Implementación de un Router/Proxy bajo Linux

Proyecto fin de carrera

Departamento de Álgebra, área de Arquitectura y Tecnología de Computadores

Alumno: Javier Prieto Martínez

Tutor: Miguel Ángel Rodriguez Jódar

Copia cerrada el miércoles, 18 de abril de 2001

Índice

Capítulo 1: Introducción

En este capítulo se detallarán los objetivos del presente proyecto, partiendo de una descripción general de la problemática tratada hasta definir cómo se solucionará ésta en los siguientes capítulos. Hay que tener en cuenta que, aunque el sistema que se describe es completamente funcional, se pretende dar más relieve a su valor didáctico, ya que supone un repaso a los servicios de red más comúnmente utilizados. Así, aunque se profundizará en determinados conceptos, se pretende sentar una base para realizar proyectos más avanzados.

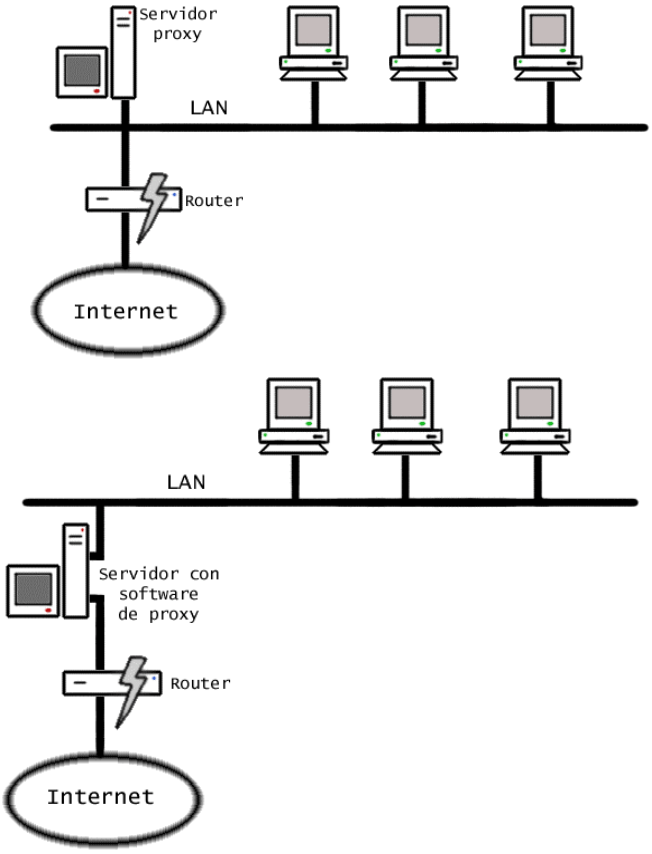
Es necesario comentar que en esta introducción se hará mención a algunos conceptos que se detallarán más adelante, dando sólo una descripción somera de su significado. Para algunos otros conceptos que se nombran pero que no forman parte de este estudio se hará referencia a la bibliografía electrónica existente al respecto. Para ello se indicará entre corchetes el código correspondiente, tal y como se haya definido en el Anexo 3, de esta manera: *[CÓDIGO]*.

1.1 – Problemática a resolver

En muchas instalaciones informáticas, tanto en empresas como en domicilios particulares, nos encontramos con el mismo problema: la conexión entre sí de varias redes distintas y heterogéneas.

Un ejemplo muy común de esto es la conexión a Internet para una red interna. En redes pequeñas o en pymes, generalmente se ha abogado en estos casos por la instalación en un PC de un modem, bien sea RTC o RDSI, y un software de proxy. Para redes mayores la conexión la suele hacer, o bien a través de un router RDSI o frame relay, o bien usando un servidor proxy dedicado conectado a un modem de cable.

En los siguientes gráficos se presentan ambas opciones. Nótese que el hecho de que un router proporcione una salida a Internet no elimina la posibilidad de existencia de servidores proxy internos que añadan otras funcionalidades, tal y como se tratará en el capítulo 4.



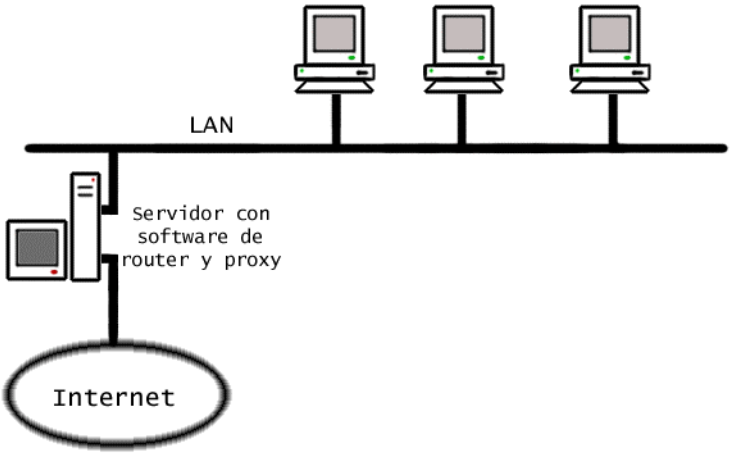
Por lo general estas propuestas son caras ya que, o bien requieren ampliar el parque informático de la empresa con nuevos equipos (bien sea un nuevo servidor o un router) o bien porque requieren la ampliación de la capacidad de proceso de los servidores preexistentes para hacerse cargo de las nuevas tareas de interconexión.

Así, se pretende desarrollar un software capaz de realizar algunas de las tareas propias de un router, con varios propósitos básicos:

1. minimizar el coste del hardware y el software necesario para ofrecer una alternativa barata a las ya existentes.
2. partir de un sistema estándar para evitar el aprendizaje de conceptos no implicadas en este estudio.

3. ser transparente al usuario, ocultando la información sobre las herramientas utilizadas y presentando un interfaz de usuario lo más simple posible.

En concreto, nuestra solución pretende aunar en un único servidor las funciones propias de un router y un proxy, posibilitando un esquema de red como el siguiente imagen. Es necesario recalcar que, aunque es posible que siga siendo necesario algún tipo de router para la conexión a internet, sería suficiente un modelo muy simple al limitarse sus funciones a dar acceso básico a un único servidor.



1.2 – Análisis de las soluciones existentes

Viendo con un poco más de profundidad las soluciones más comúnmente aceptadas nos encontramos básicamente con el espectro de productos hardware y software que detallamos a continuación. Ya que este proyecto se centra en redes de pequeño o mediano tamaño trataremos más profundamente las conexiones más económicas, aunque se mencionarán otras posibilidades como las conexiones por frame relay o satélite.

- *Software de proxy*

En el caso de que la conexión entre redes se haga a través de un equipo preexistente, ya sea a través de un modem integrado o a través de la propia red interna (como es común en la conexión a través de redes de cable), puede instalarse un software en dicho equipo para permitir la conexión a través suya de los puestos de trabajo cliente.

Ya que el proxy trabaja en el nivel de aplicación, no todas las comunicaciones son transparentes y necesita estar adaptado a cada servicio (telnet, web, FTP, etc.) que quiera proporcionar. Por tanto, un proxy no es tanto una solución completa a las necesidades de conectividad como un servicio añadido a otros que nos podamos encontrar.

En principio es una solución barata, pero con el tiempo conlleva el coste oculto de la ampliación del servidor para soportar la carga que conlleva el tratar todos los paquetes que circulan por la red.

Para redes Windows un ejemplo típico sería Microsoft Proxy Server para Windows NT 4.0 *[MS-PROXY]*, o más recientemente, Microsoft ISA Server *[MS-ISA]* para Windows 2000. Otras soluciones comúnmente utilizadas son programas shareware[1] como Wingate *[WINGATE]*, de Deerfield.com. Podemos encontrar en la red índices de otros productos con esta funcionalidad como el de Tucows *[TUCOWS]*.

- *Router RDSI*

Es la solución más aplicada en el ámbito empresarial. Suelen contar con una conexión de cable de red coaxial o de par trenzado para conectarse a la red local y una conexión RDSI para el acceso a Internet o a otro router de similares características.

Como todas las soluciones basadas en routers tiene la ventaja de que no requiere prácticamente modificar la red preexistente (en principio sólo haría falta configurar los equipos clientes para que tengan la dirección IP del router como ruta por defecto) y de que su instalación y configuración suele ser rápida y sencilla. Su principal desventaja radica en su precio, ya que por lo general superan el precio de un equipo de gama media que podría cubrir igualmente las funciones de enrutado.

En España nos encontramos con un mercado dominado principalmente por Cisco Systems *[CISCO]*, aunque los productos de otras compañías como 3Com *[3COM]* (en la gama baja) o Cabletron *[CABLETRON]* (en la gama alta) son también ampliamente utilizados.

[1]Se denomina así a los productos software cuya licencia permite un periodo de evaluación con funcionalidad limitada anterior a la compra del producto

- *Router interno*

Si la finalidad del enrutado es la conexión de dos redes de área local o bien si la conexión a Internet es proporcionada directamente por la red interna, es necesario un router con dos o más conexiones de red local. Excepto por este detalle la solución es similar a la anterior.

Este tipo de sistemas es muy común en redes locales empresariales con un gran número de máquinas o en entornos donde se ha migrado de un tipo de cableado Ethernet a otro (en este caso tendríamos conexiones tanto BNC o AUI como RJ45). En el mundo empresarial nos encontramos principalmente con los mismos fabricantes que cuando hablamos de los routers RDSI.

