IPv6, pero puede ser conveniente o útil establecer comunicación entre un nodo IPv6 y un nodo IPv4. Esto se puede realizar de varias formas, utilizando traducción a nivel de aplicación, por ejemplo, o traduciendo a nivel de red o incluso mediante la asignación temporal de una dirección IPv4 en la máquina IPv6. Un traductor de protocolos mapea campos de un protocolo en campos semejantes de otro protocolo. Nótese que existen aplicaciones IPv4 que utilizan información de la capa de red, una de las más conocidas es "ftp". Esto hace necesario no sólo traducir los paquetes a nivel de red sino también traducir los paquetes a nivel de aplicación, necesitándose lo que se conoce como un "ALG" o *"Application Level Gateway"*.

La mayoría de los mecanismos que se van a presentar a continuación hacen uso de implementaciones de ambas pilas de protocolos de red, lo que se conoce como implementaciones *"dual stack"*. El modelo *"dual stack"* presenta las siguientes características:

- Puede evitar túneles: si routers que antes eran IPv4 ahora son *"dual-stack"*, ya no son necesarios.
- Puede evitar traducciones: pero sólamente cuando la aplicación soporte IPv6 e IPv4.

Además, la mayoría de las implementaciones del protocolo IPv6 se han realizado integrando el nuevo protocolo dentro de IPv4, por lo que la mayoría de las implementaciones actuales son *"dual-stack"*, aunque el término no sea el más apropiado puesto que a nivel de implementación no existen dos pilas de protocolos diferenciadas.

Resumiendo, en este apartado, después de describir el mecanismo DSTM, se describen los mecanimos relacionados con la traducción entre protocolos, comentando en primer lugar los mecanismos de traducción a nivel de red, en concreto SIIT, NAT-PT y BIS. Posteriormente se trata el mecanismo TRT de traducción a nivel de transporte y finalmente se comentan los mecanismos disponibles a nivel de aplicación: SOCKS y BIA.

Para finalizar esta introducción, resulta necesario comentar que existen otros mecanismos de transición que no han sido descritos en este apartado. Muchos de ellos se encuentran en estado de *"draft"* en el momento de escribir este trabajo. Para una explicación técnica detallada de éstos y otros mecanismos de transición, se remite al lector a los documentos proporcionados por el grupo de trabajo *"ngtrans"* del IETF[7].

[7] http://www.ietf.org/html.charters/ngtrans-charter.html

2.4.1 DSTM.

El mecanismo denominado en inglés DSTM o *"Dual Stack Transition Mechanism"*, definido en [**?**], es un mecanismo que permite a un nodo *"dual stack"* cuya pila IPv4 está habilitada pero todavía no configurada adquirir una dirección IPv4 para comunicarse con otras aplicaciones IPv4-only. A continuación se realiza una descripción resumida de su funcionamiento.

Introducción.

La idea principal consiste en que cuando un nodo IPv4/IPv6 necesita una dirección IPv4, solicita una al servidor DSTM. La comunicación con el servidor DSTM se realiza sobre IPv6.

En la ausencia de una infraestructura de encaminamiento de IPv4 (en redes IPv6-only, por ejemplo), la máquina *"dual stack"* encapsulará paquetes IPv4 dentro de paquetes IPv6 hasta el extremo del túnel el cual los desencapsulará y los inyectará en la infraestructura IPv4. Esta encapsulacion se realiza mediante una interfaz virtual.

Por consiguiente, la arquitectura DSTM se compone de los siguiente elementos:

- Servidor DSTM: el servidor se encarga fundamentalmente de asignar direcciones IPv4 a los clientes que lo soliciten. Esta asignación resulta bastante sencilla ya que no existe ningún propósito de localización sino sólamente de identificación. Además, el servidor de DSTM tiene que garantizar la unicidad de la dirección IPv4 sólo durante un instante de tiempo determinado. Algunas implementaciones puede incluir un rango de puertos como parte del proceso de asignación de direcciones, de esta forma, se podría usar la misma dirección IPv4 varias veces de forma simultánea. Nótese que la forma exacta de realizar la comunicación entre servidor y cliente (y viceversa) no se especifica en [**?**].
- Router DSTM: también denominado TEP o *"Tunnel End Point"*, se trata de un *border router* entre el dominio IPv6-only y el mundo IPv4. Este elemento realiza la encapsulación y desencapsulación de paquetes para asegurar un reenvío bidireccional entre ambos mundos.
- Máquinas DSTM: las máquinas que hacen uso del mecanismo DSTM deben ser capaces de configurar dinámicamente su pila IPv4 y deben ser capaces de establecer túneles IPv4 sobre IPv6.

Funcionamiento.

En el diagrama de la figura 2.11 se muestra de una forma esquemática los pasos y elementos involucrados en el funcionamiento del mecanismo DSTM:

Cuando una máquina necesita enviar un paquete IPv4, dicha máquina debe contactar con un servidor DSTM. El servidor proporciona al nodo una dirección IPv4 temporal junto con la dirección IPv6 del TEP que actuará como extremo remoto del túnel.

Esta información se utiliza para configurar la interfaz túnel de la máquina origen. La encapsulación se produce introduciendo el paquete IPv4 dentro de un paquete IPv6, por lo tanto estamos hablando de un túnel *"4-over-6"*, al contrario que la mayoria de los túneles, que resultan ser *"6-over-4"*. Podemos decir que en este punto es donde se configura la pila IPv4 de la máquina.

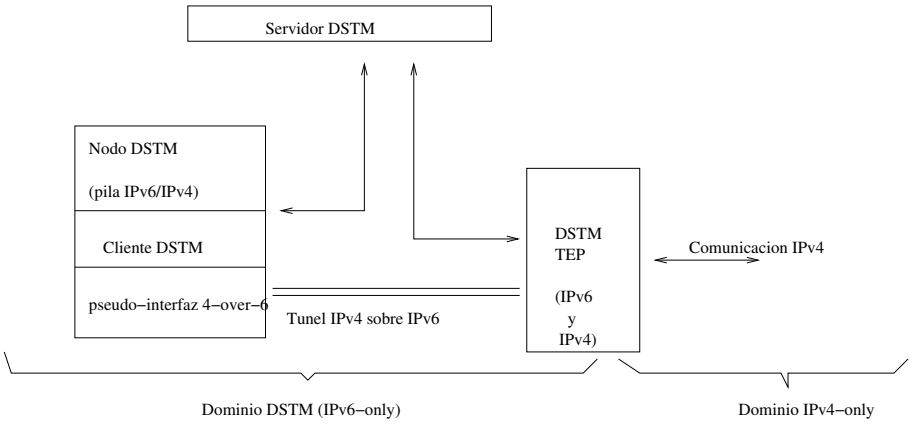

Figura 2.11: Esquema DSTM.

De esta forma, el paquete IPv4 se envía a través de dicha pseudo-interfaz. El extremo del túnel se ha obtenido mediante el servidor DSTM, aunque también podría ser configurado de forma estática. Cuando el paquete llega al extremo remoto, éste lo desencapsula y el paquete se procesa por la pila IPv4. Dicho paquete será reenviado utilizando la infraestructura IPv4 a partir de estos momentos. Por consiguiente, el extremo del túnel tiene que ser una máquina que actúe entre la frontera de los mundos IPv6 e IPv4, como se ha comentado anteriormente.

El servidor DSTM debe memorizar el mapeo entre la dirección IPv6 del nodo solicitante y la dirección IPv4 asignada a dicho nodo. Si la dirección IPv4 temporal asignada está a punto de caducar, el nodo debe renovar su alquiler contactando con el servidor. Se trata de un mecanismo muy semejante a un servidor de DHCPv6, pero donde se alquila una dirección IPv4 en lugar de una dirección IPv6.

El sistema de encaminamiento de IPv4 debe garantizar que el *"pool"* de direcciones IPv4 manejadas por el servidor DSTM se encamina hacia uno o más TEPs del dominio DSTM. Cuando el servidor DSTM asigna una dirección IPv4 también debe comunicar el TEP encargado de dicha dirección. Adicionalmente, el servidor DSTM puede configurar la tabla de mapeos IPv4/IPv6 del TEP.

Por supuesto, la comunicación entre el cliente DSTM y el servidor se debe producir por IPv6. Además, el servidor de DSTM y el TEP pueden estar situados en una misma máquina, siempre y cuando dicha máquina posea conectividad IPv6 e IPv4, requisito imprescindible para poder actuar como TEP.

Clasificación del mecanismo.

- Ambito de aplicación: dominio.
- Requisitos de IPv4: ninguno.
- Requisitos de las direcciones IPv4: >=1 por sitio.
- Requisitos de IPv6: extensiones para DHCPv6.
- Requisitos de las direcciones IPv6: ninguno.

- Requisitos de máquinas: pila IPv4/IPv6 con extensiones.
- Requisitos de routers: ninguno.
- Impacto del NAT: las comunicaciones se realizan utilizando IPv4, de forma que pueden ser penalizadas por NATs que se encuentren a lo largo del camino.
- Otros requisitos: estructura de encaminamiento de IPv4.

2.4.2 SIIT.

El protocolo SIIT, descrito en [**?**], traduce los paquetes a nivel de red entre IPv4 e IPv6. La traducción se limita a la cabecera IP. Este traductor opera "sin estado" lo que significa que la traducción debe realizarse para cada paquete. En los sucesivos apartados se describe con más detalle el mecanismo.

Introducción.

Además de las direcciones IPv6 definidas en [**?**] el mecanismo SIIT introduce las direcciones *"IPv4-translated"* o direcciones "IPv4 traducidas". De esta forma, este mecanismo hace uso de los siguientes tipos de direcciones:

- Direcciones IPv4 mapeadas: son las direcciones tipo "::ffff:a.b.c.d". Estas direcciones se utilizan dentro de este mecanismo de transición para identificar a una máquina IPv4.
- Direcciones IPv4 traducidas: son las direcciones del tipo "::ffff:0:a.b.c.d" que identifican a máquinas IPv6.

Cuando una aplicación que se ejecuta en una máquina IPv6-only intenta enviar un paquete a una dirección IPv4 o a una dirección IPv4 mapeada, el paquete será descartado y se devolverá un mensaje de error a la aplicación. Para poder utilizar el mecanismo SIIT se debería enviar un paquete IPv6 con dirección destino una dirección IPv4 mapeada y como dirección fuente una dirección IPv4 traducida. Este razonamiento también se aplica a máquinas de doble pila cuando no tienen configurada ninguna dirección IPv4. Por consiguiente, las máquinas que hagan uso de los traductores SIIT necesitan modificar ligeramente la implementación del protocolo IPv6 para ser capaces de realizar las siguientes tareas:

- Permitir el envío y la recepción de paquetes IPv6 con direcciones IPv4 mapeadas.
- Determinar cuando una dirección IPv4 traducida necesita ser asignada o refrescada.
- Asegurar, como parte del mecanismo de selección de dirección origen, que cuando una dirección destino resulta ser una dirección IPv4 mapeada la dirección fuente debe ser una dirección IPv4 traducida. Las direcciones IPv4 traducidas sólo pueden utilizarse en conjunción con las direcciones IPv4 mapeadas.

A continuación se describe el funcionamiento del traductor en ambos sentidos, sin entrar en detalles técnicos de implementación.

Funcionamiento.

Una de las diferencias existentes entre el protocolo IPv4 y el protocolo IPv6 es que el proceso de descubrimiento de la MTU es obligatorio en IPv6 mientras que resulta opcional en IPv4. Esto significa en que IPv6 un router nunca fragmentará un paquete (sólo el origen podrá hacerlo).

De esta forma, cuando se traduce de IPv4 a IPv6, se debe tener en cuenta algunos aspectos relacionados con la MTU:

- El proceso de descubrimiento de la MTU, que en IPv4 se realiza utilizando el bit "DF" de la cabecera, puede pasar a través del traductor sin ningún problema. Los posibles mensajes ICMP de routers IPv4 o IPv6 pueden reenviarse al nodo IPv4. En el caso de mensajes ICMP procedentes de routers IPv6, el mesanje se debe traducir adecuadamente.

- Si el nodo no realiza descubrimiento de la MTU, el traductor tiene que asegurar que el paquete no excede la MTU en la parte IPv6. Esto se realiza fragmentando el paquete IPv4 de forma que el paquete IPv6 ocupe 1280 bytes, puesto que IPv6 garantiza que 1280 bytes nunca necesitarán ser fragmentados. Para que el paquete IPv6 construido pueda atravesar otros traductores de IPv6 a IPv4, resulta necesario incluir una cabecera de fragmentación. De esta forma se indica que el origen, que en este caso es una máquina IPv4, permite fragmentación.

El mecanismo SIIT define los mapeos necesarios entre las cabeceras IPv4 e IPv6 y también entre cabeceras ICMPv4 y cabeceras ICMPv6. Si existen opciones de IPv4, son ignoradas. No obstante, si se encuentra una opción de encaminamiento basado en fuente, el paquete se debe descartar y se debe enviar un mensaje ICMPv4 de *"destination unreachable/source route failed"*. También se define la forma de añadir una cabecera de fragmentación cuando el bit "DF" no está activado o cuando el paquete resulta ser un fragmento.

Respecto a la traducción de paquetes de IPv6 a IPv4, existen también algunas consideraciones relativas a la fragmentación y la MTU que deben tenerse en cuenta.

Un enlace IPv6 debe tener al menos una MTU de 1280 bytes. El límite más pequeño en IPv4 se encuentra en 68 bytes. Así, a priori parece que la traducción desde IPv6 a IPv4 resulta imposible. No obstante, en la especificación del protocolo IPv6[**?**] se establece que un nodo IPv6 que recibe un paquete ICMP *"too big"* debe actuar reduciendo la MTU hasta 1280 y añadiendo una cabecera de fragmentación en cada paquete. De esta forma, este mecanismo no permite realizar el proceso de descubrimiento de MTU siempre y cuando el camino sea mayor o igual que 1280 bytes. Cuando el camino se vea reducido por debajo de 1280 bytes, el nodo IPv6 origen enviará paquetes de tamaño 1280 que resultarán fragmentados por los routers IPv4, después de ser traducidos.

Si el paquete IPv6 posee cabeceras *"hop-by-hop"*, cabeceras de destino, o cabeceras de *"routing header"* con el campo *"segments left"* igual a cero, dichas extensiones resultan ignoradas, es decir, no se realiza ningún esfuerzo para intentar traducirlas. No obstante, el campo de longitud total y el campo de protocolo de la cabecera IPv6 deben ajustarse para saltar dichas cabeceras de extensión.

Si el paquete IPv6 posee una *"routing header"* con algún segmento, dicho paquete no se debe traducir y se debe enviar un paquete ICMP *"parameter problem/erroneus header field*

encountered" al nodo fuente. De igual modo que en el caso anterior, si el paquete IPv6 contiene una cabecera de fragmentación, el mecanismo define las acciones necesarias para traducirla a IPv4.

La forma de traducir cada campo de las cabeceras IPv4 e IPv6 se puede consultar en [**?**].

Clasificación del mecanismo.

- Ambito de aplicación: dominio.
- Requisitos de IPv4: ninguno.
- Requisitos de las direcciones IPv4: 1 dirección temporal por cada máquina IPv6.
- Requisitos de IPv6: ninguno.
- Requisitos de las direcciones IPv6: direcciones IPv4 mapeadas y direcciones IPv4 traducidas para identificar los nodos IPv4 y los nodos IPv6 respectivamente.
- Requisitos de máquinas: pila IPv6.
- Requisitos de routers: ninguno.
- Impacto del NAT: puede que el paquete se traduzca más de una vez.
- Otros requisitos: algún mecanismo de asignación de direcciones. Los requisitos necesarios para que las aplicaciones funcionen correctamente a través de este mecanismo son exactamente los mismos que las aplicaciones deben cumplir para poder funcionar en un entorno *"dual-stack"*.

2.4.3 NAT-PT.

El mecanismo de transición denominado NAT-PT, descrito en [**?**], permite establecer comunicación entre nodos IPv6-only y nodos IPv4-only. La comunicación se realiza utilizando un dispositivo dedicado que mantiene el estado durante el tiempo que dure una sesión. NAT-PT también incluye un DNS-ALG para traducir los mensajes de consulta y respuesta del DNS. A lo largo de los siguientes subapartados se describe en profundidad el mecanismo.

Introducción.

Antes de describir el mecanismo de transición conviene comentar los distintos tipos de NATs que existen en IPv4. Actualmente se pueden distinguir los siguientes tipos, según se describe en [**?**]:

- NAT tradicional: permiten que máquinas del dominio interno accedan de forma transparente a máquinas de redes externas. En este tipo de NATs, las sesiones son unidireccionales y *"outbound"*, es decir, desde el dominio interno hacia el dominio externo. Estos NATs no anuncian redes privadas hacia el dominio externo pero pueden anunciar redes externas dentro del dominio privado. Suelen ser utilizados por sitios que poseen direccionamiento privado. Existen dos variaciones del NAT tradicional:

  - NAT básico: se basa en la utilización de un *"pool"* de direcciones para realizar los mapeos.
  - NAPT: utiliza además de la traducción de direcciones la traducción de puertos, multiplexando varias traducciones sobre una única dirección externa.

- NAT bidireccional: permiten que el tráfico se pueda iniciar tanto desde el dominio externo como desde el dominio interno. Se requiere un DNS-ALG para facilitar la traducción entre los nombres y las direcciones en el DNS.

- Doble NAT: en este caso, el NAT traduce tanto la dirección fuente como la dirección destino. Esto contrasta con el NAT tradicional y el NAT bidireccional, donde sólo una de las direcciones se traduce. Estos tipos de NATs resultan necesarios cuando los dominios privados y externos colisionan. Por ejemplo, cuando un dominio utiliza un conjuto de direcciones globales que pertenecen a otra organización.

- NAT *"multihomed"*: los NATs anteriores poseen algunas limitaciones, por ejemplo, paquetes pertenecientes a la misma sesión deben encaminarse hacia el mismo NAT, lo que introduce un único punto de fallo en la red. Para asegurar que la conectividad se mantiene cuando el NAT falla, puede ser deseable utilizar *"multihoming"* junto con uno o varios NATs. En este caso, los NATs deberán intercambiar información de estado.

El mecanismo NAT-PT permite que los nodos IPv6 se comuniquen con nodos IPv4 de forma transparente utilizando una única dirección IPv4. En dicho mecanismo se asigna una dirección IPv4 junto con un puerto cuando el NAT-PT identifica el comienzo de una sesión. Las políticas de configuración determinan qué tipos de sesiones se permiten y en qué dirección se pueden establecer (*"inbound"* o desde fuera hacia adentro y *"outbound"* o desde dentro hacia fuera del NAT). Las sesiones *"inbound"* estan restringidas a un servidor por servicio, asignado mediante un mapeo estático de puertos. En el dibujo de la figura 2.12 se puede observar un esquema típico de configuración del mecanismo:

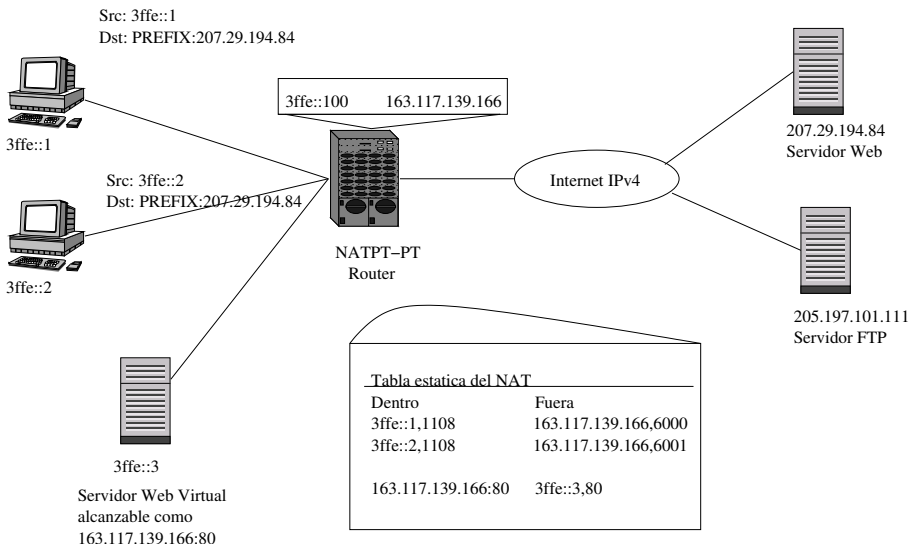

Figura 2.12: Un esquema típico de NAT-PT.

Funcionamiento.

El mecanismo de transición NAT-PT funciona del siguiente modo:

De IPv6 a IPv4: se construye una dirección IPv6 "falsa" (*"fake address"*) utilizando la dirección IPv4 del destino y anteponiendo el prefijo del NAT. Dicho prefijo de 96 bits de longitud se debe configurar en el NAT-PT para poder establecer este tipo de comunicaciones. Además, las direcciones IPv6 así construidas debe encaminarse hacia el servidor que implementa el mecanismo de NAT-PT. De esta forma, el NAT-PT examina los paquetes y cuando detecta un paquete con una dirección destino "falsa", traduce dicho paquete a IPv4, poniendo como destino la dirección IPv4 que lleva la dirección "falsa" embebida y como dirección origen la dirección IPv4 del NAT. Los puertos se mapean según las reglas que se hayan definido dentro del NAT y además se establece dinámicamente un mapeo inverso, de forma que los paquetes de respuesta puedan ser traducidos y reenviados al nodo IPv6 origen.

De IPv4 a IPv6: el mecanismo NAT-PT funciona como un NAT bidireccional[8]. De esta forma, la traducción se realiza de forma semejante al caso anterior, generando un paquete IPv6 con dirección origen la dirección IPv6 "falsa" que contiene en su interior la dirección IPv4 del nodo que inicia la comunicación, y como dirección IPv6 destino la dirección IPv6 de la máquina que se haya establecido en el mapeo. Dicho mapeo se establece interaccionando con el DNS-ALG.

Una vez conocido el funcionamiento del NAT-PT, conviene describir brevemente el funcionamiento del traductor DNS-ALG para poder utilizar el mecanismo de resolución de nombres junto con el mecanismo de transición. En la siguiente discusión se van a obviar los registros de DNS de tipo "A6", ya que su uso se encuentra discontinuado, aunque el mecanismo que se describe a continuación funcionaría igualmente con este tipo de registros.

El mecanismo DNS-ALG junto con NAT-PT permite traducir mensajes de DNS entre las islas IPv4 e IPv6, de la siguiente forma:

Un nodo IPv4 solicita la resolución de un nombre de una máquina IPv6: se realiza una consulta tipo "A" que se dirige hacia el servidor DNS de la red IPv6. Si consideramos que el mecanismo DNS-ALG reside en el *"border router"* entre las redes IPv4 e IPv6[9], el paquete es capturado por el DNS-ALG el cual modifica la consulta de tipo "A" por otra de tipo "AAAA". Una vez que se ha alcanzado el DNSv6 y se ha obtenido la dirección IPv6 de la máquina por la que se preguntaba, el DNS-ALG intercepta de nuevo el paquete de respuesta y realiza lo siguiente:

- Traduce el registro "AAAA" en un registro tipo "A".
- Reemplaza la dirección IPv6 resuelta por el DNSv6 por una dirección IPv4 del *"pool"* de direcciones del NAT. Debe ser una dirección del *"pool"* de direcciones disponibles porque no se puede establecer una correspondencia de puertos, ya que todavía no se sabe contra qué puerto iniciará la comunicación el nodo originador.

[8] Aunque la implementación de KAME probada en el laboratorio se comporta como un NAT tradicional, teniendo que definirse mapeos manuales para las sesiones *"inbound"*.

[9] Con esto se quiere significar que el paquete tiene que pasar obligatoriamente por dicho sitio, obviamente, para que pueda ser traducido.

Si es un inicio de comunicación, el NAT puede establecer en este momento el mapeo entre dichas máquinas.

En interesante resaltar que el mecanismo para mapear direcciones para sesiones de entrada, tal como se ha descrito previamente, es susceptible de ataques de denegación de servicio ya que se pueden realizar múltiples consultas de nodos residentes en la subred IPv6 causando que el DNS-ALG acabe por mapear todas las direcciones IPv4 del *"pool"* de direcciones del NAT y acabar bloqueando de esta forma el establecimiento de sesiones de entrada "legítimas". Soluciones a este y otros problemas se describen en [**?**].

Un nodo IPv6 solicita la resolución de un nombre de una máquina IPv4: en este caso, la máquina origen realiza una consulta tipo "AAAA", que al alcanzar el DNS-ALG, es reenviada tal cual junto con una consulta añadida de tipo "A" para el mismo nodo. Si existe un registro tipo "AAAA" para el destino, se devolverá al DNS-ALG el cual simplemente lo reenviará sin realizar ningún cambio a la máquina que lo solicitó. Si, por contra, existe un registro tipo "A" para el nodo destino, la respuesta también alcanzará al DNS-ALG. Entonces, se traduce la respuesta pasando del tipo "A" al tipo "AAAA" y añadiendo el prefijo del NAT a la dirección IPv4 de respuesta para obtener una dirección IPv6 "falsa", la cual se utilizará por el nodo originador IPv6 para iniciar la comunicación con el máquina IPv4.

Un caso a tener en cuenta en el mecanismo descrito anteriormente es cómo el DNSv6 de la subred IPv6 establece una comunicación con el dominio IPv4 que constituye el resto del mundo. Se recuerda que se supone que los DNS se encuentran en sus respectivas islas IPv4 o IPv6 y ni siquiera tienen que estar ejecutándose en máquinas *"dual-stack"*. Para solucionar este problema, por ejemplo, el DNSv4 externo podría apuntar a una dirección IPv4 que forme parte del *"pool"* de direcciones IPv4 disponibles para el NAT. El NAT mantendría una correspondencia uno-a-uno entre esta dirección IPv4 y la dirección IPv6 del servidor DNSv6 interno.

En el otro sentido, el servidor DNSv6 apuntaría a una dirección IPv6 construida mediante el prefijo del NAT y la dirección IPv4 del DNSv4 externo. Este mecanismo podría extenderse de forma semejante para utilizar servidores secundarios de DNS.

Clasificación del mecanismo.

- Ambito de aplicación: dominio.
- Requisitos de IPv4: ninguno.
- Requisitos de las direcciones IPv4: >= 1 por sitio.
- Requisitos de IPv6: ninguno.
- Requisitos de las direcciones IPv6: ninguno.
- Requisitos de máquinas: pila IPv6.
- Requisitos de routers: ninguno, pero el router puede ser además el NAT-PT.
- Impacto del NAT: pueden necesitarse dos o más niveles de traducción.

- Otros requisitos: DNS dentro de la red IPv6, ALG para aplicaciones que utilicen explicitamente direcciones.

2.4.4 BIS.

El mecanismo de transición denominado *Bump-In-The-Stack*, se encuentra descrito en [**?**], y permite que aplicaciones que no entienden IPv6 y que se están ejecutando en máquinas IPv4 (o máquinas con doble pila) se comuniquen con máquinas IPv6-only.

Este mecanismo puede resultar útil para aquellas aplicaciones que no se hayan migrado a IPv6 o que no puedan migrarse[10] y que deseen establecer comunicación con máquinas IPv6-only.

Introducción.

El mecanismo consiste en la inserción de determinados módulos dentro del kernel, módulos que actúan en un nivel intermedio entre la capa de red y la capa de interfaz modificando los paquetes que fluyen en uno y otro sentido.

Cuando una aplicación IPv4 trata de comunicarse con una aplicación IPv6, se realiza un mapeo entre una dirección IPv6 y una dirección IPv4. Posteriormente, cuando la aplicación trata de enviar un paquete IPv4, el mecanismo entra en acción y traduce dicho paquete a IPv6. De esta forma, parece como si tanto la máquina como la aplicación estuvieran preparadas para ejecutar aplicaciones en entornos *"dual-stack"*.

En la figura 2.13 se observan los bloques estructurales que constituyen el mecanismo:

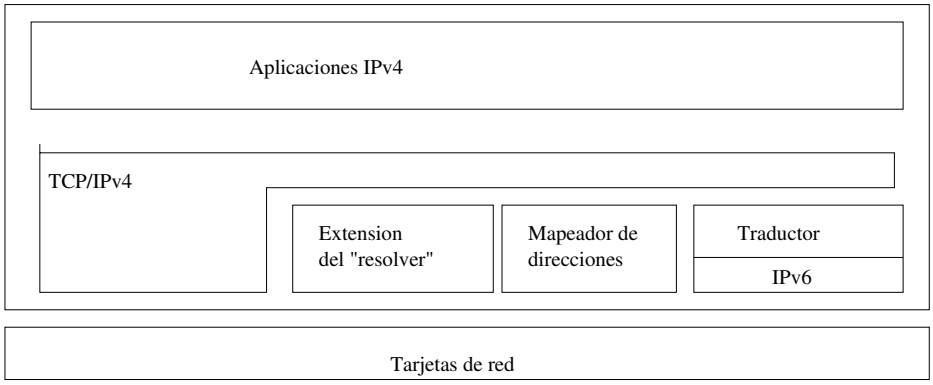

Figura 2.13: Diagrama de bloques de BIS.

A continuación se describe cada uno de los elementos:

- Traductor: traduce entre IPv4 e IPv6 utilizando el mencanismo de transición denominado SIIT[**?**]. Este módulo recibe paquetes IPv4 de las aplicaciones, convierte el paquete IPv4 en un paquete IPv6 y fragmenta el paquete, debido a que la cabecera IPv6 es normalmente 20 bytes más grande que la cabecera IPv4, y envía el nuevo paquete IPv6 por la red. De forma semejante, cuando recibe paquetes IPv6 provinientes de redes IPv6,

[10] Por no tener el código fuente, por ejemplo.

traduce el paquete de IPv6 a IPv4, sin necesidad de fragmentar el paquete IPv4 así obtenido.

- Extensión del *"resolver"*: es el módulo encargado de devolver una respuesta adecuada a las peticiones de DNS de las aplicaciones IPv4. Las aplicaciones típicamente envían una consulta de tipo "A" al servidor de nombres. Este módulo captura dicha consulta y crea otra consulta utilizando registros tipo "AAAA". Si se resuelve la consulta de tipo "A", se devuelve a la aplicación sin más. En este caso, como existe comunicación IPv4, no se necesita utilizar el módulo de traducción. Si sólo se encuentra disponible el registro de tipo "AAAA", se solicita al módulo mapeador que asigne una dirección IPv4 a la correspondiente dirección IPv6 que se acaba de recibir. Después crea un registro de tipo "A" con la dirección IPv4 asignada y devuelve la respuesta a la aplicación. Como se observa, el mecanismo resulta idéntico al funcionamiento del DNS-ALG descrito en [**?**].

- Mapeador de direcciones: es el módulo encargado de mantener el *"pool"* de direcciones IPv4. Normalmente se utilizan direcciones IPv4 privadas, ya que estas direcciones se utilizan únicamente dentro de la propia máquina que implementa el mecanismo. Este módulo también mantiene una tabla con las asignaciones efectuadas hasta el momento. Dicha tabla se debe inicializar con el mapeo entre la verdadera dirección IPv4 de la máquina y la verdadera dirección IPv6, para que el mecanismo sea capaz de traducir la dirección IPv4 fuente cuando envíe paquetes IPv4 a nodos IPv6 y para que se pueda traducir también de forma semejante la dirección IPv6 destino de un paquete que va dirigido a una aplicación IPv4 de dicha máquina. El módulo entra en juego cuando el *"resolver"* solicita la asignación de una dirección IPv4 a una dirección IPv6 determinada.

Funcionamiento.

A modo de ejemplo, se va a comentar paso a paso las acciones que se realizan cuando la máquina que implementa este mecanismo actúa como originadora de la comunicación y también cuando actúa como receptora.

En el primer caso, suponiendo que una aplicación IPv4 trata de enviar un paquete a una aplicación que se encuentra en una máquina IPv6, se realizarían los siguientes pasos:

1. La aplicación realiza una consulta contra el DNS para poder iniciar la comunicación con el extremo remoto.

2. Como sólo se resuelve la consulta de tipo "AAAA" realizada por el módulo *"resolver"*, éste solicita al módulo mapeador que establezca una correspondencia entre la dirección IPv6 destino y una dirección IPv4 del *"pool"* de direcciones disponibles.

3. El resolver crea un paquete de respuesta para la aplicación de tipo "A" con la dirección IPv4 recién asignada.

4. La aplicación ve al destino como una dirección IPv4 y comienza con el envío de paquetes.

5. Los paquetes IPv4 son capturados por el traductor, que realiza la traducción adecuada utilizando los mapeos definidos en el módulo mapeador para traducir la dirección IPv4 destino y la dirección IPv4 fuente. La dirección IPv4 fuente se traduce gracias a que previamente la tabla de correspondencias ha sido inicializada con un mapeo entre la dirección IPv4 y la dirección IPv6 de la máquina.

6. El traductor envía el paquete IPv6 construido por el interfaz apropiado.

7. El paquete alcanza el destino IPv6-only y como respuesta dicho destino envía un paquete IPv6 hacia el nodo originador de la comunicación.

8. El paquete IPv6 alcanza el nodo originador.

9. El traductor traduce el paquete IPv6 utilizando la tabla de asignaciones del módulo mapeador y entrega el paquete IPv4 así construido a la aplicación final.

De una forma semejante, cuando la máquina que implementa el mecanismo actúa como receptora de la comunicación, los pasos que se producen son los siguientes:

1. Un paquete IPv6 alcanza el nodo que implementa el mecanismo.

2. El traductor captura el paquete y trata de traducirlo. El traductor utiliza el módulo mapeador para obtener la correspondencia entre la dirección IPv6 destino y la dirección IPv4. Esta correspondencia ha tenido que ser fijada con anterioridad. Pero no existe ningún mapeo definido para la dirección IPv6 origen del paquete, de forma que el módulo mapeador selecciona una dirección IPv4 del *"pool"* de direcciones disponibles y establece el mapeo, devolviendo al módulo traductor la dirección IPv4 seleccionada. De esta forma, el módulo traductor puede traducir el paquete IPv6 y construir un paquete IPv4 equivalente.

3. El traductor entrega el paquete IPv4 así creado a la aplicación.

4. La aplicación como respuesta envía un paquete IPv4 al nodo originador de la comunicación.

5. El resto de pasos son idénticos a los descritos para el primer caso.

Resumiendo, la idea básica de este mecanismo de transición consiste en que cuando una aplicación IPv4 necesita comunicarse con una máquina IPv6-only, la dirección IPv6 de dicha máquina se mapea en una dirección IPv4 del *"pool"* de direcciones locales. El paquete IPv4 generado se traduce en un paquete IPv6 utilizando el algoritmo SIIT y se envía por el interfaz de red adecuado.

Se puede ver el mecanismo *"Bump-In-The-Stack"* como una implementación específica de NAT-PT dentro de la pila IP de una máquina. Un mecanismo similar se puede implementar a nivel de librería en una máquina con doble pila de protocolos.

Clasificación del mecanismo.

- Ambito de aplicación: host.
- Requisitos de IPv4: ninguno.
- Requisitos de las direcciones IPv4: espacio privado de direcciones por máquina.
- Requisitos de IPv6: ninguno.
- Requisitos de las direcciones IPv6: ninguno.

- Requisitos de hosts: doble pila más extensiones.
- Requisitos de routers: ninguno.
- Impacto del NAT: La presencia de un NAT no tendrá ningún efecto en el tráfico IPv6, puesto que la dirección IPv4 se utiliza de forma interna.
- Otros requisitos: ALG para aplicaciones que utilicen direcciones de una forma literal.

2.4.5 TRT.

El mecanismo denominado *"Transport Relay Translator"*, especificado en [**?**] y denominado también por sus siglas inglesas TRT, permite la comunicación directa entre aplicaciones IPv6 y aplicaciones IPv4. Este mecanismo, a diferencia del NAT-PT, actúa a nivel de transporte, y a diferencia de BIS, actúa como una pasarela entre ambos protocolos, estableciendo por un lado la conexión IPv4 y por otro la conexión IPv6, y reenviando los paquetes entre ambas conexiones. En el momento de escribir este trabajo, no se conoce ninguna implementación para paquetes UDP, es decir, todas las implementaciones existentes funcionan únicamente con el protocolo de transporte TCP.

Introducción.

Cuando se intenta implantar una red IPv6-only, se debe intentar mantener el acceso a los recursos IPv4 de redes externas, tales como servidores web IPv4-only. Para resolver el problema que surge durante la coexistencia de ambos protocolos de red, se puede utilizar el mecanismo de pasarela de traducción a nivel de transporte.

Este mecanismo posee algunas ventajas sobre otros mecanismos de transición. Los traductores de cabeceras IPv4/IPv6 sufren restricciones en cuanto a la máxima MTU que pueden utilizar y también poseen problemas relacionados con la fragmentación, de los que TRT no tiene que preocuparse. Por otro lado, TRT posee las siguientes desventajas:

- TRT soporta únicamente tráfico bidireccional. Los traductores IPv6-a-IPv4 pueden llegar a soportar otros casos, como datagramas unidireccionales (conexiones iniciadas desde el exterior, por ejemplo.)
- TRT necesita un sistema que almacene el estado entre los nodos que establecen la comunicación, al igual que los sistemas NAT. Mientras que resulta posible situar varios sistemas TRT dentro de un mismo sitio, una conexión a nivel de transporte se establece contra un único sistema TRT. El sistema TRT se considera como un único punto de fallo, de igual forma que el NAT. Otros mecanismos, como SIIT, utilizan sistemas de traducción sin estado los cuales pueden evitar dicho punto de fallo.
- Se necesita código especial para reenviar protocolos o aplicaciones que no son compatibles con NAT [11], por ejemplo IPsec, ftp, telnet, etc.

[11] *NAT-unfriendly*

El sistema de reenvío a nivel TCP necesita capturar paquetes que no van destinados hacia su propia dirección. Normalmente, esto se consigue mediante una modificación del mecanismo de encaminamiento. A nivel UDP, si existe alguna forma de reconocer el tráfico de entrada y de salida de un mismo flujo, se puede implementar un mecanismo de pasarela similar al realizado para TCP. Por ejemplo, una implementación podría reconocer pares de tráfico UDP de forma semejante a como los reconoce una implementación de NAT, grabando pares de dirección/puerto en una tabla y utilizando *"timeouts"*.

Funcionamiento.

En [**?**] se describe el funcionamiento del mecanismo desde IPv6 hacia IPv4. En el esquema número 2.14 se presenta un dibujo que va a servir para explicar su funcionamiento.

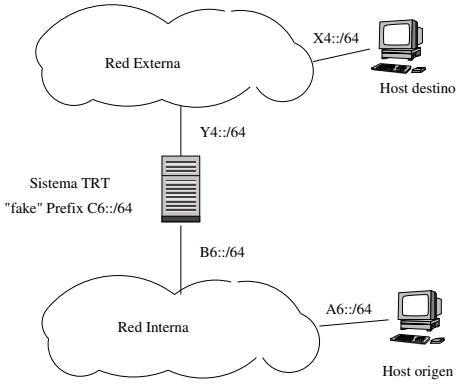

Figura 2.14: Esquema de funcionamiento del TRT.

Según este esquema, cuando la máquina que inicia la comunicación (con dirección IPv6 representada como A6::/64) desea establecer una conexion con la máquina destino (con dirección IPv4 X4::/64), necesita establecer una conexión TCP sobre IPv6 con C6::X4/64. Por ejemplo, si C6::/64 es igual a "fec0:0:0:1::/64" y X4 es igual a "10.1.1.1", la dirección de destino que debe usarse es "fec0:0:0:1::10.1.1.1". El paquete se encaminará hacia el sistema de TRT. El TRT acepta la conexión IPv6 y examina los últimos 32 bits de la dirección de destino para obtener la dirección IPv4 del verdadero destinatario de la conexión TCP. Entonces, el TRT establece una conexión TCP sobre IPv4 con dicho destino, y reenvia el tráfico entre ambas conexiones. Por consiguiente, se mantienen dos conexiones TCP. Una sobre IPv6 y la otra sobre IPv4.

