PROYECTO FIN DE CARRERA INGENIERÍA INFORMÁTICA

Generación, análisis y optimización de escenarios de migración de redes IPv4/IPv6 mediante programación funcional

Por José Franco Campos
5º de Ingeniería Informática
Diciembre 2008

Dirigido por D. Alfonso Gazo Cervero

Escuela Politécnica

Universidad de Extremadura

Escuela Politécnica

Ingeniería Informática

Proyecto Fin de Carrera

Generación, análisis y optimización de escenarios de migración de redes IPv4/IPv6 mediante programación funcional

Autor: José Franco Campos

Fdo.:

Director: Alfonso Gazo Cervero

Fdo.:

Tribunal Calificador

Presidente:
Fdo:

Secretario:
Fdo.:

Vocal:
Fdo.:

CALIFICACIÓN:
FECHA:

Agradecimientos

Gracias a mis padres, por su paciencia y por haberme permitido desarrollar mi afición por la informática, y a mi hermano Rafa, por haberme tenido que aguantar estos años. Todo el dinero invertido durante años en hardware, libros y revistas ha dado sus frutos.

Tengo que dar también las gracias a mis compañeros de clase, que han hecho la carrera mucho más llevadera. César, Mario, Antonio, Miguel, Pepe, Carlos, José, Hugo, Román... Sin vosotros, todas esas horas en clase o en la cafetería no hubieran sido lo mismo.

Gracias a Alfonso, por haber sido el mejor tutor de proyecto que podría desear, y a todos los buenos profesores que han sabido impartir asignaturas interesantes y llenas de conocimientos.

Por último, sólo me queda dar las gracias a la comunidad de Haskell, por desarrollar un lenguaje y unas herramientas excelentes y libres. Es el lenguaje que más ha cambiado mi forma de pensar desde que empecé con BASIC hace ya 14 años.

Índice general

Índice de figuras

Parte I

Presentación

1 Introducción y objetivos

Cuando el protocolo IPv4 se estandarizó a principios de los años 80, el uso de Internet se limitaba a universidades y organismos gubernamentales. Esto motivó algunas decisiones de diseño, como usar direcciones de sólo 32 bits. Sin embargo, al empezar a extenderse su uso comercial, el número y tamaño de las redes aumentó de manera explosiva. Esto ha provocado que el número de direcciones empiece a agotarse. Aunque algunos mecanismos como CIDR, NAT o el hosting virtual han logrado aliviar parcialmente el problema, los últimos estudios indican que el espacio de direcciones se agotará entre febrero del 2010 y mayo del 2011.

Para resolver este y otros problemas, el IETF empezó a trabajar en la definición empezó a trabajar a mediados de los años 90 en la siguiente revisión del protocolo IP, conocido como IPv6[1]. Esta nueva versión ofrece ventajas significativas, como un espacio de direcciones de 128, un enrutado más sencillo, o un formato de cabecera más flexible. Por desgracia, estas mejoras se obtienen a costa de perder la compatibilidad: los routers IPv4 no son capaces de procesar paquetes IPv6. Esto implica que, para poder aprovechar todas las ventajas del nuevo protocolo, habría que actualizar todos los routers, lo que supone un coste prohibitivo.

Para facilitar la migración se han definido varios mecanismos de transición, en forma de túneles que encapsulan una versión de IP dentro de otra, y sistemas de traducción de paquetes. De esta manera, el paso de IPv4 a IPv6 se puede hacer de manera gradual. Aun así, en una red grande puede ser difícil elegir que mecanismos usar. Por tanto, interesa desarrollar una metodología que permita formalizar la toma de decisiones, y de esta manera poder desarrollar una herramienta informática que ayude al administrador de red.

Este trabajo ya se llevó a cabo en el proyecto fin de carrera de Jorge García Domínguez. En él se obtuvo una aplicación, escrita en el lenguaje Java, que permitía resolver este problema. También se dejaron una serie de propuestas para futuras mejoras. El objetivo del presente proyecto es completar esas tareas que quedaron pendientes, usando además el lenguaje de programación funcional Haskell. Los lenguajes funcionales se remontan al Cálculo Lambda, por lo que su historia es tan vieja como la de los lenguajes imperativos. Pese a ello, su uso a nivel comercial ha sido reducido, por lo que hay un gran desconocimiento en torno a ellos. Implementando nuestra aplicación en Haskell pretendemos estudiar su viabilidad a la hora de implementar aplicaciones complejas, y comparar su productividad frente a los lenguajes imperativos.

[1] La versión 5 fue un protocolo experimental para transmisión de flujos de datos en tiempo real. No se llegó a usar nunca de manera pública.

En resumen, los objetivos concretos del proyecto son:

- Estudiar las ventajas que aporta IPv6 con respecto a IPv4.
- Estudiar los mecanismos existentes para migrar de IPv4 a IPv6.
- Formalizar matemáticamente el problema, de tal manera que pueda ser tratado por un ordenador.
- Estudiar la viabilidad de los lenguajes de programación funcionales para hacer frente a este tipo de problemas.
- Desarrollar una aplicación que permita a un administrador de red la ruta óptima dentro de una red heterogénea.
- Analizar el resultado obtenido, y decidir si recomendar el uso de Haskell en desarrollos futuros.

Parte II

Estudio del problema

2 Limitaciones de IPv4

IPv4 fue diseñado para trabajar con redes más pequeñas de las que existen hoy en día. Su principal limitación es el reducido espacio de direcciones. Mientras que en los Estados Unidos, que es donde surgió Internet, esto no es un problema, los países que entraron tarde se encuentran con que reciben bloques de direcciones muy pequeños en comparación con su población.

IPv4 utiliza direcciones de 32 bits, lo que ofrece un total de 4.294.967.296 (2^{32}) direcciones. Sin embargo, no todas las direcciones pueden usarse, ya que parte del espacio está reservado para redes locales, grupos de multicasting, y extensiones futuras. El espacio restante se asigna de manera jerárquica, por lo que algunas direcciones se desperdician.

En el protocolo IP las direcciones se dividen en dos partes: el identificador de la red, y el identificador del nodo. Originalmente estos campos tenían un ancho de 8 y 24 bits respectivamente, pero pronto se vio que 256 redes era una cantidad muy reducida. Para superar este límite, se definió el direccionamiento por clases, que divide el espacio de direcciones en cinco clases (A, B, C, D, y E), cada una de las cuales se identifica por un prefijo:

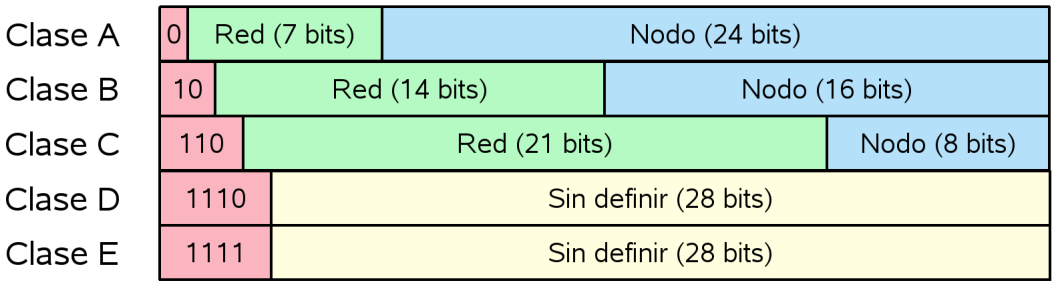

Figura 2.1: Esquema de direccionamiento IPv4 por clases

Este esquema es muy sencillo de implementar, pero tiene el inconveniente de que desperdicia muchas direcciones. El bloque asignable más pequeño es de 256 direcciones, por lo que una red que tuviera, por ejemplo, 30 nodos, desaprovecharía el 88 % del espacio. Para mitigar este problema, en 1993 se introdujo el esquema CIDR (*classless interdomain routing*), que permite dividir las direcciones en segmentos de longitud variable. De esta manera, nuestra red de 30 nodos recibiría un bloque de 5 bits, es decir, 32 direcciones. Un bloque se puede a su vez subdividir en varios bloques más pequeños, y varios bloques contiguos pertenecientes a la misma organización se pueden agregar en un superbloque, con el objetivo de reducir el número de entradas en las tablas de enrutado.

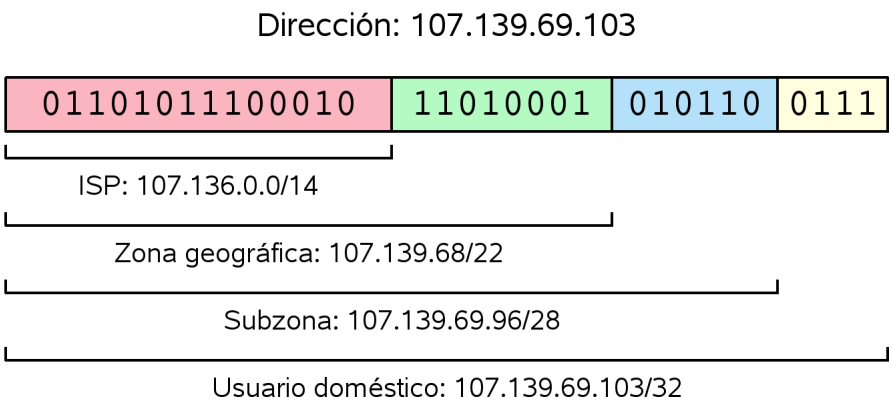

Figura 2.2: Ejemplo de direccionamiento CIDR

Aún así, el número de direcciones sigue siendo limitado. Para evitar tener que volver a modificar la estructura de la red, se han implementado varias soluciones software:

- Las direcciones pertenecientes al espacio reservado para redes locales no se pueden direccionar de manera global. Esto permite que varias redes privadas compartan el mismo rango de direcciones sin que surja ningún conflicto entre ellas.
- La traducción de direcciones (NAT, *network address translation*) es un mecanismo que permite que todos los nodos de una red local accedan a Internet por medio de una única dirección pública. El router que conecta la red se encarga de reescribir las direcciones presentes en las cabeceras de los paquetes, identificando los nodos de la red local por medio de los puertos TCP/UDP. Este mecanismo se usa mucho en las redes domésticas y de pequeñas empresas. Tiene el problema de que si un nodo de la red local quiere actuar como servidor, hay que configurarlo de manera explícita en el router.
- El hosting virtual permite que varios servidores de servicios web, con diferentes dominios, convivan en la misma dirección IP. El servidor concreto se identifica por medio de la URL que aparece en la cabecera HTTP. Este mecanismo tiene el problema de que si los servidores DNS fallan y el cliente tiene que conectarse usando directamente la dirección IP, el servidor no podrá identificar el dominio. Otro problema es que no pueden coexistir varios servidores que usen SSL en la misma dirección IP.

Por último, IPv4 añade algunas características que, aunque útiles, resultan desfasadas. Una de ellas es la detección de errores: si bien el algoritmo usado no resulta muy costoso, a día de hoy resulta redundante porque esta funcionalidad también la implementa la capa de enlace. Otra característica redundante es la fragmentación y el reensamblado de datagramas: supone una gran carga de trabajo, por lo que, si se puede, se delega a la capa de transporte.

3 El protocolo IPv6

3.1. Orígenes

Cuando a mediados de la década de los 90 se introdujo el esquema CIDR, quedaba claro que sólo era un parche temporal. Si se deseaba que Internet siguiera creciendo, haría falta un rediseño completo. A finales de 1992 el IETF anunció el establecimiento del grupo de trabajo "IP Next Generation" (IPng), y solicitó el envío de artículos con propuestas de diseño del futuro protocolo. En 1996 se publicaron los primeros RFC que definían el futuro protocolo IPv6 [DH95, Cra96]. Estos documentos incluyen detalles como el nuevo formato de datagrama, o el esquema de asignación de direcciones.

El formato de datagrama de IPv6 no es compatible con el de IPv4, por lo que es necesario adaptar todos los nodos de la red. Este proceso es muy costoso, por lo que para simplificar la migración se publicaron una serie de documentos que definen varios "mecanismos de transición". Estos mecanismos permiten que nodos IPv6 puedan acceder a servicios IPv4, y que conectarse con otros nodos IPv6 atravesando zonas IPv4. Estudiaremos los diversos mecanismos existentes en el siguiente capítulo.

Por otro lado, con el fin de experimentar con el funcionamiento de IPv6, se crearon redes como 6bone, una plataforma de ámbito mundial que permitía a islas IPv6 dispersas comunicarse unas con otras utilizando la infraestructura IPv4 existente, por medio de túneles IPv6-sobre-IPv4. En su momento cumbre, 6bone englobaba 150 redes que cubrían más de 1000 subredes en 50 países. Una vez que la disponibilidad de IPv6 estuvo asegurada y empezaron a surgir redes paralelas, se dejó de expandir 6bone, hasta finalmente parar su uso.

3.2. Direccionamiento

La principal diferencia con respecto a IPv4 es que IPv6 ofrece un espacio de direcciones de 128 bits, lo que proporciona un total de $3,4 \times 10^{38}$ direcciones. Para hacernos una idea, se podrían dar más de mil billones de direcciones a cada estrella existente en el universo.

Las direcciones IPv6 se representan como una secuencia de 8 números hexadecimales de 4 dígitos (16 bits), separados por un carácter de dos puntos. Para abreviar, si dos o más números contiguos son 0, se puede usar la notación "::", pero sólo una vez por dirección para

evitar ambigüedades. Veamos un ejemplo de dirección IPv6, y su representación compacta:

$$\text{fe80} : 0000 : 0000 : 0000 : 0219 : \text{dbff} : \text{fe4c} : \text{09ed}$$
$$\Downarrow$$
$$\text{fe80} :: 219 : \text{dbff} : \text{fe4c} : \text{9ed}$$

Las direcciones IPv6 se clasifican en tres tipos:

- UNICAST: identifican a una única interfaz de red. Un paquete enviado a una dirección de este tipo será entregado a la interfaz correspondiente.
- ANYCAST: identifican a un grupo de interfaces de red. Un paquete enviado a una dirección de este tipo será entregado a una única interfaz, normalmente a la más cercana al nodo de origen. Su formato es idéntico al de las direcciones unicast, sólo se diferencian en que aparecen de manera distinta en las tablas de enrutado de routers diferentes.
- MULTICAST: identifican a un grupo de interfaces de red. Un paquete enviado a una dirección de este tipo será entregado a todas las interfaces del grupo. Su estructura es diferente al resto de direcciones. El primer octeto es todo unos; es decir, todas tienen el prefijo ff00 :: /8. El segundo octeto indica el ámbito de la dirección. Los 112 bits restantes identifican al grupo de multicasting.

En nuestro proyecto sólo trabajaremos con direcciones unicast, por lo que vamos a explicarlas con más detenimiento. Al igual que ocurría en IPv4, las direcciones IPv6 se dividen en dos partes: identificador de la red, e identificador del nodo. Ambas tienen un ancho de 64 bits. El identificador del nodo se calcula a partir de la dirección física de la interfaz de red, lo que facilita la configuración automática de la red y evita el uso de ARP. El identificador de red se puede subdividir a su vez en varios campos para facilitar el enrutamiento jerárquico. Un posible esquema era el usado por 6bone [HFP98]:

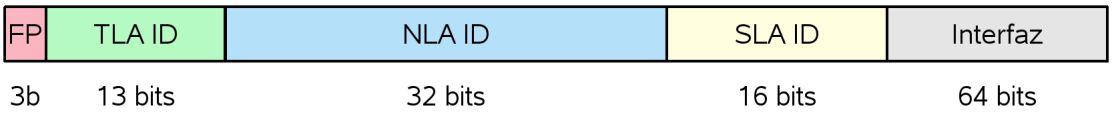

Figura 3.1: Formato de direcciones IPv6

El significado de cada uno de estos campos es el siguiente:

- FP (*Format Prefix*): indica el tipo de dirección. Todas las direcciones *unicast* globales usan el prefijo 001.
- TLA ID (*Top-Level Aggregation Identifier*): este campo identifica a la entidad agregadora de nivel superior. Estas entidades son los ISP que forman el backbone, como Sprint o AT&T. La longitud de este campo es tan reducida para que las tablas de enrutado de los routers de nivel superior, que son los que más tráfico soportan, puedan ser más pequeñas.

- NLA ID (*Next-Level Aggregation Identifier*): este campo identifica a una entidad de nivel medio, normalmente un ISP de menor tamaño o una empresa de gran tamaño que necesita direccionar varias redes. El espacio asignado a un NLA se puede dividir en varias partes, cada una de las cuales será asignada a un cliente.
- SLA ID (*Site-Level Aggregation Identifier*): este campo identifica las subredes pertenecientes a una organización. No permite delegar direcciones a otras entidades. En este campo se encuentran grandes empresas y pequeños ISP locales.
- Interfaz: este campo es equivalente al identificador de nodo de IPv4. Identifica a una interfaz de red, y normalmente se calcula a partir de la dirección física de ésta.

Antes hemos comentado que las direcciones que empiezan por 001 se reservan para *unicast* global. En la siguiente tabla se muestran algunos de los tipos de direcciones reservadas y su uso [HD06]:

Rango	Prefijo	Notas
Dirección no especificada	:: /128	Esta dirección no se puede asignar a ninguna interfaz, sólo se usa antes de que el software haya sido configurado.
Dirección de *loopback*	:: 1/128	Identifica al propio nodo o *localhost*. Cualquier paquete enviado a esta dirección volverá al propio nodo. Es equivalente a la dirección 127,0,0,1 en IPv4.
Autoconfiguración de redes locales	fe80 :: /10	Indica que la dirección sólo se puede usar en el mismo segmento de una red local.
Direcciones locales únicas	fc00 :: /7	Estas direcciones sólo se pueden usar dentro de un conjunto de sitios; por ejemplo, la red privada de una empresa. No deben salir a Internet.
Direcciones de *multicasting*	ff00 :: /8	Los 112 bits bajos identifican al grupo de multidifusión.
Direcciones normales	2001 :: /16	Identifica una dirección global normal.
Mapeado de direcciones IPv4	:: ffff : 0 : 0/96	Se usa para asignar una dirección IPv6 a un nodo IPv4 (ver el capítulo 5).
Túneles Teredo	2001 :: /32	Identifica la dirección como un túnel Teredo (ver el capítulo 5).
Túneles 6to4	2002 :: /16	Identifica la dirección como un túnel 6to4 (ver el capítulo 5).
ORCHID	2001 : 10 :: /28	Se usan para distribuir identificadores criptográficos.
Documentación	2001 : db8 :: /32	Siempre que se desee dar una dirección de ejemplo, deberá pertenecer a este prefijo.
Direcciones de prueba de 6bone	3ffe :: /16	Sólo se usaban dentro de 6bone. Se han dejado de conceder.

Figura 3.2: Rangos de direcciones IPv6 reservados

3.3. Enrutamiento

Otro objetivo de IPv6 es facilitar el enrutamiento. Al asignar las direcciones, es interesante seguir un esquema jerárquico, ya que así un router de nivel superior puede agrupar múltiples redes con un prefijo común, y delegar parte del trabajo en routers de nivel inferior. De lo contrario los routers tendrían que conocer la dirección de todas las redes existentes en el mundo, lo cual no es factible.

Ya vimos que IPv4 utiliza el esquema CIDR para esta tarea. Aunque flexible, CIDR no está carente de problemas. El primero de ellos es que es una técnica muy compleja, ya que, al ser los prefijos de longitud variable, exige el uso de estructuras de datos más sofisticadas. Otro problema es que CIDR se empezó a usar cuando una buena parte del espacio de direcciones ya había sido asignado, por lo que hay grandes bloques que quedan fuera de la jerarquía y no se usan de manera óptima.

IPv6 pretende resolver este problema dividiendo las direcciones en varios segmentos de longitud fija. Como ya hemos visto, los 64 bits de menor peso identifican un nodo dentro de una red. Se generan a partir de la dirección física de la interfaz de red, lo que permite incluir los 48 bits de las red Ethernet, y muchos otros protocolos de enlace más. Los 64 bits de mayor peso identifican a la red, y se subdividen a su vez en varios campos cortos de longitud fija (TLA ID, NLA ID, SLA ID). De esta manera se simplifica el trabajo de los routers. Si una organización necesita más flexibilidad, puede subdividir el campo NLA ID como desee.

Otra ventaja de IPv6 es que la fragmentación y reensamblado de los paquetes sólo se puede realizar en el origen y el destino respectivamente, liberando de esta carga a los routers intermedios. Para descubrir la longitud máxima de los datagramas se usa el servicio P*ath MTU Discovery* [MDM96]. Este servicio empieza enviando datagramas con la opción DF (*don't fragment*) activa. Si alguno de los routers intermedios necesita fragmentar el datagrama, devuelve un mensaje de error al origen. El proceso se repite, reduciendo la longitud de los datagramas, hasta que se encuentra un valor de MTU apropiado. Si el valor cambia una vez que se ha establecido la comunicación, se ejecutará todo el proceso de nuevo. La MTU mínima para un dispositivo que soporte IPv6 debe ser de 1280 octetos.

3.4. Formato del datagrama

IPv6 simplifica el formato del datagrama con respecto a IPv4. Esto se consigue de varias maneras:

- Se elimina el campo OPCIONES de la cabecera, sustituyéndose por cabeceras de extensión que no necesitan ser procesadas por todos los routers. La cabecera pasa a ser de tamaño fijo, eliminándose también el campo LONGITUD.
- Se sustituye el campo TIPO DE SERVICIO por un nuevo mecanismo de QoS basado en etiquetas de flujos.

- El campo TTL, que históricamente anotaba el tiempo máximo de vida de los paquetes en segundos, pasa a llamarse LÍMITE DE SALTOS.
- Desaparece el campo de detección de errores, cuya tarea se delega a la capa de enlace.
- Desaparecen los campos de fragmentación.

Gracias a estas modificaciones, el formato resultante es muy sencillo:

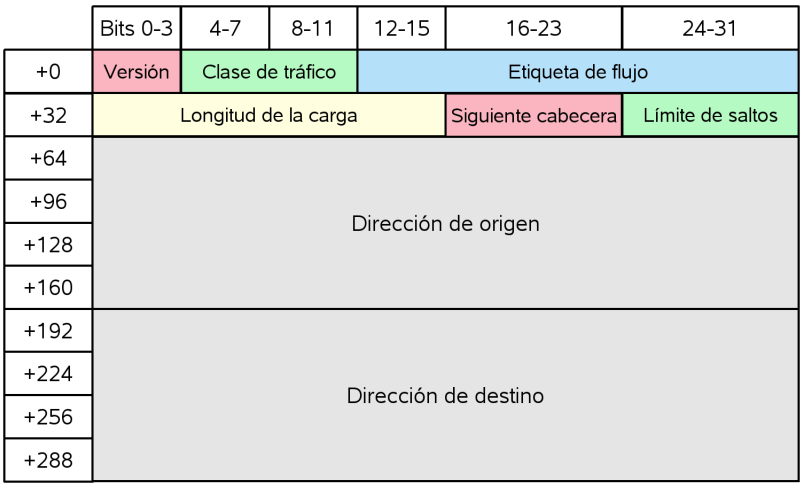

Figura 3.3: Formato de la cabecera IPv6

El significado de cada campo es el siguiente:

- VERSIÓN (4 bits): versión del protocolo IP, su valor es de 6.
- CLASE DE TRÁFICO (8 bits): permite al nodo de origen especificar la prioridad del datagrama. Su uso aún se está estudiando.
- ETIQUETA DE FLUJO (20 bits): permite identificar flujos de datos en tiempo real que deben recibir un tratamiento especial. Actualmente no se usa.
- LONGITUD DE LA CARGA (16 bits): longitud del resto del datagrama, excluida la cabecera, en octetos. Si su valor es cero, indica que es un datagrama *jumbo*; la longitud real estará en una cabecera de extensión.
- SIGUIENTE CABECERA (8 bits): identifica el tipo de protocolo encapsulado. Normalmente será un protocolo de transporte, pero pueden ser cabeceras con opciones adicionales. Usa los mismos valores que IPv4.
- LÍMITE DE SALTOS (8 bits): especifica el número de enlaces que puede seguir un datagrama antes de ser descartado.
- DIRECCIÓN DE ORIGEN (128 bits): la dirección del remitente del paquete.
- DIRECCIÓN DE DESTINO (128 bits): la dirección del destinatario del paquete.

A modo de conclusión, aunque las direcciones IPv6 son cuatro más grandes que las IPv4, el tamaño de la cabecera es sólo el doble (40 octetos frente a 20 octetos), y el número de campos obligatorios se reduce de doce a ocho.

3.5. Compatibilidad de direcciones

Para facilitar el proceso de migración de IPv4 a IPv6, y garantizar la compatibilidad de ambos protocolos mientras convivan, se han reservado varios rangos de direcciones IPv6. Cada uno de estos rangos se ha asignado a un mecanismo de transición particular, que describiremos con más detenimiento en el capítulo 5. Por el momento haremos un breve resumen. Partiendo de una dirección IPv4 x.y.z.w, se pueden construir las siguientes direcciones especiales:

- IPV4-MAPPED IPV6: usan el formato :: ffff : x.y.z.w, e identifican a un nodo IPv4 en un nodo IPv6. Nunca se usan como dirección de origen o destino de un paquete.
- IPV4-COMPATIBLE IPV6: usan el formato :: x.y.z.w, y sirven para comunicar nodos IPv4 e IPv6 por medio de túneles. Estas direcciones están obsoletas y no deben usarse nunca.
- IPV4-TRANSLATABLE IPV6: usan el formato :: ffff : 0 : x.y.z.w, y se emplean en el mecanismo de migración SIIT.
- 6TO4: usan el formato 2002 : xxyy : zzww :: /48, donde xxyy : zzww es la representación hexadecimal de la dirección. Se emplean en el mecanismo de tunelado *6to4*.
- 6SOBRE4: usan el formato :: xxyy : zzww, donde xxyy : zzww es la representación hexadecimal de la dirección, al que añaden el prefijo correspondiente a la interfaz. Por ejemplo, la dirección de enlace local del nodo 112,151,16,72 se representaría como fe80 :: 7097 : 1048. Estas direcciones se utilizan en el mecanismo de tunelado definido en el RFC 2529, así como en los mecanismo NAT-PT y TRT.
- ISATAP: usan el formato :: 5efe : x.y.z.w, al que añaden el prefijo correspondiente a la interfaz; por ejemplo, fe80 :: 5efe : 192,168,0,1. Se usan en el mecanismo de tunelado *Intra-Site Automatic Tunnel Addressing Protocol*.

3.6. El sistema de nombres de dominio

Las direcciones IPv6 se representan en el sistema de nombres de dominio (DNS) mediante registros AAAA, también llamados *quad-A* por analogía con los registros A de IPv4. El servicio de consultas inversas se aloja en el dominio ip6.arpa. El formato de los registros y el funcionamiento general del servicio, así como los cambios necesarios para adaptar el sistema DNS, se describen en el RFC 3596 [THKS03].

Durante el diseño de la arquitectura DNS para IPv6 se consideró otro sistema basado en registros A6. Este sistema se describe en el RFC 2874 [CH00]; sus virtudes y defectos frente a los registros AAAA se estudian en el RFC 3364 [Aus02]. De momento se han relegado a un estado experimental, y no se recomienda usarlos.

En junio del 2004 la IANA anunció que se habían actualizado los servidores DNS raíz para soportar el nuevo formato de registros, por lo que se pueden empezar a asignar direcciones IPv6 a dominios existentes. Mientras dure la transición, los nodos que soporten IPv6 deberán elegir que protocolo usar al acceder a un dominio remoto. El RFC 3484 [Dra03] describe el criterio a seguir, incluyendo la selección de dirección a partir de la información obtenida del DNS.

4 Transición de IPv4 a IPv6

4.1. Introducción

Para facilitar la convivencia entre ambos protocolos, se han definido una serie de mecanismos de transición que permiten que un nodo sólo-IPv6 pueda acceder a servicios IPv4, y que redes IPv6 aisladas puedan comunicarse entre sí atravesando zonas IPv4. Los documentos RFC 2185 [CH97] y RFC 4213 [NG05] definen las siguientes técnicas:

- Dual Stack: el mecanismo más sencillo consiste en que los nodos soporten ambos protocolos por medio de una pila TCP/IP dual.
- Tunelado: permiten encapsular los datagramas IPv6 dentro de IPv4, de tal manera que este último actúa como si fuera la capa de enlace.
- Traducción de protocolos: estas técnicas reescriben los paquetes, convirtiendo las cabeceras de una versión a otra.

En las siguientes secciones describiremos cada técnica con mayor detenimiento.

4.2. Dual Stack (RFC 4213)

El mecanismo más sencillo consiste en instalar ambas pilas de protocolos en un mismo nodo. Ambos protocolos operarán en paralelo. Las capas superiores elegirán el protocolo a usar en función de la dirección devuelta por el DNS, es decir, según el protocolo que soporte la aplicación de destino. El sistema operativo enviará el paquete a la pila que se le solicite.

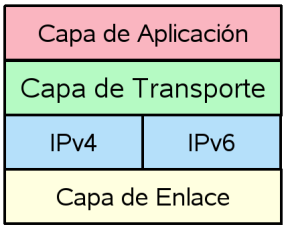

Figura 4.1: Arquitectura de pila TCP/IP dual

Resulta relativamente sencillo escribir una pila dual, ya que el único nivel alterado es el de red, no viéndose afectadas el resto de capas. Por tanto, se puede compartir una gran

cantidad de código. Esto ha hecho que este mecanismo se convierta en el más usado, siendo soportado por la mayoría de los sistema operativos modernos.

En principio, este mecanismo sólo permite la comunicación entre nodos que soporten la misma versión de IP, pero no entre un nodo exclusivamente IPv4 y otro exclusivamente IPv6. En estos casos necesitaremos mecanismos más sofisticados, como los que estudiaremos posteriormente.

4.3. Túneles (RFC 4213)

4.3.1. Introducción

El mecanismo de pila doble no es capaz de cubrir todos los casos, por lo que surge la necesidad de usar mecanismos de tunelado. Los túneles funcionan como una VPN, encapsulando un paquete dentro de otro de la misma capa de la pila. Se usan para atravesar zonas intermedias que no soportan el protocolo usado por los nodo originales. Los extremos de los túneles no tienen porqué corresponderse con los nodos terminales de la transmisión, sino que puede encontrarse en un punto medio del camino.

Los extremos del túnel deben ser nodos duales, ya que conectarán zonas con versiones distintas de IP. Cuando un paquete de la versión *A* llega al inicio del túnel, el router inicial le añadirá una cabecera de la versión *B* con la dirección de destino del final del túnel, y lo reenviará. Una vez que haya atravesado la zona intermedia, se eliminará la cabecera *B* y se extraerá el paquete original, que seguirá su curso hacia su destino real. Durante su trayecto, un mismo paquete podrá atravesar varios túneles, e incluso se podrán establecer túneles anidados.

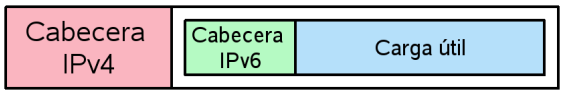

Figura 4.2: Esquema de tunelado

Los túneles se pueden clasificar de varias maneras. La más directa es atendiendo a los valores de *A* y *B*. De esta manera tendremos:

- Túneles IPv6 dentro de IPv4: son los más comunes dada la situación actual.
- Túneles IPv4 dentro de IPv6: podrán darse en un futuro cuando la versión de IP predominante sea la 6.
- Túneles UDP: los túneles normales se identifican por medio del valor 41 en el campo de protocolo. Algunos dispositivos bloquean estos paquetes, por lo que una alternativa es encapsularlos dentro de un datagrama UDP.

Otro criterio de clasificación atiende al tipo de nodos presentes en los extremos del túnel:

- Router-router: los extremos del túnel residen en dos routers intermedios.
- Host-router: el nodo de origen establece un túnel con un router intermedio, que a su vez se comunicará de manera directa con el destino.
- Router-host: caso simétrico al anterior, esta vez el túnel se establece entre el router y el nodo de destino.
- Host-host: en este caso, dos nodos duales crean un túnel que supone el camino entero de la comunicación. El protocolo encapsulado sólo verá un salto.

El último criterio de clasificación depende del proceso de configuración del túnel:

- Manual: implica establecer los extremos del túnel de manera explícita, ya sea por medio de un operador humano o a través de un servicio conocido como *broker de túneles*.
- Automática: los extremos del túnel son deducidos por la propia infraestructura de red a partir de las direcciones IPv6 de origen o destino.

4.3.2. Túnel IPv6 sobre IPv4 manual

Estos túneles añaden una cabecera IPv4 a un paquete IPv6, para que estos puedan ser enviados por una zona IPv4. Se suelen para establecer comunicaciones con ISP remotos a través de Internet. La cabecera que se añade tiene el siguiente formato:

- El valor del campo protocolo se pone a 41.
- Las direcciones de origen y destino se asignan a partir de la dirección de los extremos del túnel. Los extremos se configuran manualmente, o se derivan a partir de la dirección del próximo salto que seguirá la ruta.

Como resultado de añadir una cabecera adicional, la MTU de la ruta se reduce en 20 bytes. Esto puede provocar que un paquete tenga que fragmentarse al entrar en el túnel. En este caso, el nodo inicial del túnel deberá poner la bandera DF a cero.

4.3.3. Túneles automáticos

Suelen usarse cuando el extremo final del túnel coincide con el de la comunicación. Su funcionamiento básico es el mismo que el de los túneles manuales. La diferencia radica en la manera en que se calcula la dirección de destino. Como el destino es el mismo para las direcciones IPv4 e IPv6, se puede derivar una a partir de la otra por medio de una dirección *IPv4-compatible IPv6* con el siguiente formato:

:: a.b.c.d

Donde a.b.c.d es una dirección IPv4 a la que se añaden 96 ceros. Estas direcciones se asignan sólo para su uso en túneles automáticos. El uso de este mecanismo se ha abandonado, ya que la configuración *6to4* lo resuelve de una manera más eficiente y genérica.

4.3.4. Broker de túneles

Los *brokers de túneles* [DFGL01] son servicios que ofrecen la configuración y gestión centralizada de un gran número de túneles. Se usan cuando hay que administrar muchos túneles. Por ejemplo, un ISP puede usar un broker para gestionar el uso de la red por parte de sus clientes, o para obtener estadísticas del uso de cada túnel. También son útiles para conectar pequeñas redes IPv6 aisladas a redes IPv6 ya existentes.

Su funcionamiento se basa en el uso de una aplicación en el cliente, la existencia de uno o más servidores de túneles, y un servidor broker que reside en la zona IPv4. El cliente se conecta al servidor broker para solicitar la activación de un túnel. Una vez autorizado, proporcionará el resto de la información necesaria, como su dirección IPv4 o el tiempo de vida requerido para la conexión. El broker creará el túnel, registrará las direcciones de los extremos en el DNS, y enviará los parámetros de configuración al cliente para que este pueda iniciar la comunicación.

Las direcciones IPv6 asignadas por el broker serán siempre globales y pertenecerán al espacio que se le haya asignado. Una vez que el cliente ya no necesite el túnel, se lo podrá notificar al servidor para que lo libere, y así no esté ocupado un túnel que no se está usando.

4.3.5. Túneles 6to4

El mecanismo *6to4* [CM01] es una generalización del tunelado automático. Se usa para conectar nodos IPv6 aislados con redes remotas a través de una infraestructura IPv4. Al igual que sucedía en los túneles automáticos, se encapsula un paquete IPv6 dentro de otro IPv4, cuya dirección se calcula a partir de la dirección *6to4*. Estas direcciones tienen el siguiente formato:

2002 : aabb : ccdd : id subred : interfaz de red

Donde aabb : ccdd es la representación hexadecimal de la dirección IPv4 pública del nodo en el que se desea instalar el mecanismo *6to4*. Por tanto, este nodo debe ser accesible desde Internet v4. Se definen los siguientes elementos para *6to4*:

- Host 6to4: es un nodo que posee al menos una dirección *6to4*, y que desea comunicarse con un nodo IPv6 a través de una zona IPv4.
- Router 6to4: es un router dual que posee al menos una interfaz *6to4*, y que permite reenviar tráfico a nodos y routers *6to4* encapsulando automáticamente los paquetes IPv6 sobre paquetes IPv4.
- Router de reenvío 6to4: es un router capaz de encaminar paquetes IPv6 entre direcciones *6to4* y nodos IPv6 nativos.

6to4 es el mecanismo de tunelado más usado actualmente. Su funcionamiento es el siguiente. Suponiendo que dos nodos deseen intercambiar tráfico IPv6, pueden darse tres casos:

- Si ambos nodos están dentro de la misma red, podrán comunicarse directamente sin necesidad de usar un túnel.
- El segundo caso es que ambos nodos estén en redes distintas, separadas por una zona IPv4. En ese caso, el nodo que inicia la comunicación preguntará la dirección (o direcciones) de la máquina destino al DNS, y elegirá la dirección correspondiente al mecanismo *6to4*. Los paquetes IPv6 llegarán al router frontera, donde serán encapsulados automáticamente, deduciendo las direcciones IPv4 de origen y destino a partir de las direcciones *6to4*. Cuando el paquete llegue a la red de destino serán desencapsulados y entregados al nodo final.
- El tercer caso se da cuando una isla IPv6 quiere comunicarse con una red IPv6 nativa. En este caso, al llegar el paquete a la red de destino continuaría hasta el nodo final aprovechando la infraestructura IPv6 existente.

4.3.6. Túneles DSTM

El mecanismo DSTM (*Dual Stack Transition Mechanism*) [Ric05] se usa cuando un nodo presente en un océano IPv6 necesita comunicarse con una aplicación IPv4 remota. La solución más sencilla sería instalar una pila dual, pero esto obligaría a configurar también todos los routers intermedios. DSTM permite encapsular paquetes IPv4 dentro de IPv6. Está orientado a orientado a nodos duales individuales pertenecientes a una red IPv6.

DSTM funciona asignando una dirección IPv4 temporal al nodo que desea comunicarse, y estableciendo el túnel que transportará los paquetes. La comunicación puede iniciarse tanto por el nodo local IPv6 como por el nodo remoto IPv4. El sistema se compone de tres elementos:

- El nodo DSTM, con una pila dual instalada. La pila IPv4 se configurará dinámicamente en el momento de iniciar la comunicación.
- El router frontera que comunica las zonas de distinta versión, encargándose de la encapsulación de los paquetes.
- El servidor DSTM, que es un servidor DHCPv6 que asigna las direcciones IPv4 temporales a los nodos que lo solicitan, y les proporciona la dirección del router frontera. Mantiene una tabla con las equivalencias entre direcciones IPv4 e IPv6.

La principal ventaja de este mecanismo es que es transparente al resto de la red, y que las direcciones IPv4 sólo se reservan en el momento de usarse, liberándose en cuanto dejan de ser necesarias.

Cuando un nodo IPv6 desea iniciar una comunicación IPv4 remota, realiza una solicitud al servidor. Éste notificará al router frontera la creación del nuevo túnel, y enviará al cliente su nueva dirección IPv4 y la dirección del router frontera. Si la petición se realiza al revés, es decir, un nodo IPv4 remoto desea iniciar una comunicación con un nodo IPv6 local,

el servidor creará automáticamente el túnel si no existía de antes, notificará al nodo IPv6 su nueva dirección, la registrará en el DNS, y la devolverá al nodo IPv4 para que pueda iniciar la comunicación.