- *Otras configuraciones*

Actualmente existen una gran variedad de posibilidades en el acceso a Internet, pero todas ellas se pueden cubrir de una u otra forma según los puntos anteriores. Así, un acceso a una red de cable o DSL[2] puede realizarse a través de un router interno (si la conexión proporcionada por el suministrador de acceso es de tipo Ethernet, como es el caso de la red de cable de Supercable) o a través de un software de proxy si el acceso lo suministra un servidor con hardware adicional (como ocurre con los modem USB que proporciona Telefónica para el acceso por ADSL[3]).

[2] Acceso a través de la red telefónica básica a altas frecuencia

[3] Conexión DSL asimétrica (con un cauce de bajada de ancho de banda distinto al de subida)

1.3 – Capacidades de enrutado en los entornos más comunes

A continuación se detallan las principales características técnicas y funcionalidades de los Sistemas Operativos más comúnmente utilizados, no considerando aquellas que sean proporcionadas por software no incluido con el sistema básico.

- *Windows 95/98*

Actualmente es el sistema operativo más utilizado en el entorno doméstico y de pequeñas empresas. Aunque tiene muy pocas capacidades como servidor de red (de hecho es un sistema pensado única y exclusivamente como cliente), la gran cantidad de servidores proxy que corren sobre él ha hecho que en muchas ocasiones se utilice para compartir un acceso a la red.

Ciñéndonos sólo a las facilidades que proporciona el sistema, sólo la versión Windows 98 Second Edition ha posibilitado compartir la conexión a Internet para una serie de máquinas. Esto se realiza, por una parte, instalando un servidor DHCP en la máquina y, por la otra, enmascarando las direcciones internas que proporciona por la dirección pública correspondiente (de manera similar a como describiremos en el capítulo 3). Ninguna de estas opciones es, en cualquier caso, configurable, ya que todo el proceso se realiza de manera automática por parte del sistema.

Exceptuando este punto, estos sistemas operativos no proporcionan ninguna de las demás características que precisamos, así como tampoco facilita un sistema de desarrollo más complejo que los scripts[4] del intérprete de comandos, a todas luces insuficientes. En cualquier caso, Windows 95/98 será el sistema operativo cliente de referencia para el servidor que estamos desarrollando.

- *Windows 2000 Server*

Planteado como sistemas operativo servidores, sí supone una opción a considerar como vía para obtener los servicios propuestos. No se ha tratado Windows NT ya que actualmente se encuentra descatalogado a favor la versión más reciente. De todas formas, aunque Windows 2000 ha supuesto una gran innovación con respecto a Windows NT en muchos aspectos, en los que conciernen a este estudio se ha mantenido una funcionalidad muy similar.

En concreto, como parte integrante del sistema, nos encontramos con la posibilidad de enrutar entre interfaces de red, incluyendo las mismas posibilidades que nos ofrecía Windows 98 Second Edition pero añadiendo una completa configuración gráfica. También disponemos de un servidor de direcciones dinámicas por DHCP, de un servidor DNS y de algunos de los servicios que se incluyen como referencia a futuras aplicaciones.

La principal carencia que podemos encontrar es que el software de proxy no se encuentra disponible directamente, sino como un producto externo que hay que comparar aparte. Tampoco encontramos un sistema de desarrollo potente como para hacer el interfaz de acceso a la máquina y, aunque compráramos

[4] Se llaman así a las líneas de código interpretadas por el mismo interfaz de comandos del sistema operativo.

los productos necesarios, enmascarar el acceso a la máquina para ocultar información innecesaria requeriría trabajar a muy bajo nivel con el sistema.

Otra carencia que podemos encontrar en este sistema operativo es su ligadura a la arquitectura Intel, ya que Windows NT 4.0 fue el último sistema operativo para servidores que Microsoft portó a otras arquitecturas. En cualquier caso, al ser la arquitectura Intel la más extendida y barata del mercado, no podemos considerar esto como un gran problema.

- *GNU/Linux*

Como alternativa al software comercial, en los años 80, una comunidad de desarrolladores, encabezados por Richard Stallman, decidieron ofrecer al mundo el código fuente de sus desarrollos, iniciando lo que se dio a llamar el proyecto GNU *[GNU]*. En palabras del propio Stallman, en su documento "Por Qué El Software No Debe Tener Propietarios" *[STALLMAN-1994]*, podemos leer: *"Te mereces ser capaz de cooperar abierta y libremente con otras personas que usan software. Te mereces ser capaz de aprender cómo funciona el software, y enseñar a tus estudiantes con él. Te mereces ser capaz de contratar a tu programador favorito para arreglarlo cuando se rompa. Te mereces software libre"*.

De esta filosofía nació Linux. Este sistema operativo, nacido en las universidades de todo el mundo, se caracteriza por distribuirse con licencia GPL[5], lo cual implica que cualquier usuario, por el hecho de serlo, se convierte un en desarrollador en potencia para el sistema. Al crearse esta licencia para ser perpetua, ya que los programas nacidos de otro libre deben continuar siéndolo, se eliminó la posibilidad de que un código nacido para ser compartido acabe formando parte de un proyecto comercial.

Actualmente es la punta de lanza de la filosofía del software libre, ya que, aunque Linux en sí mismo es solo un núcleo de sistema basado en Unix, se apoya en una gran cantidad de software de libre distribución diseñado, en gran parte, específicamente para él.

Esto le confiere una gran potencia, ya que cuando los usuarios encuentran carencias en el sistema o problemas en la configuración tienen todo tipo de facilidades para resolver problemas que, de otra manera, requerirían buscar software adicional o confiar en una actualización de sistema por el fabricante. De cualquier forma, esto es un arma de doble filo, ya que muchas veces el esfuerzo de desarrollar una solución a un problema informático que no la tenía no compensa el desembolso de comprar un producto comercial que, además, asegure una estabilidad en el soporte técnico y en el desarrollo de nuevas funcionalidades.

Otra de las peculiaridades del software libre, que le confiere una gran ventaja en este caso, es que por lo general se encuentra muy optimizado en recursos del sistema. Esto es causa directa de la disponibilidad de su código fuente ya que así podemos adaptar la versión genérica a nuestras necesidades. Por otro lado evita que, por motivos de marketing, el fabricante limite la funcionalidad del software.

Hablando más concretamente de Linux, éste se distribuye en forma de distribuciones. Éstas consisten en un núcleo de Linux arropado por un conjunto determinado de software, con un interfaz determinado para instalación y configuración del sistema. Las principales que podemos encontrar son Debian *[DEBIAN]*,

[5] General Public License: licencia de código abierto nacida para la distribución de software libre

gratuita y basada única y exclusivamente en software libre, y RedHat *[REDHAT]*, comercial y apoyada además en software de pago o de licencias más restrictivas. Basadas en mayor o menor medida en estas distribuciones actualmente encontramos otras muy extendidas como SUSE *[SUSE]*, TurboLinux *[TURBOLINUX]* o Slackware *[SLACKWARE]*, por citar algunas.

1.4 – Descripción general de la solución propuesta

En nuestro caso, elegiremos la opción GNU/Linux, ya que la problemática expuesta ha sido ampliamente solucionada con programas estables y de total confiabilidad. En posteriores capítulos veremos sólo un pequeño porcentaje de la potencia que el núcleo de Linux y el software GPL nos proporcionan, obteniendo aún así toda la funcionalidad exigida.

Además, y sin entrar en luchas ideológicas entre software libre y comercial, hay que considerar que, usando Windows en cualquiera de sus versiones, necesitaríamos de programas externos para poder realizar este proyecto, comenzando por un entorno de desarrollo que en ningún caso se proporciona. Sumando el coste de la máquina al del software necesario nos encontraríamos en un rango de precios similar al de las soluciones hardware ya existentes en el mercado. Usando software libre, en cambio, sólo necesitamos un hardware que incluso puede ser menos potente que el necesario para tareas básicas de ofimática.

Aprovechando el amplio desarrollo del software libre podemos tomar las posibilidades que ofrece un núcleo Linux, apoyado por programas de libre distribución, para realizar las mismas funciones que un router hardware tradicional. En este caso diseñaremos un router interno, con una dirección interna fija y otra externa automática, orientado a unir una red local a una red de cable a través de un modem cable. Además, se añadirá el servicio de proxy para minimizar el tráfico a través de Internet que se pueda generar.

Concretamente, utilizaremos una distribución Turbolinux Server 6.0, actualizada con la versión 2.2.14 del núcleo del sistema. Se ha apostado por esta versión en lugar de una de la serie la 2.4 por estar mucho más probadas sus capacidades como servidor de red, y sobre todo debido a que el uso del nuevo kernel implicaría el uso de un nuevo sistema de enmascaramiento de IPs, a través de la utilidad "`iptables`", mucho menos utilizado hasta la fecha. La plataforma usada será Intel, por ser la que más software tiene disponible.

Continuando con la idea de sustituir a un router tradicional, se diseñará un programa de configuración en modo texto, accesible tanto por telnet[6] como por puerto serie (esto último es necesario para la configuración inicial, ya que el acceso por Ethernet sólo sería posible tras configurar su dirección IP), que pueda modificar todos los parámetros necesarios.

Como lenguaje de desarrollo se ha elegido C por su universalidad y porque gran parte del núcleo de Linux y de los programas GPL utilizados lo utilizan. Además, así se facilita la portabilidad del software entre las múltiples plataformas para las que el núcleo está disponible.