Resumiendo, el sistema de TRT utiliza el reenvio de paquetes a nivel de transporte para traducir el tráfico IPv6 en tráfico IPv4. De esta forma se obtienen las siguientes ventajas:

- La implementación del sistema TRT se realiza de una forma sencilla.
- El mecanismo de "descubrimiento de la MTU"[**?**] en IPv6 puede realizarse de forma independiente de la MTU para IPv4.

Incluso simplicando así las cosas, el sistema TRT puede tratar con la mayoría de las aplicaciones utilizadas en el día a día (http, smtp, ssh y otros muchos protocolos).

Para protocolos no compatibles con NAT, el sistema TRT puede necesitar modificar datos a nivel de aplicación, exactamente igual que cualquier otro traductor. Además, como el sistema de TRT reside a nivel de transporte, no es posible traducir protocolos de transporte desconocidos por el TRT.

Un factor a tener en cuenta es que normalmente los usuarios no quieren traducir tráfico de DNS utilizando el sistema de TRT por las siguientes razones:

- Transporte: es mucho más fácil proporcionar un servicio de DNS, accesible por IPv6, que traducir consultas/respuestas de DNS a través del sistema de TRT. Si alguien intenta pedir al TRT que traduzca paquetes de DNS, debería poner en "/etc/resolv.conf" la dirección C6::X/64 (donde C6 es el prefijo reservado para el TRT y X es la dirección IPv4 del servidor de DNS, como se ha explicado con anterioridad). Esta configuración es más complicada de lo que sería normalmente deseable.
- *Payload*: algunos usuarios pueden querer atravesar el sistema de TRT sin modificar el tipo de consulta/respuesta, mientra que otros usuarios pueden requerir el uso de un ALG para traducir consultas de tipo AAAA en consultas de tipo A y viceversa. Esto depende de la configuración e instalación de cada dominio.

En vez de ello, tiene más sentido ejecutar un servidor de DNS, o un servidor de DNS especial que ayude al sistema de TRT, dentro de la propia red IPv6.

Clasificación del mecanismo.

- Ambito de aplicación: dominio.
- Requisitos de IPv4: ninguno.
- Requisitos de las direcciones IPv4: 1 por sitio.
- Requisitos de IPv6: ninguno.
- Requisitos de las direcciones IPv6: un prefijo para encaminar los paquetes hacia el traductor.
- Requisitos de máquinas: ninguno.
- Requisitos de routers: ninguno, pero se necesita que una determinada máquina actue como TRT.
- Impacto del NAT: depende de la aplicación.
- Otros requisitos: servidores de DNS que puedan mapear direcciones IPv4 a direcciones IPv6.

2.4.6 SOCKS64.

La pasarela SOCKS, descrita en [**?**], es un sistema que acepta conexiones SOCKS[**?**] desde máquinas IPv4 y las reenvía a los *hosts* destinatarios (IPv4 o IPv6). SOCKS proporciona un mecanismo sencillo (especialmente para aquellas aplicaciones "socksificadas", es decir, aquellas aplicaciones que utilizan clientes conscientes del mecanismo SOCKS y de la existencia de un servidor de SOCKS) para que máquinas IPv4 se comuniquen con máquinas IPv6 y viceversa, sin necesidad de realizar modificaciones en el DNS o en el mapeo de las direcciones. A continuación se describe brevemente el mecanismo y su funcionamiento.

Funcionamiento.

En la figura 2.15 se observan las distintas capas que constituyen el mecanismo de SOCKS:

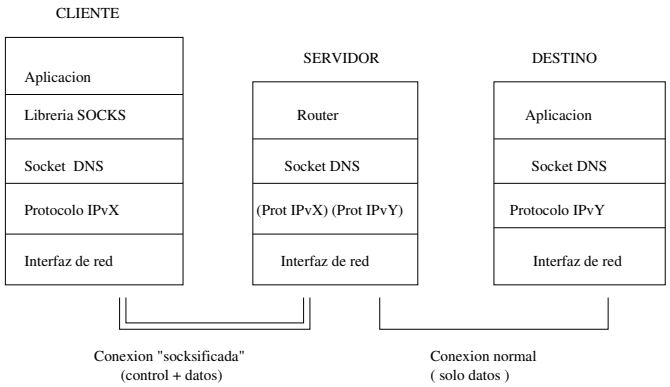

Figura 2.15: Diagrama de bloques del mecanismo SOCKS.

En la figura, el cliente inicia la comunicación. Existen dos elementos funcionales nuevos:

- Librería de SOCKS: se localiza entre la capa de aplicación y la capa de sockets, y reemplaza a las funciones de manejo de sockets y de resolución de nombres del DNS, por ejemplo *gethostbyname()*, *getaddrinfo()*, etc. Cada aplicación "socksificada" posee su propia librería de SOCKS.
- Servidor: es router que se instala entre el cliente y la máquina destino. Se trata de un servidor de SOCKS que actúa de *"relay"* entre el cliente y el destino. Cuando la librería de SOCKS solicita una pasarela al servidor, el servidor crea un proceso o *"thread"* nuevo para realizar la función de *"relay"*.

En la figura, se pueden dar las siguientes combinaciones de protocolos de red:

- IPv4-IPv4: SOCKS normal u homogéneo.
- IPv4-IPv6: SOCKS heterogéneo.
- IPv6-IPv4: SOCKS heterogéneo.
- IPv6-IPv6: SOCKS homogéneo.

Como la librería de SOCKS se comunica con el servidor de SOCKS utilizando el protocolo de SOCKS, definido en [**?**], se trata de una conexión especial denominada "conexión socksificada" que se utiliza para transmitir información de control[12] además de los datos.

Por otro lado, la conexión entre el servidor de SOCKS y el destino es una conexión normal, que no resulta modificada. Una aplicación servidora que se ejecute en el nodo destino no percibe la existencia del verdadero cliente, sino que percibe como nodo cliente al servidor de SOCKS.

Para establer cualquier comunicación, se necesita obtener la dirección IP destino. Sin embargo, en entornos heterogéneos como en el caso de dominios IPv4 *"versus"* IPv6, es teóricamente imposible obtener una información correcta, puesto que una aplicación IPv4 no puede tratar con direcciones IPv6[13]. La aplicación IPv4 no puede almacenar una dirección IPv6 en los cuatro bytes que tiene reservados. Se trata de un problema crítico causado por diferencias en el tamaño de las direcciones.

Para solventar este problema, existe una característica del mecanismo SOCKS64 que permite realizar una "delegación" de la resolución de nombres del DNS. Como el servidor de SOCKS es una máquina con doble pila de protocolos, puede resolver tanto consultas IPv4 como consultas IPv6. A continuación se explica el funcionamiento con un ejemplo:

1. Una aplicación situada en la máquina cliente trata de obtener la dirección IP del nodo destino. En este momento, el nombre se pasa como un argumento a la función encargada de la resolución (por ejemplo, *gethostbyname*), lo cual es interceptado por la librería SOCKS.

2. Como la librería SOCKS reemplaza a las funciones del *"resolver"*, la verdadera función no se llega a ejecutar. En vez de ello, el nombre se registra en una tabla de correspondencias dentro de la librería y se selecciona una dirección IP "falsa" para retornar a la aplicación, realizando el mapeo entre dicha dirección y el nombre buscado.

3. La aplicación recibe la dirección IP "falsa" y prepara un socket. La dirección IP "falsa" se utiliza para crear el socket, y posteriormente el socket se pasa como parámetro a las funciones encargadas de enviar paquetes.

4. Como la librería SOCKS ha reemplazado a las funciones relacionadas con el envío de paquetes a través de sockets, la verdadera función no se llama. En vez de ello, la dirección IP del socket se chequea. Si la dirección pertenece al espacio de direcciones especial reservado para las direcciones "falsas", se busca en la tabla de correspondencias para encontrar el correspondiente nombre de la máquina destino.

5. El nombre se envía al servidor SOCKS.

6. El servidor SOCKS obtiene la dirección IP verdadera de la máquina destino y crea el socket utilizando dicha información. Después, establece comunicación con el destino.

El problema que surge con el mecanismo descrito es que los fallos en el proceso de resolución del nombre se detectan de forma incorrecta en el nodo origen. Estos fallos se ven en el cliente como fallos de establecimiento de la conexión.

[12] Como información sobre localización del destino.
[13] ver [**?**], capítulo 3. "DNS Name Resolving Procedure".

Se pueden anidar varios servidores SOCKS, aunque el proceso se ralentiza y no existe límite en cuanto al número de pasarelas que se pueden anidar.

Clasificación del mecanismo.

- Ambito de aplicación: dominio.
- Requisitos de IPv4: ninguno.
- Requisitos de las direcciones IPv4: 1 por pasarela o servidor de SOCKS, también se hace uso de direcciones IPv4 "falsas", que no se utilizan en comunicaciones reales, tales como "0.0.0.x".
- Requisitos de IPv6: >= 1 por sitio.
- Requisitos de las direcciones IPv6: ninguno.
- Requisitos de máquinas: los clientes deberían estar *"socksificados"*.
- Requisitos de routers: ninguno.
- Impacto del NAT: el servidor de SOCKS y el NAT necesitan cooperar.
- Otros requisitos: el servidor de SOCKS debe ser *"dual stack"*.

2.4.7 BIA.

El mecanismo denominado *"Bump in the API"*, descrito en [**?**], permite que máquinas *"dual-stack"* se comuniquen con máquinas IPv6 utilizando aplicaciones IPv4. El objetivo de este mecanismo es equivalente al mecanismo BIS[**?**], pero, en este caso, el objetivo se alcanza sin necesidad de traducir cabeceras IP entre ambos protocolos. Mientras que BIS se utilizan en sistemas que no poseen pila IPv6, BIA se utiliza en sistemas *"dual-stack"*, pero en los cuales existen aplicaciones que no se pueden migrar a IPv6.

Introducción.

El mecanismo BIA introduce un traductor entre el interfaz de programación de aplicaciones y el módulo TCP/IP, de tal forma que traduce las APIs de IPv4 en IPv6 y viceversa.

Cuando las aplicaciones IPv4 intentan comunicarse con aplicaciones que se ejecutan en máquinas IPv6, este traductor detecta las llamadas a las funciones de socket y llama a las funciones IPv6 correspondientes. Para poder soportar este estilo de comunicación, se necesita asignar un conjunto de direcciones IPv4.

En la figura 2.16 se observa un esquema de bloques del mecanismo:

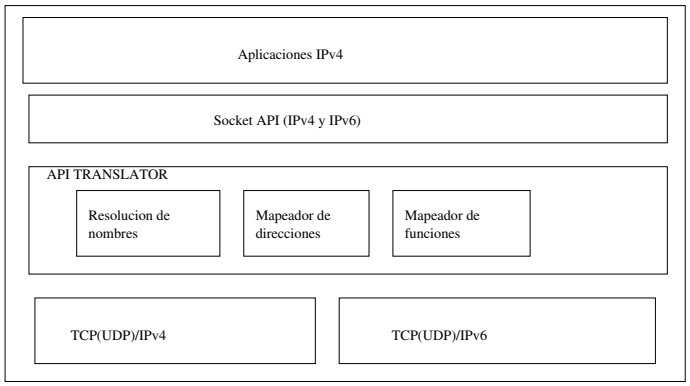

Figura 2.16: Diagrama de bloques del mecanismo BIA.

A continuación se describe cada uno de los elementos nuevos:

- Mapeador de funciones: módulo que traduce entre las funciones de las interfaces de programación de aplicaciones para IPv4 e IPv6.
- Módulo de resolución de nombres: módulo encargado de devolver una respuesta adecuada a las preguntas de las aplicaciones IPv4. Cuando una aplicación IPv4 envía una consulta de tipo "A" a un servidor de nombres, se detecta la consulta y se crea además una consulta de tipo "AAAA". Si sólo se recibe respuesta de la consulta "AAAA", entonces la máquina destino es IPv6 y se solicita al módulo mapeador de direcciones que asigne una dirección IPv4, dirección que se devuelve como respuesta al paquete de tipo "A" que la aplicación envió originalmente.
- Mapeador de direcciones: se trata de un módulo que mantiene internamente una tabla de pares de direcciones IPv4 e IPv6. Las direcciones IPv4 se asignan de direcciones IPv4 no permitidas en el enlace, como por ejemplo del rango "0.0.0.0" hasta "0.0.0.255".

Funcionamiento.

A continuación se va a describir paso a paso el funcionamiento del mecanismo haciendo uso de un ejemplo.

1. Cuando una aplicación IPv4 envía una consulta al DNS, se intercepta, se crea una consulta nueva de tipo "AAAA", y en el caso de que sólo se obtenga respuesta de la consulta de tipo "AAAA", se realiza una asignación entre la dirección IPv6 y la dirección IPv4 extraida del conjunto de direcciones disponibles por el mapeador de direcciones. Finalmente se devuelve la dirección IPv4 utilizada a la aplicación, como respuesta a su consulta.
2. La aplicación crea un socket IPv4 y comienza a enviar paquetes.
3. Las llamadas a las funciones del API son interceptadas por el módulo mapeador de funciones. Si la llamada es una llamada del API de IPv4, necesita obtener una dirección IPv6 para llamar a la función equivalente en IPv6, para lo cual solicita una dirección al módulo mapeador de direciones. El módulo mapeador de direcciones selecciona una

dirección IPv4 de su tabla y devuelve la correspondiente dirección IPv6. Finalmente, la dirección IPv6 se utiliza para llamar a la función correspondiente.

4. El paquete alcanza el destino.

5. La máquina destino procesa el paquete y envía un paquete IPv6 de respuesta.

6. El paquete IPv6 llega a la máquina destino y es detectado por el módulo mapeador de funciones.

7. El módulo mapeador de funciones necesita una dirección IPv4 para invocar a la correspondiente función IPv4 encargada de establecer la comunicación con la aplicación. Esta dirección la proporciona el módulo mapeador de direcciones. Finalmente, el módulo mapeador de funciones invoca a la correspondiente función y entrega los datos a la aplicación.

Este mecanismo permite la utilización de mecanismos de seguridad a nivel de la capa de red, puesto que la traducción se realiza por encima de ella. Además, como el interfaz de programación de aplicaciones para IPv6 soporta nuevas características y extensiones propias del protocolo IPv6, resulta complicado traducir dichas funciones utilizando el API proporcionado por IPv4. De esta forma, algunos paquetes IPv6 con cabeceras de extensión pueden no ser traducidos y ser descartados.

Clasificación del mecanismo.

- Ambito de aplicación: máquina.
- Requisitos de IPv4: ninguno.
- Requisitos de las direcciones IPv4: espacio privado de direcciones por máquina.
- Requisitos de IPv6: ninguno.
- Requisitos de las direcciones IPv6: ninguno.
- Requisitos de máquinas: doble pila más extensiones.
- Requisitos de routers: ninguno.
- Impacto del NAT: el mecanismo no resulta afectado por la presencia de NATs.
- Otros requisitos: ALG para aplicaciones que utilicen direcciones de una forma literal.

2.5 Comentarios sobre los distintos mecanismos de transición.

En este apartado se van a comparar los distintos mecanismos de transición que se han descrito.

Los túneles configurados son los más utilizados en el momento de escribir este trabajo para obtener una conectividad IPv6 de una forma estable. No obstante, el túnel necesita de una

configuración manual en ambos extremos, y se debe solicitar un subrango de direcciones IPv6, independientemente de la configuración del túnel. El mecanismo de *tunnel broker* resuelve el problema de configuración en un extremo, pero presenta problemas de escalabilidad.

En segundo lugar podría colocarse a los túneles 6to4, por su simplicidad de configuración y por la posibilidad de obtener no sólo una dirección IPv6 sino un prefijo completo a partir de una única dirección IPv4. También resultan útiles para obtener conectividad IPv6 a través de proveedores de servicio que sólamente ofrecen conectividad IPv4. Existen *"scripts"* para configurar de forma automática estos túneles a partir de la dirección IPv4 de una interfaz "ppp"[14].

Los túneles 6over4 están en decadencia, fundamentalmente debido a la necesidad de ofrecer servicio de multicast en IPv4 para que funcionen. el soporte de multicast en la red IPv4 no acaba de despegar por lo que se convierte en un requisito muy fuerte para la implantación de dicho mecanismo de transición.

Los túneles automáticos presentan algunos problemas de seguridad hasta el punto de que KAME ha decidido no implementarlos. Desde el punto de vista de la pila IPv6, una dirección compatible con IPv4 se considera una dirección IPv6 unicast como cualquier otra, salvo que el paquete será transmitido usando una encapsulación automática en IPv4. De esta forma, se pueden realizar varios ataques, uno de los cuales se describe en [**?**]:

- Supongamos que poseemos un servidor de DNS que se ejecuta en una máquina *"dual-stack"*. Esta máquina pertenece a la subred 10.1.1.0/24.
- Un atacante transmite un paquete UDP IPv6 al puerto de DNS, con dirección fuente "::10.1.1.255". No importa si este paquete se transmite por IPv6 o si se transmite de forma encapsulada.
- El paquete llegará a nuestro DNS y éste responderá con un paquete IPv6 copiando la dirección fuente en la dirección destino.
- Esta réplica será encapsulada de forma automática, de acuerdo a la especificación de los túneles automáticos. Como resultado, se transmitirá un paquete IPv4 dirigido a la subred "10.1.1.255".

No obstante, también existe el mismo problema con los túneles 6to4 y en general con cualquier mecanismo de transición que encapsule de forma automática utilizando la dirección IPv4 embebida dentro de la propia dirección IPv6. En [**?**] se comentan las distintas soluciones posibles y también problemas relacionados con el uso de direcciones mapeadas.

Respecto al mecanismo ISATAP, éste presenta un ámbito de aplicabilidad de dominio y necesita de una dirección IPv4 por máquina. Por otro lado, el mecanismo Teredo resulta más flexible, pudiéndose utilizar a nivel de dominio o a nivel global, pero tiene problemas con el filtrado de entrada de los routers y el anuncio de direcciones anycast. El propio autor recomienda el uso de este mecanismo sólamente cuando todo lo demás falle, como último recurso, fundamentalmente porque la sobrecarga de encapsular IPv6 dentro de UDP y el encaminamiento triangular que normalmente se produce lo hacen más ineficiente que si se utilizan otros mecanismos, como por ejemplo el 6to4. No obstante, Teredo resulta fundamental puesto que permite realizar la transición a través de los actualmente muy difundidos NATs.

[14] NetBSD y FreeBSD poseen dicho *"script"* en su distribución, escrito en Perl.

Respecto a los mecanismos de traducción, el mecanismo TRT resulta muy útil, pero presenta problemas en aquellas aplicaciones que no sean compatibles con NATs. Además, en el momento de escribir este trabajo, sólo se encuentra implementado para TCP y la implementación para UDP resulta más complicada como se comenta en [**?**].

El resto de mecanismos, en especial el mecanismo NAT-PT debe utilizarse sólamente para interaccionar entre máquinas IPv6-only y máquinas IPv4-only, debido a que presenta los mismos problemas que el tradicional NAT en IPv4 que se intenta evitar en IPv6, y además crea nuevos problemas, por lo que desde todos los ámbitos se está recomendando que se debería buscar una solución basada en máquinas *"dual-stack"* y sólo utilizar mecanismos de traducción como último recurso.

El mecanismo DSTM necesita hacer uso de DHCPv6 y de túneles IPv4 en IPv6. Por lo demás, permite que máquinas IPv6 *dual-stack* que se encuentran en un dominio IPv6, es decir, sin ningún tipo de encaminamiento IPv4, se comuniquen con máquinas IPv4. Dichas máquinas IPv6 no necesitan poseer ninguna dirección IPv4, pero sí necesitan ser *"dual-stack"*. DSTM constituye una forma de evitar el uso de NATs para establecer comunicación con máquinas IPv4-only durante la fase de desarrollo inicial de las redes IPv6-only.

Los mecanismos de SOCKS64 y BIA están enfocados a la interoperabilidad de aplicaciones que no pueden ser migradas. Esto significa que su ámbito de aplicabilidad será muy reducido, puesto que dichos mecanismos no deben utilizarse como excusa para retardar la migración de aplicaciones, como específicamente se comenta en las correspondientes RFCs. No obstante, se prevé que existirán aplicaciones que por la pérdida del código fuente, o simplemente porque son aplicaciones propietarias, no podrán migrarse a corto plazo. Estos mecanismos permiten la interoperabilidad a nivel IP de dichas aplicaciones, por lo que resultan útiles y necesarios.

El mecanismo de BIS, muy parecido a BIA, resulta un poco extraño puesto que trata de interoperar aplicaciones que corren en máquinas IPv4 con las correspondientes en IPv6, pero para ello debe modificar la pila IP de tal forma que parece que al final se obtiene una implementación reducida de IPv6. Resulta más fácil convertir la máquina en un *host "dual-stack"* e instalar BIA que implantar este mecanismo.

La tecnología de NAT resulta bastante conocida, pero tiene problemas de escalado debido a que debe mantener el estado de las conexiones, y además presenta un único punto de fallo, a diferencia de SIIT, que al ser sin estado, resulta más robusto. El autor no conoce ninguna implementación de SIIT en el momento de realizar este trabajo. Por otro lado, la tecnica de traducción mediante NATs es justamente lo que IPv6 pretende evitar. A pesar de todo, el mecanismo NAT-PT puede demostrar su utilidad en entornos corporativos de ámbito restringido y el número de aplicaciones que no funcionan a través de NATs es hoy por hoy reducido, y para muchas de las apicaciones "rotas" se podrían implementar traducciones específicas o *"ad-hoc"*. No obstante, y a riesgo de resultar repetitivo, el autor quiere clara constancia de que el futuro de la internet de nueva generación pasa por la existencia de una red IP sin NATs.

Hablando de una forma más concreta, en lo que se refiere al mecanismo de transición NAT-PT, en [**?**] se han detectado varios problemas que presenta dicho mecanismo cuando se aplica junto con el mecanismo de traducción a nivel de apicación DNS-ALG:

- La comunicación entre un nodo IPv6-only y una máquina externa *"dual-stack"* utilizará el camino que pasa a través del NAT en vez de usar el camino IPv6 directo. Esto ocurre debido a que el DNS-ALG devolverá dos registros "AAAA", uno con la dirección IPv6 normal y otro con la dirección IPv4 traducida. Si la máquina origen aplica el algoritmo

de selección de dirección destino, seleccionará la dirección "traducida" debido a que coincide con el prefijo de su propia dirección.

- Las aplicaciones que se encuentran detrás del NAT-PT pueden pensar de forma errónea que están comunicándose con una aplicación/máquina IPv6, cuando realmente utilizan el camino "IPv6 + NAT-PT + IPv4". Esto puede ocasionar problemas dependiendo de la aplicación, puesto que se podrían intentar utilizar opciones de IPv6 que no podrían ser traducidas.

- Para las conexiones iniciadas desde el exterior, resulta extremadamente sencillo realizar ataques de denegación de servicio puesto que NAT-PT debe reservar una dirección del *"pool"* de direcciones por cada consulta contra el DNS. De esta forma, en cuestión de segundos se puede vaciar el *"pool"* de direcciones y además consumiendo muy poco ancho de banda.

Resumiendo y para finalizar este capítulo, podríamos preguntarnos cúal es el mejor mecanismo de transición, pero la respuesta a esta pregunta depende de tantos factores que no puede responderse directamente. Un buen administrador de redes debe conocer los "pros" y "contras" de cada mecanismo y tomar una decisión de acuerdo a la topología de su red y a sus necesidades de conectividad. Por supuesto, puede resultar útil y necesario utilizar más de un mecanismo de transición.

Capítulo 3

Configuración y pruebas de mecanismos de transición.

A continuación se describe en profundidad la configuración de los distintos mecanismos de transición que han sido probados dentro de la red del proyecto LONG. En este capítulo sólo se describen los pasos de configuración necesarios para su implantación en las distintas plataformas utilizadas. Se trata de presentar una guía de configuración.

No se han probado todos los mecanismos de transición, puesto que algunos todavía no están implementados, y algunos sólo se han probado en *FreeBSD*, por ser el sistema operativo más utilizado dentro de la red del proyecto LONG, marco en el que se han desarrollado la mayor parte de las pruebas realizadas en le presente proyecto fin de carrera.

3.1 Mecanismo 1: Túneles 6to4.

El mecanismo 6to4, definido en [**?**], porporciona una solución al problema de utilizar túneles configurados manualmente especificando un único prefijo de encaminamiento para cada usuario final (u organización) que utilice (al menos) un punto de acceso IPv4 a la red. Es importante recalcar el hecho de que cada organización, con al menos una dirección IPv4 pública, puede poseer un prefijo IPv6 globalmente direccionable gracias a este mecanismo.

3.1.1 Configuración.

En los sistemas operativos BSD, existe un *"script"* en Perl denominado "6to4" que permite configurar automáticamente este mecanismo, generando una dirección 6to4 a partir de la dirección IPv4 de la interfaz especificado y configurando la ruta por defecto de IPv6 contra uno de los varios *"relay routers"* existentes en la actualidad.

El script, básicamente realiza los siguientes pasos:

- Levantar el interzar "stf"[1]. Esto se realiza asignando a dicha interfaz una dirección 6to4 generada a partir de una dirección IPv4 válida, por ejemplo:

[1]Son las siglas de "six to four"

```
ifconfig stf0 inet6 add 2002:a375:8ba6:ffff::1 prefixlen 64
```

Dicho comando configura la interfaz 6to4 con una dirección 6to4 correspondiente a la dirección IPv4 "163.117.139.166" o en hexadecimal "a375:8ba6".

En linux, el dispositivo equivalente es "sit", utilizándose la siguiente línea de comando:

```
ifconfig sit0 up 2002:a375:8ba6:ffff::1/64
```

- Establecer la ruta por defecto. Para ello, se debe añadir una ruta /16 utilizando como siguiente salto un *"relay router"*. Por ejemplo, en *FreeBSD* el comando sería el siguiente:

```
route add -inet6 default 2002:: <ipv6-of-RR> prefixlen 16
```

De esta forma, el *"relay router"* es capaz de encaminar nuestro paquete hasta el destino deseado. Se alcanza al *"relay router"* a través de la interfaz 6to4. Los paquetes de respuesta, viajarán por la red IPv6 nativa hasta algún lugar en donde se deberá establecer un túnel con nosotros, puesto que la dirección IPv4 de destino de este túnel se puede extraer de nuestra propia dirección 6to4.

La configuración del mecanismo 6to4 en Windows 2000 resulta extremadamente sencilla, basta con ejecutar este comando:

```
6to4cfg -R <IPv4>
```

3.1.2 Pruebas funcionales.

Las pruebas han consistido en comprobar que existe conectividad IPv6 utilizando el siguiente ping, donde se alcanza la dirección IPv6 de "www.kame.net" utilizando el *"relay router"* de *Microsoft* denominado "6to4.ipv6.microsoft.com". Esta prueba se realiza utilizando *FreeBSD*.

La configuración de la interfaz túnel se muestra a continuación:

```
stf0: flags=1<UP> mtu 1280
  inet6 2002:a375:8ca6:1::1 prefixlen 16
```

Se ha añadido una ruta por defecto hacia el *"relay router"*, como se puede observar a continuación:

```
root@mira:/usr/home/jrh# netstat -rn | grep default
default  2002:836b:213c:1:e0:8f08:f020:8 UGSc       stf0
```

De esta forma, el ping6 muestra que existe conectividad IPv6:

```
root@mira:/usr/home/jrh# ping6 2002:a375:8ca6:1::1 www.kame.net
PING6(56=40+8+8 bytes) 2002:a375:8ca6:1::1 --> 2001:200:0:4819:210:f3ff:fe03:4d0
16 bytes from 2001:200:0:4819:210:f3ff:fe03:4d0, icmp_seq=0 hlim=59 time=317.83 ms
16 bytes from 2001:200:0:4819:210:f3ff:fe03:4d0, icmp_seq=1 hlim=59 time=317.63 ms
^C
--- apple.kame.net ping6 statistics ---
2 packets transmitted, 2 packets received, 0% packet loss
round-trip min/avg/max/std-dev = 317.629/317.730/317.830/0.100 ms
root@mira:/usr/home/jrh#
```

Mediante el siguiente traceroute6 se puede observar el camino de ida de los paquetes enviados:

```
root@mira:/usr/home/jrh# traceroute6 -s 2002:a375:8ca6:1::1 www.kame.net
traceroute6 to apple.kame.net (2001:200:0:4819:210:f3ff:fe03:4d0) \
         from 2002:a375:8ca6:1::1, 30 hops max, 12 byte packets
 1  2002:836b:213c:1:e0:8f08:f020:8  191.372 ms  190.49 ms  189.979 ms
 2  3ffe:c00:8023:c::2  192.064 ms  191.706 ms  191.794 ms
 3  fe-tu0.pao.ipv6.he.net  194.46 ms  193.324 ms  194.95 ms
 4  v6-paIX01.kddnet.ad.jp  315.638 ms  316.631 ms  315.276 ms
 5  2001:268:ff00:5::1  315.382 ms  315.298 ms  315.329 ms
 6  hitachi1.otemachi.wide.ad.jp  315.516 ms  315.317 ms  316.047 ms
 7  pc7.otemachi.wide.ad.jp  315.553 ms  314.83 ms  315.047 ms
 8  pc3.nezu.wide.ad.jp  316.553 ms  316.032 ms  316.767 ms
 9  pc7.nezu.wide.ad.jp  316.6 ms  317.131 ms  316.854 ms
10  pc3.yagami.wide.ad.jp  316.909 ms  317.375 ms  317.752 ms
11  gr2000.k2c.wide.ad.jp  318.624 ms  317.805 ms  318.439 ms
12  2001:200:0:4819:210:f3ff:fe03:4d0  318.476 ms  318.378 ms  318.463 ms
```

Esta misma prueba se ha realizado de forma equivalente para *Linux*.

3.1.3 Consideraciones finales.

Hay que tener en cuenta que el mecanismo sólo funciona si no existen impedimentos para la recepción de paquetes encapsulados. Por ejemplo, si nuestro sitio se encuentra dentro de una red protegida mediante un *"firewall"* que impide el acceso exterior de paquetes encapsulados, aunque nuestros paquetes van a poder alcanzar cualquier destino (bien sea encapsulándolos en el origen o bien encaminándolos hacia algún router 6to4 de nuestro entorno), cuando se intente enviar un paquete de respuesta, en algún momento se intentará realizar un túnel desde el exterior hacia la máquina origen, túnel que será filtrado por el *"firewall"*, impidiendo así la comunicación.

Los sistemas BSD poseen un script que automatiza el mecanismo de configuración de los túneles 6to4. Se basa en un archivo de configuración donde se indica la interfaz a partir del cual se obtiene la dirección IPv4 y el *"relay router"* que se desea utilizar, y el script se encarga de levantar la interfaz 6to4, generar la direccion 6to4 y añadir la ruta por defecto. Dicho script, escrito en *Perl*, se puede utilizar en sistemas *Linux* realizando mínimas modificaciones.

3.2 Mecanismo 2: NAT-PT.

El mecanismo de transición denominado NAT-PT se encuentra descrito en [**?**]. NAT-PT ofrece una solución directa basada en encaminamiento transparente y traducción de direcciones, permitiendo que un gran número de aplicaciones en entornos IPv4 e IPv6 puedan interoperar sin requerir ningún cambio. Este mecanismo está basado en el mecanismo SIIT[**?**], que a diferencia del que nos ocupa, es un algoritmo de traducción "sin estado".

3.2.1 Configuración.

El mecanismo de NAT-PT ha sido probado en el laboratorio bajo el sistema operativo *FreeBSD*, aunque el proyecto USAGI también proporciona una implementación muy parecida para máquinas *Linux*. En *FreeBSD* se debe compilar el kernel con las siguientes opciones:

```
# Network Address Translation - Protocol Translation (NAT-PT)
#options        NATPT           # IPv6 -> IPv4 translation.
#options        NATPT_NAT       # IPv4 -> IPv4 NAT.
                                # Valid if "options NATPT" is defined
```

Una vez arrancado el nuevo kernel, se dispone de la herramienta "natptconfig" que permite definir reglas de mapeo internas (*"inbound"*) y externas (*"outbound"*), hacia un determinado rango de puertos. La herramienta proporciona una gran flexibilidad y sencillez a la hora de manejar los mapeos. Hay que tener en cuenta que los desarrolladores del proyecto KAME han sido los implementadores del NAT para IPv4 en *FreeBSD*, por lo que poseen experiencia en el tema. Posee además una buena opción de depuración y una opción que presenta en cada momento los mapeos (tanto estáticos como dinámicos) establecidos, junto con el número de paquetes que los han utilizado. Para más información se remite al lector a la correspondiente página del manual, en concreto a "man natptconfig" para arrancar y depurar el mecanismo y tambiés resulta conveniente mirar "man natpt.conf" para la sintaxis de las reglas y mapeos.

3.2.2 Pruebas funcionales.

En este apartado se van a comentar las dos pruebas más importantes realizadas a través de NAT-PT, aunque no son las únicas pruebas que se han realizado. Otras pruebas de funcionalidad realizadas utilizando el mecanismo de transición NAT-PT han consistido en la conexión a un servidor de *Quake2* IPv4 desde clientes IPv6, y el acceso a otros servicios más simples como "ftp", "web" y "telnet".

NFS.

En este apartado se va a describir la utilización del mecanismo de transición NAT-PT para hacer uso de un sistema de ficheros exportado por una máquina IPv4 en máquinas IPv6. Por consiguiente, el esquema consiste en un servidor NFSv4, un sistema traductor NAT-PT y un (aunque podrían ser varios) cliente NFSv6.

El soporte de NFS para IPv6 resulta muy escaso, debido a que NFS se considera una herramienta de uso privado e intra-dominio, en donde se puede utilizar direcciones IPv4 privadas,

por lo que no resulta una aplicación crítica para la introducción de IPv6 en el mercado. No obstante, NFS para IPv6 está soportado a nivel cliente/servidor para la rama *FreeBSD*-5.0 y existe un parche para la rama *FreeBSD*-4.5, que es la versión que se ha utilizado para proporcionar NFS sobre IPv6 en la máquina (IPv6) cliente. El parche proporciona IPv6 para RPCs (*"Remote Procedure Calls"*) y puede encontrarse en:
"ftp://ftp.imag.fr/pub/ipv6/NFS/NFS_IPV6_FreeBSD4.5.tgz".

Tanto el sistema NAT-PT como el servidor de NFS IPv4 utilizan una versión del kernel de KAME para *FreeBSD*-4.6. La configuración del sistema NAT-PT es la siguiente:

```
prefix 2001:720:410:1009::

map from any6 to 163.117.140.201

map enable
```

Con la regla "prefix" se especifica el prefijo de los paquetes susceptibles de ser tratados por el NAT-PT. La siguiente regla especifica que cualquier paquete cambiará su dirección IPv6 origen por la dirección 163.117.140.184, que es la dirección del NAT-PT y de esta forma los posibles paquetes de respuesta se encaminarán de vuelta hacia el sistema NAT, el cual los reenviará alñ nodo IPv6 cliente del servicio de NFS.

La regla "map enable" activa el funcionamiento del NAT-PT, en cualquier momento se puede desactivar el mecanismo con el comando "natptconfig map disable". La máquina que actúa como NAT-PT posee la dirección IPv4 fija "163.117.140.184". Además dispone de otras tres direcciones utilizadas para poder realizar mapeos en la configuración del NAT, pertenecientes a lo que se denomina el *"pool"* de direcciones del NAT, concretamente son las siguientes:

- 163.117.140.200: mapea la transferencia de zonas del DNS como se explicará a continuación.
- 163.117.140.201: utilizada para el mapeo del servicio de NFS.
- 163.117.140.202: Libre en el momento de realizar este proyecto fin de carrera, se utiliza en pruebas temporales.

Ya sólo nos queda realizar la solicitud para montar el sistema de ficheros NFS, de la siguiente forma:

```
mount_nfs [2001:720:410:1009::163.117.140.41]:/usr/home /home2
```

En este comando se aprecia que la dirección IPv4 del servidor de NFS es "163.117.140.41". Dicha solicitud alcanza el sistema NAT, que traduce la solicitud a IPv4 y traduce de vuelta los paquetes de respuesta. No obstante, el cliente reporta el siguiente mensaje de error:

```
[udp6] 2001:720:410:1009::163.117.140.41:/usr: RPCPROG_NFS: RPC:
(unknown error code)
```

Si se ejecuta el comando "rpcinfo", se obtiene el mismo error:

```
root@mira:/usr/home/jrh# rpcinfo -T udp6 \
       2001:720:410:1009::163.117.140.41 100005 3
(unknown error code)
```

Para solucionar este problema, hay que editar el archivo "/etc/netconfig". Dicho archivo se utiliza únicamente en el código RPC de la librería de C. El contenido de dicho archivo se muestra a continuación:

```
# Entries consist of:
#
# <network_id> <semantics> <flags> <protofamily> <protoname> \
#               <device> <nametoaddr_libs>
#
# The <device> and <nametoaddr_libs> fields are always
# empty in FreeBSD.
#
udp6    tpi_clts      v   inet6    udp   -   -
tcp6    tpi_cots_ord  v   inet6    tcp   -   -
udp     tpi_clts      v   inet     udp   -   -
tcp     tpi_cots_ord  v   inet     tcp   -   -
rawip   tpi_raw       -   inet      -    -   -
unix    tpi_cots_ord  -   loopback  -    -   -
```

El cambio que se debe realizar a este archivo consiste en sustituir "inet6" por "inet". Una vez hecho este cambio, tanto el "rpcinfo" como el "mount" funcionan perfectamente a través del sistema NAT-PT, como se observa a continuación para el caso del "rpcinfo":

```
root@mira:/usr/home/jrh# rpcinfo -T udp6 \
       2001:720:410:1009::163.117.140.41 100005 3
program 100005 version 3 ready and waiting
```

Un "mount" nos muestra que el sistema de ficheros se encuentra correctamente montado:

```
root@mira:/usr/home/jrh/# mount
/dev/da0s3a on / (ufs, local)
/dev/da0s3f on /usr (ufs, local)
/dev/da0s3e on /var (ufs, local)
procfs on /proc (procfs, local)
[2001:720:410:1009::163.117.140.41]:/usr/home on /home2 (nfs)
```

Una vez hecho el montaje, se ha comprobado que es posible modificar el sistema de ficheros exactamente igual que en IPv4. También se ha comprobado que al parar el sistema NAT-PT el cliente de NFS se queda colgado hasta que vuelve a entrar en funcionamiento.

Transferencia de zonas a través de NAT-PT.

En este apartado se comenta brevemente el funcionamiento del sistema NAT-PT utilizado para realizar una trasferencia de zona entre un servidor DNS maestro IPv4 y un servidor DNS esclavo IPv6.

A continuación se muestra la configuración de ambos servidores:

```
(master /etc/named/named.conf)
zone "ipv6.it.uc3m.es." {
        type master;
        file "entries.ipv6.it.uc3m.es";
        allow-transfer{
                2001:720:410:1009::163.117.140.41;
        };

};

(slave /etc/named/named.conf)
zone "ipv6.it.uc3m.es" {
        type slave;
        file "ipv6.it.uc3m.es";
        masters {
                163.117.140.200;
        };
```

Ambos servidores utilizan la versión de DNS denominada "bind9.2.1". Después de borrar el archivo "ipv6.it.uc3m.es" en el esclavo, se reinician ambos demonios y dicho archivo se vuelve a crear, lo que indica que la tranferencia de la zona se ha efectuado correctamente.

3.2.3 Consideraciones finales.

Por un lado, el mecanismo de NAT-PT combinado con el mecanismo de DNS-ALG porporciona conectividad bidireccional entre el mundo IPv6 y el mundo IPv4. Este hecho hace que NAT-PT resulte útil para aquellas redes IPv6-only que necesiten desarrollar servidores visibles desde el mundo IPv4.