4.3.7. ISATAP

ISATAP (*Intra-Site Automatic Tunnel Addressing Protocol*) [TGT08] es un mecanismo que permite que dos nodos duales puedan transmitir paquetes IPv6 por medio de una red local IPv4. Un nodo que quiera comunicarse por medio de ISATAP creará una dirección IPv6 virtual con la forma:

prefijo :: 5efe : aabb : ccdd

Donde aabb : ccdd es la representación hexadecimal de la dirección IPv4 asignada al nodo, y prefijo es el identificador de 64 bits propio de la red. Todos los nodos de la misma red local comparten el mismo prefijo. Para descubrir el prefijo se buscará el router local por medio del protocolo *Neighbor Discovery*, que será enviado por broadcast. En caso de que el protocolo no sea capaz de descubrir ningún router, se debe configurar una lista manual de routers a los que se consultará periódicamente.

4.3.8. Teredo

Teredo es un protocolo que permite establecer obtener conectividad IPv6 a nodos situados detrás de un NAT. Los mecanismos tradicionales como *6to4* exigen que el nodo que desea establecer el túnel posea una dirección IPv4 pública. Sin embargo, debido a la escasez de direcciones IPv4, muchos nodos se conectan a Internet a través de uno o más dispositivos NAT. En este caso la única dirección pública existente se asigna al dispositivo NAT, por lo que debería ser éste el que creara el túnel, lo cual no siempre es posible. Además, la mayoría de los dispositivos NAT sólo permiten el paso de paquetes TCP/UDP, y descartarían los paquetes con el protocolo 41.

Teredo resuelve este problema encapsulando los paquetes IPv6 dentro de datagramas UDP IPv4. Se compone de cuatro elementos:

- Cliente Teredo: es un nodo que desea se encuentra detrás de un NAT y quiere establecer conectividad IPv6 por medio de Teredo. Se le asignará una dirección de la forma 2002 :: /32.
- Servidor Teredo: es un nodo conocido, con una dirección IPv4 pública, que se encarga de la configuración inicial de los túneles. No reenvía ningún tipo de tráfico.
- Router de reenvío: actúa como el nodo extremo de un túnel. Se encarga de reenviar todo el tráfico que recibe de los clientes. Cada router de reenvío sirve a un rango de direcciones; sólo reenvía tráfico entre un cliente Teredo y una dirección perteneciente a su rango.

- Nodo de reenvío: es un nodo dual que actúa como router de reenvío de sí mismo. Se conecta a la Internet IPv4 por medio de una dirección pública o privada con NAT, y a la Internet IPv6 por medio de conexión nativa o un túnel *6to4*.

Los servidores asignan direcciones IPv6 a sus clientes con el formato:

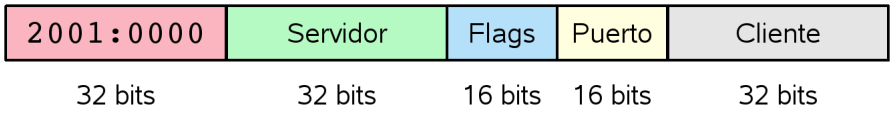

Figura 4.3: Esquema de direccionamiento Teredo

Donde "servidor" es la dirección IPv4 pública del servidor en hexadecimal, "flags" es una serie de banderas que indican el tipo de NAT tras el que se encuentra el cliente, "puerto" es el número de puerto UDP asignado por el NAT a la conexión, con sus bits invertidos, y "cliente" es la dirección IPv4 pública del dispositivo NAT, también con sus bits invertidos.

Cuando un cliente desea obtener una dirección, envía un mensaje de solicitud IPv6 al servidor Teredo, encapsulado en un paquete UDP IPv4. Al atravesar el NAT, se efectuará la sustitución de dirección y puerto. El servidor determinará el tipo de NAT (cono o restringido), y devolverá la respuesta al cliente, incluyendo la dirección resultado de la traducción, para que éste construya su dirección IPv6. Para evitar que se cierre la entrada de NAT, el cliente enviará periódicamente paquetes UDP al servidor.

Si dos clientes Teredo quieren comunicarse, seguirán el siguiente proceso. El cliente que inicia la comunicación enviará un *paquete burbuja* al servidor del destinatario, cuya dirección IPv4 puede extraer de la dirección IPv6 del cliente. El servidor reenviará el paquete al destinatario, que se pondrá en contacto con el router de reenvío para poder así establecer la comunicación.

Cuando un cliente Teredo quiere comunicarse con un nodo IPv6 nativo, enviará un paquete ICMPv6 *Echo Request* a través de su servidor. El servidor lo desencapsula y lo envía al destinatario. El destinatario responderá con un paquete *Echo Reply*, que será dirigido hacia el router de reenvío Teredo más cercano. Este router se pondrá en contacto con el cliente Teredo, e iniciará así la comunicación.

4.4. Traducción de protocolos

4.4.1. Introducción

La traducción de protocolos es el tercer y último grupo de mecanismos de transición que existen. Estos mecanismos reciben un paquete para una versión dada de un protocolo, y los reescriben adaptándolos a otra versión. La traducción puede hacerse a nivel de red, de transporte o de aplicación.

Dentro de esta familia, los cuatro mecanismos más asentados son SIIT, TRT, NAT-PT y SOCKS. No obstante, hay que mencionar que existen otros mecanismos experimentales,

como BIA [LSK+02], que no estudiaremos.

4.4.2. Stateless IP/ICMP Translation (SIIT)

El mecanismo SIIT [Nor00] permite que un nodo sólo IPv6 se pueda comunicar con un nodo sólo IPv4. Para ello, el nodo IPv6 necesita disponer de una dirección IPv4 global. A partir de ésta se le asignará una dirección *IPv4-translatable IPv6*, cuyo formato fue descrito en la sección 3.5. La traducción se limita a las cabeceras IP e ICMP, y no realiza control de estado, por lo que cada paquete se traduce individualmente.

Cuando un nodo IPv6 desea enviar un paquete a un nodo IPv4, construye un datagrama con dirección de origen la *IPv4-translatable IPv6* propia, y dirección de destino la *IPv4-mapped IPv6* del destinatario. Cuando el router SIIT recibe el paquete, deduce las direcciones IPv4 correspondientes, reescribe el paquete, y lo reenvía por la Internet IPv4 hacia el destinatario. Si la comunicación es en sentido contrario, el router SIIT recibe un paquete IPv4, a partir del cual deduce las direcciones IPv6 correspondientes, reescribe el paquete, y lo envía por la zona IPv6.

Una limitación de SIIT es que no especifica la manera de obtener las direcciones IPv4, ni cómo realizar el enrutado. Además, requiere que cada nodo IPv6 disponga de una dirección IPv4 global. Tampoco tiene en cuenta el caso de que un paquete IPv4 deba fragmentarse al entrar en la red IPv6.

4.4.3. Transport Relay Translator (TRT)

TRT [HY01] es un mecanismo que traduce no sólo la capaz de red, sino también la capa de transporte. El funcionamiento es el siguiente. Cuando un nodo inicial A desea establecer una comunicación TCP con un nodo B que pertenece a una red distinta, los paquetes son redirigidos hacia el sistema TRT, al que llamaremos X. El sistema aceptará la conexión, aunque la dirección de destino no se corresponda con la suya, y se presentará ante A como si fuera B. A continuación, establecerá una conexión con B haciéndose pasar por A. De esta manera, se crearán dos conexiones, $A \leftrightarrow X$ y $X \leftrightarrow B$.

Cuando un nodo IPv6 desee establecer una conexión con un nodo IPv4, se conectará a la dirección prefijo :: a.b.c.d, donde prefijo es el identificador de red de 64 bits del sistema TRT, y a.b.c.d es la dirección IPv4 de 32 bits del destinatario. El sistema TRT puede extraer automáticamente la dirección IPv4 y establecer la conexión.

La principal ventaja de TRT es que permite conectar un gran número de nodos IPv6 a través de una única dirección IPv4 global. Además, permite definir MTUs distintas para cada conexión, y resolver así el problema de la fragmentación. La principal desventaja es que, al centralizar todo el tráfico, consume mucho ancho de banda, lo que puede presentar problemas de escalabilidad.

4.4.4. Network Address Translation - Port Translation (NAT-PT)

NAT-PT [TS00] es un mecanismo que permite a aplicaciones y nodos IPv6 nativos comunicarse con aplicaciones y nodos IPv4 nativos. El tráfico puede ser unidireccional o bidireccional. El mecanismo resulta transparente a los nodos, no requiriendo ninguna configuración por parte de estos, ni la instalación de una pila dual.

Su funcionamiento se basa en un router NAT-PT, que conecta las dos zonas y realiza la traducción. Este router dispone de un rango de direcciones IPv4 públicas, que asignará dinámicamente a los nodos IPv6 que lo solicite, guardando el estado de cada comunicación en una tabla. El router traduce tanto los paquetes IP como los puertos TCP/UDP, de ahí su nombre.

Cuando un nodo IPv6 quiera enviar un paquete a un nodo IPv4, usará como dirección de destino prefijo :: a.b.c.d, donde prefijo es el identificador de red de 64 bits del router NAT-PT, y a.b.c.d es la dirección IPv4 de 32 bits del destinatario. Al llegar el paquete al router, se realizará la traducción. El router sustituirá la dirección de origen por la que tenga almacenada en la tabla de sesiones, y extraerá la dirección de destino IPv4 de la cabecera IPv6. Si es necesario, también se traducirán los números de puerto. Cuando el nodo IPv4 quiera enviar un paquete al nodo IPv6, se realizará el proceso inverso.

Si el nodo que inicia la comunicación está en el lado IPv6, la sesión se puede iniciar dinámicamente cuando sea necesario. En cambio, si el nodo que establece la conexión está en el lado IPv4, es necesario reservar un par dirección-puerto, que será registrado en el DNS, y configurar una traducción estática a la dirección IPv6 interna.

4.4.5. SOCKS

El mecanismo SOCKS [Kit01] se instala entre la capa de aplicación y la de transporte, reemplazando la API de *sockets* y la resolución de nombres. Su uso originalmente permitía que las aplicaciones pudieran atravesar cortafuegos de manera transparente y segura. Con el tiempo se han añadido diversas extensiones, que han desembocado en el uso de un servidor dual llamados SOCKS 64. El funcionamiento del mecanismo se divide en tres partes:

- Cada cliente debe tener instalada la biblioteca SOCKS, que se sitúa entre la aplicación y el nivel de transporte y modifica las comunicaciones con el servidor.
- El servidor SOCKS 64 actúa de puerta de enlace, encargándose de la traducción de protocolos y la resolución DNS.
- Los nodos sobre los que se ejecutan las aplicaciones redirigirán las comunicaciones a la biblioteca SOCKS instalada.

Así, cuando un nodo realice una llamada a la biblioteca de sockets, será interceptada por SOCKS, que la redirigirá al servidor correspondiente. Éste se encargará de traducir los protocolos, y de asignar una dirección privada al nodo si es necesario.

4.5. Resumen

Como conclusión, haremos un resumen final de todas las técnicas expuestas:

- Doble pila: se basa en instalar ambas pilas de protocolos en cada nodo. Permite el intercambio de ambos tipos de paquetes. Si los nodos se encuentran en redes diferentes conectadas por zonas con distinta versión de IP, habrá que usar otro mecanismo.
- Túneles:
  - Manuales: el usuario configura manualmente la dirección de los extremos del túnel.
  - Automático: se establecen automáticamente a partir de una dirección *IPv4-compatible IPv6*.
  - Broker de túneles: consta de un servidor central que gestiona la creación de túneles, al que se conectan los clientes para solicitar la configuración.
  - 6to4: permite comunicar sitios IPv6 a través de zonas IPv4, por medio de direcciones de la forma 2002 : a.b.c.d :: /48.
  - DSTM: permite la comunicación entre aplicaciones IPv4 a través de zonas IPv6.
  - ISATAP: sirve para comunicar dos nodos duales por medio de IPv6 en una misma red IPv4. Usa direcciones con el formato prefijo :: 5efe : a.b.c.d.
  - Teredo: permite establecer conexiones IPv6 atravesando una red IPv4 con NAT. Debido a su complejidad, se usa como último recurso.
- Traducción de protocolos:
  - SIIT: traduce paquetes IP e ICMP sin guardar el estado de la conexión. Usa direcciones *IPv4-translatable IPv6*.
  - TRT: traduce conexiones TCP/UDP entre una zona IPv6 y otra zona IPv4, separándolas en dos conexiones independientes.
  - NAT-PT: conecta dos zonas con versiones de IP distintas sin necesidad de proporcionar una dirección IPv4 independiente a cada nodo IPv6.
  - SOCKS: traduce paquetes a nivel de aplicación.

Parte III

Análisis

5 Informe de alcance y objetivos

Después de haber estudiado los aspectos teóricos de la migración de IPv4 a IPv6, consideramos complidos los primeros objetivos del proyecto. Queda claro que es un problema complejo. Son muchas las posibles opciones a considerar, lo que dificulta en gran medida el trabajo de los administradores de red. Llegamos a la conclusión de que interesa crear una herramienta que automatice el proceso en la medida de lo posible.

Retomando los objetivos expuestos en el primer capítulo, nos centramos en los objetivos del desarrollo de la herramienta. Expondremos brevemente cada uno de ellos, para después describirlos en mayor profundidad:

- Formalizar matemáticamente el problema, de tal manera que pueda ser tratado por un ordenador.
- Definir una estrategia de resolución que permita tratar el problema de manera eficiente.
- Diseñar y desarrollar la aplicación que implementa la metodología anterior.

Como primer objetivo, planteamos el estudio de una metodología que permita formalizar el problema. Hasta ahora, los mecanismos de transición están definidos en lenguaje coloquial, cosa que no puede ser entendida por una máquina. Si pretendemos automatizar el problema, debemos dar una definición matemática. Esta definición debe comprender todos los elementos que conforman un escenario de transición (aplicaciones, nodos terminales, y routers), así como un conjunto de reglas que permitan transformar un escenario hasta alcanzar una solución. Por tanto, deberemos estudiar los siguientes aspectos:

- Representación matemática de los escenarios.
- Simplificación de los escenarios, con el fin de obtener una representación más compacta que acelere su procesamiento posterior.
- Definición de una base de reglas, que caracterice cada uno de los mecanismos de transición. Cada regla definirá una transformación que se aplicará a un escenario concreto.

El segundo objetivo que planteamos es el estudio de una estrategia de resolución. Partiendo de una topología de red arbitraria, necesitamos poder encontrar todas las rutas posibles entre un nodo de origen y otro de destino. A partir de ahí, se deben evaluar todas las posibles soluciones para cada ruta en particular. El algoritmo debe ser los más eficiente posible. Los subobjetivos concretos de este apartado serán:

- Buscar un formato sencillo y flexible para almacenar topologías de red en disco.
- Definir un algoritmo de búsqueda que permita encontrar todas las rutas existentes entre dos nodos de una topología.
- Definir un método para, a partir de una ruta lineal, buscar todas las soluciones resultado de aplicar la base de reglas.
- Obtener un sistema para evaluar y ordenar las soluciones obtenidas en base a diferentes criterios.

El tercer y último objetivo que planteamos es desarrollar una aplicación informática que implemente la anterior metodología, de tal manera que el administrador de red pueda introducir la topología deseada y los parámetros de búsqueda, y dejar que la aplicación encuentre todas las soluciones existentes. En concreto, la aplicación debe ofrecer las siguientes opciones:

- GESTIONAR TOPOLOGÍAS: el programa permitirá cargar, guardar, y crear nuevas topologías.
- EDITAR TOPOLOGÍAS: a partir de una topología dada, permitirá añadir, editar y borrar nodos y enlaces.
- CONFIGURAR COSTES: cada mecanismo de transición tendrá un coste asociado, que podrá ser configurado por el usuario.
- BUSCAR SOLUCIONES: el programa pedirá al usuario que introduzca las opciones de búsqueda. A continuación calculará todos los resultados, y mostrará un listado con las soluciones encontradas.
- ORDENAR SOLUCIONES: el programa permitirá ordenar las soluciones por diversos criterios de coste; por ejemplo, por latencia, ancho de banda consumido, etc.
- RESALTAR SOLUCIÓN: indicará la ruta seguida por una solución en la topología, sin entrar en detalles.
- DETALLAR SOLUCIÓN: mostrará toda la información referente a los mecanismos que se han aplicado a un escenario para alcanzar la solución; por ejemplo, el tipo y los extremos de los túneles.

6 Caracterización del problema

6.1. Introducción

Hasta ahora, las decisiones sobre que mecanismos de transición se debían aplicar eran tomadas por personas. Nuestro objetivo es automatizar el problema, de manera que pueda ser resuelto por una máquina. Para ello necesitamos representar el problema de manera matemática. Al ser nuestro proyecto continuación de otro, heredaremos su metodología: la *Metodología MENINA* [Cas04], que proporciona una notación para representar escenarios de red, y un álgebra para trabajar con ellos.

La representación de un problema debe tener en cuenta todos los elementos que lo componen; esto es, aplicaciones, nodos terminales, y red de interconexión. MENINA ofrece una representación abstracta de cada uno de estos tres componentes. También formaliza los mecanismos de transición empleados, ofreciendo un conjunto de reglas que permiten refinar progresivamente un escenario de red hasta encontrar una solución. MENINA define tres tipos de componentes:

- Aplicaciones que se ejecutarán en los escenarios, usando una versión de IP concreta.
- Escenarios de red, que permiten la comunicación entre las aplicaciones. Están formados por los nodos terminales y la red de interconexión que hay entre ambos.
- Base de reglas que formaliza los mecanismos de transición, por medio de una definición algebraica.

Aunque se definen por separado, la metodología tiene en cuenta todos los elementos que forman la red en conjunto. A continuación veremos cada etapa del proceso con más detenimiento.

6.2. Modelo conceptual

6.2.1. Escenarios de red

MENINA trabaja con escenarios lineales; es decir, la red que comunica los nodos terminales es una secuencia lineal de routers intermedios. Por tanto, si partimos de un escenario con una topología arbitraria, deberemos convertirlo primero en un conjunto de escenarios lineales, buscando todos los caminos posibles entre el nodo de origen y el de destino.

Los escenarios se definen para todos los elementos que los forman: aplicaciones finales, nodos terminales, y red de interconexión. Por tanto, un escenario físico E_f quedará representado como una tupla:

$$E_f = \langle A_1, N_1, RD_f, N_2, A_2 \rangle$$

Donde A_1 y A_2 son las aplicaciones finales, N_1y N_2 son los nodos terminales sobre los que se ejecutan A_1 y A_2 respectivamente, y RD_f es la red de destribución física. MENINA asume que las redes son simétricas (es decir, el camino de ida es el mismo que el de vuelta), y sólo se preocupa por el nivel IP, ya que el resto de niveles deberían funcionar de manera transparente a IP.

6.2.2. Aplicaciones

Este proyecto sólo trata aplicaciones *unicast* donde las comunicaciones son 1 a 1. De cada aplicación sólo interesa saber la versión de IP que soporta:

- Aplicación exclusivamente IPv4, A_4.
- Aplicación exclusivamente IPv6, A_6.
- Aplicación dual, A_d.— Puede usar ambos protocolos en función del nodo sobre el que se ejecute.

El conjunto de todos los tipos de aplicaciones posibles se representan en MENINA como $\mathcal{A}$:

$$\mathcal{A} = \{A_4, A_6, A_d\}$$

6.2.3. Nodos terminales

Al tratar los nodos sólo importa conocer su capacidad para usar diferentes versiones de IP; es decir, las pilas de protocolos que tienen instaladas. La metodología no tiene en cuenta el sistema operativo que esté instalado en cada nodo, pero al evaluar un escenario real habría que considerarlo, porque no todos los sistemas operativos ofrecen todos los mecanismos posibles.

Existen ciertos mecanismos de transición que se instalan en los nodos terminales, dotándolos de capacidades especiales. En estos casos, MENINA usa la notación

$$TransiciónNodo\,(N_i) = N_j$$

donde N_i y N_j son nodos terminales. Estos mecanismos son independientes de la red, y sólo afectan al propio nodo. Para los casos que hemos estudiado, el único mecanismo existente es *IPv4-mapped IPv6*, que permite que una aplicación IPv6 pueda comunicarse con otra aplicación IPv4.

Teniendo en cuenta todo lo anterior, se definen cuatro tipos de nodos:

- Nodo exclusivamente IPv4, N_4.
- Nodo exclusivamente IPv6, N_6.
- Nodo dual, N_d.— Tiene instaladas ambas pilas de protocolos y puede ejecutar tanto aplicaciones IPv4 como IPv6.
- Nodo dual con direcciones IPv4-mapped IPv6, $N_{d,MAP}$.— Es el resultado de aplicar el mecanismo $Mapped\,(N_d) = N_{d,MAP}$ a un nodo dual.

El conjunto de todos los tipos de nodos posibles se representan en MENINA como $\mathcal{N}$:

$$\mathcal{N} = \{N_4, N_6, N_d, N_{d,MAP}\}$$

6.2.4. Routers intermedios

Los routers forman el camino que conecta los nodos terminales del escenario de red. La primera manera de clasificarlos sería atendiendo a la versión del protocolo IP que soportan.

Sin embargo, muchos routers añaden capacidades adicionales. La primera es la traducción de direcciones IPv4 o NAT, que sólo puede existir en los routers que tengan instalada la pila IPv4 (incluyendo los duales). La traducción de direcciones IPv6 no existe, por lo que no debemos preocuparnos por ella.

Existen otros mecanismos que se instalan en los routers, y se representan como

$$TransiciónRouter\,(R_i) = R_j$$

donde R_i y R_j son routers intermedios. Estos mecanismos se usan para conectar dos redes adyacentes que usan versiones de IP distintas, por medio de traducción de paquetes. La traducción sólo se usa como último recurso, ya que implica modificar los paquetes y acarrea cierta pérdida de información. Para que un router soporte traducción debe ser forzosamente dual, ya que debe reconocer ambas versiones de IP. Los mecanismos de traducción que hemos estudiado son SIIT, TRT, NAT-PT, y SOCKS.

Teniendo en cuenta todo lo anterior, se definen los siguientes tipos de routers:

- Router exclusivamente IPv4, R_4.
- Router exclusivamente IPv6, R_6.
- Router dual, R_d.—Puede encaminar tanto tráfico IPv4 como IPv6.
- Router IPv4 con traducción de direcciones, R_4^t.— Puede encaminar tráfico IPv4, y traducir direcciones IPv4 (NAT).
- Router dual con traducción de direcciones, R_d^t.— Puede encaminar ambos protocolos, y traducir direcciones IPv4.

- Router dual con traducción de protocolos, $R_{d,TP}$.− Es el resultado de aplicar un mecanismo de traducción a un router dual: $\textit{Traducción}\,(R_d) = R_{d,TP}$.

- Router dual con traducción de protocolos y de direcciones, $R^t_{d,TP}$.− Es el resultado de aplicar un mecanismo de traducción a un router dual con NAT: $\textit{Traducción}\left(R^t_d\right) = R^t_{d,TP}$.

El conjunto de todos los tipos de routers posibles se representan en MENINA como $\mathcal{R}$:

$$\mathcal{R} = \left\{R_4, R_6, R_d, R^t_4, R^t_d, R_{d,TP}, R^t_{d,TP}\right\}$$

6.2.5. Red de distribución

La red de distribución de un escenario es la secuencia de cero o más routers que forman el camino entre dos nodos terminales. MENINA define para la conexión de dos routers el símbolo $\leftrightarrow$, de tal modo que la conexión de dos routers R_i y R_j sería:

$$R_i \leftrightarrow R_j \qquad \text{donde } R_i, R_j \in \mathcal{R}$$

Para indicar que un router se conecta linealmente con uno o más routers, sin indicar la cantidad exacta, se utiliza la notación:

$$R_i \left(\leftrightarrow R_j\right) \star \qquad \text{donde } R_i, R_j \in \mathcal{R}$$

De esta manera, podemos representar la red de distribución de cualquier escenario:

$$RD_f \in \left\{\emptyset, R_i \left(\leftrightarrow R_j\right) \star\right\} \qquad \text{donde } R_i, R_j \in \mathcal{R}$$

La conexión entre nodos terminales y routers también se representa con el símbolo $\leftrightarrow$. Por tanto, tenemos $N_i \leftrightarrow N_j$ para conectar dos nodos directamente sin pasar por routers intermedios, y $N_i \leftrightarrow R_j$ y $R_i \leftrightarrow N_j$ para comunicar un nodo terminal y un router.

6.2.6. Representación de los escenarios

Vista ya la representación de cada elemento de un escenario por separado, podemos definir un escenario completo.

La ejecución de una aplicación sobre un nodo terminal se representa con el símbolo $\diamond$, de tal manera que si una aplicación A_i se ejecuta sobre un nodo N_j, la representación sería:

$$A_i \diamond N_j \qquad \text{donde } A_i \in \mathcal{A},\ N_j \in \mathcal{N}$$

Finalmente, un escenario de transición E_f que comunica dos aplicaciones A_1 y A_2

quedaría representado así:

$$E_f = \langle A_1, N_1, RD_f, N_2, A_2 \rangle$$
$$\Downarrow$$
$$A_1 \diamond N_1 \, (\leftrightarrow R_i) \star \leftrightarrow N_2 \diamond A_2$$
donde $A_1, A_2 \in \mathcal{A},\ N_1, N_2 \in \mathcal{N},\ R_i \in \mathcal{R}$

6.3. Formalización

Una vez descrito el escenario, la segunda fase de la metodología es la formalización. El resultado es un nuevo escenario equivalente al primero, pero más compacto. Para ello, se reconocen patrones dentro de la red, y se sustituyen por otros más simples. Al hacer esto se pierde la correspondencia con la red física, quedándonos con una representación abstracta. La formalización sólo trata la red de distribución, no teniendo en cuenta ni las aplicaciones ni los nodos terminales.

El proceso de formalización se divide a su vez en dos etapas: creación de zonas y definición de los operadores de conexión. Estudiaremos ambas a continuación.

6.3.1. Creación de zonas

La primera etapa busca agrupar secuencias de nodos del mismo tipo en una única zona común que será tratada como una sola entidad. Dentro de una zona se cumplen dos propiedades: cualquier nodo puede ser alcanzado desde cualquier otro nodo, y los paquetes IP no se modifican. Las zonas se caracterizan por la versión de IP que usan, por lo que podemos distinguir cuatro tipos distintos:

- Zona IPv4 (Z_4).— Es una subred que sólo soporta tráfico IPv4, y que no realiza traducción de direcciones (NAT). La transformación resultante es la siguiente:

$$R_4 \, (\leftrightarrow R_4) \star = Z_4$$
$$\Downarrow$$
$$A1 \diamond N1 \leftrightarrow R_4 \, (\leftrightarrow R_4) \star \leftrightarrow N2 \diamond A2 = A1 \diamond N1 \leftrightarrow Z_4 \leftrightarrow N2 \diamond A2$$

- Zona IPv6 (Z_6).— Es una subred que sólo soporta tráfico IPv4. La transformación quedaría así:

$$R_6 \, (\leftrightarrow R_6) \star = Z_6$$
$$\Downarrow$$
$$A1 \diamond N1 \leftrightarrow R_6 \, (\leftrightarrow R_6) \star \leftrightarrow N2 \diamond A2 = A1 \diamond N1 \leftrightarrow Z_6 \leftrightarrow N2 \diamond A2$$

- Zona dual (Z_d).— Es una subred que soporta tanto tráfico IPv4 como IPv6, pero sin traducción de direcciones. La transformación del escenario sería así.

$$R_d\,(\leftrightarrow R_d)\star = Z_d$$
$$\Downarrow$$
$$A1 \diamond N1 \leftrightarrow R_d\,(\leftrightarrow R_d)\star \leftrightarrow N2 \diamond A2 = A1 \diamond N1 \leftrightarrow Z_d \leftrightarrow N2 \diamond A2$$

- Zona vacía.− Es un tipo especial de zona que surge cuando los nodos terminales se conectan directamente el uno al otro, sin que haya un router intermedio. En este caso, al no haber restricciones de ningún tipo, se puede tomar la zona más genérica posible; esto es, una zona dual. Por tanto, la transformación sería la siguiente:

$$A1 \diamond N1 \leftrightarrow N2 \diamond A2 = A1 \diamond N1 \leftrightarrow Z_d \leftrightarrow N2 \diamond A2$$

6.3.2. Definición de operadores de conexión

La segunda etapa busca unificar zonas adyacentes por medio de operadores de conexión, que suelen realizar funciones de traducción. Estos operadores se definen en función del tráfico que soportan. En MENINA existen cinco tipos de operadores, cada uno con un símbolo correspondiente:

- Operador de conexión dual ($\otimes_d$).− Permite conectar dos zonas IPv4 ó IPv6 a través de una zona dual:

$$Z_i \leftrightarrow R_d\,(\leftrightarrow R_d)\star \leftrightarrow Z_j = Z_i \leftrightarrow Z_d \leftrightarrow Z_j = Z_i \otimes_d Z_j$$
$$\text{donde } Z_i, Z_j \in \{Z_4, Z_6\}$$

- Operador de conexión IPv4 con traducción de direcciones IPv4 ($\odot_4$).− Permite conectar dos zonas IPv4, realizando una traducción de direcciones (NAT):

$$Z_i \leftrightarrow R_4^t \leftrightarrow Z_j = Z_i \odot_4 Z_j$$

- Operador de conexión dual con traducción de direcciones IPv4 ($\odot_d$).− Combina los dos operadores anteriores, permitiendo el paso de tráfico IPv4 e IPv6 y traduciendo direcciones IPv4:

$$Z_i \leftrightarrow (Z_d \leftrightarrow)\star R_d^t\,(\leftrightarrow Z_d)\star \leftrightarrow Z_j = Z_i \odot_d Z_j$$
$$\text{donde } Z_i, Z_j \in \{Z_4, Z_6\}$$

- Operador de conexión dual con traducción de protocolos IPv4 e IPv6 ($\boxtimes_d$).− Permite el paso de ambos protocolos, y además puede traducir de IPv4 a IPv6 y viceversa:

$$Z_i \leftrightarrow (Z_d \leftrightarrow) \star R_{d,TP} (\leftrightarrow Z_d) \star \leftrightarrow Z_j = Z_i \boxtimes_d Z_j$$
$$\text{donde } Z_i, Z_j \in \{Z_4, Z_6\}, Z_i \neq Z_j$$

- Operador de conexión dual con traducción de direcciones IPv4 y traducción de protocolos IPv4 e IPv6 ($\boxdot_d$).— Es el operador más genérico, ya que soporta las operaciones de todos los anteriores:

$$Z_i \leftrightarrow (Z_d \leftrightarrow) \star R^t_{d,TP} (\leftrightarrow Z_d) \star \leftrightarrow Z_j = Z_i \boxdot_d Z_j$$
$$\text{donde } Z_i, Z_j \in \{Z_4, Z_6\}, Z_i \neq Z_j$$

6.4. Canonización

Si bien la fase de formalización simplifica la representación de un escenario, éste se puede simplificar aún mas mediante la canonización. La forma canónica de un escenario se define como la forma más simple posible, resultado de aplicar las siguientes reglas de reescritura:

- Conexión de zonas IPv4 utilizando $\otimes_d$.— El resultado es una nueva zona IPv4 que engloba a las anteriores, y que opera como una zona básica:

$$Z_4 (\otimes_d Z_4) \star = Z_4$$

- Conexión de zonas IPv4 utilizando $\odot_d$.— Puesto que sólo se usa el direccionamiento IPv4, se puede simplificar a un operador IPv4 con traducción de direcciones:

$$Z_4 (\odot_d Z_4) = Z_4 \odot_4 Z_4$$

- Conexión de zonas IPv4 utilizando $\boxdot_d$.— Como sólo se usa el direccionamiento IPv4, se puede descartar la capacidad para traducir protocolos, obteniendo el mismo resultado que en el caso anterior:

$$Z_4 (\boxdot_d Z_4) = Z_4 \odot_4 Z_4$$

- Conexión de zonas IPv6 utilizando $\otimes_d$.— Mismo caso que para IPv4. El resultado es una nueva zona IPv6 que engloba a las anteriores, y que opera como una zona básica:

$$Z_6 (\otimes_d Z_6) \star = Z_6$$

- Conexión de zonas IPv6 utilizando $\boxtimes_d$.− Como ambas zonas son IPv6, se puede descartar la capacidad para traducir protocolos, obteniendo una zona IPv6 básica:

$$Z_6 \, (\boxtimes_d Z_6) \star = Z_6 \, (\otimes_d Z_6) \star = Z_6$$

- Conexión de zonas IPv6 utilizando $\odot_d$ ó $\boxdot_d$.− Al igual que en el caso, al ser ambas zonas del mismo tipo no es necesaria la capacidad para traducir protocolos. Por otro lado, la capacidad para traducir direcciones sólo se aplica a IPv4, por lo que también se puede descartar:

$$Z_6 \boxdot_d Z_6 = Z_6 \odot_d Z_6 = Z_6$$

- Conexión de zonas duales utilizando $\otimes_d$.− Como ocurría con las Z_4 y las Z_6, el resultado es una nueva zona que contiene a las anteriores, y que opera como una zona básica.

$$Z_d \, (\otimes_d Z_d) \star = Z_d$$

- Conexión de una zona dual en una posición intermedia utilizando $\otimes_d$.− Como las zonas duales permiten tanto el tráfico IPv4 como IPv6, si encontramos una secuencia de zonas duales no terminales entre dos zonas de cualquier tipo, podemos descartarlas:

$$Z_i \otimes_d Z_d \, (\otimes_d Z_d) \star \otimes_d Z_j = Z_i \otimes_d Z_j$$
$$\text{donde } Z_i, Z_j \in \{Z_4, Z_6, Z_d\}$$

6.5. Túneles

Una vez que tenemos un escenario formalizado y canonizado, podemos empezar a aplicar reglas de transformación que nos permitan alcanzar la conectividad entre los nodos terminales. Estas reglas vienen en forma de tres tipos de túneles que se establecen entre diferentes partes del escenario: túneles entre zonas, entre zonas y nodos terminales, y entre los dos nodos terminales. Al aplicar estas reglas buscamos convertir una red heterogénea en una red homogénea, simplificando al máximo el escenario. A continuación veremos cada regla con más detalle.

6.5.1. Túneles entre zonas intermedias

Estos mecanismos se aplican a las zonas intermedias que forman la red de interconexión del escenario. El objetivo es que dos zonas con la misma versión de IP se puedan comunicar a través de una zona con una versión distinta, estableciendo para ello un túnel.

Existen tres clases de tunelado, que veremos a continuación: túneles manuales, broker de túneles, y túneles automáticos *6to4*.

6.5.1.1. Túneles manuales

Se distinguen tres tipos de túneles, en función del tráfico que va a pasar por la zona intermedia: IPv4 dentro de IPv6, IPv6 dentro de IPv4, e IPv6 dentro de IPv4 con UDP:

- IPv4 dentro de IPv6.— Dos zonas IPv4 quieren comunicarse a través de una zona IPv6. Si había traducción de direcciones, el túnel debe respetarla:

$$T_{v6(v4)}\left(A1 \diamond N1 \leftrightarrow \underline{Z_4 op_1 Z_6 op_2 Z_4} \leftrightarrow N2 \diamond A2\right) = A1 \diamond N1 \leftrightarrow Z_4 \leftrightarrow N2 \diamond A2$$
donde $op_1, op_2 \in \{\otimes_d, \boxtimes_d\}$

$$T_{v6(v4)}\left(A1 \diamond N1 \leftrightarrow \underline{Z_4 op_1 Z_6 op_2 Z_4} \leftrightarrow N2 \diamond A2\right) = A1 \diamond N1 \leftrightarrow Z_4 \odot_4 Z_4 \leftrightarrow N2 \diamond A2$$
donde $op_1 \in \{\otimes_d, \boxtimes_d\}, op_2 \in \{\odot_d, \boxdot_d\}$

$$T_{v6(v4)}\left(A1 \diamond N1 \leftrightarrow \underline{Z_4 op_1 Z_6 op_2 Z_4} \leftrightarrow N2 \diamond A2\right) = A1 \diamond N1 \leftrightarrow Z_4 \odot_4 Z_4 \odot_4 Z_4 \leftrightarrow N2 \diamond A2$$
donde $op_1, op_2 \in \{\odot_d, \boxdot_d\}$

- IPv6 dentro de IPv4.— Dos zonas IPv6 quieren comunicarse a través de una zona IPv4. Este caso es más sencillo ya que no hay traducción de direcciones:

$$T_{v4(v6)}\left(A1 \diamond N1 \leftrightarrow \underline{Z_6 op_1 Z_4 op_2 Z_6} \leftrightarrow N2 \diamond A2\right) = A1 \diamond N1 \leftrightarrow Z_6 \leftrightarrow N2 \diamond A2$$
donde $op_1, op_2 \in \{\otimes_d, \boxtimes_d, \odot_d, \boxdot_d\}$

- IPv6 dentro de UDP IPv4.— Este mecanismo se usa cuando dos zonas IPv6 quieren comunicarse a través de una o más zonas IPv4 que emplean traducción de direcciones. Para ello se deben usar paquetes UDP, y así poder atravesar el NAT:

$$T_{udp(v4(v6))}\left(A1 \diamond N1 \leftrightarrow \underline{Z_6 op_1 Z_4 \left(\odot_4 Z_4\right) \star op_2 Z_6} \leftrightarrow N2 \diamond A2\right) = A1 \diamond N1 \leftrightarrow Z_6 \leftrightarrow N2 \diamond A2$$
donde $op_1, op_2 \in \{\otimes_d, \boxtimes_d, \odot_d, \boxdot_d\}$

6.5.1.2. Broker de túneles

El broker de túneles es mur parecido al caso anterior, con las mismas situaciones. La diferencia es que ahora los túneles se configuran de manera automática. Como antes, distinguimos tres tipos de túneles:

- IPv4 dentro de IPv6:

$$Broker_{v6(v4)}\left(A1 \diamond N1 \leftrightarrow \underline{Z_4 op_1 Z_6 op_2 Z_4} \leftrightarrow N2 \diamond A2\right) = A1 \diamond N1 \leftrightarrow Z_4 \leftrightarrow N2 \diamond A2$$
donde $op_1, op_2 \in \{\otimes_d, \boxtimes_d\}$

$$Broker_{v6(v4)}\left(A1 \diamond N1 \leftrightarrow \underline{Z_4 op_1 Z_6 op_2 Z_4} \leftrightarrow N2 \diamond A2\right) = A1 \diamond N1 \leftrightarrow Z_4 \odot_4 Z_4 \leftrightarrow N2 \diamond A2$$
donde $op_1 \in \{\otimes_d, \boxtimes_d\}, op_2 \in \{\odot_d, \boxdot_d\}$

$$Broker_{v6(v4)}\left(A1 \diamond N1 \leftrightarrow \underline{Z_4 op_1 Z_6 op_2 Z_4} \leftrightarrow N2 \diamond A2\right) = A1 \diamond N1 \leftrightarrow Z_4 \odot_4 Z_4 \odot_4 Z_4 \leftrightarrow N2 \diamond A2$$
donde $op_1, op_2 \in \{\odot_d, \boxdot_d\}$

- IPv6 dentro de IPv4:

$$Broker_{v4(v6)}\left(A1 \diamond N1 \leftrightarrow \underline{Z_6 op_1 Z_4 op_2 Z_6} \leftrightarrow N2 \diamond A2\right) = A1 \diamond N1 \leftrightarrow Z_6 \leftrightarrow N2 \diamond A2$$
donde $op_1, op_2 \in \{\otimes_d, \boxtimes_d, \odot_d, \boxdot_d\}$

- IPv6 dentro de UDP IPv4:

$$Broker_{udp(v4(v6))}\left(A1 \diamond N1 \leftrightarrow \underline{Z_6 op_1 Z_4 \left(\odot_4 Z_4\right) \star op_2 Z_6} \leftrightarrow N2 \diamond A2\right) = A1 \diamond N1 \leftrightarrow Z_6 \leftrightarrow N2 \diamond A2$$
donde $op_1, op_2 \in \{\otimes_d, \boxtimes_d, \odot_d, \boxdot_d\}$

6.5.1.3. Túneles 6to4

Este mecanismo establece un túnel automático que permite al tráfico IPv6 atravesar una zona IPv4, siempre que ésta no emplee traducción de direcciones. Es por tanto un mecanismo de transición entre zonas:

$$6to4\left(A1 \diamond N1 \leftrightarrow \underline{Z_6 op_1 Z_4 op_2 Z_6} \leftrightarrow N2 \diamond A2\right) = A1 \diamond N1 \leftrightarrow Z_6 \leftrightarrow N2 \diamond A2$$
donde $op_1, op_2 \in \{\otimes_d, \boxtimes_d, \odot_d, \boxdot_d\}$

6.5.2. Túneles entre una zona y un nodo terminal

Estos mecanismos permiten a un nodo terminal con una determinada versión de IP comunicarse con una zona intermedia de la red de interconexión que usa la misma versión de IP, a través de zonas con una versión de IP distinta. Según el estudio que se hizo, se distinguen cinco tipos de tunelado: túneles manuales, broker de túneles, DSTM, Teredo, e ISATAP.