Por último, es interesante recalcar que he participado, como integrante de la empresa Optima Technologies S.L *[OPTIMA]*, en el desarrollo de un producto llamado G.A.N.S.O.[7], basado en este proyecto. Esta solución fue diseñada como una alternativa barata a los routers empleados para el acceso de las PYME a las redes de cable, y actualmente es utilizado por varios de sus principales clientes. En

[6] Protocolo de comunicación en modo texto a servidores de terminales Unix, actualmente usado como medio de acceso de múltiples elementos incluyendo elementos de red.

[7] Acrónimo de General Access Network Server for the Office

concreto, la versión que se describe en este documento corresponde a la versión 1.0 del producto, liberada en el año 1999, eliminando de ésta el sistema de claves y el de acceso por web, entre otros, por haber intervenido en ellos otras personas y por motivos de seguridad.

Capítulo 2 – Direccionamiento IP

No se sabría decir si el nacimiento de Internet extendió el uso de TCP/IP al nivel que está ahora o si esto ocurrió al revés, pero es indudable que éste es ahora mismo el protocolo de red más usado. Para describirlo la mejor referencia sobre la que podemos trabajar es la estructura de capas definida por el estándar ISO/OSI, que representa con varios niveles de abstracción, la comunicación entre dos máquinas.

7	Aplicación
6	Presentación
5	Sesión
4	Transporte
3	Red
2	Enlace de Datos
1	Físico

Así tenemos desde el nivel físico, en que trataríamos aspectos como el tipo de cables usados, los valores eléctricos implicados en la transmisión (impedancias y voltajes de cada tipo de señal, valores de resistencias necesarias para la correcta transmisión, ruidos en la línea, etc.) hasta el nivel de aplicación, donde trataríamos los protocolos implicados (FTP, HTTP, Netbios, IPX, etc.) a alto nivel, abstrayéndonos de todos los conceptos implicados en los niveles inferiores. Aunque con cualquiera de ellos podemos describir perfectamente el proceso de comunicación entre dos máquinas, para cada uno de ellos hay definidos una serie de protocolos que realizan el proceso de una u otra manera.

La instancia concreta más extendida de esta modelo es el estándar TCP/IP. Basándose en cualquier protocolo físico (coaxial, par trenzado, fibra óptica, etc.) y de red (802.2, CSMA/CD, X.25, etc.), define como protocolo de nivel 3 el IP (Internet Protocol), así como los protocolos TCP y UDP a nivel 4. Los niveles superiores se unen en un nivel genérico “Aplicación”.

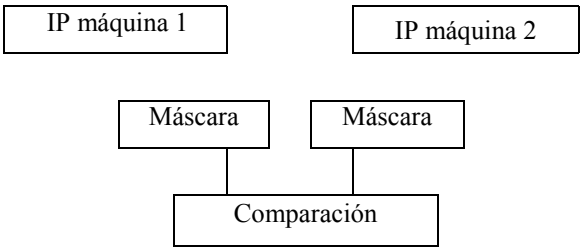
	ISO / OSI	TCP / IP
7	Aplicación	Aplicación
6	Presentación	
5	Sesión	
4	Transporte	Transporte (TCP / UDP)
3	Red	Internet (IP)
2	Enlace de Datos	Red
1	Físico	Físico

Como protocolo más usado en el nivel de enlace de datos, el protocolo Ethernet[8] define una identificación única de interfaces de red para cada máquina según un número único de 48 bits, que cada fabricante certifica como único en el mundo.

[8] tipo de redes con estructura de bus, que actualmente constituyen la principal base instalada de redes de área local

Subiendo un peldaño más, a nivel de red nos encontramos con el protocolo IP (*internet protocol*), que define una identificación única en la red para cada servicio (un mismo interfaz de red puede tener más de una dirección IP asignada). Esta identificación consiste en un número de 32 bits (normalmente representado por 4 octetos) que el administrador de redes debe asignar para cada máquina según la estructura de su red.

Junto con la dirección IP se asigna otro número de 32 bits, llamado máscara de subred, que identifica qué grupo de direcciones se considerarán locales a la máquina y cuáles no. Para comprobar si una dirección y otra están dentro de la misma subred (subconjunto de direcciones dentro de una misma red) se aplica un AND lógico entre ambas direcciones con su máscara de subred, resultando ambas compatibles si el número resultante es el mismo.



Generalmente las máscaras de red están formadas por un grupo de unos a la izquierda seguido de un grupo de ceros a la derecha, de tal manera de que para reflejar una subred con 2^n máquinas tomaríamos como máscara de red el número binario resultante de unir (32-n) unos con n ceros (o dicho de otra manera, el número decimal 2^{32}-2^n).

Por ejemplo, para una subred de 2^8 (es decir, 256) máquinas tomaríamos como máscara de subred el número binario 11111111 11111111 11111111 00000000, que se representaría agrupado en octetos como 255.255.255.0 o, abreviadamente, como 24 (correspondiente al número de unos). Así, cuando hablamos de la subred 192.168.200.0/24 hablamos de una subred que comprende todas las máquinas cuyos tres primeros octetos son 192.168.200.0, es decir, desde la dirección 192.168.200.0 a la dirección 192.168.200.255.

Cuando un paquete Ethernet llega a una máquina que utilice el protocolo TCP/IP, ésta comparará la dirección de destino para saber si el paquete está o dirigido a ella. Usualmente, además, se toma la última dirección IP de cada subred (en el ejemplo anterior, la 192.168.200.255) como dirección de broadcast, de manera que los paquetes que se manden con ese destino serán leídos por todas las tarjetas que tengan una dirección IP asignada dentro de esta subred.

El dividir la red en subredes no imposibilita a las máquinas pertenecientes a ámbitos distintos el comunicarse entre sí, pero hace necesario definir entre ambas redes un sistema de enrutado, tal y como se verá en el capítulo siguiente. El problema que acarrea el posibilitar que los paquetes circulen entre subredes conocidas y desconocidas (normalmente, entre nuestra LAN[9] e Internet) es que podemos encontrar el caso de que subredes distintas controladas por distintos administradores hayan elegido un mismo sistema de direccionamiento.

Para solucionar este problema en el caso específico de Internet se definen una serie de subredes que nunca se asignan a máquinas públicas, de manera que pueden usarse en una LAN sin peligro de que al

[9] Redes de área local, en contraposición a las WAN o redes de área extensa. Se llaman así a las redes que cubren poco espacio, normalmente dentro del mismo edificio u oficina.

conectarla en Internet se confundan las direcciones de ambos extremos. Las direcciones asignadas a estas subredes son 10.0.0.0/8, 172.16.0.0/12 y 192.168.0.0/16, aunque normalmente no se utilizan estos rangos enteros, sino otros de tipo 10.x.0.0/16 o 192.168.x.0/24 (donde 'x' es cualquier valor), para dividirlos algo más en otros más pequeños y obtener así mayor granularidad.

Una vez asignadas direcciones IPs a los interfaces de red, y subiendo un paso en la escala OSI, es necesario asignar números de puerto TCP o UDP a los servicios concretos. Así, dentro de una máquina, podemos diferenciar los servicios por un identificador al que se conectan las máquinas externas. De esta manera, conectando la máquina local al puerto 80 de otra máquina, sabemos que estamos haciendo una conexión para una aplicación concreta (en ete caso, y por convenio, para el servicio de WWW).

Se mantienen en esta capa dos protocolos distintos por tener cada uno capacidades que se adaptan a distintos tipos de necesidades, aunque tengan características comunes como lo es el hecho de dividir los datos en paquetes de tamaños fijos que se mandan en secuencia. Como principal característica del protocolo TCP podemos señalar que está orientado a conexión, lo cual implica que se establece un canal de transmisión de datos virtual entre ambas máquinas antes de transmitir los datos, y se garantiza que todos los datos se han recibido correctamente y ordenados. UDP, en cambio, no establece una conexión para el envío del mensaje ni garantiza que los paquetes que forman el mensaje lleguen en el orden correcto o que no se formen duplicados.

Aunque usualmente las aplicaciones utilizan paquetes TCP, eligieramos paquetes UDP, cuando necesitemos una alta velocidad de transmisión de datos y cuando las pérdidas ocasionales de paquetes puedan ser tratadas por la aplicación. Así, la mayoría de los videojuegos utilizan este tipo de paquetes ya que el aprovechamiento del ancho de banda de la red es crítico y los ordenadores clientes tienen capacidades de proceso suficientes como para analizar el tráfico requerido, reordenando los paquetes y decidiendo en cuáles es necesario el reenvío de las partes erróneas.

2.1 - Direccionamiento dinámico

Dependiendo del tamaño de nuestra red podemos encontrarnos que la asignación de direcciones IP se convierte en un problema. Si bien en necesario que los servidores sean siempre accesibles por la misma dirección IP (aunque exista un DNS que los asocie a un mnemónico, puede haber servicios cliente que utilicen direcciones numéricas), los puestos clientes, al no ofrecer servicios al resto de la red, pueden cambiar de dirección sin problemas.

El hecho de que el administrador de la red no tenga que conocer las direcciones de cada máquina simplifica la administración de esta, especialmente en redes muy grandes en número de máquinas o en extensión geográfica. Localizar la máquina que corresponda a una dirección es en cualquier caso posible consultando las tablas de asignación de direcciones de los servidores encargados de asignarlas.