Por otro lado, Entre los problemas inherentes a este mecanismo y a cualquier mecanismo tipo "NAT", podemos enunciar los siguientes:

- Como NAT-PT realiza traducción de direcciones, cualquier aplicación que haga uso de direcciones IP en capas más altas que la capa IP no funcionará adecuadamente. En este caso, es necesario habilitar mecanismos de *"application layer gateway"* (ALG) para solucionar este problema.
- La seguridad IP extremo a extremo no es posible. También puede ocurrir que la seguridad a nivel de transporte y aplicación se vea comprometida para aquellas aplicaciones que transporten direcciones IP en estas capas. Independientemente del mecanismo de NAT-PT, *"end-to-end"* IPsec no es posible entre diferentes zonas de direccionamiento. Los nodos extremos que soliciten IPSec deben soportar ambos IPv4 o ambos IPv6.

- Todos los paquetes pertenecientes a una sesión deben ser encaminados hacia el mismo NAT para que la comunicación sea posible.

3.3 Mecanismo 3: Túneles automáticos.

En esta sección describimos la configuración del mecanismo de transición denominado como túneles automáticos. Dichos túneles se encuentran descritos en [**?**].

3.3.1 Configuración.

KAME no da soporte los túneles automaticos, por considerarlos deprecados y por los posibles agujeros de seguridad que una implementación de la RFC podría conllevar.

En *Linux*, al levantar la interfaz "sit", automáticamente se activa el mecanismo de túnel automático (valga la redundancia), asignandose una dirección ::<IPv4>/96 por cada dirección IPv4 de la máquina.

Dado que estamos hablando de túneles "automáticos", la configuración en este caso resulta inexistente.

3.3.2 Pruebas funcionales.

Las pruebas funcionales se reducen a comprobar la conectividad mediante un ping, como se observa a continuación:

```
root@pulgon:/usr/src/usagi/usagi/ip# ping6 ::163.117.140.41
PING ::163.117.140.41 (::163.117.140.41): 56 data bytes
64 bytes from ::163.117.140.41: icmp_seq=0 ttl=64 time=9.712 ms
64 bytes from ::163.117.140.41: icmp_seq=1 ttl=64 time=0.268 ms
64 bytes from ::163.117.140.41: icmp_seq=2 ttl=64 time=0.26 ms

--- ::163.117.140.41 ping statistics ---
3 packets transmitted, 3 packets received, 0% packet loss
round-trip min/avg/max = 0.26/3.413/9.712 ms
```

La configuración automática se puede observar haciendo un "ifconfig sit0":

```
sit0      Link encap:IPv6-in-IPv4
          inet6 addr: ::163.117.140.177/96 Scope:Compat
          inet6 addr: ::127.0.0.1/96 Scope:Unknown
          UP RUNNING NOARP  MTU:1480  Metric:1
          RX packets:3 errors:0 dropped:0 overruns:0 frame:0
          TX packets:3 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:0
```

3.3.3 Consideraciones finales.

En [?] se describen un conjunto de ataques utilizando túneles automáticos y túneles 6to4. Estos ataques se basan en la utilización de direcciones IPv4 de broadcast en el origen para que cuando el nodo destino envíe un paquete de respuesta, inunde su propia red. En [?] se describen además otros tipos de ataques.

En general, se deben examinar las direcciones de la cabecera de encapsulación y la cabecera interna para evitar este tipo de ataques. Todo aquel mecanismo que encapsule paquetes de una forma semejante a los túneles automáticos es susceptible de sufrir estos ataques.

Por otro lado, la ventaja de este mecanismo es su nula configuración. Un sistema operativo con soporte de túneles automáticos, como en el caso de *Linux*, permite obtener comunicación con otras islas IPv6 de una forma totalmente directa.

KAME no los implementa debido a las consideraciones de seguridad enunciadas en [?], pero la correspondiente RFC que describe el mecanismo, a día de hoy, es un documento completamente válido.

3.4 Mecanismo 4: Túneles configurados.

En el mecanismo de transición denominado "túneles configurados", las direcciones de los extremos del túnel se determinan mediante la información de configuración del propio nodo encapsulador. Para cada extremo del túnel, el nodo debe almacenar la dirección del otro extremo. Cuando un paquete IPv6 se transmite por el túnel, la dirección del extremo del túnel que se encuentra configurada se utiliza como la dirección de destino en la cabecera IPv4 externa.

La determinación de qué paquetes encapsular se realiza, como es natural, utilizando información de la tabla de rutas. Este mecanismo resulta bastante sencillo, posee una arquitectura punto a punto y como contrapartida necesita de una configuración manual en ambos extremos, por lo que sólo resulta útil para enlaces permanentes.

3.4.1 Configuración.

Los túneles configurados se crean de forma semejante tanto en *Linux* como en *FreeBSD*, a través del comando "ifconfig" y utilizando el dispositivo adecuado ("sit" o "tun" en *Linux* y "gif" en *FreeBSD*). Por poner un ejemplo, a continuación se muestra un tunel configurado en *FreeBSD*:

```
root@alacran:/usr/home/jrh\$ gifconfig gif0
gif0: flags=c051<UP,POINTOPOINT,RUNNING,LINK2,MULTICAST> mtu 1280
        inet6 2001::1  prefixlen 64
        inet6 fe80::2c0:26ff:fea3:5df6%gif0  prefixlen 64
        physical address inet 163.117.140.44 --> 163.117.140.166
```

Dicho túnel se puede crear en dos pasos, utilizando primero el comando "gifconfig" para establecer los extremos del túnel, y posteriormente utilizando el comando "ifconfig" para

establecer la dirección IPv6 de la pseudo-interfaz, o en un único paso utilizando sólamente el comando "ifconfig" de la siguiente forma:

```
ifconfig gif0 inet6 add 2001:710:420:ffff::1 2001:710:420:ffff::2 prefixlen 126
gifconfig gif0 inet 163.117.140.44 163.117.140.166
```

También se podía haber escrito en una sola línea, utilizando la opción *"tunnel"* del comando *"ifconfig"*:

```
ifconfig gif0 inet6 add 2001:710:420:ffff::1 2001:710:420:ffff::2 \
   tunnel 163.117.140.44 163.117.140.166 prefixlen 126
```

Al igual que ocurre con todos los pseudo-interfaces en *FreeBSD*, si no existe ningún dispositivo "gif0" en nuestro sistema, pero el kernel los soporta, se pueden crear con el comando "ifconfig gif0 create".

3.4.2 Pruebas funcionales.

Las pruebas consisten en comprobar la conectividad entre los extremos del túnel. Previamente a su realización se muestra la configuración de los túneles en ambos extremos:

Este es nuestro extremo:

```
gif0: flags=8051<UP,POINTOPOINT,RUNNING,MULTICAST> mtu 1280
   tunnel inet 193.146.185.156 --> 213.96.215.56
   inet6 3ffe:3328:6::f571 prefixlen 126
   inet6 fe80::290:27ff:fe86:93d%gif0 prefixlen 64 scopeid 0xa
```

Este es el extremo remoto:

```
gif0: flags=8051<UP,POINTOPOINT,RUNNING,MULTICAST> mtu 1280
   tunnel inet 213.96.215.56 --> 193.146.185.156
   inet6 3ffe:3328:6::f572 prefixlen 126
   inet6 fe80::290:3023:32a3:0ad3%gif0 prefixlen 64 scopeid 0xa
```

Para comprobar la conectividad entre los extremos, se suele realizar un ping6 a la dirección multicast de *"all-host"*, en cuyo caso se deben obtener dos respuestas, una la de nuestro propia interfaz y otra la del extremo remoto. Si este ping funciona, existe conectividad entre los extremos. Para el ping que se muestra a continuación, se ha utilizado una pseudo-interfaz que encapsula IPv6 sobre IPv6:

```
rh@mira:/usr/home/jrh$ ping6 ff02::1%gif2
PING6(56=40+8+8 bytes) fe80::290:27ff:fe86:93d%gif2 --> ff02::1%gif2
16 bytes from fe80::290:27ff:fe86:93d%gif2, icmp_seq=0 hlim=64 time=0.285 ms
16 bytes from fe80::290:27ff:fe87:9bfc%gif2, icmp_seq=0 hlim=64 time=53.733 ms(DUP!)
16 bytes from fe80::290:27ff:fe86:93d%gif2, icmp_seq=1 hlim=64 time=0.298 ms
16 bytes from fe80::290:27ff:fe87:9bfc%gif2, icmp_seq=1 hlim=64 time=59.309 ms(DUP!)
```

3.4.3 Consideraciones finales.

El mecanismo de encapsulación no resulta nuevo. Los túneles IPv4-en-IPv4 ya existían desde hace tiempo. Por consiguiente, resulta un mecanismo relativamente sencillo de implementar y flexible. En *FreeBSD* se permiten realizar tantas encapsulaciones anidadas como queramos, basta con encaminar el paquete de nuevo hacia la interfaz túnel. El número de anidaciones se puede configurar utilizando la siguiente variable de *"sysctl"*:

```
net.link.gif.max_nesting
```

Este mecanismo ha sido el primero en ser implementado y actualmente resulta el más utilizado. Por ejemplo, la red de pruebas *"6bone"* utiliza túneles configurados para establecer sus enlaces.

3.5 Mecanismo 5: TRT.

Este mecanismo, muy sencillo de configurar, permite realizar una pasarela a nivel TCP entre IPv6 e IPv4. En el momento de escribir este trabajo, el autor sólo conoce la implementación de la distribución *FreeBSD*.

3.5.1 Configuración.

El mecanismo TRT se ha probado bajo máquinas *FreeBSD*, para lo cual, primeramente se debe compilar el kernel para que soporte la pseudo-interfaz "faith" añadiendo la siguiente línea al archivo de configuración:

```
pseudo-device  faith  1  #IPv6-to-IPv4 relaying (translation)
```

Posteriormente, el TRT se configura utilizando el demonio "faithd".

Cuando "faithd" se invoca directamente por línea de comandos, se debe especificar el puerto TCP o servicio sobre el que va a actuar. Si se encuentra tráfico TCPv6 hacia dicho puerto, la conexión será reenviada. De esta forma, como "faithd" está escuchando en un puerto determinado, no es posible ejecutar el correspondiente demonio local en ese mismo puerto. Para solventar este problema, "faithd" posee una opción que permite invocar al demonio local si la dirección de destino es una dirección local de la interfaz de la máquina y en caso contrario, realizar la pasarela.

Antes de invocar al demonio, se debe configurar la interfaz "faith" de forma adecuada, utilizando los siguientes comandos:

```
sysctl net.inet6.ip6.accept_rtadv=0
sysctl net.inet6.ip6.forwarding=1
sysctl net.inet6.ip6.keepfaith=1

ifconfig faith0 up
```

Ya sólo queda especificar qué paquetes van a ser tratados por el demonio de TRT, lo cual se realiza añadiendo una ruta hacia la pseudo-interfaz "faith". Al prefijo utilizado para encaminar tráfico hacia el traductor se le denomina, al igual que ocurre en el mecanismo NAT-PT presentado con anterioridad, *"fake-prefix"*. Los paquetes susceptibles de ser modificados por el traductor poseen *"fake-addresses"* como direcciones destino según la termilogía introducida. A continuación se muestra la ruta, que en *FreeBSD* se debe añadir en dos pasos, dado que *FreeBSD* no permite añadir directamente una ruta a través de la pseudo-interfaz "faith":

```
route add -inet6 3ffe:501:4819:ffff:: -prefixlen 96 ::1
route change -inet6 3ffe:501:4819:ffff:: -prefixlen 96 -ifp faith0
```

Finalmente, el demonio puede ser invocado directamente o a través del "inetd". Como ejemplo se muestra la ejecución del demonio para que actúe como pasarela del servicio de telnet:

```
faithd telnet
```

Donde el nombre "telnet" se resuelve a un número de puerto utilizando el archivo "/etc/services". Por supuesto, también podríamos haber ejecutado lo siguiente:

```
faithd 23
```

3.5.2 Pruebas funcionales.

El mecanismo de TRT se utiliza de forma estable en nuestra red de pruebas para acceder a un servidor web y para permitir la comunicación entre dos servidores de IRC. A continuación se presentan brevemente ejemplos de configuración y funcionamiento:

Servidor web.

Se dispone de un servidor web *apache* que sólo acepta conexiones IPv4. Para acceder por IPv6 se utiliza el mecanismo TRT, que se encuentra escuchando en el puerto 80. Además de levantar el mecanismo de TRT en la máquina donde se ejecuta el servidor web, resulta necesario encaminar el tráfico hacia dicho traductor, lo que se realiza inyectando una ruta de *host* desde la máquina que actúa de traductor.

En el extrato siguiente se presenta un ejemplo de conectividad utilizando IPv4 y el comando "telnet" contra el puerto web, número 80:

```
jrh@zangano:/usr/home/jrh$ telnet 163.117.140.166 80
Trying 163.117.140.166...
Connected to mira.it.uc3m.es.
Escape character is '^]'.
```

A continuación se presenta el caso donde entra en juego el traductor TRT, mostrando además una petición de una página inexistente, para dejar constancia del correcto funcionamiento del servidor:

```
jrh@zangano:/usr/home/jrh$ telnet www.ipv6.it.uc3m.es 80
Trying 2001:720:410:1001:1:0:a375:8ca6...
Connected to www.ipv6.it.uc3m.es.
Escape character is '^]'.
GET /nofile.html
<!DOCTYPE HTML PUBLIC "-//IETF//DTD HTML 2.0//EN">
<HTML><HEAD>
<TITLE>404 Not Found</TITLE>
</HEAD><BODY>
<H1>Not Found</H1>
The requested URL /nofile.html was not found on this server.<P>
<HR>
<ADDRESS>Apache/1.3.26 Server at mira.it.uc3m.es Port 80</ADDRESS>
</BODY></HTML>
Connection closed by foreign host.
```

En este ejemplo se observa que el *"fake-prefix"* es "2001:720:410:1001:1::", siendo la dirección IPv4 de la máquina destino 163.117.140.166, lo cual expresado en hexadecimal se corresponde con "a375:8ca6".

Comunicación entre servidores de IRC.

Para realizar esta comunicación se realiza una configuración muy semejante a la del ejemplo anterior, cambiando el puerto del demonio de "faith" por el utilizado en la comunicación de los servidores de irc, concretamente el puerto 6667. El *"fake-prefix"* utilizado es el mismo, al igual que la máquina que hace uso del mecanismo de transición, ya que el demonio de TRT puede ejecutarse en puertos distintos.

A modo de ejemplo, se va a simular la conectividad entre un servidor de IRC IPv4 y otro servidor de IRC IPv6 utilizando un telnet contra el puerto 6667:

```
root@pulgon:/usr/home/jrh# telnet 2001:720:410:1001:1::163.117.140.166 6667
Trying 2001:720:410:1001:1:0:a375:8ca6...
Connected to 2001:720:410:1001:1::163.117.140.166.
Escape character is '^]'.
```

Además, también puede verse el establecimiento de la conexión utilizando el comando "netstat" en la máquina cliente, como se muestra a continuación. Nótese que se ha editado la salida del comando para ajustarlo al encuadre de este trabajo:

```
root@pulgon:/usr/home/jrh# netstat -ln
Active Internet connections

Proto Recv-Q Send-Q
tcp6  0      0

Local Address
2001:720:410:1001:2c0:26ff:fea3:68f4.1744
```

```
Foreing Address
2001:720:410:1001:1:0:a375:8ca6.6667

(state)
ESTABLISHED
```

Aunque en ambos ejemplos tanto el mecanismo de transición como el servicio al que se quiere transitar se ofrecen en la misma máquina, el sistema de traducción puede situarse en una máquina distinta (el caso más normal), y ajustar el enrutamiento entre ambos mundos (el IPv4 y el IPv6) para encaminar los paquetes de ida y vuelta a través del traductor.

3.5.3 Consideraciones finales.

Dentro de la red del proyecto LONG, se ha utilizado este mecanismo para conectar dos servidores de IRC. El servidor IPv6 utiliza el prefijo definido por el mecanismo TRT para encaminar paquetes hasta la máquina que realiza el *"relay"*. El servidor de IRC IPv4 se encuentra situado en la misma máquina que implementa el mecanismo de TRT.

La implementación de TRT en *FreeBSD* permite definir controles de acceso al servicio, basado en direcciones. El archivo "/etc/faithd.conf" realiza este control. A continuación se muestra un ejemplo de dicho archivo:

```
3ffe:501:ffff::/48 deny 10.0.0.0/8
3ffe:501:ffff::/48 deny 127.0.0.0/8
3ffe:501:ffff::/48 permit 0.0.0.0/0
```

En este ejemplo, se permite a cualquier paquete proviniente de la red "3ffe:501:ffff::/48" utilizar el traductor para conectarse a cualquier destino excepto a la subred "10.0.0.0/8" y a la subred "127.0.0.0/8". No se permite ningún otro tipo de conexiones. Para más información sobre la configuración del mencanismo, ver "man faithd".

3.6 Mecanismo 6: ISATAP.

ISATAP son las iniciales de *"Intra-Site Automactic Tunnel Addressing Protocol"*. Se trata de un mecanismo de transición que habilita el desarrollo incremental de IPv6 utilizando la infraestructura de IPv4 como un enlace *"Non-Broadcast Multiple Access"* o NBMA.

Este mecanismo hace uso de un nuevo formato de identificador de interfaz que contiene en su interior una dirección IPv4 (que puede ser pública o privada), lo que permite la encapsulación automática en IPv4.

Cada máquina debe preguntar a un router ISATAP para obtener la información de encaminamiento adecuada. Cuando la máquina envía paquetes hacia internet, dichos paquetes son encaminados utilizando un router ISATAP, mientras que paquetes destinados hacia otras máquinas ISATAP dentro de nuestro dominio se encapsulan directa y automáticamente hasta el destino.

Los clientes ISATAP normalmente realizan autoconfiguración sin estado utilizando el mecanismo de "descubrimiento automático de routers ISATAP" descrito en [**?**], pero también se pueden utilizar direcciones ISATAP asignadas de forma estática.

En [**?**] se describe la autoconfiguración de los clientes ISATAP utilizando el mecanismo de "descubrimiento automático de routers ISATAP", mecanismo que interactúa con el servidor de nombres. Tambíen existe la posibilidad de configurar manualmente los clientes. Aunque durante la realización de este proyecto se ha probado la autoconfiguración de los clientes utilizando la entrada de DNS "isatap.domain-name", en los ejemplos de los siguientes apartados se va a describir como se realizaría una configuración manual del mecanismo. La configuración automática a través del DNS hace uso de la ejecución de un script escrito en *Perl*.

Se ha probado el mecanismo tanto en *FreeBSD* como en máquinas con sistema operativo *Linux*. A continuación se va a describir su configuración para cada una de estas plataformas.

3.6.1 Configuración en *Linux*.

Para instalar el mecanismo ISATAP en máquinas *Linux*, se necesita:

- Un kernel 2.4.x con soporte ISATAP, proporcionado por USAGI.
- Paquete "iproute" modificado por USAGI.
- "radvd" con soporte de ISATAP para los routers ISATAP.

La implementación actual de ISATAP, escrita por *"Fred Templin"* se encuentra disponible en USAGI desde el paquete "2001-11-12". Nótese que sólo está disponible para el kernel 2.4 de *Linux*. Cuando se compila el kernel, se debe activa la opción *"ISATAP interface support"*.

El paquete "iproute" se distribuye dentro de las fuentes de USAGI, y está modificado para poder crear túneles de tipo ISATAP.

Por último, para los routers se necesita tener la versión "radvd 0.7.0" o superior, que posee soporte para redes *"non-broadcast"*.

Configuración del router ISATAP.

La funcionalidad de ISATAP se obtiene creando un túnel de tipo ISATAP utilizando el comando "ip" de la siguiente forma:

```
ip tunnel add is0 mode isatap local 163.117.140.177 ttl 64
```

Una vez que se ha creado la interfaz ISATAP, se necesita asignar una dirección IPv6 ISATAP a dicha interfaz:

```
ip link set is0 up
ip addr add 2001:720:410:1005::5efe:163.117.140.177/64 dev is0
```

Nótese que la parte de la dirección correspondiente a "5efe"pertenece a la defición del formato de las direcciones ISATAP.

Por último, una vez que se ha creado y configurado la interfaz ISATAP, es necesario configurar el demonio "radvd" para que advierta el prefijo ISATAP a los clientes. La configuración del demonio de *"router advertisement"* para un enlace ISATAP se realiza de forma semejante a la configuración sobre un enlace *"ethernet"*, salvo que debe aparecer la etiqueta *"UnicastOnly"* en la definición de la interfaz, por ejemplo:

```
interface is0
{
    AdvSendAdvert on;
    UnicastOnly on;
    AdvHomeAgentFlag off;

    prefix 2001:720:410:1005::/64
    {
        AdvOnLink on;
        AdvAutonomous on;
        AdvRouterAddr off;
    };
};
```

Configuración del cliente ISATAP.

La configuración del cliente es mucho más sencilla que la del router. El único paso de configuración necesario consiste en la creación de la interfaz ISATAP. No obstante, en los clientes se debe especificar la dirección IPv6 del router ISATAP de forma que el cliente sepa dónde enviar mensajes de *"router solicitation"*. El comando es el siguiente:

```
ip tunnel add is0 mode isatap local 163.117.140.41 v4any 163.117.140.177 ttl 64
ip link set is0 up
```

De esta forma, los clientes solicitan la dirección y la información de encaminamiento al router ISATAP y se autoconfiguran de forma apropiada.

Bajo determinadas circunstancias puede resultar últil omitir la configuración automática de los nodos ISATAP. En este caso, no se define ningún router ISATAP y cada cliente debe configurar su propia dirección e información de routeo de forma manual. Para ellos, en el comando anterior basta con omitir la opción "v4any V4ADDR_RTR" y añadir una ruta adecuada.

3.6.2 Configuración en *FreeBSD*.

El mecanismo ISATAP para máquinas *FreeBSD* se encuentra recientemente implementado en la distribución que proporciona KAME, concretamente desde el *"snap"* del dia 13 de Enero del 2003. Esta implementación se encuentra basada en [**?**].

La configuración del mencanismo de transición ISATAP en máquinas *FreeBSD* se realiza utilizando la interfaz "stf". De este modo, vemos que la interfaz "stf" soporta dos mecanismos de transición, de la siguiente forma:

- Interfaz "stf0": implementa el mecanismo "6to4".
- Interfaz "stf1": implementa el mecanismo "ISATAP".

Por consiguiente, se debe recompilar el núcleo para dar soporte al menos a dos interfaces "stf", utilizandose para ello la siguiente línea del archivo de configuración:

```
pseudo-device   stf   2
```

Para configurar el mecanismo ISATAP es necesario realizar los siguientes pasos:

- Configurar una dirección ISATAP *"link local"* en la interfaz: deben ser del estilo "fe80::0000:5efe:xxyy:zzuu/64", donde xxyy:zzuu es la traducción a hexadecimal de cualquier dirección IPv4 utilizada por la máquina.
- Configurar al menos la dirección IPv4 de un router ISATAP: esto se puede realizar de varias formas, utilizando un servidor de DHCP, utilizando una entrada especial en el servidor de nombres (isatap.domain-name) o mediante configuración manual.

Para que una máquina se comporte como un router ISATAP se tiene que asignar una dirección IPv6 a la interfaz *"stf1"* que no sea *"link local"*, además de realizar los pasos descritos anteriormente. Por supuesto, al actuar como un router, se deben redistribuir los prefijos utilizando el demonio *"rtadvd"* de forma equivalente a como se haría con cualquier otra interfaz física.

A modo de ejemplo, a continuación se describen los comandos necesarios para configurar un cliente y un router ISATAP:

Configuración del cliente ISATAP.

```
ifconfig fxp0 inet 163.117.140.182 netmask 0xffffff00

ifconfig stf1 isataprtr 163.117.140.184
ifconfig stf1 inet6 fe80::5efe:163.117.140.182 prefixlen 64

rtsold stf1
```

Debido a que los routers ISATAP no pueden enviar *"unsolicited router advertisements"*, resulta necesario ejecutar el demonio de solicitud "explícita" de los mismos, lo cual se realiza mediante el comando "rtsold stf1".

La interfaz ISATAP del cliente en *FreeBSD* posee el siguiente aspecto una vez configurado:

```
stf1: flags=c041<UP,RUNNING,LINK2,MULTICAST> mtu 1280
  inet6 fe80::5efe:a375:8cb6%stf1 prefixlen 64 scopeid 0x9
  inet6 2001:720:410:1008:0:5efe:a375:8cb6 prefixlen 64 autoconf
  isatapmode on
  isataprtr 163.117.140.184
```

Configuración del router ISATAP.

```
ifconfig fxp0 inet 163.117.140.184 netmask 0xffffff00

ifconfig stf1 isataprtr 163.117.140.184
ifconfig stf1 inet6 fe80::5efe:163.117.140.184 prefixlen 64
ifconfig stf1 inet6 2001:720:410:1008::5efe:163.117.140.184 prefixlen 64 alias

rtadvd stf1
```

A diferencia de la implementación de *Linux*, no es necesario especificar un archivo de configuración para el demonio de *"router advertisement"*, sino que basta con indicar la interfaz de salida, anunciandose automáticamente todos los prefijos globales que dicha interfaz pueda poseer.

Pruebas funcionales.

Para las pruebas de funcionamiento se han utilizado máquinas *Linux*, aunque también se han probado configuraciones basadas exclusivamente en *FreeBSD* y configuraciones mixtas actuando *FreeBSD* como router ISATAP y las máquinas *Linux* como clientes, sin que se haya producido ningún problema de interacción entre las distintas implementaciones.

En el esquema 3.1 se muestra gráficamente la topología de red y las máquinas utilizadas para la realización de la prueba:

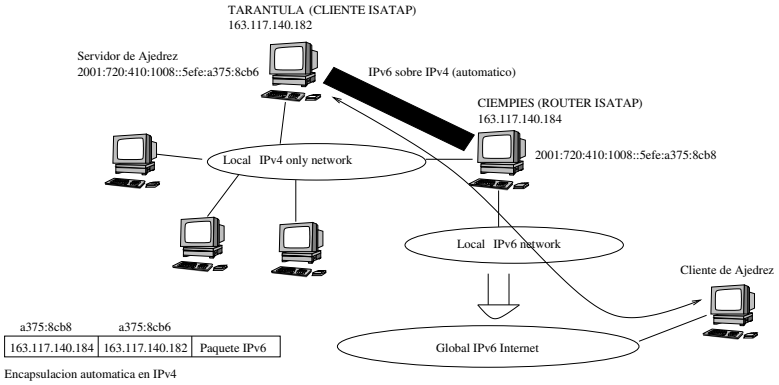

Figura 3.1: Máquinas utilizadas para probar el mecanismo ISATAP.

Se han utilizado máquinas linux con la ultima versión estable de USAGI, concretamente se ha utilizado el paquete "usagi-linux24-stable-20021007.tar.bz2". Esta distribución implementa el mecanismo ISATAP. Por otro lado, en las pruebas que se van a describir a continuación se ha utilizado el paquete "radvd-0.7.2.tar.gz" para poder enviar *"router advertisements"*. Dicho paquete se puede descargar de su pagina oficial localizada en "http://v6web.litech.org/radvd/".

3.6.3 Consideraciones finales.

De esta forma, y siguiendo los pasos de configuración explicados en el apartado anterior, el cliente ISATAP queda configurado de la siguiente forma:

```
eth0      Link encap:Ethernet  HWaddr 00:02:B3:3C:DA:50
          inet addr:163.117.140.41  Bcast:163.117.140.255  Mask:255.255.255.0
          inet6 addr: fe80::202:b3ff:fe3c:da50/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:7322 errors:0 dropped:0 overruns:0 frame:0
          TX packets:3935 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:100
          Interrupt:10

is0       Link encap:IPv6-in-IPv4
          inet6 addr: 2001:720:410:1005:0:5efe:a375:8c29/64 Scope:Global
          inet6 addr: fe80::5efe:a375:8c29/64 Scope:Link
          UP RUNNING NOARP  MTU:1480  Metric:1
          RX packets:190 errors:0 dropped:0 overruns:0 frame:0
          TX packets:242 errors:49 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:0
```

Como se puede observar en el extracto del comando "ifconfig", la máquina posee una interfaz con una dirección global IPv4, y la pseudo-interfaz que simula el comportamiento de ISATAP denominado "is0".

De una forma similar, se configura el router ISATAP:

```
is0       Link encap:IPv6-in-IPv4
          inet6 addr: 2001:720:410:1005:0:5efe:a375:8cb1/64 Scope:Global
          inet6 addr: fe80::5efe:a375:8cb1/64 Scope:Link
          UP RUNNING NOARP  MTU:1480  Metric:1
          RX packets:233 errors:0 dropped:0 overruns:0 frame:0
          TX packets:187 errors:2 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:0
```

Aunque sólo se muestra la pseudo-interfaz "is0" que implementa el mecanismo ISATAP, el router posee interfaces con direcciones IPv6 y otros interfaces con direcciones IPv4. El router ISATAP envía *"router advertisements"*, para lo cual se ha creado el archivo "/usr/local/etc/radvd.conf" con la sintaxis enunciada en el apartado anterior, es decir, se anuncia el "prefijo 2001:720:410:1005::" bajo demanda (opción *"UnicastOnly"*).

Una vez configurados el cliente y el router ISATAP, se prueba el correcto funcionamiento del mecanismo efectuando un ping6 a una máquina IPv6 externa, por ejemplo:

```
root@zangano:/usr/src/usagi/usagi/ip $ ping6 www.kame.net
PING www.kame.net from 2001:720:410:1005:0:5efe:a375:8c29 : 56 data bytes
64 bytes from 3ffe:501:4819:2000:210:f3ff:fe03:4d0: icmp_seq=0 time=587.151 msec
64 bytes from 3ffe:501:4819:2000:210:f3ff:fe03:4d0: icmp_seq=1 time=583.513 msec
64 bytes from 3ffe:501:4819:2000:210:f3ff:fe03:4d0: icmp_seq=2 time=598.286 msec
64 bytes from 3ffe:501:4819:2000:210:f3ff:fe03:4d0: icmp_seq=3 time=439.408 msec

--- www.kame.net ping statistics ---
4 packets transmitted, 4 packets received, 0% packet loss
round-trip min/avg/max/std-dev = 439.408/552.090/598.286/65.284 ms
```

Por otro lado, se realiza también un "traceroute", donde se puede observar que el cliente envía los paquetes hacia el router ISATAP, y el router ISATAP se encarga de encaminar los paquetes por la red IPv6 hasta alcanzar el destino:

```
root@zangano:~ $ traceroute6 www.kame.net
traceroute to www.kame.net (3ffe:501:4819:2000:210:f3ff:fe03:4d0) \
        from 2001:720:410:1005:0:5efe:a375:8c29, 30 hops max, 32 byte packets
 1  2001:720:410:1005:0:5efe:a375:8cb1   0.292 ms  0.176 ms  0.146 ms
 2  grillo.ipv6.it.uc3m.es   1.121 ms *  1.168 ms
 3  2001:720:410::1   8.606 ms *  8.588 ms
 4  3ffe:3300::29:5   20.33 ms  20.687 ms  20.895 ms
 5  2001:660:1102:4007::1   58.19 ms  68.859 ms  58.52 ms
 6  2001:660:1102:4003::2   80.994 ms  76.155 ms  79.826 ms
 7  3ffe:8120::2   199.099 ms  209.317 ms  204.997 ms
 8  3ffe:3900:1000:1::1   392.758 ms  470.26 ms *
 9  wide-6tapsl.6tap.net   439.181 ms  442.862 ms  439.288 ms
10  pc3.nezu.wide.ad.jp   439.386 ms  440.075 ms  440.854 ms
11  pc7.nezu.wide.ad.jp   440.481 ms  444.486 ms  436.58 ms
12  pc3.yagami.wide.ad.jp   439.279 ms  443.52 ms  441.542 ms
13  gr2000.k2c.wide.ad.jp   442.513 ms  441.591 ms  439.454 ms
14  apple.kame.net   439.515 ms  444.02 ms  446.518 ms
```

Obviamente, mediante un "tcpdump" en el router ISATAP se puede observar que los paquetes que fluyen entre el cliente y el router van encapsulados en IPv4.

3.6.4 Consideraciones finales.

ISATAP constituye una forma rápida y sencilla de ofrecer conectividad IPv6 para aquellas subredes locales que, por una u otra causa, todavía no posean un prefijo IPv6 global. No obstante, en el momento de realizar este proyecto fin de carrera todavía quedan algunas cuestiones sin cerrar, sobre todo respecto al tiempo de vida de las direcciones ISATAP o respecto a la interacción de ISATAP con el algoritmo de selección de direcciones fuente. También hay que tener en cuenta que ISATAP no funcionará a través de NATs, pero por otro lado sí funciona dentro de la parte privada de la red gestionada por el NAT.

3.7 Mecanismo 7: DSTM.

Durante el desarrollo inicial de IPv6, las máquinas situadas en redes IPv6 nativas necesitarán mantener conectividad con máquinas y/o aplicaciones que sólo pueden ser alcanzadas utilizando IPv4. El mecanismo denominado *"dual stack transition mechanism"* o DSTM proporciona un medio para conseguir dicha conectividad basándose en el uso de túneles y en la ubicación temporal de direcciones IPv4 globales.

Más concretamente, la dirección temporal IPv4 se asigna a un nodo *"dual-stack"* sólo si la conexión no puede establecerse utilizando IPv6, pero el encaminamiento dentro de la red se produce siempre utilizando IPv6. También debe quedar claro que este mecanismo sólo se puede aplicar en nodos *"dual-stack"*.

3.7.1 Configuración.

Para instalar el mecanismo, se deben seguir los siguiente pasos, tanto para instalar el cliente como para instalar el servidor o TEP:

1. Instalar la distribución inicial de *FreeBSD*-4.5, lo cual se puede hacer mediante un "CV-Sup" utilizando la etiqueta "RELENG_4_5_0_RELEASE". Después basta con hacer en "/usr/src" el correspondiente *"make world"* y seguir los pasos indicados para la instalación de la distribución.

2. Instalar el soporte de IPv6 para las RPCs, para lo cual se necesita instalar el parche suministrado por *Jean-Luc Richier* en:

   `ftp://ftp.imag.fr/pub/ipv6/NFS/NFS_IPV6_FreeBSD4.5.tgz`

   y seguir las instrucciones que contiene en su interior para su correcta instalación.

3. A continuación se debe instalar el parche para el mecanismo DSTM, parche que se puede descargar de:

   `ftp://ftp.ipv6.rennes.enst-bretagne.fr/pub/FreeBSD/dstm/dstm-1.0-FreeBSD45.tgz`

   Después de descomprimir el parche en el directorio raíz, se debe configurar el kernel con las siguientes opciones:

   ```
   options DTI
   pseudo-device gif 4
   ```

   y finalmente compilar el kernel de la forma habitual[2] y reiniciar.

4. El último paso consiste en instalar el demonio del cliente DSTM, denominado "dstmd". El paquete se distribuye con un binario ya compilado e instalado, pero conviene compilarlo de nuevo para asegurarnos que los pasos anteriores se han realizado de forma correcta. Para ello, se deben seguir los siguientes pasos:

   (a) copiar "/sys/netinet/in_dti.h" al directorio "/usr/include/netinet".

   (b) cd /usr/src/sbin/dstmd

   (c) make depend ; make ; make install

Llegados a este punto, el demonio del cliente DSTM y la página del manual deberían estar correctamente instalados.

Para instalar el servidor o el *tunnel end point*, los pasos a seguir son:

1. El TEP necesita un número de interfaces túnel para poder funcionar correctamente. En este caso, si hemos configurado el kernel para tener cuatro interfaces "gif", deberemos hacer lo siguiente:

[2] *config KERNELCONF; cd ../../compile/KERNELCONF ; make depend ; make ; make install*

```
ifconfig gif0 create
ifconfig gif1 create
ifconfig gif2 create
ifconfig gif3 create
```

2. El TEP actúa como una pasarela entre IPv6 e IPv4, por lo que necesita actuar como un router IPv4:

   ```
   sysctl -w net.inet.ip.forwarding=1
   ```

3. A continuación, se necesita instalar el demonio del servidor de DSTM. Aunque ya tenemos una versión compilada en nuestra distribución del parche del DSTM, conviene volverlo a instalar:

   (a) cd /usr/src/sbin/dstmd/server

   (b) make depend ; make ; make install

4. El último paso consiste en iniciar el demonio del servidor del DSTM. Este demonio, denominado "rpcdstmd", se configura a traves del archivo "/usr/local/etc/rpcdstmd.conf". Existe un archivo de ejemplo en "/usr/local/etc/rpcdstmd.conf.sample" donde se puede ver la sintaxis y las opciones de configuración permitidas. Básicamente, en este archivo se debe indicar el espacio de direcciones IPv4 que DSTM utilizará, la forma de distribuirlas y la dirección IPv6 del TEP.

5. El demonio del servidor mantiene una base de datos dinámica con las dirección IPv4 alquiladas en el archivo "/var/db/rpcdstmd.lease". Antes de ejecutar el servidor, se debe crear un archivo vacio de la siguiente forma:

   ```
   touch /var/db/rpcdstmd.lease
   ```

6. Finalmente, se debe ejecutar el demonio del servidor del DSTM:

   ```
   /usr/src/sbin/dstmd/server/rpcdstmd
   ```

Se recomienda inicializar el servidor antes que los clientes.

Para instalar el cliente, los pasos a seguir son los siguientes:

1. Antes de lanzar el demonio, es necesario ejecutar el siguiente comando:

   ```
   ifconfig gif1 create
   ```

2. Ajustar el contador "dti". Este contador indica al kernel el tiempo que debe permanecer a la espera de que el demonio "dstmd" obtenga alguna respuesta. Esto se realiza con el siguiente comando:

   ```
   sysctl -w net.inet.ip.dti_magic=10
   ```

3. Finalmente, se debe lanzar el demonio de DSTM, indicando el nombre del servidor, en este caso "mira.ipv6.it.uc3m.es":

   ```
   /sbin/dstmd --rpcserver mira.ipv6.it.uc3m.es
   ```

3.7.2 Pruebas funcionales.

Se ha probado el mecanismo con el servidor de DSTM actuando también como TEP. La configuración ha sido la siguiente:

- La máquina cliente utiliza sólamente direcciones IPv6 para conectarse al servidor DSTM. No posee direcciones IPv4.
- Tanto la máquina cliente como la máquina servidor están correctamente configurados, según se ha explicado en el apartado anterior, y ejecutan sus respectivos demonios de DSTM.

Se han utilizado las máquinas que se muestran en 3.2:

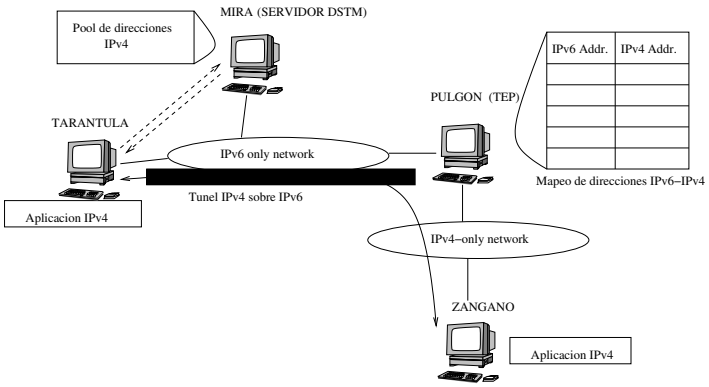

Figura 3.2: Máquinas utilizadas para probar el mecanismo DSTM.