6.5.2.1. Túneles manuales

Estos túneles son muy similares a los que se establecían entre zonas intermedias. Al igual que sucedía con aquellos, podemos distinguir tres casos en función de la zona inter-

media que encaminará el tráfico:

- IPv4 dentro de IPv6.— Un nodo dual quiere intercambiar tráfico IPv4 a través de una zona IPv6. Si hay traducción de direcciones, el túnel debe mantenerla:

$$T_{v6(v4)}\left(A1 \diamond \underline{N_d \leftrightarrow Z_6 op_1 Z_4} \leftrightarrow N2 \diamond A2\right) = A1 \diamond N_d \leftrightarrow Z_4 \leftrightarrow N2 \diamond A2$$
donde $op_1 \in \{\otimes_d, \boxtimes_d\}$

$$T_{v6(v4)}\left(A1 \diamond \underline{N_d \leftrightarrow Z_6 op_1 Z_4} \leftrightarrow N2 \diamond A2\right) = A1 \diamond N_d \leftrightarrow Z_4 \odot_4 Z_4 \leftrightarrow N2 \diamond A2$$
donde $op_1 \in \{\odot_d, \boxdot_d\}$

- IPv6 dentro de IPv4.— Un nodo dual quiere intercambiar tráfico IPv6 a través de una zona IPv4, que no usa traducción de direcciones:

$$T_{v4(v6)}\left(A1 \diamond \underline{N_d \leftrightarrow Z_4 op_1 Z_6} \leftrightarrow N2 \diamond A2\right) = A1 \diamond N_d \leftrightarrow Z_6 \leftrightarrow N2 \diamond A2$$
donde $op_1 \in \{\otimes_d, \boxtimes_d, \odot_d, \boxdot_d\}$

- IPv6 dentro de UDP IPv4.— Un nodo dual quiere intercambiar tráfico IPv6 a través de una zona IPv4, que usa traducción de direcciones. Para poder atravesar el NAT, se encapsulan los datagramas IPv6 dentro de paquetes UDP:

$$T_{udp(v4(v6))}\left(A1 \diamond \underline{N_d \leftrightarrow Z_4\,(\odot_4 Z_4) \star op_1 Z_6} \leftrightarrow N2 \diamond A2\right) = A1 \diamond N_d \leftrightarrow Z_6 \leftrightarrow N2 \diamond A2$$
donde $op_1 \in \{\otimes_d, \boxtimes_d, \odot_d, \boxdot_d\}$

6.5.2.2. Broker de túneles

Al igual que ocurría con los túneles entre zonas intermedias, el broker de túneles proporciona configuración automática para los túneles que se acaban de ver. Se distinguen tres casos:

- IPv4 dentro de IPv6:

$$Broker_{v6(v4)}\left(A1 \diamond \underline{N_d \leftrightarrow Z_6 op_1 Z_4} \leftrightarrow N2 \diamond A2\right) = A1 \diamond N_d \leftrightarrow Z_4 \leftrightarrow N2 \diamond A2$$
donde $op_1 \in \{\otimes_d, \boxtimes_d\}$

$$Broker_{v6(v4)}\left(A1 \diamond \underline{N_d \leftrightarrow Z_6 op_1 Z_4} \leftrightarrow N2 \diamond A2\right) = A1 \diamond N_d \leftrightarrow Z_4 \odot_4 Z_4 \leftrightarrow N2 \diamond A2$$
donde $op_1 \in \{\odot_d, \boxdot_d\}$

- IPv6 dentro de IPv4:

$$Broker_{v4(v6)}\left(A1 \diamond \underline{N_d \leftrightarrow Z_4 op_1 Z_6} \leftrightarrow N2 \diamond A2\right) = A1 \diamond N_d \leftrightarrow Z_6 \leftrightarrow N2 \diamond A2$$
donde $op_1 \in \{\otimes_d, \boxtimes_d, \odot_d, \boxdot_d\}$

- IPv6 dentro de UDP IPv4:

$$Broker_{udp(v4(v6))}\left(A1 \diamond \underline{N_d \leftrightarrow Z_4 (\odot_4 Z_4) \star op_1 Z_6} \leftrightarrow N2 \diamond A2\right) = A1 \diamond N_d \leftrightarrow Z_6 \leftrightarrow N2 \diamond A2$$
donde $op_1 \in \{\otimes_d, \boxtimes_d, \odot_d, \boxdot_d\}$

6.5.2.3. DSTM

DSTM permite a un nodo dual comunicarse con una zona IPv4, atravesando una zona IPv6 por medio de un túnel. Si el operador intermedio realiza traducción de direcciones, el escenario resultante lo conservará:

$$DSTM\left(A1 \diamond \underline{N_d \leftrightarrow Z_6 op_1 Z_4} \leftrightarrow N2 \diamond A2\right) = A1 \diamond N1 \leftrightarrow Z_4 \leftrightarrow N2 \diamond A2$$
donde $op_1 \in \{\otimes_d, \boxtimes_d\}$

$$DSTM\left(A1 \diamond \underline{N_d \leftrightarrow Z_6 op_1 Z_4} \leftrightarrow N2 \diamond A2\right) = A1 \diamond N1 \leftrightarrow Z_4 op_1 Z_4 \leftrightarrow N2 \diamond A2$$
donde $op_1 \in \{\odot_d, \boxdot_d\}$

6.5.2.4. Teredo

Teredo es un mecanismo que permite que un nodo dual se pueda comunicar con una zona IPv6, atravesando una zona IPv4, cuando en ésta zona se realiza traducción de direcciones. Para ello se encapsulan los paquetes IPv6 dentro de UDP IPv4:

$$Teredo\left(A1 \diamond \underline{N_d \leftrightarrow Z_4 (\odot_4 Z_4) \star op_1 Z_6} \leftrightarrow N2 \diamond A2\right) = A1 \diamond N1 \leftrightarrow Z_6 \leftrightarrow N2 \diamond A2$$
donde $op_1 \in \{\otimes_d, \boxtimes_d, \odot_d, \boxdot_d\}$

6.5.2.5. ISATAP

ISATAP configura túneles IPv6 dentro de IPv4, normalmente dentro de una misma red en la que no existe traducción de direcciones:

$$ISATAP\left(A1 \diamond \underline{N_d \leftrightarrow Z_4 op_1 Z_6} \leftrightarrow N2 \diamond A2\right) = A1 \diamond N_d \leftrightarrow Z_6 \leftrightarrow N2 \diamond A2$$
donde $op_1 \in \{\otimes_d, \boxtimes_d\}$

6.5.3. Túneles entre nodos terminales

Estos mecanismos se aplican sobre escenarios cuya red de interconexión tiene una única zona. Permiten que dos nodos terminales con la misma versión de IP se comuniquen

a través de una zona con una versión distinta. Encontramos dos tipos de tunelado: manual e ISATAP.

6.5.3.1. Túneles manuales

Como en los casos anteriores, se dan tres tipos de túneles en función del tráfico que pasa a través de la zona intermedia:

- IPv4 dentro de IPv6.– Dos nodos duales quieren intercambiar tráfico IPv4 a través de una zona IPv6:

$$T_{v6(v4)}\left(A1 \diamond \underline{N_d \leftrightarrow Z_6 \leftrightarrow N_d} \diamond A2\right) = A1 \diamond N_d \leftrightarrow Z_4 \leftrightarrow N_d \diamond A2$$

- IPv6 dentro de IPv4.– Dos nodos duales quieren intercambiar tráfico IPv6 a través de una zona IPv4:

$$T_{v4(v6)}\left(A1 \diamond \underline{N_d \leftrightarrow Z_4 \leftrightarrow N_d} \diamond A2\right) = A1 \diamond N_d \leftrightarrow Z_6 \leftrightarrow N_d \diamond A2$$

- IPv6 dentro de UDP IPv4.– Dos nodos duales quieren comunicarse a través de una zona IPv4 con NAT, para lo que encapsulan datagramas IPv6 dentro de paquetes UDP:

$$T_{udp(v4(v6))}\left(A1 \diamond \underline{N_d \leftrightarrow Z_4\left(\odot_4 Z_4\right)\star \leftrightarrow N_d} \diamond A2\right) = A1 \diamond N_d \leftrightarrow Z_6 \leftrightarrow N_d \diamond A2$$

6.5.3.2. ISATAP

ISATAP configura túneles IPv6 dentro de IPv4, normalmente dentro de una misma red en la que no existe traducción de direcciones. En este caso se aplica entre los dos nodos terminales:

$$ISATAP\left(A1 \diamond \underline{N_d \leftrightarrow Z_4 \leftrightarrow N_d} \diamond A2\right) = A1 \diamond N_d \leftrightarrow Z_6 \leftrightarrow N_d \diamond A2$$

6.6. Modificaciones sobre redes finales

La última fase del proceso es la modificación sobre redes finales. Una vez que se han terminado de aplicar todas las transformaciones posibles, y ya no quedan más opciones, pueden darse tres situaciones:

- Que el escenario no ofrezca ningún tipo de conectividad.
- Que el escenario permita la conexión directa de los dos nodos terminales.
- Que el escenario no permita la conexión directa, pero ésta pueda alcanzarse modificando alguno de los nodos terminales.

Si nos encontramos en el último caso, se aplicará alguna de las reglas que se verán a continuación. Estas reglas se clasifican a su vez en cuatro categorías, según el tipo de red que nos encontremos: redes básicas e irreducibles, redes de dos zonas, de tres zonas, y fragmentos intermedios de la red.

6.6.1. Redes básicas

Las redes básicas son aquellas que no se pueden reducir. Esto incluye los tres tipos de redes de una sola zona (Z_4, Z_6 y Z_d), así como las redes IPv4 con traducción de direcciones ($Z_4 \odot_4 Z_4$). Por otro lado, podemos encontrar tres combinaciones de aplicaciones: que ambas sean IPv4, que ambas sean IPv6, o que usen versiones distintas de IP. Esto nos da un total de doce combinaciones, que expondremos por separado.

6.6.1.1. Zonas IPv4 con aplicaciones IPv4

Expondremos en primer lugar los escenarios en los que se puede establecer la conectividad:

- Escenario solución 1:

$$A_4 \diamond N_4 \leftrightarrow Z_4 \leftrightarrow N_4 \diamond A_4$$

- Escenario solución 2:

$$A_4 \diamond N_4 \leftrightarrow Z_4 \leftrightarrow N_d \diamond A_4$$

- Escenario solución 3:

$$A_4 \diamond N_d \leftrightarrow Z_4 \leftrightarrow N_d \diamond A_4$$

A continuación expondremos los escenarios en los que no se puede establecer la conectividad:

- Escenario no solución 1. En caso de tener una aplicación IPv4 sobre un nodo IPv6, éste se transformará en dual por medio de la instalación del mecanismo Dual Stack:

$$\begin{gathered} A_4 \diamond N_6 \leftrightarrow Z_4 \leftrightarrow N_i \diamond A_4 \\ \text{donde } N_i \in \{N_4, N_6, N_d\} \\ \Downarrow \\ N_6 \rightarrow N_d: \quad A_4 \diamond N_d \leftrightarrow Z_4 \leftrightarrow N_i \diamond A_4 \end{gathered}$$

6.6.1.2. Zonas IPv4 con aplicaciones IPv6

Expondremos en primer lugar los escenarios en los que se puede establecer la conectividad:

- Escenario solución 1:

$$A_6 \diamond N_{d,MAP} \leftrightarrow Z_4 \leftrightarrow N_{d,MAP} \diamond A_6$$

A continuación expondremos los escenarios en los que no se puede establecer la conectividad:

- Escenario no solución 1. En caso de tener una aplicación A_6 sobre un nodo N_4 ó N_6 conectado a una zona Z_4, deberá transformarse el nodo en dual mediante la instalación del mecanismo Dual Stack:

$$\begin{gathered} A_6 \diamond N_j \leftrightarrow Z_4 \leftrightarrow N_i \diamond A_6 \\ \text{donde } N_i \in \{N_4, N_6, N_d\},\ N_j \in \{N_4, N_6\} \\ \Downarrow \\ N_j \rightarrow N_d: \quad A_6 \diamond N_d \leftrightarrow Z_4 \leftrightarrow N_i \diamond A_4 \end{gathered}$$

- Escenario no solución 2. En caso de que ambas aplicaciones se ejecuten sobre nodos duales, hay dos opciones. La primera sería instalar el mecanismo Mapped sobre ambos nodos, mientras que la segunda sería crear un túnel IPv6 en IPv4:

$$\begin{gathered} A_6 \diamond N_d \leftrightarrow Z_4 \leftrightarrow N_d \diamond A_6 \\ \Downarrow \\ N_d \rightarrow N_{d,MAP}: \quad A_6 \diamond N_{d,MAP} \leftrightarrow Z_4 \leftrightarrow N_{d,MAP} \diamond A_6 \\ T_{v4(v6)}\left(A_6 \diamond \underline{N_d \leftrightarrow Z_4 \leftrightarrow N_d} \diamond A_6\right) = A_6 \diamond N_d \leftrightarrow Z_6 \leftrightarrow N_d \diamond A_6 \end{gathered}$$

6.6.1.3. Zonas IPv4 con aplicaciones heterogéneas

Expondremosen primer lugar los escenarios en los que se puede establecer la conectividad:

- Escenario solución 1:

$$A_4 \diamond N_4 \leftrightarrow Z_4 \leftrightarrow N_{d,MAP} \diamond A_6$$

- Escenario solución 2:

$$A_4 \diamond N_d \leftrightarrow Z_4 \leftrightarrow N_{d,MAP} \diamond A_6$$

A continuación expondremos los escenarios en los que no se puede establecer la conectividad:

- Escenario no solución 1. Si una aplicación se ejecuta sobre un nodo con distinta versión de IP, éste deberá transformarse en dual mediante la instalación del mecanismo Dual Stack:

$$\begin{gathered} A_4 \diamond N_6 \leftrightarrow Z_4 \leftrightarrow N_i \diamond A_6 \\ \text{donde } N_i \in \{N_4, N_6, N_d\} \\ \Downarrow \\ N_6 \rightarrow N_d: \quad A_4 \diamond N_d \leftrightarrow Z_4 \leftrightarrow N_i \diamond A_6 \end{gathered}$$

- Escenario no solución 2. Si la versión de IP de un nodo terminal es distinta de la zona a la que se conecta o de la aplicación que ejecuta, deberá transformarse en dual mediante la instalación del mecanismo Dual Stack:

$$\begin{gathered} A_4 \diamond N_i \leftrightarrow Z_4 \leftrightarrow N_j \diamond A_6 \\ \text{donde } N_i \in \{N_4, N_6, N_d\}, \ N_j \in \{N_4, N_6\} \\ \Downarrow \\ N_j \rightarrow N_d: \quad A_4 \diamond N_i \leftrightarrow Z_4 \leftrightarrow N_d \diamond A_6 \end{gathered}$$

- Escenario no solución 3. Si la aplicación A_6 se ejecuta sobre un nodo dual, podrá instalarse el mecanismo Mapped:

$$\begin{gathered} A_4 \diamond N_i \leftrightarrow Z_4 \leftrightarrow N_d \diamond A_6 \\ \text{donde } N_i \in \{N_4, N_d\} \\ \Downarrow \\ N_d \rightarrow N_{d,MAP}: \quad A_4 \diamond N_i \leftrightarrow Z_4 \leftrightarrow N_{d,MAP} \diamond A_6 \end{gathered}$$

6.6.1.4. Zonas IPv6 con aplicaciones IPv4

El primer caso para zonas IPv6 que podemos encontrar es que las aplicaciones sean IPv4. En esta situación las únicas soluciones posibles son transformar la aplicación en dual (lo que queda fuera del alcance de nuestro proyecto), o transformar los nodos terminales en duales para posteriormente crear un túnel, y así llegar a una de las situaciones expuestas anteriormente. Veamos en primer lugar los escenarios en los que se puede establecer conectividad:

- Escenario solución 1:

$$A_4 \diamond N_d \leftrightarrow Z_6 \leftrightarrow N_d \diamond N_4$$

A continuación expondremos los escenarios en los que no se puede establecer la conectividad:

- Escenario no solución 1. Si la versión de IP del nodo difiere de la aplicación o de la red, deberá instalarse el mecanismo Dual Stack:

$$A_4 \diamond N_j \leftrightarrow Z_6 \leftrightarrow N_i \diamond A_4$$
$$\text{donde } N_i \in \{N_4, N_6, N_d\}\,,\ N_j \in \{N_4, N_6\}$$
$$\Downarrow$$
$$N_j \rightarrow N_d: \quad A_4 \diamond N_d \leftrightarrow Z_6 \leftrightarrow N_i \diamond A_4$$

- Escenario no solución 2. Si ambos nodos son ya de por sí duales, se puede establecer un túnel IPv4 en IPv6:

$$A_4 \diamond N_d \leftrightarrow Z_6 \leftrightarrow N_d \diamond A_4$$
$$\Downarrow$$
$$T_{v6(v4)}\left(A_4 \diamond \underline{N_d \leftrightarrow Z_6 \leftrightarrow N_d} \diamond A_4\right) = A_4 \diamond N_d \leftrightarrow Z_4 \leftrightarrow N_d \diamond A_4$$

6.6.1.5. Zonas IPv6 con aplicaciones IPv6

Otro caso que podemos encontrar es que las aplicaciones sean IPv6. Si los nodos terminales son IPv6 o duales, no habrá problemas:

- Escenario solución 1:
$$A_6 \diamond N_6 \leftrightarrow Z_6 \leftrightarrow N_6 \diamond A_6$$
- Escenario solución 2:
$$A_6 \diamond N_6 \leftrightarrow Z_6 \leftrightarrow N_d \diamond A_6$$
- Escenario solución 3:
$$A_6 \diamond N_d \leftrightarrow Z_6 \leftrightarrow N_d \diamond A_6$$

Veamos los escenarios en los que no se puede establecer la conectividad:

- Escenario no solución 1. Si alguno de los nodos es IPv4, habrá que recurrir a la instalación del mecanismo Dual Stack

$$A_6 \diamond N_4 \leftrightarrow Z_6 \leftrightarrow N_i \diamond A_6$$
$$\text{donde } N_i \in \{N_4, N_6, N_d\}$$
$$\Downarrow$$
$$N_4 \rightarrow N_d: \quad A_6 \diamond N_d \leftrightarrow Z_6 \leftrightarrow N_i \diamond A_6$$

6.6.1.6. Zonas IPv6 con aplicaciones heterogéneas

El último caso que podemos encontrar es que las dos aplicaciones usen versiones de IP distintas. Este caso es problemático, porque la única manera de alcanzar una solución es transformar los dos nodos terminales en duales, así como la aplicación IPv4 en dual. Como ya se dijo antes, esto queda fuera del ámbito de nuestro proyecto. De todas maneras,

estudiaremos las transformaciones necesarias para acercar el escenario lo más posible a la solución final. Veamos en primer lugar los escenarios sobre los que se podría establecer la conectividad:

- Escenario solución 1:

$$A_6 \diamond N_d \leftrightarrow Z_6 \leftrightarrow N_d \diamond A_4$$

  Nota: transformar A_4 en A_d

Veamos ahora los escenarios en los que no se puede establecer la conectividad:

- Escenario no solución 1. En caso de que la versión de IP de la aplicación y del nodo terminal sean diferentes, se instalará el mecanismo Dual Stack:

$$\begin{gathered} A_6 \diamond N_4 \leftrightarrow Z_6 \leftrightarrow N_i \diamond A_4 \\ \text{donde } N_i \in \{N_4, N_6, N_d\} \\ \Downarrow \\ N_4 \rightarrow N_d: \quad A_6 \diamond N_d \leftrightarrow Z_6 \leftrightarrow N_i \diamond A_4 \end{gathered}$$

- Escenario no solución 2. Si la versión de IP de un nodo terminal es diferente de la zona a la que se conecta, se instalará el mecanismo Dual Stack:

$$\begin{gathered} A_6 \diamond N_i \leftrightarrow Z_6 \leftrightarrow N_j \diamond A_4 \\ \text{donde } N_i \in \{N_4, N_6, N_d\}, \quad N_j \in \{N_4, N_6\} \\ \Downarrow \\ N_j \rightarrow N_d: \quad A_6 \diamond N_i \leftrightarrow Z_6 \leftrightarrow N_d \diamond A_4 \end{gathered}$$

- Escenario no solución 3. Si ambos nodos terminales son duales, se puede instalar un túnel para crear una zona virtual IPv4, encontrándonos así uno de los casos expuestos anteriormente:

$$\begin{gathered} A_6 \diamond N_d \leftrightarrow Z_6 \leftrightarrow N_d \diamond A_4 \\ \Downarrow \\ T_{v6(v4)}\left(A_6 \diamond \underline{N_d \leftrightarrow Z_6 \leftrightarrow N_d} \diamond A_4\right) = A_6 \diamond N_d \leftrightarrow Z_4 \leftrightarrow N_d \diamond A_4 \end{gathered}$$

6.6.1.7. Zonas duales con aplicaciones IPv4

Si ambas aplicaciones son IPv4, la zona dual se comportará como si fuera también IPv4. Por tanto, se transformará la Z_d en Z_4, y se considerará el escenario igual que en el apartado 6.6.1.1:

$$A_4 \diamond N_i \leftrightarrow Z_d \leftrightarrow N_j \diamond A_4 = A_4 \diamond N_i \leftrightarrow Z_4 \leftrightarrow N_j \diamond A_4$$

6.6.1.8. Zonas duales con aplicaciones IPv6

Si ambas aplicaciones son IPv6, la zona dual se comportará como si fuera también IPv6. Por tanto, se transformará la Z_d en Z_4, y se considerará el escenario igual que en el apartado 6.6.1.5:

$$A_6 \diamond N_i \leftrightarrow Z_d \leftrightarrow N_j \diamond A_6 = A_6 \diamond N_i \leftrightarrow Z_6 \leftrightarrow N_j \diamond A_6$$

6.6.1.9. Zonas duales con aplicaciones heterogéneas

Si las aplicaciones son heterogéneas, no podemos decidir a priori que tipo de encaminamiento usar. En ese caso, la única opción es tratar las dos alternativas posibles:

$$A_4 \diamond N_i \leftrightarrow Z_d \leftrightarrow N_j \diamond A_6 = \begin{cases} A_4 \diamond N_i \leftrightarrow Z_4 \leftrightarrow N_j \diamond A_6 \\ A_4 \diamond N_i \leftrightarrow Z_6 \leftrightarrow N_j \diamond A_6 \end{cases}$$

6.6.1.10. Zonas IPv4 con traducción de direcciones

En este caso, si al menos una de las dos aplicaciones es IPv4, se tratará como si hubiera una única zona IPv4. Sólo cuando ambas aplicaciones sean IPv6 habrá que tratar el escenario de manera especial. La situación ideal es que ambos nodos terminales soporten NAT, en cuyo caso se podrá establecer conectividad directa:

- Escenario solución 1:

$$A_6 \diamond N_{d,MAP} \leftrightarrow Z_4 \odot_4 Z_4 \leftrightarrow N_{d,MAP} \diamond A_6$$

A continuación expondremos los escenarios en los que no se puede establecer la conectividad:

- Escenario no solución 1. Si tenemos una aplicación A_6 nodo N_4 ó N_6 conectado a una zona Z_4, deberemos transformar el nodo en dual mediante la instalación del mecanismo Dual Stack:

$$\begin{array}{c} A_6 \diamond N_j \leftrightarrow Z_4 \odot_4 Z_4 \leftrightarrow N_i \diamond A_6 \\ \text{donde } N_i \in \{N_4, N_6, N_d\},\ N_j \in \{N_4, N_6\} \\ \Downarrow \\ N_j \rightarrow N_d: \quad A_6 \diamond N_d \leftrightarrow Z_4 \odot_4 Z_4 \leftrightarrow N_i \diamond A_6 \end{array}$$

- Escenario no solución 2. Si ambos nodos son duales, tenemos dos opciones. La primera sería instalar el mecanismo Mapped en ambos, mientras que la segunda sería usar un túnel Teredo:

$$A_6 \diamond N_d \leftrightarrow Z_4 \odot_4 Z_4 \leftrightarrow N_d \diamond A_6$$
$$\Downarrow$$
$$N_d \rightarrow N_{d,MAP}: \quad A_6 \diamond N_{d,MAP} \leftrightarrow Z_4 \leftrightarrow N_{d,MAP} \diamond A_6$$
$$Teredo\left(A_6 \diamond \underline{N_d \leftrightarrow Z_4 \odot_4 Z_4 \leftrightarrow N_d} \diamond A_6\right) = A_6 \diamond N_d \leftrightarrow Z_6 \leftrightarrow N_d \diamond A_6$$

6.6.2. Redes de dos zonas

Una vez estudiadas las redes básicas, el siguiente paso es extender nuestro estudio a las redes de dos zonas. Así, podrán enlazarse redes Z_4y Z_6, dándose los siguientes casos:

- Composición de zonas IPv4
- Composición de zonas IPv6
- Composición de zonas IPv4 e IPv6

Dos zonas IPv4 pueden conectarse por medio de cualquiera de los operadores de conexión ya vistos, pues todos soportan tráfico IPv4. Se dan los siguientes casos, que ya han sido estudiados anteriormente:

- $Z_4 \otimes_d Z_4 = Z_4$
- $Z_4 \odot_4 Z_4 = Z_4 \odot_4 Z_4$
- $Z_4 \odot_d Z_4 = Z_4 \odot_4 Z_4$
- $Z_4 \boxtimes_d Z_4 = Z_4$
- $Z_4 \boxdot_d Z_4 = Z_4 \odot_4 Z_4$

De la misma manera, dos zonas IPv6 pueden conectarse por medio de cualquier operador, excepto $\odot_4$, que sólo soporta tráfico IPv4. Tenemos los siguientes casos, que ya han sido estudiados:

- $Z_6 \otimes_d Z_6 = Z_6$
- $Z_6 \odot_d Z_6 = Z_6$
- $Z_6 \boxtimes_d Z_6 = Z_6$
- $Z_6 \boxdot_d Z_6 = Z_6$

El último caso es más complicado, ya que hay que entrar a considerar las aplicaciones usadas. Como en las redes básicas, puede ocurrir que ambas aplicaciones sean IPv4, que ambas sean IPv6, o que sean heterogéneas. Veamos cada situación por separado.

6.6.2.1. Composición de zonas con aplicaciones IPv4

En este caso no se puede establecer conectividad de manera directa. Expondremos por tanto sólo los escenarios sin conectividad:

- Escenario no solución 1. Si la versión de IP del nodo terminal y de la aplicación son diferentes, se instalará el mecanismo Dual Stack:

$$\begin{gathered} A_4 \diamond N_i \leftrightarrow Z_4 op Z_6 \leftrightarrow N_j \diamond A_4 \\ \text{donde } N_i \in \{N_4, N_6, N_d\}\,,\ N_j \in \{N_4, N_6\}\,,\ op \in \{\otimes_d, \odot_d, \boxtimes_d, \boxdot_d\} \\ \Downarrow \\ N_j \to N_d: \quad A_4 \diamond N_i \leftrightarrow Z_4 op Z_6 \leftrightarrow N_d \diamond A_4 \end{gathered}$$

- Escenario no solución 2. Si la versión de IP del nodo terminal y de la aplicación son diferentes, se instalará el mecanismo Dual Stack:

$$\begin{gathered} A_4 \diamond N_6 \leftrightarrow Z_4 op Z_6 \leftrightarrow N_d \diamond A_4 \\ \text{donde } op \in \{\otimes_d, \odot_d, \boxtimes_d, \boxdot_d\} \\ \Downarrow \\ N_6 \to N_d: \quad A_4 \diamond N_d \leftrightarrow Z_4 op Z_6 \leftrightarrow N_d \diamond A_4 \end{gathered}$$

- Escenario no solución 3. Si el nodo terminal que conecta con la zona IPv6 es dual, se puede establecer un túnel IPv4 en IPv6, para convertir la red en una sola zona homogénea:

$$\begin{gathered} A_4 \diamond N_i \leftrightarrow Z_4 op Z_6 \leftrightarrow N_d \diamond A_4 \\ \text{donde } N_i \in \{N_4, N_d\}\,,\ op \in \{\otimes_d, \boxtimes_d\} \\ \Downarrow \\ T_{v6(v4)}\left(A_4 \diamond N_i \leftrightarrow \underline{Z_4 op Z_6 \leftrightarrow N_d} \diamond A_4\right) = A_4 \diamond N_i \leftrightarrow Z_4 \leftrightarrow N_d \diamond A_4 \end{gathered}$$

- Escenario no solución 4. Si el nodo terminal que conecta con la zona IPv6 es dual, se puede establecer un túnel IPv4 en IPv6. Se respetará la traducción de direcciones si la hay:

$$\begin{gathered} A_4 \diamond N_i \leftrightarrow Z_4 op Z_6 \leftrightarrow N_d \diamond A_4 \\ \text{donde } N_i \in \{N_4, N_d\}\,,\ op \in \{\odot_d, \boxdot_d\} \\ \Downarrow \\ T_{v6(v4)}\left(A_4 \diamond N_i \leftrightarrow \underline{Z_4 op Z_6 \leftrightarrow N_d} \diamond A_4\right) = A_4 \diamond N_d \leftrightarrow Z_4 \odot_4 Z_4 \leftrightarrow N_d \diamond A_4 \end{gathered}$$

6.6.2.2. Composición de zonas con aplicaciones IPv6

En este caso sí nos encontramos con escenarios en los que se puede establecer la conectividad:

- Escenario solución 1:

$$A_6 \diamond N_{d,MAP} \leftrightarrow Z_4 op Z_6 \leftrightarrow N_6 \diamond A_6$$
$$\text{donde } op \in \{\boxtimes_d, \boxdot_d\}$$

- Escenario solución 2:

$$A_6 \diamond N_{d,MAP} \leftrightarrow Z_4 op Z_6 \leftrightarrow N_d \diamond A_6$$
$$\text{donde } op \in \{\boxtimes_d, \boxdot_d\}$$

A continuación trataremos los escenarios sin conectividad:

- Escenario no solución 1. Si la versión de IP del nodo terminal y de la aplicación son diferentes, se instalará el mecanismo Dual Stack:

$$A_6 \diamond N_j \leftrightarrow Z_4 op Z_6 \leftrightarrow N_i \diamond A_6$$
$$\text{donde } N_i \in \{N_4, N_6, N_d\},\ N_j \in \{N_4, N_6\},\ op \in \{\otimes_d, \odot_d, \boxtimes_d, \boxdot_d\}$$
$$\Downarrow$$
$$N_j \rightarrow N_d: \quad A_6 \diamond N_d \leftrightarrow Z_4 op Z_6 \leftrightarrow N_i \diamond A_6$$

- Escenario no solución 2. Si la versión de IP del nodo terminal y de la aplicación son diferentes, se instalará el mecanismo Dual Stack:

$$A_6 \diamond N_i \leftrightarrow Z_4 op Z_6 \leftrightarrow N_4 \diamond A_6$$
$$\text{donde } N_i \in \{N_4, N_6, N_d\},\ op \in \{\otimes_d, \odot_d, \boxtimes_d, \boxdot_d\}$$
$$\Downarrow$$
$$N_4 \rightarrow N_d: \quad A_6 \diamond N_i \leftrightarrow Z_4 op Z_6 \leftrightarrow N_d \diamond A_6$$

- Escenario no solución 3. Cuando el nodo terminal que conecta con la zona Z_4, surgen dos opciones. La primera es crear un túnel IPv6 en IPv4, mientras que la segunda opción, sólo si el operador es $\{\otimes_d, \odot_d\}$, es instalar el mecanismo Mapped:

$$A_6 \diamond N_d \leftrightarrow Z_4 op Z_6 \leftrightarrow N_i \diamond A_6$$
$$\text{donde } N_i \in \{N_6, N_d\},\ op \in \{\otimes_d, \odot_d, \boxtimes_d, \boxdot_d\}$$
$$\Downarrow$$
$$T_{v4(v6)} = \left(A_6 \diamond \underline{N_d \leftrightarrow Z_4 op Z_6} \leftrightarrow N_i \diamond A_6\right) = A_6 \diamond N_d \leftrightarrow Z_6 \leftrightarrow N_i \diamond A_6$$
$$N_d \rightarrow N_{d,MAP},\ \{\otimes_d, \odot_d\} \rightarrow \{\boxtimes_d, \boxdot_d\}: \quad A_6 \diamond N_{d,MAP} \leftrightarrow Z_4 trad(op) Z_6 \leftrightarrow N_i \diamond A_6$$

6.6.2.3. Composición de zonas con aplicaciones heterogéneas

Cuando las aplicaciones finales son diferentes, pueden darse dos casos bien diferenciados:

- A_4 se ejecuta sobre la zona Z_4, y A_6 sobre la zona Z_6.
- A_4 se ejecuta sobre la zona Z_6, y A_6 sobre la zona Z_4.

Veremos en primer lugar los escenarios con conectividad para el primer caso:

- Escenario solución 1:
$$A_4 \diamond N_4 \leftrightarrow Z_4 op Z_6 \leftrightarrow N_6 \diamond A_6$$
- Escenario solución 2:
$$A_4 \diamond N_4 \leftrightarrow Z_4 op Z_6 \leftrightarrow N_d \diamond A_6$$
- Escenario solución 3:
$$A_4 \diamond N_d \leftrightarrow Z_4 op Z_6 \leftrightarrow N_6 \diamond A_6$$
- Escenario solución 4:
$$A_4 \diamond N_d \leftrightarrow Z_4 op Z_6 \leftrightarrow N_d \diamond A_6$$

Expondremos a continuación los escenarios sin conectividad para el primer caso:

- Escenario no solución 1. Si un nodo terminal no es compatible con la aplicación, se instalará el mecanismo Dual Stack:
$$\begin{array}{c} A_4 \diamond N_6 \leftrightarrow Z_4 op Z_6 \leftrightarrow N_i \diamond A_6 \\ \text{donde } N_i \in \{N_4, N_6, N_d\}\,,\ op \in \{\otimes_d, \odot_d, \boxtimes_d, \boxdot_d\} \\ \Downarrow \\ N_6 \rightarrow N_d: \quad A_4 \diamond N_d \leftrightarrow Z_4 op Z_6 \leftrightarrow N_i \diamond A_6 \end{array}$$
- Escenario no solución 2. Si un nodo terminal no es compatible con la zona a la que se conecta, se instalará el mecanismo Dual Stack:
$$\begin{array}{c} A_4 \diamond N_i \leftrightarrow Z_4 op Z_6 \leftrightarrow N_4 \diamond A_6 \\ \text{donde } N_i \in \{N_4, N_d\}\,,\ op \in \{\otimes_d, \odot_d, \boxtimes_d, \boxdot_d\} \\ \Downarrow \\ N_4 \rightarrow N_d: \quad A_4 \diamond N_i \leftrightarrow Z_4 op Z_6 \leftrightarrow N_d \diamond A_6 \end{array}$$
- Escenario no solución 3. Si el operador de conexión no soporta traducción de protocolos, se transformará el nodo de la zona IPv6 en dual para posteriormente poder crear un túnel:
$$\begin{array}{c} A_4 \diamond N_i \leftrightarrow Z_4 op Z_6 \leftrightarrow N_6 \diamond A_6 \\ \text{donde } N_i \in \{N_4, N_6, N_d\}\,,\ op \in \{\otimes_d, \odot_d\} \\ \Downarrow \\ N_6 \rightarrow N_d: \quad A_4 \diamond N_i \leftrightarrow Z_4 op Z_6 \leftrightarrow N_d \diamond A_6 \end{array}$$

- Escenario no solución 4. Si ambos nodos terminales son compatibles con sus aplicaciones correspondientes, y el operador de conexión no soporta traducción de protocolos, se sustituirá por uno que sí lo haga:

$$\begin{gathered} A_4 \diamond N_j \leftrightarrow Z_4 op Z_6 \leftrightarrow N_i \diamond A_6 \\ \text{donde } N_i \in \{N_6, N_d\}\,,\ N_j \in \{N_4, N_d\}\,,\ op \in \{\otimes_d, \odot_d\} \\ \Downarrow \\ A_4 \diamond N_j \leftrightarrow Z_4 trad(op) Z_6 \leftrightarrow N_i \diamond A_6 \end{gathered}$$

- Escenario no solución 5. Si el nodo de la zona IPv6 es dual, y el operador no soporta traducción de direcciones, se creará un túnel IPv4 en IPv6:

$$\begin{gathered} A_4 \diamond N_i \leftrightarrow Z_4 op Z_6 \leftrightarrow N_d \diamond A_6 \\ \text{donde } N_i \in \{N_6, N_d\}\,,\ op \in \{\otimes_d, \boxtimes_d\} \\ \Downarrow \\ T_{v4(v6)}\left(A_4 \diamond N_i \leftrightarrow \underline{Z_4 op Z_6 \leftrightarrow N_d} \diamond A_6\right) = A_4 \diamond N_i \leftrightarrow Z_4 \leftrightarrow N_d \diamond A_6 \end{gathered}$$

- Escenario no solución 6. Si el nodo de la zona IPv6 es dual, y el operador sí soporta traducción de direcciones, se creará un túnel IPv4 en IPv6 que respete la traducción:

$$\begin{gathered} A_4 \diamond N_i \leftrightarrow Z_4 op Z_6 \leftrightarrow N_d \diamond A_6 \\ \text{donde } N_i \in \{N_6, N_d\}\,,\ op \in \{\odot_d, \boxdot_d\} \\ \Downarrow \\ T_{v4(v6)}\left(A_4 \diamond N_i \leftrightarrow \underline{Z_4 op Z_6 \leftrightarrow N_d} \diamond A_6\right) = A_4 \diamond N_i \leftrightarrow Z_4 \odot_4 Z_4 \leftrightarrow N_d \diamond A_6 \end{gathered}$$

En el segundo caso no hay escenarios con conectividad directa, por lo que sólo veremos aquellos sin conectividad:

- Escenario no solución 1. Si un nodo terminal no es compatible con la aplicación o con la zona a la que se conecta, se instalará el mecanismo Dual Stack:

$$\begin{gathered} A_4 \diamond N_i \leftrightarrow Z_6 op Z_4 \leftrightarrow N_j \diamond A_6 \\ \text{donde } N_i \in \{N_4, N_6\}\,,\ N_j \in \{N_4, N_6, N_d\}\,,\ op \in \{\otimes_d, \odot_d, \boxtimes_d, \boxdot_d\} \\ \Downarrow \\ N_i \rightarrow N_d: \quad A_4 \diamond N_d \leftrightarrow Z_6 op Z_4 \leftrightarrow N_j \diamond A_6 \end{gathered}$$

- Escenario no solución 2. Si un nodo terminal no es compatible con la aplicación o con la zona a la que se conecta, se instalará el mecanismo Dual Stack:

$$A_4 \diamond N_d \leftrightarrow Z_6 op Z_4 \leftrightarrow N_i \diamond A_6$$
$$\text{donde } N_i \in \{N_4, N_6\}\,,\ op \in \{\otimes_d, \odot_d, \boxtimes_d, \boxdot_d\}$$
$$\Downarrow$$
$$N_i \rightarrow N_d: \quad A_4 \diamond N_d \leftrightarrow Z_6 op Z_4 \leftrightarrow N_d \diamond A_6$$

- Escenario no solución 3. Si ambos nodos terminales son duales, y el operador no soporta traducción de direcciones, se creará un túnel IPv4 en IPv6:

$$A_4 \diamond N_d \leftrightarrow Z_6 op Z_4 \leftrightarrow N_d \diamond A_6$$
$$\text{donde } op \in \{\otimes_d, \boxtimes_d\}$$
$$\Downarrow$$
$$T_{v6(v4)}\left(A_4 \diamond \underline{N_d \leftrightarrow Z_6 op Z_4} \leftrightarrow N_d \diamond A_6\right) = A_4 \diamond N_d \leftrightarrow Z_4 \leftrightarrow N_d \diamond A_6$$

- Escenario no solución 4: Si ambos nodos terminales son duales, y el operador sí soporta traducción de direcciones, se creará un túnel IPv4 en IPv6 que respete la traducción:

$$A_4 \diamond N_d \leftrightarrow Z_6 op Z_4 \leftrightarrow N_d \diamond A_6$$
$$\text{donde } op \in \{\odot_d, \boxdot_d\}$$
$$\Downarrow$$
$$T_{v6(v4)}\left(A_4 \diamond \underline{N_d \leftrightarrow Z_6 op Z_4} \leftrightarrow N_d \diamond A_6\right) = A_4 \diamond N_d \leftrightarrow Z_4 \odot_4 Z_4 \leftrightarrow N_d \diamond A_6$$

6.6.3. Redes de tres zonas

El último tipo de escenario que consideraremos por separado es aquel que tiene tres zonas. Podemos encontrar dos situaciones: dos zonas IPv4 con una zona IPv6 entre medias, y viceversa. En ninguno de estos casos podremos establecer una conexión directa, por lo que todos los escenarios serán sin conectividad. Trataremos cada caso por separado.