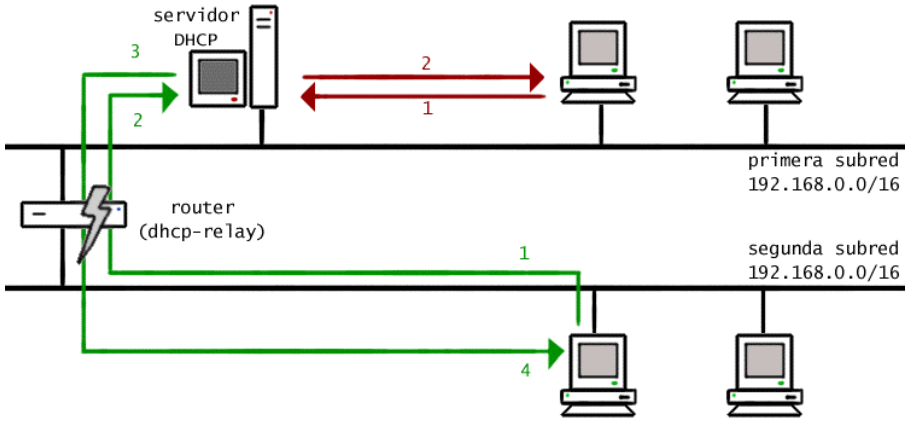
El protocolo que hace esto posible es el DHCP (Dinamic Host Configuration Protocol), a través del cual seremos capaces de asignar direcciones bajo demanda a los puestos de trabajo. Este protocolo se encarga de asignar a cada máquina una dirección dentro del rango de direcciones soportadas, siendo, de ser posible, siempre la misma. Además de esta dirección, por DHCP se puede recibir cualquier otro tipo de dato, como la ruta por defecto o el servidor DNS.

De cara al servidor, se puede definir el tiempo de duración de las asignaciones de estas direcciones, de manera que el cliente deba pedir una nueva dirección IP transcurrido ese tiempo. De cara al cliente, en cada asignación o reasignación se puede solicitar un nombre y una dirección concretos, de manera que de ser posibles, se le asignará como nombre el identificativo de la máquina en el sistema operativo, y como dirección, la misma que se le asignó la última vez (o la primera libre si esa máquina nunca ha tendio una dirección válida).

Igualmente, a través de este protocolo podemos transmitir a los clientes cualquier tipo de información. Los parámetros estándar incluyen, además de la dirección IP y la máscara de subred del adaptador cliente, otros parámetros necesarios para definir la conectividad de éste, tales como la ruta por defecto o la dirección IP de los servidores DNS o WINS[10]. Además podemos añadir cualquier otro parámetro definido por el usuario para propósitos específicos. Estos parámetros pueden ser asociados a todas las direcciones, o bien ser específicos para un rango de direcciones o para una máquina en concreto.

El principal problemas que nos podemos encontrar en la implantación de este protocolo es el hecho de que actúa en base a paquetes broadcast, con lo cual no es enrutable entre subredes. Para ello se utilizan máquinas con la función de DHCP-relay, capaces de reenrutar las peticiones DHCP de los clientes de su subred al servidor que centralice los servicios de direccionamiento dinámico en la red. Este servidor deberá tener distintos rangos de direcciones para cada subred a la que de servicio.

[10] Protocolo similar al DNS para asociar direcciones IP a nombres del protocolo NetBios, propio de redes Windows



Según el gráfico anterior, cuando un cliente de la primera subred (en la parte superior del esquema) arranque, ocurrirá lo siguiente:

1. El cliente hace un broadcast pidiendo una dirección IP a algún servidor de su subred
2. El servidor, según la dirección hardware del cliente, le asigna una dirección IP del rango 1

Por otra parte, en la segunda subred el proceso es este:

1. El cliente hace la petición broadcast
2. El router recibe la petición y la reenvía al servidor DHCP indicado en su configuración
3. El servidor DCHP responde con una dirección del rango 2
4. El router la reenvía al cliente

En nuestro caso, y para el servidor que estudiamos, consideramos que internamente (a través del interfaz /dev/eth0) es accesible por una dirección estática (lo cual es necesario por su condición de servidor). Externamente (en el interfaz /dev/eth1) tendrá una dirección dinámica asignada por DHCP, a traves de la cual también recibirá la dirección del servidor DNS del que actuará como replica.

2.2 – Direccionamiento estático en Linux: ifconfig

En Linux, como en cualquier otro sistema tipo Unix, encontramos los interfaces de red como dispositivos dentro del directorio /dev. En el caso de tarjetas de red Ethernet estándar (el caso estudiado), el único dispositivo con IP estática será el asociado a /dev/eth0.

En el arranque de la máquina debemos ser capaces de activar (en argot, *"levantar"*) dicho interfaz asociándolo a una dirección IP determinada y con una serie de parámetros. Para ello usaremos el comando *"ifconfig"*, que tiene la siguiente sintaxis:

```
ifconfig {interfaz} [{familia} {opciones | dirección}]
```

En este caso no será necesario especificar una familia de interfaces, pero de tratarse de un protocolo distinto a IPv4[11], valor que será tomado por defecto.

Como principales opciones usaremos:

- up: activa el interfaz de red especificada
- down: desactiva el interfaz.
- metric: indica el coste que acarrea la trasmisión de paquetes por el interfaz[12].
- netmask {dirección}: indica la máscara de red del interfaz.
- broadcast {dirección}: indica la dirección de broadcast de interfaz.
- {dirección}: indica la dirección a asignar al interfaz.

Primero inicializaremos la tarjeta de red con la dirección IP indicada (la máscara de red y la dirección de broadcast en principio no serían necesarias, pero se suelen por compatibilidad con redes con direccionamientos no estándar). En principio la máscara de subred es más complicada de deducir, teniendo en cuenta que para direcciones no enrutables de tipo 192.168.x.x o 10.x.x.x es muy predecible, pero en una dirección válida en internet depende en gran medida de el proveedor de servicios que la proporcione. La dirección de broadcast, en principio, es siempre la dirección más alta de la subred, pero puede interesar cambiarla para intereses concretos.

Tras inicializar el interfaz no tenemos rutas por la que enviar los paquetes que se dirigen a la red interna, así que debemos usar el comando "*route*" para crearla. Este comando permite modificar la tabla de rutas del sistema con la siguiente sintaxis (se indican sólo los comandos para añadir rutas):

```
route add {destino} [-net|-host] [netmask {máscara}] [gw {dirección}] [metric {métrica}] [dev {interfaz}]
```

Las opciones que usaremos serán:

- [-net | -host]: indica si el destino es una máquina o una red completa.
- netmask {máscara}: indica la máscara de subred del destino.
- gw {dirección}: indica la dirección a la que se enrutarán los paquetes que se dirijan al destino.
- metric {métrica}: indica el coste del interfaz.
- dev {interfaz}: asocia esta ruta a un interfaz concreto.

[11] Versión 4 de protocolo IP, actualmente la más usada, pese a la existencia de la versión 6.

[12] En el caso de destinos alcanzable por varias reglas de la tabla de enrutado, se escogerá la que acarree menor coste.

En concreto, con el siguiente comando indicaremos al sistema que todos los paquetes que tengan como destino la subred interna se encaminen a través del interfaz eth0:

```
route add –net {dirección de red interna} eth0
```

Una vez hecho esto sólo falta por inicializar el segunto interfaz por DHCP, tal y como vemos en el siguiente apartado, para disponer de total conectividad.

2.3 – Direccionamiento dinámico en Linux: dhclient

En nuestro caso usaremos el cliente *dclient* del Internet Software Consortium en su versión 2.0[13] *[DHCPCD]*. Este cliente está incluido en un paquete completo que incluye un cliente DHCP, un servidor de direcciones dinámicas, y un servidor de relay.

El cliente se invoca con la siguiente línea de comandos:

```
dhclient [-d] {interfaces}
```

- -d: indica que el programa correrá en primer plano (útil para hacer pruebas).
- interfaces: lista de interface de red que deseamos configurar por DHCP.

Para cada interfaz se leerá el fichero `/etc/dhclient.conf` buscando instrucciones específicas de configuración. De esta manera se puede definir qué parámetros pedirá el cliente al servidor como preferentes, aunque luego se aplicará la política de concesión de direcciones del servidor, que decidirá si se le darán esos u otros valores.

Una vez realizada la asignación de direcciones, en el fichero `/var/dhcp/dhclient.leases` se almacenan los datos obtenidos, que luego tomaremos para que sean visualizados en el programa de configuración.

[13] Actualmente existe una versión 3.0, pero se encuentra aún en fase Beta.

2.4 – Implementacion en Turbolinux Server 6.0

Para automatizar estos cambios (y hacer, así, que no sea necesario volver a teclearlos al iniciar cada sesión, la distribución Turbolinux propone definir una configuración estándar en el directorio "*/etc/sysconfig/network-scripts*". La configuración existente aquí se leerá desde los scripts correspondientes al nivel de arranque usado de manera que la red se configurará o desconfigurará automáticamente.

En concreto, para la configuración de red nos encontramos con una serie de ficheros "ifcfg-{interfaz}" que define los parámetros para cada interfaz de red. Cada uno de estos ficheros contiene las siguientes líneas:

- DEVICE={dispositivo}: nombre de dispositivo a configurar (debe coincidir con el indicado en el nombre de fichero)
- IPADDR={dirección}: direción IP.
- NETMASK={máscara}: máscara de subred.
- NETWORK={dirección}: dirección de la red.
- BROADCAST={dirección}: dirección de broadcast.
- ONBOOY={yes|no}: activación automática de la confiugración.
- BOOTPROTO={none|dhcp}: método de autoconfiguración ("none" para direcciónamiento estático y "dhcp" para direccionamiento dinámico). Para configuraciones por DHCP, se ignoran todos los parámetros y se obtienen del servidor de red.