La configuración del servidor DSTM es la siguiente:

```
subnet 163.117.140.0 netmask 255.255.255.0 {
        default-lease-time 20;
        tep6 2001:720:410:1001:290:27ff:fe86:93d;
        tep4 163.117.140.166;
        range 163.117.140.190 163.117.140.250;
}
```

Donde se observa que se proporciona servicio a la subred "163.117.140.0", utilizando un *"pool"* de direcciones entre la "163.117.140.190" y la "163.117.140.250"

Después de lanzar el demonio de DSTM en el cliente, se observa que al intentar hacer un ping se producen automáticamente los siguientes hechos:

- El cliente establece comunicación con el servidor DSTM. El servidor DSTM proporciona la información necesaria y actualiza el fichero "/var/db/rpcdstmd.lease" mostrando la siguiente información:

```
lease 163.117.140.190 {
        starts 4 2002/06/27 11:09:08;
        ends 4 2002/06/27 11:09:18;
        link-local fe80:001:000:000:2c0:26ff:fea3:7ff9;
        hostname ciempies.it.uc3m.es;
        tep6 2001:720:410:1001:290:27ff:fe86:93d;
        tep4 163.117.140.166;
        netmask 255.255.255.0;
}
lease 163.117.140.190 {
        starts 4 2002/06/27 11:09:19;
        ends 4 2002/06/27 11:09:39;
        link-local fe80:001:000:000:2c0:26ff:fea3:7ff9;
        hostname ciempies.it.uc3m.es;
        tep6 2001:720:410:1001:290:27ff:fe86:93d;
        tep4 163.117.140.166;
        netmask 255.255.255.0;
}
```

En este fichero se observa la dirección IPv4 alquilada, el período de tiempo por el cual se alquila, en este caso 20 segundos, y datos referentes al TEP.

- El servidor, que en este caso actúa como TEP, autoconfigura su extremo del túnel "IPv4-en-IPv6":

```
gif0: flags=8051<UP,POINTOPOINT,RUNNING,MULTICAST> mtu 1280
  tunnel inet6 2001:720:410:1001:290:27ff:fe86:93d -->
                        2001:720:410:100e:2c0:26ff:fea3:884b
  inet6 fe80::290:27ff:fe86:93d%gif0 prefixlen 64 scopeid 0x9
  inet 163.117.140.166 --> 163.117.140.190 netmask 0xffffff00
```

- El cliente recibe la información proporcionada por el servidor DSTM y también autoconfigura su extremo del túnel:

```
gif1: flags=8051<UP,POINTOPOINT,RUNNING,MULTICAST> mtu 1280
  tunnel inet6 2001:720:410:100e:2c0:26ff:fea3:884b -->
                        2001:720:410:1001:290:27ff:fe86:93d
  inet6 fe80::2c0:26ff:fea3:7ff9%gif1 prefixlen 64 scopeid 0x9
  inet 163.117.140.190 --> 163.117.140.166 netmask 0xffffff00
```

Pero el ping no funciona. Es necesario volver a ejecutar el "ping", una vez configurado el túnel en el cliente, para que funcione. Cada 20 segundos, la dirección temporal caduda, por lo que se destruye el túnel y se vuelve a solicitar otra dirección. Este hecho interrumpe el funcionamiento del ping, aunque la nueva dirección temporal sea la misma que la anterior.

Por consiguiente, se han detectado los siguientes problemas en la implementación, que deberían ser subsanados:

- El mecanismo DSTM arranca cuando la aplicación necesita una dirección IPv4. Pero una vez que se ha adquirido dicha dirección, la aplicación no consigue establecer comunicación, siendo necesario reiniciarla.

- Cada vez que se necesita renovar el alquiler de la dirección, el tunel se desconfigura y se vuelve a configurar, interrumpiendo las conexiones IPv4 en curso. La solicitud de renovación debería producirse antes de que finalice el alquiler, manteniendo el túnel de tal forma que las aplicaciones puedan seguir funcionando.

Teniendo todo lo anterior en cuenta, en el siguiente extracto se puede observar el funcionamiento del ping:

```
PING 163.117.140.44 (163.117.140.44): 56 data bytes
64 bytes from 163.117.140.44: icmp_seq=0 ttl=63 time=1.178 ms
64 bytes from 163.117.140.44: icmp_seq=1 ttl=63 time=1.344 ms
64 bytes from 163.117.140.44: icmp_seq=2 ttl=63 time=1.335 ms
```

3.7.3 Consideraciones finales.

El mecanismo de transición DSTM es capaz de habilitar una conexión temporal IPv4 dentro de una red IPv6-only. Este no es el caso más común en las primeras fases de IPv6, donde se prevé que los usuarios ya posean al menos conectividad IPv4. Por otro lado, su configuración e instalación resulta, hoy por hoy, relativamente compleja, existiendo algunos defectos (por ejemplo, fallos de conectividad temporales) cuando se renueva el alquiler de una dirección IPv4.

3.8 Conclusiones.

Los nuevos servicios y la futura demanda de los usuarios móviles muestran que el actual protocolo IPv4 no puede satisfacer el número creciente de equipos que necesitarán conectarse a la red en el futuro. Estas nuevas aplicaciones, que se espera que estén conectadas a la red, serán las impulsoras de la introducción del nuevo protocolo a corto plazo.

El desarrollo de IPv6 debe realizarse de tal forma que no se interrumpa el acceso a la actual infraestructura IPv4 y a los servicios asociados con ella. Para poder realizar esta transición, se han desarrollado multitud de mecanismos que posibilitan la interoperabilidad entre ambos protocolos de red.

El estudio realizado en los dos capítulos anteriores me lleva a concluir que los mecanismos de transición existentes son suficientes para satifacer los requisitos de la mayoria de los escenarios posibles.

Cada mecanismo de transición tiene sus propias características. Su aplicabilidad depende de los requisitos que se definan para un determinado escenario. Además, un determinado escenario puede requerir el uso simultáneo de varios mecanismos. Por consiguiente, se puede afirmar que la combinación de varios requisitos y criterios da lugar a la elección del mecanismo de transición más apropiado. Entre los criterios a tener en cuentra se encuentran el diseño

topológico de la red, los costes de gestión, la escalabilidad y la facilidad de uso y acceso por parte de los usuarios finales, por citar sólo algunos de los más importantes.

También hay que comentar que las implementaciones disponibles para algunos mecanismos de transición son, a dia de hoy, escasas. La mayoría de las implementaciones funcionan en ordenadores personales y se espera que en un futuro no muy lejano los principales fabricantes de equipos de red implementen dichos mecanismos.

Capítulo 4

Definición e implantación de la red utilizada.

En este capítulo se presentan las características de la red IPv6 que se ha utilizado para probar los distintos mecanismos de transición. Esta red también ha servido para realizar pruebas enmarcadas dentro del proyecto LONG. A continuación se va a describir la topología y el direccionamiento utilizado, así como los protocolos de encaminamiento, fundamentalmente "BGP4+" entre los socios del proyecto LONG y "RIPng" a nivel interno.

4.1 Topología.

A continuación se describen las máquinas que componen el *"testbed"* utilizado para la realización de las pruebas:

1. Máquinas *Linux*:

   - TARANTULA:
     dir.IP: 163.117.140.182
     Sistema Operativo: *SuSe Linux-4.2 + USAGI patch 12/04/2001*
     Microprocesador: *PENTIUM III (500-MHz)*
     Memoria: 130 Mbytes

   - LIBELULA:
     dir.IP: 163.117.140.43
     Sistema Operativo: *SuSe Linux-4.2 + USAGI patch 12/04/2001*
     Microprocesador: *PENTIUM III (800-MHz)*
     Memoria: 200 Mbytes

   - ZANGANO:
     dir.IP: 163.117.140.41
     Sistema Operativo: *SuSe Linux-4.2 + USAGI patch 12/04/2001*
     Microprocesador: *PENTIUM III (800-MHz)*
     Memoria: 200 Mbytes

- MOSCARDON:
  dir.IP 163.117.140.55
  Sistema Operativo: *SuSe Linux-4.2 + USAGI patch 12/04/2001*
  Microprocesador: *PENTIUM III (800-MHz)*
  Memoria: 200 Mbytes

- CIEMPIES:
  dir.IP 163.117.140.183
  Sistema Operativo: *SuSe Linux-4.2 + USAGI patch 12/04/2001*
  Microprocesador: *PENTIUM III (800-MHz)*
  Memoria: 200 Mbytes

2. Máquinas *FreeBSD*:

   - MIRA:
     dir.IP: 163.117.140.166
     Sistema Operativo: *FreeBSD-4.3 RC + KAME patch 09/07/2001*
     Microprocesador: *PENTIUM III (500 MHz)*
     Memoria: 250 Mbytes

   - PULGON:
     dir.IP: 163.117.140.177
     Sistema Operativo: *FreeBSD-4.3 RC + KAME patch 09/07/2001*
     Microprocesador: *PENTIUM III (733 MHz)*
     Memoria: 130 Mbytes

3. Routers comerciales:

   - CISCO 7506:
     dir.IP: 163.117.140.181
     Sistema Operativo: *RSP Software (RSP-JSV-M), Version 12.2(2)T*
     Microprocesador: *cisco RSP2 (R4700)* con 65 Mbytes de memoria.

Los sistemas operativos se han ido actualizando durante la vida del presente proyecto fin de carrera, las versiones mencionadas se corresponden con las versiones iniciales de este trabajo, pero a medida que han surgido actualizaciones se han ido instalando.

Este conjunto de máquinas se distribuyen físicamente de la forma mostrada en la figura 4.1:

Como se observa en la figura, el router MIRA va a anunciar un prefijo globalmente enrutable para todas las máquinas del departamento. Además, se han creado dos subredes IPv6 nativas (esto significa que no tienen direcciones IPv4) unidas a través del router PULGON para realizar pruebas. Se ha decidido utilizar máquinas *FreeBSD* como routers debido a que la pila IPv6 para *FreeBSD* está más desarrollada que la de *Linux*, ofreciendo servicios de RIPng, BPG4+ y un *"resolver"* que soporta consultas tipo "AAAA" contra el DNS, todo ello integrado en la distribución que KAME proporciona. Aunque no se muestra en el dibujoEl router

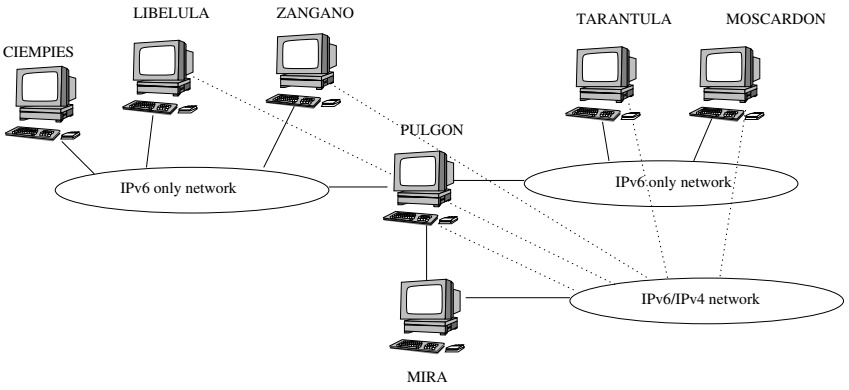

Figura 4.1: Máquinas utilizadas en el proyecto.

MIRA ha sido sustituido por un router comercial CISCO-7506 con soporte de IPv6, que actúa como *border-router* del servicio de IPv6 para los laboratorios del departamento de Ingeniería Telemática. La IOS con la que se ha estado trabajando durante la realización de este proyecto ha sido la 12.2(T), un beta con soporte de IPv6.

Todas las máquinas poseen al menos dos interfaces de red. Por un lado, los routers poseen direcciones IPv6 e IPv4 en la misma interfaz. Por otro lado, las máquinas de las subredes IPv6 nativas poseen en una interfaz una dirección IPv4 y en otra interfaz las direcciones IPv6 utilizas para la red IPv6 nativa. La interfaz IPv4 de estas máquinas ni siquiera posee una dirección IPv6 *link-local*, es decir, se tiene una tarjeta utilizando sólamente IPv4 y otra tarjeta utilizando sólamente IPv6. De esta forma, se obtienen los siguientes beneficios:

- Dado que existen multitud de aplicaciones todavía no migradas a IPv6, esta configuración nos permite utilizar dichas aplicaciones en dichas máquinas (sin necesidad de utilizar mecanismos de transición). El tráfico IPv4 fluye a través de una interfaz y el tráfico IPv6 fluye por la otra interfaz, pudiéndose realizar pruebas de rendimiento de ancho de banda sin que un tráfico interfiera con el otro.

- Al deshabilitar IPv6 en la interfaz conectada a la red general del departamento de Ingeniería Telemática , es posible separar completamente las direcciones IPv6 anunciadas a la red del departamento de las direcciones utilizadas en las subredes IPv6 nativas. De esta forma nos aseguramos que los paquetes de las máquinas IPv6 only se encaminan de forma adecuada.

- Es posible reiniciar las máquinas y trabajar con otro sistema operativo que no soporte IPv6 sin realizar ninguna configuración extra hardware o software.

A largo plazo se prevé eliminar la tarjeta IPv4 de estas máquinas, convirtiéndolas en estaciones de trabajo totalmente funcionales a través del nuevo protocolo de red, de una forma transparente para los usuarios.

Una subred funciona a 10 Mbits/seg a traves de un switch 3COM 280 y la otra a 100 Mbits/seg utilizando un Hub-Switch D-Link DFE-905DX. Esta configuración se ha pensado para realizar pruebas de calidad de servicio. La configuración se muestra en 4.2:

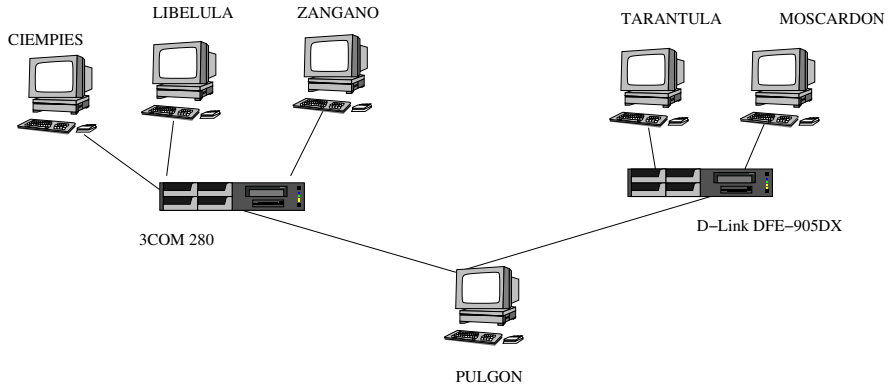

Figura 4.2: Máquinas utilizadas en el proyecto.

4.1.1 Configuración.

Como ya se ha comentado, en las máquinas de las subredes IPv6 se pretende separar por completo el protocolo IPv6 del protocolo IPv4, de forma que por una interfaz se pueda acceder y administrar las máquinas (usando IPv4) y por la otra interfaz se pueda trabajar con el protocolo IPv6 de forma exclusiva. Para ello, la autoconfiguración de direcciones ha sido deshabilitada en las máquinas "IPv6 only" en la interfaz IPv4 utilizanda el siguiente comando:

- Para máquinas *Linux*:

  ```
  echo '0' > /proc/sys/net/ipv6/conf/eth1/autoconf
  ```

- Para máquinas *FreeBSD*:

  ```
  sysctl -w net.inet6.ip6.auto_linklocal=0
  ```

En estas mismas interfaces se ha inhibido la escucha de *"router advertisement"* mediante los siguientes comandos:

- Para máquinas *Linux*:

  ```
  echo '0' > /proc/sys/net/ipv6/conf/eth1/accept_ra
  ```

- Para máquinas *FreeBSD*:

  ```
  sysctl -w net.inet6.ip6.accept_rtadv=0
  ```

Respecto a la inhibición de la escucha de *"router advertisements"* en las máquinas *FreeBSD*, dicha inhibición se realiza para todos las interfaces, no siendo posible activar/desactivar su escucha por interfaz como se hace en *Linux*. Por consiguiente, la única forma de evitar que las máquinas *FreeBSD* hagan caso omiso de los paquetes de *"router advertisements"* consiste en aplicar reglas de *"firewall"*. Para ello, basta con tener compilado el kernel con soporte de *"firewall"* para IPv6, y añadir la siguiente regla con el comando "ipfw6":

Si se quiere utilizar esta configuración de una manera estable, se añade en el archivo de configuración general "/etc/rc.conf" la etiqueta que define define el arranque del *"firewall"*, se define el tipo de *"firewall"* como "OPEN" y se añade la regla de inhibición de *"router advertisements"* en el archivo de configuración del *"firewall"* "/etc/rc.firewall6".

De esta forma, salvo que se les asigne una dirección IPv6 de forma explícita, dichas tarjetas actúan como interfaces IPv4 sólamente.

4.2 Direccionamiento.

Podemos dividir las direcciones utilizadas en la red en los siguientes grupos:

- Direcciones *link-local* autoconfiguradas.
- Direcciones globales de RedIris.
- Direcciones globales de un segundo proveedor.
- Direcciones 6to4.
- Direcciones anycast.

El hecho de tener más de una dirección global nos permite hacer pruebas de multi-homing, mientras que las direcciones 6to4 nos permiten acceder a islas IPv6 que utilicen el mismo tipo de direcciones. Durante una fase inicial, cuando no se disponía de direcciones globales delegadas, se utilizó el mecanismo 6to4 para acceder no sólo a otros sitios 6to4, sino también al resto de la red IPv6 utilizando como router por defecto un relay router de acceso público[1].

Las direcciones anycast, por otro lado, nos permiten construir escenarios de tolerancia a fallos utilizando técnicas de multi-homing.

4.2.1 Direcciones *link-local*.

Las direcciones *link-local* se utilizan en un único enlace y poseen el siguiente formato, mostrado en 4.3:

Figura 4.3: Formato de direcciones *link-local*.

Un ejemplo de este tipo de direcciones se puede observar en la siguiente salida de un "ifconfig":

[1] Concretamente 6to4.ipv6.microsoft.com.

```
fxp0: flags=8843<UP,BROADCAST,RUNNING,SIMPLEX,MULTICAST> mtu 1500
  inet 163.117.140.166 netmask 0xffffff00 broadcast 163.117.140.255
  inet6 fe80::290:27ff:fe86:93d%fxp0 prefixlen 64 scopeid 0x1
        ^^^^^^^^^^^^^^^^^^^^^^^^
  ether 00:90:27:86:09:3d
  media: autoselect (100baseTX <full-duplex>) status: active
  supported media: autoselect 100baseTX <full-duplex> 100baseTX
        10baseT/UTP <full-duplex> 10baseT/UTP
```

Estas direcciones están diseñadas para utilizarse dentro de un enlace con própositos de autoconfiguración de direcciones, descubrimiento de vecinos, o cuando no hay ningún router en la subred. Los routers no deben reenviar ningún paquete con dirección fuente o destino *link-local* hacia otros enlaces.

Gracias al proceso de autoconfiguración de direcciones característico de IPv6, todas las máquinas poseen una dirección *link-local* con los últimos 64 bits formados con el EUI-64 obtenido a partir de la dirección MAC de la tarjeta *ethernet*.

4.2.2 Direcciones de RedIris.

Las direcciones proporcionadas por RedIris y las direcciones proporcionadas por el segundo proveedor pertenecen al formato definido en [**?**] como *"Aggregatable Global Unicast Address"*, cuyo formato es el que se muestra a continuación, en la figura 4.4:

Figura 4.4: Formato de direcciones globales.

```
001         Format Prefix (3 bit) for Aggregatable Global
            Unicast Addresses
TLA ID      Top-Level Aggregation Identifier
RES         Reserved for future use
NLA ID      Next-Level Aggregation Identifier
SLA ID      Site-Level Aggregation Identifier
INTERFACE ID Interface Identifier
```

La entidad RedIris proporciona el siguiente subrango de direcciones para el departamento de Ingeniería Telemática, lugar donde se encuentran los laboratorios donde se ha realizado el presente proyecto fin de carrera:

- 2001:720:410:1000::/60

Con este subrango, quedan a disposición del departamento 16 subredes IPv6, de las que se utilizan en este proyecto las siguientes:

2001:720:410:1001::/64 prefijo anunciado a las máquinas del departamento de Ingeniería Telemática.

2001:720:410:100f::/64 prefijo anunciado en una subred IPv6 nativa.

2001:720:410:100e::/64 prefijo anunciado en la otra subred IPv6 nativa.

La conexión se realiza mediante un túnel configurado hacia una máquina Solaris del centro de cálculo que mediante una serie de túneles se encarga de conectarnos al exterior a través de la entidad RedIris.

4.2.3 Direcciones del segundo proveedor.

Este proveedor pone a nuestra disposición el siguiente subrango de sus propias direcciones. Esta delegación de direcciones se produce gracias a la participación de ambas entidades dentro del proyecto LONG [**?**].

- 3ffe:3328:6:ffff::/64

Como se puede observar, sólo se delega un prefijo, prefijo que será utilizado principalmente para pruebas de conectividad entre los socios de dicho proyecto.

La conexión se realiza mediante un túnel configurado con un router propiedad de este proveedor que se encarga, mediante una serie de túneles configurados, de encaminarnos hacia el exterior.

4.2.4 Direcciones 6to4.

Las direcciones 6to4 son un tipo especial de direcciones globales que comienzan siempre por 2002. El router MIRA actúa como un "6to4 router" o "6to4 border router", que se define de la siguiente forma:

Se utiliza el router MIRA para implementar un túnel 6to4, asignándose las siguientes direcciones IPv6:

2002:a375:8ca6:1f::/64 prefijo anunciado a las máquinas del Departamento de IT.

2002:a375:8ca6:1e::/64 prefijo anunciado a una subred IPv6 nativa.

2002:a375:8ca6:1d::/64 prefijo anunciado a la otra subred IPv6 nativa.

Nótese que "a375:8ca6" es la codificación hexadecimal de la dirección IPv4 de MIRA (163.117.140.166). También es importante ver que aunque se definen tres subredes, todas poseen el mismo prefijo de 48 bits. Esto significa que los paquetes con origen 2002:a375:8ba6::/48 van a ser respondidos mediante una encapsulación con dirección IPv4 de destino MIRA.

El mecanismo de túneles 6to4 se basa en [**?**] y se comenta su funcionamiento en este mismo trabajo, al describir los diferentes mecanismos de transición que se han utilizado.

4.2.5 Direcciones anycast.

Una dirección anycast en IPv6 en la práctica puede ser cualquier dirección unicast que se encuentre repetida dentro de un mismo dominio administrativo. En [**?**] se define las direcciones IPv6 anycast de la siguiente forma:

Una dirección anycast es una dirección que se asigna a más de una interfaz (típicamente de distintos nodos), con la propiedad de que un paquete enviado a una dirección anycast se encamina a la interfaz más cercano que posea dicha dirección, de acuerdo con la medida de distancias del protocolo de encaminamiento utilizado.

En el momento de escribir este trabajo, las direcciones anycast presentan dos restricciones en cuanto a su uso:

- Una dirección anycast no debe ser utilizada como dirección origen de un paquete IPv6.
- Una dirección anycast no debe ser asignada a máquinas, sólo puede ser asignada a routers.

En nuestra red IPv6 de pruebas, definimos la siguiente dirección anycast.

- fec0::1/64

Esta dirección se asigna a ambos routers y como se puede observar, se ha utilizado una dirección site-local. Las direcciones site-local se han diseñado para direccionamiento dentro de un sitio y tienen definido el formato que se muestra en 4.5:

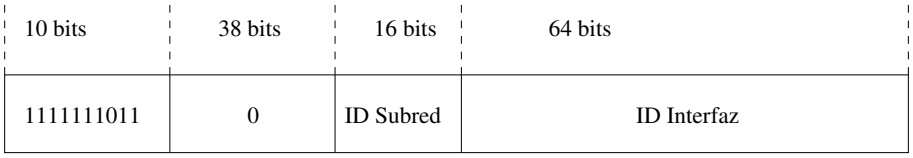

Figura 4.5: Formato de direcciones site-local.

4.3 *Router advertisements.*

Los routers envían periódicamente *"router advertisements"* o cuando reciben un paquete de *"router solicitation"*. Para más información sobre el funcionamiento del mecanismo se remite al lector a *"Neighbor Discovery for IP Version 6 (IPv6)"* [**?**]. Basta saber que mediante este mecanismo no es necesario recorrer una a una cada máquina de la red para asignarles una dirección IPv6, sino que gracias a este mecanismo es posible numerar una red IP y cambiar de direcciones de una forma dinámica. El formato del paquete de *"router advertisement"* se muestra en 4.6:

Type	Code			Checksum
Cur Hop Limit	M	O	Reserved	Router Lifetime
Reachable Time				
Retrans Timer				
Options...				

Figura 4.6: Formato del paquete de *"router advertisement"*.

Los campos son autoexplicativos, salvo el bit "M" ("Managed address configuration") que cuando está puesto a uno significa que la máquina debe usar además del mecanismo de autoconfiguración "stateless" un mecanismo administrado o "stateful".

El bit "O" ("Other stateful configuration") se utiliza para configurar otra información distinta de las direcciones utilizando el mecanismo "stateful".

En el campo opciones se especifican, entre otros parámetros, los prefijos con los que deben autoconfigurarse las máquinas.

En el diagrama 4.7 se pueden observar las direcciones que se anuncian a cada una de las subredes IPv6:

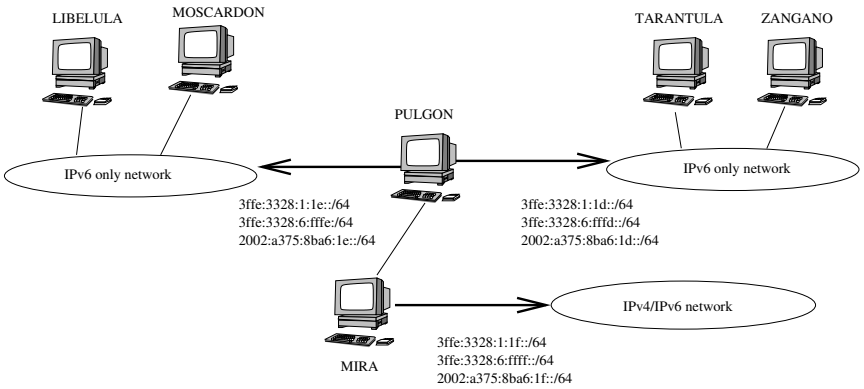

Figura 4.7: Direcciones anunciadas.

A continuación se presenta el archivo de configuración del *"router advertisement"* en el router PULGON, dicho archivo se encuentra en "/etc/rtadvd.conf".

```
rl1:\
:maxinterval#15:addrs#3:addr0="3ffe:3328:6:fffd::":prefixlen0#64:\
                vltime0#60:pltime0#30:\
               :addr1="2002:a375:8ba6:1d::":prefixlen1#64:\
                vltime1#60:pltime1#30\
               :addr2="3ffe:3328:1:1d::":prefixlen3#64:\
                vltime3#60:pltime3#30
fxp0:\
:maxinterval#15:addrs#3:addr0="3ffe:3328:6:fffe::":prefixlen0#64:\
                vltime0#60:pltime0#30:\
```

```
                :addr1="2002:a375:8ba6:1e::":prefixlen1#64:\
                 vltime1#60:pltime1#30\
                :addr2="3ffe:3328:1:1e::":prefixlen3#64:\
                 vltime3#60:pltime3#30
```

El archivo de configuración del *"router advertisement"* del router MIRA es el siguiente:

```
fxp0:\
:maxinterval#15:addrs#3:addr0="3ffe:3328:6:ffff::":prefixlen0#64:\
                 vltime0#60:pltime0#30:\
                :addr1="2002:a375:8ba6:1f::":prefixlen1#64:\
                 vltime1#60:pltime1#30\
                :addr2="3ffe:3328:1:1f::":prefixlen2#64:\
                 vltime2#60:pltime2#30
```

Como se puede observar, se utiliza un tiempo de vida de 60 segundos y un tiempo de vida "preferente" de 30 segundos, mientras que el tiempo máximo entre *"router advertisements"* es de 15 segundos. Una dirección preferente pasa a "caducada" cuando su tiempo de vida preferente expira. Una dirección caducada puede continuar usándose como dirección origen en las comunicaciones en curso, pero no debería utilizarse en nuevas comunicaciones si una dirección "alternativa" (no-caducada) está disponible y posee suficiente alcance. La capa IP debe continuar aceptando datagramas destinados a una dirección caducada ya que una dirección caducada es todavía una dirección válida para la interfaz.

Estos valores de *"timeout"*, que son muy bajos, permiten hacer pruebas de renumeración de redes IPv6, ya que en un minuto es posible migrar el conjunto de direcciones IPv6. Como contrapartida, además de consumir ancho de banda, es posible que las máquinas pierdan comunicaciones en curso si, por ejemplo, se cae alguna de las máquinas que anuncian los prefijos o si una dirección caduca sin haber terminado la comunicación que la estaba utilizando.

4.4 Encaminamiento.

Antes de comenzar a explicar los distintos mecanismos y protocolos utilizados para establecer comunicación entre las distintas subredes del proyecto, conviene definir los siguientes términos:

1. Encaminamiento estático (*"static routing"*): se refiere a destinos que son configurados manualmente en el router. La conectividad de la red no depende de si el destino está activo o no. El tráfico siempre se reenvía hacia el destino especificado.

2. Encaminamiento por defecto (*"default routing"*): tráfico hacia destinos que son desconocidos para el router son enviados utilizando esta ruta. El encaminamiento por defecto es la forma más fácil de encaminar para un dominio con un único punto de salida al exterior.

3. Encaminamiento dinámico (*"dynamic routing"*): se refiere a rutas que son aprendidas utilizando un protocolo de encaminamiento interno o externo. La conectividad de nuestra red dependerá del estado de los dominos vecinos. Si un destino esta "muerto", la ruta hacia él desaparecerá de nuestra tabla de rutas y el tráfico no se encaminará hacia dicho destino.

Aunque se han puesto de moda distintos protocolos de encaminamiento en los últimos años, no debe olvidarse nunca que las configuraciones más estables se basan en el encaminamiento estático y en el encaminamiento por defecto. Obligar a utilizar encaminamiento dinámico en situaciones que no lo requieren supone un desperdicio de ancho de banda, esfuerzo y dinero. No viene mal recordar en este punto el "principio del beso"[2]:

Keep It Simple, Stupid!

Este principio indica que la solución más simple disponible resulta ser frecuentemente la mejor solución.

Como ya hemos comentado, haciendo uso del mecanismo de autoconfiguración de IPv6 y enviando *"router advertisements"*, las máquinas de las subredes IPv6 nativas se autoconfiguran directamente utilizando como router por defecto al router PULGON, asimismo ocurre con las máquinas IPv6 del departamento de Ingeniería Telemática utilizando como router por defecto a MIRA. De esta forma, no es necesario utilizar ningún protocolo de encaminamiento para comunicar las subredes IPv6 nativas entre sí.

Sin embargo, para establecer comunicación exterior, o para que máquinas de las subredes IPv6 nativas se comuniquen con máquinas de la red del Departamento o viceversa, se debe utilizar como router intermedio a MIRA. Por consiguiente, se presentan dos opciones:

- Configurar estáticamente las rutas entre los routers MIRA y PULGON.
- Utilizar protocolos de encaminamiento con soporte para IPv6.

En una primera fase del proyecto, las rutas se establecieron estáticamente. Posteriormente, se han utilizado las siguientes herramientas:

- route6d: implementación de RIPng para *FreeBSD* realizada por KAME.
- ripngd: implementación de RIPng del software de encaminamiento de Zebra [**?**], que funciona tanto en máquinas *Linux* como en *FreeBSD*.
- BGP4+ : extensión multiprotocolo de BGP4, también llamado MBGP, se ha utilizado la implementación del software de encaminamiento de Zebra [**?**].

El proceso de encaminamiento se puede dividir en dos actividades básicas: determinación de caminos óptimos y el envío de paquetes hacia un determinado destino a través de la red. El reenvío de paquetes a través de la red es una actividad sencilla. La determinación del camino, por otro lado, puede ser muy compleja. En esta sección se presentan los distintos protocolos de encaminamiento que se han utilizado para determinar caminos óptimos, tanto dentro de nuestra red de pruebas como entre los socios del proyecto LONG [**?**], marco dentro del cual se han desarrollado la mayoria de las experiencias objeto del presente proyecto fin de carrera, tal como se ha comentado en alguna ocasión anterior.

[2] *"the KISS principle"*, definido en *"The New Hacker's Dictionary"*, también conocido como *"The Jargon's File"*. http://www.tuxedo.org/ esr/jargon/jargon.html

4.4.1 RIPng.

Generalmente, la mayoría de los protocolos de encaminamiento utilizados hoy en día se pueden clasificar en dos grandes grupos: los de "estado de enlace" y los de "vector de distancias".

RIP es un protocolo de vector de distancias, también denominados protocolos *"Bellman-Ford"*, en honor al inventor del algortimo distribuido utilizado para calcular los caminos más cortos. El término "vector de distancias" se deriva del hecho de que este protocolo incluye una lista de distancias (saltos u otras métricas) asociadas con cada destino incluido en un mensaje de encaminamiento.

Se utiliza un algoritmo de computación distribuido para que cada nodo calcule el mejor camino para llegar a cada destino. Después de seleccionar el mejor camino, el router envía los vectores de distancia a sus vecinos. En paralelo, sus vecinos también calculan el mejor camino para cada destino disponible y anuncian a sus vecinos su conectividad y métricas asociadas. Cuando llegan estos mensajes, el router puede determinar que existe un camino todavía mejor para alcanzar un determinado destino, actualizando su tabla.

La simplicidad inherente a los protocolos de vector de distancias ha hecho crecer su popularidad. Uno de sus mayores problemas es la lenta convergencia de estos protocolos. El término convergencia hace referencia al tiempo que tarda la red en su totalidad en darse cuenta de que una ruta a cambiado. A continuación se describe la configuración de RIPng utilizada en las pruebas de nuestra red. Para más información sobre RIP, se puede consultar [**?**].

Configuración.

Dentro de nuestra red, se ha probado RIPng para IPv6. Dichas pruebas han sido pruebas básicas de funcionamiento, interoperabilidad e instalación. No se ha tratado de hacer un estudio de rendimiento, por lo que las pruebas han consistido en ver que "funciona". Se ha probado a redistribuir las rutas de BGP4+ dentro de RIPng y viceversa, constatando que las implementaciones de ambos protocolos funcionan correctamente en su versión modificada para IPv6. A continuación, en la figura 4.8, se muestra la distribución de los routers internos utilizados para dichas pruebas dentro de nuestro sistema autónomo:

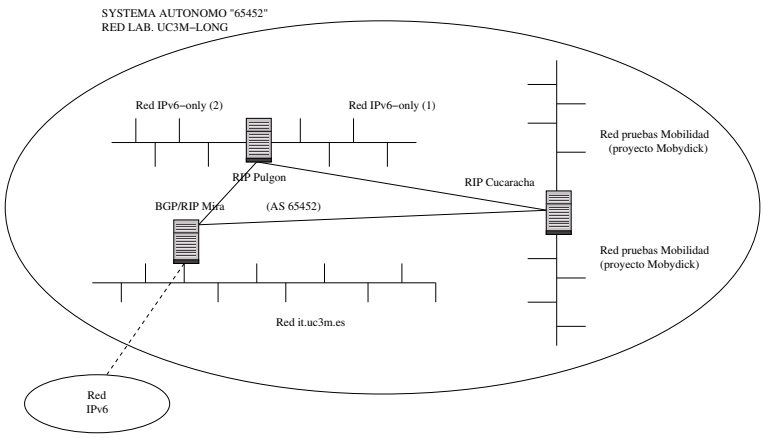

Figura 4.8: Routers dentro de nuestro sistema autónomo.

El software utilizado para habilitar RIPng ha sido el proporcionado por Zebra[3]. Zebra es un software de licencia GNU que gestiona varios protocolos de encaminamiento. Soporta BGP4, BGP4+, RIP, RIPng, OSPFv2 y OSPFv3. Está diseñado en base a una arquitectura modular, de tal forma que posee un proceso para cada protocolo y puede utilizar tecnología *"multithread"* si el sistema operativo lo permite. Además, cada módulo se puede actualizar de una forma independiente del resto.

Cada demonio gestiona un protocolo de encaminamiento distinto, y todos los demonios se comunican con un demonio central, denominado "zebrad". Este demonio es el encargado de realizar las modificaciones oportunas en la tabla de rutas del kernel. Por consiguiente, se necesita configurar por un lado el demonio de Zebra y por otro lado el demonio de RIPng, denominado "ripngd".

A continuación se muestra el archivo de configuración del demonio "zebrad":

```
!
! zebra sample configuration file
!
! $Id: zebra.conf.sample,v 1.14 1999/02/19 17:26:38 developer Exp $
!
hostname mira-router
password ‘‘password-ficticio’’
enable password ‘‘password-ficticio’’
!
! Interface’s description.
!
interface fxp0
interface gif0
!
log file /var/log/zebra.log
```

Y a continuación se presenta el archivo de configuración del demonio "ripngd":

```
!
! RIPngd sample configuration file
!
! $Id: ripngd.conf.sample,v 1.12 1999/02/19 17:35:39 developer Exp $
!
hostname mira-ripngd
password ‘‘password-ficticio’’
!
!
router zebra
!
router ripng
 network fxp0
!
 default-information originate
```

[3]Ver http://www.zebra.org.

```
 redistribute connected
 redistribute static
!
log stdout
```

En este ejemplo de configuración se puede observar la configuración típica de una topología RIP. Cada demonio de RIP identificado por "router ripng" en el archivo de configuración necesita al menos una interfaz por la cual realizar anuncios y recibir actualizaciones, interfaz que se especifica con el comando *"network"*. A continuación basta con especificar que se distribuyan las rutas que la máquina posee como "conectadas". De esta forma, todos los routers de la red intercambias sus subredes y se alcanza una configuración estable donde todas las redes se encuentran interconectadas. Si se desea redistribuir alguna ruta estática a través de RIP se puede realizar utilizando el comando *"redistribute static"* como se observa en el archivo de configuración anterior. Aunque no se muestra en dicho archivo, también se puede anunciar un prefijo determinado, aunque no esté configurado dentro del router, con la finalidad de atraer un determinado tráfico. En concreto esta técnica se ha utilizado dentro de la red de pruebas del laboratorio para atraer el tráfico del prefijo utilizado por el mecanismo de transición TRT. Dado que el mecanismo de transición TRT y el router se encuentran en la misma máquina, no se necesita asignar el prefijo utilizado por el TRT en ninguna interfaz de la máquina, tan sólo se necesita atraer el tráfijo dirigido a dicho prefijo. Una vez que el router recibe dicho tráfico, automáticamente el mecanismo TRT se encarga de reenviarlo por IPv4. Para atraer dicho prefijo se utiliza, dentro de *"router ripng"* el comando *"route"*.