6.6.3.1. Composición de zonas IPv4 - IPv6 - IPv4

Veamos en primer lugar el comportamiento de estas redes para aplicaciones IPv4:

- Escenario no solución 1. Se creará un túnel IPv4 en IPv6 que atraviese la zona Z_6 intermedia:

$$A_4 \diamond N_i \leftrightarrow Z_4 op_1 Z_6 op_2 Z_4 \leftrightarrow N_j \diamond A_4$$
$$\text{donde } N_i, N_j \in \{N_4, N_6, N_d\}\,,\ op_1, op_2 \in \{\otimes_d, \odot_d, \boxtimes_d, \boxdot_d\}$$
$$\Downarrow$$
$$T_{v6(v4)}\left(A_4 \diamond N_i \leftrightarrow \underline{Z_4 op_1 Z_6 op_2 Z_4} \leftrightarrow N_j \diamond A_4\right) = A_4 \diamond N_i \leftrightarrow Z_4 \leftrightarrow N_j \diamond A_4$$

- Escenario no solución 2. Si uno de los nodos terminales es dual, se creará un túnel IPv6 en IPv4 del nodo a la zona Z_6 intermedia:

$$\begin{gathered} A_4 \diamond N_i \leftrightarrow Z_4 op_1 Z_6 op_2 Z_4 \leftrightarrow N_d \diamond A_4 \\ \text{donde } N_i \in \{N_4, N_6, N_d\}\,,\ op_1, op_2 \in \{\otimes_d, \odot_d, \boxtimes_d, \boxdot_d\} \\ \Downarrow \\ T_{v4(v6)}\left(A_4 \diamond N_i \leftrightarrow Z_4 op_1 \underline{Z_6 op_2 Z_4 \leftrightarrow N_d} \diamond A_4\right) = A_4 \diamond N_i \leftrightarrow Z_4 op_1 Z_6 \leftrightarrow N_d \diamond A_4 \end{gathered}$$

A continuación expondremos el problema para aplicaciones IPv6:

- Escenario no solución 1. Se creará un túnel IPv4 en IPv6 que atraviese la zona Z_6 intermedia:

$$\begin{gathered} A_6 \diamond N_i \leftrightarrow Z_4 op_1 Z_6 op_2 Z_4 \leftrightarrow N_j \diamond A_6 \\ \text{donde } N_i, N_j \in \{N_4, N_6, N_d\}\,,\ op_1, op_2 \in \{\otimes_d, \odot_d, \boxtimes_d, \boxdot_d\} \\ \Downarrow \\ T_{v6(v4)}\left(A_6 \diamond N_i \leftrightarrow \underline{Z_4 op_1 Z_6 op_2 Z_4} \leftrightarrow N_j \diamond A_6\right) = A_6 \diamond N_i \leftrightarrow Z_4 \leftrightarrow N_j \diamond A_6 \end{gathered}$$

- Escenario no solución 2. Si uno de los nodos terminales es dual, se creará un túnel IPv6 en IPv4 del nodo a la zona Z_6 intermedia:

$$\begin{gathered} A_6 \diamond N_i \leftrightarrow Z_4 op_1 Z_6 op_2 Z_4 \leftrightarrow N_d \diamond A_6 \\ \text{donde } N_i \in \{N_4, N_6, N_d\}\,,\ op_1, op_2 \in \{\otimes_d, \odot_d, \boxtimes_d, \boxdot_d\} \\ \Downarrow \\ T_{v4(v6)}\left(A_6 \diamond N_i \leftrightarrow Z_4 op_1 \underline{Z_6 op_2 Z_4 \leftrightarrow N_d} \diamond A_6\right) = A_6 \diamond N_i \leftrightarrow Z_4 op_1 Z_6 \leftrightarrow N_d \diamond A_6 \end{gathered}$$

Por último, veamos los casos para aplicaciones heterogéneas:

- Escenario no solución 1. Se creará un túnel IPv4 en IPv6 que atraviese la zona Z_6 intermedia:

$$\begin{gathered} A_4 \diamond N_i \leftrightarrow Z_4 op_1 Z_6 op_2 Z_4 \leftrightarrow N_j \diamond A_6 \\ \text{donde } N_i, N_j \in \{N_4, N_6, N_d\}\,,\ op_1, op_2 \in \{\otimes_d, \odot_d, \boxtimes_d, \boxdot_d\} \\ \Downarrow \\ T_{v6(v4)}\left(A_4 \diamond N_i \leftrightarrow \underline{Z_4 op_1 Z_6 op_2 Z_4} \leftrightarrow N_j \diamond A_6\right) = A_4 \diamond N_i \leftrightarrow Z_4 \leftrightarrow N_j \diamond A_6 \end{gathered}$$

- Escenario no solución 2. Si la aplicación A_6 se ejecuta sobre un nodo dual, se creará un túnel IPv6 en IPv4 del nodo a la zona Z_6 intermedia:

$$A_4 \diamond N_i \leftrightarrow Z_4 op_1 Z_6 op_2 Z_4 \leftrightarrow N_d \diamond A_6$$
$$\text{donde } N_i \in \{N_4, N_6, N_d\},\ op_1, op_2 \in \{\otimes_d, \odot_d, \boxtimes_d, \boxdot_d\}$$
$$\Downarrow$$
$$T_{v4(v6)}\left(A_4 \diamond N_i \leftrightarrow Z_4 op_1 \underline{Z_6 op_2 Z_4 \leftrightarrow N_d} \diamond A_6\right) = A_4 \diamond N_i \leftrightarrow Z_4 op_1 Z_6 \leftrightarrow N_d \diamond A_6$$

- Escenario no solución 3. Si la aplicación A_4 se ejecuta sobre un nodo dual, se creará un túnel IPv6 en IPv4 del nodo a la zona Z_6 intermedia:

$$A_4 \diamond N_d \leftrightarrow Z_4 op_1 Z_6 op_2 Z_4 \leftrightarrow N_i \diamond A_6$$
$$\text{donde } N_i \in \{N_4, N_6, N_d\},\ op_1, op_2 \in \{\otimes_d, \odot_d, \boxtimes_d, \boxdot_d\}$$
$$\Downarrow$$
$$T_{v4(v6)}\left(A_4 \diamond \underline{N_d \leftrightarrow Z_4 op_1 Z_6} op_2 Z_4 \leftrightarrow N_i \diamond A_6\right) = A_4 \diamond N_d \leftrightarrow Z_6 op_2 Z_4 \leftrightarrow N_i \diamond A_6$$

6.6.3.2. Composición de zonas IPv6 - IPv4 - IPv6

Veamos en primer lugar el comportamiento de estas redes para aplicaciones IPv4:

- Escenario no solución 1. Se creará un túnel IPv6 dentro de IPv4 que atraviese la zona Z_4 intermedia:

$$A_4 \diamond N_i \leftrightarrow Z_6 op_1 Z_4 op_2 Z_6 \leftrightarrow N_j \diamond A_4$$
$$\text{donde } N_i, N_j \in \{N_4, N_6, N_d\},\ op_1, op_2 \in \{\otimes_d, \odot_d, \boxtimes_d, \boxdot_d\}$$
$$\Downarrow$$
$$T_{v4(v6)}\left(A_4 \diamond N_i \leftrightarrow \underline{Z_6 op_1 Z_4 op_2 Z_6} \leftrightarrow N_j \diamond A_4\right) = A_4 \diamond N_i \leftrightarrow Z_6 \leftrightarrow N_j \diamond A_4$$

- Escenario no solución 2. Si un nodo terminal es dual, y la zona Z_6 conecta con la zona Z_4 por medio de un operador sin traducción de direcciones, se creará un túnel IPv4 dentro de IPv6:

$$A_4 \diamond N_i \leftrightarrow Z_6 op_1 Z_4 op_2 Z_6 \leftrightarrow N_d \diamond A_4$$
$$\text{donde } N_i \in \{N_4, N_6, N_d\},\ op_1 \in \{\otimes_d, \odot_d, \boxtimes_d, \boxdot_d\},\ op2\ \{\otimes_d, \boxtimes_d\}$$
$$\Downarrow$$
$$T_{v6(v4)}\left(A_4 \diamond N_i \leftrightarrow Z_6 op_1 \underline{Z_4 op_2 Z_6 \leftrightarrow N_d} \diamond A_4\right) = A_4 \diamond N_i \leftrightarrow Z_6 op_1 Z_4 \leftrightarrow N_d \diamond A_4$$

- Escenario no solución 3. Si un nodo terminal es dual, y la zona Z_6 conecta con la zona Z_4 por medio de un operador con traducción de direcciones, se creará un túnel IPv4 dentro de IPv6 con traducción:

$$A_4 \diamond N_i \leftrightarrow Z_6 op_1 Z_4 op_2 Z_6 \leftrightarrow N_d \diamond A_4$$
$$\text{donde } N_i \in \{N_4, N_6, N_d\},\ op_1 \in \{\otimes_d, \odot_d, \boxtimes_d, \boxdot_d\},\ op2\ \{\odot_d, \boxdot_d\}$$
$$\Downarrow$$
$$T_{v6(v4)}\left(A_4 \diamond N_i \leftrightarrow Z_6 op_1 \underline{Z_4 op_2 Z_6 \leftrightarrow N_d} \diamond A_4\right) = A_4 \diamond N_i \leftrightarrow Z_6 op_1 Z_4 \odot_4 Z_4 \leftrightarrow N_d \diamond A_4$$

A continuación expondremos el problema para aplicaciones IPv6:

- Escenario no solución 1. Se creará un túnel IPv6 dentro de IPv4 que atraviese la zona Z_4 intermedia:

$$A_6 \diamond N_i \leftrightarrow Z_6 op_1 Z_4 op_2 Z_6 \leftrightarrow N_j \diamond A_6$$
$$\text{donde } N_i, N_j \in \{N_4, N_6, N_d\},\ op_1, op_2 \in \{\otimes_d, \odot_d, \boxtimes_d, \boxdot_d\}$$
$$\Downarrow$$
$$T_{v4(v6)}\left(A_6 \diamond N_i \leftrightarrow \underline{Z_6 op_1 Z_4 op_2 Z_6} \leftrightarrow N_j \diamond A_6\right) = A_6 \diamond N_i \leftrightarrow Z_6 \leftrightarrow N_j \diamond A_6$$

- Escenario no solución 2. Si un nodo terminal es dual, y la zona Z_6 conecta con la zona Z_4 por medio de un operador sin traducción de direcciones, se creará un túnel IPv4 dentro de IPv6:

$$A_6 \diamond N_i \leftrightarrow Z_6 op_1 Z_4 op_2 Z_6 \leftrightarrow N_d \diamond A_6$$
$$\text{donde } N_i \in \{N_4, N_6, N_d\},\ op_1 \in \{\otimes_d, \odot_d, \boxtimes_d, \boxdot_d\},\ op_2 \in \{\otimes_d, \boxtimes_d\}$$
$$\Downarrow$$
$$T_{v6(v4)}\left(A_6 \diamond N_i \leftrightarrow Z_6 op_1 \underline{Z_4 op_2 Z_6 \leftrightarrow N_d} \diamond A_6\right) = A_6 \diamond N_i \leftrightarrow Z_6 op_1 Z_4 \leftrightarrow N_d \diamond A_6$$

- Escenario no solución 3. Si un nodo terminal es dual, y la zona Z_6 conecta con la zona Z_4 por medio de un operador con traducción de direcciones, se creará un túnel IPv4 dentro de IPv6 con traducción:

$$A_6 \diamond N_i \leftrightarrow Z_6 op_1 Z_4 op_2 Z_6 \leftrightarrow N_d \diamond A_6$$
$$\text{donde } N_i \in \{N_4, N_6, N_d\},\ op_1 \in \{\otimes_d, \odot_d, \boxtimes_d, \boxdot_d\},\ op_2 \in \{\odot_d, \boxdot_d\}$$
$$\Downarrow$$
$$T_{v6(v4)}\left(A_6 \diamond N_i \leftrightarrow Z_6 op_1 \underline{Z_4 op_2 Z_6 \leftrightarrow N_d} \diamond A_6\right) = A_6 \diamond N_i \leftrightarrow Z_6 op_1 Z_4 \odot_4 Z_4 \leftrightarrow N_d \diamond A_6$$

Por último, veamos los casos para aplicaciones heterogéneas:

- Escenario no solución 1. Se creará un túnel IPv6 dentro de IPv4 que atraviese la zona Z_4 intermedia:

$$A_4 \diamond N_i \leftrightarrow Z_6 op_1 Z_4 op_2 Z_6 \leftrightarrow N_j \diamond A_6$$
$$\text{donde } N_i, N_j \in \{N_4, N_6, N_d\},\ op_1, op_2 \in \{\otimes_d, \odot_d, \boxtimes_d, \boxdot_d\}$$
$$\Downarrow$$
$$T_{v4(v6)}\left(A_4 \diamond N_i \leftrightarrow \underline{Z_6 op_1 Z_4 op_2 Z_6} \leftrightarrow N_j \diamond A_6\right) = A_4 \diamond N_i \leftrightarrow Z_6 \leftrightarrow N_j \diamond A_6$$

- Escenario no solución 2. Si la aplicación A_6 se ejecuta sobre un nodo dual, y éste se conecta a una zona Z_6 que se comunica con la zona Z_4 por medio de un operador sin traducción de direcciones, se creará un túnel IPv4 dentro de IPv6:

$$A_4 \diamond N_i \leftrightarrow Z_6 op_1 Z_4 op_2 Z_6 \leftrightarrow N_d \diamond A_6$$
$$\text{donde } N_i \in \{N_4, N_6, N_d\},\ op_1 \in \{\otimes_d, \odot_d, \boxtimes_d, \boxdot_d\},\ op_2 \in \{\otimes_d, \boxtimes_d\}$$
$$\Downarrow$$
$$T_{v6(v4)}\left(A_4 \diamond N_i \leftrightarrow Z_6 op_1 \underline{Z_4 op_2 Z_6 \leftrightarrow N_d} \diamond A_6\right) = A_4 \diamond N_i \leftrightarrow Z_6 op_1 Z_4 \leftrightarrow N_d \diamond A_6$$

- Escenario no solución 3. Como en el caso anterior, pero esta vez si el operador soporta traducción de direcciones, el túnel también lo hará:

$$A_4 \diamond N_i \leftrightarrow Z_6 op_1 Z_4 op_2 Z_6 \leftrightarrow N_d \diamond A_6$$
$$\text{donde } N_i \in \{N_4, N_6, N_d\},\ op_1 \in \{\otimes_d, \odot_d, \boxtimes_d, \boxdot_d\},\ op_2 \in \{\odot_d, \boxdot_d\}$$
$$\Downarrow$$
$$T_{v6(v4)}\left(A_4 \diamond N_i \leftrightarrow Z_6 op_1 \underline{Z_4 op_2 Z_6 \leftrightarrow N_d} \diamond A_6\right) = A_4 \diamond N_i \leftrightarrow Z_6 op_1 Z_4 \odot_4 Z_4 \leftrightarrow N_d \diamond A_6$$

- Escenario no solución 4. Si la aplicación A_4 se ejecuta sobre un nodo dual, y éste se conecta a una zona Z_6 que se comunica con la zona Z_4 por medio de un operador sin traducción de direcciones, se creará un túnel IPv4 dentro de IPv6:

$$A_4 \diamond N_d \leftrightarrow Z_6 op_1 Z_4 op_2 Z_6 \leftrightarrow N_i \diamond A_6$$
$$\text{donde } N_i \in \{N_4, N_6, N_d\},\ op_1 \in \{\otimes_d, \boxtimes_d\},\ op_2 \in \{\otimes_d, \odot_d, \boxtimes_d, \boxdot_d\}$$
$$\Downarrow$$
$$T_{v6(v4)}\left(A_4 \diamond \underline{N_d \leftrightarrow Z_6 op_1 Z_4} op_2 Z_6 \leftrightarrow N_i \diamond A_6\right) = A_4 \diamond N_d \leftrightarrow Z_4 op_2 Z_6 \leftrightarrow N_i \diamond A_6$$

- Escenario no solución 5. Como en el caso anterior, pero esta vez si el operador soporta traducción de direcciones, el túnel también lo hará:

$$A_4 \diamond N_d \leftrightarrow Z_6 op_1 Z_4 op_2 Z_6 \leftrightarrow N_i \diamond A_6$$
$$\text{donde } N_i \in \{N_4, N_6, N_d\},\ op_1 \in \{\odot_d, \boxdot_d\},\ op_2 \in \{\otimes_d, \odot_d, \boxtimes_d, \boxdot_d\}$$
$$\Downarrow$$
$$T_{v6(v4)}\left(A_4 \diamond \underline{N_d \leftrightarrow Z_6 op_1 Z_4} op_2 Z_6 \leftrightarrow N_i \diamond A_6\right) = A_4 \diamond N_d \leftrightarrow Z_4 \odot_4 Z_4 op_2 Z_6 \leftrightarrow N_i \diamond A_6$$

6.6.4. Partes de escenarios

No es viable contemplar todos los tamaños de escenario posibles, ya que su número es infinito. Por tanto, necesitamos algún método para poder simplificar los escenarios que tengan más de tres zonas, con el objetivo de reducirlos hasta el punto de que se pueda aplicar alguna de las reglas vistas anteriormente. Veremos reglas de partes de escenario, que se aplican sólo a los nodos intermedios de la red de interconexión. No hará falta tener en cuenta ni las aplicaciones ni los nodos terminales. Estas reglas se aplicarán en todas las posiciones posibles usando un algoritmo de búsqueda de patrones:

- Escenario no solución 1. Se creará un túnel IPv4 en IPv6:

$$\begin{gathered} Z_4 op_1 Z_6 op_2 Z_4 \\ \text{donde } op_1, op_2 \in \{\otimes_d, \odot_d, \boxtimes_d, \boxdot_d\} \\ \Downarrow \\ T_{v6(v4)}\left(Z_4 op_1 Z_6 op_2 Z_4\right) = Z_4 \end{gathered}$$

- Escenario no solución 2. Se creará un túnel IPv6 en IPv4:

$$\begin{gathered} Z_6 op_1 Z_4 op_2 Z_6 \\ \text{donde } op_1, op_2 \in \{\otimes_d, \odot_d, \boxtimes_d, \boxdot_d\} \\ \Downarrow \\ T_{v4(v6)}\left(Z_6 op_1 Z_4 op_2 Z_6\right) = Z_6 \end{gathered}$$

7 Conclusiones del análisis

Una vez estudiado el problema, podemos proceder a extraer las conclusiones apropiadas. Con ello pretendemos establecer una serie de criterios que nos resultarán de ayuda posteriormente en la fase de diseño, así como resumir lo expuesto en esta parte:

- El proceso de migración es muy complejo, ya que son muchas las posibles alternativas a considerar. Por tanto, resultaría interesante crear un programa que facilitara, en la medida de lo posible, el trabajo del administrador de red.
- La metodología MENINA es un ejemplo de buena ingeniería del software, al formalizar el problema y así poder tratarlo de manera rigurosa.
- MENINA sólo trabaja con escenarios lineales, pero sabemos que todas las redes reales tienen topologías mucho más complejas. Por tanto, necesitaremos encontrar una manera de transformar una topología arbitraria en una secuencia de uno o más escenarios lineales.
- Las reglas para la creación de túneles entre nodos terminales son problemáticas, porque al conservar la longitud de los escenarios, es posible que la aplicación de una secuencia concreta de reglas sobre un escenario devuelva el escenario original. Si se vuelven a aplicar, el programa entraría en un bucle infinito. Por tanto, estas reglas se reservarán para el final, y sólo se podrá aplicar una sobre un escenario.
- La metodología MENINA representa los escenarios de manera matemática, usando multitud de símbolos. Esto implica que tendremos que buscar un lenguaje de programación que permita definir y manipular de la manera más sencilla posible estructuras de datos complejas.
- El proceso de búsqueda de soluciones es no determinista; es decir, tomando un escenario concreto, podremos aplicarle varias reglas en diferentes puntos. Esto favorece el uso de un lenguaje que facilite el no determinismo.
- En una topología grande el número de rutas entre dos nodos puede ser muy elevado, y cada ruta puede generar gran cantidad de soluciones. Por tanto, interesa usar un lenguaje que ofrezca un alto rendimiento.

Parte IV

Diseño

8 Entradas y salidas

8.1. Introducción

En este capítulo se busca definir cómo será la interfaz de usuario, y las acciones que éste podrá llevar a cabo. Para ello partiremos de un diagrama de casos de uso, que definirá el modelo de comportamiento de la aplicación. A continuación, definiremos el formato que debe tener la entrada y la salida de datos. Por último, modelaremos la interfaz de usuario, intentando que sea lo más sencilla y atractiva posible.

8.2. Diagrama de casos de uso

Actores

- Usuario: el único tipo de usuario al que se dirige la herramienta es el administrador de red, por lo que no se consideran más actores.

Casos de uso

- Gestionar topologías: este caso de uso se corresponde con las necesidades de cualquier programa de guardar y restaurar sus datos.
  - Cargar topología: permite leer topologías desde un fichero almacenado en disco.
  - Guardar topología: permite escribir la topología actualmente en memoria a un fichero en disco.
  - Crear topología: con esta opción el usuario podrá crear una topología en blanco, sin ningún nodo.
- Editar topologías: este será el caso de uso principal, ya que permitirá al usuario modificar la topología para adaptarla a sus necesidades.
  - Añadir nodo: crea un nuevo nodo, asignándole el protocolo seleccionado en ese momento.
  - Borrar nodo: elimina un nodo que ya no es necesario.
  - Editar nodo: permite elegir el tipo de protocolo que soportará el nodo.
  - Añadir enlace: crea un nuevo enlace que conectará dos nodos existentes.
  - Borrar enlace: elimina un enlace que ya no es necesario.

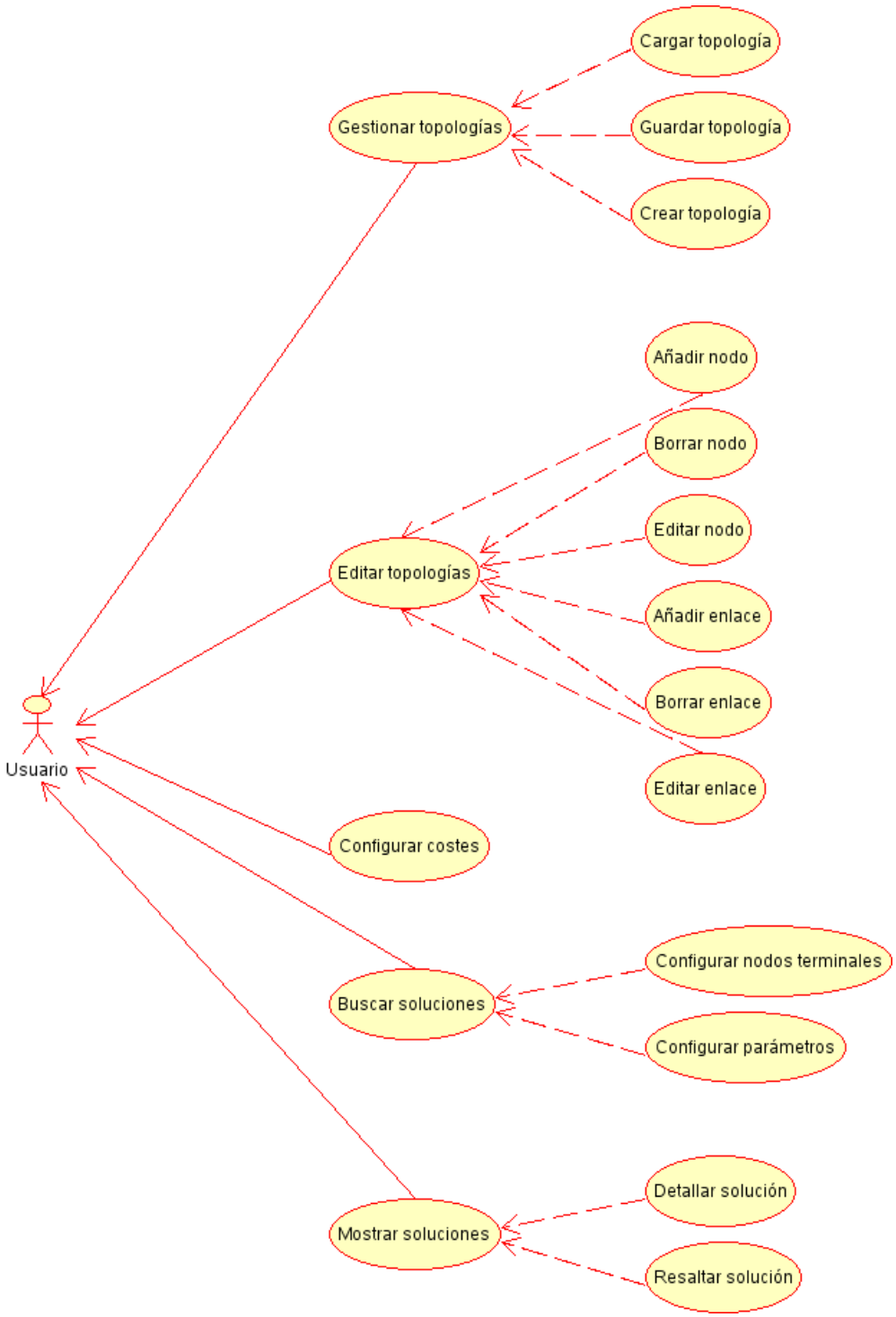

Figura 8.1: Diagrama de casos de uso

    - Editar enlace: permite configurar los parámetros de latencia, ancho de banda, etc. de un enlace.

- Configurar costes: permite establecer el coste asociado a cada mecanismo de transición, según diversos criterios como latencia, ancho de banda, etc.

- Buscar soluciones: este caso de uso sólo podrá ejecutarse sobre escenarios que tengan al menos un nodo.

    - Configurar nodos terminales: establece a qué routers están conectados los nodos terminales, y los protocolos que soportan ellos y las aplicaciones que ejecutan.
    - Configurar parámetros: permite establecer opciones de optimización de escenarios, para recortar el exceso de soluciones en topologías muy complejas.

- Mostrar soluciones: con este caso de uso el usuario podrá visualizar los resultados obtenidos con diferentes grados de detalle.

    - Resaltar solución: indicará la ruta seguida en la topología por una solución particular.
    - Detallar solución: indicará los túneles que se han creado y sus extremos, así como otros mecanismos de transición, para una solución particular.

8.3. Entradas

La única entrada que tomará el sistema será la configuración de la topología. Por motivos de eficiencia y simplicidad, la base de reglas estará integrada en el propio código del programa, por lo que no se considera como entrada.

La primera manera de configurar la topología será usar la propia interfaz visual del programa. El usuario contará con una ventana de edición que representará la posición de cada router y enlace en el espacio, así como un panel para configurar los atributos de estos y elegir la apariencia gráfica.

La segunda manera será cargar la topología desde un fichero almacenado en disco. Este fichero se guardará en formato GML, que explicaremos más adelante. Añadiremos una serie de extensiones al formato para guardar los datos propios de nuestra aplicación.

Hay que tener en cuenta que la definición de las topologías sólo incluirá la disposición de los routers intermedios, pero no incluirá información sobre los nodos terminales o las aplicaciones. Por tanto, habrá que configurar estos parámetros por separado antes de realizar una búsqueda de soluciones.

8.4. Salidas

La salida consistirá en un listado con las soluciones encontradas para un escenario dado. Podrá mostrarse de manera gráfica o mediante texto, aprovechando los caracteres

Unicode para representar los operadores y símbolos especiales. Cada una de las soluciones contendrá los siguientes campos:

- Ruta seguida en la topología.
- Escenario inicial, antes de aplicar ninguna regla.
- Escenario final con conectividad, después de aplicar las reglas necesarias.
- Lista con las reglas que ha hecho falta aplicar para llegar del estado inicial al final.
- Costes de la solución según diversos criterios.

8.5. Interfaz de usuario

8.5.1. Ventana de edición

La ventana de edición constituirá la interfaz de trabajo principal. Implementará directamente los casos de uso GESTIONAR TOPOLOGÍAS y EDITAR TOPOLOGÍAS, y permitirá el acceso al resto de casos de uso.

La ventana de constará de cuatro secciones. La primera de ellas será la *barra de menús*, situada en la parte superior, desde la que se podrá acceder al resto de diálogos. La segunda estará situada en la parte izquierda, y se dividirá a su vez en tres subsecciones:

- *Modo de edición*: permitirá elegir la operación a realizar, entre las cuatro posibles (creación y edición de routers y enlaces).
- *Configuración de la apariencia gráfica*: aquí se podrá elegir el color con el que se representará cada tipo de router, según su protocolo (IPv4, IPv6, dual, etc).
- *Edición de atributos*: si se ha seleccionado una operación de edición, aquí se podrán establecer los nuevos valores del elemento seleccionado, como el protocolo de un router o la latencia de un enlace.

La tercera sección será el *área de edición*, situada en la parte central. Ofrecerá una representación gráfica de la topología de red. El usuario podrá seleccionar los elementos con el ratón para editarlos y desplazarlos a su antojo. Igualmente, al crear nuevos routers y enlaces se podrá usar el ratón para establecer su posición inicial. También se ofrecerá una función de zoom, para ajustar el campo de visión.

La cuarta y última sección será el *listado de soluciones*, situado en la parte derecha. Inicialmente estará vacía, hasta que el usuario haga una búsqueda. En ese momento mostrará las soluciones encontradas, ordenadas por coste. Cada solución estará formada por una columna con la ruta, y una columna para cada tipo de coste. Pinchando en las columnas de coste se podrán reordenar los resultados según convenga. Al seleccionar una solución, se

resaltará la ruta seguida en el área de edición. Al hacer doble click sobre una solución, se invocará el diálogo de presentación de solución, para mostrarla con más detalle.

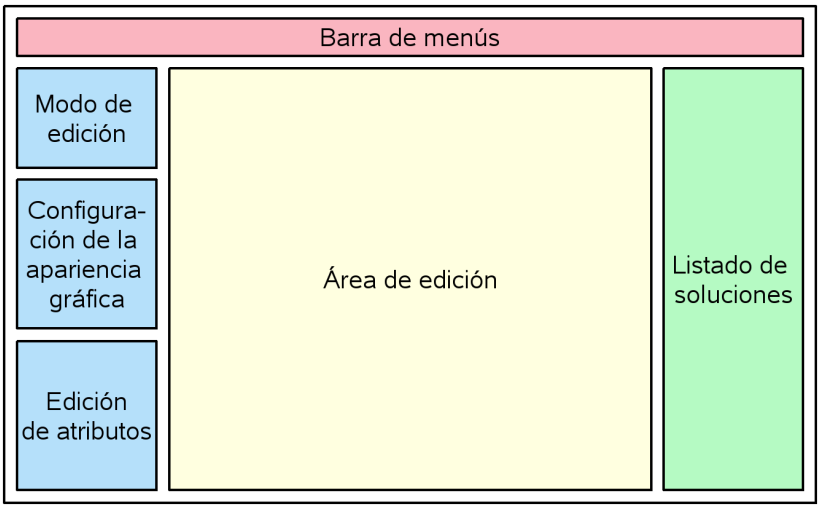

Figura 8.2: Esquema de la ventana de edición

8.5.2. Configuración de costes

Este diálogo permitirá al usuario establecer el coste de cada mecanismo de transición. Consistirá en una ventana con varias pestañas, una por cada tipo de coste. El contenido de cada pestaña será el mismo, una secuencia de campos de entrada en los que el usuario podrá introducir los valores deseados. La aplicación comprobará que el contenido de los campos sea siempre numérico y no negativo.

Además, el diálogo contará con dos botones: *Aceptar*, que guardará las modificaciones, y *Cancelar*, que las descartará. Por otro lado, cuando la aplicación se empiece a ejecutar, se inicializarán los costes a sus valores predeterminados.

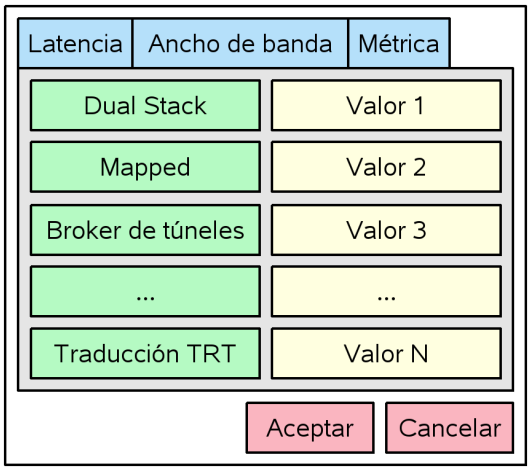

Figura 8.3: Esquema de la configuración de costes

8.5.3. Diálogo de búsqueda

Este diálogo permitirá al usuario iniciar al proceso de búsqueda de soluciones, para lo cual primero tendrá que establecer varios parámetros. Los parámetros se agruparán en tres secciones.

La sección superior contendrá las opciones relativas al nodo de origen. El usuario podrá establecer a que router está conectado el nodo terminal, el tipo de aplicación que ejecuta (A_4, A_6, A_d), y los protocolos que soporta el nodo (N_4, N_6, N_d, $N_{d,MAP}$).

La sección media será similar a la superior, pero en esta ocasión establecerá las opciones relativas al nodo de destino. En ambos casos, la aplicación comprobará que los valores introducidos son correctos, para que, por ejemplo, no se seleccione un router que no existe.

La tercera sección permitirá elegir si se va a usar algún algoritmo de optimización. Estos algoritmos permiten recortar el número de soluciones obtenidas en caso de que éste sea excesivo. Se ofrecerán varios algoritmos, que el usuario podrá combinar como desee.

El botón *Aceptar* guardará los parámetros para la siguiente vez, y lanzará el proceso de búsqueda. Al terminar, rellenará el campo *listado de soluciones* de la ventana de edición con los resultados obtenidos. Se dará al usuario la opción de cancelar la búsqueda en todo momento si ve que tarda demasiado. El botón *Cancelar* descartará los parámetros y volverá a la ventana de edición.

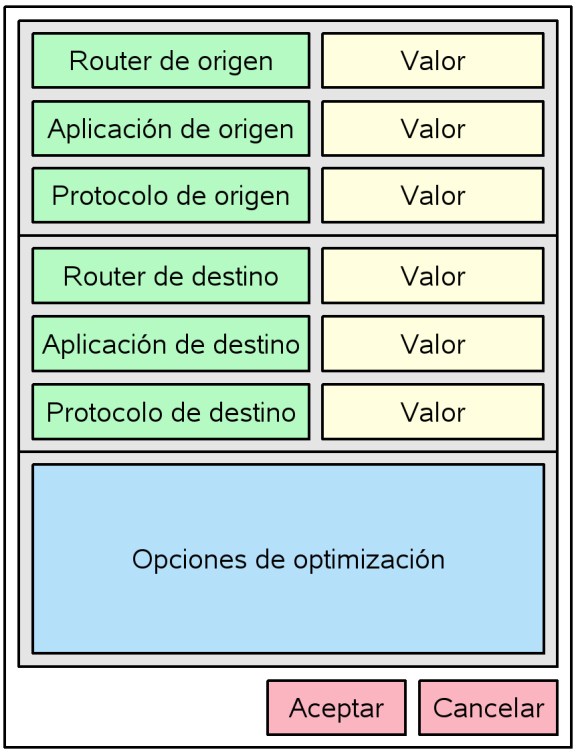

Figura 8.4: Esquema del diálogo de búsqueda

8.5.4. Presentación de una solución

Este diálogo mostrará toda la información relativa a una solución. En concreto, se indicará la ruta seguida, el coste total según los diferentes criterios, los mecanismos que se han tenido que aplicar, detallando los extremos de los túneles, y el escenario final que se alcanza después de aplicar las reglas. Se procurará que la representación sea lo más visual posible, para facilitar su comprensión.