Una vez definida la configuración en estos ficheros podemos utilizar los comandos "ifup {interfaz}" e "ifdown {interfaz}" para levantar o bajar los servicios de red asociados a éstos. En cualquier caso, los scripts preconfigurados se encargarán de arrancar o bajar los servicios automáticamente en el incio o la parada de la máquina.

Capítulo 3 – Enrutado y cortafuegos

El direccionamiento IP, tal y como se define en el capítulo anterior, limita el tráfico de paquetes en la red a una subred en concreto. Cuando surge la necesidad de conectar las subredes entre sí (lo que llamamos enrutar), también nace la preocupación por la seguridad de unos datos que ahora no sólo se intercambian necesariamente entre clientes próximos entre sí.

El concepto de enrutado entre redes es simple: cada máquina tiene definida una tabla de rutas donde indica a través de que pasarela encamina los paquetes según su destino. Para las subredes asociadas a interfaces de red conectados a la máquina, se indica que la pasarela es la propia tarjeta de red, ya que dichas direcciones son alcanzables directamente. Normalmente una de estas reglas corresponde a la subred 0.0.0.0/0 (es decir, a cualquier máquina), e indica qué router es capaz de tratar con cualquier dirección no conocida (normalmente, con internet).

Cuando tratamos con configuraciones configuraciones complejas, con muchos routers que encaminan las peticiones de los clientes, es necesario un mecanismo que intercambie las tablas de enrutado ellos. Así, existen varios protocolos de comunicación entre routers que evitan la necesidad de reconfigurar a mano las tablas de rutas de todos ellos después de cualquier cambio en la red. Los más comunmente usados son RIP e IGRP.

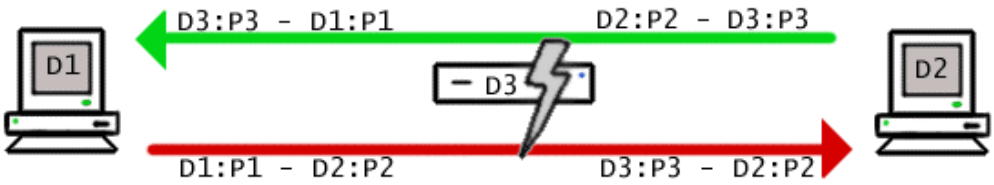
Junto con las funcionalidades de conectividad que proporcionan los routers surge el problema de la seguridad informática. Si nuestros datos pueden transpasar la barrera de nuestra máscara de red para permitir la comunicación externa, puede que sean accedidos por quien no debería. Para evitar esto se añaden en la red servicios de cortafuegos que, o bien se integran en los mismos routers, o se instalan en máquinas independientes. Estos servidores controlan el tráfico que circula por ellos para decidir si debe o no producirse la transmisión.

3.1 -Enrutado, mapeo estático de puertos y NAT

La principal función que realiza el servidor estudiado es la de enrutador. Esto es, será capaz de las dos subredes a las que hemos llamado externa e interna (normalmente Internet y nuestra intranet). Cuando se de el caso de que la subred interna usa direcciones IP no enrutables, hay que idear un mecanismo para que sea posible el acceso a internet, ya que las máquinas externas no serán capaces de por sí de encontrar nuestras máquinas. En este caso, además de enrutar los paquetes, dado que en la red externa sólo disponemos de una dirección IP válida, se enmascararán todas las direcciones internas por una dicha dirección. Este proceso se denomina NAT, acrónimo de Network Address Translation (traducción de direcciones de red).

El proceso completo se puede resumir en los siguientes pasos:

- Una máquina interna accede desde una dirección D1 por un puerto P1 a una dirección externa D2 por un puerto P2.
- El router, con dirección externa D3, enmascara la petición como si la hubiera hecho él desde un puerto P3 que genera dinámicamente.
- La máquina externa ve una petición desde D3:P3 a D2:P2 y responde.
- El router vuelve a encaminar la petición de D3:P3 al origen D1:P1



En realidad el proceso no es tan simple, ya que el router dispone de dos direcciones de red (una en cada subred), con lo cual realmente la dirección D3 ve la máquina interna no corresponde con la que ve la máquina externa. El puerto que se ve externamente sí es el mismo, independientemente del interfaz de red por el que se acceda al router.

Con esto conseguimos que máquinas de ambas redes, con máscaras de red incompatibles, sean accesibles entre sí a través de una IP puente entre ambas. Hay que tener en cuenta que de esta manera todos los ordenadores de la subred interna son visibles externamente a través de una sola dirección IP, sin ser accesibles por la real. Así, no es posible que se inicien conexiones desde el exterior hacia las máquinas internas, ya que sus direcciones reales no son enrutadas desde Internet.

Este problema, por otra parte, es una ventaja en seguridad, al estar seguros por completo de que no habrá accesos externos hacia nuestra red. En cualquier caso, para permitir acceso externo a estas máquinas (lo que en Linux se llama *reverse-masquerading*), es posible hacer mapeos de puertos de manera que determinados puertos concretos de éstas sean accesibles a traves de otros del router. De esta manera, asignando que el puerto 25 del router redireccione al mismo puerto del servidor de correo interno (correspondiente al protocolo SMTP[14]), el correo electrónico llegará a nuestra organización a través de su IP externa de manera transparente a través del router, aunque realmente lo procese un servidor interno.

[14] Simple Mail Transfer Protocol: protocolo estándar para el envío de correos electrónicos por Internet.

3.2 – Enrutado y NAT en Linux: ipchains

En el caso concreto de Linux, la funcionalidad de NAT y de cortafuegos la realizan entre el propio núcleo del sistema y un programa llamado *ipchains [IPCHAINS]*. Esto es cierto para todos los núcleos de la serie 2.2, ya que en versiones anteriores se adoptó una solución muy similar a la ofrecida por BSD, mientras que en la versión 2.4 varía radicalmente el soporte de firewall ofreciendo nuevas funcionalidades.

En concreto, ipchains funciona dividiendo los paquetes entre tres listas de reglas o cadenas, denominadas input, output y forward (entrada, salida y redirección). Al llegar al cortafuegos se procesan por las reglas de la cadena de entrada, para después ser tratados por la cadena de redirección si estaban destinados a otra máquina. Por último, cuando abanandonan el cortafuegos por algún interfaz de red, independientemente de que hayan pasado o no por reglas de redirección, los paquetes son procesados por la cadena de salida. Como un caso especial de reglas de redirección están las reglas de *masquerading* (enmascaramiento), nombre que se le da en Linux al proceso de conversión de direcciones o NAT.

En cualquiera de estos pasos puede decidirse rechazar el paquete o permitir su paso a la siguiente cadena. En cada paso, las reglas pertenecientes a la cadena se procesan en orden hasta encontrar una que sea aplicable al paquete. Si ninguna lo es, se aplica la política por defecto para esa cadena. También es remarcable el hecho de que podemos crear nuestras propias cadenas de reglas para organizarlas más fácilmente.

En nuestro caso, ya que no vamos a desarrollar un firewall completo, simplemente un router con soporte de NAT, sólo activaremos las reglas de enmascaramiento de direcciones, dejando pasar de manera transparente a todos los paquetes por las reglas de entrada y salida.

Para que funcionen correctamente los comandos que se describirán a continuación debemos configurar las siguientes opciones en el núcleo del sistema:

- `CONFIG_FIREWALL=y`
- `CONFIG_IP_FIREWALL=y`
- `CONFIG_IP_ADVANCED_ROUTER=y`
- `CONFIG_IP_MASQUERADE=y`
- `CONFIG_IP_MASQUERADE_ICMP=y`
- `CONFIG_IP_MASQUERADE_IPPORTFW=y`

Por último, para activar el soporte de NAT en el núcleo, tras activar las tarjetas de red debemos escribir "1" dentro del fichero virtual `/proc/sys/net/ipv4/ip_forward`.

Una vez hecho esto, con el comando ipchains podemos definir las cadenas de reglas. Dada la complejidad de uso de este comando, sólo se describen a continuación las órdenes concretas que activan el soporte de enmascaramiento de Ips:

- `ipchains -P {input|output|forward} {ALLOW|DENY}`: admite o deniega por defecto el paso de paquetes por una determinada cadena, a no ser que se añadan reglas específicas para ellos. En nuestro caso, y por defecto, activaremos el paso por todas las cadenas menos por la de redirección.

- `ipchains -A {input|output|forward} [-j MASQ] -s {subred origen} -d {subred destino} -i {interfaz}` : añade una regla para paquetes que pasen por la cadena

especificada, con el origen y el destino indicados, y utilizando la interfaz de red descrita. En caso de tratarse de una regla para la cadena de redirección, podemos añadir la opción "`-j MASQ`" para indicar que, además de permitir el paso por dicha cadena, se activará el soporte de NAT.

- `ipchains –D {input|output|forward} [–j MASQ] –s {subred origen} –d {subred destino} –i {interfaz}` : elimina una regla creada con los parámetros indicados.