Finalmente, basta con ejecutar los demonios de la siguiente forma:

```
/usr/local/sbin/zebra -df /usr/local/etc/zebra/zebra.conf
/usr/local/sbin/ripngd -df /usr/local/etc/zebra/ripngd.conf
```

Posteriormente, los demonios pueden ser accedidos utilizando una sesión de "telnet", para realizar modificaciones sobre la marcha, utilizando los siguientes comandos:

```
telnet ::1 2601  # para acceder al router Zebra
telnet ::1 2603  # para acceder al demonio de RIPng
```

4.4.2 BGP4+.

El protocolo BGP es un protocolo de encaminamiento exterior, o "entre dominios", utilizado para intercambiar información de encaminamiento entre sistemas autónomos. BGP pertenece a los protocolos denominados "vector de distancias", pero emplea un mecanismo adicional denominado *"path vector"* o "vector de caminos", que sirve para evitar bucles, como se verá más adelante. BGP funciona sobre TCP (puerto 179). Fue desarrollado para reemplazar a su predecesor, el ahora obsoleto EGP. Como con cualquier otro protocolo de encaminamiento, BGP mantiene tablas de rutas, transmite actualizaciones, y basa las decisiones de encaminamiento en métricas. La principal función de un sistema BGP no es otra que intercambiar información de conectividad de distintas redes. Esta información permite construir un grafo de conectividades entre sistemas autónomos[4] sin bucles y donde cada SA puede establecer su

[4]Abreviado como SA de aquí en adelante.

propia política de restricciones. Cada router BGP mantiene un tabla de rutas que enumera todos los caminos posibles para alcanzar una determinada red. El router no refresca esta tabla, sino que la información de encaminamiento recibida de un router vecino se almacenada y se modifica mediante actualizaciones incrementales. Cuando un router se conecta por primera vez a la red, los vecinos intercambian sus tablas de encaminamiento (completas) con el nuevo router. A continuación y de forma semejante, cuando alguna entrada de la tabla de rutas cambia, los routers vecinos envían la porción de la tabla que ha cambiado. Además, sólo se anuncian los caminos óptimos a una determinada subred. A continuación, el la figura 4.9 se puede observar el proceso de decisión y los distintos módulos que componen un sistema BGP:

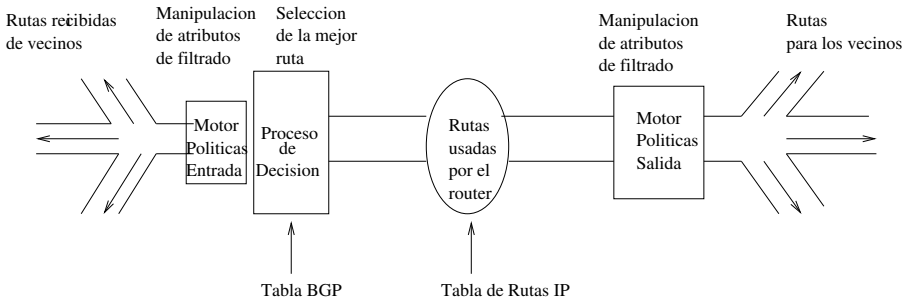

Figura 4.9: Esquema del proceso de encaminamiento en BGP.

En [**?**] sólamente se definen cuatro tipos de mensajes BGP, aquí vamos a resumirlos sin entrar en detalles de formato:

1. *Open*: resulta ser el primer mensaje que cada nodo se envia depués de haberse establecido la conexión TCP. Estos mensajes se confirman mendiante el envío de mensajes *keep-alive* y deben ser aceptados por el extremo remoto antes de que se pueda enviar cualquier otro tipo de mensajes.

2. *Update*: los mensajes de update se utilizan para enviar actualizaciones de la tabla de rutas a otros sistemas BGP. Los *"update messages"* también pueden borrar una o más rutas de la tabla de rutas y a la vez advertir otras nuevas rutas.

3. *Notification*: se envían cuando se detecta una condición de error. Estos mensajes se utilizan para cerrar una conexión activa y para informar sobre la razón de terminación de la conexión.

4. *Keep-Alive*: sirven para avisar a un extremo BGP que el sistema está activo. Estos mensajes se envían con la periodicidad suficiente como para eviar que las sesiones BGP caduquen.

BGP utiliza atributos del camino (*"PATH attributes"*) recorrido para proporcionar información sobre cada ruta. Un camino de SS.AA.[5] está formado por el número de cada SA que la ruta ha atravesado, lo que proporciona información para evitar bucles. Los atributos del camino pueden ser utilizados para distinguir entre grupos de rutas en función de preferencias administrativas. Existen muchos atributos (ver [**?**]) que se distribuyen en las siguientes categorías:

[5]Sistemas Autónomos.

- *Well-known mandatory*: atributo que debe estar incluido en un paquete *BGP Update* y debe ser reconocido por todas las implementaciones conformes al protocolo BGP. Si un atributo de este tipo no es enviado, la sesión se cierra y se notifica el fallo. Un ejemplo de estos atributos son el AS_PATH o el NEXT_HOP. Este último atributo normalmente indica la dirección del router que envía la ruta, salvo que se trate de un *"third party route"* [6].

- *Well-known discretinary*: atributo que es reconocido por todas las implementeciones conformes al protocolo BGP pero que puede ser o no ser enviado en un paquete *BGP Update*, como por ejemplo el atributo LOCAL_PREF.

- *Optional transitive*: no es necesario que sea reconocido por todas las implementaciones conformes al protocolo BGP. Si un atributo opcional no es reconocido, pero lleva el *flag* de "transitividad" puesto a uno, significa que dicha implementación de BGP debe aceptar dicho atributo y pasarlo a otros vecinos BGPs.

- *Optional nontransitive*: cuando un atributo opcional con el flag de transitividad puesto a cero no es reconocido por una implementación BGP, dicho atributo debe ser silenciosamente descartado y no reenviado hacia otros vecinos BGPs.

BGP soporta dos tipos básicos de sesiones entre vecinos: internas (o IBGP) y externas (o EBGP). Las sesiones internas se establecen entre routers pertenecientes a un mismo SA. Las rutas recibidas de un SA interno no llevan el número del SA que las envía, a diferencia de lo que ocurre con las sesiones externas, donde cada router añade al camino enviado su propio número de SA. Además, BGP debe ser capaz de establecer comunicación con cualquier IGP (Interior Gateway Protocol) para poder transportar información BGP de forma exitosa a través del SA.

Las sesiones externas BGP pueden incluir una métrica, que BGP denomina MED (Multi-Exit Discriminator), junto con los atributos del camino recorrido. Se prefieren los valores más pequeños de MED. Esta métrica se utiliza para decidir sobre un conjunto de rutas con igual longitud.

Las sesiones internas BGP llevan al menos una métrica en sus atributos de camino, que BGP denomina *Local_Pref*. Se seleccionan las rutas con valores más altos de *Local_Pref*. Sesiones internas pueden opcionalmente incluir una segunda métrica, el MED, almacenada para sesiones externas.

Para evitar bucles de información dentro de un SA, BGP no reenvía a vecinos internos rutas que se han aprendido de otros vecinos BGP internos. De esta forma, resulta muy importante mantener una malla de comunicación entre todos los IBGPs de un SA. Aunque esta conectividad en grafo completo no escala para redes grandes, se han descrito varios mecanismos para mitigarlo, como por ejemplo las *"confederations"* [**?**] y el mecanismo de *"route reflection"* [**?**].

Además, por definición, la conducta por defecto de BGP requiere que se encuetre sincronizado con el IGP antes de que BGP pueda anunciar rutas de tránsito (rutas que tienen

[6] *"third party route"* es una forma especial de anuncio de rutas a los vecinos externos. En particular, cualquier anuncio que tiene como atributo de siguiente salto una dirección IP diferente de la dirección IP del router que envía el anuncio se denomina *"third party"*. Un anuncio de *"third party"* tiene sentido cuando el siguiente salto que se anuncia se encuentra en la red del vecino con el que se está estableciendo la comunicación.

un origen y destino distinto del propio SA) a SA externos. Por ejemplo, si un nodo IBGP anunciara una ruta a un vecino externo antes de que todos los routers del SA hubieran aprendido dicha ruta utilizando IGP, el SA podría recibir tráfico hacia destinos que algunos de los routers del SA no sabrían encaminar. Siempre que un router BGP recibe una actualización de un vecino IBGP sobre un destino, el router trata de verificar su conectividad interna antes de anunciar dicha ruta a otros vecinos BGP externos. Para ello, comprueba que posee una ruta para alcanzar el siguiente salto y después comprueba que el prefijo del destino se encuentra en la tabla de rutas del IGP. Este mecanismo supone la inyección de rutas BGP dentro del IGP, lo cual resulta muy costoso, sobre todo en términos de escalabilidad de protocolos IGP, porque los protocolos IGPs no están diseñados para soportar un número alto de rutas. Por ello, la mayoria de las implementaciones BGP permiten obviar este comportamiento, como se verá más adelante.

La negociación con el vecino BGP se produce pasando por diferentes estados antes de que la conexión se establezca por completo. En la página 129 se muestra un diagrama de transición de estados que subraya los eventos más importantes que se producen, junto con una reseña que hace referencia a los mensajes BGP que se intercambian al transitar entre cada estado. Se trata de una máquina de estados finitos, que se encuentra completamente especificada en la correspondiente RFC, y que se comenta de una forma simplificada a continuación.

El diagrama de transición de estados de un sistema BGP es el que se muestra en la figura 4.10 de la página 129:

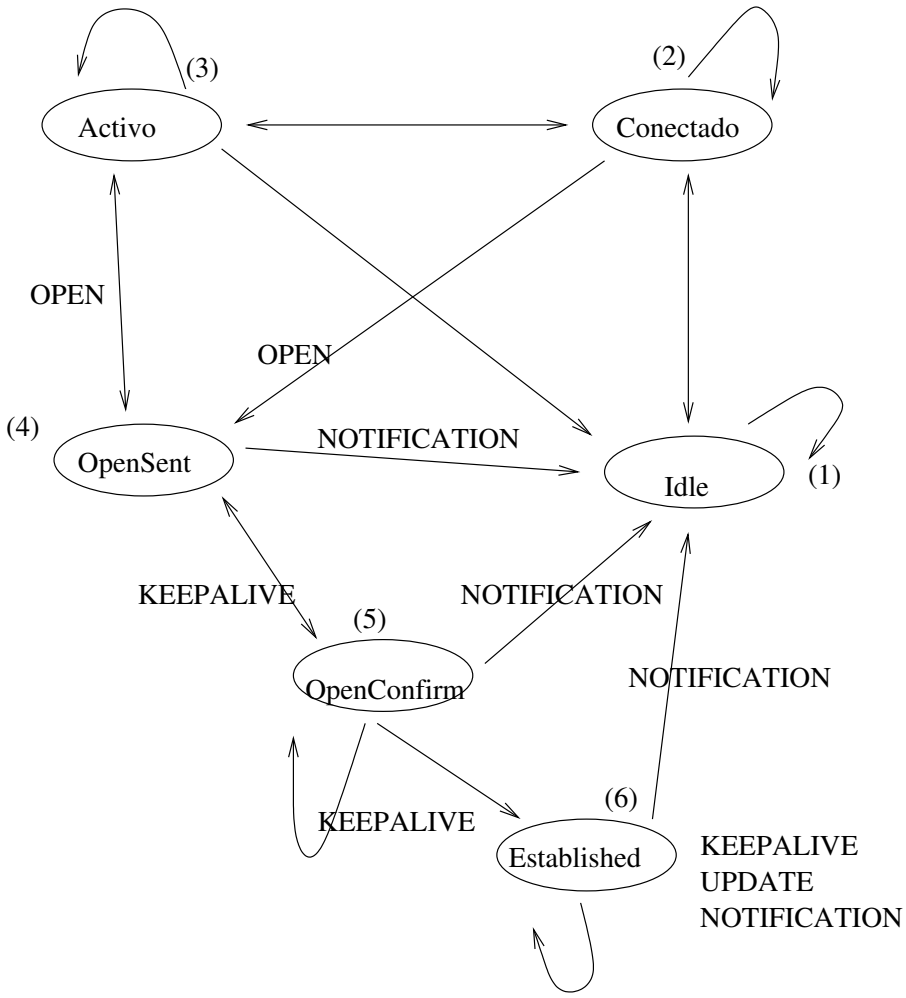

Figura 4.10: Diagrama de transición de estados en BGP.

Como se observa en el diagrama, cuando un demonio BGP se ejecuta, se transita entre los siguientes estados:

1. *Idle*: este es el primer estado de la conexión. BGP está esperando por un evento de comienzo. Este evento se provoca a través de la configuración manual del router o reseteando una configuración ya existente. Después de este evento, BGP inicializa sus recursos, comienza el establecimiento de la conexión BGP contra el vecino especificado y se pone a la escucha para recibir conexiones iniciadas por dicho vecino. A continuación, BGP transita al estado *"Connect"*. En caso de errores, BGP se mantiene en el estado *"Idle"*.

2. *Connect*: BGP está esperando a que se complete la conexión de transporte. Si la conexión de transporte se establece con éxito, se transita al estado *"OpenSent"* y se envía un mensaje BGP de OPEN. Si la conexión no puede realizarse, se transita al estado *"Active"*. Se reintenta varias veces la conexión, según indica el temporizador

"ConnectRetry". Si el temporizador expira, se reintenta la conexión manteniendose en el estado *"Connect"*. En caso de cualquier otro evento, se transita al estado *"Idle"*.

3. *Active*: BGP intenta comunicarse con el vecino. De forma semejante al estado anterior, salvo que ahora BGP continúa escuchando conexiones que pueden ser iniciadas desde un vecino remoto. Si expira el temporizador *"ConnectRetry"*, se transita al estado *"Connect"*. En caso de cualquier otro evento, se transita al estado *"Idle"* y si la conexión TCP se puede establecer, como en el caso anterior, se transitará al estado *"OpenSent"* y se enviará el mensaje BGP de OPEN. En general, un comportamiento oscilatorio entre los estados *"Connect"* y *"Active"* indica que algo esta mal en la conexión TCP.

4. *OpenSent*: BGP se encuentra en este estado esperando la recepción de un mensaje de OPEN enviado por su vecino. En caso de error (SA incorrecto o versión del protocolo incompatible), se envía un mensaje de NOTIFICATION y se transita al estado *"Idle"*. Si no se produce ningún error, BGP comienza el envío de mensajes KEEPALIVE y reinicia el contador KEEPALIVE, transitando al estado *"OpenConfirm"*. Cuando la conexión TCP se pierde, se transita al estado *"Active"*. En cualquier otro caso, BGP envía un mensaje de NOTIFICATION explicando el error y transita al estado *"Idle"*.

5. *OpenConfirm*: BGP está esperando un mensaje de *keep-alive*. Si se recibe dicho mensaje, se transita al estado *"Established"*, y la negociación con el vecino se da por terminada. Si se recibe un mensaje de NOTIFICATION o algún otro evento, se transita al estado *"Idle"*. Mientras que no se reciba el KEEPALIVE, BGP enviará sus propios mensajes KEEPALIVE de forma periódica según indique el contador *"keepalive"*.

6. *Established*: Es en este estado cuando BGP comienza con el intercambio de mensajes UPDATE. En caso de cualquier tipo de error (desconexión TCP, parada abrupta por parte del administrador, fallo en la recepción de KEEPALIVES, etc), se transitará al estado *"Idle"*.

Configuración.

La configuración de los routers frontera dentro del proyecto LONG [**?**] es la que se muestra a continuación, en la figura 4.11:

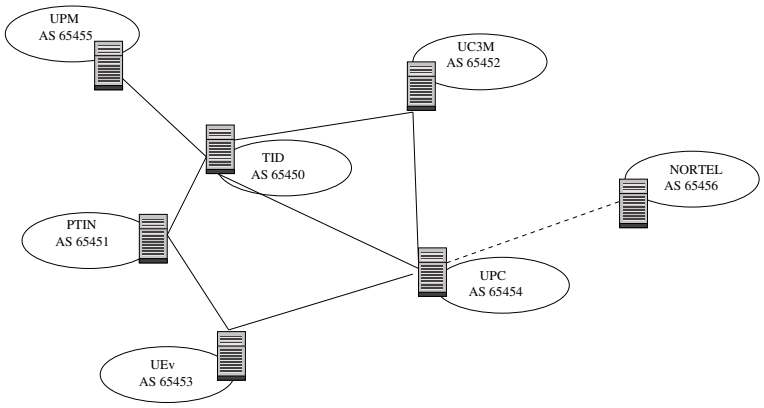

Figura 4.11: Asignación de números de sistemás autónomos a los socios.

El software utilizado para BGP4+ ha sido el proporcionado por Zebra[7]. Zebra es un software de licencia GNU que gestiona varios protocolos de encaminamiento. Soporta BGP4, BGP4+, RIP, RIPng, OSPFv2 y OSPFv3. Está diseñado en base a una arquitectura modular, de tal forma que posee un proceso para cada protocolo y puede utilizar tecnología *"multithread"* si el sistema operativo lo permite. Además, cada módulo se puede actualizar de una forma independiente del resto.

Los diferentes demonios se comunican con el demonio "zebrad", que es el encargado de modificar la tabla de rutas del kernel. De esta forma, se necesita configurar por un lado el demonio de BGP4+, denominado "bgpd" y por otro lado el demonio de Zebra.

A continuación se muestra el archivo de configuración del demonio "zebrad". En el archivo, sólamente se indican las interfaces que Zebra debe tener en cuenta:

```
!
! zebra sample configuration file
!
! $Id: zebra.conf.sample,v 1.14 1999/02/19 17:26:38 developer Exp $
!
hostname mira-router
password ‘‘password-ficticio’’
enable password ‘‘password-ficitio’’
!
! Interface’s description.
!
interface fxp0
interface gif0
!
log file /var/log/zebra.log
```

El archivo de configuración de "bgpd" para el router MIRA es el que se presenta a continuación:

```
!
! BGPd sample configuratin file
!
! $Id: bgpd.conf.sample,v 1.19 1999/02/19 17:17:27 developer Exp $
!
hostname mira-bgpd
password ‘‘passwd-ficticio’’
enable password ‘‘passwd-ficticio’’
!
!bgp mulitple-instance
!
! this means that routes go through zebra and into the kernel
!
router zebra
```

[7] Ver http://www.zebra.org.

```
!
!
router bgp 65452
 no synchronization
 bgp router-id 163.117.140.166
 no bgp default ipv4-unicast
 ipv6 bgp neighbor 3ffe:3328:6:3::fff0:1 remote-as 65450
 ipv6 bgp neighbor 3ffe:3328:6:3::fff0:1 next-hop-self
 ipv6 bgp neighbor 3ffe:3326:3:916::ffff:fff1 remote-as 65454
 ipv6 bgp neighbor 3ffe:3326:3:916::ffff:fff1 next-hop-self
!
 address-family ipv6
  network 3ffe:3328:6:fff0::/60
  neighbor 3ffe:3328:6:3::fff0:1 activate
  neighbor 3ffe:3326:3:916::ffff:fff1 activate
 exit-address-family
!
 ipv6 access-list all permit any
!
log stdout
```

A continuación se describen los comandos más característicos del archivo de configuración. Comandos específicos del sotfware zebra no se describen en este apartado y pueden consultarse en [**?**].

- El comando *"neighbor"* se utiliza para establecer la dirección del router remoto con el que se establece comunicación BGP.
- El comando *"router ID"* se utiliza para identificar al router a nivel IP, se utiliza una dirección IPv4 para asegurar su unicidad.
- El comando *"no bgp default ipv4-unicast"* se utiliza para inhibir el envío por defecto de información IPv4 cuando se establecen sesiones con los vecinos.
- El comando *"network"* se utiliza para anunciar la red a los vecinos BGP. La red que se anuncia debe estar en la tabla de rutas IP.
- El modificador *"next-hop-self"* asegura que el siguiente salto siempre contiene la dirección del router que realiza el Update.
- El comando *"no synchronization"* sirve para inhibir la sincronización con el IGP, tal como se ha explicado anteriormente.

Finalmente, basta con ejecutar los demonios de la siguiente forma:

```
/usr/local/sbin/zebra -df /usr/local/etc/zebra/zebra.conf
/usr/local/sbin/bgpd -df /usr/local/etc/zebra/bgpd.conf
```

Se puede acceder al terminal del router Zebra o del protocolo BGP4+, utilizando los siguientes comandos:

```
telnet ::1 2601  # para acceder al router Zebra
telnet ::1 2605  # para acceder al demonio de BGP4+
```

Desde el terminal, se pueden configurar los respectivos demonios utilizando un interface muy semejante al proporcionado por los routers CISCO.

4.5 *6bone.*

El *6bone* es una red de pruebas IPv6 que nace dentro del proyecto IPng del IETF que pretende reemplazar eventualmente al actual protocolo IPv4. El *6bone* comenzó como una red virtual utilizando IPv4 para transportar IPv6 y poco a poco se está migrando hacia enlaces IPv6 nativos. El objetivo inicial del *6bone* era probar estándares e implementaciones de IPv6, mientras que el objetivo actual consiste en probar mecanismos de transición y procedimientos operacionales. El *6bone* funciona utilizando el subrango de direcciones definido en *"IPv6 Testing Address Allocation"* [**?**]. Un esquema general de esta red se puede observar en 4.12:

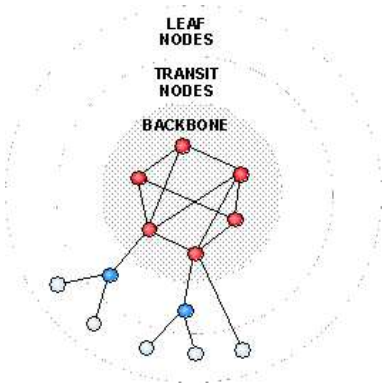

Figura 4.12: Estructura de la red del *6bone*.

Los nodos del backbone actuan como *"top level aggregators"* experimentales (también llamados "pTLAs") y son responsables de asignar las direcciones IPv6 de forma que se establezca una red jerárquica de direcciones capaz de asegurar la "agregación" de la información de encaminamiento.

El *6bone* está abierto a cualquier organización que desee ganar experiencia sobre el nuevo protocolo. En la página web http://www.6bone.net se explican los pasos que se deben realizar para formar parte de esta red experimental.

En particular, para adquirir un subrango de direcciones IPv6, se debe tener en cuenta que las direcciones IPv6 direccionables globalmente presentan tres niveles de jerarquía[**?**] como se observa en la figura 4.13:

Existe en primer lugar una topología pública, definida por un prefijo de tamaño /48, una topología de sitio y un identificador de interfaz generado de forma automática.

Ahora bien, la topología pública presenta dos o más niveles jerárquicos, dividiéndose en TLAs, NLAs y SLAs. Hasta la fecha, existen dos tipos de prefijos definidos por la IANA:

- 3ffe::/16, utilizado para el *6bone*. Los TLAs en este caso se denominan pseudo-TLAs y se asignan mediante un proceso definido dentro de la comunidad del *6bone*.

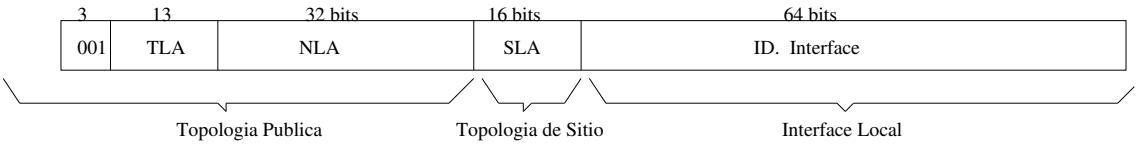

Figura 4.13: Direcciones IPv6 basadas en la agregación.

- 2001::/16, utilizada para la progresiva introducción del nuevo protocolo IPv6. Los TLAs se denominan en este caso sub-TLAs y se asignan a través de las entidades de registro internacionales o RIRs, siguiendo una política determinada.

Según este esquema, para conseguir una dirección IPv6 del *6bone*, se requiere ser un usuario final de un pseudo-TLA existente, quien delegará un prefijo de tamaño /48. También se puede solicitar ser un pseudo-TLA si se quiere actuar como un ISP.

Sin embargo, para adquirir una dirección IPv6 de la actual red IPv6, se necesita contactar con un sub-TLA. Si se quiere actuar como un sub-TLA, se deberán complir los requisitos definidos por la correspondiente entidad de registro.

Existe además un tercer método de adquisión de direccionamiento IPv6 global, para formar parte de la actual red IPv6, que consiste en la utilización del mecanismo de transición 6to4. Para más información sobre este mecanismo, se remite al lector al apartado referente a mecanismos de transición dentro de este mismo proyecto fin de carrera.

Resumiendo, Nuestra red de pruebas posee dos conexiones hacia el exterior:

- A través de RedIris: se trata de una conexión con la actual red IPv6 de producción, utilizando un prefijo del tipo "2001::"
- A través de nuestro segundo proveedor: se trata de una conexión con la actual red IPv6 "de pruebas" denominada *6bone*, utilizando un prefijo del tipo "3ffe::"

No obstante ambos mundos están interconectados, por lo que es posible acceder a máquinas situadas en el *6bone* desde la actual red IPv6 y viceversa.

4.6 *m6bone.*

La red *m6bone*, que se originó a finales del mes de Julio del año 2001, es una red de pruebas constituida sobre la actual red IPv6. El objetivo de esta red consiste en ofrecer un servicio multicast sobre IPv6 para todos aquellos interesados. Con más de veinticinco sitios conectado, se proporciona la posibilidad de transmitir eventos utilizando herramientas de vídeoconferencia comunes. Se puede encontrar más información en la página oficial del proyecto *"http://www.m6bone.net"*.

Además, esta red de pruebas permite aprender sobre el mecanismo de multicast en IPv6, el cual todavía se considera una función avanzada del protocolo, por lo que resulta interesante utilizar una red de pruebas para aprender su funcionamiento y mejorar las implementaciones existentes antes de introducir dicho mecanismo en la red IPv6 comercial.

A continuación se comentan brevemente las características del mecanismo de multicast en IPv6, entrando posteriormente a describir la configuración que se ha utilizado para formar parte de esta red experimental.

4.6.1 La paradoja multicast.

El mecanismo de multicast en IPv6 presenta a día de hoy la siguiente paradoja:

- Existen pocos routers que implementen multicast para IPv6: esto implica la necesidad de diseñar distintas topologías para las redes unicast y multicast.
- No existen tablas de encaminamiento específicas para multicast en IPv6: esto implica que resulta necesario utilizar la misma topología para las redes unicast y multicast.

El segundo punto guarda relación con el mecanismo de chequeo denominado *"reverse path forwarding"*, que se utiliza en la mayoría de las implementaciones de los algoritmos de encaminamiento multicast. Este mecanismo trata de asegurar que los paquetes multicast se encaminan correctamente utilizando la topologia multicast, para lo cual compara la dirección origen del paquete multicast recibido con la tabla de rutas. Si la interfaz de salida es distinto de la interfaz por el que ha llegado el paquete, el chequeo falla y dicho paquete no se tiene en cuenta.

La red *m6bone* tiene que utilizar de forma obligada (debido a que la mayoría de los routers IPv6 no implementan multicast) una topología diferente para el tráfico multicast y unicast. Cuando ocurre esto, el protocolo de encaminamiento multicast realiza *"reverse path forwarding"* utilizando la tabla de rutas unicast. Es por causa de esta paradoja que surge la necesidad de implementar una tabla de rutas multicast, a día de hoy inexistente. Para solventar el problema del *"reverse path forwarding"*, los routers multicast tienen que intercambiar sus tablas de encaminamiento unicast. Dentro de la red de pruebas *m6bone* se ha definido la utilización de "RIPng" para realizar este intercambio, aunque podría utilizarse cualquier otro protocolo de encaminamiento intra-dominio, como por ejemplo "OSPF".

Resumiendo, las acciones que se deben llevar a cabo para solventar la paradoja y poder establecer una red multicast (y unirse a la actual red *m6bone*) son las siguientes:

- Utilizar diferentes routers para el encaminamiento multicast y el encaminamiento unicast, salvo que todos los routers soporten multicast, lo cual es imposible a día de hoy, y más en IPv6.
- Como consecuencia del primer punto, se deben interconectar los routers multicast mediante túneles IPv6-en-IPv6.
- Anunciar los prefijos de cada participante utilizando "RIPng" entre los routers multicast.

4.6.2 Topología de red multicast en IPv6.

En el momento de escribir este proyecto fin de carrera no existe ningún protocolo multicast inter-dominio para IPv6, lo que significa que sólamente se puede manejar un único dominio multicast[8]. Para unirse a la red *m6bone* se han de realizar las siguientes tareas:

[8]Esto puede significar un impulso para el protocolo *"pim ssm"*.

- Dar de alta el prefijo que se desea utilizar en el router central: para ello se requiere enviar un correo electrónico a las personas encargadas de la gestión del proyecto *m6bone*.

- Establecer un túnel IPv6-en-IPv6 directamente con el router central o con algun otro participante: en nuestro caso se ha escogido la última opción, utilizando a la Universidad de Murcia como extremo del túnel. La Universidad de Murcia, a su vez, posee otro túnel con el router central.

- Ejectutar *"RIPng"* para IPv6 sobre la interfaz túnel, anunciando los prefijos que se desean utilizar para la red multicast: en nuestro caso se ha utilizado el software de *"zebra"*, que ya se estaba utilizado a nivel local en nuestra red IPv6. Otras opciones son el *"route6d"* que viene con la distribución estándar de *FreeBSD*.

- Ejecutar el demonio de encaminamiento multicast *"pim6sd"*: esta opción solo puede realizarse en *FreeBSD*, pues *Linux* en el momento de escribir este proyecto fin de carrera no ofrece este tipo de encaminamiento multicast.

Se han dado de alta en el router central los siguientes prefijos de nuestra red IPv6. Esto es necesario debido a que, como se ha comentado anteriormente, el router central filtra todos aquellos prefijos no dados de alta.

- 2001:720:410:1001::/64: prefijo utilizado de forma estable para dar servicio de IPv6 multicast en la red del laboratorio.

- 2001:720:410:100a::/64: prefijo utilizado para pruebas. Su disponibilidad dentro de la red del laboratorio es temporal.

La topología utilizada para unirse a esta red ha sido la que se muestra en la figura 4.14:

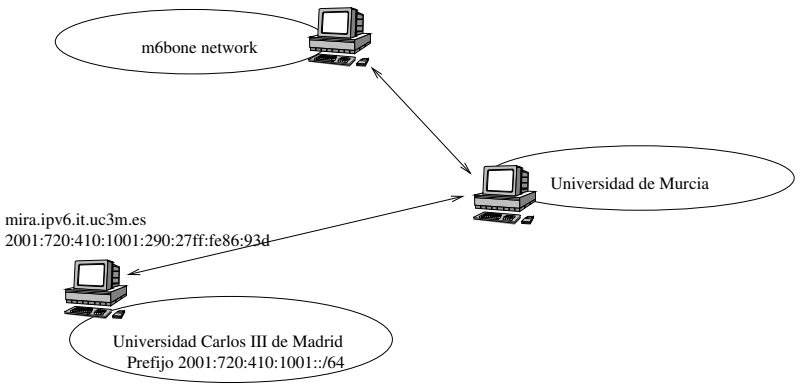

Figura 4.14: Conexión al *m6bone*.

De esta forma, cualquier usuario que se encuentre en las redes dadas de alta para su utilización dentro de la red de pruebas *m6bone* será capaz de unirse a sesiones multicast en IPv6.

También es posible participar en sesiones multicast IPv4 gracias a la existencia de traductores que redistribuyen los anuncios de sesión y realizan traducción de paquetes IPv6 a IPv4 y viceversa.

4.6.3 Aplicaciones multicast en IPv6.

La aplicación MGEN, desarrollada dentro de este proyecto fin de carrera, soporta multicast en IPv6. También se han utilizado las herramientas multicast proporcionadas por *"UCL Network and Multimedia Research Group"*[9]. Dichas herramientas están disponibles para *FreeBSD*, *Linux* y *Windows*, y las últimas versiones disponibles soportan IPv6.

A modo de ejemplo, en las figuras 4.15, 4.16 y 4.17 se presentan unas capturas de las herramientas *"sdr"*, *"vic"*, y *"rat"*, respectivamente. La primera de ellas se utiliza para recibir anuncios de sesiones multicast, y permite lanzar el resto de las aplicaciones de vídeo y audio. *"Vic"* es la herramienta más utilizada para transmitir y recibir vídeo multicast y *"rat"* permite transmitir y recibir audio.

Figura 4.15: *"Sesión Directory" (sdr).*

Figura 4.16: *"Video Conference" (vic).*

Dependiendo del sistema operativo utilizado, se deben realizar determinadas acciones para que estas herramientas funcionen correctamente. Brevemente se van a describir los pasos necesarios para que dichas aplicaciones funcionen correctamente en cada uno de los sistemas operativos que se han utilizado dentro de la red multicast.

[9]http://www-mice.cs.ucl.ac.uk/multimedia/software/

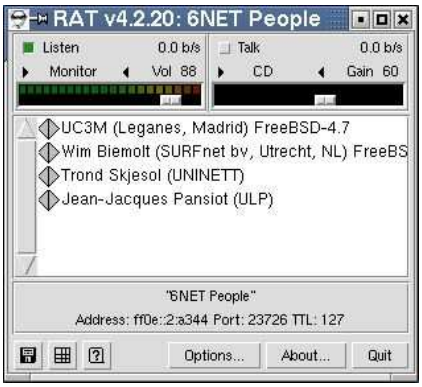

Figura 4.17: *"Robust Audio Tool" (rat).*

Configuración en *Windows*.

Para instalar la pila IPv6 en *Windows XP* basta con escribir desde un terminal *"ipv6 install"*. No obstante, para que las herramientas multicast funcionen correctamente, se debe instalar el *"Service Pack 1"*, o en caso contrario no será posible recibir ningún tipo de tráfico multicast IPv6.

Después de instalar las aplicaciones multicast en un mismo directorio, se debe configurar la variable de entorno "HOMEDIR" para que apunte hacia dicho directorio. Además, el nombre de la máquina tiene que resolverse tanto a IPv4 como a IPv6.

Respecto a *Windows 2000*, se debe instalar la correspondiente pila IPv6, que se puede encontrar en "http://msdn.microsoft.com/downloads/sdks/platform/tpipv6.asp". La variable "HOMEDIR" y el nombre de la máquina deben ajustarse como en el caso anterior. Además, en *Windows 2000*, se debe añadir una ruta multicast para todas las direcciones multicast a través de una determinada interfaz:

- *"ipv6 if"*: para obtener el número identificativo de la interfaz utilizado, llámemoslo "X".
- *"ipv6 rtu ff00::/8 X"*: donde "X" es la interfaz que se obtiene en el paso anterior.

Configuración en *Linux* y *FreeBSD*.

Para instalar las aplicaciones en *Linux* y *FreeBSD* no se requiere realizar ningún paso de configuración especial, salvo que el nombre de la máquina debe poder resolverse tanto a una dirección IPv4 como a una dirección IPv6.

Existe un pequeño problema en las herramientas precompiladas para *FreeBSD*, que impide la ejecución de *"vic"* directamente desde el *"sdr"*. Se trata de un error en la generación de los paramétros para el *"vic"*, que se puede solucionar facilmente con un *"script"* intermedio que elimine aquellos parámetros que *"vic"* no reconozca. Por ejemplo el siguiente *"script"* desarrollado en *"Perl"* elimina todos los parámetros accesorios excepto la dirección, puerto y la codificación del vídeo:

```
#!/usr/bin/perl
```

```
while ($param = shift)
{
  $linea = "$linea $param";
}

$linea =~ /^(.*)\/(.*)\/(.*)\/(.*)$/;
$output= "/$2/$3";
$aux=$1;
$aux=~ /^(.*) (.*)$/;

exec "./vic_bueno $2$output\n"
```

Además, si se quieren ejecutar las herramientas multicast en un router multicast *FreeBSD*, se necesita añadir la siguiente ruta para los paquetes multicast, que se encarga de encaminar los paquetes multicast hacia algún router multicast[10]:

```
ipv6 route ff00::/8 fe80::290:27ff:fe87:9bfc gif2
```

La sintáxis se corresponde al demonio de encaminamiento *"zebrad"*. La ruta especifica como siguiente salto el router multicast que se encuentra directamente conectado a través de la interfaz túnel "gif2".

4.7 Conclusiones.

La gestión de una red IPv6 y de los servicios asociados es semejante a la gestión realizada en IPv4. Aunque IPv6 parece llevar asociado una gestión más automática de las direcciones, se necesitan configurar mecanismos adicionales como DHCPv6 para obtener un mayor control sobre la red. También puede resultar necesario introducir mecanismos nuevos para gestionar la renumeración de grandes redes y se debe tener en cuenta la coexistencia e interoperabilidad con las máquinas IPv4 utilizando mecanismos de transición. Por consiguiente, lejos de disminuir el trabajo a los administradores de sistemas, IPv6 supone un coste añadido sobre la gestión de la red.

No existen diferencias substanciales entre el trabajo requerido para implantar una red IPv4 o una red IPv6. Hay que tener en cuenta que en IPv6 existen diferentes tipos de direcciones agrupadas por su ámbito de aplicación. Todavía no está claro el papel que jugarán las direcciones IPv6 tipo *"site local"*. Se prevé que las direcciones multicast *site-local* serán muy utilizadas[?]. El campo de ámbito de las direcciones IPv6 puede utilizarse para presentar distintos *"rendevouz points"*, uno para cada ámbito administrativo.

El encaminamiento unicast se realiza de forma semejante a IPv4, utilizando los mismos protocolos migrados a IPv6. Las máquinas no necesitan configurarse a nivel IP gracias al envío periódico de *"router advertisements"*, lo que supone una ventaja respecto al protocolo IPv4. El hecho de tener un largo período de coexistencia con IPv4 por delante supone tener que configurar mecanismos de transición adecuados para poder hacer la transición entre ambos protocolos de una forma transparente para los usuarios finales, los cuales no deberían tener que preocuparse de si están utilizando IPv4 o IPv6.

[10]Por supuesto, si el router *FreeBSD* ya posee una ruta por defecto hacia algún otro router multicast, esta ruta resulta innecesaria.

Capítulo 5

Servicios de aplicación instalados.

En este capítulo se describen los servicios avanzados que se han probado dentro de la red IPv6. Muchos servicios ya habían sido migrados a IPv6, como ssh, telnet, ftp, etc. Otros servicios han sido migrados por los socios del proyecto europeo LONG[1]. También se han probado herramientas para IPv4 haciendo uso de mecanismos de transición.

En primer lugar se describe el servicio de DNS. Fundamentalmente aspectos de instalación y configuración dentro de la red de pruebas. A continuación se describe el servidor web Apache, cómo aplicar el parche para IPv6 desarrollado por KAME y cómo configurarlo, al igual que aspectos más particulares referentes a la representación interna de las conexiones IPv6, y finalmente se describe la instalación, configuración y administración de la red de servidores de IRC gestionada entre varios socios del proyecto LONG.

En cada uno de los apartados que se presentan a continuación, se trata de hacer énfasis en los aspectos de configuración y gestión, no en aspectos de implementación. El código fuente de estas aplicaciones no ha sido modificado, y si en algún momento ha sido necesario modificarlo, se comenta de forma explícita.

Dentro de nuestra red de pruebas, se pretenden ofrecer los siguientes servicios:

- Servidor Web IPv6.
- Servicios de conexión con soporte IPv6 (telnet, ftp, ssh).
- Servidor DNSv6.
- Servicio de IRC.
- Servicio de pasarela www/irc.
- Servicio de NFS para IPv6.
- Tolerancia a fallos.

El objetivo no es otro que brindar un entorno IPv6 puro en el que los usuarios puedan trabajar de forma transparente, sin que noten ninguna diferencia. Además se pretenden ofrecer las siguientes aplicaciones para testear la red IPv6:

[1] servidor de streaming, servidor y cliente de ajedrez, pasarela web-irc o la plataforma de videoconferencia ISABEL, por citar sólo algunos.

- Una herramienta de generación de paquetes IPv6 que sea capaz de obtener medidas de rendimiento. Esta herramienta debe ser capaz de generar patrones de tráfico en tiempo real y debe dar soporte al máximo de cabeceras de extensión de IPv6. Además, debe funcionar tanto en *FreeBSD* como en *Linux*.

- Una herramienta que sea capaz de discernir si el extremo remoto de un túnel 6to4 funciona adecuadamente. Esta herramienta debe ser capaz de ejecutarse en máquinas sin soporte IPv6.

Estas herramientas se comentan en un capítulo posterior, dentro del apartado de migración de aplicaciones.

5.1 Servidor de DNS.

En este apartado se describe la configuración IPv6 del DNS. Se supone al lector con conocimientos sobre el funcionamiento del DNS. Una introducción al tema se puede consultar en [**?**] y en el apartado referente a la especificación del protocolo IPv6 en este mismo trabajo.

5.1.1 Introducción.

El DNS pretende proporcionar formas adecuadas para hacer referencia a sitios disponibles en Internet, ofreciendo a los usuarios un mecanismo fácil y fiable de identificación unívoca de sitios web, servidores de correo electrónico y de otros muchos servicios. Gracias al DNS, Internet ha conseguido alcanzar el objetivo de "medio de comunicación global".