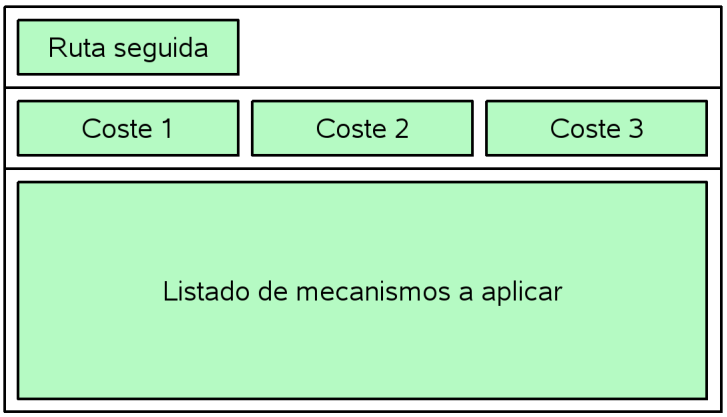

Figura 8.5: Esquema de la presentación de una solución

9 Casos de uso detallados

9.1. Introducción

En las siguientes secciones definiremos con mayor detalle cada caso de uso. Indicaremos, para cada uno de ellos, si depende de algún otro caso, los actores que intervienen, las precondiciones que debe cumplir el programa antes de que se llame al caso de uso, las postcondiciones que se cumplirán cuando el caso de uso haya terminado de ejecutarse, el flujo normal de ejecución, los posibles flujos alternativos que pueden darse si el usuario cancela la acción o surge alguna situación excepcional, y las observaciones adicionales que estimemos necesario.

9.2. Casos de uso principales

Nombre del caso	Gestionar topologías
Resumen	Permite al usuario llevar el control de las topologías que tiene almacenadas en el disco duro.
Dependencias	Extiende los casos de uso ”Cargar topología”, ”Guardar topología”, y ”Crear topología”.
Actores	Usuario.
Precondiciones	Ninguna.
Postcondiciones	La nueva topología se marcará como no modificada.
Curso normal	Si el programa se invoca sin ningún argumento, se ejecutará automáticamente ”Crear topología”.
Cursos alternativos	Si el programa se invoca con un argumento, que corresponderá con la ruta de un fichero de topología, se ejecutará automáticamente ”Cargar topología”.
Observaciones	

Nombre del caso	Editar topologías
Resumen	Permite al usuario manipular la topología cargada en ese momento.

Nombre del caso	Editar topologías
Dependencias	Extiende los casos de uso ”Añadir nodo”, ”Borrar nodo”, ”Editar nodo”, ”Añadir enlace”, ”Borrar enlace”, y ”Editar enlace”.
Actores	Usuario.
Precondiciones	Ninguna.
Postcondiciones	La topología se guardará como modificada.
Curso normal	El usuario seleccionará la acción a realizar en el panel de *modo de edición* de la interfaz. A continuación, usará el ratón para manipular los elementos deseados dentro del *área de edición*.
Cursos alternativos	
Observaciones	

Nombre del caso	Configurar costes
Resumen	Permite establecer el coste de cada mecanismo de transición, para cada uno de los criterios considerados.
Dependencias	
Actores	Usuario.
Precondiciones	Ninguna.
Postcondiciones	Guardará la nueva tabla de costes, o la dejará como estaba si el usuario canceló la operación.
Curso normal	El diálogo mostrará los valor que habían sido configurados anteriormente. El usuario introducirá los nuevos costes, y pulsará en el botón *aceptar*. Se guardarán los valores.
Cursos alternativos	Si el usuario pulsa el botón *cancelar*, se descartarán las modificaciones.
Observaciones	La primera vez que se ejecute este caso de uso se inicializarán los costes a un valor predeterminado.

Nombre del caso	Buscar soluciones
Resumen	Lanza el proceso de búsqueda de soluciones, calcula el coste asociado a cada una, y devuelve el resultado al usuario.
Dependencias	Extiende los casos de uso ”Configurar nodos terminales” y ”Configurar parámetros”.
Actores	Usuario.
Precondiciones	La topología debe contar al menos con un router.
Postcondiciones	Devolverá la lista de soluciones encontradas.

Nombre del caso	Buscar soluciones
Curso normal	Se ejecutarán los dos casos de uso extendidos, para que el usuario introduzca los parámetros de la búsqueda. Cuando el usuario pulse el botón *aceptar*, se llamará a la función de búsqueda de soluciones, y se devolverá una lista con los resultados encontrados. Se guardarán los parámetros para la siguiente vez.
Cursos alternativos	Si el usuario pulsa el botón *cancelar* durante la fase de configuración o durante la búsqueda, se parará el proceso. No se guardarán las modificaciones, y se devolverá como resultado una lista vacía.
Observaciones	Este caso de uso puede devolver una lista vacía en dos circunstancias: si no se puede encontrar ninguna solución, o si el usuario canceló la operación.

Nombre del caso	Mostrar soluciones
Resumen	Ofrece al usuario la información relativa a las soluciones encontradas, con diferentes grados de detalle.
Dependencias	Extiende los casos de uso ”Resaltar solución” y ”Detallar solución”.
Actores	Usuario.
Precondiciones	Sólo se podrá invocar después de haber ejecutado ”Buscar soluciones”.
Postcondiciones	Mostrará en pantalla los resultados encontrados por el caso de uso anterior.
Curso normal	Una vez que el usuario haya ejecutado el caso de uso ”Buscar soluciones”, se rellenará el *listado de soluciones* con los resultados. Si el usuario hace un click sobre una solución, se llamará al caso de uso ”Resaltar solución”. Si hace doble click, se llamará a ”Detallar solución”.
Cursos alternativos	Si el usuario pincha sobre una de las columnas de la tabla, se reordenarán las soluciones en función de esa columna.
Observaciones	

9.3. Casos de uso extendidos

9.3.1. Extendidos de Gestionar topologías

Nombre del caso	Cargar topología
Resumen	Lee un fichero que contiene la descripción de la topología. Sólo se considera la disposición de los routers y los enlaces entre estos. No se guarda la información relativa a los nodos terminales.
Dependencias	
Actores	Usuario.
Precondiciones	La ruta seleccionada es válida.
Postcondiciones	La topología se ha cargado y mostrado en pantalla.
Curso normal	El usuario elegirá la ruta, se leerá el fichero, y se mostrará la topología en pantalla.
Cursos alternativos	Si el fichero contiene algún error, se mostrará un mensaje al usuario. Si el usuario cancela la operación, se volverá a la topología que ya estaba cargada.
Observaciones	Si la topología actual ha sido modificada y aún no se ha guardado en disco, se preguntará al usuario si quiere hacerlo antes de cargar la nueva.

Nombre del caso	Guardar topología
Resumen	Escribe un fichero que contiene la descripción de la topología. Sólo se considera la disposición de los routers y los enlaces entre estos. No se guarda la información relativa a los nodos terminales.
Dependencias	
Actores	Usuario.
Precondiciones	La ruta seleccionada es válida.
Postcondiciones	Se ha guardado la topología en el fichero.
Curso normal	El usuario elegirá el nombre del fichero, y éste se escribirá en el disco.

Nombre del caso	Guardar topología
Cursos alternativos	Si el fichero ya existe, se preguntará al usuario si quiere sobreescribirlo. Si se produce algún error (disco lleno, permisos insuficientes, etc.), se mostrará un mensaje de error al usuario. Si el usuario cancela la operación, no se realizará ninguna escritura.
Observaciones	Después de guardar una topología que estaba marcada como modificada, dejará de estarlo.

Nombre del caso	Crear topología
Resumen	Crea un nuevo fichero de topología, inicialmente vacío.
Dependencias	
Actores	Usuario.
Precondiciones	Ninguna.
Postcondiciones	Se ha inicializado una nueva topología con los valores predeterminados.
Curso normal	Se crea una topología completamente vacía, sin ningún router ni enlace.
Cursos alternativos	
Observaciones	Si la topología actual ha sido modificada y aún no se ha guardado en disco, se preguntará al usuario si quiere hacerlo antes de crear la nueva.

9.3.2. Extendidos de Editar topologías

Nombre del caso	Añadir nodo
Resumen	Crea un nuevo router en la posición actual del puntero del ratón, asignándole el protocolo seleccionado.
Dependencias	
Actores	Usuario.
Precondiciones	El usuario ha seleccionado un protocolo.
Postcondiciones	Se ha agregado un nuevo router al escenario.
Curso normal	El usuario pincha en las coordenadas deseadas para el nuevo router, y éste se añade.
Cursos alternativos	

Nombre del caso	Añadir nodo
Observaciones	Cada router tendrá un identificador numérico, que será generado automáticamente a partir de una secuencia global creciente.

Nombre del caso	Borrar nodo
Resumen	Elimina el router seleccionado.
Dependencias	
Actores	Usuario.
Precondiciones	Se han seleccionado uno o más routers.
Postcondiciones	Se han borrado los routers seleccionado.
Curso normal	El usuario hace click sobre un router, seleccionándolo. Podrá seleccionar varios manteniendo pulsada la tecla SHIFT. A continuación pulsa la tecla SUPRIMIR, y los routers se borran.
Cursos alternativos	Si el usuario pulsa sobre un espacio vacío, se deseleccionan los routers y se cancela la operación.
Observaciones	Cuando se seleccione un router, se resaltará mostrándolo de otro color.

Nombre del caso	Editar nodo
Resumen	Permite cambiar el tipo de protocolo soportado por los routers seleccionados, así como desplazarlos a una nueva posición.
Dependencias	
Actores	Usuario.
Precondiciones	Se han seleccionado uno o más nodos.
Postcondiciones	Se ha cambiado el protocolo de los routers seleccionados.
Curso normal	El usuario elige uno o más routers, igual que en el caso de uso anterior. A continuación, elige el nuevo protocolo de una lista desplegable. Se actualizan los atributos de los routers seleccionados.
Cursos alternativos	Si el usuario pulsa sobre un espacio vacío, se deseleccionan los routers y se cancela la operación.
Observaciones	Cuando se seleccione un router, se resaltará mostrándolo de otro color.

Nombre del caso	Añadir enlace
Resumen	Crea un nuevo enlace entre los dos routers seleccionados.
Dependencias	
Actores	Usuario.
Precondiciones	El escenario posee al menos dos routers.
Postcondiciones	Se ha agregado un nuevo enlace al escenario.
Curso normal	El usuario pulsará sobre un router, marcándolo como el origen. A continuación pulsará sobre el segundo router, y se creará el enlace.
Cursos alternativos	Si el usuario pulsa sobre un espacio vacío antes de elegir el router de destino, se borrará la selección y se cancelará la operación.
Observaciones	Cuando se seleccione el primer router, se resaltará mostrándolo de otro color. Después de crear el enlace se borrará la selección.

Nombre del caso	Borrar enlace
Resumen	Elimina los enlaces seleccionados.
Dependencias	
Actores	Usuario.
Precondiciones	Se han seleccionado uno o más enlaces.
Postcondiciones	Se han borrado los enlaces seleccionados.
Curso normal	El usuario hace click sobre un enlace, seleccionándolo. Podrá seleccionar varios manteniendo pulsada la tecla SHIFT. A continuación pulsa la tecla SUPRIMIR, y los enlaces se borran.
Cursos alternativos	Si el usuario pulsa sobre un espacio vacío, se deseleccionan los enlaces y se cancela la operación.
Observaciones	Cuando se seleccione un enlace, se resaltará mostrándolo de otro color.

Nombre del caso	Editar enlace
Resumen	Permite modificar los parámetros de latencia, ancho de banda, etc., de los enlaces seleccionados.
Dependencias	
Actores	Usuario.
Precondiciones	Se han seleccionado uno o más enlaces.

Nombre del caso	Editar enlace
Postcondiciones	Se han cambiado los atributos de los enlaces seleccionados.
Curso normal	El usuario elige uno o más enlaces, igual que en el caso de uso anterior. A continuación, introduce los nuevos atributos. Los enlaces seleccionados se actualizan automáticamente.
Cursos alternativos	Si el usuario pulsa sobre un espacio vacío, se deseleccionan los enlaces y se cancela la operación.
Observaciones	Cuando se seleccione un enlace, se resaltará mostrándolo de otro color.

9.3.3. Extendidos de Buscar soluciones

Nombre del caso	Configurar nodos terminales
Resumen	Permite elegir los routers origen y destino de la ruta, los protocolos soportados por los nodos terminales, y el tipo de aplicaciones que ejecutan.
Dependencias	
Actores	Usuario.
Precondiciones	La topología posee al menos un router.
Postcondiciones	Devolverá los parámetros introducidos por el usuario, comprobando que los routers indicados existen en la topología.
Curso normal	El usuario introducirá los valores deseados, pulsará el botón *aceptar*, y se devolverán los valores.
Cursos alternativos	Si el usuario pulsa el botón *cancelar*, se terminará el caso de uso sin devolver nada.
Observaciones	El valor inicial de los parámetros será: ▪ $R_{origen} = 1$ ▪ $R_{destino} = 1$ ▪ $N_{origen} = N_4$ ▪ $N_{destino} = N_4$ ▪ $A_{origen} = A_4$ ▪ $A_{destino} = A_4$

Nombre del caso	Configurar parámetros
Resumen	Permite elegir los algoritmos de poda de soluciones que se usarán para acelerar la búsqueda.
Dependencias	
Actores	Usuario.
Precondiciones	Ninguna.
Postcondiciones	Devolverá los parámetros introducidos por el usuario.
Curso normal	El usuario introducirá los valores deseados, pulsará el botón *aceptar*, y se devolverán los valores.
Cursos alternativos	Si el usuario pulsa el botón *cancelar*, se terminará el caso de uso sin devolver nada.
Observaciones	Inicialmente todas las opciones estarán desactivadas.

9.3.4. Extendidos de Mostrar soluciones

Nombre del caso	Resaltar solución
Resumen	Indica la ruta seguida por una solución, sin entrar en detalles.
Dependencias	
Actores	Usuario.
Precondiciones	El usuario ha ejecutado antes el caso de uso "Buscar soluciones".
Postcondiciones	Se ha actualizado el *área de edición* para resaltar la ruta seguida.
Curso normal	El usuario hará un solo click sobre una entrada de la lista de resultados, y se resaltarán los enlaces entre los routers por los que pasa la solución.
Cursos alternativos	
Observaciones	Para resaltar un enlace, se mostrará de un color que contraste con los demás.

Nombre del caso	Detallar solución
Resumen	Indica la ruta seguida por una solución, su coste, la secuencia de mecanismos de transición que se han aplicado, y el resultado final que se alcanza.
Dependencias	
Actores	Usuario.

Nombre del caso	Detallar solución
Precondiciones	El usuario ha ejecutado antes el caso de uso ”Buscar soluciones”.
Postcondiciones	Se ha abierto una nueva ventana con toda la información.
Curso normal	El usuario hará doble click sobre una entrada de la lista de resultados, y se abrirá el diálogo de presentación de solución para mostrar la información.
Cursos alternativos	
Observaciones	

10 Estructura del programa

10.1. Introducción

En este capítulo definiremos la manera en la que estará estructurado el programa. Ésta es una de las partes más complejas del desarrollo, ya que son muchas las posibilidades a considerar. Hemos procurado elegir aquellas que hagan el programa más eficiente, pero manteniendo la implementación sencilla, modular y abierta a futuras modificaciones.

En primer lugar, describiremos la manera de almacenar topologías. A continuación, expondremos la estrategia de resolución que usará el programa, con el objetivo de aplicar la base de reglas a los escenarios. Después definiremos el flujo de ejecución del programa, suponiendo un uso normal del mismo. También expondremos los tipos de datos abstractos. Por último, mostraremos la jerarquía de módulos del programa en forma de diagramas.

10.2. Almacenamiento de las topologías

Para almacenar las topologías en disco usaremos el formato GML (*Graph Modelling Language*)[1]. Es un formato muy sencillo, basado en texto estructurado jerárquicamente.

Un documento GML se define como una lista de pares clave–valor. Las claves son cadenas de texto que empiezan por una letra, seguida de un número arbitrario de letras y dígitos. Los valores pueden ser números enteros, números reales, cadenas de texto, o listas anidadas. La definición sintáctica completa de GML es la siguiente:

```
GML         ::= List
List        ::= (whitespace* Key whitespace+ Value)*
Value       ::= Integer | Real | String | '[' List ']'
Key         ::= [ a-z A-Z ] [ a-z A-Z 0-9 ]*
Integer     ::= sign digit+
Real        ::= sign digit* '.' digit* mantissa
String      ::= '"' instring '"'
sign        ::= empty | '+' | '-'
digit       ::= [0-9]
Mantissa    ::= empty | 'E' sign digit
instring    ::= ASCII - {'&','"'} | '&' character+ ';'
whitespace  ::= space | tabulator | newline
```

Figura 10.1: Definición sintáctica de GML

[1] http://www.infosun.fim.uni-passau.de/Graphlet/GML/gml-tr.html

La sintaxis dada permite representar estructuras arbóreas arbitrarias. Para representar grafos, se definen las claves GRAPH, NODE y EDGE. El valor asociado con una clave GRAPH será una lista de elementos de tipo NODE, EDGE, y otros atributos arbitrarios. Los valores para las claves NODE y EDGE serán listas con los atributos de los nodos y arcos, respectivamente. La estructura topológica del grafo se modela con los atributos ID para los nodos y SOURCE y TARGET para los arcos: ID asigna un número a cada nodo, que es referenciado por SOURCE y TARGET. El siguiente código define una topología de ejemplo, un anillo con tres nodos:

```
graph [
        directed 0
        node [
                id 1
                protocol "R4"
        ]
        node [
                id 2
                protocol "R4"
        ]
        node [
                id 3
                protocol "R4"
        ]
        edge [
                source 1
                target 2
        ]
        edge [
                source 2
                target 3
        ]
        edge [
                source 3
                target 1
        ]
]
```

Figura 10.2: Ejemplo de grafo en GML

GML define una serie de atributos estándar, algunos de los cuales son de uso obligado por cualquier aplicación, y otros que son de uso opcional. Representaremos la ruta a un atributo con la notación $k_1, k_2, \ldots, k_n$, donde k_i es una clave. A continuación enumeramos los atributos que usará nuestra aplicación:

- .GRAPH *lista* – Define un grafo.
- .GRAPH.NODE *lista* – Define un nodo. Cada nodo debe contener un atributo .graph.node.id,

cuyo valor será único dentro del grafo.

- .GRAPH.EDGE *lista* – Define un arco. Cada arco debe tener los atributos .GRAPH.EDGE.SOURCE y .GRAPH.EDGE.TARGET, que harán referencia a nodos existentes en el grafo.
- .GRAPH.EDGE.SOURCE *entero* – Define el nodo de origen de un arco.
- .GRAPH.EDGE.TARGEt *entero* – Define el nodo de destino de un arco.
- .GRAPH.DIRECTED *entero* – Su valor es 1 si el grafo es dirigido, y 0 si no lo es.
- .ID *entero* – Asigna un identificador a un objeto, normalmente un nodo.
- .LABEL *texto* – Asigna una etiqueta a un objeto.
- .NAME *texto* – Asigna un nombre a un objeto.
- .GRAPHICS *lista* – Describe los atributos gráficos para dibujar un objeto, en nuestro caso un nodo. Dentro de este atributo usaremos la clave CENTER, que define la posición espacial del nodo, y se compone a su vez de los atributos X e Y.

Además de los valores estándar, GML permite a las aplicaciones definir sus propios atributos. Hemos añadido los siguientes:

- .NODE.PROTOCOL *texto* – Define los protocolos que soporta un router. Los valores permitidos son:
  - R4 – Router IPv4.
  - R6 – Router IPv6.
  - Rd – Router dual.
  - R4_td – Router IPv4 con traducción de direcciones.
  - Rd_td – Router dual con traducción de direcciones.
  - Rd_tp – Router dual con traducción de protocolos.
  - Rd_tptd – Router dual con traducción de protocolos y direcciones.
- .EDGE.LATENCY *real* – Define la latencia de un enlace.
- .EDGE.BANDWIDTH *real* – Define el ancho de banda del que dispone un enlace.
- .EDGE.METRIC *real* – Permite asignar un coste arbitrario a un enlace, según una métrica definida por el usuario.

Hasta aquí hemos visto cómo se estructura un fichero GML. Queda por definir el proceso de carga. Lo dividiremos en dos partes: conversión de texto a un árbol sintáctico, y extracción del grafo. El proceso de almacenamiento se hará al revés.

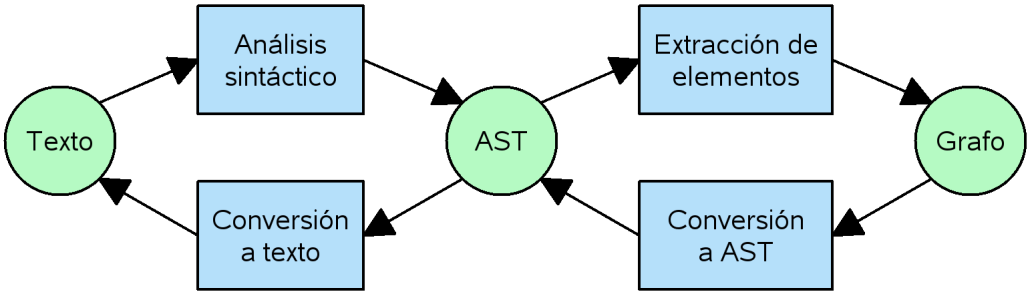

Figura 10.3: Carga y almacenamiento de topologías

El proceso de extracción se hará de manera recursiva. Necesitaremos una función que convierta de GML a cada tipo de elemento (grafo, nodo y arco). Partiendo de un grafo vacío, los pasos a seguir serán los siguientes:

- Buscar la clave GRAPH en el árbol, y procesar sus hijos.
- Si se encuentra la clave DIRECTED, leer su valor. Si no, se supone que el grafo es no dirigido.
- Para cada clave NODE encontrada, procesarla de manera recursiva, y añadir el nodo resultante al grafo.
- Para cada clave EDGE encontrada, procesarla de manera recursiva, y añadir el arco resultante al grafo.
- Añadir el resto de atributos no reconocidos, sin procesarlos.

La conversión de grafo a árbol sintáctico se hará justo al revés:

- Generar un elemento de tipo GRAPH.
- Añadir la clave DIRECTED, con su valor correspondiente.
- Para cada nodo del grafo, convertirlo a un árbol de manera recursiva, y añadir una clave NODE con el valor resultante.
- Para cada arco del grafo, convertirlo a un árbol de manera recursiva, y añadir una clave EDGE con el valor resultante.
- Añadir el resto de atributos.

10.3. Estrategia de resolución

Los problemas de búsqueda se suelen modelar conceptualmente como un proceso de tres pasos: generación de un espacio de soluciones con estructura de árbol, poda del árbol, y recorrido de las hojas para obtener las soluciones finales. El problema es que, para cualquier

problema no trivial, el árbol de soluciones puede ser muy grande, potencialmente infinito. Por tanto, necesitaremos definir algún criterio de poda que permita eliminar resultados innecesarios.

La primera etapa de la resolución es la obtención de escenarios lineales a partir de la topología. El programa contará con una función que tome como entrada un grafo y los extremos de las rutas, y devuelva como resultado una lista con todas las rutas encontradas. Las rutas serán a su vez una lista de los nodos por los que pasa.

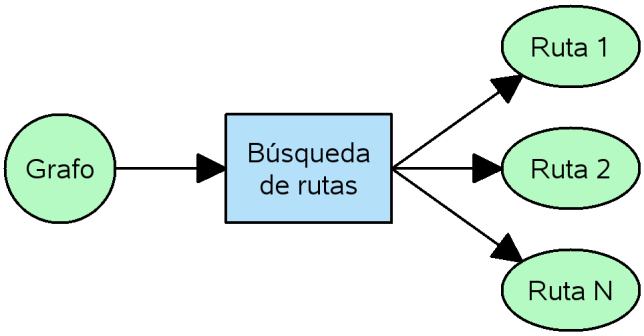

Figura 10.4: Esquema de búsqueda rutas

La segunda etapa será la aplicación de reglas. El primer paso será tomar las redes de distribución física encontradas en la etapa anterior, y construir los escenarios de transición. Este paso es muy sencillo, ya que los escenarios son tuplas de cinco elementos, de los cuales las aplicaciones finales y los nodos terminales son comunes:

$$\forall a_1, a_2 \in \mathcal{A},\ \forall n_1, n_2 \in \mathcal{N} \Longrightarrow escenarios = \{\langle a_1, n_1, red, n_2, a_2\rangle \mid red \in rutas\}$$

Figura 10.5: Esquema de generación de escenarios de transición

Una vez tengamos los escenarios de transición, habrá que resolver cada uno por separado. Para ello, primero se formalizará y canonizará el escenario, con el fin de simplificarlo lo más posible. A continuación se aplicarán las reglas de creación de túneles y las de modificación sobre redes finales.

Hay que tener en cuenta que algunas reglas, como las de creación de túneles entre nodos terminales, sólo se deben aplicar al final, cuando ya no queden más opciones, y como mucho una vez por escenario. Esto se debe a que existen pares de reglas recíprocas, es decir, dos reglas R_1 y R_2 que aplicadas secuencialmente a un escenario E darían el escenario original, $R_2\left(R_1\left(E\right)\right) = E$. Si aplicamos estas reglas sin cuidado, el programa entrará en un bucle infinito.

Por tanto, dividiremos la aplicación de la base de reglas en tres etapas:

- Túneles entre zonas intermedias, y entre zonas y nodos terminales.
- Túneles entre nodos terminales.

- Mecanismos de modificación de escenarios finales.

La aplicación de la base de reglas a un escenario tendrá como resultado una lista de escenarios modificados. Habrá que repetir el proceso de manera recursiva para cada nuevo escenario. El proceso terminará cuando no se pueda aplicar ninguna regla más. El resultado final será la obtención de una lista de soluciones candidatas a partir de cada escenario:

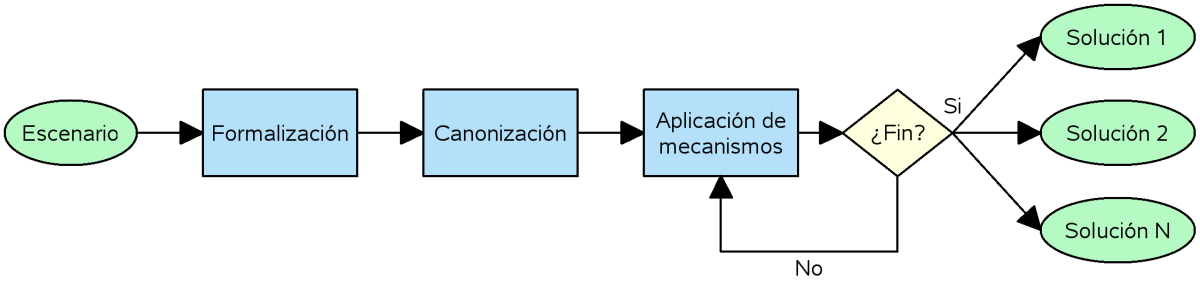

Figura 10.6: Esquema de búsqueda de soluciones

Después de resolver los escenarios, tendremos una lista de listas de soluciones. El último paso será concatenarlas en un sola lista, evaluar el coste de cada solución individual, y ordenarlas de acuerdo con el criterio establecido por el usuario.

10.4. Flujo de ejecución

La ejecución normal de la aplicación se compondrá de cuatro fases, que se ejecutarán secuencialmente. En primer lugar, el usuario cargará el fichero de topología, o creará uno nuevo. A continuación configurará la topología, editando los routers y enlaces. Después podrá configurar, si así lo desea, los costes asociados a cada mecanismo de transición. Por último, lanzará el proceso de búsqueda de soluciones. Cuando éste termine, mostrará en pantalla los resultados. Cada fase se puede descomponer en varias subtareas, como se muestra en el diagrama:

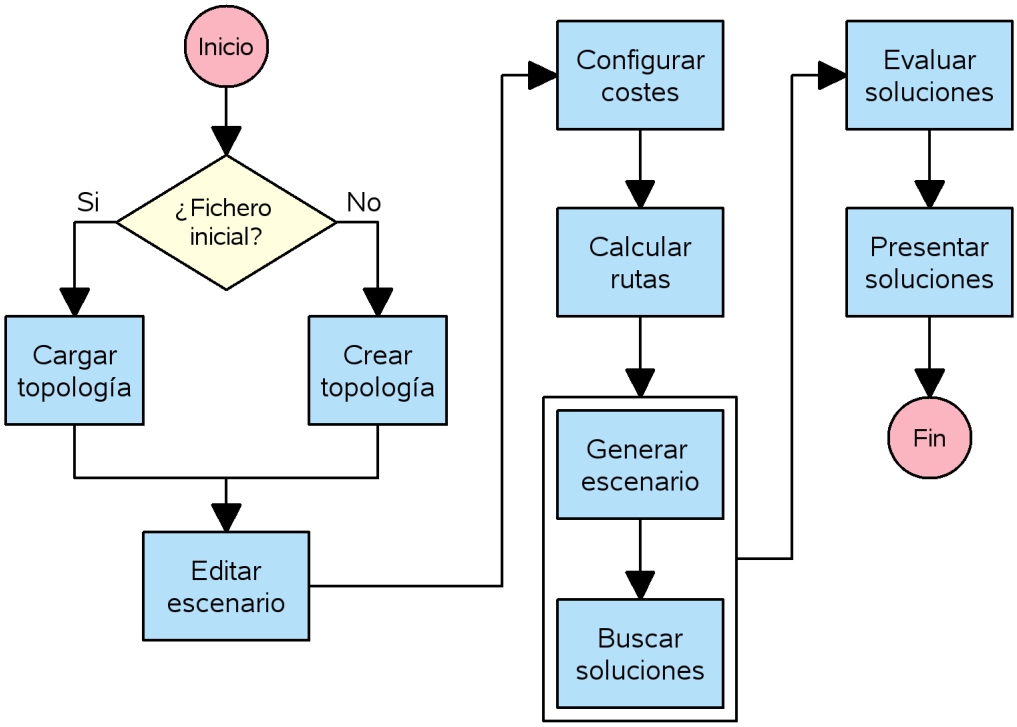

Figura 10.7: Flujo de la aplicación

10.5. Tipos de datos

Nombre	Valor
Utilidad	Representación de un valor en un árbol GML

Definición	`data Valor = VI Int`
	`| VR Double`
	`| VS String`
	`| VM Atributos`

Operaciones	LEERGML (TEXTO): VALOR – Convierte una cadena de texto en un árbol GML. ESCRIBIRGML (VALOR): TEXTO – Convierte un árbol GML en una cadena de texto

Nombre	Atributos
Utilidad	Lista de pares clave–valor en un árbol GML

Nombre	Atributos
Definición	`type Atributos = [(String, Valor)]`
Operaciones	

Nombre	Vector
Utilidad	Coordenadas de un punto en el espacio euclídeo de tres dimensiones.
Atributos	X: REAL – Valor de la coordenada X. Y: REAL – Valor de la coordenada Y. Z: REAL – Valor de la coordenada Z.
Operaciones	SUMAR (V1, V2): VECTOR – Devuelve un vector cuyas coordenadas son la suma de *v1* y *v2*. DISTANCIA_VV (V1, V2): REAL – Calcula la distancia entre dos vectores. DISTANCIA_LV (V0, V1, V2): REAL – Calcula la distancia entre el vector *v0* y la línea definida por *v1* y *v2*. DISTANCIA_PV (V, N, P): REAL – Calcula la distancia entre el vector *v* y el plano de centro *n* y normal *p*. TOGML (VECTOR): VALOR – Convierte un vector a su representación en GML. FROMGML (AST): VECTOR – Extrae un vector de un árbol sintáctico GML.

Nombre	Router
Utilidad	Tipo enumerado que define los diferentes tipos de routers posibles.

Nombre	Router
Definición	`data Router = R4 | R6 | Rd | R4_td | Rd_td | Rd_tp | Rd_tptd | Z4 | Z6 | Zd`
Operaciones	

Nombre	Nodo
Utilidad	Representación de un nodo del grafo (router).
Atributos	IDENT: ENTERO – Identificador del nodo. PROTOCOLO: ROUTER – Tipo de protocolo que soporta el nodo. POSICION: VECTOR – Posición del nodo en el espacio. NOMBRE: TEXTO – Nombre del nodo. ETIQUETA: TEXTO – Etiqueta que se mostrará si el nodo no tiene nombre. ATR_NODO: ATRIBUTOS – Resto de atributos presentes en el fichero GML y no procesados por nuestra aplicación.
Operaciones	TOGML (NODO): VALOR – Convierte un nodo a su representación en GML. FROMGML (AST): NODO – Extrae un nodo de un árbol sintáctico GML.

Nombre	Arco
Utilidad	Representación de un arco entre dos nodos del grafo.
Atributos	ORIGEN: ENTERO – Nodo inicial del arco. DESTINO: ENTERO – Nodo final del arco. LATENCIA: REAL – Coste por latencia del arco. ANCHOBANDA: REAL – Coste por ancho de banda del arco. METRICA: REAL – Coste definido por el usuario. ATR_ARCO – Resto de atributos presentes en el fichero GML y no procesados por nuestra aplicación.

Nombre	Arco
Operaciones	toGML (ARCO): VALOR – Convierte un arco a su representación en GML. fromGML (AST): ARCO – Extrae un arco de un árbol sintáctico GML.

Nombre	Grafo
Utilidad	Representación en memoria de una topología.
Atributos	NODOS: MAP ENTERO NODO – Array asociativo de nodos, ordenados por identificador. ARCOS: MAP ENTERO [ARCO] – Array asociativo de arcos, ordenados por identificador del nodo origen. DIRIGIDO: BOOLEANO – Indica si el grafo es dirigido o no. ATR_GRAFO: ATRIBUTOS – Resto de atributos presentes en el fichero GML y no procesados por nuestra aplicación.
Operaciones	VACIO: GRAFO – Crea un grafo vacío. INSERTARNODO (GRAFO, NODO): GRAFO – Añade un nodo a un grafo. INSERTARARCO (GRAFO, ARCO): GRAFO – Añade un arco a un grafo. BORRARNODO (GRAFO, ID): GRAFO – Borra el nodo cuyo identificador es *id*. BORRARARCO (GRAFO, ORIGEN, DESTINO): GRAFO – Borra el arco que parte del nodo *origen* y llega a *destino*. BUSCARNODO (GRAFO, VECTOR): NODO – Busca el nodo que se encuentra en la posición *vector*. BUSCARARCO (GRAFO, VECTOR): ARCO – Busca el arco que se encuentra en la posición *vector*. BUSCARRUTAS (GRAFO, ORIGEN, DESTINO, OPCIONES): [[ENTERO]] – Busca todas las rutas existentes entre los nodos *origen* y *destino*. toGML (GRAFO): VALOR – Convierte un grafo a su representación en GML. fromGML (AST): GRAFO – Extrae un grafo de un árbol sintáctico GML.

Nombre	Aplicación
Utilidad	Tipo enumerado que define los diferentes tipos de aplicaciones existentes.

Nombre	Aplicación
Definición	`data Aplicacion = A4 | A6 | Ad`
Operaciones	

Nombre	NodoTerminal
Utilidad	Tipo enumerado que define los diferentes tipos de nodos terminales existentes.
Definición	`data NodoTerminal = N4 | N6 | Nd | Nmap`
Operaciones	

Nombre	Conexión
Utilidad	Tipo enumerado que define los diferentes tipos de operadores de conexión existentes.
Definición	`data Conexion = Directa | OpD | Op4_td | OpD_td | OpD_tp | OpD_tptd`
Operaciones	

Nombre	Red
Utilidad	Representación de la red de distribución de un escenario.

Nombre	Red
Definición	`type Red = [(Conexion, Router)]`
Operaciones	

Nombre	Escenario
Utilidad	Representación de un escenario de transición.
Definición	`type Escenario = (Aplicacion, NodoTerminal, Red, NodoTerminal, Aplicacion)`
Operaciones	

Nombre	TipoCambio
Utilidad	Tipo enumerado que define los diferentes tipos de cambios que se pueden producir al aplicar una regla a un escenario.
Definición	`data TipoCambio = ZZ_manual_4en6 | ZZ_manual_6en4 | ZZ_manual_udp ...`
Operaciones	

Nombre	Cambio
Utilidad	Representación de los cambios resultado de aplicar una regla a un escenario.
Atributos	TIPO: TIPOCAMBIO – Regla aplicada. INICIO: ENTERO – Posición del nodo a partir del cual se empieza a aplicar la regla. ANTES: ENTERO – Número de nodos a los que se aplica la regla. DESPUES: ENTERO – Número de nodos después de aplicar la regla

Nombre	Cambio
Operaciones	MOSTRAR (ESCENARIO, CAMBIO) – Muestra en pantalla el resultado de aplicar un cambio a un escenario.

Nombre	Solución
Utilidad	Representación de una solución resultado del proceso de búsqueda.
Atributos	RUTA: [ENTERO] – Lista de los identificadores de los nodos por los que pasa la ruta. INICIAL: ESCENARIO – Escenario original, antes de aplicar ninguna regla. FINAL: ESCENARIO – Escenario resultante de aplicar las reglas. CAMBIOS: [CAMBIO] – Lista de reglas que se aplican para llegar al escenario final. COSTE_LAT: REAL – Coste de la solución por latencia. COSTE_BW: REAL – Coste de la solución por ancho de banda. COSTE_MET: REAL – Coste de la solución por métrica definida por el usuario.
Operaciones	BUSCARSOLUCIONES (GRAFO, ORIGEN, DESTINO, OPCIONES): [SOLUCION] – Busca todas las soluciones posibles para conectar los nodos *origen* y *destino*. CALCULARCOSTE (SOLUCION): SOLUCION – Evalúa el coste de una solución. ORDENAR_LAT (SOLUCIONES): [SOLUCION] – Ordena una lista de soluciones por latencia. ORDENAR_BW (SOLUCIONES): [SOLUCION] – Ordena una lista de soluciones por ancho de banda. ORDENAR_MET (SOLUCIONES): [SOLUCION] – Ordena una lista de soluciones por métrica. MOSTRAR (SOLUCION) – Muestra una solución en pantalla.