En nuestro caso, y para la distribución elegida, habremos de modificar el fichero "`/etc/sysconfig/ipchains.rules`". En este fichero se definen las cadenas a usar con sus reglas por defecto, así como las reglas a aplicarles, con la misma sintaxis que usamos con el comando "`ipchains`". El fichero que generará el programa de configuración será el siguiente:

```
:input ACCEPT
:forward DENY
:output ACCEPT
-A forward -j MASQ -s {subred}/{máscara} d 0.0.0.0/0.0.0.0 -i eth1
```

Así, permitiremos la redirección de paquetes desde la subred interna (con la máscara indicada) a través del interfaz externo del router.

3.3 – Mapeo estático de puertos en Linux: ipmasqadm

Una vez activado en el núcleo del sistema el soporte de enmascaramiento de direcciones, disponemos de comando *ipmasqadm* para indicarle al núcleo del sistema qué mapeos de puertos vamos a activar. Aunque este software es capaz de manejar distintos tipos de mapeos de puertos, sólo trataremos con el módulo estándar "PORTFW".

Su sintaxis para añadir puertos es la siguiente:

`portfw -a -P {protocolo} -L {ip_local} {puerto_local} -R {ip_remota} {puerto_remoto}`

- `protocolo` : protocolo a usar en el puerto mapeado. Puede ser "tcp" o "udp".
- `ip_origen` : dirección IP local a mapear.
- `puerto_local` : puerto local a ser mapeado.
- `ip_remota` : dirección IP a la que mapearemos la local.
- `puerto_remoto` : puerto asociado a la dirección IP remota, que ofrece el servicio que queremos que sea accesible.

Para eliminara puertos ya creados, la sintaxis sería la misma que para crearlos, aunque modificando el parámetro "`-a`" por "`-d`". En este caso sólo sería necesario añadir los últimos parámetros ("`-R {ip_remota} {puerto_remoto}`") de haber posibilidad de confusión entre los mapeos preexistentes.

Capítulo 4 – Proxy

La manera más común de compartir una conexión es enrutando los paquetes a nivel IP, pero trabajando a un nivel más alto podemos conseguir funcionalidades adicionales al tratar los paquetes con conocimiento de su estructura interna. La desventaja de trabajar así es que debemos personalizar el programa servidor para cada protocolo que vayamos a tratar.

Como categoría especial de los servidores proxy, cobran especial relevancia los servidores de caché. Este tipo de servidores mantienen un almacenamiento temporal de páginas web ya visitadas que se refresca automáticamente, de manera que los clientes obtienen las páginas más frecuentemente visitadas a muy alta velocidad. Para ello, el servidor tiene una serie de algoritmos de optimización de la caché, manteniendo un balance entre los recursos del servidor (disco, memoria, procesador...) y el tipo de objetos cacheados[15] (las webs más visitadas, los ficheros que menos ocupan, los que se predice que se van a utilizar próximamente, etc.).

Además de éstos (que se suelen utilizar para servicios de FTP y http), otros servidores de proxy que se suelen usar son los de DNS o de Telnet. Para usos más genéricos existen proxys de Socks y Winsocks, los protocolos estándar de Unix y Windows para puertos TCP/IP, que hacen enrutables a este nivel gran cantidad de aplicaciones.

[15] En este documento se utiliza el verbo castellano “cachear” como traducción literal del verbo inglés “to cache”, dándole el significado de almacenar temporalmente datos para un acceso posterior más rápido.

4.1- Servidores de caché

Dado que el protocolo HTTP (junto, quizá, con el FTP) es tradicionalmente el que más ancho de banda consume en una WAN (en redes locales el tráfico suele ser directo de ordenador a ordenador), será en el que centremos el estudio sobre cacheado de tráfico.

Como indica David Guerrero en su artículo "Mejora la navegación de tus usuarios y ahorra ancho de banda usando servidores proxy para cachear páginas web" *[GUERRERO-1999]*: "Ignorando el hecho del rápido crecimiento de la web, se solicitan los mismos documentos y se visitan las mismas páginas una y otra vez. Es sorprendente saber cuántos usuarios leen las páginas de NBA.COM, o cuántas veces cruzan las líneas los GIFs de Altavista."

Continuando con su artículo, leemos que "incluso si no sabes nada de cacheado de webs, probablemente lo estás usando en tu navegador. Los navegadores más comunes usan esta aproximación con los documentos y objetos que se descargan de la web, manteniendo una copia de los documentos más recientes en memoria o en disco. Cada vez que pulsas el botón de *Atrás* o visitas la misma página, ésta está en memoria y no es necesario volver a descargarla. Éste es el primer nivel de cacheado, y esta técnica es aplicable a toda la web".

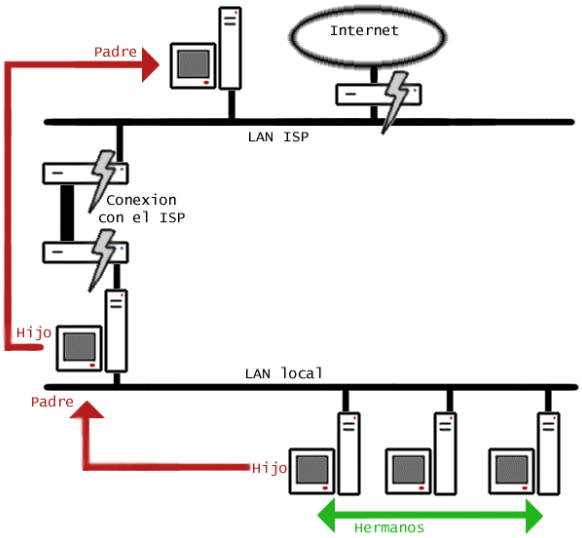
Tal y como veíamos en la introducción, no es necesario que físicamente el proxy ocupe el lugar entre la red interna y la externa, ya que los dentro de el protocolo usado, llamado ICP (Internet Cache Protocol), queda implícito el camino por el que deben circular los paquetes.

Una vez que un cliente hace una petición ICP a un proxy, éste responde con un mensaje de tipo *miss* (fallo) o *hit* (acierto). En el caso de un *hit*, el servidor contenía previamente un dato fiable con el que responder al usuario, con lo cual se evita un acceso externo. Si la respuesta fue *miss*, el servidor no disponía del dato solicitado (o disponía de una versión anticuada), con lo cual el servidor debe pedir ese dato a la red externa antes de facilitárselo al usuario.

Cuando no es un PC cliente el que accede a un servidor proxy, sino otro servidor del mismo tipo, hablamos de una jerarquía de servidores. Así, se crean complejas estructuras de servidores proxy enlazados entre sí para mantener una caché coherente con la cantidad y el tipo de datos que intercambian. Si un servidor tiene necesariamente que remitir sus peticiones a otro para conseguir acceso externo, su relación es de padre-hijo. Por otra parte, si ambos servidores son accesibles igualmente por los clientes pero se dividen entre sí la información cacheada para un mayor rendimiento, cumplirán la función de hermanos..

En el siguiente esquema aparecen reflejados estos casos, poniendo como ejemplo el acceso de una red corporativa a un ISP[16]. En este ejemplo, el proxy padre de la red interna actúa también como enrutador entre esta red y la del ISP, mientras que el proxy que realiza dicha función en el ISP (y que en esta categoría de encuentra como único servidor independiente) se encuentra como servidor interno.

[16] Internet Service Provider: empresa que se encarga de proporcionar acceso a Internet a través de sus líneas.



Una vez determinada la función de un servidor proxy concreto, hay varios parámetros de rendimiento que deben afinarse para cada instalación. Principalmente hay que especificar el tipo de datos a cachear, ya que no tiene sentido guardar en el servidor objetos que varíen frecuentemente, datos considerados como seguros (éstos deben ser destruidos del servidor tan pronto como el cliente acceda a ellos), ni datos excesivamente grandes que impidan almacenar objetos más pequeños que quizá sean más comúnmente solicitados.

Continuando con la parametrización de los servidores, puede definirse el tiempo que los objetos se consideran o no válidos. Así, se definen tres estados llamados *fresh*, *normal* y *stale* (freso, normal y pasado respectivamente). En el caso de un objeto fresco, será ofrecido directamente a los clientes sin ningún tipo comprobación. Para los objetos normales se comprobará si la fecha de la última modificación de la réplica del servidor es mayor o menor de la del objeto real, refrescando o no el objeto según el caso. En el caso de los que se encuentren obsoletos, siempre se refrescará el dato antes de ser ofrecido al usuario.

4.2 – Proxy en Linux: squid

Squid es el software de proxy más ampliamente utilizado en la comunidad GNU. Aunque en nuestro caso vamos a cachear sólo peticiones HTTP y FTP (no tiene mucho sentido utilizar proxys para servicios que no se van a beneficiar de la caché, ya que estos acceden directamente por NAT), sus capacidades van más allá con soporte de SSL[17] o de WCCP[18].

Como es usual, el programa se instala de tal manera que su ejecución se controle como un servicio estándar del sistema, arrancándose desde `/etc/inet.d/squid`, y leyendo su configuración desde `/etc/squid.conf`.

Dentro de dicho fichero de configuración deberemos configurar, al menos, las siguientes opciones básicas:

- `http_port {puerto}:` define el número de puerto TCP asociado al servicio.
- `cache_dir {directorio} {tamaño en megabytes} {número de subdirectorios} {número de subdirectorios de segundo nivel}`: define que la estructura de almacenamiento de páginas web se albergará en el directorio especificado, y se dividirá en dos niveles de subdirectorios con un número concreto de ficheros contenidos encada uno. También se controla el tamaño máximo que ocupará esta estructura. Esta estructura interna tan compleja pretende optimizar al máximo el acceso a la caché.
- `cache_mgr {email}`: dirección a la que el software envía sus registros de errores internos y que aparece como contacto en los errores enviados a los usuarios.