Se puede definir el DNS como una base de datos globalmente distribuida, que mantiene (entre otras cosas) nombres de dominio. Uno de sus objetivos fundamentales consiste en proporcionar de manera fiable la misma respuesta a la misma consulta realizada desde cualquier posible origen en Internet. Para alcanzar este objetivo se necesita un espacio de nombres único, público y global derivado de una y sólo una raíz. Exiten varios gTLD (*"generic Top Level Domains"*: .com, .net, .edu, etc) y para aumentar la fiabilidad existen varios servidores *"root"* distribuidos de forma geográfica por todo el mundo. En la figura 5.1 de la página 142 se puede observar una estructura arborescente típica del sistema de DNS:

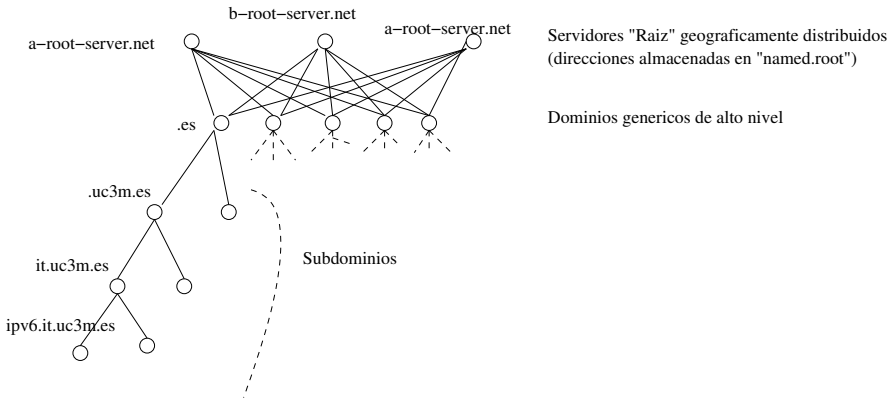

Figura 5.1: Arbol del DNS.

El DNS se utiliza principalmente para obtener una dirección a partir de un nombre de dominio. El mapeo de un nombre a una dirección IPv4 se realiza mediante registros tipo "A" del DNS mientras que en IPv6 se utilizan registros tipo "AAAA". También se han definido registros tipo "A6" y "DNAME", semejantes a los registros "NS" y "CNAME", pero estos tipos de registros presentan problemas y no han sido utilizados para configurar el sistema de DNS, a pesar de que la versión del programa servidor utilizada permite su definición.

5.1.2 Configuración.

Se utiliza "bind-9.2.1" como servidor de DNS. Nuestro servidor de nombres es *"authoritative"* para la zona "ipv6.it.uc3m.es" y el servidor primario es "mira.it.uc3m.es", como se observa en el registro SOA (*"Start Of Authority"*):

```
ipv6.it.uc3m.es.  IN  SOA  mira.it.uc3m.es.   jrh.it.uc3m.es. (
                        1       ;       Serial
                        300     ;       Refresh after 5 min.
                        900     ;       Retry after 15 min.
                        600     ;       Expire after 10 min.
                        600 )   ;       Minimum TTL of 10 min.
```

A continuación se enumeran las direcciones IPv6 de cada máquina del Departamento mediante registros tipo AAAA, como se puede observar en el siguiente extracto:

```
mira        IN    AAAA  3ffe:3328:6:ffff:290:27ff:fe86:93d
mosca       IN    AAAA  3ffe:3328:6:ffff:2c0:26ff:fea0:a9eb
...
```

Por último, en el archivo de configuración "named.conf", se especifican los ficheros que contienen la lista de servidores raíz ("named.root"), se especifica que se acepten conexiones tanto IPv4 como IPv6 y se activa la característica de reenvío hacia el servidor que nos ha delegado el dominio "ipv6", que posee dos direcciones IPv4, como se puede observar a continuación:

```
options {
        directory "/etc/namedb";

        forward only;

        forwarders {
                163.117.139.31;
                163.117.139.120;
        };

        listen-on { any; };
        listen-on-v6 { any; };
};

zone "." {
        type hint;
        file "named.root";
};

zone "ipv6.it.uc3m.es." {
        type master;
        file "db.ipv6";
};
```

La característica de reenvío es necesaria debido a que el sistema de DNS se encuentra detrás de un *firewall* que impide la comunicación directa con los servidores raíz, por lo que se ha optado en redirigir todas las peticiones al servidor "padre", el cual posee conectividad total.

5.1.3 Decisiones tomadas.

A la hora de establecer el servidor de nombres se han tomado las siguientes decisiones:

- Subdominio "ipv6" *versus* integración dentro del dominio IPv4: se ha seleccionado la primera opción debido a que no se puede gestionar/configurar el servidor de nombres del dominio "it.uc3m.es".
- Servidor de nombres en una máquina con *"dual stack" versus* servidor en una máquina *"IPv6 only"*: se ha seleccionado la primera opción por las siguientes razones:
  - El servidor de nombres que nos delega el subdominio necesita una dirección IPv4 (no soporta IPv6).
  - En este caso nuestro DNS puede hablar directamente con el servidor "padre" sin necesidad de utilizar mecanismos de transición.
- Redirección de consultas hacia el servidor "padre" *versus* consulta a los servidores *"root"*: se ha elegido la primera opción por motivos de eficiencia y para poder utilizar el me-

canismo de transición NAT-PT junto con el proxy de DNS[2]. Además, después de la introducción del *firewall* en el Departamento y la migración del servidor de DNS a la subred protegida, la comunicación entre el servidor de DNS y los servidores *"root"* no es posible. La redirección hacia el servidor "padre" se realiza en el archivo de configuración utilizando la etiqueta *"forwarders"*.

Además, para probar mecanismos de *"multi-homing"* se han establecido nombres de máquinas con más de una dirección IPv6 asociada, de forma que el DNS devuelva dinámicamente unas u otras.

5.2 Servidor de web "Apache".

El servidor web utilizado ha sido Apache 1.3.19. Esta versión del servidor no soporta IPv6, para ello es necesario aplicar el parche de KAME denominado "apache-1.3.19-v6-20010309a.diff.gz".

La versión 2.0, que soporta IPv6, en el momento de escribir este trabajo se encuentra en fase de desarrollo *Alfa* o inestable, por lo que se ha preferido no utilizarla.

Gracias a este parche y después de modificar ligeramente el archivo de configuración "httpd.conf", el servidor web permite conexiones IPv4 e IPv6. Las conexiones IPv4 aparecen en los ficheros de *log* y en la variable "REMOTE_ADDR" como direcciones IPv6 mapeadas a IPv4[3]. Además, el archivo ".htaccess" cambia ligeramente su sintaxis para tener en cuenta este nuevo tipo de conexiones, como se observa en el siguiente ejemplo:

```
Order deny,allow

Deny from all
Allow from 163.117.139.0/24
Allow from 3ffe:3328::/32
Allow from ::ffff:0/96
```

Dentro de la web del proyecto europeo LONG [?] se ha utilizado un pequeño *cgi* que discrimina entre conexiones IPv4 y conexiones IPv6, mostrando una página diferente en cada caso. Este *script* se basa en la variable de entorno "REMOTE_ADDR", que en el caso de conexiones IPv4 contendrá direcciones IPv4 mapeadas a IPv6, como se ha comentado anteriormente. El *script* es el siguiente:

```
#!/bin/sh

aux=‘echo $REMOTE_ADDR | grep ":"‘

if [ -z $aux ]; then
  ip6=0
else
  ip6=1
```

[2]Estos mecanismos se explican con detalle en el apartado "mecanismos de transición", dentro del presente proyecto fin de carrera.

[3] *"IPv4-mapped IPv6 address"* con el formato "::ffff:<dir IPv4>", ver [?]

```
fi

echo Content-type: text/html
echo
echo
cat ../scope1.html

if [ $ip6 = 1 ]; then
  echo "<tr align="center"><td><b>IPv6 Stack detected - Welcome aboard!</b></td></tr>"
fi

cat ../scope2.html
echo "<b>You are connected using: $REMOTE_ADDR</b>"
echo "<br><br><br>"
echo "</body>"
echo "</html>"

exit 0
```

La página de inicio se encuentra cortada en dos páginas html, "scope1.html" y "scope2.html". Cuando se detecta una conexión IPv6, se añade un mensaje entre las dos páginas y en cualquier caso, al final se muestra la dirección IP del cliente.

5.3 Red de servidores de IRC.

En este apartado se comenta la instalación de un conjunto de servidores de IRC dentro de la red del proyecto europeo LONG. El objetivo de esta red es probar distintos mecanismos de transición utilizando un servicio basado en TCP. También se ha utilizado el servicio para coordinar otros eventos como por ejemplo videoconferencias a través de "ISABEL"[4].

5.3.1 Introducción.

IRC significa *"Internet Relay Chat"*. Se trata de un programa originalmente escrito por *"Jarkko Oikarinen"* (jto@tolsun.oulu.fi) en 1988. Desde sus inicios en Finlandia, ha sido utilizado en más de 60 paises de todo el mundo. Fue diseñado para sustituir al programa *"talk"* pero ha llegado a ser mucho más importante. IRC es un sistema de conversación (o *chat*) multi-usuario, donde grupos de personas se reunen en "canales" (sitios virtuales, normalmente con un tema de conversación específico) para charlar en grupos o de forma privada.

[4]ISABEL es una herramienta de videoconferencia para *Linux*. Ver http://www.agoratechnologies.com/

5.3.2 Instalación y configuración de "ircd".

Se ha instalado la version de irc proporcionada por IRCNet[5]. El paquete instalado es "irc2.10.3.tar.gz". Se ha elegido la siguiente distribución de servidores, como se puede observar en 5.2:

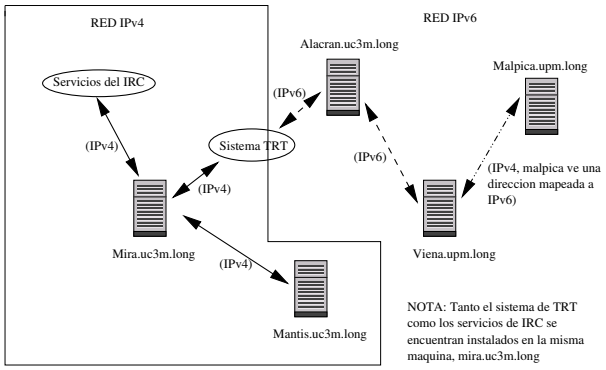

Figura 5.2: Red de servidores de IRC.

Cada servidor de IRC utiliza un nombre identificativo que no tiene por qué coincidir con el nombre de la máquina donde se ubica. Las flechan bidireccionales indican que la conexión a nivel de aplicación es bidireccional realizándose un proceso de autenticación en cada servidor antes de que la conexión pueda considerarse establecida. El servidor "alacran.uc3m.long" es un servidor que sólo acepta conexiones IPv6, por lo que resulta necesario utilizar un mecanismo de transición para establecer comunicación con el servidor IPv4 "mira.uc3m.long". Se ha elegido el mecanismo de transición TRT ya que únicamente se necesita traducir la conexión TCP contra un puerto determinado. Además, dado que tanto las conexiones entre los servidores y las conexiones entre cliente-servidor se realizan a través del mismo puerto, el mecanismo de transición también nos sirve para probar la conectividad de los clientes.

Dado que se trata de una red de servidores de IRC, se han definido una serie de reglas de configuración para ofrecer el mismo comportamiento de cara al usuario sin importar el servidor concreto al que se conecte. Dichas reglas guardan relación con el fichero de configuración denominado "config.h", que permite poner a punto las distintas capacidades y comportamientos que un servidor de IRC puede ofrecer. A continuación se describen las constantes que debe configurar cualquier servidor de IRC que desee unirse a nuestra red:

1. *"DEFAULT_INVISIBLE"*: cuando se encuentra definida, los usuarios que entren en el servidor obtendran automaticamente el modo "i". Este modo indica que dicho usuario es invisible para los comandos "/WHO" o "/NAMES" a menos que el usuario que los origina se encuentre en el mismo canal que el usuario buscado. Esta opción se encuentra normalmente desactivada en los servidores de IRC públicos, aunque dentro de nuestra red se encuentra habilitada.

2. *"LOCAL_KILL_ONLY"*: esta variable limita el comando "/KILL" a los clientes que se encuentran conectados al mismo servidor desde donde se origina el "/KILL". Esto significa que el rango de acción de los operadores es local al servidor al que se cuentran conectados, siendo más sencillo asignar responsabilidades en caso de problemas. Habilitada.

[5]http://www.ircnet.org

3. *"CLIENT_FLOOD"*: controla el número de bytes que el servidor permite enviar a un cliente sin su procesamiento antes de ser desconectado por inundación o *"flooding"*. El valor utilizado en nuestra red es el valor que viene por defecto.

4. *"UNDEF_ZIP_LINK"*: al definir esta constante se deshabilita la compresión de los canales de datos entre servidores. No se ha habilitado esta constante para poder trazar de una forma visual sencilla los comandos entre los servidores utilizando un simple "telnet".

5. *"CRYPT_OPER_PASSWORD"* y *"CRYPT_LINK_PASSWORD"*: constantes utilizadas para definir passwords encriptadas en el archivo de configuración. En nuestra red no han sido definidas, por lo que los archivos de configuración poseen passwords en claro.

Existan otras muchas variables que pueden configurarse en tiempo de compilación para moldear el servidor a gusto del administrador de la red, pero dentro de la red de servidores que hemos descrito estas variables son las únicas que hemos tenido en consideración a la hora de definir el servicio y su configuración.

Respecto al funcionamiento de los servidores de IRC, hay que tener en cuenta que dos servidores que establecen conexión se comportan de igual a igual, no existiendo relación de maestro-esclavo. Los servidores que forman parte de una red de IRC se clasifican en dos grupos:

- Concentradores: son servidores que poseen más de una conexión con otro servidor.
- Nodos hoja: son servidores que como mucho poseen una sola conexión con otro servidor.

El establecimiento y aceptación de conexiones entre servidores de IRC se configura utlizando las líneas "C/N" del archivo de configuración, que normalmente se encuentra en "/usr/local/etc/ircd.conf", como se observa a continuación:

```
c:163.117.139.166:passwd-1:alacran.ipv6::2
N:163.117.139.166:passwd-2:alacran.ipv6::2

c:163.117.139.160:passwd-3:mantis.ipv4:8005:2
N:163.117.139.160:passwd-4:mantis.ipv4::2
```

En este ejemplo se observa que todas las conexiones entre servidores se autentifican mediante el intercambio de una contraseña (que dependiendo de las opciones de compilación puede enviarse cifrada). La línea "C" especifica contra quién se establece la conexión y la línea "N" indica a quién se permite establecer una conexión con nosotros. Siempre deben ir en pares. En el archivo de configuración pueden definirse clases de conexiones (mediante las líneas "Y") donde se establece el número máximo de conexiones, el intervalo de comprobación del enlace y otros parámetros que quedan fuera del alcance de este proyecto. El número 2 (último parámetro de las líneas "C/N") indica la clase que se utiliza para controlar dicha conexión.

Para configurar un demonio como concentrador se necesita recompilar el paquete definiendo las siguientes constantes en el archivo "config.h"

```
#define HUB
#define MAXLINKS 10
```

Con la configuración de líneas "C/N" mostradas anteriormente "alacran" puede conectarse con "mira" y "mira" puede establecer comunicación con "mantis" pero "alacran" no podría establecer comunicación con "mantis" ni "mantis.ipv4" con "alacran.ipv6". Esto se debe a que en los nodos hojas se deben especificar los servidores que pueden establecer comunicación con uno mismo a través de concentradores, lo cual se realiza utilizando las líneas "H" del archivo de configuración del servidor. A continuación se muestra un ejemplo para la configuración de líneas H en "alacran":

```
# Linea H de 'alacran.ipv6'
H:*::mira.uc3m.long::
H:*::viena.upm.long::
```

Estas lineas vienen a decir que permitimos al concentrador "mira.uc3m.long" y al concentrador "viena.upm.long" que conecten con nosotros a cualquier (*) servidor que ellos tengan conectado.

Por último, se ha añadido una entrada al DNS bajo el dominio "ipv6.it.uc3m.es" con los siguientes valores:

```
irc         IN      A     163.117.139.166
irc         IN      A     163.117.139.160
irc         IN      AAAA  3ffe:3328:6:ffff:2c0:26ff:fea3:5df6
```

De esta forma, los nodos IPv4-only obtienen como respuesta un servidor IPv4 de forma alternativa, mientras que aquellos clientes con soporte[6] y conectividad IPv6 obtendrán la dirección IPv6 del servidor "alacran.ipv6".

Por defecto, el "ircd" se compila de tal forma que se muestran en la lista de canales algunos canales de depuración y de estadísticas del servidor, como los siguientes:

```
&AUTH             1
&SERVICES         1
...
```

Aunque estos canales no muestran información a usuarios normales, es preferible ocultarlos. Para ello, se debe modificar el código fuente y añadir en la función "setup_server_channels()" del archivo "channel.c" el modo secreto, quedando la línea de la siguiente forma:

```
smode = MODE_MODERATED|MODE_TOPICLIMIT|MODE_NOPRIVMSGS|
        MODE_ANONYMOUS|MODE_QUIET|MODE_SECRET;
```

Para finalizar, decir que se mantiene un *log* del canal "#long", que puede ser consultado utilizando la web del proyecto. Tambíen se han definido reglas para enganchar más servidores y reglas "de etiqueta". Para más información consultar http://www.uc3m.ist-long.com/software.html

[6] xchat-1.8.6 ya soporta IPv6.

5.3.3 Instalación y configuración de "ircservices"

Aunque la última versión del paquete es la 5.0, se ha utilizado la versión 4.3.3 de "ircservices" aplicando un parche para hacerlo compatible con la versión 2.10 de "ircd", ya que la versión 5.0 en el momento de escribir este trabajo no da soporte para el servidor "ircd".

Este paquete se compila de la forma habitual realizando "./configure" "./gmake" y "./gmake install". Se recomienda no instalarlo con permisos de superusuario por motivos de seguridad. Después de realizar esto, basta editar el fichero "services.conf", donde se debe especificar al menos la dirección y el puerto del servidor al que se va a conectar, junto con el/los usuarios que tienen permiso de *"root"* para utilizar los servicios que proporciona el programa (algo semejante al *"IRC op"* para el servidor de IRC) mediante la etiqueta "ServicesRoot". El resto de parámetros permiten un ajuste fino de las características de cada servicio.

Este paquete proporciona servicios añadidos mediante la utilización de una serie de clientes que permiten registrar canales y *nicks* de forma permanente, entre otras muchas cosas. Estos servicios se proporcionan enviando mensajes privados a los siguientes demonios:

- *NickServ*: servidor de registro de *nicks*.
- *ChanServ*: servidor de registro de canales.
- *MemoServ*: servidor de mensajes para usuarios *offline*.
- *HelpServ*: servidor de ayuda.

Utilizando estos demonios, se pueden registrar *nicks* y canales, fijar determinados modos de usuario y de canal, grabar la conversación de un canal e incluso fijar un mensaje de bienvenida en cada canal registrado, por citar sólo algunos ejemplos.

5.4 Pasarela web/IRC.

Se ha utilizado un programa desarrollado en Perl-5 y que se puede obtener descargándolo de http://cgiirc.sourceforge.net. La versión utilizada ha sido la 0.4.3, que no soporta IPv6, por lo que se ha modificado ligeramente el código para crear sockets IPv6. La modificación del código se ha visto restringida a la función encargada de abrir el *socket* y se ha podido sustituir el *socket* v4 por el *socket* v6, sin tener ninguna repercusión en el resto del código, por lo que la migración ha resultado bastante simple[7].

Este programa se basa en un *cgi* que permite conectarse a cualquier servidor de IRC y chatear a través de una página web. Con la proliferación de firewalls y NATs en las redes actuales, resulta un servicio muy útil puesto que la conexión se realiza contra el puerto 80 del servidor web, y posteriormente el *cgi* se encarga de establecer la conexión "tcp" con el servidor de irc, actuando como una pasarela.

Este programa establece las siguientes conexiones:

- Por un lado, se abre un *socket "unix"* para comunicarse entre el servidor IRC y el servidor web. Los datos recibidos por el *socket unix* son mostrados por el *cgi* y los datos enviados por el usuario son reenviados hacia el servidor de IRC.

[7] En el momento de escribir este trabajo, la versión 0.5 acaba de salir con soporte IPv6 y funciones mejoradas.

- Por otro lado, se establece una conexión tcp entre el servidor web y el servidor de IRC. Esta conexión puede ser IPv6 o IPv4, dependiendo de la dirección utilizada. Si un nombre se resuelve a más de una dirección, la primera dirección devuelta por el sistema de DNS es la que se utiliza.

Además, para establecer comunicaciones IPv6 ha sido necesario compilar un módulo de *sockets* para Perl, que puede obtenerse del CPAN[8]. El módulo en cuestión se denomina "Socket6-0.11". Este módulo soporta las funciones *"getaddrinfo"* y *"getnameinfo"* para poder proporcionar programación independiente del protocolo. En el caso concreto de IPv6, dicho módulo proporciona las definiciones de constantes necesarias para manejar dicho protocolo, y en concreto, la constante "AF_INET6".

La modificación del código es bastante simple, y se circunscribe a la función "connect_to_server" del archivo "nph-irc.cgi". El código modificado para permitir tanto direcciones IPv4 como direcciones IPv6 o nombres (realizando una resolución mediante DNS) es el que se muestra a continuación:

```
use Socket;
use Socket6;

my $family = -1;
my $socktype;
my $proto;
my $saddr;
my $canonname;

my @res = getaddrinfo($server, $port, AF_UNSPEC, SOCK_STREAM);
($family,$socktype,$proto,$saddr,$canonname ) = @res;
socket(Socket_handle, $family, $socktype, $proto);
connect(Socket_handle, $saddr) || error("Cannot connect to $server: $!");

$remote = \*Socket_handle;
```

La variable *"remote"* que antes almacenaba un *socket* conectado IPv4 posee ahora un *socket* IPv6.

Este servicio de acceso al IRC se ha mantenido en la página web del proyecto europeo LONG[**?**], junto con un *"log"* diario de las conversaciones del canal "#long". También se han definido una serie de reglas para aquellos servidores que quieran unirse a nuestra red. Para más información, consultar [**?**].

5.5 Servicio de NFS.

El servicio de NFS para IPv6 se ha probado satisfactoriamente en *FreeBSD-5.0* y en la versión estable *FreeBSD-4.5*. También se encuentra implementado en *OpenBSD*. utilizando un parche

[8] *Comprehensive Perl Archive Network*. Por defecto, Perl versión 5 no soporta *sockets* IPv6.

para proporcionar soporte para RPC en IPv6, desarrollado por *"Jean-Luc Richier"*, que se puede obtener en ftp://ftp.imag.fr/pub/ipv6/NFS/NFS_IPV6_FreeBSD4.5.tgz

No existen muchas implementaciones de NFS para IPv6 en el momento de escribir este trabajo, fundamentalmente por que NFS se considera un mecanismo utilizado sólo en redes de área local, por lo que puedes utilizar direcciones IPv4 privadas para ofrecerlo, por consiguiente su migración a IPv6 no se considera prioritaria.

La configuración se realiza exactamente igual que en IPv4, pero utilizando direcciones IPv6. A modo de ejemplo, se presenta un archivo de configuración típico:

```
/usr/home -maproot=root -network 2001:720:410:100e::/64
```

Para montar este directorio, basta con ejecutar:

```
mount mira.ipv6.it.uc3m.es:/usr/home /tmp
```

Dentro del capítulo dedicado a mecanismos de transición, se comenta la utilización del mecanismo NAT-PT para posibilitar que cliente de NFS IPv6 accedan a un directorio exportado por un cliente IPv4. Para que dicha configuración funcione en *FreeBSD*, resulta necesario modificar el archivo "/etc/netconfig". Para más información se remite al lector a las pruebas realizadas con el mecanimo de transición NAT-PT, en este mismo proyecto fin de carrera.

5.6 Conclusiones.

La instalación de los servicios no difiere esencialmente de los correspondientes para IPv4. Algunos programas requieren la aplicación de parches para soportar IPv6, pero en general puede decirse que actualmente se dispone de los mismos servicios básicos que pueden instalarse en una red IPv4. Aunque no se ha comentado en este capítulo, los mecanismos de transición establecidos dentro de una red IPv6 se consideran también servicios de aplicación. De esta forma, y al menos durante los primeros años de coexistencia con IPv4, resulta fundamental ofrecer mecanismos que posibiliten la conexión entre el mundo IPv6 y el mundo IPv4.

Algunos servicios son más complicados de ofrecer que otros. Por ejemplo, resulta sencillo instalar un servidor web Apache con soporte para IPv6 aplicando un parche, pero en cambio no existe un parche para el mismo servidor con soporte de SSL, por lo que ofrecer servicio de web y de SSL sobre IPv6 resulta casi imposible con la versión del servidor web utilizada. No obstante, dicho servicio de web más SSL se puede ofrecer a lás máquinas IPv6 utilizando el software en su versión original para IPv4 junto con algún mecanismo de transición, por ejemplo el mecanismo TRT.

Respecto al servicio de NFS, dado que NFS se considera un servicio de ámbito restringuido, su migración a IPv6 no es prioritaria, puesto que siempre se puede ofrecer utilizando direcciones IPv4 privadas. Es por ello que en el momento de escribir este trabajo solo se encuentra disponible una implementación para *FreeBSD-5.0*.

Respecto al servicio de IRC, se ha comprobado que resulta un servicio fiable, que permite varias posibilidades de conexión, dependiendo de la resolución de nombres del DNS, quien se encarga de conectar transparentemente a los usuarios utilizando IPv6 o IPv4. De acuerdo con

el funcionamiento del IRC explicado anteriormente, no importa el servidor al que te conectes, pues todos los servidores se encuentran interconectados ofreciendo al usuario una única vista de todos los canales y *nicks* existentes en la red. La versión del programa cliente más conocida es el denominado *"xchat"*. Dicho cliente sólamente intenta conectarse utilizando la primera dirección devuelta por el servidor de nombres.

Dentro del proyecto europeo LONG[**?**] hemos utilizado constantemente la red de servidores del IRC para comunicarnos en los momentos previos a las reuniones por videoconferencia. También se ha realizado una encuesta al resto de socios del proyecto preguntando sobre la calidad y el grado de usabilidad del servicio ofrecido, de la cual se han extraido las siguientes conclusiones:

- La configuración del servidor es igual de complicada que la configuración correspondiente en IPv4.
- El acceso de los clientes resulta idéntico al accesso con IPv4, habiéndose probado los siguientes clientes con soporte de IPv6:
  - "xchat" sobre *Linux/FreeBSD/Windows*: soporta IPv6 desde la version 1.8
  - "kvirc" sobre *Windows*: no puede realizar consultas AAAA ni A6 contra el DNS.
  - "irc" sobre *Linux/FreeBSD*: programa que viene con el paquete "ircd".
- Se han experimentado muchas separaciones entre los servidores (*"server splits"*) que han sido debidas a problemas de encaminamiento entre los servidores de los distintos socios del proyecto europeo LONG. Se trata, en cualquier caso, de proeblemas de conectividad ajenos al servicio ofrecido.
- El protocolo DCC (*"Direct Client Connection"*, muy extendido entre los clientes de *chat* más populares y que permite la transferencia de ficheros entre los clientes sin pasar por el servidor, no funciona entre clientes IPv4 y clientes IPv6, a menos que se extienda el protocolo para soportar el rango de direccionamiento de IPv6. Un cambio de tal naturaleza en el protocolo implicaría cambiar las implementaciones actuales de los clientes IPv4 e IPv6. Esto no parece que vaya a ocurrir a corto plazo debido fundamentalmente a que se trata de un servicio añadido que prestan los clientes sobre el servicio de IRC y debido también al enorme abanico de clientes existentes.
- El retardo o *"lag"* no ha sido apreciable en ningún momento, aunque es un resultado esperado puesto que la carga de usuarios de los distintos servidores ha sido muy baja.

Respecto a la pasarela www/irc, resulta bastante lenta y difícil de manejar en cuanto se quieren realizar acciones que cualesquiera otros clientes de IRC permiten realizar de una forma sencilla. No obstante, resulta útil debido a que el servicio "http" está disponible a través de la mayoría de los *"firewalls"*.

Por último, y por ello el más importante de los servicios que se han instalado, se encuentra el servicio de DNS que se ha mantenido durante toda la vida del proyecto europeo LONG[**?**]. Durante este tiempo se han delegado zonas utilizando IPv6 y se han mantenido más de 150 registros tipo "AAAA". Para no interferir con el servidor de nombres del Departamento de Ingeniería Telemática, se ha delegado el dominio "ipv6.it.uc3m.es" para poder hacer pruebas sin comprometer el servidor de nombres actualmente en producción. Esta situación ha provocado

que en algunos casos haya sido necesario añadir un registro tipo "A" dentro de este servidor, para configurar una entrada para el router ISATAP o para poder ejecutar algunos programas en IPv6[9].

Para terminar, queda por comentar que en este proyecto fin de carrera no se han probado todos los servicios que pueden resultar útiles en una red IP, por no ser el objetivo principal de este trabajo y también por falta de implementaciones suficientemente maduras, como ocurre en el caso del servicio de DHCPv6. No obstante, dado que el mecanismo de autoconfiguración *"statefull"* es un requisito del protocolo IPv6, se puede imaginar un escenario de red IPv6 nativo con DNS, DHCP, servidor de correo, NFS, servidor web y cualquier otro servicio que se pudiera pensar en instalar en una red IPv4.

[9]Por ejemplo, el programa de videoconferencia *"rat"* no funciona salvo que el nombre de la máquina se resuelva tanto a una dirección IPv6 como a una dirección IPv4, aunque sólo se quiera utilizar en IPv6. Este ejemplo se puede considerar además como un ejemplo de una mala o incorrecta migración de aplicaciones a IPv6.

Capítulo 6

Migración e implementación de aplicaciones.

En este apartado se presentan las aplicaciones desarrolladas en el marco del proyecto LONG [?]. Estas aplicaciones nos han proporcionado la oportunidad de manejar el API de IPv6 de una forma avanzada, así como hacer pruebas de rendimiento del nuevo protocolo IP, experimentar con mecanismos de transición y probar distintas cabeceras de extensión.

En primer lugar se comentan aspectos de implementación relacionados con la migración de la aplicación MGEN a IPv6. Se pretende que esta descripción sirva como guía para la realización de migraciones posteriores de otras herramientas.

El segundo programa, implementado en su totalidad, muestra una forma de programar la encapsulación de paquetes IPv6 dentro de paqutes IPv4. Se ha aplicado a la implementación de una herramienta que simula el comportamiento de una interfaz 6to4 mediante el envío de pings encapsulados, lo que permite comprobar si el extremo 6to4 remoto funciona correctamente.

Por último se presenta la migración de un servidor de ajedrez, y la migración de la interfaz gráfica asociada.

6.1 Herramienta MGENv6.

MGEN son las iniciales de *"Multi-Generator Toolset"*. Esta herramienta ha sido desarrollada por *"The Naval Research Laboratory"* [?] y proporciona flujos de paquetes UDP/IP multicast/unicast. La herramienta MGEN transmite y recibe (y graba) paquetes numerados y con *"time-stamp"*. Un análisis posterior del fichero de *log* permite obtener distintas medidas relacionadas con la habilidad de la red para soportar el tráfico inyectado, como pueden ser la tasa de pérdidas, retardo, variación del retardo (*"jitter"*), etc. La utilidad MGEN genera patrones de tráfico en tiempo real de tal forma que la red se puede someter a diferentes tipos de carga. Es posible crear archivos de configuración (*"scripts"*) que simulan distintos tipos de flujos. La herramienta Se presenta en versión línea de comandos y también con interfaz gráfica. El código fuente está disponible y el autor nos ha autorizado explícitamente a modificar su herramienta y a distribuirla libremente.

La herramienta MGEN se ha migrado a IPv6 y se ha restringido su funcionalidad en los siguientes aspectos:

- El código referente a RSVP no se ha soportado.
- La nueva herramienta sólo soporta IPv6.
- De las utilidades que acompañan a la herramienta para tratar posteriormente el fichero de *log*, sólo se ha soportado "mcalc".

Resumiendo, el nuevo paquete MGENv6 se distribuye con las siguientes herramientas:

- Mgen (*"Multi-GENerator"*): genera patrones de tráfico en tiempo real para destinos unicast/multicast según un *"script"* de configuración o por línea de comandos. Se pueden especificar tamaños de paquetes y velocidades de transmisión para flujos individuales, entre otros muchos parámetros.
- Drec (*"Dynamic-RECeiver"*): recibe y almacena el tráfico generado por la herramienta Mgen.
- Mcalc (*"Multi-CALCulator"*): examina el archivo de *log* creado por Drec y calcula estadísticas por flujo recibido.

A continuación se va a explicar en profundidad el funcionamiento interno de la herramienta migrada a IPv6. La herramienta se encuentra disponible dentro de la página web del proyecto europeo LONG [**?**].

6.1.1 Cómo compilar el paquete.

El paquete MGENv6 ha sido desarrollado para las siguientes versiones:

- *FreeBSD-4.3-RELEASE* con el kernel parcheado que proporciona el proyecto KAME ("kame-20010924-freebsd43-snap.tgz")
- *Linux-2.4.2* con el kernel parcheado que proporciona el proyecto USAGI ("20010917-snapshot").

Es posible que con núcleos más avanzados la herramienta no compile, debido al cambio de nombre de constantes o simplemente a la "caducidad" de alguna de las APIs que en el momento de escribir este trabajo se encuentran en fase de *"draft"*. Durante la duración del proyecto europeo LONG [**?**] se ha dado soporte técnico a la herramienta (tanto la versión IPv6 como la versión original IPv4) y se han realizado sucesivas mejoras en ambos programas[1].

Los pasos a seguir para compilar son los siguientes:

1. El kernel debe ser compilado con los parches proporcionados por KAME o USAGI, respectivamente. Además, se deben instalar los ficheros de cabecera (*.h) del kernel de dichas distribuciones.
2. Editar el fichero "Makefile" para seleccionar el sistema operativo correspondiente. Concretamente, se trata de seleccionar uno de los siguientes grupos de constantes, como se observa a continuación:

[1]Tanto para las versiones migradas a IPv6 como para la versión IPv4.

```
SYSTEM_CFLAGS="You must select your OS for correct compilation"
SYSTEM_PATH="You must select your OS for correct compilation"

#  IMPORTANT: Which system are you compiling for?
#  Select your Operating System here

#SYSTEM_CFLAGS = -D_FREEBSD -g
#SYSTEM_PATH = FreeBSD

#SYSTEM_CFLAGS = -D_LINUX -g
#SYSTEM_PATH = Linux
```

3. Ejecutar "make clean" y "make".

La aplicación se debe ejecutar con permisos de superadministrador. Esto resulta necesario para poder enviar algunas cabeceras de extensión. En versiones recientes de la herramienta, resulta necesario además para poder cambiar la prioridad el proceso y hacer la ejecución más eficiente.

Existe una versión gráfica que se distribuye con la librería *LessTif* tanto para *Linux* como para *FreeBSD*, pero su desarrollo ha sido discontinuado y no se recomienda su utilización.

El paquete se enlazan dinámicamente con la librería "libinet6.a", que da soporte a opciones todavía no implementadas en la libería "glibc" que se incluye en las distribuciones estándar para IPv6. Con el paquete MGEN se distribuye una versión compilada de esta librería para *Linux* y para *FreeBSD*. Su actualización está discontinuada y en caso de problemas de compilación se recomienda actualizar la librería[2].

En el apéndice se puede ver una descripción detallada del funcionamiento interno de MGEN y del código fuente.

[2]Desde su inclusión como parte del paquete MGEN, no se ha reportado ningún problema en relación con la librería.

6.1.2 Interfaz gráfica.

Mgen presenta una interfaz gráfica realizada utilizando *LessTif*, una versión libre de *Motif*. *Motif* es una interfaz gráfica de usuario basada en *"X Window System"*. *Motif* utiliza las funciones proporcionadas por las librerías "Xlib" y "Xt" para realizar su trabajo. Además de proporcionar *"widgets"* de propósito general, *LessTif* añade otras características que resultan de interés general para aplicaciones y usuarios, convirtiéndose en una interfaz de alto nivel para programar ventanas en C.

La interfaz gráfica de Mgen presenta la misma funcionalidad que la versión de línea de comando. Para interactuar mediante *"script"*, se utiliza la ventana de la figura 6.1:

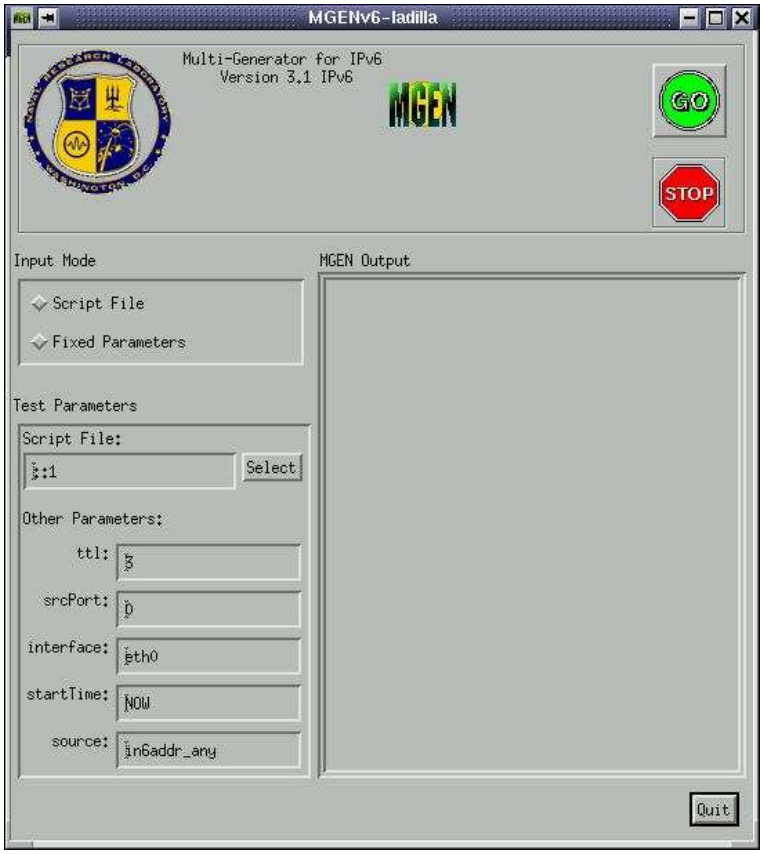

Figura 6.1: Interfaz gráfica de MGENv6.