10.6. Diagrama de módulos

El programa se organiza en tres grandes bloques: gestión de topología, aplicación de reglas, e interfaz de usuario. El módulo principal se encargará de llamar a cada bloque según las acciones del usuario. La estructura de módulos resultante es la siguiente:

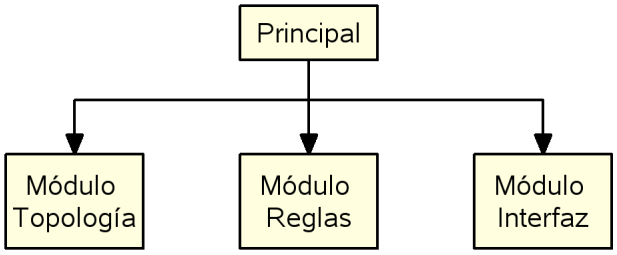

Figura 10.8: Diagrama de módulos principal

- TOPOLOGÍA: es el encargado de cargar y guardar ficheros GML, la edición de las topologías, y el proceso de búsqueda de rutas.
- REGLAS: implementa la base de reglas con los mecanismos de transición, y el motor de búsqueda de soluciones.
- INTERFAZ: se encarga de la interacción con el usuario.

El módulo TOPOLOGÍA se divide en los siguientes submódulos:

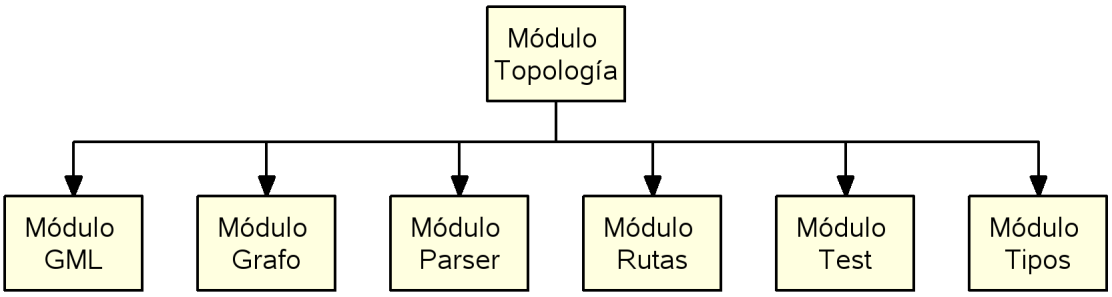

Figura 10.9: Diagrama de módulos de la topología

- GML: conversión de los diferentes tipos de datos hacia y desde GML.
- GRAFO: definición e implementación del tipo de datos *Grafo*, con todas sus operaciones.
- PARSER: conversión de texto a GML, y de GML a texto.
- RUTAS: algoritmos de búsqueda y poda de rutas, para la obtención de escenarios lineales a partir del grafo.
- TEST: batería de pruebas de los diferentes submódulos que componen el módulo TOPOLOGÍA.
- TIPOS: definición de los tipos de datos *Valor*, *Vector*, *Router*, *Nodo* y *Arco*.

El módulo REGLAS se divide en los siguientes submódulos:

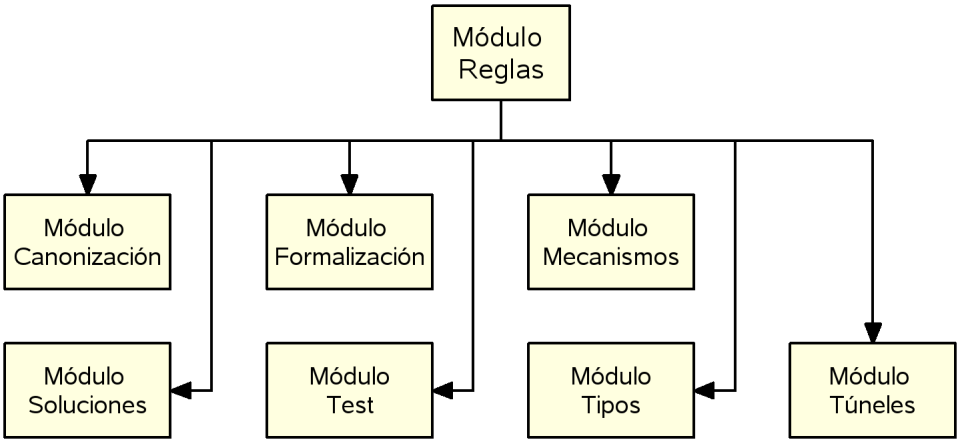

Figura 10.10: Diagrama de módulos de la base de reglas

- CANONIZACIÓN: implementación de las reglas de canonización.
- FORMALIZACIÓN: implementación de las reglas de creación de zonas y establecimiento de los operadores de conexión.
- MECANISMOS: implementación de las reglas de modificación sobre redes finales.
- SOLUCIONES: algoritmo de conversión de rutas a escenarios de transición, búsqueda de soluciones para un escenario, y evaluación y ordenación de soluciones.
- TEST: batería de pruebas de los diferentes submódulos que componen el módulo REGLAS.
- TIPOS: definición de los tipos de datos *Aplicación*, *NodoTerminal*, *Conexión*, *Red*, *Escenario*, *TipoCambio*, *Cambio* y *Solución*.
- TÚNELES: implementación de las reglas de creación de túneles.

El módulo INTERFAZ se divide en los siguientes submódulos:

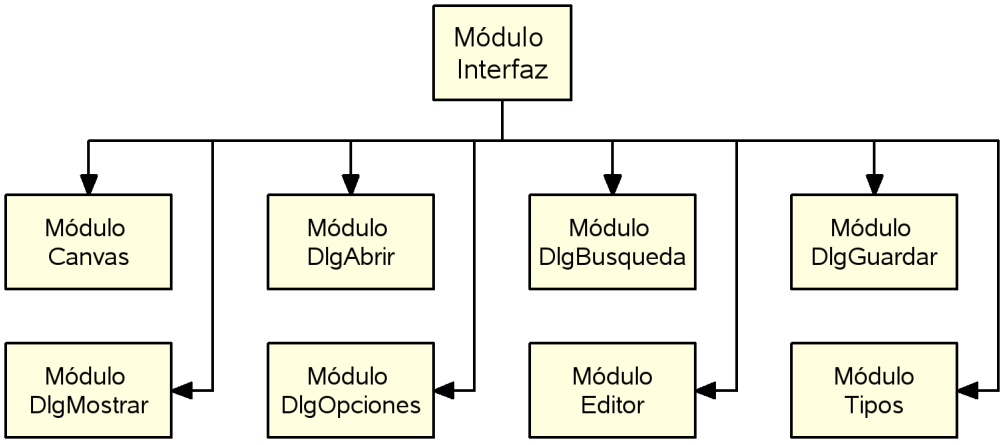

Figura 10.11: Diagrama de módulos de la interfaz

- CANVAS: implementación del área de edición, con las operaciones necesarias para dibujar y manipular el grafo.
- DLGABRIR: diálogo para cargar topologías.
- DLGBÚSQUEDA: diálogo para lanzar el proceso de búsqueda de soluciones.
- DLGGUARDAR: diálogo para guardar topologías.
- DLGMOSTRAR: diálogo de presentación de los detalles de una solución.
- DLGOPCIONES: diálogo de configuración de los costes de los mecanismos de transición.
- EDITOR: implementación de la ventana de edición del programa. Se encarga de llamar al resto de diálogos.
- TIPOS: define los tipos de datos necesarios para guardar el estado de la interfaz.

Parte V

Implementación y pruebas

11 Alternativas de implementación

Teniendo en cuenta las conclusiones alcanzadas en los capítulos anteriores, tenemos que considerar el lenguaje de programación a usar, buscando un equilibrio entre eficiencia y elegancia. Consideraremos cuatro opciones:

- La primera propuesta consiste en usar un lenguaje imperativo tradicional. Como ventajas una mayor familiaridad, mayor número de bibliotecas disponibles, y el hecho de que los compiladores de este tipo suelen estar más optimizados y generan código más eficiente. Como desventajas tenemos que los lenguajes imperativos que conocemos no suelen facilitar el uso de estructuras de datos complejas. Sin embargo, el principal problema es que el uso de este tipo de lenguajes va directamente contra los objetivos del proyecto, por lo que los descartamos sin más.

- Podríamos considerar el uso de un lenguaje lógico como Prolog. La estructura de nuestro proyecto (representación algebraica, no determinismo) se ajusta muy bien a las capacidades de este tipo de lenguajes. La principal limitación es que el rendimiento de las implementaciones que conocemos es muy inferior a otros tipos de lenguaje, y que no suelen ofrecer buenas capacidades para desarrollar aplicaciones enteras, sino que deben integrarse dentro de una aplicación mayor escrita en otro lenguaje.

- Otra posible opción sería utilizar un lenguaje funcional estricto, como OCaml o algún dialecto moderno de Lisp. Las principales ventajas de estos lenguajes es que ofrecen características como estructuras de datos algebraicas y reconocimiento de patrones, que simplifican mucho el código. Además, los compiladores son bastante maduros y generan código muy eficiente. Por contra, carecen de algunas de las características más interesantes del grupo que consideraremos a continuación.

- El último tipo de lenguaje que consideraremos es el de los lenguajes funcionales puros con evaluación perezosa, en especial Haskell. Estos lenguajes poseen la peculiaridad de que los valores no se evalúan hasta que no es completamente necesario, lo que permite tratar estructuras de datos potencialemente infinitas sin problemas. Esto mejora en gran medida la modularidad y permite separar mejor los distintos pasos de un algoritmo. Además, compiladores como GHC generan un código muy eficiente, aunque quizás no tanto como otros lenguajes. La desventaja es que estamos menos acostumbrados a su modelo de desarrollo. No obstante, será la opción que elijamos.

12 Implementación

12.1. Metodología de implementación

En este capítulo describiremos cómo se ha implementado el programa. Muchas veces no está clara cual es la mejor manera de implementar una función. La metodología que se ha seguido, siempre que han surgido dudas, ha sido proponer varias alternativas, evaluar cada una de ellas, y elegir la que parecía mas conveniente. A la hora de elegir una alternativa, se han tenido en cuenta varios parámetros:

- La eficiencia de la solución.
- Su elegancia y simplicidad.
- Su facilidad de integración con la base de código existente.
- Que fuera extensible, para que si en el futuro hay que añadir nuevas funcionalidades, se pueda hacer de manera sencilla.

De esta manera, se ha intentado alcanzar un equilibrio óptimo entre los diferentes criterios. En las siguientes secciones se expondrán las situaciones que se han ido encontrando durante la implementación de los diferentes módulos vistos en la sección 10.6.

12.2. Módulo Topologías

Para cargar los ficheros de grafos se pueden usar tres opciones:

- La más directa, y probablemente la que mayor rendimiento proporcione, es escribir el cargador manualmente. También es la más compleja, por lo que se descarta.
- Otra opción es usar Parsec[1], una biblioteca de combinadores que permite escribir analizadores sintácticos dentro del propio código en Haskell. Es muy flexible y potente, y además tiene la ventaja de que no requiere herramientas externas, pero tiene dos desventajas: al combinar el análisis léxico y sintáctico en una sola pasada se puede complicar la gramática, y la compilación del código resultante es bastante lenta.
- La tercera opción es usar Alex[2] y Happy[3], dos herramientas similares a Lex y Yacc.

[1] http://legacy.cs.uu.nl/daan/download/parsec/parsec.html
[2] http://www.haskell.org/alex/
[3] http://www.haskell.org/happy/

Puesto que la gramática de GML es muy sencilla, y no interesa complicar el código más de lo necesario, se ha elegido usar Parsec. De esta manera se puede escribir el analizador sintático en un solo fichero de código autocontenido. También se ha implementado una función recíproca, que toma un árbol sintáctico y lo convierte a texto.

Para completar el proceso de carga, se ha definido la clase de tipos GML. Esta clase define dos funciones: *toGML*, para convertir el tipo dado a un árbol sintáctico, y *fromGML*, para convertir del árbol al tipo dado. Se han definido como pertenecientes a esta clase los tipos *Grafo*, *Nodo*, *Arco* y *Vector*.

Una duda que surgió era como almacenar los grafos en memoria. Los nodos están indexados por el campo identificador, que debe ser único. Por tanto, se ha usado la estructura de datos *Map*, que implementa un array asociativo, utilizando como clave el identificador. Para representar los arcos las dos opciones más comunes son usar una matriz de adyacencia, o asignar una lista de nodos adyacentes a cada nodo del grafo. Las matrices son más eficientes cuando el grado de conectividad es muy alto, siendo mejor usar listas en caso contrario. Para determinar cual era la solución óptima para nuestro problema, se hizo un estudio estadístico de varias topologías disponibles en Internet, obteniendo los resultados mostrados en la figura 12.1. El significado de cada campo es el siguiente:

- Fichero: topología analizada.
- Nodos: número total de nodos en la topología.
- Enlaces: número total de enlaces, sumando los enlaces de todos los nodos.
- Conectividad absoluta: número de arcos que salen de cada nodo.
- Conectividad relativa: porcentaje de nodos a los que se conecta cada nodo.
- Densidad: siendo A el número total de arcos, y N el número total de nodos, la densidad de un grafo no dirigido se define como

$$D = \frac{2\,|A|}{|V|\,(|V|-1)}$$

  Su valor mínimo es 0 para grafos inconexos, y 1 para grafos completamente conexos.

Observamos que el grado de conectividad general es bajo. La conectividad media es de 5 enlaces por nodos. Hay algún nodo con una conectividad muy alta, pero suelen ser únicamente los routers que forman el núcleo de la red. Si usáramos una matriz, la densidad representaría el número de celdas que contendrían algún valor. Vemos que el valor más alto es del 18 %, siendo bastante bajos en el resto de casos.

Fichero	Nodos	Enlaces	Conectividad absoluta			Conectividad relativa			Densidad
			Mínima	Media	Máxima	Mínima	Media	Máxima	
abilene.gml	11	14	2	2,55	3	18,18 %	23,14 %	27,27 %	0,1209
abovenet.6461.r0.cch.gml	368	966	0	5,25	39	0,00 %	1,43 %	10,60 %	0,0008
att.7018.r0.cch.gml	731	2253	0	6,16	55	0,00 %	0,84 %	7,52 %	0,0003
att.gml	154	188	1	2,44	29	0,65 %	1,59 %	18,83 %	0,0088
colt.gml	43	59	1	2,74	8	2,33 %	6,38 %	18,61 %	0,0251
cw.gml	33	107	2	6,49	13	6,06 %	19,65 %	39,39 %	0,0058
dfn.gml	30	97	1	6,47	20	3,33 %	21,56 %	66,67 %	0,0064
ebone.1755.r0.cch.gml	161	307	0	3,81	17	0,00 %	2,37 %	10,56 %	0,0034
exodus.3967.r0.cch.gml	246	540	0	4,39	18	0,00 %	1,79 %	7,32 %	0,0017
geant-20041125.gml	23	37	2	3,22	6	8,70 %	13,99 %	26,09 %	0,0345
geant.gml	23	37	2	3,22	6	8,70 %	13,99 %	26,09 %	0,0345
level3.3356.r0.cch.gml	625	5298	0	16,95	169	0,00 %	2,71 %	27,04 %	0,0000
sprint.1239.r0.cch.gml	549	1593	0	5,8	51	0,00 %	1,06 %	9,29 %	0,0004
switch.gml	31	42	1	2,71	10	3,23 %	8,74 %	32,26 %	0,0360
telekom.gml	10	17	2	3,4	7	20,00 %	34,00 %	70,00 %	0,0735
tiscali.3257.r0.cch.gml	248	405	0	3,27	31	0,00 %	1,32 %	12,50 %	0,0030
verrio.2914.r0.cch.gml	911	2217	0	4,87	46	0,00 %	0,53 %	5,05 %	0,0004
vsnl.4755.r0.cch.gml	12	12	0	2	4	0,00 %	16,67 %	33,33 %	0,1818

Figura 12.1: Estadísticas de conectividad de varias topologías de ejemplo

A partir de los resultados anteriores se deduce que es mejor usar listas de adyacencia. Los arcos se indexan por el nodo de origen, guardando una lista de arcos para cada nodo. La estructura de datos resultante es la siguiente:

```
data Grafo = Grafo {
    nodos :: Map Int Nodo,
    arcos :: Map Int [Arco],
    dirigido :: Bool,
    atr_grafo :: Atributos }
```

El último aspecto a considerar en cuanto a las topologías es el proceso de búsqueda de rutas. Nuestra aplicación necesita considerar todas las rutas posibles, por lo que usaremos una recorrido en profundidad del grafo. Sin embargo, hay que tener en cuenta que una topología con varias decenas de nodos puede generar cientos de miles de rutas. Interesa por tanto implementar el algoritmo de tal manera que resulte fácil añadir criterios de poda, con el fin de reducir el número de resultados si el usuario lo considera oportuno.

El problema se ha modelado de la siguiente manera: partiendo del nodo inicial de la ruta, se desenrolla el grafo, convirtiéndolo en un árbol. A continuación, se poda el árbol, eliminando las rutas innecesarias. Por último, se hace un recorrido en profundidad del árbol, descartando todas aquellas rutas que llegan a un nodo que no es el deseado. Como Haskell utiliza evaluación perezosa, si una rama se poda nunca se llegarán a evaluar sus hijos. De esta manera podemos encadenar varios criterios de poda de manera sencilla.

Por ejemplo, dado el siguiente grafo, si quisieramos buscar todas las rutas entre el nodo 1 y el nodo 4, obtendríamos el siguiente árbol:

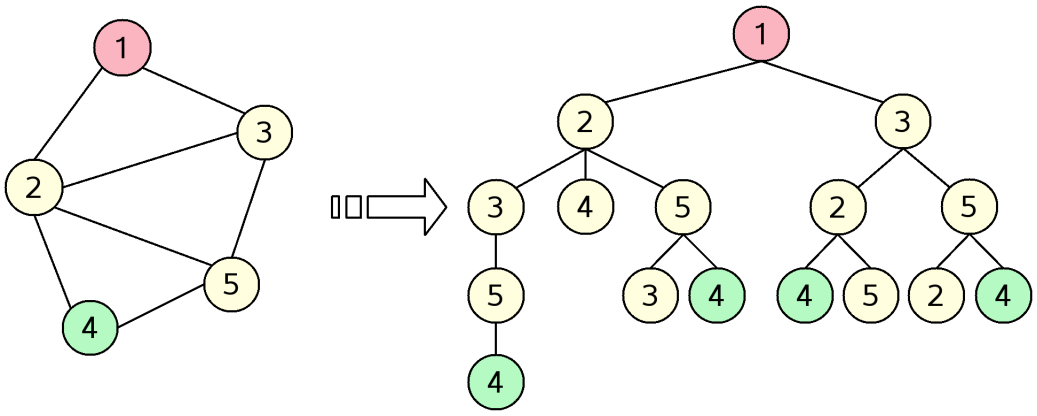

Figura 12.2: Conversión de grafo a árbol

Si la topología es grande, el número de rutas entre dos nodos, y por tanto el tiempo de procesamiento, puede dispararse. Para evitar que esto ocurra se puede hacer una poda de las rutas. Así reduciremos considerablemente el número de soluciones a considerar, a costa de perder algunas soluciones que podrían ser válidas. Se dejará que sea el usuario el que elija los criterios a usar.

Se han definido dos criterios de poda. El primero es un criterio por longitud. Dado un umbral k, si la longitud de la ruta más corta es L_{min}, este criterio eliminaría todas aquellas rutas cuya longitud fuera $L_{ruta} > L_{min} + k$. Su implementación es muy sencilla. En primer lugar, se hace un recorrido en anchura del árbol, buscando la posición del nodo de destino con menor profundidad. A continuación, se podan todas aquellas ramas que estén a una profundidad mayor que el mínimo mas el umbral.

El segundo criterio es mucho más agresivo. Lo que hace es que si dos nodos A y B son adyacentes, se descartan todas aquellas rutas que conecten A y B a través de uno o mas nodos intermedios. Su implementación también es sencilla. Se recorre el árbol en profundidad, manteniendo un conjunto con los nodos adyacentes a los visitados, y se podan todas aquellas ramas que se dirijan a uno de los nodos del conjunto.

Se ha implementado un algoritmo de preproceso opcional. Este algoritmo toma un grafo no dirigido, y lo transforma en un grafo dirigido, intentando eliminar los bucles existentes. De esta manera se fuerza a que todos los enlaces alejen la ruta del nodo origen y la acercen al destino. Por ejemplo, el resultado de aplicar el algoritmo al siguiente grafo, tomando como origen el nodo 1 y como destino el 6, sería el siguiente:

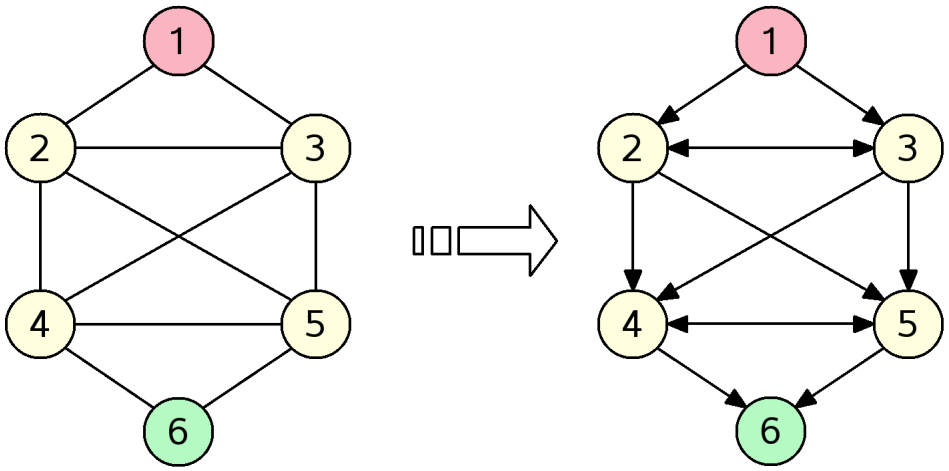

Figura 12.3: Ejemplo de eliminación de bucles en grafos

El algoritmo funciona clasificando los nodos por distancia al origen, y eliminando los enlaces que llevan de un nodo a distancia d_1 a otro a distancia $d_2 < d_1$. Para evitar que surjan conflictos cuando un camino se bifurca en varias subcaminos de longitudes diferentes que luego vuelven a converger, se recorre el grafo simultáneamente desde el destino. El esquema resultante, en pseudocódigo, es el siguiente:

```
eliminar_bucles (grafo, origen, destino):
    F = {origen}
    B = {destino}
    visitados = {}
    acc = borrar_arcos (grafo)
    mientras (F != vacio) y (B != vacio):
        a = {arcos que salen de F} - visitados
        b = {arcos que llegan a B} - visitados
        acc = insertar_arcos (acc, a)
        acc = insertar_arcos (acc, b)
        tmp = visitados + F + B
        F = adyacentes(F) - visitados
        B = adyacentes(B) - visitados
        visitados = tmp
    devolver acc
```

Figura 12.4: Algoritmo de eliminación de bucles en grafos

12.3. Módulo Reglas

Como ya vimos, la resolución de escenarios se compone de una serie de pasos. Un lenguaje con evaluación estricta obliga a complicar el código, fusionando los pasos, o se corre el riesgo de agotar la memoria. Sin embargo, el esquema de evaluación perezosa usado por Haskell resultará muy útil, ya que podemos implementar el algoritmo de manera directa, sabiendo que si una rama del árbol no se visita, nunca se llegará a evaluar

Los escenarios se representan como una tupla de cinco elementos:

```
type Escenario = (Aplicacion,
                  NodoTerminal,
                  Red,
                  NodoTerminal,
                  Aplicacion)
```

Donde *Aplicacion* y *NodoTerminal* son tipos enumerados con los posibles valores de los conjuntos $\mathcal{A}$ y $\mathcal{N}$, respectivamente. A su vez, *Red* es una lista de tuplas, que define los routers por los que pasa la conexión, y los operadores con los que se conectan:

```
type Red = [(Operador, Router)]
```

Donde *Router* es un tipo enumerado con los posibles valores del conjunto $\mathcal{R}$. El campo *Operador* es un tipo enumerado con los posibles valores del conjunto $\{\leftrightarrow, \otimes_4, \otimes_d, \odot_d, \boxtimes_d, \boxtimes_4\}$, y representa el operador con el que el router se conecta a su predecesor. Se usa esta representación porque es el formato que usan la mayoría de las reglas. El primer operador de una red no se usa para nada, pero por convención será siempre $\leftrightarrow$.

La búsqueda de soluciones funciona generando un árbol de búsqueda, podándolo, y recorriendo las hojas para recolectar los resultados. A partir de un escenario inicial, se le

aplican todas las reglas posibles, lo que da como resultado una lista de nuevos escenarios. Estos escenarios se procesan de manera recursiva, hasta que ya no quedan más reglas por aplicar. Por tanto, tendremos un árbol *n-ario*, donde los hijos de un rama se representarán con una lista. Las hojas del árbol serán nodos con la lista vacía. Además, necesitamos una manera de almacenar los cambios que se producen como resultado de la aplicación de las reglas. Se han considerado cuatro opciones:

- La primera opción sería guardar en cada nodo los cambios que se han aplicado para llegar del nodo padre al actual. Para obtener las soluciones habría que recorrer el árbol acumulando los cambios en una lista. La estructura resultante sería:

  ```
  data Arbol = Arbol Escenario Cambio [Arbol]
  ```

- La estructura anterior tiene un defecto, y es que el nodo raíz no tiene padre, y por tanto tampoco cambios. Esto nos obligaría a introducir algún tipo de cambio nulo. Para evitar este problema podemos registrar los cambios que se producen para llegar del nodo actual a los nodos hijos:

  ```
  data Arbol = Arbol Escenario [(Cambio, Arbol)]
  ```

- La tercera opción es guardar en cada nodo la lista completa de cambios. La longitud de esta lista sería proporcional al nivel del nodo en el árbol, siendo 0 para el nodo raíz. Los cambios se irían añadiendo al final de la lista.

  ```
  data Arbol = Arbol Escenario [Cambio] [Arbol]
  ```

- La opción anterior tiene un gran defecto, y es que almacena mucha información redundante. Por ejemplo, si un nodo tiene una lista de cambios $[c_1, c_2, c_3 \dots c_n]$, sus hijos tendrían una lista $[c_1, c_2, c_3 \dots c_n, c_{n+1}]$. Es decir, para cada nodo se repiten n elementos, siendo n la profundidad del nodo en el árbol. Por tanto, el espacio desperdiciado crece exponencialmente. Sin embargo, podemos aprovechar una característica de los lenguajes funcionales. Éstos suelen definir la listas de manera inductiva, lo que permite que varias listas compartan la misma cola. Si guardamos la lista de cambios en orden inverso, es decir, $[c_n \dots c_3, c_2, c_1]$, los nodos hijos tendrían una lista $[c_{n+1}, c_n \dots c_3, c_2, c_1]$ y podrían compartir la lista de cambios del padre. La estructura resultante es la misma que en el caso anterior, sólo cambia la forma en la que se registran los cambios.

  ```
  data Arbol = Arbol Escenario [Cambio] [Arbol]
  ```

Cada una de las soluciones anteriores tiene una serie de ventajas y desventajas. Las dos primeras son las más compactas, y también las más sencilla en cuanto a la generación del árbol, pero a costa de complicar el recorrido. Las dos últimas son el caso opuesto. La opción más equilibrada es la última, por lo que es la que hemos elegido. La estructura de datos resultante tiene el siguiente aspecto:

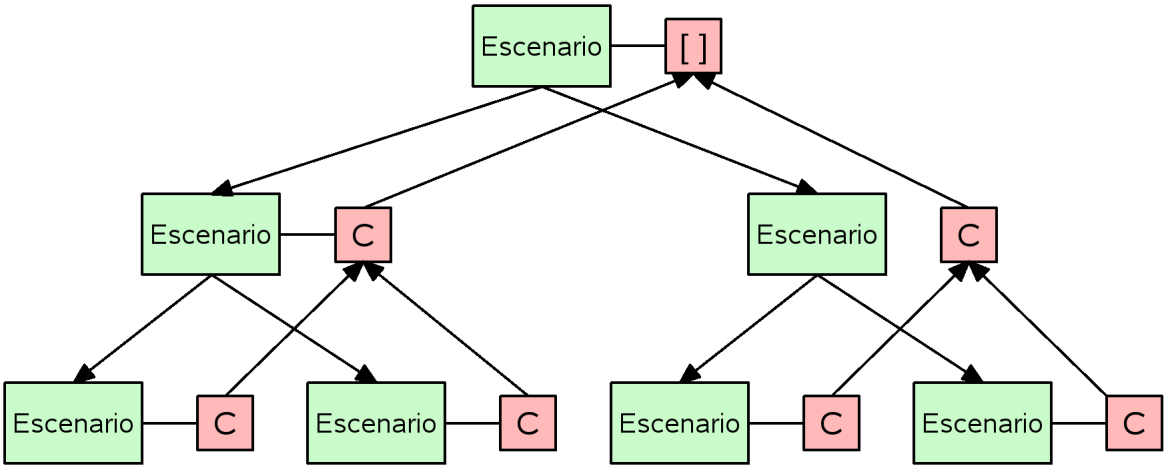

Figura 12.5: Estructura del árbol de búsqueda

Un problema que surge a la hora de implementar la base de reglas es que algunas son de longitud variable. Para resolverlo, hemos usado HaRP[4], una herramienta que permite usar expresiones regulares en patrones de listas. HaRP lee un fichero en Haskell con anotaciones especiales, y lo traduce a Haskell puro. El resultado es una combinación muy potente de expresividad y concisión. Su única desventaja es que utiliza una extensión del lenguaje llamada *pattern guards* que no todos los compiladores soportan.

Los patrones aparecen encerrados entre los símbolos $[/ \ldots /]$. Se pueden usar los operadores habituales en las expresiones regulares, como $\star$ para hacer concidir cero o más elementos, $+$ para uno o más elementos, ? para un elemento opcional, y $|$ para una alternativa entre dos elementos. Por ejemplo, la primera regla de canonización $Z_4\,(\otimes_d Z_4)\star = Z_4$, se implementaría así:

```
canonizacion [/ (conexion,Z4) , (OpD, Z4)+! , xs@_* /]
    = (conexion,Z4) : xs
```

Otro detalle a considerar es que algunas reglas no tienen en cuenta la posibilidad de tratar zonas del tipo Z_d. Esto podría provocar que no se pudiera aplicar ninguna regla, y por tanto resolver el escenario, aún cuando éste tuviera conectividad. Sin embargo, podemos observar que una zona dual puede comportarse tanto como una zona IPv4 o como una zona IPv6. Por tanto, si una regla encuentra un escenario con una zona dual y no es capaz de procesarlo, lo trata como si fueran dos escenarios, sustituyendo Z_d por Z_4 y Z_6.

El último paso de la búsqueda es la evaluación de costes. Para cada solución, se calcula por separado el coste de la propia red y el de los mecanismos de transición aplicados, y se suman. Al calcular el coste de los mecanismos sólo se tiene en cuenta el tipo de estos, pero no la longitud de los túneles. La razón de esto es que, al crear un túnel, los únicos nodos que están al tanto son los extremos, siendo su existencia transparente al resto de la red.

A continuación mostraremos el código relativo a la implementación de la base de reglas. Esto nos dará una idea de la validez de Haskell para este tipo de problemas, ya que,

[4]http://www.cs.chalmers.se/~d00nibro/harp/

gracias al uso de *pattern matching*, la representación obtenida es muy compacta.

12.3.1. Reglas de formalización

```
-- Operador de conexion dual
op1 :: ReglaP
op1 [/ (conexion,zi) , routers@(Directa,Zd)+! , (Directa,zj) , xs@_* /] = if
        subset [zi,zj] [Z4,Z6]
        then Just (
                (conexion,zi) : (OpD,zj) : xs,
                Cambio Operador1 1 (2 + length routers) 2)
        else Nothing
op1 _ = Nothing

-- Operador de conexion IPv4 con traducción de direcciones IPv4
op2 :: ReglaP
op2 [/ (conexion,zi) , (Directa,R4_td) , (Directa,zj) , xs@_* /] = if
        (subset [zi,zj] [Z4,Z6,Zd])
        then Just (
                (conexion,zi) : (Op4_td,zj) : xs,
                Cambio Operador2 1 3 2)
        else Nothing
op2 _ = Nothing

-- Operador de conexion dual con traducción de direcciones IPv4
op3 :: ReglaP
op3 [/ (conexion,zi) , r1@(Directa,Zd)*! , (Directa,Rd_td) , r2@(Directa,Zd)*!
    , (Directa,zj) , xs@_* /] = if
        --(subset [zi,zj] [Z4,Z6])
        (subset [zi,zj] [Z4,Z6,Zd])
        then Just (
                (conexion,zi) : (OpD_td,zj) : xs,
                Cambio Operador3 1 (3 + length r1 + length r2) 2)
        else Nothing
op3 _ = Nothing

-- Operador de conexion dual con traducción de protocolos IPv4 e IPv6
op4 :: ReglaP
op4 [/ (conexion,zi) , r1@(Directa,Zd)*! , (Directa,Rd_tp) , r2@(Directa,Zd)*!
    , (Directa,zj) , xs@_* /] = if
        --(subset [zi,zj] [Z4,Z6,Zd]) &&
        (subset [zi,zj] [Z4,Z6]) &&
        (zi /= zj)
        then Just (
                (conexion,zi) : (OpD_tp,zj) : xs,
                Cambio Operador4 1 (3 + length r1 + length r2) 2)
        else Nothing
op4 _ = Nothing

-- Operador de conexion dual con traducción de direcciones IPv4 y traducción
```

```
-- de protocolos IPv4 e IPv6
op5 :: ReglaP
op5 [/ (conexion,zi) , r1@(Directa,Zd)*! , (Directa,Rd_tptd) , r2@(Directa,Zd)*!
    , (Directa,zj) , xs@_* /] = if
        --(subset [zi,zj] [Z4,Z6,Zd]) &&
        (subset [zi,zj] [Z4,Z6]) &&
        (zi /= zj)
        then Just (
                (conexion,zi) : (OpD_tptd,zj) : xs,
                Cambio Operador5 1 (3 + length r1 + length r2) 2)
        else Nothing
op5 _ = Nothing
```

12.3.2. Reglas de canonización

```
-- Conexión de zonas IPv4 utilizando (X)d
can1 :: ReglaP
can1 [/ (conexion,Z4) , r@(OpD, Z4)+! , rs@_* /] = Just (
        (conexion,Z4) : rs,
        Cambio Canonizacion1 1 (1 + length r) 1)
can1 [/ (conexion,Z4) , r@(OpD, Zd)+! , rs@_* /] = Just (
        (conexion,Z4) : rs,
        Cambio Canonizacion1 1 (1 + length r) 1)
can1 [/ (conexion,Zd) , r@(OpD, Z4)+! , rs@_* /] = Just (
        (conexion,Z4) : rs,
        Cambio Canonizacion1 1 (1 + length r) 1)
can1 _ = Nothing

-- Conexión de zonas IPv4 utilizando (·)d
can2 :: ReglaP
can2 ((conexion,Z4) : (OpD_td,Z4) : rs) = Just (
        (conexion,Z4) : (Op4_td,Z4) : rs,
        Cambio Canonizacion2 1 2 2)
can2 ((conexion,Z4) : (OpD_td,Zd) : rs) = Just (
        (conexion,Z4) : (Op4_td,Z4) : rs,
        Cambio Canonizacion2 1 2 2)
can2 ((conexion,Zd) : (OpD_td,Zd) : rs) = Just (
        (conexion,Z4) : (Op4_td,Z4) : rs,
        Cambio Canonizacion2 1 2 2)
can2 _ = Nothing

-- Conexión de zonas IPv4 utilizando |·|d
can3 :: ReglaP
can3 ((conexion,Z4) : (OpD_tptd,Z4) : rs) = Just (
        (conexion,Z4) : (Op4_td,Z4) : rs,
        Cambio Canonizacion3 1 2 2)
can3 ((conexion,Z4) : (OpD_tptd,Zd) : rs) = Just (
        (conexion,Z4) : (Op4_td,Z4) : rs,
        Cambio Canonizacion3 1 2 2)
can3 ((conexion,Zd) : (OpD_tptd,Z4) : rs) = Just (
        (conexion,Z4) : (Op4_td,Z4) : rs,
```

```
        Cambio Canonizacion3 1 2 2)
can3 _ = Nothing

-- Conexión de zonas IPv6 utilizando (X)d
can4 :: ReglaP
can4 [/ (conexion,Z6), r@(OpD,Z6)+!, rs@_* /] = Just (
        (conexion,Z6) : rs,
        Cambio Canonizacion4 1 (1 + length r) 1)
can4 [/ (conexion,Z6), r@(OpD,Zd)+!, rs@_* /] = Just (
        (conexion,Z6) : rs,
        Cambio Canonizacion4 1 (1 + length r) 1)
can4 [/ (conexion,Zd), r@(OpD,Z6)+!, rs@_* /] = Just (
        (conexion,Z6) : rs,
        Cambio Canonizacion4 1 (1 + length r) 1)
can4 _ = Nothing

-- Conexión de zonas IPv6 utilizando |X|d
can5 :: ReglaP
can5 [/ (conexion,Z6), r@(OpD_tp,Z6)+!, rs@_* /] = Just (
        (conexion,Z6) : rs,
        Cambio Canonizacion5 1 (1 + length r) 1)
can5 [/ (conexion,Z6), r@(OpD_tp,Zd)+!, rs@_* /] = Just (
        (conexion,Z6) : rs,
        Cambio Canonizacion5 1 (1 + length r) 1)
can5 [/ (conexion,Zd), r@(OpD_tp,Z6)+!, rs@_* /] = Just (
        (conexion,Z6) : rs,
        Cambio Canonizacion5 1 (1 + length r) 1)
can5 _ = Nothing

-- Conexión de zonas IPv6 utilizando (·)d ó |·|d
can6 :: ReglaP
can6 [/ (conexion,Z6), r@(OpD_td,Z6)+!, rs@_* /] = Just (
        (conexion,Z6) : rs,
        Cambio Canonizacion6 1 (1 + length r) 1)
can6 [/ (conexion,Z6), r@(OpD_td,Zd)+!, rs@_* /] = Just (
        (conexion,Z6) : rs,
        Cambio Canonizacion6 1 (1 + length r) 1)
can6 [/ (conexion,Zd), r@(OpD_td,Z6)+!, rs@_* /] = Just (
        (conexion,Z6) : rs,
        Cambio Canonizacion6 1 (1 + length r) 1)
can6 [/ (conexion,Z6), r@(OpD_tptd,Z6)+!, rs@_* /] = Just (
        (conexion,Z6) : rs,
        Cambio Canonizacion6 1 (1 + length r) 1)
can6 [/ (conexion,Z6), r@(OpD_tptd,Zd)+!, rs@_* /] = Just (
        (conexion,Z6) : rs,
        Cambio Canonizacion6 1 (1 + length r) 1)
can6 [/ (conexion,Zd), r@(OpD_tptd,Z6)+!, rs@_* /] = Just (
        (conexion,Z6) : rs,
        Cambio Canonizacion6 1 (1 + length r) 1)
can6 _ = Nothing
```

```
87
-- Conexión de zonas duales utilizando |X|d
can7 :: ReglaP
can7 [/ (conexion,Zd), r@(OpD,Zd)+! , rs@_* /] = Just (
        (conexion,Zd):rs,
92      Cambio Canonizacion7 1 (1 + length r) 1)
can7 _ = Nothing

-- Conexión de una zona dual en una posición intermedia utilizando |X|d
can8 :: ReglaP
97 can8 [/ (conexion,zi) , routers@(OpD,Zd)+! , (OpD,zj) , rs@_* /] = if
        (subset [zi,zj] [Z4,Z6,Zd])
        then Just (
                (conexion,zi) : (OpD,zj) : rs,
                Cambio Canonizacion8 1 (2 + length routers) 2)
102     else Nothing
can8 _ = Nothing
```