Para definir una jerarquía de cachés usaremos la siguiente opción:

- `cache_peer {dirección} [parent|sibling] {puerto_http} {puerto_icp} {opciones}`: define una relación de padre o de hermano con un proxy determinado, con el que se comunica por los puertos de HTTP e ICP definidos.

Entre las opciones más comunes para este parámetro están:

- `default`: redirecciona por defecto todas las peticiones a el servidor definido, anulando la posibilidad de acceder directamente a la página si éste está inactivo.
- `weight={peso}`: indica la prioridad del servidor. Se accederá más frecuentemente a los servidores con mayor peso.
- `no-query`: nunca intenta hacer peticiones ICP al servidor, ignorando el puerto definido para este protocolo.

Una vez definidas, al menos, estas opciones, hay que definir una política de permisos de acceso a los clientes de proxy en base a listas de ACLs (listas de control de acceso). Para una red pequeña será

[17] Secure Socket Layer: protocolo para transmisión de información cifrada.

[18] Protocolo de Cisco que las peticiones de páginas web sean procesadas automáticamente por el proxy sin necesidad de reconfigurar los navegadores de los clientes.

suficiente con asegurarnos de que las peticiones circularán desde la red interna hacia el exterior, y no en sentido contrario.

Para ello, primero definimos clases para luego asociarlas a reglas concretas para HTTP e ICP (según si el cliente es navegador u otro servidor proxy). La sintaxis es la siguiente:

- `acl {clase} src {subred}`: define una clase con el nombre indicado para la subred indicada.
- `[http|icp]_access [allow|deny] {clase}`: permite o deniega el acceso a través del proxy a las máquinas de la clase indicada, a través de HTTP o ICP.
- `[never|always]_direct {clase}`: desactiva o activa el acceso directo a determinadas máquinas. Si el acceso directo está permitido, aunque haya definida una caché padre, el proxy buscará los datos directamente.

Por último, para determinar la persistencia de los objetos en la caché, se aplica un complejo algoritmo que se apoya, entre otros, en los siguientes parámetros:

- `refresh_pattern {patrón} {minutos mínimos} {porcentaje} {minutos máximos}`: define, para los ficheros que cumplan un determinado patrón, los minutos de permanencia en la caché y el porcentaje de éstos que permanecen como *frescos*.
- `reference_age {tiempo} {unidad}`: tiempo que se mantienen en la caché los objetos después de su último acceso, con su correspondiente unidad (horas, días, meses, etc.)

En cualquier caso, cambiar estos parámetros requiere un conocimiento interno del funcionamiento del programa que se escapa del ámbito de este estudio.

En el caso del programa de configuración, sólo se modifican los siguientes con respecto a los definidos en el fichero de configuración por defecto:

```
http_port 8080
cache_peer {dirección_padre} parent {puerto} {puerto}
cache_dir /cache 1024 16 256
```

Así, cambiamos el puerto por el que se accede este servicio al 8080, más estándar que el 3128 que se define por defecto, definimos un proxy maestro con la dirección especificada, y cambiamos el valor del tamaño de la caché a 1Gb.

Capítulo 5 – DNS

En los inicios de las redes de tipo TCP/IP, y dado que las redes eran pequeñas y no se encontraban conectadas entre sí, conocer las direcciones de origen y destino de cada máquina era suficiente para intercambiar datos entre ellas. Con el tiempo, y principalmente cuando se comenzaron a aplicar sistemas de enrutado entre ellas, era complicado recordar la función de cada máquina según su dirección IP.

Esto se solucionó en parte haciendo una pequeña base de datos, almacenada en el caso de los sistemas tipo Unix en un fichero llamado "/etc/hosts", que asociaba nombres de máquinas a direcciones IP. Es más, es usual que una máquina apareciese varias veces con varios nombres, uno por cada servicio ofrecido a la red, para simplificar el proceso de transpaso de servicios entre máquinas.

La forma en que máquinas de varias redes compartieran esta base de datos en principio era tan simple como que el administrador del sistema cogía el fichero de un repositorio común y actualizaba con él sus datos. Para crear estructuras de red más complejas fue necesario desarrollar un protocolo que actualizase automáticamente los datos entre todas las máquinas implicadas en la comunicación, el DNS.

En este protocolo las máquinas se asocian en dominios y, dentro de ellos, en subdominios. Partiendo de dominios globales como 'es.' o 'com', se crean dominios más específicos como 'empresa.es' y, dentro de allí, subdominios como 'delegacion.empresa.es' o más concretos como 'departamento.delegacion.empresa.es'. Una nombre de máquina se dice que está perfectamente cualificado si incluye su dominio completo, como 'correo.delegacion.empresa.es'.

Para ello, cada servidor DNS sirve una serie de dominios, pudiendo delegar subdominos dentro de estos en otras máquinas de su organización. Por ejemplo, puede que la delegación de Sevilla de una empresa sirva el dominio 'empresa.es', así como el subdominio 'sevilla.empresa.es', pero delegue la gestión del dominio 'cadiz.empresa.es' en un servidor que se encuentre en otra ciudad.

Para el caso que estamos estudiando, el servidor DNS implementado será mucho más básico, ya que sólo actuará como caché de peticiones DNS. El servidor almacenará las peticiones de nombres de máquinas más frecuentes, pidiendo sólo al exterior las que no tenga almacenadas o las que se encuentren obsoletas, con una filosofía similar a la de un sistema proxy.

5.1 - Servidores de nombres

El DNS es un sistema jerárquico, que define todas las direcciones a través de una raíz (*root*) que se escribe como '**.**', bajo la cual encontramos dominios de nivel superior como ORG, NET, EDU, COM o los identificadores nacionales (ES, UK, etc.).

Para hacer una búsqueda de una máquina concreta como "ganso.fie.us.es", el cliente primero hará una petición su servidor DNS por defecto. Éste buscará en su tabla de caché de direcciones, y, de no encontrar allí este servidor, preguntará a uno de los servidores raíz[19] por el servidor maestro del dominio ".es". Una vez localizado éste, le preguntará por el que a su vez sirva el dominio ".us.es", y a éste quién es el maestro del dominio ".fie.us.es". Este último sí debe conocer la dirección IP concreta de la máquina buscada.

Durante esta búsqueda, todos los servidores intermedios habrán almacenado en su caché de direcciones los resultados obtenidos de manera que en sucesivas búsquedas no será necesario profundizar más en el árbol de servidores.

En sentido contrario, para localizar el nombre de un servidor a través de su IP, estaríamos hablando de resolución inversa de direcciones. Para ello existe un dominio especial llamado "in-addr.arpa", de manera que cada máquina tiene un nombre DNS dentro de éste del estilo de "159.143.214.150.in-addr.arpa", si la dirección de la máquina fuese la 150.214.143.159.

El porqué de la inversión de los números que conforman la cadena de la dirección IP viene por la forma de búsqueda de las direcciones. Nuestro DNS primario haría una búsqueda sucesiva por los maestros de los dominios "in-addr.arpa", "150.in-addr.arpa", "214.150.in-addr.arpa" y "143.214.150.in-addr.arpa", y éste le dará la conversión del nombre "159.143.214.150.in-addr.arpa" a "ganso.fie.us.es".

Para este tipo de búsquedas, existen en los servidores registros de tipo PTR, que no son otra cosa sino asociaciones entre números de máquina y nombres DNS. Por ejemplo, existiría un registro PTR llamado "159" en el servidor maestro del dominio "143.214.150.in-addr.arpa" que lo asociaría con "ganso.fie.us.es".

Otro tipo de registros muy necesarios en el funcionamiento de la red son los de tipo MX, que indican en cada dominió qué máquina se encarga de tratar el correo. Si un cliente de correo quisiera enviar un mensaje a "root@ganso.fie.us.es", preguntaría a su servidor DNS qué máquinas sirven el correo para el dominio "fie.us.es", y obtendría como respuesta "antena.fie.us.es". Después, abriría una conexión directa al puerto 25 de este servidor y entregaría el correo según los estándares de protocolo SMTP.

[19] Estos servidores conocen qué máquinas sirven las direcciones de cada dominio de nivel superior.

5.2 – DNS en Linux: bind

Vamos a estudiar la configuración de un servidor de "sólo cacheo", ya que un estudio completo de la configuración del servidor *bind [BIND]* se escapa ampliamente del ámbito de este proyecto.

La configuración básica se encuentra en un fichero llamado /etc/named.boot. Dado que no vamos a definir ningún dominio ni subdominio, la configuración puede ser la siguiente:

```
directory                       /var/named
cache     .                     named.ca
primary   0.0.127.in-addr.arpa  named.local
```

Aquí estamos definiendo el directorio donde se almacenan los demás ficheros de configuración, así como dós dominios: uno para el interfaz de *loopback* y otro para la caché de direcciones para el resto de interfaces.

El fichero "named.ca" ya existe en la distribución y contiene una lista de los servidores de nombre raíz del mundo, que Internic se encarga de hacer público. El fichero "named.local", también se encuentra configurado por defecto, define las máquinas que encontramos en la subred 127.0.0 (asignada al interfaz de bucle), y que sólo incluye la máquina "localhost" con la dirección 127.0.0.1.