La ventana que muestra las opciones de línea de comando es la que se muestra en 6.2:

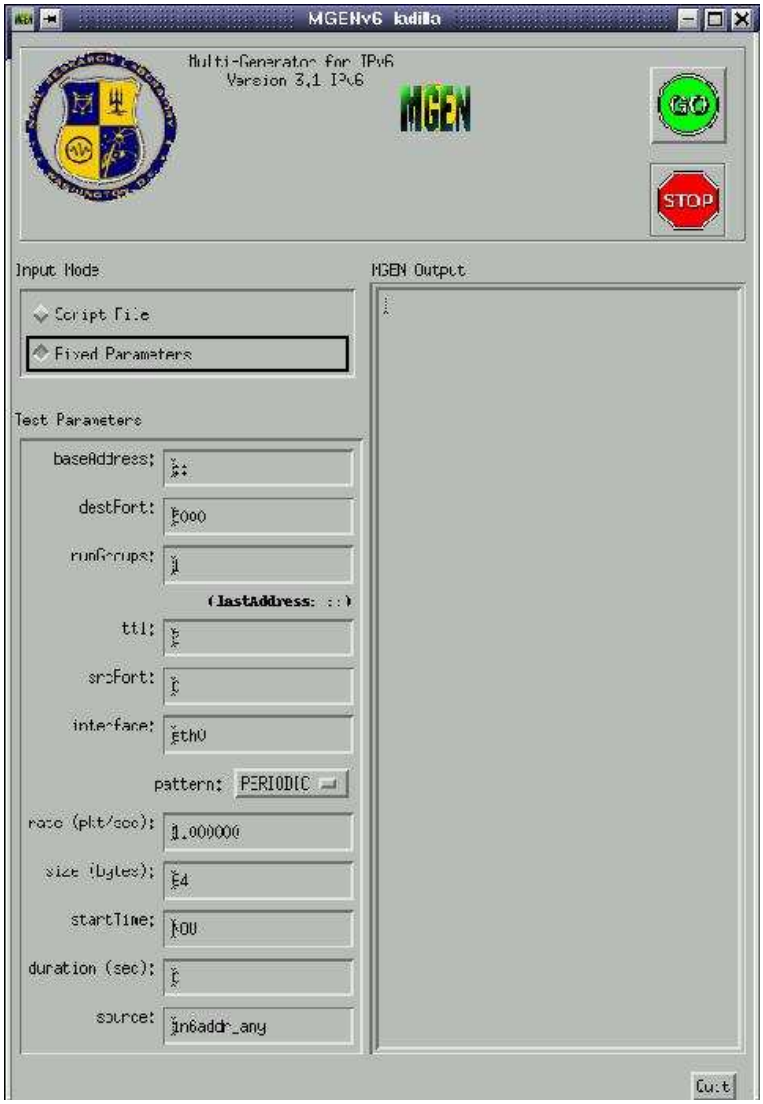

Figura 6.2: Interfaz gráfica de MGENv6.

En ambas ventanas, en la parte derecha se muestra la salida que produce la ejecución del programa. Al terminar la ejecución o al ser interrumpida pulsando CTRL+C, se muestra la siguiente información (tanto en la versión gráfica como en la versión línea de comandos):

1. Paquetes transmitidos (*"packets Tx'd"*): número de envíos a través del socket. Mgen utiliza un contador interno que actualiza cada vez que se manda un paquete.

2. Tramas transmitidas (*"frames Tx'd"*): número de paquetes que realmente salen por la interfaz. Este cálculo se efectúa capturando la salida de tcpdump/netstat al comienzo y al final de la ejecución, y calculando la diferencia. Normalmente será un número igual o mayor que el de paquetes transmitidos, salvo que la tarjeta no sea capaz de procesar los paquetes que tiene en la cola.

3. Periodo de transmisión (*"transmission period"*): segundos que ha estado Mgen en ejecución enviando paquetes.

4. Promedio de velocidad de transmisión de paquetes: velocidad media de todos los paquetes enviados, en paquetes por segundo.

5. Transmisiones erróneas y colisiones (*"tx errors, collisions"*): obtenidas a partir de tcpdump/netstat, muestra el número de *"frames"* erróneos y el número de colisiones que se han producido en la interfaz de salida.

A continuación y a modo de resumen se presenta un ejemplo de la información que muestra como salida la herramienta Mgen:

```
MGEN: Version 3.1 IPv6
MGEN: Loading event queue ...
MGEN: Seeding random number generator ...
MGEN: Beginning packet generation ...
      (Hit <CTRL-C> to stop)^C

MGEN: Packets Tx'd       :        2
MGEN: Transmission period:     1.754 seconds.
MGEN: Ave Tx pkt rate    :     1.141 pps.
MGEN: Interface Stats    :      rl0
             Frames Tx'd :        2
             Tx Errors :         0
             Collisions :        0
```

Por otro lado, la interfaz gráfica de Drec presenta la misma funcionalidad que puede obtenerse mediante la versión no gráfica, pudiendo utilizarse tanto el estilo de parámetros prefijados a través de la línea de comandos, como utilizando un *"script"* de configuración para escuchar dinámicamente diferentes grupos multicast en diferentes instantes de tiempo. A continuación, en la figura 6.3 y en la figura 6.4 de la página 161 se presentan las dos ventanas de Drec:

Figura 6.3: Interfaz gráfica de DRECv6.

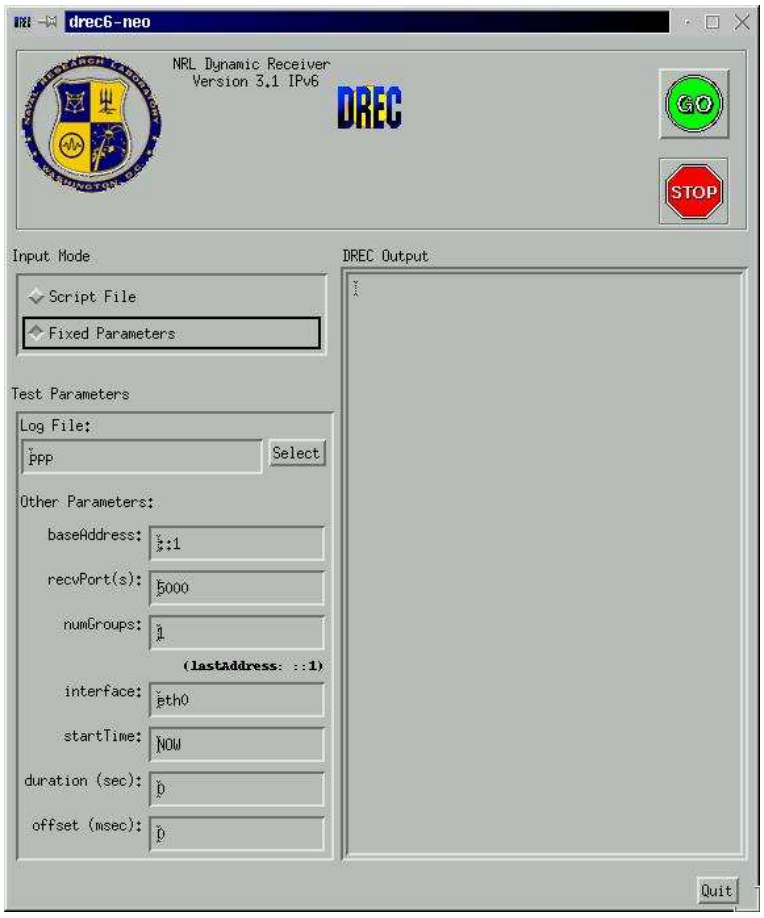

Figura 6.4: Interfaz gráfica de DRECv6.

Drec graba los paquetes recibidos en un archivo. Dicho archivo puede utilizarse para obtener distintas estadísticas, mediante un procesamiento posterior.

6.1.3 Cabeceras de extensión para IPv6.

Mgen soporta las siguientes cabeceras de extensión:

1. *"Routing header (Type 0)"*: especificada en [**?**], es el equivalente al encaminamiento fuente para IPv6.

2. *"Router alert hop-by-hop option"*: cabecera que es procesada por los routers y que puede llevar información sobre mensajes *"multicast listener discovery"*, mensajes de RSVP o mensajes de redes activas, quedando una serie de bits reservados para futuros usos. Su definición se encuentra en [**?**].

3. *"Jumbo payload hop-by-hop option"*: para enviar paquetes con un tamaño mayor que el soportado por el campo longitud de la cabecera IPv6, se encuentra definida en [**?**].

4. Cabeceras de movilidad: se encuentran definidas en [**?**]:

   - *"Home Address destination option"*.

- *"Binding Update destination option".*
- *"Binding Acknowledge destination option".*
- *"Binding Request destination option".*

Además, se puede seleccionar la dirección origen de los paquetes de cada flujo, de entre todas las direcciones disponibles para la máquina.

Estas cabeceras se pueden insertar en cualquier flujo y activar y desactivar en cualquier instante de tiempo. Pueden coexistir varias cabeceras en el mismo flujo, y por supuesto pueden enviarse varios flujos con distintas cabeceras.

Actualmente la identificación de los flujos que genera Mgen se transporta a nivel UDP, pero se podría utilizar el campo *"flowlabel"* de la cabecera IPv6 para transportar dicha información, en caso de que dicho valor pueda seleccionarse a nivel de aplicación, lo cual en el momento de escribir este proyecto fin de carrera no está decidido.

6.1.4 Cálculo de estadísticas.

En este apartado se presenta la herramienta encargada de generar estadísticas denominada Mcalc. Mcalc es el programa encargado de examinar el archivo de *log* generado por Drec, y mostrar estadísticas generales y agrupadas por flujos. Los resultados son aproximados, ya que por ejemplo si se envía un único flujo de 2000 paquetes y los últimos 500 no llegan al destino, Mcalc no tiene forma de saber examinando el archivo generado por Drec, si el flujo se componía de 1500 paquetes o si se componía de más y en consecuencia mostrará que la tasa de pérdidas ha sido cero.

El funcionamiento de Mcalc es bastante simple, va leyendo las líneas almacenadas en el fichero de *log*, ignorando las entradas de "JOIN" y "LEAVE" y las entradas donde se muestran detalles de las cabeceras de extensión recibidas, y utilizando variables auxiliares va generando estadísticas sobre los flujos que vuelca a otro fichero. Las entradas "JOIN" y "LEAVE" se utilizan únicamente para obtener el tiempo que transcurre hasta que comienzas/dejas de recibir paquetes de un determinado flujo multicast.

6.1.5 Mejoras realizadas a la herramienta.

A lo largo del proyecto europeo LONG [**?**] se ha modificado la herramienta y solucionado *bugs* que se encontraban tanto en la versión IPv4 como *bugs* de la nueva versión para IPv6. Las modificaciones más importantes realizadas en las sucesivas versiones de MGEN se pueden resumir en los siguientes aspectos destacables:

- La interfaz gráfica no funcionaba correctamente en IPv4. Además de solucionar los *bugs* que presentaba, se ha modificado para poder funcionar con IPv6.
- La llamada a "ifconfig" que se utiliza para mostrar el número de *"frames"* enviados/recibidos por la tarjeta ha sido modificada puesto que la version IPv4 no reconoce las entradas "inet6" de la salida de "ifconfig".

- La escritura de Drec en disco resulta muy ineficiente, por lo que en las últimas versiones de MGEN se ha programado una optimización que consiste en escribir la información recibida en formato binario, utilizando las mínimas llamadas posibles a la llamada al sistema "write". A continuación, es necesario ejecutar un nuevo programa llamado "decode" que se encarga de traducir el fichero binario en un fichero de texto con exactamente el mismo formato que el Drec original. Sobre este último fichero, se puede trabajar igual que antes y aplicar Mcalc. Para automatizar el proceso, es posible unir mediante "cauces" la ejecución de "decode" y "mcalc" reduciendo a uno el numero de pasos necesarios para procesar la salida de Drec y mejorando el rendimiento de Drec bajo condiciones de carga altas en un 15-20%.

Respecto al último punto hay que destacar que aunque se ha mejorado la escritura del programa Drec, ésta sigue siendo el principal cuello de botella de la aplicación, sobre todo cuando se generan paquetes a altas velocidades y con tamaños pequeños (64 bytes). Por esto, se recomienda escribir en un sistema de ficheros virtual en memoria RAM, que está soportado tanto por *FreeBSD* como por *Linux*:

- En linux: el módulo que permite montar un sistema de ficheros virtual en memoria RAM se encuentra en *"FileSystems"* → *"Simple RAM-based FileSystem Support"*, se selecciona como módulo y se recompilan con "make modules" y "modules_install". Una vez hecho esto, el sistema de ficheros se monta ejecutando:

  ```
  mount -t ramfs ramfs /tmp
  ```

  A diferencia de otros sistemas de ficheros en RAM, este sistema crece dinámicamente para dar cabida a todos los ficheros en él almacenados.

- En *FreeBSD*: es necesario recompilar el núcleo con la siguiente opción: *"options MFS #Memory File System"*, luego basta ejecutar lo siguiente:

  ```
  mount_mfs <particion_de_swap> /tmp
  ```

Otro punto a tener en cuenta es la imposibilidad de manejar el campo *"flowlabel"* de IPv6 ya que las implementaciones de IPv6 existentes no permiten modificar ni leer este campo, debido a que no existe una API bien definida para manejar esta información y a que no está claro si dicha información debe ser visible/modificable por las máquinas finales. El campo *"flowlabel"* está formado por 20 bits. En [**?**] se describe un API para manejar la información de flujo. En concreto, esta API permitiría especificar y leer el valor de la etiqueta de flujo por cada paquete.

6.2 Herramienta ping6to4.

Esta herramienta tiene como objetivo detectar si el extremo remoto de un túnel 6to4 [**?**] se encuentra funcionando correctamente, y además presenta la característica de que se ha desarrollado sin hacer uso de la nueva API para IPv6, por lo que puede ser utilizada en máquinas IPv4. Por supuesto, existen otros mecanismos para averiguar si una máquina remota responde a una determinada solicitud, fundamentalmente haciendo uso de herramientas del

estilo del potente "tcpdump". Pero en este caso, se trata de desarrollar una herramienta específica para simular un túnel 6to4 en el extremo emisor y ser capaces de procesar la respuesta del extremo remoto, actuando como un ping tradicional.

El código está parcialmente basado en la implementación del "ping" para IPv4 realizada por *"Mike Muuss"* del *"U.S Army Ballistic Research Laboratory"* en 1983, posteriormente modificada por la universidad de *Berkeley*[3]. Este código es de dominio público y puede ser distribuido sin ninguna limitación.

La compilación resulta muy sencilla y basta con hacer "make". En caso de tener problemas con la compilación de la herramienta, puede ser necesario modificar el archivo "Makefile".

6.2.1 Funcionamiento interno.

Dado que el código debe funcionar en máquinas IPv4, se deben definir todas las cabeceras y tipos de datos relacionados con IPv6, como por ejemplo "in6_addr", "sockaddr_in6", "ip6_hdr", "icmp6_rr", etc. Además se definen las siguientes constantes relacionadas con IPv6:

```
#define IPPROTO_IPV6 41 /* protocol type for encapsulation of IPv6 inside of
                           IPv4 packets */
#define INET6_ADDRSTRLEN 46
#define IPPROTO_ICMPV6 58
#define ICMP6_ECHO_REQUEST  128
#define ICMP6_ECHO_REPLY    129
```

Antes de entrar en el bucle principal, se planifica una interrupción cada segundo con "alarm(1)". Dicha interrupción es la encargada de enviar un paquete IPv6 encapsulado en IPv4. También se habilita la interrupción "CTRL+C" para finalizar el programa y mostrar las estadísticas obtenidas, de la siguiete forma:

```
signal(SIGALRM, sig_alrm);
signal( SIGINT, finish );
```

El bucle principal del programa es un bucle infinito encargado de recibir los paquetes de respuesta, procesarlos y almacenar las estadísticas. Justo antes de entrar al bucle se envía el primer paquete llamando directamente a la función "sig_alrm()". El bucle infinito se encuentra en la función "readloop()" y presenta la siguiente estructura:

```
for( ; ; )
{
  n = recvfrom( sockfd, recvbuf, sizeof(recvbuf), 0,
                (struct sockaddr *)&origen_ip, &len_origen_ip );
  if( n < 0 )
  {
    if( errno == EINTR)
      continue;
```

[3]También me he basado en la implementacion de ping utilizada en "UNIX Network Progamming - Networking APIs: Sockets and XTI", de R. Stevens.

```
      else
      {
        printf("recvfrom error\n");
        exit(-1);
      }
    }

    gettimeofday( &tval, NULL );
    process_packet( recvbuf, n, &tval ); // increments nreceived if all is ok.
  }
```

A continuación se describen las funciones encargadas de recibir y de enviar los paquetes.

Envio de paquetes.

La función encargada de enviar paquetes "send_packet()" construye paso a paso el formato de un paquete ICMPv6 encapsulado en IPv4 de la siguiente forma:

```
  ip6= (struct ip6_hdr *)sendbuf;
  ip6->ip6_ctl.ip6_hdrctl.ip6_flow = 0;
  ip6->ip6_ctl.ip6_hdrctl.ip6_plen = htons( sizeof(struct icmp6_rr) + datalen );
  ip6->ip6_ctl.ip6_hdrctl.ip6_nxt = IPPROTO_ICMPV6;
  ip6->ip6_ctl.ip6_hdrctl.ip6_hlim = 255;
  ip6->ip6_ctl.ip6_vtc = 6 << 4;
  memcpy( &ip6->ip6_src, &origen_ip6.sin6_addr, 16 );
  memcpy( &ip6->ip6_dst, &destino_ip6.sin6_addr, 16 );

  icmp6= (struct icmp6_rr *)(sendbuf + sizeof(struct ip6_hdr));
  icmp6->icmp6_type = ICMP6_ECHO_REQUEST;
  icmp6->icmp6_code = 0;
  icmp6->icmp6_id = pid;
  icmp6->icmp6_seq = nsent++;
```

El tamaño del paquete ICMPv6 es de 56 bytes (*"datalen"*) por defecto, y en este espacio se graba el instante de tiempo en el que se envía el paquete. También es importante notar que el identificador del paquete ICMPv6 se construye utilizando el "pid" del proceso, para poder diferenciar los paquetes en caso de más de una ejecución simultánea del programa.

Para calcular el checksum de ICMPv6, es necesario anteponer al paquete una pseudo-cabecera con el siguiente formato:

```
struct ip6_pseudo_hdr
{
  struct in6_addr ip6_src;
  struct in6_addr ip6_dst;
  u_int32_t       ip6_plen;
  u_int32_t       ip6_nxt;
};
```

El campo "icmp6_cksum" se inicializa a cero y después se rellena la pseudo-cabecera y se llama a la función que calcula el checksum, como se observa en el siguiente trozo de código:

```
icmp6->icmp6_cksum = 0;

memcpy( auxbuf + sizeof(struct ip6_pseudo_hdr), icmp6,
                              sizeof(struct icmp6_rr) + datalen );
pseudo_ip6= (struct ip6_pseudo_hdr *)auxbuf;

pseudo_ip6->ip6_src= ip6->ip6_src;
pseudo_ip6->ip6_dst= ip6->ip6_dst;
pseudo_ip6->ip6_plen= htonl( sizeof(struct icmp6_rr) + datalen );
pseudo_ip6->ip6_nxt = htonl( IPPROTO_ICMPV6 );

icmp6->icmp6_cksum = in_cksum( (u_short *)pseudo_ip6,
                                  sizeof(struct ip6_pseudo_hdr) +
                                  sizeof(struct icmp6_rr) +
                                  datalen );
```

El algoritmo utilizado en la función "in_cksum()" es un algoritmo típico que se utiliza por su simplicidad, aunque no resulta muy eficiente. Utilizando un acumulador de 32 bits, se van añadiendo palabras de 16 bits sobre él, y al final se desplazan los 16 bits más altos (todos los bits de acarreo) hacia los 16 bits más bajos, devolviendo el resultado.

Por último se llama a la función "sendto()" y se comprueban los posibles errores en el envío, aumentándose el contador de paquetes enviados.

Recepción y procesamiento de paquetes.

Para recibir los paquetes encapsulados se tiene que utilizar un socket "crudo" (*"raw socket"*) especificando el tipo de protocolo utilizado para encapsular IPv6 dentro de IPv4, que es el número 41 y se encuentra definido en la constante "IPPROTO_IPV6", además los sockets crudos sólo pueden ser abiertos por el super-usuario, por lo que el programa necesita permisos de *"root"* para ser ejecutado. Después de especificar un tamaño para el buffer de recepción de paquetes y de asignar una dirección IPv4 origen al paquete mediante la llamada "bind()", se entra en el bucle infinito encargado de la recepción de paquetes mediante la llamada "recvfrom()". Todo esto se puede observar en el siguiente trozo de código:

```
if( (sockfd= socket(PF_INET, SOCK_RAW, IPPROTO_IPV6 )) < 0 )
{
  perror("error in socket():");
  exit(-1);
}
setuid(getuid()); /* don't need special permissions any more */

size= 60*1024;
if( setsockopt( sockfd, SOL_SOCKET, SO_RCVBUF, &size, sizeof(size) ) < 0 )
{
```

```
    perror("error in setsockopt():");
    exit(-1);
  }
```

El procesamiento de los paquetes recibidos consiste en ir examinando las distintas cabeceras, comprobar que el paquete encapsulado es un paquete ICMPv6, comprobar que el tipo de paquete ICMPv6 es un paquete de respuesta ("ICMP6_ECHO_REPLY") y que es una respuesta a un paquete enviado por nuestro programa. Después de esto, se coge el instante de tiempo en el que se ha recibido el paquete, se muestra la información del "ping" por pantalla y se aumenta el contador de paquete recibidos.

En el código anterior se observa que se incrementa el tamaño del buffer de recepción del socket hata 60*1024 para el caso en que se realicen "pings" a direcciones multicast que pueden generar gran cantidad de réplicas.

Cuando el programa finaliza (pulsando CONTROL+C), se muestra el siguiente resumen de información:

```
./ping6to4 -s 163.117.139.44 2002:a375:8ba6::1
PING6-4 2002:a375:8ba6::1 : 56 data bytes
64 bytes from 2002:a375:8ba6::1: seq=0, hlim=64, rtt=0.350 ms
64 bytes from 2002:a375:8ba6::1: seq=1, hlim=64, rtt=0.385 ms
^C
----2002:a375:8ba6::1 PING6-4 Statistics----
2 packets transmitted, 2 packets received, 0% packet loss
round-trip (ms)  min/avg/max = 0.350/0.368/0.385
```

El RTT se calcula restando el tiempo en el que fue enviado el mensaje del tiempo en el que fue recibido. Se observa que es necesario especificar por línea de comandos la dirección IPv4 origen [4] , pues dicha dirección se utiliza para asignar el origen de los paquetes en la llamada "bind" del socket y además se utiliza para construir la dirección 6to4 origen del paquete IPv6 encapsulado.

6.2.2 Consideraciones finales sobre la herramienta.

En el archivo "utils.c" se encuentran una serie de funciones utilizadas para manejar direcciones IPv6, en concreto para construir la dirección 6to4 origen a partir de la dirección IPv4 y para extraer la dirección IPv4 de la dirección 6to4 destino. Dichas funciones se basan en las implementaciones de la librería "libinet6.a" de KAME.

La implementación de esta herramienta ha sido bastante sencilla, partiendo del código para IPv4, la estructura general de la aplicación ya se encontraba definida, y el desarrollo básicamente ha consistido en implementar el encapsulamiento, cambiar el tipo de socket a utilizar, calcular distintos valores para rellenar las nuevas cabeceras e implementar funciones auxiliares para manejar el tratamiento de direcciones IPv6. Las estructuras definidas para las

[4] Aunque se podría hacer de forma automática, en el momento de escribir este trabajo, se debe especificar la dirección IPv4. De esta forma, resulta más sencillo de implementar, sobre todo cuando se poseen varias interfaces de red.

cabeceras de IPv6 se han copiado de los archivos de cabecera del núcleo que KAME proporciona, necesitándose unos mínimos retoques para su compilación dentro de esta herramienta, sobre todo en lo concerniente a constantes definidas en otros *"include"s*.

Esta herramienta puede mostrar su utilidad en los años venideros, donde se espera una coexistencia de ambos protocolos (IPv4 e IPv6) y la paulatina migración hacia un entorno IPv6 nativo, transición que se prevé bastante larga en el momento de escribir este proyecto fin de carrera.

6.3 Servidor de ajedrez FICS.

A continuación se comenta la migración de la versión del servidor de ajedrez más utilizado en la actualidad, denominado *"Free International Chess Server"* o simplemente "FICS". El autor de este *software* ha cambiado la licencia y desde hace unos años no es de código libre, así que se ha tenido que migrar una versión más antigua que la actual.

En los siguientes apartados se comentan por separado la migración del programa servidor y de la interfaz gráfica más utilizada.

6.3.1 Migración de FICS

La última versión del servidor de ajedrez FICS, desarrollado principalmente por *"J. Nash"* , es la "1.6.2". Esta versión posee algunos *bugs*, por ejemplo la continuación de una partida aplazada entre dos jugadores registrados causaba la finalización abrupta del servidor debido a que la información del tablero se guardaba incorrectamente. Este y otros *bugs* se han resuelto aprovechando el esfuerzo realizado en la migración del servidor.

La migración se ha realizado modificando los siguientes archivos:

- "network.c/h": las estructuras se han modificado para soportar sockets IPv6.
- "playerdb.c/h": cuando un jugador registrado se desconecta, se guarda siempre determinados datos sobre dicho jugador. Esta información incluye la dirección IP del jugador. De esta forma, se ha tenido que cambiar el código para poder salvar/recuperar la dirección IPv6 del jugador.
- "utils.c/h": las funciones que tratan con conversiones entre direcciones en formato de red y direcciones en formato representable gráficamente se han adaptado para soportar IPv6 utilizando las funciones "inet_ntop()" e "inet_pton()".

6.3.2 Migración de Xboard

La interfaz gráfica más utilizada para conectarse a los distintos servidores de ajedrez públicamente disponibles en internet es el programa "Xboard". Concretamente la versión migrada ha sido la "4.2.6" que no soporta conexiones IPv6. La migración se ha realizado de una forma bastante sencilla, y dado que no existe ningún servidor de ajedrez con soporte para conexiones IPv6, en este caso podemos decir que la falta de soporte por parte de "Xboard" está justificada. Pero dado que se ha migrado el servidor, resulta lógico migrar una interfaz gráfica para poder

tener un conjunto completo de herramientas IPv6 para jugar al ajedrez. Aunque no se ha comentado, se puede jugar en el servidor de ajedrez utilizando un simple "telnet" contra el puerto 5000, pero una interfaz gráfica simplifica muchísimo la forma de interactuar con el servidor.

Concretamente la migración se ha centrado en el cambio de una única función, que es la encarga de realizar la conexión TCP con el servidor. Se ha hecho uso de la función "getaddrinfo()", lo que permite utilizar IPv6 o IPv4 de una forma transparente interaccionando con el DNS.

6.4 Conclusiones.

En este apartado se van a dar unas recomendaciones generales para la migración de aplicaciones al nuevo protocolo IPv6. Por otro lado, en el capítulo dedicado a conclusiones y trabajos futuros se describe la experiencia del autor en la migración concreta de la herramienta MGEN.

Antes de describir las nuevas funciones y los requisitos necesarios para poder migrar una aplicación a IPv6 conviene comentar brevemente el código involucrado en cualquier aplicación cliente/servidor, esté funcionando sobre IPv4 o sobre IPv6. La secuencia de instrucciones típica para una aplicación servidora es la siguiente:

1. *socket()*
2. *bind()*
3. *listen()*
4. *accept()*

mientras que desde el punto de vista de una aplicación cliente, la secuencia de código es la siguiente:

1. *socket()*
2. *connect()*
3. *read()/write()/recvfrom()/sendto()*

Estas instrucciones son independientes del protocolo de red que se utilice para desarrollar la aplicación. El problema surge debido a que el tamaño de la dirección IP resulta visible para las aplicaciones a través de la interfaz de sockets, por lo que se ha necesitado realizar cambios en partes del API que exponen el tamaño de la dirección IP (nuevas estructuras de datos para IPv6), ademas de los cambios necesarios en aquellas aplicaciones que manipulen directamente direcciones IP.

Los cambios realizados en la interfaz de sockets se resumen brevemente en la tabla 6.5 de la página 170:

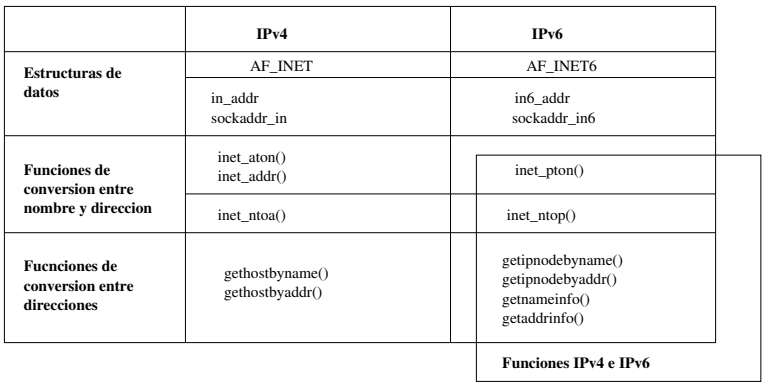

Figura 6.5: Correspondencia entre el API de IPv4 y el API de IPv6.

El comportamiento de los *socket raw* en IPv6 es ligeramente distinto al existente para el protocolo IPv4. Con un *socket raw* IPv4 se pueden recibir o enviar paquetes IPv4 completos, mientras que un *socket raw* IPv6 no permite enviar ni recibir cabeceras de extensión. Para ello se tienen que utilizar opciones de socket específicas u otras técnicas a nivel de enlace.

A continuación se da una breve guia sobre las partes de una aplicación que deben localizarse y cambiarse para conseguir que dicha aplicación funcione correctamente sobre el nuevo protocolo de red IPv6:

- Para una aplicación servidor:
  - Cambiar las funciones del API de sockets.
  - Ajustar las funciones de *"logging"* de forma que puedan manejar direcciones IP más grandes.
  - Aumentar todas aquellas estructuras (y cualquier aplicación relacionada con el programa, por ejemplo, bases de datos) que almacenen direcciones IP.
- Para una aplicación cliente:
  - Los mismos puntos que para la parte de migración del servidor.
  - Además, ajustar la interfaz con el usuario de tal forma que pueda manejar direcciones IPv6.

También se debe tener en cuenta que algunas aplicaciones utilizan ":" para separar la dirección IP del puerto. Como IPv6 utiliza ":" dentro de las direcciones, se debe cambiar el separador. Se recomienda utilizar el formato definido para las *"urls"* en [**?**], que consiste en agrupar la dirección IPv6 dentro de corchetes y separar el puerto mediante ":".

Se recomienda a los implementadores usar la función *"getaddrinfo()"* para interactuar con el servidor de nombres, ya que se trata de una función independiente del protocolo y de esta forma se maximiza la portabilidad de las aplicaciones. De forma semejante, existen las nuevas funciones *"inet_ntop()"* e *"inet_pton()"*. La primera de ellas convierte una dirección expresada en *"network byte order"* a un formato de representación externo (una cadena de caracteres), mientras que la segunda hace justamente lo contrario. Las iniciales *"ntop"* se corresponden con la descripción en inglés de su funcionamiento, *"network identifier to presentation format"*.

De forma equivalente a *"getaddrinfo()"*, también existe la función independiente del protocolo denominada *"getnameinfo()"*. Esta función realiza la acción contraria a *"getaddrinfo()"*, es decir, obtiene el nombre a partir de una dirección IP, sea esta IPv6 o IPv4. Con este grupo de funciones, es posible programar aplicaciones de una forma independiente al protocolo de red subyacente.

Migrar una aplicación a iPv6 no debe resultar excesivamente costoso, pero en general dependerá del tipo de aplicación y de cómo esté programada. En [**?**] se comenta la migración del juego *"Quake2"*, donde se tuvo que cambiar ligeramente la interfaz de consola y las comunicaciones de red que se encontraban agrupadas en un único fichero. Según [**?**], se tardó un total de treinta y dos horas en recuperar el software, leer el código, encontrar dónde se debían realizar los cambios, implementar los cambios, levantar el servidor y jugar por primera vez al *"Quake2"* sobre IPv6, todo ello utilizando a dos programadores, de los cuales uno no sabía mucho sobre el nuevo protocolo de red. Sólo se necesitaron cambiar doscientas líneas de unas ciento cincuenta mil, el mismo número de lineas que se codificaron durante la migración a IPv6 de la herramienta de audioconferencia *"rat"*.

Para concluir este capítulo, a continuación se realizan una serie de comentarios generales y recomendaciones para ayudar al implementador a conseguir una migración sencilla y suave hacia el nuevo protocolo de red. Por supuesto, cada aplicación es única en su forma de implementación, y lo que resulta sencillo para algunas, puede resultar extremadamente complicado para otras. En general, puede afirmarse lo siguiente:

- Se debe programar de una forma independiente del protocolo de red subyacente: el código no tiene que preocuparse de si los clientes son IPv4 o IPv6, y a nivel de aplicación no se deben manejar direcciones de máquinas, sino sólo nombres. La gestión de las comunicaciones debe aislarse del resto de los módulos funcionales de la aplicación y deben eliminarse opciones de compilación relacionadas con la capa de red, especialmente cuando se encuentran dispersas por toda la aplicación. Siguiendo estas directrices la adición de caracterísiticas avanzadas del protocolo de red y la futura actualización a nuevos protocolos resultará mucho más sencilla.

- Es conveniente migrar la aplicación: aunque se pueden utilizar mecanismos de transición para comunicar aplicaciones IPv4 con aplicaciones IPv6 y viceversa, la migración de una aplicación a IPv6 permite utilizar las caracterísiticas añadidas del nuevo protocolo, haciendo a la aplicación "más extensible" y ofreciendo un abanico de nuevas posibilidades inexistentes en IPv4 (verdadero *"peer to peer"* sin traducciones intermedias, por poner un ejemplo). No obstante, durante el período de transición a IPv6, para dar servicio a redes IPv4, la aplicación debe diseñarse para funcionar en cualquier tipo de entorno (IPv4 o IPv6), para lo cual es un requisito indispensable la existencia de máquinas *"dual-stack"*. Resumiendo, al migrar una aplicación a IPv6 se convierte en una aplicación lista para ser utilizada en la actual internet de nueva generación, abriéndose nuevos nichos de mercado.

- La migración suele resultar sencilla para un programador: las mismas llamadas a sockets y unas cuantas funciones nuevas. Los cambios se encuentran localizados y no es necesario cambiar mucho código. Existen herramientas, por ejemplo, que examinan el código fuente e informan sobre aquellas partes del código que deberían ser modificadas, baste citar el *"IPv6 Socket Scrubber"*, que puede obtenerse de forma gratuita de "http://www.sun.com/solaris/ipv6".

Resumiendo, todos estos requisitos, y más en general la funcionalidad de red de cualquier aplicación de tamaño mediano o grande, se recomienda que sean implementados como una librería dentro de la aplicación o sistema que se está desarrollando. Se trata de realizar una interfaz *"middleware"* con la capa de red para aislar las comunicaciones del resto del programa. De esta forma, se simplifican las operaciones de mantenimiento futuras y se puede reutilizar como una interfaz de red general para aplicaciones similares. Si la aplicación está bien diseñada, la migración a IPv6 no sólo no resulta costosa, sino que aporta beneficios.

Capítulo 7

Conclusiones y trabajos futuros.

En este capítulo se comenta de una forma resumida el trabajo que se ha relizado durante el presente proyecto fin de carrera. También se muestran las posibles líneas continuadoras y trabajos futuros que guardan relación con el trabajo realizado.

7.1 Conclusiones.

El protocolo de red IPv6 se encuentra en fase de experimentación pre-comercial, y aunque su introducción en Internet se va a producir de una forma paulatina y lenta, el hecho es que ya se está utilizando en producción en algunas partes del mundo, como por ejemplo Japón, donde existen varios proveedores de internet comerciales que ofrecen conexión IPv6. Con la realización de este proyecto fin de carrera, enmarcado dentro del proyecto europeo LONG[**?**], se ha conseguido acumular una gran cantidad de experiencia en la gestión y configuración de redes IPv6 y también en la migración de aplicaciones, experiencia que resultará de gran valor para poder afrontar los nuevos retos de la futura Internet con garantías de éxito.

Por consiguiente, en los subsiguientes apartados se resume el trabajo realizado durante el presente proyecto fin de carrera, agrupando los temas que se han tratado en tres grandes bloques relacionados entre sí:

- Gestión de red.
- Mecanismos de transición.
- Migración de aplicaciones.

7.1.1 Gestión de red.

El presente trabajo, realizado como proyecto fin de carrera, ha servido para gestionar el establecimiento y configuración de una red IPv6 de tamaño medio dentro de los laboratorios proporcionados para tal fin por el Departamento de Ingeniería Telemática. Esta red, que incluye varias máquinas pertenecientes a otros proyectos, se puede observar en la figura 7.1 de la página 174, que se muestra a continuación:

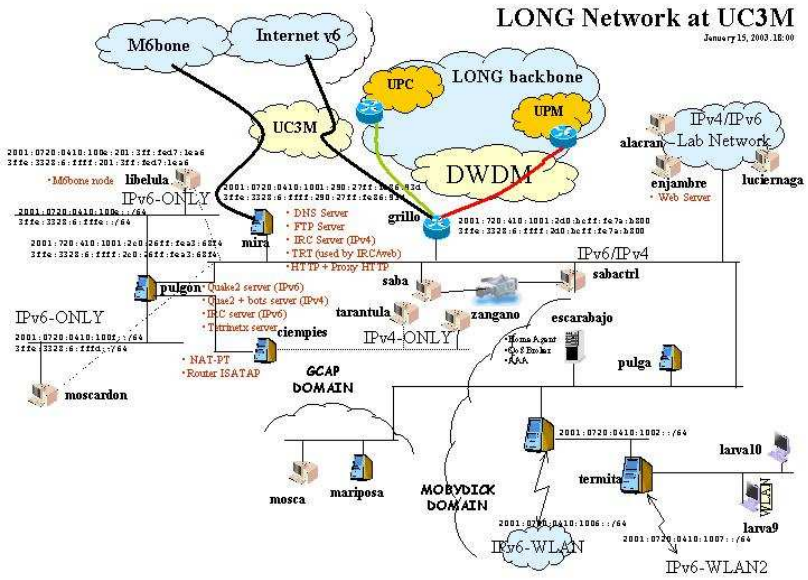

Figura 7.1: Red IPv6 del laboratorio de Ingeniería Telemática.

La gestión de la red no se diferencia mucho respecto a la gestión de una red IPv4 tradicional. A continuación se van a comentar brevemente los pasos necesarios para construir una red IPv6 a partir de la infraestructura que IPv4 pone a nuestra disposición, de una forma genérica, pero que puede servir como ejemplo para la migración de nuevas redes haciendo uso de la experiencia adquirida en la gestión de la infraestructura descrita en la figura anterior.

Básicamente se necesita comprar un router con soporte IPv6 o actualizar un router para que soporte el nuevo protocolo de red. Después, se debe decidir qué parte de nuestra red necesita soportar IPv6. La infraestructura IPv4 no tiene que cambiarse, y puede resultar conveniente ofrecer IPv6 a un determinado grupo de máquinas, o a toda nuestra red. En principio se podrían actualizar todos los dispositivos de red para que comprendieran el nuevo protocolo IPv6, pero es una operación costosa de difícil justificación para una compañía de tamaño mediano, por lo que resulta más conveniente ir añadiendo nuevos segmentos con IPv6, según se vayan necesitando. Se pueden añadir segmentos que sean sólo IPv6. Esto implica hacer uso de algún mecanismo de transición para posibilitar la comunicación entre el mundo IPv6 y el mundo IPv4, donde la mayoría de las redes y servicios residen en la actualidad. Existen multitud de mecanismos de transición a nuestra disposición para poder ofrecer dicha comunicación, y más adelante se comentarán de forma resumida las ventajas e incovenientes de cada uno de ellos. Si sólamente se quiere dar servicio IPv6 a un segmento de red, el router de acceso con soporte IPv6 deberá proporcionar acceso o establecerá comunicación con la red IPv6 externa a través de mecanismos de encapsulación de IPv6 en IPv4, al menos de forma transitoria hasta que se pueda establecer un servicio IPv6 nativo con el exterior. Como se ha comentado anteriormente, se hace uso de la infraestructura de encaminamiento de IPv4 para dar servicio IPv6 durante esta primera fase de introducción del nuevo protocolo.

Respecto al direccionamiento, se debe tener en cuenta los distintos tipos de direcciones que el nuevo protocolo IPv6 pone a nuestra disposición: direcciones *"link-local"*, direcciones *"site-local"* y direcciones globales.