12.3.3. Reglas de creación de túneles

```
-------------------------------------------------------------------------------
2  -- Túneles entre zonas intermedias
-------------------------------------------------------------------------------

-- Túnel manual IPv4 dentro de IPv6
zz_manual_4en6 :: ReglaP
7  zz_manual_4en6 [(Directa,Z4), (op1,Z6), (op2,Z4)]
        | subset [op1,op2] [OpD, OpD_tp] = Just (
                [(Directa,Z4)],
                Cambio ZZ_manual_4en6 1 3 1)
        | (elem op1 [OpD, OpD_tp]) && (elem op2 [OpD_td, OpD_tptd]) = Just (
12              [(Directa,Z4), (Op4_td,Z4)],
                Cambio ZZ_manual_4en6 1 3 2)
        | subset [op1,op2] [OpD_td, OpD_tptd] = Just (
                [(Directa,Z4), (Op4_td,Z4), (Op4_td,Z4)],
                Cambio ZZ_manual_4en6 1 3 3)
17      | otherwise = Nothing
zz_manual_4en6 _ = Nothing

-- Túnel manual IPv6 dentro de IPv4
zz_manual_6en4 :: ReglaP
22 zz_manual_6en4 [(Directa,Z6), (op1,Z4), (op2,Z6)]
        | subset [op1,op2] [OpD, OpD_tp, OpD_td, OpD_tptd] = Just (
                [(Directa,Z6)],
                Cambio ZZ_manual_6en4 1 3 1)
        | otherwise = Nothing
27 zz_manual_6en4 _ = Nothing

-- Túnel manual IPv6 dentro de IPv4 UDP
zz_manual_udp :: ReglaP
zz_manual_udp [/ (Directa,Z6), (op1,Z4), r@(Op4_td,Z4)*! , (op2,Z6) /]
```

```
        | subset [op1,op2] [OpD, OpD_tp, OpD_td, OpD_tptd] = Just (
                [(Directa,Z6)],
                Cambio ZZ_manual_udp 1 (3 + length r) 1)
        | otherwise = Nothing
zz_manual_udp _ = Nothing

-- Broker de túneles IPv4 dentro de IPv6
zz_broker_4en6 :: ReglaP
zz_broker_4en6 [(Directa,Z4), (op1,Z6), (op2,Z4)]
        | subset [op1,op2] [OpD, OpD_tp] = Just (
                [(Directa,Z4)],
                Cambio ZZ_broker_4en6 1 3 1)
        | (elem op1 [OpD, OpD_tp]) && (elem op2 [OpD_td, OpD_tptd]) = Just (
                [(Directa,Z4), (Op4_td,Z4)],
                Cambio ZZ_broker_4en6 0 3 2)
        | subset [op1,op2] [OpD_td, OpD_tptd] = Just (
                [(Directa,Z4), (Op4_td,Z4), (Op4_td,Z4)],
                Cambio ZZ_broker_4en6 1 3 3)
        | otherwise = Nothing
zz_broker_4en6 _ = Nothing

-- Broker de túneles IPv6 dentro de IPv4
zz_broker_6en4 :: ReglaP
zz_broker_6en4 [(Directa,Z6), (op1,Z4), (op2,Z6)]
        | subset [op1,op2] [OpD, OpD_tp, OpD_td, OpD_tptd] = Just (
                [(Directa,Z6)],
                Cambio ZZ_broker_6en4 1 3 1)
        | otherwise = Nothing
zz_broker_6en4 _ = Nothing

-- Broker de túneles IPv6 dentro de IPv4 UDP
zz_broker_udp :: ReglaP
zz_broker_udp [/ (Directa,Z6), (op1,Z4), r@(Op4_td,Z4)*! , (op2,Z6) /]
        | subset [op1,op2] [OpD, OpD_tp, OpD_td, OpD_tptd] = Just (
                [(Directa,Z6)],
                Cambio ZZ_broker_udp 1 (3 + length r) 1)
        | otherwise = Nothing
zz_broker_udp _ = Nothing

-- Túnel 6to4
zz_6to4 :: ReglaP
zz_6to4 [(Directa,Z6), (op1,Z4), (op2,Z6)]
        | subset [op1,op2] [OpD, OpD_tp, OpD_td, OpD_tptd] = Just (
                [(Directa,Z6)],
                Cambio ZZ_6to4 1 3 1)
        | otherwise = Nothing
zz_6to4 _ = Nothing


--------------------------------------------------------------------------------
```

```
-- Túneles entre zonas y nodos terminales
--------------------------------------------------------------------------------

-- Túnel manual IPv6 dentro de IPv4
zn_manual_4en6 :: ReglaC
zn_manual_4en6 (a1, Nd, ((Directa,Z6):(op1,Z4):xs), n2, a2)
        | elem op1 [OpD, OpD_tp] = Just (
                (a1, Nd, ((Directa,Z4):xs), n2, a2),
                Cambio ZN_manual_4en6 0 3 2)
        | elem op1 [OpD_td, OpD_tptd] = Just (
                (a1, Nd, ((Directa,Z4):(Op4_td,Z4):xs), n2, a2),
                Cambio ZN_manual_4en6 0 2 1)
        | otherwise = Nothing
zn_manual_4en6 _ = Nothing

-- Túnel manual IPv6 dentro de IPv4
zn_manual_6en4 :: ReglaC
zn_manual_6en4 (a1, Nd, ((Directa,Z4):(op1,Z6):xs), n2, a2)
        | elem op1 [OpD, OpD_tp, OpD_td, OpD_tptd] = Just (
                (a1, Nd, ((Directa,Z6):xs), n2, a2),
                Cambio ZN_manual_6en4 0 3 2)
        | otherwise = Nothing
zn_manual_6en4 _ = Nothing

-- Túnel manual IPv6 dentro de IPv4 UDP
zn_manual_udp :: ReglaC
zn_manual_udp (a1, Nd, [/ (Directa,Z4), r@(Op4_td,Z4)*! , (op1,Z6) , xs@_* /],
              n2, a2)
        | elem op1 [OpD, OpD_tp, OpD_td, OpD_tptd] = Just (
                (a1, Nd, ((Directa,Z6):xs), n2, a2),
                Cambio ZN_manual_udp 0 (3 + length r) 2)
        | otherwise = Nothing
zn_manual_udp _ = Nothing

-- Broker de túneles IPv6 dentro de IPv4
zn_broker_4en6 :: ReglaC
zn_broker_4en6 (a1, Nd, ((Directa,Z6):(op1,Z4):xs), n2, a2)
        | elem op1 [OpD, OpD_tp] = Just (
                (a1, Nd, ((Directa,Z4):xs), n2, a2),
                Cambio ZN_broker_4en6 0 3 2)
        | elem op1 [OpD_td, OpD_tptd] = Just (
                (a1, Nd, ((Directa,Z4):(Op4_td,Z4):xs), n2, a2),
                Cambio ZN_broker_4en6 0 2 1)
        | otherwise = Nothing
zn_broker_4en6 _ = Nothing

-- Broker de túneles IPv6 dentro de IPv4
zn_broker_6en4 :: ReglaC
zn_broker_6en4 (a1, Nd, ((Directa,Z4):(op1,Z6):xs), n2, a2)
        | elem op1 [OpD, OpD_tp, OpD_td, OpD_tptd] = Just (
```

```
                (a1, Nd, ((Directa,Z6):xs), n2, a2),
                Cambio ZN_broker_6en4 0 3 2)
        | otherwise = Nothing
zn_broker_6en4 _ = Nothing

-- Broker de túneles IPv6 dentro de IPv4 UDP
zn_broker_udp :: ReglaC
zn_broker_udp (a1, Nd, [/ (Directa,Z4), r@(Op4_td,Z4)*! , (op1,Z6) , xs@_* /],
               n2, a2)
        | elem op1 [OpD, OpD_tp, OpD_td, OpD_tptd] = Just (
                (a1, Nd, ((Directa,Z6):xs), n2, a2),
                Cambio ZN_broker_udp 0 (3 + length r) 2)
        | otherwise = Nothing
zn_broker_udp _ = Nothing

-- Túnel DSTM
zn_dstm :: ReglaC
zn_dstm (a1, Nd, ((Directa,Z6):(op1,Z4):xs), n2, a2)
        | elem op1 [OpD, OpD_tp] = Just (
                (a1, Nd, ((Directa,Z4):xs), n2, a2),
                Cambio ZN_dstm 0 3 2)
        | elem op1 [OpD_td, OpD_tptd] = Just (
                (a1, Nd, ((Directa,Z4):(op1,Z4):xs), n2, a2),
                Cambio ZN_dstm 0 3 3)
        | otherwise = Nothing
zn_dstm _ = Nothing

-- Túnel Teredo
zn_teredo :: ReglaC
zn_teredo (a1, Nd, [/ (Directa,Z4), r@(Op4_td,Z4)*! , (op1,Z6) , xs@_* /],
          n2, a2)
        | elem op1 [OpD, OpD_tp, OpD_td, OpD_tptd] = Just (
                (a1, Nd, ((Directa,Z6):xs), n2, a2),
                Cambio ZN_teredo 0 (3 + length r) 2)
        | otherwise = Nothing
zn_teredo _ = Nothing

-- Túnel ISATAP
zn_isatap :: ReglaC
zn_isatap (a1, Nd, ((Directa,Z4):(op1,Z6):xs), n2, a2)
        | elem op1 [OpD, OpD_tp] = Just (
                (a1, Nd, ((Directa,Z6):xs), n2, a2),
                Cambio ZN_isatap 0 3 2)
        | otherwise = Nothing
zn_isatap _ = Nothing


--------------------------------------------------------------------------------
-- Túneles entre nodos terminales
--------------------------------------------------------------------------------
```

```
182
-- Túnel manual IPv4 dentro de IPv6
nn_manual_4en6 :: ReglaC
nn_manual_4en6 (a1, Nd, [(Directa,Z6)], Nd, a2) = Just (
        (a1, Nd, [(Directa,Z4)], Nd, a2),
187     Cambio NN_manual_4en6 0 3 3)
nn_manual_4en6 _ = Nothing

-- Túnel manual IPv6 dentro de IPv4
nn_manual_6en4 :: ReglaC
192 nn_manual_6en4 (a1, Nd, [(Directa,Z4)], Nd, a2) = Just (
        (a1, Nd, [(Directa,Z6)], Nd, a2),
        Cambio NN_manual_6en4 0 3 3)
nn_manual_6en4 _ = Nothing

197 -- Túnel manual IPv6 dentro de IPv4 UDP
nn_manual_udp :: ReglaC
nn_manual_udp (a1, Nd, [/ (Directa,Z4), r@(Op4_td,Z4)+! /], Nd, a2) = Just (
        (a1, Nd, [(Directa,Z6)], Nd, a2),
        Cambio NN_manual_udp 0 (3 + length r) 3)
202 nn_manual_udp _ = Nothing

-- Túnel ISATAP
nn_isatap :: ReglaC
nn_isatap (a1, Nd, [(Directa,Z4)], Nd, a2) = Just (
207     (a1, Nd, [(Directa,Z6)], Nd, a2),
        Cambio NN_isatap 0 3 3)
nn_isatap _ = Nothing
```

12.3.4. Modificaciones sobre redes finales

```
1  ------------------------------------------------------------------------------
   -- Escenarios solución.
   --
   -- La función "solucion" devuelve True si el escenario es solución (es decir,
   -- hay conectividad de manera directa), y False si es necesario aplicar algún
6  -- mecanismo de transición adicional.
   ------------------------------------------------------------------------------

   solucion :: Escenario -> Bool

11 -- Zonas IPv4 con aplicaciones IPv4.
   solucion (A4, N4, [(_,Z4)], N4, A4) = True
   solucion (A4, N4, [(_,Z4)], Nd, A4) = True
   solucion (A4, Nd, [(_,Z4)], N4, A4) = True
   solucion (A4, Nd, [(_,Z4)], Nd, A4) = True
16
   -- Zonas IPv4 con aplicaciones IPv6.
   solucion (A6, Nmap, [(_,Z4)], Nmap, A6) = True

   -- Zonas IPv4 con aplicaciones heterogéneas.
```

```
solucion (A4, N4, [(_,Z4)], Nmap, A6) = True
solucion (A4, Nd, [(_,Z4)], Nmap, A6) = True
solucion (A6, Nmap, [(_,Z4)], N4, A4) = True
solucion (A6, Nmap, [(_,Z4)], Nd, A4) = True

-- Zonas IPv6 con aplicaciones IPv4.
solucion (A4, Nd, [(_,Z6)], Nd, A4) = True

-- Zonas IPv6 con aplicaciones IPv6.
solucion (A6, N6, [(_,Z6)], N6, A6) = True
solucion (A6, N6, [(_,Z6)], Nd, A6) = True
solucion (A6, Nd, [(_,Z6)], N6, A6) = True
solucion (A6, Nd, [(_,Z6)], Nd, A6) = True

-- Zonas IPv6 con aplicaciones heterogéneas (transformar A4 en Ad).
solucion (A6, Nd, [(_,Z6)], Nd, A4) = True
solucion (A4, Nd, [(_,Z6)], Nd, A6) = True

-- Zonas IPv4 con traducción de direcciones. Si una de las dos aplicaciones
-- es IPv4, la red se trata como si tuviera un único elemento.
solucion (A4, n1, [(_,Z4),(Op4_td,Z4)], n2, a2) =
        solucion (A4, n1, [(Directa,Z4)], n2, a2)
solucion (a1, n1, [(_,Z4),(Op4_td,Z4)], n2, A4) =
        solucion (a1, n1, [(Directa,Z4)], n2, A4)
solucion (A6, Nmap, [(_,Z4),(Op4_td,Z4)], Nmap, A6) = True

-- Composición de dos zonas con aplicaciones IPv6.
solucion (A6, Nmap, [(_,Z4),(op,Z6)], N6, A6) = elem op [OpD_tp, OpD_tptd]
solucion (A6, Nmap, [(_,Z4),(op,Z6)], Nd, A6) = elem op [OpD_tp, OpD_tptd]

-- Composición de dos zonas con aplicaciones heterogéneas.
solucion (A4, N4, [(_,Z4),(op,Z6)], N6, A6) = elem op [OpD_tp, OpD_tptd]
solucion (A4, N4, [(_,Z4),(op,Z6)], Nd, A6) = elem op [OpD_tp, OpD_tptd]
solucion (A4, Nd, [(_,Z4),(op,Z6)], N6, A6) = elem op [OpD_tp, OpD_tptd]
solucion (A4, Nd, [(_,Z4),(op,Z6)], Nd, A6) = elem op [OpD_tp, OpD_tptd]
solucion (A6, N6, [(_,Z6),(op,Z4)], N4, A4) = elem op [OpD_tp, OpD_tptd]
solucion (A6, N6, [(_,Z6),(op,Z4)], Nd, A4) = elem op [OpD_tp, OpD_tptd]
solucion (A6, Nd, [(_,Z6),(op,Z4)], N4, A4) = elem op [OpD_tp, OpD_tptd]
solucion (A6, Nd, [(_,Z6),(op,Z4)], Nd, A4) = elem op [OpD_tp, OpD_tptd]

-- Las aplicaciones duales pueden comportarse como A4 o A6, según interese.
solucion (Ad, n1, red, n2, a2) = s4 || s6
        where
        s4 = solucion (A4, n1, red, n2, a2)
        s6 = solucion (A6, n1, red, n2, a2)
solucion (a1, n1, red, n2, Ad) = s4 || s6
        where
        s4 = solucion (a1, n1, red, n2, Ad)
        s6 = solucion (a1, n1, red, n2, Ad)
```

```
-- El resto de escenarios no son solución.
solucion _ = False


--------------------------------------------------------------------------------
-- Redes básicas IPv4 con aplicaciones IPv4
--------------------------------------------------------------------------------

-- Caso 1
base_Z4_A4 :: ReglaC
base_Z4_A4 (A4, N6, red@[(_,Z4)], n2, A4) = Just (
        (A4, Nd, red, n2, A4),
        Cambio Dual_stack 0 1 1)
base_Z4_A4 (A4, n1, red@[(_,Z4)], N6, A4) = Just (
        (A4, n1, red, Nd, A4),
        Cambio Dual_stack 2 1 1)
base_Z4_A4 _ = Nothing


--------------------------------------------------------------------------------
-- Redes básicas IPv4 con aplicaciones IPv6
--------------------------------------------------------------------------------

-- Caso 1
base_Z4_A6_1 :: ReglaC
base_Z4_A6_1 (A6, n1, red@[(_,Z4)], n2, A6) = if
        (elem n1 [N4, N6]) &&
        (elem n2 [N4, N6, Nd])
        then Just (
                (A6, Nd, red, n2, A6),
                Cambio Dual_stack 0 1 1)
        else Nothing
base_Z4_A6_1 _ = Nothing

-- Caso 2
base_Z4_A6_2 :: ReglaC
base_Z4_A6_2 (A6, Nd, red@[(_,Z4)], Nd, A6) = Just (
        (A6, Nmap, red, Nmap, A6),
        Cambio Mapped 0 3 3)
base_Z4_A6_2 _ = Nothing

-- Caso 3
base_Z4_A6_3 :: ReglaC
base_Z4_A6_3 (A6, Nd, [(_,Z4)], Nd, A6) = Just (
        (A6, Nd, [(Directa,Z6)], Nd, A6),
        Cambio NN_manual_6en4 0 3 3)
base_Z4_A6_3 _ = Nothing


--------------------------------------------------------------------------------
```

```
-- Redes básicas IPv4 con aplicaciones heterogéneas
--------------------------------------------------------------------------------

-- Caso 1
base_Z4_Ah_1 :: ReglaC
base_Z4_Ah_1 (A4, N6, red@[(_,Z4)], n2, A6) = if
        elem n2 [N4, N6, Nd]
        then Just (
                (A4, Nd, red, n2, A6),
                Cambio Dual_stack 0 1 1)
        else Nothing
base_Z4_Ah_1 _ = Nothing

-- Caso 2
base_Z4_Ah_2 :: ReglaC
base_Z4_Ah_2 (A4, n1, red@[(_,Z4)], n2, A6) = if
        (elem n1 [N4, N6, Nd]) &&
        (elem n2 [N4, N6])
        then Just (
                (A4, n1, red, Nd, A6),
                Cambio Dual_stack 2 1 1)
        else Nothing
base_Z4_Ah_2 _ = Nothing

-- Caso 3
base_Z4_Ah_3 :: ReglaC
base_Z4_Ah_3 (A4, n1, red@[(_,Z4)], Nd, A6) = if
        elem n1 [N4, Nd]
        then Just (
                (A4, n1, red, Nmap, A6),
                Cambio Mapped 2 1 1)
        else Nothing
base_Z4_Ah_3 _ = Nothing


--------------------------------------------------------------------------------
-- Redes básicas IPv6 con aplicaciones IPv4
--------------------------------------------------------------------------------

-- Caso 1
base_Z6_A4_1 :: ReglaC
base_Z6_A4_1 (A4, n1, red@[(_,Z6)], n2, A4) = if
        (elem n1 [N4, N6]) &&
        (elem n2 [N4, N6, Nd])
        then Just (
                (A4, Nd, red, n2, A4),
                Cambio Dual_stack 0 1 1)
        else Nothing
base_Z6_A4_1 _ = Nothing
```

```
-- Caso 2
base_Z6_A4_2 :: ReglaC
base_Z6_A4_2 (A4, Nd, [(_,Z6)], Nd, A4) = Just (
        (A4, Nd, [(Directa,Z4)], Nd, A4),
        Cambio NN_manual_4en6 0 3 3)
base_Z6_A4_2 _ = Nothing


-------------------------------------------------------------------------------
-- Redes básicas IPv6 con aplicaciones IPv6
-------------------------------------------------------------------------------

-- Caso 1
base_Z6_A6 :: ReglaC
base_Z6_A6 (A6, N4, red@[(_,Z6)], n2, A6) = if
        elem n2 [N4, N6, Nd]
        then Just (
                (A6, Nd, red, n2, A6),
                Cambio Dual_stack 0 1 1)
        else Nothing
base_Z6_A6 _ = Nothing


-------------------------------------------------------------------------------
-- Redes básicas IPv6 con aplicaciones heterogéneas
-------------------------------------------------------------------------------

-- Caso 1
base_Z6_Ah_1 :: ReglaC
base_Z6_Ah_1 (A6, N4, red@[(_,Z6)], n2, A4) = if
        elem n2 [N4, N6, Nd]
        then Just (
                (A6, Nd, red, n2, A4),
                Cambio Dual_stack 0 1 1)
        else Nothing
base_Z6_Ah_1 _ = Nothing

-- Caso 2
base_Z6_Ah_2 :: ReglaC
base_Z6_Ah_2 (A6, n1, red@[(_,Z6)], n2, A4) = if
        (elem n1 [N4, N6, Nd]) &&
        (elem n2 [N4, N6])
        then Just (
                (A6, n1, red, Nd, A4),
                Cambio Dual_stack 2 1 1)
        else Nothing
base_Z6_Ah_2 _ = Nothing

-- Caso 3
base_Z6_Ah_3 :: ReglaC
```

```
base_Z6_Ah_3 (A6, Nd, [(_,Z6)], Nd, A4) = Just (
        (A6, Nd, [(Directa,Z4)], Nd, A4),
        Cambio NN_manual_4en6 0 3 3)
base_Z6_Ah_3 _ = Nothing


-------------------------------------------------------------------------------
-- Redes básicas duales con aplicaciones IPv4
-------------------------------------------------------------------------------

-- Caso 1
base_Zd_A4 :: ReglaC
base_Zd_A4 (A4, n1, [(_,Zd)], n2, A4) = Just (
        (A4, n1, [(Directa,Z4)], n2, A4),
        Cambio Nada 1 1 1)
base_Zd_A4 _ = Nothing



-------------------------------------------------------------------------------
-- Redes básicas duales con aplicaciones IPv6
-------------------------------------------------------------------------------

-- Caso 1
base_Zd_A6 :: ReglaC
base_Zd_A6 (A6, n1, [(_,Zd)], n2, A6) = Just (
        (A6, n1, [(Directa,Z6)], n2, A6),
        Cambio Nada 1 1 1)
base_Zd_A6 _ = Nothing



-------------------------------------------------------------------------------
-- Redes básicas duales con aplicaciones heterogéneas
-------------------------------------------------------------------------------

-- Caso 1
base_Zd_Ah_1 :: ReglaC
base_Zd_Ah_1 (A4, n1, [(_,Zd)], n2, A6) = Just (
        (A4, n1, [(Directa,Z4)], n2, A6),
        Cambio Nada 1 1 1)
base_Zd_Ah_1 _ = Nothing

-- Caso 2
base_Zd_Ah_2 :: ReglaC
base_Zd_Ah_2 (A4, n1, [(_,Zd)], n2, A6) = Just (
        (A4, n1, [(Directa,Z6)], n2, A6),
        Cambio Nada 1 1 1)
base_Zd_Ah_2 _ = Nothing



-------------------------------------------------------------------------------
```

```
-- Redes básicas IPv4 con traducción de direcciones
--------------------------------------------------------------------------------

-- Caso 1
base_NAT_1 :: ReglaC
base_NAT_1 (A6, n1, [(_,Z4), (Op4_td,Z4)], n2, A6) = if
        (elem n1 [N4, N6]) &&
        (elem n2 [N4, N6, Nd])
        then Just (
                (A6, Nd, [(Directa,Z4), (Op4_td,Z4)], n2, A6),
                Cambio Dual_stack 0 1 1)
        else Nothing
base_NAT_1 _ = Nothing

-- Caso 2
base_NAT_2 :: ReglaC
base_NAT_2 (A6, Nd, [(_,Z4), (Op4_td,Z4)], Nd, A6) = Just (
        (A6, Nmap, [(Directa,Z4)], Nmap, A6),
        Cambio Mapped 0 4 3)
base_NAT_2 _ = Nothing

-- Caso 3
base_NAT_3 :: ReglaC
base_NAT_3 (A6, Nd, [(_,Z4), (Op4_td,Z4)], Nd, A6) = Just (
        (A6, Nd, [(Directa,Z6)], Nd, A6),
        Cambio NN_teredo 0 3 3)
base_NAT_3 _ = Nothing



--------------------------------------------------------------------------------
-- Composición de dos zonas del mismo tipo
--------------------------------------------------------------------------------

-- Caso 1
dos_iguales :: ReglaC
dos_iguales (a1, n1, [(_,Z4),(OpD,     Z4)], n2, a2) = Just (
        (a1, n1, [(Directa,Z4)], n2, a2),
        Cambio Nada 1 2 1)
dos_iguales (a1, n1, [(_,Z4),(OpD_td,  Z4)], n2, a2) = Just (
        (a1, n1, [(Directa,Z4), (Op4_td,Z4)], n2, a2),
        Cambio Nada 1 2 2)
dos_iguales (a1, n1, [(_,Z4),(OpD_tp,  Z4)], n2, a2) = Just (
        (a1, n1, [(Directa,Z4)], n2, a2),
        Cambio Nada 1 2 1)
dos_iguales (a1, n1, [(_,Z4),(OpD_tptd,Z4)], n2, a2) = Just (
        (a1, n1, [(Directa,Z4), (Op4_td,Z4)], n2, a2),
        Cambio Nada 1 2 2)
dos_iguales (a1, n1, [(_,Z6),(OpD,     Z6)], n2, a2) = Just (
        (a1, n1, [(Directa,Z6)], n2, a2),
        Cambio Nada 1 2 1)
```

```
dos_iguales (a1, n1, [(_,Z6),(OpD_td,  Z6)], n2, a2) = Just (
        (a1, n1, [(Directa,Z6)], n2, a2),
        Cambio Nada 1 2 1)
dos_iguales (a1, n1, [(_,Z6),(OpD_tp,  Z6)], n2, a2) = Just (
        (a1, n1, [(Directa,Z6)], n2, a2),
        Cambio Nada 1 2 1)
dos_iguales (a1, n1, [(_,Z6),(OpD_tptd,Z6)], n2, a2) = Just (
        (a1, n1, [(Directa,Z6)], n2, a2),
        Cambio Nada 1 2 1)
dos_iguales _ = Nothing


-------------------------------------------------------------------------------
-- Composición de dos zonas diferentes con aplicaciones IPv4
-------------------------------------------------------------------------------

-- Caso 1
dos_A4_1 :: ReglaC
dos_A4_1 (A4, n1, red@[(_,Z4),(op,Z6)], n2, A4) = if
        (elem n1 [N4, N6, Nd]) &&
        (elem n2 [N4, N6]) &&
        (elem op [OpD, OpD_td, OpD_tp, OpD_tptd])
        then Just (
                (A4, n1, red, Nd, A4),
                Cambio Dual_stack 3 1 1)
        else Nothing
dos_A4_1 _ = Nothing

-- Caso 2
dos_A4_2 :: ReglaC
dos_A4_2 (A4, N6, red@[(_,Z4),(op,Z6)], Nd, A4) = if
        elem op [OpD, OpD_td, OpD_tp, OpD_tptd]
        then Just (
                (A4, Nd, red, Nd, A4),
                Cambio Dual_stack 0 1 1)
        else Nothing
dos_A4_2 _ = Nothing

-- Caso 3
dos_A4_3 :: ReglaC
dos_A4_3 (A4, n1, [(_,Z4),(op,Z6)], Nd, A4) = if
        (elem n1 [N4, Nd]) &&
        (elem op [OpD, OpD_tp])
        then Just (
                (A4, n1, [(Directa,Z4)], Nd, A4),
                Cambio ZN_manual_4en6 1 3 2)
        else Nothing
dos_A4_3 _ = Nothing

-- Caso 4
```

```
dos_A4_4 :: ReglaC
dos_A4_4 (A4, n1, [(_,Z4),(op,Z6)], Nd, A4) = if
        (elem n1 [N4, Nd]) &&
        (elem op [OpD_td, OpD_tptd])
        then Just (
                (A4, n1, [(Directa,Z4),(Op4_td,Z4)], Nd, A4),
                Cambio ZN_manual_4en6 1 3 3)
        else Nothing
dos_A4_4 _ = Nothing


-------------------------------------------------------------------------------
-- Composición de dos zonas diferentes con aplicaciones IPv6
-------------------------------------------------------------------------------

-- Caso 1
dos_A6_1 :: ReglaC
dos_A6_1 (A6, n1, red@[(_,Z4),(op,Z6)], n2, A6) = if
        (elem n1 [N4, N6]) &&
        (elem n2 [N4, N6, Nd]) &&
        (elem op [OpD, OpD_td, OpD_tp, OpD_tptd])
        then Just (
                (A6, Nd, red, n2, A6),
                Cambio Dual_stack 3 1 1)
        else Nothing
dos_A6_1 _ = Nothing

-- Caso 2
dos_A6_2 :: ReglaC
dos_A6_2 (A6, n1, red@[(_,Z4),(op,Z6)], N4, A6) = if
        (elem n1 [N4, N6, Nd]) &&
        (elem op [OpD, OpD_td, OpD_tp, OpD_tptd])
        then Just (
                (A6, n1, red, Nd, A6),
                Cambio Dual_stack 3 1 1)
        else Nothing
dos_A6_2 _ = Nothing

-- Caso 3
dos_A6_3 :: ReglaC
dos_A6_3 (A6, Nd, [(_,Z4),(op,Z6)], n2, A6) = if
        (elem n2 [N6, Nd]) &&
        (elem op [OpD, OpD_td, OpD_tp, OpD_tptd])
        then Just (
                (A6, Nd, [(Directa,Z6)], n2, A6),
                Cambio ZN_manual_6en4 0 3 2)
        else Nothing
dos_A6_3 _ = Nothing

-- Caso 4
```

```
dos_A6_4 :: ReglaC
dos_A6_4 (A6, Nd, [(_,Z4),(op,Z6)], n2, A6) = if
        (elem n2 [N6, Nd]) &&
        (elem op [OpD, OpD_td])
        then Just (
                (A6, Nmap, [(Directa,Z4),(op',Z6)], n2, A6),
                Cambio Mapped 0 1 1)
        else Nothing
        where
                op' = if op == OpD then OpD_tp else OpD_tptd
dos_A6_4 _ = Nothing


-------------------------------------------------------------------------------
-- Composición de dos zonas diferentes con aplicaciones heterogéneas
-------------------------------------------------------------------------------

-- Caso 1
dos_Ah_1 :: ReglaC
dos_Ah_1 (A4, N6, red@[(_,Z4),(op,Z6)], n2, A6) = if
        (elem op [OpD, OpD_td, OpD_tp, OpD_tptd]) &&
        (elem n2 [N4, N6, Nd])
        then Just (
                (A4, Nd, red, n2, A6),
                Cambio Dual_stack 0 1 1)
        else Nothing
dos_Ah_1 _ = Nothing

-- Caso 2
dos_Ah_2 :: ReglaC
dos_Ah_2 (A4, n1, red@[(_,Z4),(op,Z6)], N4, A6) = if
        (elem op [OpD, OpD_td, OpD_tp, OpD_tptd]) &&
        (elem n1 [N4, Nd])
        then Just (
                (A4, n1, red, Nd, A6),
                Cambio Dual_stack 3 1 1)
        else Nothing
dos_Ah_2 _ = Nothing

-- Caso 3
dos_Ah_3 :: ReglaC
dos_Ah_3 (A4, n1, red@[(_,Z4),(op,Z6)], N6, A6) = if
        (elem op [OpD, OpD_td]) &&
        (elem n1 [N4, N6, Nd])
        then Just (
                (A4, n1, red, Nd, A6),
                Cambio Dual_stack 3 1 1)
        else Nothing
dos_Ah_3 _ = Nothing
```

```
-- Caso 4
dos_Ah_4 :: ReglaC
dos_Ah_4 (A4, n1, [(_,Z4),(op,Z6)], n2, A6) = if
        (elem op [OpD, OpD_td]) &&
        (elem n2 [N6, Nd]) &&
        (elem n1 [N4, Nd])
        then Just (
                (A4, n1, [(Directa,Z4),(op',Z6)], n2, A6),
                Cambio Traduccion 1 2 2)
        else Nothing
        where
                op' = if op == OpD then OpD_tp else OpD_tptd
dos_Ah_4 _ = Nothing

-- Caso 5
dos_Ah_5 :: ReglaC
dos_Ah_5 (A4, n1, [(_,Z4),(op,Z6)], Nd, A6) = if
        (elem op [OpD, OpD_tp]) &&
        (elem n1 [N6, Nd])
        then Just (
                (A4, n1, [(Directa,Z4)], Nd, A6),
                Cambio ZN_manual_4en6 1 3 2)
        else Nothing
dos_Ah_5 _ = Nothing

-- Caso 6
dos_Ah_6 :: ReglaC
dos_Ah_6 (A4, n1, [(_,Z4),(op,Z6)], Nd, A6) = if
        (elem op [OpD_td, OpD_tptd]) &&
        (elem n1 [N6, Nd])
        then Just (
                (A4, n1, [(Directa,Z4),(Op4_td,Z4)], Nd, A6),
                Cambio ZN_manual_4en6 1 3 3)
        else Nothing
dos_Ah_6 _ = Nothing

-- Caso 7
dos_Ah_7 :: ReglaC
dos_Ah_7 (A4, n1, red@[(_,Z6),(op,Z4)], n2, A6) = if
        (elem op [OpD, OpD_td, OpD_tp, OpD_tptd]) &&
        (elem n1 [N4, N6]) &&
        (elem n2 [N4, N6, Nd])
        then Just (
                (A4, Nd, red, n2, A6),
                Cambio Dual_stack 0 1 1)
        else Nothing
dos_Ah_7 _ = Nothing

-- Caso 8
dos_Ah_8 :: ReglaC
```

```
dos_Ah_8 (A4, Nd, red@[(_,Z6),(op,Z4)], n2, A6) = if
        (elem op [OpD, OpD_td, OpD_tp, OpD_tptd]) &&
        (elem n2 [N4, N6])
        then Just (
                (A4, Nd, red, Nd, A6),
                Cambio Dual_stack 3 1 1)
        else Nothing
dos_Ah_8 _ = Nothing

-- Caso 9
dos_Ah_9 :: ReglaC
dos_Ah_9 (A4, Nd, [(_,Z6),(op,Z4)], Nd, A6) = if
        (elem op [OpD, OpD_tp])
        then Just (
                (A4, Nd, [(Directa,Z4)], Nd, A6),
                Cambio ZN_manual_4en6 0 3 2)
        else Nothing
dos_Ah_9 _ = Nothing

-- Caso 10
dos_Ah_10 :: ReglaC
dos_Ah_10 (A4, Nd, [(_,Z6),(op,Z4)], Nd, A6) = if
        (elem op [OpD_td, OpD_tptd])
        then Just (
                (A4, Nd, [(Directa,Z4),(Op4_td,Z4)], Nd, A6),
                Cambio ZN_manual_4en6 0 3 3)
        else Nothing
dos_Ah_10 _ = Nothing


-------------------------------------------------------------------------------
-- Composición de tres zonas v4-v6-v4 con aplicaciones IPv4
-------------------------------------------------------------------------------

-- Caso 1
tres_464_A4_1 :: ReglaC
tres_464_A4_1 (A4, n1, [(_,Z4),(op1,Z6),(op2,Z4)], n2, A4) = if
        (subset [n1, n2] [N4, N6, Nd]) &&
        (subset [op1, op2] [OpD, OpD_td, OpD_tp, OpD_tptd])
        then Just (
                (A4, n1, [(Directa,Z4)], n2, A4),
                Cambio ZZ_manual_4en6 1 3 1)
        else Nothing
tres_464_A4_1 _ = Nothing

-- Caso 2
tres_464_A4_2 :: ReglaC
tres_464_A4_2 (A4, n1, [(_,Z4),(op1,Z6),(op2,Z4)], Nd, A4) = if
        (elem n1 [N4, N6, Nd]) &&
        (subset [op1, op2] [OpD, OpD_td, OpD_tp, OpD_tptd])
```

```
        then Just (
                (A4, n1, [(Directa,Z4),(op1,Z6)], Nd, A4),
                Cambio ZZ_manual_6en4 2 3 2)
        else Nothing
tres_464_A4_2 _ = Nothing


-------------------------------------------------------------------------------
-- Composición de tres zonas v4-v6-v4 con aplicaciones IPv6
-------------------------------------------------------------------------------

-- Caso 1
tres_464_A6_1 :: ReglaC
tres_464_A6_1 (A6, n1, [(_,Z4),(op1,Z6),(op2,Z4)], n2, A6) = if
        (subset [n1, n2] [N4, N6, Nd]) &&
        (subset [op1, op2] [OpD, OpD_td, OpD_tp, OpD_tptd])
        then Just (
                (A6, n1, [(Directa,Z4)], n2, A6),
                Cambio ZZ_manual_4en6 1 3 1)
        else Nothing
tres_464_A6_1 _ = Nothing

-- Caso 2
tres_464_A6_2 :: ReglaC
tres_464_A6_2 (A6, n1, [(_,Z4),(op1,Z6),(op2,Z4)], Nd, A6) = if
        (elem n1 [N4, N6, Nd]) &&
        (subset [op1, op2] [OpD, OpD_td, OpD_tp, OpD_tptd])
        then Just (
                (A6, n1, [(Directa,Z4),(op1,Z6)], Nd, A6),
                Cambio ZZ_manual_6en4 2 3 2)
        else Nothing
tres_464_A6_2 _ = Nothing


-------------------------------------------------------------------------------
-- Composición de tres zonas v4-v6-v4 con aplicaciones heterogéneas
-------------------------------------------------------------------------------

-- Caso 1
tres_464_Ah_1 :: ReglaC
tres_464_Ah_1 (A4, n1, [(_,Z4),(op1,Z6),(op2,Z4)], n2, A6) = if
        (subset [n1, n2] [N4, N6, Nd]) &&
        (subset [op1, op2] [OpD, OpD_td, OpD_tp, OpD_tptd])
        then Just (
                (A4, n1, [(Directa,Z4)], n2, A6),
                Cambio ZZ_manual_4en6 1 3 1)
        else Nothing
tres_464_Ah_1 _ = Nothing

-- Caso 2
```

```
tres_464_Ah_2 :: ReglaC
tres_464_Ah_2 (A4, n1, [(_,Z4),(op1,Z6),(op2,Z4)], Nd, A6) = if
        (elem n1 [N4, N6, Nd]) &&
        (subset [op1, op2] [OpD, OpD_td, OpD_tp, OpD_tptd])
        then Just (
                (A4, n1, [(Directa,Z4),(op1,Z6)], Nd, A6),
                Cambio ZZ_manual_6en4 2 3 2)
        else Nothing
tres_464_Ah_2 _ = Nothing

-- Caso 3
tres_464_Ah_3 :: ReglaC
tres_464_Ah_3 (A4, Nd, [(_,Z4),(op1,Z6),(op2,Z4)], n2, A6) = if
        (elem n2 [N4, N6, Nd]) &&
        (subset [op1, op2] [OpD, OpD_td, OpD_tp, OpD_tptd])
        then Just (
                (A4, Nd, [(Directa,Z6),(op2,Z4)], n2, A6),
                Cambio ZZ_manual_6en4 0 3 2)
        else Nothing
tres_464_Ah_3 _ = Nothing


-------------------------------------------------------------------------------
-- Composición de tres zonas v6-v4-v6 con aplicaciones IPv4
-------------------------------------------------------------------------------

-- Caso 1
tres_646_A4_1 :: ReglaC
tres_646_A4_1 (A4, n1, [(_,Z6),(op1,Z4),(op2,Z6)], n2, A4) = if
        (subset [n1, n2] [N4, N6, Nd]) &&
        (subset [op1, op2] [OpD, OpD_td, OpD_tp, OpD_tptd])
        then Just (
                (A4, n1, [(Directa,Z6)], n2, A4),
                Cambio ZZ_manual_6en4 1 3 1)
        else Nothing
tres_646_A4_1 _ = Nothing

-- Caso 2
tres_646_A4_2 :: ReglaC
tres_646_A4_2 (A4, n1, [(_,Z6),(op1,Z4),(op2,Z6)], Nd, A4) = if
        (elem n1 [N4, N6, Nd]) &&
        (elem op1 [OpD, OpD_td, OpD_tp, OpD_tptd]) &&
        (elem op2 [OpD, OpD_tp])
        then Just (
                (A4, n1, [(Directa,Z6),(op1,Z4)], Nd, A4),
                Cambio ZZ_manual_4en6 2 3 2)
        else Nothing
tres_646_A4_2 _ = Nothing

-- Caso 3
```

```
tres_646_A4_3 :: ReglaC
tres_646_A4_3 (A4, n1, [(_,Z6),(op1,Z4),(op2,Z6)], Nd, A4) = if
        (elem n1 [N4, N6, Nd]) &&
        (elem op1 [OpD, OpD_td, OpD_tp, OpD_tptd]) &&
        (elem op2 [OpD_td, OpD_tptd])
        then Just (
                (A4, n1, [(Directa,Z6),(op1,Z4),(Op4_td,Z4)], Nd, A4),
                Cambio ZZ_manual_4en6 2 3 3)
        else Nothing
tres_646_A4_3 _ = Nothing


-------------------------------------------------------------------------------
-- Composición de tres zonas v6-v4-v6 con aplicaciones IPv6
-------------------------------------------------------------------------------

-- Caso 1
tres_646_A6_1 :: ReglaC
tres_646_A6_1 (A6, n1, [(_,Z6),(op1,Z4),(op2,Z6)], n2, A6) = if
        (subset [n1, n2] [N4, N6, Nd]) &&
        (subset [op1, op2] [OpD, OpD_td, OpD_tp, OpD_tptd])
        then Just (
                (A6, n1, [(Directa,Z6)], n2, A6),
                Cambio ZZ_manual_6en4 1 3 1)
        else Nothing
tres_646_A6_1 _ = Nothing

-- Caso 2
tres_646_A6_2 :: ReglaC
tres_646_A6_2 (A6, n1, [(_,Z6),(op1,Z4),(op2,Z6)], Nd, A6) = if
        (elem n1 [N4, N6, Nd]) &&
        (elem op1 [OpD, OpD_td, OpD_tp, OpD_tptd]) &&
        (elem op2 [OpD, OpD_tp])
        then Just (
                (A6, n1, [(Directa,Z6),(op1,Z4)], Nd, A6),
                Cambio ZZ_manual_4en6 2 3 2)
        else Nothing
tres_646_A6_2 _ = Nothing

-- Caso 3
tres_646_A6_3 :: ReglaC
tres_646_A6_3 (A6, n1, [(_,Z6),(op1,Z4),(op2,Z6)], Nd, A6) = if
        (elem n1 [N4, N6, Nd]) &&
        (elem op1 [OpD, OpD_td, OpD_tp, OpD_tptd]) &&
        (elem op2 [OpD_td, OpD_tptd])
        then Just (
                (A6, n1, [(Directa,Z6),(op1,Z4),(Op4_td,Z4)], Nd, A6),
                Cambio ZZ_manual_4en6 2 3 3)
        else Nothing
tres_646_A6_3 _ = Nothing
```

```
721
-------------------------------------------------------------------------------
-- Composición de tres zonas v6-v4-v6 con aplicaciones v4-v6-v4
-------------------------------------------------------------------------------
726
-- Caso 1
tres_646_Ah_1 :: ReglaC
tres_646_Ah_1 (A4, n1, [(_,Z6),(op1,Z4),(op2,Z6)], n2, A6) = if
        (subset [n1, n2] [N4, N6, Nd]) &&
731     (subset [op1, op2] [OpD, OpD_td, OpD_tp, OpD_tptd])
        then Just (
                (A4, n1, [(Directa,Z6)], n2, A6),
                Cambio ZZ_manual_6en4 1 3 1)
        else Nothing
736 tres_646_Ah_1 _ = Nothing

-- Caso 2
tres_646_Ah_2 :: ReglaC
tres_646_Ah_2 (A4, n1, [(_,Z6),(op1,Z4),(op2,Z6)], Nd, A6) = if
741     (elem n1 [N4, N6, Nd]) &&
        (elem op1 [OpD, OpD_td, OpD_tp, OpD_tptd]) &&
        (elem op2 [OpD, OpD_tp])
        then Just (
                (A4, n1, [(Directa,Z6),(op1,Z4)], Nd, A6),
746             Cambio ZZ_manual_4en6 2 3 2)
        else Nothing
tres_646_Ah_2 _ = Nothing

-- Caso 3
751 tres_646_Ah_3 :: ReglaC
tres_646_Ah_3 (A4, n1, [(_,Z6),(op1,Z4),(op2,Z6)], Nd, A6) = if
        (elem n1 [N4, N6, Nd]) &&
        (elem op1 [OpD, OpD_td, OpD_tp, OpD_tptd]) &&
        (elem op2 [OpD_td, OpD_tptd])
756     then Just (
                (A4, n1, [(Directa,Z6),(op1,Z4),(Op4_td,Z4)], Nd, A6),
                Cambio ZZ_manual_4en6 2 3 3)
        else Nothing
tres_646_Ah_3 _ = Nothing
761
-- Caso 4
tres_646_Ah_4 :: ReglaC
tres_646_Ah_4 (A4, Nd, [(_,Z6),(op1,Z4),(op2,Z6)], n2, A6) = if
        (elem n2 [N4, N6, Nd]) &&
766     (elem op1 [OpD, OpD_tp]) &&
        (elem op2 [OpD, OpD_td, OpD_tp, OpD_tptd])
        then Just (
                (A4, Nd, [(Directa,Z4),(op2,Z6)], n2, A6),
                Cambio ZZ_manual_4en6 0 3 2)
```

```
771         else Nothing
    tres_646_Ah_4 _ = Nothing

    -- Caso 5
    tres_646_Ah_5 :: ReglaC
776 tres_646_Ah_5 (A4, Nd, [(_,Z6),(op1,Z4),(op2,Z6)], n2, A6) = if
        (elem n2 [N4, N6, Nd]) &&
        (elem op1 [OpD_td, OpD_tptd]) &&
        (elem op2 [OpD, OpD_td, OpD_tp, OpD_tptd])
        then Just (
781             (A4, Nd, [(Directa,Z4),(Op4_td,Z4),(op2,Z6)], n2, A6),
                Cambio ZZ_manual_4en6 0 3 3)
        else Nothing
    tres_646_Ah_5 _ = Nothing
```