Las direcciones *"link-local"* sólo pueden utilizarse para comunicaciones dentro de un enlace de red. Todos los interfaces de red IPv6 se autoconfiguran con una dirección *"link-local"* al

arrancar. Este tipo de direcciones se utiliza principalmente para funciones de control, y casi nunca se usan para intercambio real de datos entre aplicaciones. Por supuesto, paquetes con direcciones *"link-local"* no pueden atravesar los routers. Las direcciones *"site-local"* son efectivas y únicas dentro de un determinado dominio administrativo, especificado por el diseñador de la red. Por otro lado, el tercer tipo de direcciones, las direcciones globales, son únicas a nivel global. Si la conexión al mundo IPv6 se proporciona a través de algún proveedor de servicios, dicho proveedor delegará un rango de direcciones IPv6 globales. Si por otro lado se quiere disponer de una red IPv6 a nivel de *intranet*, sin salida IPv6 al exterior, hay que tener en cuenta el direccionamiento que se puede utilizar. Básicamente existen tres posibilidades para asignar direcciones en el caso de una *intranet* corporativa sin acceso IPv6 al exterior:

- Uilizar un rango de direcciones cualquiera: opción totalmente desaconsejada porque eventualmente la red IPv6 se conectará con el exterior y será necesario renumerar dicha red.
- Utilizar el rango de direcciones definido explícitamente para pruebas en [**?**]: el problema de esta opción es el mismo que el del punto anterior, pues no se garantiza de una forma explícita la reserva de este rango de direccionamiento para siempre.
- Utilizar direcciones *"site-local"*: es la opción más consistente para el propósito de construir una red IPv6 interna, puesto que son direcciones que se encuentran definidas desde la especificación inicial de IPv6 y poseen un ámbito definido, por lo que no hay que preocuparse por su duplicidad a nivel global [1].

Respecto al protocolo de encaminamiento, tanto a nivel de intra-dominio como a nivel externo o entre dominio administrativos se dispone de las mismas herramientas y protocolos que existen para IPv4. En este proyecto fin de carrera se ha trabajado y descrito tanto *"RIPng"* como *"BGP4+"*.

También se ha experimentado como parte de este trabajo fin de carrrera con redes de pruebas, como la red multicast sobre IPv6 conocida como *"m6bone"*[**?**] y la propia red de pruebas de IPv6, construida sobre la actual infraestructura de IPv4, denominada *"6bone"*. Para más información sobre el *"6bone"*, se recomienda mirar "http://www.6bone.net".

Resumiendo, y para finalizar este apartado, se ha demostrado que la implantación de una red IPv6 puede suponer un coste añadido para la gestión de dicha red, pero con la ventaja de que se pueden utilizar los mismos mecanismos existentes en la actual IPv4. La necesidad de implantar y gestionar diversos mecanismos de transición viene impuesta por el largo periodo de coexistencia que se augura a ambos protocolos. La disponibilidad de herramientas de gestión con soporte para el nuevo protocolo capacita a cualquier administrador de red, con mínimos conocimientos de IPv6, para realizar las tareas de gestión del día a día. Evidentemente, IPv6 no es la panacea de todos los protocolos de red. Las direcciones son más largas y casi imposibles de recordar, existe el concepto de ámbito de direcciones y todavía quedan ciertos aspectos del protocolo que necesitan ser especificados con más detalle, como el papel que jugarán las direcciones *"site-local"* o la inexistencia a día de hoy de mecanismos de renumeración automática de redes IPv6, o mecanismos de *"multihoming"* adecuados al sistema de encaminamiento jerárquico que IPv6 introduce. Todos ellos son aspectos avanzados del

[1] Hay una gran discusión dentro del IETF sobre la utilidad de las direcciones *"site-local"*, es posible que desaparezcan de la especificación del protocolo IPv6. En el momento de escribir este trabajo, dichas direcciones permanecen válidas.

protocolo que quedarán resueltos en los próximos años. Se puede argumentar que IPv6 es más complicado, y posiblemente se esté en lo cierto, dependiendo de los servicios que queramos obtener del nuevo protocolo, pero sin lugar a dudas IPv6 es el único futuro posible para las comunidades *"peer-to-peer"* y para la nueva generación de dispositivos conectados a internet. El pequeño coste de gestión o de implantación asociado al nuevo protocolo se suple con creces gracias a la experiencia adquirida en la administración de redes IPv4, pues al fin y al cabo, las diferencias son mínimas, y cuando existen, son para mejor.

7.1.2 Mecanismos de transición.

Para poder realizar una migración lo menos traumática posible resulta necesario utilizar mecanismos de transición. La elección de un determinado mecanismo de transición depende de las necesidades y recursos de cada dominio. Por consiguiente, se deben examinar los mecanismos de transición disponibles y seleccionar áquel que mejor se adapte a nuestras necesidades. Puede ser necesario utilizar más de un mecanismo de transición. Cada mecanismo se encuentra definido para un determinado ámbito y posee consideraciones de aplicabilidad que deben tenerse en cuenta. Unos mecanismos són más fáciles de instalar, otros se encuentran implementados sólamente en determinadas platarformas y otros hacen uso de un mayor o menor número de direcciones IPv4. Todos estos aspectos deben tenerse en cuenta.

En general, podemos decir que la elección del mecanismo de transición adecuado es la tarea más dificil con la que deberá enfrentarse el administrador de sistemas. El IETF tiene previsto definir guías para la implantación de mecanismos de transición para determinados ámbitos o escenarios, desde grandes redes corporativas o ISPs hasta dispositivos aislados o usuarios domésticos.

Durante la realización del presente proyecto fin de carrera se han estudiado los mecanismos de transición más importantes y se han probado aquellos mecanismos que en el momento de escribir este trabajo presentaban alguna implementación en *Linux* o *FreeBSD*. También se han realizado algunas pruebas menores con la pila IPv6 de *Windows 2000*, en concreto con el mecanismo de transición 6to4. En la siguiente figura, figura número 7.2, se puede observar de una forma esquemática los mecanismos de transición que se han estudiado e implantado en la red de pruebas:

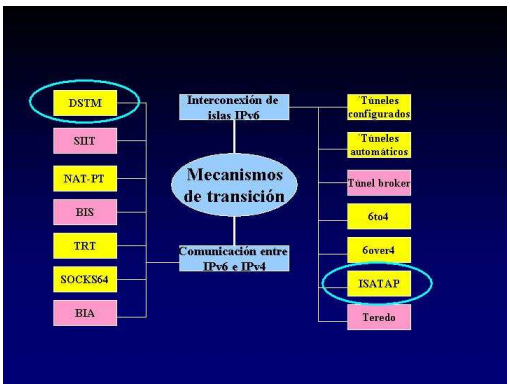

Figura 7.2: Mecanismos de transición estudiados.

7.1.3 Migración de aplicaciones.

En un entorno *"dual-stack"*, cada aplicación decide que protocolo de red utilizar. Si una aplicación soporta sólamente IPv4, dicha aplicación sólo podrá hacer uso de IPv4. No obstante y gracias al esfuerzo realizado en la migración de la mayoría de las principales aplicaciones a IPv6, existen una gran variedad de aplicaciones que actualmente soportan ambos protocolos de red. Estas aplicaciones están escritas de tal forma que primero intentan utilizar IPv6, y si no tienen éxito, tratan de establecer la comunicación utilizando IPv4. Esto es así porque un socket IPv6 puede enviar y recibir paquetes IPv4, pero no sucede lo mismo al revés.

Se recomienda programar las aplicaciones de forma independiente al protocolo de red subyacente, es decir, el código no necesita tener conciencia de si el nodo comunicante es IPv6 o si es IPv4. De esta forma, cualquier actualización posterior de la capa de red no tiene que impactar en el código de la aplicación. También se recomienda programar de una forma modular, encapsulando todas las funciones de red de tal forma que cualquier modificación se pueda realizar de una forma local sin que tenga repercusiones en el resto de los módulos de la aplicación.

Respecto al trabajo realizado en el ámbito de la migración de aplicaciones dentro del presente proyecto fin de carrera, se ha experimentado con una aplicación de red avanzada, MGEN, y con otros programas de menor importancia como el servidor de ajedrez y su correspondiente interfaz gráfico, o el pequeño programa que hace uso de *"raw sockets"* para IPv6 denominado "ping6to4".

El porcentaje de código añadido durante la migración de la herramienta MGEN ha sido un poco más alto de lo habitual, debido a que no sólo se ha migrado la herramienta si no que se ha añadido nueva funcionalidad que antes no existía. Concretamente MGENv4 tiene 12.640 líneas de código fuente[2], mientras que la versión para IPv6 posee 13.154, lo que supone un incremento de código de un 4%.

7.2 Trabajos futuros.

A continuación se presenta una lista de trabajos futuros que se prodrían realizar como continuación de este trabajo fin de carrera:

- Aumentar la funcionalidad de MGEN: se puede aumentar la funcionalidad permitiendo la utlización de cabeceras de extensión relacionadas con la seguridad.
- Añadir más servicios a la red IPv6: por ejemplo un servidor de correo IPv6, sistemas automáticos de renumeración de la red, soporte para DHCPv6 y calidad de servicio.
- Realizar pruebas de rendimiento sobre los distintos mecanismos de transición: este proyecto se ha limitado a la instalación y configuración de determinados mecanismos de transición. Los mecanismos de transición resuelven el grave problema de conectividad que surge al utilizar un nuevo protocolo IP, pero introducen sobrecarga en el rendimiento de las aplicaciones. Esta sobrecarga debe ser medida para, al menos, obtener datos que nos permitan:

[2]Ficheros .c y .h de la versión no gráfica, contadas utilizando la herramienta *unix* "wc".

- Comparar los distintos mecanismos entre sí de una forma cuantitativa.
- Comparar distintas implementaciones de un mismo mecanismo y mejorar así las implementaciones existentes.

Como se ha comprobado durante la elaboración de este proyecto, utilizando varios mecanismos de transición y con las aplicaciones migradas a IPv6 existentes en la actualidad es posible realizar el trabajo del día a día sobre IPv6 sin necesidad de utilizar IPv4 y de una forma totalmente transparente para el usuario final.

Apéndices

Apéndice A

Presupuesto.

En este capítulo se muestra detalladamente el coste total de la ejecución del presente proyecto. Para calcular este coste se ha dividido el proyecto en diferentes tareas que representan las distintas fases del desarrollo del proyecto y su evolución en el tiempo.

El presupuesto total de ejecución se obtiene sumando el coste individual de cada tarea con el coste del material utilizado durante la realización del proyecto.

Descomposición en tareas.

El proyecto ha sido dividido en diferentes actividades para su realización. Cada actividad se ha dividido en tareas. Para cada tarea se indican los objetivos, las dependencias detectadas, la duración y el trabajo asociado a la misma. El cálculo del trabajo se ha basado en una jornada de 8 horas/día y 22 días/mes.

Las actividades principales en las que se ha dividido el proyecto son las siguientes:

- Documentación y estudio del estado del arte.
- Pruebas de mecanismos de transición.
- Mantenimiento de la red de pruebas y de servicios de red.
- Implementación y desarrollo de aplicaciones.
- Documentación y memoria del proyecto.

A continuación se realiza un análisis de cada una de estas tareas.

Actividad 1: Documentación y estudio del estado del arte.

Tarea 1.1: Estudio sobre IPv6 y mecanismos de transición.

Descripción: En esta tarea se realiza un estudio de los temas relacionados con IPv6 y los mecanismos de transición definidos. La mayor fuente de información la constituye el IETF.

Objetivos:
- Comentar el estado del arte de IPv6 en general y de los distintos mecanismos de transición en particular.
- Familiarización y toma de contacto con la programación basada en sockets IPv6.
- Generación de un documento resumen como base para la memoria del presente proyecto.

Dependecias: Esta tarea comienza una vez aprobada la realización del proyecto.

Duración: 8 semanas.

Trabajo: ingeniero: 0.75 hombres-mes

Tarea 1.2: Estudio sobre migración de aplicaciones.

Descripción: Estudio del procedimiento para migrar aplicaciones cliente servidor de IPv4 a IPv6.

Objetivos:
- Estudio e identificacion de aplicaciones que todavía no están migradas.
- Estudio del mecanismo de migración de aplicaciones y de las APIs relacionadas.

Dependecias: Esta tarea comienza una vez aprobada la realización del proyecto.

Duración: 2 semanas.

Trabajo: ingeniero: 0.35 hombres-mes

Actividad 2: Pruebas de mecanismos de transición.

Tarea 2.1: Instalación de los mecanismos de transición.

Descripción: Identificación de los mecanismos de transición disponibles para cada plataforma e instalación del mecanismo para comprobar su correcto funcionamiento.

Objetivos:
- Comparar distintas implementaciones y extraer conclusiones sobre los mecanismos más adecuados basándose en distintas topologías y situaciones.
- Evaluar la facilidad de instalación del mecanismo en un entorno real.

Dependecias: Esta tarea comienza al terminar la tarea 3.1

Duración: 1 mes.

Trabajo: ingeniero: 0.75 hombres-mes.

Tarea 2.2: Prueba de los mecanismos de transición.

Descripción: Comprobar que el mecanismo permite interoperar de forma correcta entre IPv6 e IPv4.

Objetivos:
- Identificar posibles mejoras y/o fallos en la implementación así como su correcto funcionamiento.
- Ofrecer servicios de red a través de los distintos mecanismos y evaluar su funcionamiento.

Dependecias: Cuando termina la tarea 2.1

Duración: 2 meses.

Trabajo: ingeniero: 0.60 hombres-mes.

Actividad 3: Mantenimiento de la red de pruebas y de servicios de red.

Tarea 3.1: Instalación de máquinas.

Descripción: Instalar el software apropiado para realizar pruebas con IPv6. Principalmente FreeBSD y Linux parcheados para soportar opciones avanzadas.

Objetivos:
- Poder trabajar utilizando las últimas y mejores implementaciones de IPv6 disponibles.

Dependecias: Después de la tarea 1.2 y antes de la tarea 2.1

Duración: 1 semana.

Trabajo: ingeniero: 1 hombre-mes.

Tarea 3.2: Instalación de servicios.

Descripción: Se instalan distintos servicios para ser utilizados dentro de la red del proyecto LONG. Los principales servicios son: IRC, DNS. También se configura el encaminamiento utilizando BGP4+ y RIPv6.

Objetivos:
- Utilizar servicios basados en el nuevo protocolo IPv6.
- Configurar una red de pruebas que ofrezca servicios a los usuarios.

Dependecias: Al terminar la tarea 3.1

Duración: 2 meses.

Trabajo: ingeniero: 0.75 hombres-mes.

Tarea 3.3: Gestión y mantenimiento de la red.

Descripción: La red de pruebas se utiliza en otros proyectos. Se trata de asegurar que los servicios basados en IPv6 funcionan.

Objetivos:
- Actualizar los servicios según se presenten versiones mejoradas.
- Monitorizar la red y asegurar la conectividad global con la actual red IPv6.
- Aplicar parches de seguridad relacionados con servicios o implementaciones basadas en IPv6.

Dependecias: Comienza al finalizar la tarea 3.2

Duración: 8 semanas.

Trabajo: ingeniero: 0.40 hombres-mes.

Actividad 4: Implementación y desarrollo de aplicaciones.

Tarea 4.1: Aplicación MGEN.

Descripción: Utilización de la herramienta MGEN para adquirir experiencia en la migración de aplicaciones y en el uso del API avanzada de IPv6.

Objetivos:
- Migrar la aplicación MGEN a IPv6 y añadir nueva funcionalidad para poder enviar paquetes IPv6 con distintas cabeceras de extensión.
- Utilizar esta herramienta para medir el rendimiento de IPv6 bajo determinados escenarios de prueba.

Dependecias: Comienza al finalizar la tarea 1.2

Duración: 8 semanas.

Trabajo: ingeniero: 0.30 hombres-mes.
programador: 0.50 hombres-mes

Tarea 4.2: Herramienta "ping6to4".

Descripción: Implemetación de una herramienta que permite enviar "pings" utilizando el mecanismo de transición "6to4" sin tener el correspondiente interfaz implementado como parte del sistema operativo.

Objetivos:
- Averiguar si un extremo remoto que implementa "6to4" funciona correctamente.
- Implementar un método de encapsulación a nivel de usuario utilizando la API avanzada de IPv6.
- Adquirir experiencia para realizar otras migraciones más complicadas (MGEN).

Dependecias: Comienza al finalizar la tarea 1.2

Duración: 1 semana

Trabajo: ingeniero: 0.50 hombres-mes.
programador: 0.30 hombres-mes.

Actividad 5: Documentación y memoria del proyecto.

Tarea 5.1: Documentación del desarrollo del proyecto.

Descripción: Descripción del trabajo realizado desde la aprobación del proyecto y documentación del código implementado.

Objetivos:
- Descripción de los pasos de configuración necesarios para instalar los distintos mecanismos de transición.
- Redacción del documento Manual de Usuario de la aplicación MGEN para IPv6.

Dependecias: Durante la tarea 3.3

Duración: 3 semanas

Trabajo: mecanógrafo: 0.60 hombres-mes.
ingeniero: 0.20 hombres-mes.

Tarea 5.2: Redacción del documento final del proyecto.

Descripción: Redacción de la memoria final del proyecto, comentando los aspectos más relevantes de la realización del mismo. Para la realización de esta tarea se utilizará todo el *"know-how"* generado durante las fases previas.

Objetivos:
- Redacción de la memoria del proyecto fin de carrera.

Dependecias: Comienza al terminar la tarea 5.1

Duración: 8 semanas

Trabajo: mecanógrafo: 0.75 hombres-mes.
ingeniero: 0.10 hombres-mes.

Resumen del proyecto.

A continuación se presenta una tabla con la dureción de cada tarea, el trabajo asociado y el tiempo empleado en la misma. Se puede observar en la tabla A.1:

Actividades	**Duración**	**Trabajo**	**Total**
1. Documentación y estudio del estado del arte			
1.1 Estudio sobre IPv6 y mecanismos de transición			
Ingeniero	8 sem.	0.75 h-m.	264 h.
1.2 Estudio sobre migración de aplicaciones			
Ingeniero	2 sem.	0.35 h-m.	31 h.
		Total Actividad 1:	295 horas.
2. Pruebas de mecanismos de transición.			
2.1 Instalación de los mecanismos de transición.			
Ingeniero	8 sem.	0.75 h-m.	264 h.
2.2 Prueba de los mecanismos de transición.			
Ingeniero	8 sem.	0.60 h-m.	212 h.
		Total Actividad 2:	476 horas.
3. Mant. de la red de pruebas y de servicios de red.			
3.1 Instalación de máquinas.			
Ingeniero	1 sem.	1 h-m.	44 h.
3.2 Instalación de servicios			
Ingeniero	8 sem.	0.75 h-m.	264 h.
3.3 Gestión y mantenimiento de la red.			
Ingeniero	8 sem.	0.40 h-m.	140 h.
		Total Actividad 3:	448 horas.
4. Implementación y desarrollo de aplicaciones.			
4.1 Aplicación MGEN.			
Ingeniero	8 sem.	0.10 h-m.	35 h.
Programador	8 sem.	0.70 h-m.	246 h.
4.2 Herramienta “ping6to4”.			
Ingeniero	1 sem.	0.30 h-m.	13 h.
Programador	1 sem.	0.50 h-m.	22 h.
		Total Actividad 4:	316 horas.
5. Documentación y memoria del proyecto.			
5.1 Documentación del desarrollo del proyecto.			
Ingeniero	3 sem.	0.20 h-m.	26 h.
Mecanógrafo	3 sem.	0.60 h-m.	79 h.
5.2 Redacción del documento final del proyecto.			
Ingeniero	8 sem.	0.10 h-m.	35 h.
Mecanógrafo	8 sem.	0.75 h-m.	264 h.
		Total Actividad 5:	404 horas.
		TOTAL PROYECTO:	**1939 horas.**

Figura A.1: Resumen del proyecto.

Diagrama de Gantt.

En este apartado se presenta el diagrama de Gantt del proyecto, con el que se puede apreciar con más claridad las dependencias entre las tareas y su disposición en el tiempo.

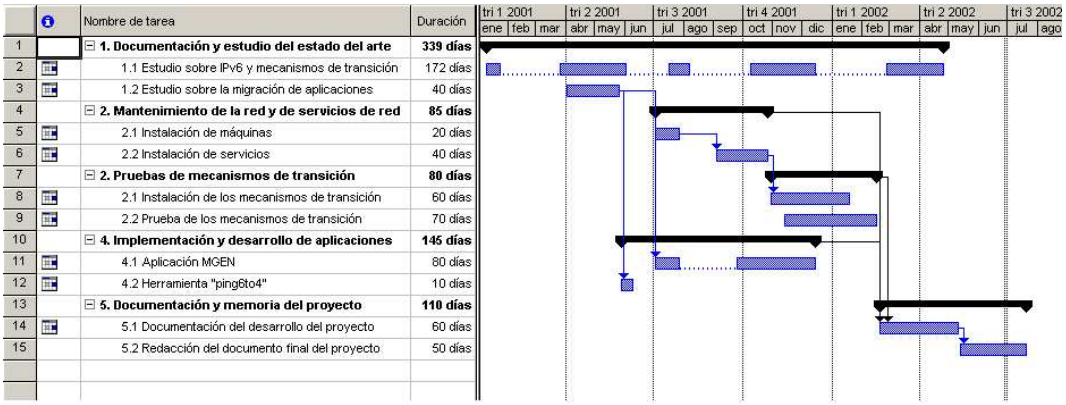

Figura A.2: Diagrama de Gantt del proyecto.

Costes del proyecto.

A continuación se presenta el coste total del proyecto, desglosado en costes de personal y costes de material.

Costes de personal.

Concepto	Cantidad	Coste unitario	Importe total
Ingeniero Informático	1328 horas	40 euros/hora	53.120 euros
Programador	268 horas	20 euros/hora	5.360 euros
Mecanógrafo	343 horas	10 euros/hora	3.430 euros
		Coste total de personal: 61.910 euros (10.300.957 ptas.)	

Figura A.3: Costes de personal.

Costes de material.

Concepto	Cantidad	Coste unitario	Importe total
Ordenador PC multimedia	6	1.200 euros	7.200 euros
Material fungible	1	500 euros	500 euros
			Coste total de material: 7.700 euros (1.258.072 ptas.)

Figura A.4: Costes de material.

Coste total del proyecto.

Concepto	Importe
Costes de personal	61.910 euros
Costes de material	7.700 euros
I.V.A. (16%)	11.137 euros
	TOTAL: 80.747 euros (13.435.170 ptas.)

Figura A.5: Coste total del proyecto.

El presupuesto total del proyecto asciende a **ochenta mil setecientos cuarenta y siete euros (80.747 euros = 13.435.170 pesetas)**.

Madrid, 10 de Abril de 2003

El ingeniero proyectista:

Fdo. Juan Francisco Rodriguez Hervella

Apéndice B

Funcionamiento interno de MGEN.

El objetivo de Mgen no es otro que el de generar paquetes en los instantes adecuados y gestionar la aparición y desaparición de flujos de forma dinámica, según se especifica en el script o a través de la línea de comando. Para ello, después de inicializar una serie de variables globales que se utilizan a través de toda la aplicación, Mgen debe analizar la línea de comandos y/o el archivo de configuración, y almacenar los datos en las estructuras adecuadas para poder generar posteriormente los paquetes IP. La estructura de datos utilizada por Mgen para almacenar los datos sobre los flujos se define en "mgenEvent.h" y es la siguiente:

```
typedef struct MgenEvent
{
    MgenMode    mode;        /* ITERATED or SCRIPTED */
    MgenCmd             cmd;
    unsigned long       id;          /* Flow's ID number */
    unsigned long       sequence;    /* Current sequence number */
    double              startTime;   /* Event start time (msec) */
    unsigned long       stopTime;    /* Event end time (msec) */
    MgenPattern         pattern;     /* PERIODIC or POISSON */
    float               interval;    /* Time between packets (msec) */
    int                 size;  /* Packet size (bytes) */
    struct sockaddr_in6 addr;  /* Flow's current destination address */
    struct in6_addr     b_addr;   /* Base address for "iterated flow */
    int                 n_groups; /* Number of groups for iterated flow */
    int                 count;  /* Current count for iterated flow */
    struct MgenEvent    *next;  /*Points to next event for_this_ flow*/
    struct MgenEvent    *parent;/* Points to event for previous flow */
    struct MgenEvent    *child; /* Points to event for next flow */

    /* extension headers for IPv6 */
    struct hbh_option options_buf[MAX_OPTIONS]; /*hop-by-hop options*/
    struct routing_h  routing;         /* routing header option. */
    struct in6_pktinfo sourceAddr;     /* Flow's current Source address,
                                          unspecified address by default. */
} MgenEvent;
```

Esta estructura se utiliza para encadenar eventos de un mismo flujo y también para encadenar la definición de distintos flujos. Contiene toda la información que se puede utilizar a la hora de definir las características de un flujo. Para enviar paquetes Mgen utiliza un único socket UDP de tipo AF_INET6.

Después de analizar el script y la línea de comandos, Mgen dispone de una lista encadenada de eventos y flujos que mantiene en la variable global "theList" y que presenta una estructura como la siguiente:

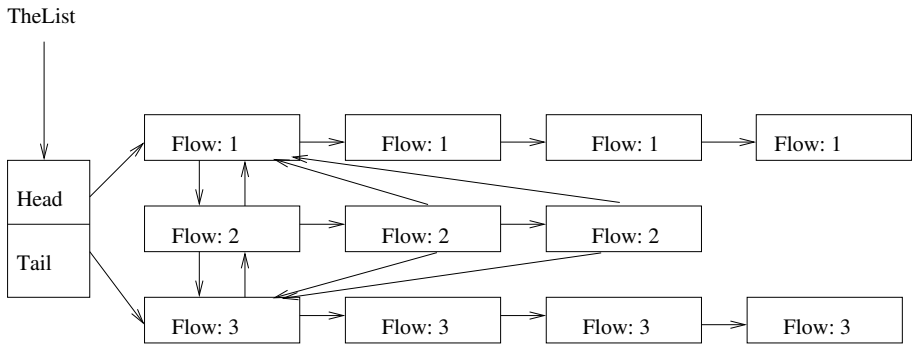

Figura B.1: Estructura de flujos de MGENv6.

En el dibujo para simplificar sólo se dibujan todos los punteros de enlace para el flujo número "dos". Cada estructura posee los siguientes punteros:

- *Next*: apunta al siguiente evento dentro de un flujo.
- *Child*: apunta al comienzo del siguiente flujo.
- *Parent*: apunta al comienzo del flujo previo.

Los flujos se encuentran ordenados en el tiempo, al igual que los eventos dentro de cada flujo. De esta forma nos ahorramos el recorrer toda la lista a la hora de planificar el siguiente flujo que debe entrar en acción. Si el flujo que estamos considerando tiene un tiempo de comienzo superior al actual, podemos asegurar que los flujos subsiguientes también lo tienen. Las funciones que manejan los flujos se encuentran en el archivo "mgenEvent.c".

Para enviar los paquetes en el instante adecuado, Mgen mantiene además otras dos variables globales:

1. *activeFlows*: lista de flujos que se encuentran activos en un determinado instante de tiempo.
2. *pendingFlows*: lista de flujos que todavía no han sido lanzados por no haber llegado su turno.

El bucle que envía los paquetes es el siguiente:

```
while(activeFlows.head || pendingFlows.head)
```

Dentro del bucle lo primero que se hace es recorrer la lista de flujos pendientes. Si no hay flujos activos, se añade a la lista el primer flujo pendiente, pero si ya existían flujos activos, los flujos pendientes sólo pasarán a ser activos en caso de que haya llegado su turno. Todo esto se comprueba mediante el siguiente trozo de código:

```
if ((nextEvent->startTime <= thisTime) || !activeFlows.head)
{
    /* A pending flow is ready, so fetch it */
    pendingFlows.head = nextEvent->child;
    if (pendingFlows.head)
        pendingFlows.head->parent = NULL;
    else
        pendingFlows.tail = NULL;
    /* Move pending flow to active flow list */
    AppendMgenFlow(nextEvent, &activeFlows);
    nextEvent = pendingFlows.head;
}
```

A continuación, se van seleccionando los flujos de la lista de flujos activa para enviarlos.

```
nextEvent = activeFlows.head;
while(nextEvent)
{
```

Si ya ha terminado el evento, se eliminan los eventos realizados hasta la fecha de la lista de eventos para dicho flujo, pero en el caso de que el mismo flujo se haya redefinido para un instante posterior, no se puede borrar el flujo por completo de la lista de flujos activos, sino que se deben actualizar los punteros, como se observa a continuación:

```
  if (nextEvent->stopTime <= thisTime)
  {
      /* if the Flow has more events, shitf them properly
         and return the next. Note that the resulting flow list are
         not sorted, but this is not important */

         nextEvent = DiscardMgenEvent(nextEvent, &activeFlows,
                                      &expiredEvents);
  }
```

En este instante, la lista de flujos activos puede quedar desordenada en el tiempo, debido a que se está insertando un flujo (una redefinición de un flujo anterior, mejor dicho) que puede comenzar en un instante que todavía no se ha producido. Esto no es un problema, puesto que todos los flujos activos en principio deben ser lanzados, independientemente del orden en la lista, pero nos obliga a comprobar que el flujo haya comenzado realmente. Por otro lado, si el evento se encuentra activo para el instante de tiempo actual (cosa que hay que comprobar), se construye el paquete y se envía, como se observa a continuación:

```
  else
  {
     /* Is event ready? It can happen that the event does not
        start yet. */
     if (nextEvent->startTime <= thisTime)
     {
        /* Send packet and calculate next packet transmission time.
           Note that we modified dinamically the startTime
           variable of the Event. */
        if (nextEvent->interval)
        {
           SendTxSocket( txSocket, txBuffer, nextEvent );
           switch(nextEvent->pattern)
           {
            case PERIODIC:
              nextEvent->startTime += nextEvent->interval;
              break;
            case POISSON:
              nextEvent->startTime += genexp(nextEvent->interval);
              break;
            }
        }
     }  /* end if(nextEvent->startTime <= thisTime) */
  }  /* end else(nextEvent->stopTime) */

  /*to progress throught the loop, we pass to the next flow of the
    active flow list. */
  nextEvent = nextEvent->child;

}  /* end while(nextEvent) */
```

Los eventos se mueven a lo largo del tiempo actualizando su instante de comienzo de acuerdo al intervalo de retransmisión que tengan definidos, que puede ser constante o siguiendo una distribución normal. Al final del bucle se selecciona el siguiente flujo de la lista de flujos activos.

Calculando el transcurso del tiempo.

En este apartado se comenta el control del tiempo utilizado en la herramienta Mgen a la hora de planificar los distintos flujos. En primer lugar, se inicializan las variables:

```
    nextTime = 0.0;   /* Start at time ZERO by default */

    /* Init timing */
    InitMgenTicker(&myTicker);

    gettimeofday(&beginTime, &tzone);
```

Se inicializa la variable utilizada para contar el instante de comienzo del siguiente flujo/evento (nextTime) y se recoge el instante de comienzo en la variable beginTime. Además, se inicializa la estructura que almacena el tiempo transcurrido desde la última captura de tiempo, y que se encuentra definida en "mgenWait.h" como:

```
typedef struct MgenTicker
{
    double count; /* Current bucket content (msec tick count) */
    struct timeval lastTime;   /* time state (system time) */
} MgenTicker;
```

A continuación, y en caso de que se haya dado una referencia de tiempo inicial distinta de cero, se actualiza la variable nextTime para que refleje el tiempo que debe transcurrir antes de que comience realmente el envío de flujos.

```
    /* If an absolute start time is given */
    if (absoluteStartTime)
    {
        if (beginTime.tv_sec > absoluteStartTime)
        {
            printf("MGEN: Specified startTime has already passed!\n");
            close(txSocket);
            exit(-1);
        }
        nextTime = (beginTime.tv_sec - absoluteStartTime) * 1000.0;
        nextTime += (beginTime.tv_usec + 500)/1000;
    }
```

Una vez dentro del bucle que recorre los flujos (mientras existan flujos, activos o pendientes), se inicializa la variable thisTime al valor de nextTime, y se pone el valor de nextTime a un valor infinito, puesto que según se recorren los flujos, nextTime se va actualizando con el objetivo de encontrar el instante de comienzo futuro más cercano al actual, o lo que es lo mismo, se realizará una búsqueda del mínimo instante de comienzo de entre todos los flujos activos, como se puede observar en el código que se presenta a continuación:

```
    while(activeFlows.head || pendingFlows.head)
    {
        thisTime = nextTime;
        nextTime = MGEN_TIME_MAX;
        /* pass pending flows to active flows, if it is appropiate */
        /* loop for active flows */
        while(nextEvent)
        {
           /* dispaching flows? */

           /* select the minimun startTime of all the active flows.
              Note that the resulting nextTime is always greater than
              "thisTime" because all the flows defined by the script (<=>
```

```
            pendingFlow initially ), are related to the initial
            "thisTime", and in pendingFlow list, they are sorted */

         if (nextEvent->startTime < nextTime)
            nextTime = nextEvent->startTime;
      }
```

Finalmente, una vez recorridos todos los flujos activos, el proceso se duerme el tiempo necesario para llegar a nextTime, es decir "nextTime-thisTime", pero además se tiene en cuenta el tiempo invertido en procesar los flujos activos, que puede resultar significativo, gracias a la variable myTicker, como se observa a continuación:

```
      /* Flow control ourselves, avoiding infinite wait.
         Note that nextTime is always greater than thisTime, because all
         the active flows, or start in the future, or start at the same
         time that "thisTime", and after dispatched it, his start time is
         incremented... */

      if (nextTime != MGEN_TIME_MAX)
         MgenWait((nextTime-thisTime), &myTicker);
   }
```

Puede ocurrir que MgenWait() no duerma el proceso. Esto ocurrirá cuando haya transcurrido más tiempo del que necesitamos dormir (es decir, que estamos lanzando los paquetes con retraso). La desviación se corregirá con el tiempo siempre y cuando la velocidad de envío de paquetes no sobrepase la capacidad de ejecución del proceso.

Apéndice C

Funcionamiento interno de DREC.

En este apartado se presenta la herramienta Drec. Drec es la parte del sistema encargada de recibir los paquetes que genera Mgen.

Para poder realizar cálculos, obtener estadísticas y agrupar resultados, se necesita la existencia de una herramienta capaz de recibir todos los paquetes que envía Mgen. Drec se utiliza como una herramienta de *log* que puede escuchar varias direcciones multicast, que es capaz de detectar los distintos flujos que genera Mgen, graba marcas de tiempo que permiten calcular retardo (con los rejoles de las máquinas origen y destino sincronizados mediante alguna fuente de referencia externa, utilizando ntp, por ejemplo) y detectar cabeceras de extensión en caso de que dichas cabeceras se pasen al nivel de aplicación.

Drec utiliza el siguiente esquema de ejecución para recibir los distintos flujos. Por un lado, debe procesar los eventos relacionados con los grupos multicast a los que debe unirse/abandonar dinámicamente, y por otro lado debe procesar los paquetes que reciba y escribirlos en el archivo de log. La forma de procesar sus propios eventos es muy parecida a la utilizada por Mgen:

```
/* Process events in queue while listening for traffic */
nextEvent = eventList.head;
while(nextEvent)
{
  /* Process any events whose time has come */
  while(nextEvent && (nextEvent->time <= thisTime))
  {
    gettimeofday(&rxTime, &tzone);

    switch(nextEvent->cmd)
    {
        case JOIN:
           /* insert the new group and write to the log file */
        case LEAVE:
           /* delete the group and write to the log file */
    }

    nextEvent = DiscardDrecEvent(nextEvent, &eventList, &eventTrash);
  }
```

A continuación, y mientras quede algún evento, se utiliza el tiempo que debe transcurrir hasta que se produzca el siguiente evento para escuchar en el conjunto de sockets mediante una llamada a select() bloqueante, con un timeout igual al tiempo que resta hasta el siguiente evento de Drec.

```
  if (nextEvent)
  {
    waitTime = nextEvent->time - thisTime;

    /* select() on recv file descriptor(s), with "waitTime" timeout */
    while (waitTime > theTicker.count)
    {
        /* Set "select()" timeout */
        ...
        /* Fill fd_set with file descriptors from our socketList */
        ...
        /* Block on select until packet or timeout */
        result = select(max_fd, (fd_set *) &fdset, (fd_set *) NULL,
                        (fd_set *) NULL, &timeout);
```

En este punto, una vez que la llamada bloqueante a select() ha retornado, recorremos la lista de sockets y en caso de que alguno de ellos haya recibido algún paquete, se procesa y se almacena en el archivo de log:

```
        /* Check socket list, handling any fd's that are ready */
        nextSocket = socketList;
        while(nextSocket)
        {
          if (FD_ISSET(nextSocket->fd, &fdset))
          {
              /* process the received packet */
              ...
          }

          /* check the next socket */
          nextSocket = nextSocket->child;
        }

      UpdateMgenTicker(&theTicker);
    }  /* end while(waitTime > theTicker.count) */

    theTicker.count -= waitTime;
    thisTime += waitTime;
  }  /* end if(nextEvent) */

}  /* end while(nextEvent) */
```

El cómputo de tiempo utiliza las mismas llamadas y estructuras que Mgen, variando únicamente la estructura que mantiene la lista de eventos, que se encuentra definida en <drecEvent.h> y es la siguiente:

```
typedef struct DrecEvent
{
    DrecCmd             cmd;    /* DREC script command */
    unsigned long       time;   /* event execution time (msec) */
    struct sockaddr_in6 addr; /* group IP address  */
    struct DrecEvent    *parent;
    struct DrecEvent    *child;
} DrecEvent;
```

Además, en el archivo "drec.c" se encuentra la estructura que almacena la lista de sockets en los que se escucha:

```
typedef struct DrecSocket
{
    int                 fd;
    unsigned short      port;
#ifdef _LIMIT_GROUPS
    int                 g_count;
    struct in6_addr     group[IP_MAX_MEMBERSHIPS];
#endif _LIMIT_GROUPS
    struct DrecSocket   *parent;
    struct DrecSocket   *child;
} DrecSocket;
```

Por línea de comandos, para escuchar más de un grupo multicast, se utiliza el valor del número de grupos (opción n") para escuchar en un rango de direcciones que va desde la dirección inicial especificada con b"hasta la dirección obtenida sumando el valor de la opción n^{a} dicha dirección inicial.

Grupos Multicast.

En Drec, para unirse/abandonar grupos multicast se utilizan las siguientes funciones:

- Join Group()
- LeaveGroup()

Para escuchar grupos multicast se utilizan las funciones del sistema setsockopt() con los siguientes valores, dependiendo del tipo de sistema operativo, lo que nos obliga a utilizar una compilación condicional utilizando directivas del preprocesador, como se observa en el siguiente trozo de código:

```
      mreq.ipv6mr_multiaddr = group_addr;
      mreq.ipv6mr_interface = if_nametoindex( interfaceName );
#ifdef IPV6_JOIN_GROUP
      setsockopt(nextSocket->fd, IPPROTO_IPV6, IPV6_JOIN_GROUP,
                  (char *)&mreq, sizeof(mreq) )
#else
      setsockopt(nextSocket->fd, IPPROTO_IPV6, IPV6_ADD_MEMBERSHIP,
                  (char *)&mreq, sizeof(mreq) )
#endif
```

Para abandonar grupos multicast, se realiza de forma análoga utilizando las constantes apropiadas, que en este caso también difieren según el sistema operativo de la máquina, como se observa a continuación:

```
      mreq.ipv6mr_multiaddr     = group_addr;
      mreq.ipv6mr_interface     = if_nametoindex( interfaceName );
#ifdef IPV6_LEAVE_GROUP
      setsockopt(theSocket->fd, IPPROTO_IPV6, IPV6_LEAVE_GROUP,
                  (char *)&mreq, sizeof(mreq) )
#else
      setsockopt(theSocket->fd, IPPROTO_IPV6, IPV6_DROP_MEMBERSHIP,
                  (char *)&mreq, sizeof(mreq) )
#endif
```