12.4. Módulo Interfaz

Haskell ofrece múltiples posibilidades a la hora de crear interfaces de usuario, desde bibliotecas de muy alto nivel basadas en programación funcional reactiva, hasta bibliotecas de bajo nivel que permiten acceder a las funciones nativas del sistema operativo. Nosotros hemos considerado las cuatro bibliotecas de nivel medio más comunes en diversos lenguajes:

- Gtk2hs: biblioteca basada en Gtk+.
- HTk: bindings para Tcl/Tk.
- qtHaskell: bindings para Qt4.
- wxHaskell: biblioteca implementada encima de wxWidgets.

La única biblioteca que cumplía las condiciones de poder compilarse en la máquina de desarrollo, y disponer de una buena documentación, es Gtk2hs[5], por lo que ha sido nuestra decisión.

El modelo de desarrollo de Gtk2hs se basa en diseñar la interfaz con la herramienta Glade. La interfaz se guarda en un fichero XML, independiente del lenguaje de programación. Durante la ejecución, el programa carga el fichero XML, extrae los *widgets* que va a usar, define los manejadores de señal, y llama al bucle principal de Gtk+. Como tiene que interactuar con el mundo exterior, el código de la interfaz se define dentro de la mónada IO.

La interfaz resultante tiene un aspecto bastante atractivo. La ventana de edición principal se divide en tres partes. La parte izquierda contiene los controles de selección de modo, configuración del aspecto gráfico, zoom, e edición de los atributos de routers y enlaces. La parte central es el área de edición, donde se muestran los routers y los enlaces. La parte derecha es la lista de soluciones. Inicialmente estará vacía, y se rellenará cuando el usuario lance el proceso de búsqueda.

[5] http://www.haskell.org/gtk2hs/

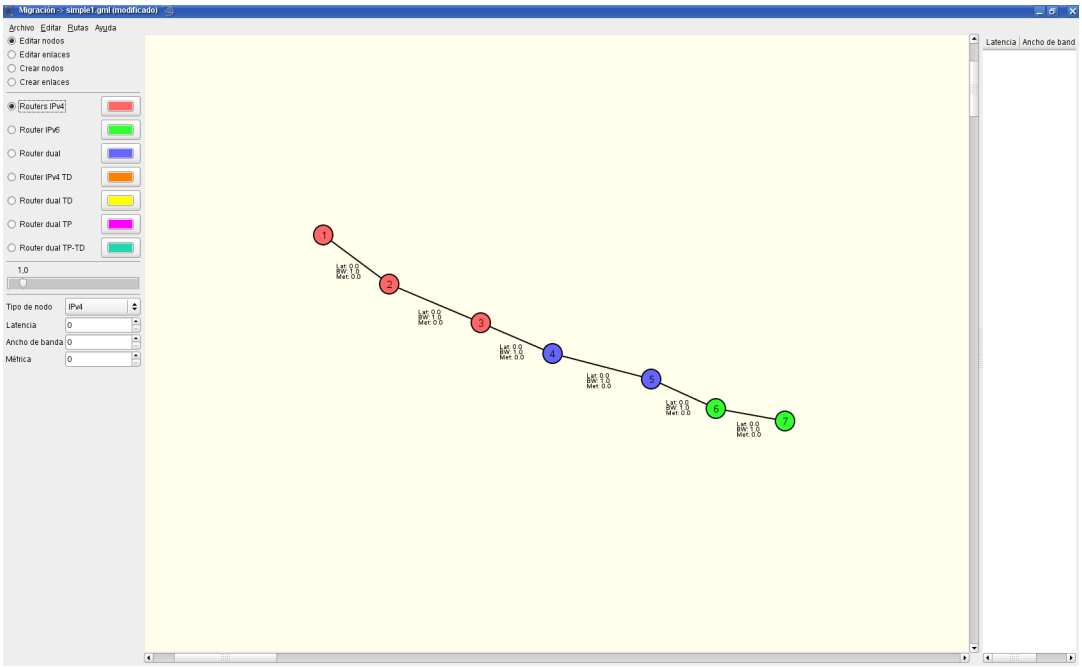

Figura 12.6: Ventana de edición

El diálogo de configuración de costes permite al usuario establecer el coste asociado a cada mecanismo. Se divide en tres pestañas (latencia, ancho de banda, y métrica personalizada), dentro de cada cual hay un campo numérico para introducir el coste de cada tipo de mecanismo.

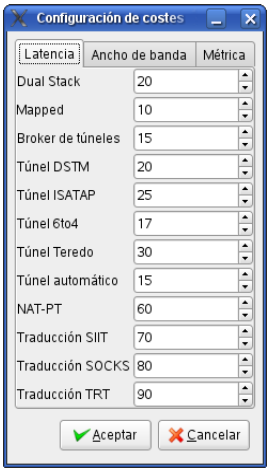

Figura 12.7: Diálogo de configuración de costes

El diálogo de búsqueda permite iniciar el proceso de resolución de escenarios. En primer lugar, el usuario debe configurar los parámetros del origen y el destino del escenario (router, tipo aplicación, y protocolo soportado por el nodo terminal). Además, el usuario puede decidir si usar algún criterio de poda. Los parámetros se guardarán para la siguiente vez. Cuando el usuario pulse el botón *Aceptar*, se iniciará el proceso de búsqueda.

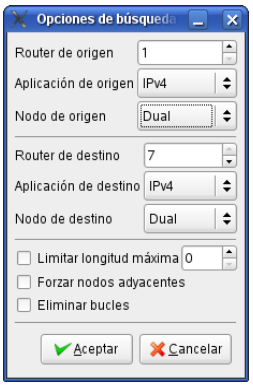

Figura 12.8: Diálogo de búsqueda

Una vez realizada la búsqueda, el usuario puede consultar los detalles de alguna de las soluciones obtenidas. El diálogo de detalles muestra la ruta seguida, los costes de la solución, y los túneles que se han tenido que crear.

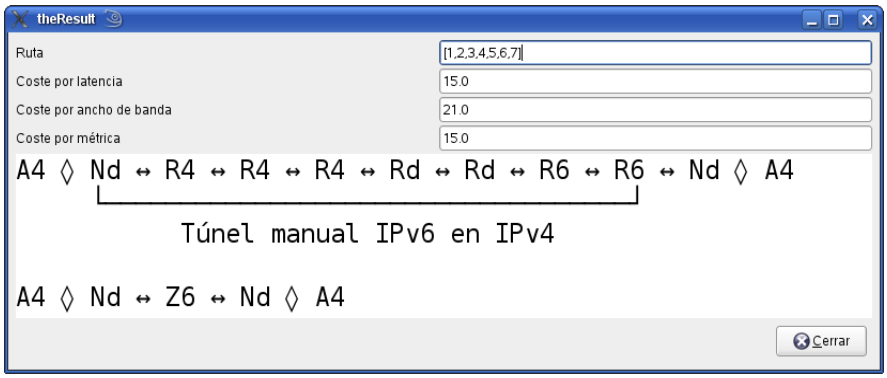

Figura 12.9: Ventana de detalles de una solución

13 Pruebas

13.1. Metodología de pruebas

En este capítulo describiremos cómo se ha probado el programa. Las pruebas se han realizado en paralelo con la implementación. Así, cada vez que se creaba una nueva función, o se modificaba función existente, se probaba que su funcionamiento era correcto. Una de las ventajas de Haskell es que, al ser puras la mayoría de las funciones, el resultado sólo depende de la entrada, por lo que se pueden probar de manera independiente sin tener en cuenta el orden de ejecución.

Se ha definido una batería de pruebas automáticas. Estas pruebas se encuentran en los módulos TEST de cada directorio. Se pueden ejecutar lanzando el programa con la opción "-test", de la siguiente manera:

```
$> ./migracion -test
```

Por otro lado, se han creado una serie de escenarios de ejemplo, para probar que los resultados devueltos por la aplicación están dentro de lo esperado. La interfaz de usuario se ha probado manualmente, ya que al ser interactiva no se pueden automatizar las pruebas.

13.2. Verificación con QuickCheck

QuickCheck es una herramienta de generación automática de baterías de prueba. Su funcionamiento básico consiste en definir propiedades que deben cumplir las funciones. Por ejemplo, si tenemos una función ordenada que devuelve un booleano indicando si una lista está ordenada, y una función insertar que añade un elemento a una lista ordenada manteniendo el orden, podríamos definir la siguiente propiedad:

```
prop_orden xs x = (ordenada xs) ==> ordenada (insertar x xs)
    where
    types = x :: Int
```

Figura 13.1: Ejemplo de propiedad con QuickCheck

Las propiedades se invocan llamando a la función quickCheck, que toma como parámetro la propiedad a verificar. En principio, genera 100 casos de prueba aleatorios. Si la propiedad se cumple para todos los casos, se considera que la prueba ha sido exitosa, mien-

tras que si alguno de ellos falla, se muestra el caso que ha fallado. Por ejemplo, para verificar el ejemplo anterior, escribiríamos lo siguiente:

```
> quickCheck prop_orden
OK, passed 100 tests.
```

Figura 13.2: Ejemplo de uso de QuickCheck

Hemos usado QuickCheck con dos objetivos. El primero ha sido validar el cargador de ficheros GML. Las pruebas consisten esencialmente en generar un grafo aleatorio, pasarlo a texto, volver a convertirlo a un grafo, y compararlo con el original. Si no son idénticos, es que algo ha ido mal. Este procedimiento ha permitido detectar fácilmente errores que de otra manera se nos hubieran escapado, como atributos que no se guardaban, u otros que se guardaban mal.

El segundo ha sido verificar propiedades genéricas de las reglas de transición. Se han definido las siguientes propiedades:

- La aplicación de una regla de transición siempre reduce el número de nodos de la red (o por lo menos lo mantiene igual).
- Dadas dos reglas de formalización o canonización, la aplicación de ambas es conmutativa.
- Tras la aplicación de las reglas de creación de zonas no puede ocurrir que haya dos zonas $\{Z_4, Z_6, Z_d\}$ adyacentes del mismo tipo.

13.3. Pruebas unitarias

A la hora de implementar el proceso de carga de topologías, se ha procurado tratar el mayor número posible de casos de error:

- Que el fichero de entrada no contenga grafos.
- Que un grafo no contenga nodos o arcos.
- Que un nodo no tenga identificador, o sea incorrecto.
- Que un arco no tenga origen o destino.
- Que el tipo de datos de algún campo básico sea incorrecto (por ejemplo, que el identificador no sea un número entero).
- Que los arcos unan nodos existentes.

Se han escrito varios ficheros GML para probar cada una de estas condiciones. Estos ficheros se encuentran en el directorio *ejemplos/pruebas*.

Por otro lado, para cada regla de transición, se han definido todos los posibles escenarios que pueden ser procesados por esa regla, junto con la solución esperada. De esta manera se prueba que las reglas funcionan correctamente. Por ejemplo, el tercer operador de conexión se ha probado de la siguiente manera:

$$Z_i \leftrightarrow (Z_d \leftrightarrow) \star R_d^t \, (\leftrightarrow Z_d) \star \leftrightarrow Z_j = Z_i \odot_d Z_j$$
$$\text{donde } Z_i, Z_j \in \{Z_4, Z_6\}$$
$$\Downarrow$$
$$Z_4 \leftrightarrow R_d^t \leftrightarrow Z_4$$
$$Z_4 \leftrightarrow R_d^t \leftrightarrow Z_6$$
$$Z_6 \leftrightarrow R_d^t \leftrightarrow Z_4$$
$$Z_6 \leftrightarrow R_d^t \leftrightarrow Z_6$$
$$Z_4 \leftrightarrow R_d^t \leftrightarrow Z_d \leftrightarrow Z_4$$
$$Z_4 \leftrightarrow Z_d \leftrightarrow R_d^t \leftrightarrow Z_4$$
$$Z_4 \leftrightarrow Z_d \leftrightarrow R_d^t \leftrightarrow Z_d \leftrightarrow Z_4$$

También se han escrito varias topologías de ejemplo sencillas, para probar el funcionamiento de los algoritmos de búsqueda de rutas. Además, se han implementado un conjunto de funciones de generación de topologías comunes (lineal, anillo, estrella y malla), con el fin de poder generar casos de prueba con mayor comodidad.

13.4. Escenarios de ejemplo

Se han tomado los escenarios de ejemplo del trabajo anterior. Estos escenarios prueban diversas situaciones comunes en redes reales. Se ha usado la misma tabla de costes para todos ellos:

Mecanismo	Coste
Dual Stack	20
Mapped	10
Broker de túneles	30
Túnel DSTM	35
Túnel ISATAP	40
Túnel 6to4	20
Túnel Teredo	30
Túnel automático	30
NAT-PT	55
Traducción SIIT	65
Traducción SOCKS	70
Traducción TRT	90

El primer escenario intenta establecer conectividad entre aplicaciones IPv6 que se ejecutan sobre una red IPv4. Los parámetros son:

- NT1: router 1, nodo IPv4, aplicación IPv6.
- NT2: router 3, nodo dual, aplicación IPv6.

La herramienta genera las siguientes soluciones:

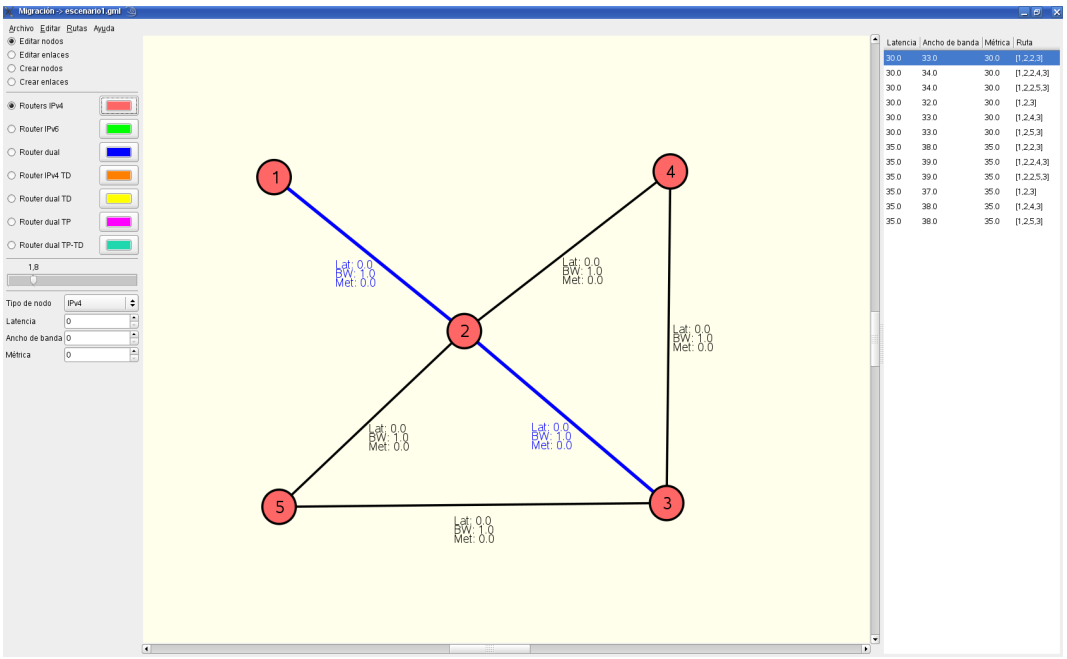

Figura 13.3: Soluciones del escenario de ejemplo 1

Podemos ver como la aplicación obtiene en primer lugar la ruta directa entre el origen y el destino. Este escenario es muy simple, ya que todos los routers son del mismo tipo, por lo que todas las soluciones obtenidas son similares. Aún así, la aplicación del primer nodo terminal es incompatible con éste, por lo que hay que instalar el mecanismo *Dual Stack*. Además, como las aplicaciones son IPv6 y los routers intermedios IPv4, se instala el mecanismo *Mapped* en ambos nodos terminales.

El segundo escenario intenta establecer conectividad entre aplicaciones IPv6 que se encuentran detrás de un NAT. Los parámetros son:

- NT1: router 1, nodo dual, aplicación IPv6.
- NT2: router 8, nodo dual, aplicación IPv6.

La herramienta genera las siguientes soluciones:

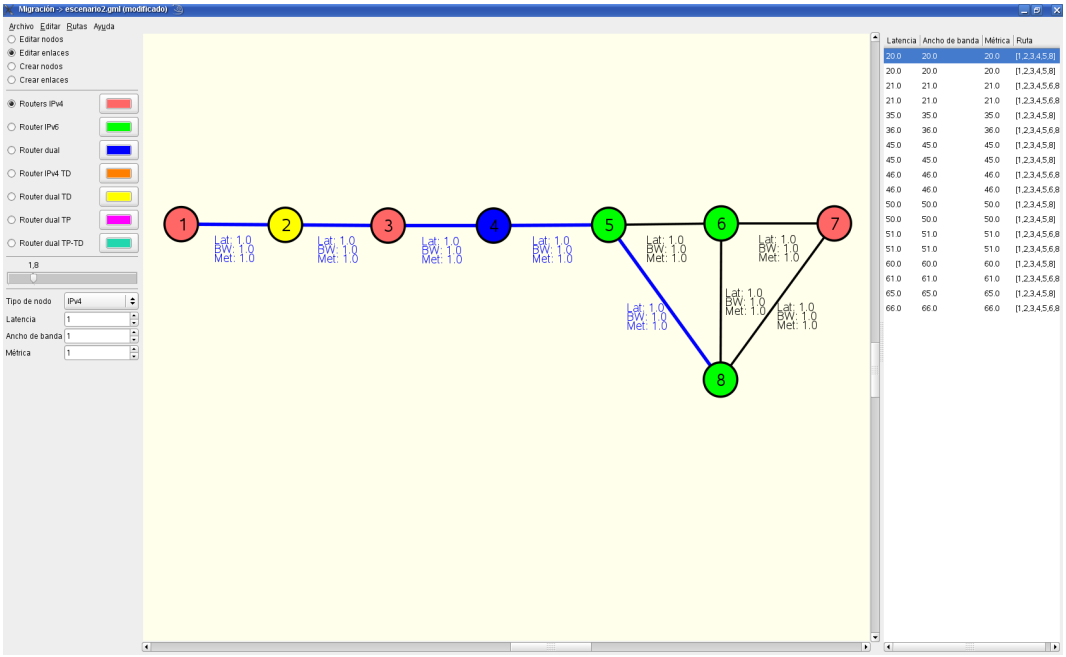

Figura 13.4: Soluciones del escenario de ejemplo 2

Vemos como la aplicación toma la ruta directa. En este caso hay routers de diferentes tipos, por lo que se tienen que crear túneles, lo que se refleja en los costes obtenidos. En concreto, el segundo router usa NAT, por lo que hay que usar un túnel IPv6 dentro de UDP IPv4.

El tercer escenario intenta establecer conectividad entre aplicaciones IPv4 e IPv6 sobre una red heterogénea. Los parámetros son:

- NT1: router 1, nodo dual, aplicación IPv4.
- NT2: router 7, nodo IPv6, aplicación IPv6.

La herramienta genera las siguientes soluciones:

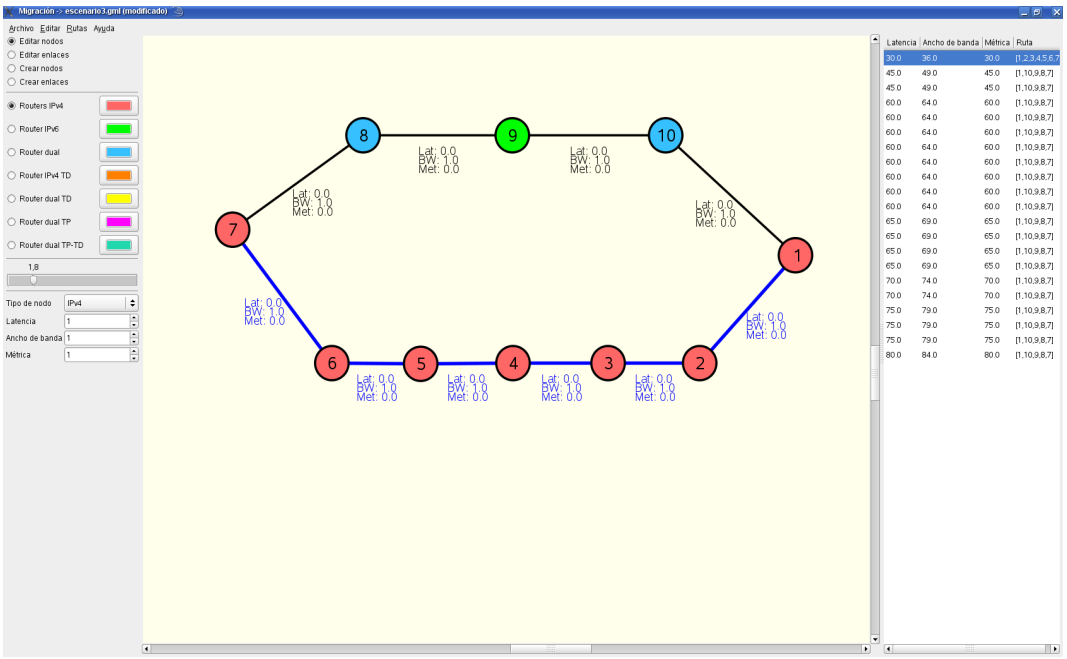

Figura 13.5: Soluciones del escenario de ejemplo 3

En este escenario podemos observar una situación interesante. La ruta menos costosa es la más larga. Esto se debe a que todos los routers son del mismo tipo, por lo que no hace falta crear túneles. Sin embargo, la aplicación que se ejecuta en el segundo nodo terminal es incompatible con su router, por lo que hará falta instalar los mecanismos *Dual Stack* y *Mapped* en el nodo.

El cuarto escenario intenta establecer conectividad entre aplicaciones IPv4 sobre una red compleja, formada por routers de varios tipos. Los parámetros son:

- NT1: router 1, nodo dual, aplicación IPv4.
- NT2: router 5, nodo dual, aplicación IPv4.

La herramienta genera las siguientes soluciones:

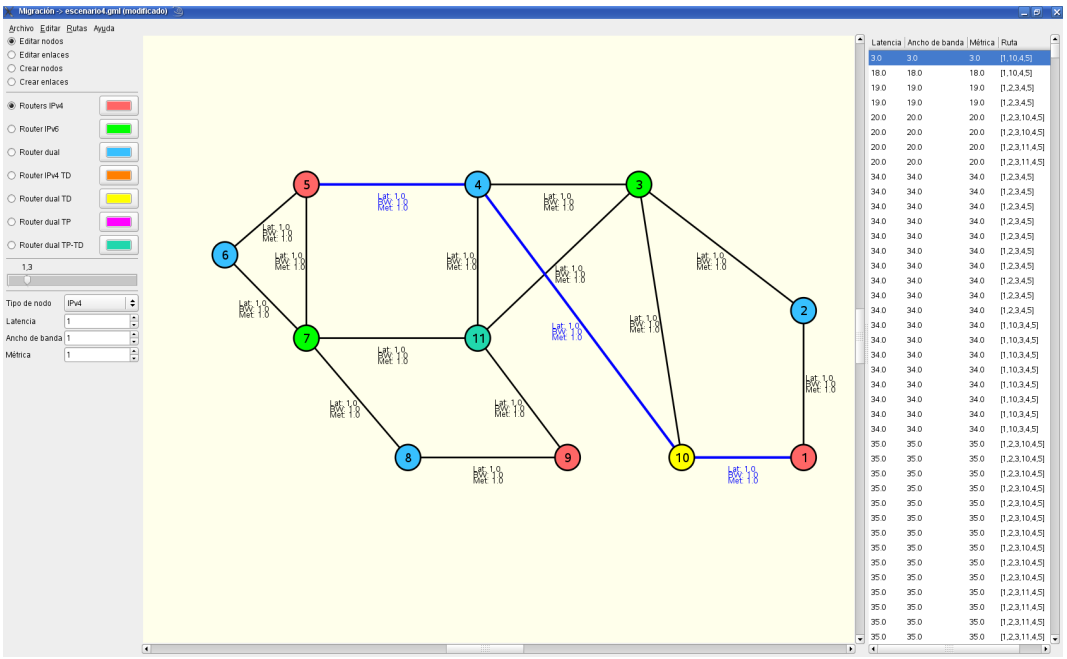

Figura 13.6: Soluciones del escenario de ejemplo 4

Al ser un escenario más complejo, el número de soluciones encontradas es muchísimo mayor que en los otros casos. La aplicación devuelve en primer lugar la ruta más corta, ya que al permitir todos los routers el paso de tráfico IPv4, no requiere la creación de ningún túnel. El resto de soluciones implican establecer túneles o usar el mecanismo *Mapped*. No hace falta instalar el mecanismo *Dual Stack*, ya que los nodos terminales ya son duales. Hay que tener en cuenta que no todas las soluciones encontradas tienen sentido; por ejemplo, las soluciones más caras implican establecer hasta cuatro túneles anidados. Esto es algo inevitable en un escenario que combina routers de diversos tipos.

Parte VI

Conclusiones

Conclusiones finales

Como hemos visto, el proceso de migración de IPv4 a IPv6 es largo y complejo, pero imprescindible. Por suerte, contamos con una amplia variedad de mecanismos de transición, capaces de adaptarse a todas las situaciones que se puedan presentar. Son muchos los proyectos abiertos entorno a la implantación de IPv6, aunque por desgracia ninguno ha llegado aún al usuario doméstico. En cualquier caso, el estudio de las diferentes estrategias de transición es sumamente importante, ya que es sólo cuestión de tiempo que empiecen a ser inevitables.

Nuestro estudio no ha profundizado en el funcionamiento de cada mecanismo de transición, sólo en su dominio de aplicación. En este sentido, la metodología MENINA ha resultado fundamental. Al proporcionar un modelo algebraico del problema, nos ha permitido alcanzar un mayor nivel de abstracción con el que tratar el problema de manera automática.

Hemos tenido que desarrollar algunos algoritmos, como la búsqueda de rutas en grafos o la resolución de escenarios. Aunque estos algoritmos están muy estudiados, hemos intentado darles un nuevo enfoque aprovechando las capacidades de los lenguajes funcionales. Se ha procurado que la implementación sea lo más clara y modular posible, para que de esta manera, si en un futuro hay que añadir nuevas capacidades a la aplicación, no suponga ningún reto.

Haskell se ha mostrado como un lenguaje perfectamente capaz para el desarrollo de aplicaciones complejas. En especial, la evaluación perezosa resulta muy útil en la resolución de problemas de búsqueda, ya que permite modularizar mejor el código. Por otro lado, las capacidades de manipulación de tipos de datos complejos permiten rivalizar con la expresividad de los modernos lenguajes interpretados, pero con la eficiencia de un lenguaje compilado.

Las herramientas proporcionadas por Haskell también han demostrado ser de gran ayuda. QuickCheck nos ha permitido probar de manera sencilla el código, encontrando situaciones que hubieran sido muy difíciles de descubrir con una batería de pruebas escrita a mano. Parsec nos ha ayudado a implementar el analizador léxico de manera compacta y legible. Es uno de los mejores ejemplos de la potencia de los lenguajes funcionales. El uso de Gtk2hs ha sido sorprendentemente sencillo y productivo. A pesar de que Gtk+ es una biblioteca escrita en C, pensada para el paradigma imperativo, se adapta perfectamente a Haskell, y su uso se nos antoja más sencillo y limpio que en C.

Antes de enfrentarnos a este proyecto, nuestra experiencia con Haskell se limitaba a pequeños programas en modo texto. El desarrollo de un programa mucho más grande nos

ha hecho descubrir nuevas técnicas y herramientas, algo que siempre resulta enriquecedor. Por tanto, y aunque sólo sea como motivo académico, el proyecto ha resultado muy satisfactorio.

Por último, sólo nos queda preguntarnos si recomendar Haskell para desarrollos futuros. La respuesta es que sí. Aunque la curva de aprendizaje inicial pueda ser muy dura para un programador acostumbrado al paradigma imperativo, adquirir soltura es sólo cuestión de práctica. El estudio de nuevos paradigmas es siempre interesante, ya que aporta nuevos enfoques a la hora de resolver problemas. Además, los lenguajes funcionales están empezando a adquirir popularidad últimamente, por lo que siempre conviene prepararse para el futuro. La única pega es que aún no disponen de herramientas y bibliotecas tan maduras como otros lenguajes.

Parte VII

Bibliografía

Bibliografía

[Aus02] R. Austein, *Tradeoffs in Domain Name System (DNS) Support for Internet Protocol version 6 (IPv6)*, RFC 3364 (Informational), August 2002.

[Cas04] Eva María Castro, *Contribución al estudio y definición de una metodología de transición gradual de ipv4 a ipv6*, 2004.

[CH97] R. Callon and D. Haskin, *Routing Aspects of IPv6 Transition*, RFC 2185 (Informational), September 1997.

[CH00] M. Crawford and C. Huitema, *DNS Extensions to Support IPv6 Address Aggregation and Renumbering*, RFC 2874 (Experimental), July 2000, Updated by RFCs 3152, 3226, 3363, 3364.

[CM01] B. Carpenter and K. Moore, *Connection of IPv6 Domains via IPv4 Clouds*, RFC 3056 (Proposed Standard), February 2001.

[Cra96] M. Crawford, *A Method for the Transmission of IPv6 Packets over Ethernet Networks*, RFC 1972 (Proposed Standard), August 1996, Obsoleted by RFC 2464.

[DFGL01] A. Durand, P. Fasano, I. Guardini, and D. Lento, *IPv6 Tunnel Broker*, RFC 3053 (Informational), January 2001.

[DH95] S. Deering and R. Hinden, *Internet Protocol, Version 6 (IPv6) Specification*, RFC 1883 (Proposed Standard), December 1995, Obsoleted by RFC 2460.

[Dra03] R. Draves, *Default Address Selection for Internet Protocol version 6 (IPv6)*, RFC 3484 (Proposed Standard), February 2003.

[HD06] R. Hinden and S. Deering, *IP Version 6 Addressing Architecture*, RFC 4291 (Draft Standard), February 2006.

[HFP98] R. Hinden, R. Fink, and J. Postel, *IPv6 Testing Address Allocation*, RFC 2471 (Historic), December 1998, Obsoleted by RFC 3701.

[HY01] J. Hagino and K. Yamamoto, *An IPv6-to-IPv4 Transport Relay Translator*, RFC 3142 (Informational), June 2001.

[Kit01] H. Kitamura, *A SOCKS-based IPv6/IPv4 Gateway Mechanism*, RFC 3089 (Informational), April 2001.

[LSK+02] S. Lee, M-K. Shin, Y-J. Kim, E. Nordmark, and A. Durand, *Dual Stack Hosts Using "Bump-in-the-API" (BIA)*, RFC 3338 (Experimental), October 2002.

[MDM96] J. McCann, S. Deering, and J. Mogul, *Path MTU Discovery for IP version 6*, RFC 1981 (Draft Standard), August 1996.

[NG05] E. Nordmark and R. Gilligan, *Basic Transition Mechanisms for IPv6 Hosts and Routers*, RFC 4213 (Proposed Standard), October 2005.

[Nor00] E. Nordmark, *Stateless IP/ICMP Translation Algorithm (SIIT)*, RFC 2765 (Proposed Standard), February 2000.

[Ric05] J. Bound L. Toutain JL. Richier, *Dual stack ipv6 dominant transition mechanism (dstm)*, October 2005.

[TGT08] F. Templin, T. Gleeson, and D. Thaler, *Intra-Site Automatic Tunnel Addressing Protocol (ISATAP)*, RFC 5214 (Informational), March 2008.

[THKS03] S. Thomson, C. Huitema, V. Ksinant, and M. Souissi, *DNS Extensions to Support IP Version 6*, RFC 3596 (Draft Standard), October 2003.

[TS00] G. Tsirtsis and P. Srisuresh, *Network Address Translation - Protocol Translation (NAT-PT)*, RFC 2766 (Historic), February 2000, Obsoleted by RFC 4966, updated by RFC 3152.