Volviendo al directorio /etc, encontramos el fichero "/etc/host.conf", donde definimos el órden de búsqueda en la resolución de máquinas. En nuestro caso, sólo es necesaria que contenga "order hosts,bind", lo cual especifica que para resolver un nombre de máquina primero debemos buscar en "/etc/hosts" y después intentar resolverlo por DNS.

Por último, en "/etc/resolv.conf" se definen los servidores de nombres que usaremos por defecto para direcciones no conocidas (en nuestro caso, para direcciones distintas a 127.0.0.1 y no definidas en /etc/hosts). Dado que presuponemos que por DHCP se nos dan todos los datos de la interfaz externa, aquí debemos encontrar algo similar a esto:

```
domain fie.us.es
search fie.us.es
nameserver 150.214.186.69
nameserver 150.214.143.154
nameserver 150.214.1.3
```

Sin entrar en detalle, vemos que en este ejemplo hay definidos tres servidores de nombres, de manera que para cada dirección consultada, se accede en secuencia a los tras hasta que alguno pueda dar una respuesta con seguridad. Además, está definido que el dominio de la máquina es "fie.us.es", y que será este mismo el que se presuponga en todas las consultas a nombres de máquinas que no especifiquen un dominio (así, la dirección "ganso" se tomará como "ganso.fie.us.es" automáticamente).

Todos los ficherosque hemos ido viendo ya están preconfigurados en la instalación estándar de *bind* en Turbolinux, al igual que ocurre en la mayoría de instalaciones de Linux. Así, no es necesario hacer desde el programa de configuración ningún cambio asociado al funcionamiento de este servicio.

Capítulo 6 – Futuras ampliaciones

En la introducción ya describíamos el ámbito de este proyecto, limitándonos a dar una alternativa basada en software libre a caros productos comeciales cerrados, centrándonos en aspecto didáctico del desarrollo. Al reducir tanto el campo del estudio, hay muchos aspectos que podían haberse tratado pero que se han dejado de lado por no considerarse imprescindibles para los objectivos marcados.

En concreto, sería interesante ampliar el uso de aspectos no utilizados del software descrito, como la posiblidad de configurar *ipchains* como cortafuegos o *bind* como servidor completo de DNS. Además, también cabría incluir funcionalidades nuevas al software de configuración, así como nuevo software de gestión integrable con otras plataformas o servicios de red comunes como el de web o de ficheros para redes Windows. En los próximos apartados describimos someramente los que se han considerando más interesantes de estos aspectos, dando algunas sugerencias sobre su implementación.

6.1 – Seguridad

Para un uso del softare descrito en un entorno de producción, sería más que recomendable ampliar la seguridad en la configuración, ya que en el estado actual cualquier persona no autorizada podría modificar la configuración de la máquina.

Una aproximación sería modificar la configuración del servidor de telnet para alejarlo del puerto estándar, aunque sería trivial para un atacante escanear todos los puertos de la máquina y analizarlos uno a uno. Quizá, en este sentido, el acceso por puerto serie sería menos explotable al requerir de proximidad física a la máquina. Basar, en cualquier caso, la seguridad del sistema en la ocultación de información debe ser siempre la última opción, así que es recomendable buscar medidas alternativas.

Lo principal es incluir algún tipo de clave o autentificación en la confiugración del sistema. Para ello, sería aconsejable utilizar sistemas de encriptación en la transmisión de los datos, así como en el almacenamiento de las claves en disco duro, similar a la que se utiliza en todos los sistemas tipo Unix.

Por una parte, la clave de administración podría guardarse en disco utilizando el algoritmo DES[20], utilizando las funciones que existen para este fin en la librería *crypt.h*. Dado que este algoritmo no permite la desencriptación del código una vez codificado, la manera de actuar sería codificar la clave introducida por el usuario para compararla con la archivada.

Por el otro lado, y para que los datos introducidos por el usuario no viajen por la red sin codificar, es recomendable encapsular sobre OpenSSL[21] *[OPENSSL]* la transmisión de datos entre el servidor y el usuario. Para una hipotética configuración a través del protocolo HTML, tal y como se describirá después, puede usarse el propio protocolo de HTTP seguro que incluye como una opción básica el software Apache.

Otra manera de potenciar la seguridad del software sería incluyendo un sistema de autentificación de usuarios completo, tanto para la configuración como para determinados servicios. Así, se podrían definir perfiles de administración independientes para distintos operarios, de manera que cada uno pudiese acceder sólo a funciones básicas (por ejemplo, consulta del estado o reinicios programados) hasta las funciones de configuración más avanzadas. También, para cada servicio ofrecido, se podría analizar la posibilidad de incluir una autentificación para los usuarios, limitando así el uso del proxy a determinados usuarios de la red. El software Squid incluye funcionalidades de este tipo tanto a nivel de IP de cliente como a través de una autentificación al inicio de cada sesión.

[20] Data Encryption Standard, algoritmo de encriptación de claves utilizado para el almacenamiento de claves en la gran mayoría de sistemas tipo Unix

[21] Implementación bajo licencia GPL del protocolo SSL (Secure Socket Layer) de transmisión segura de datos.

6.2 – Servidor SNMP: net-snmp

Una funcionalidad originaria de los elementos activos de la red y que actualmente se ha extendido a todo tipo de máquinas es el servico SNMP[22]. En principio este servicio permitía extraer de estos elementos datos de la configuración o estadísticas de uso, pero con el tiempo se ha convertido en un potente sistema de gestión remota de redes.

La base del protocolo eran unas bases de datos, denominadas MIB, que guardan toda la información que el servidor SNMP de cada dispositivo ofrecerá a los clientes, así como cuáles de estos datos son modificables remotamente. Un mismo servidor puede tratar con información de programas muy diversos tratando con bases de datos distintas.

Un puto débil que siempre ha acarredo este protocolo ha sido la autentificación, ya que se basaba en una clave única que se transmitía por la red sin codificar. En la última versión se ha añadido la posibilidad de crear varios usuarios, cada uno con ua contraseña asociada, con accesos independientes a cada dato de las MIB.

Como implementaciones a considerar en el mundo Linux, podemos destacar a *net-snmp [NETSNMP]* (que a su vez es la versión actualizada del extendido Ucd Snmp, anteriormente llamado Cmu Snmp). Sobre este software, actualmente de código libre, se han desarrollado multitud de librerías MIB adicionales que permiten controlar casi cualquier software GPL que pueda correr en la máquina.

Para nuestro servidor, en concreto, sería interesante desarrollar una base de datos MIB independiente que controlase única y excusivamente los parámetros que se considerasen en su momento interesantes. De hecho, eso podría desembocar en un proyecto de una consola centralizada de gestión para múltiples servidores basada únicamente en este protocolo.

[22] Simple Network Management Protocol

6.3 – Servidor / cliente de redes Microsoft: samba

Dado que la mayor parte del parque de estaciones de trabajo instaladas actualmente corresponde a sistemas Windows (desde la versión 95 a 2000 Proffessional), es interesante como servicio adicional integrar nuestro servidor en estas redes como servidor de ficheros.

Los sistemas tipo Windows utilizan desde la versión 3.11 su propio protocolo de comunicación a alto nivel, llamado NetBios, que cubre los algoritmos de autentificación y cualquier tipo de comunicaciones entre aplicaciones para Windows, desde información de control a transferencias de ficheros. En principio, este protocolo no se apoyaba sobre TCP/IP, sino sobre el protocolo NetBeui, propietario de Microsoft y de prestaciones muy limitadas. En concreto, su poca optimización y su dependencia de los paquetes broadcast, hace que sobrecarge mucho la red y evita la posibilidad de ser enrutado entre subredes. Por todo esto, actualmente en estas redes se ha pasado a encapsular NetBios sobre TCP/IP, opción por defecto en los sistemas operativos Microsoft más recientes.

Para redes NetBios funcionando sobre TCP/IP, y en lugar de usar el protocolo DNS, se utiliza un desarrollo similar de microsoft llamado WINS (Windows Internal Name Service), que relaciona nombres NetBios con direcciones IP. Como ventaja añadida a los sistemas DNS tradicionales, las máquinas se encargan de indicar al servidor sus incios y paradas, de manera que la base de datos siempre se encuentra actualizada sin mantenimiento por parte del administrador. Con el paso a Windows 2000 se ha adoptado la solución de utilizar sistemas DNS dinámicos estándar, aunque el protocolo WINS se ha mantenido como compatibilidad para migración de redes basadas en Windows NT.

La autentificación y, en general, la estructura usada para el almacenamiento de cuentas de usuarios y máquinas, es en cierto sentido más compleja bajo el protocolo NetBeui que en una red típica de sistemas Unix. Sin entrar en detalles, en las redes de servidores Windows NT y 2000 los recursos se estructuran en dominios NetBios, de manera que cada usuario o cada máquina pertenece unívocamente a uno de ellos. Es posible, en cambio, definir relaciones de confianza entre dominios distintos para poder acceder a recursos remotos sin tener que repoetir una autentificación de usuario. Los permisos se pueden definir en base a ACLs para la gran mayoría de recursos del sistema, incluyendo ficheros locales y compartidos.

Para nuestro servidor sería especialmente interesante incluir la posibilidad de servir ficheros a otros sistemas Windows para facilitar su integración en sistemas de copia de seguridad corporativos. También es posible con el software *samba [SAMBA]*, considerado unos de los mayores logros del software libre y no exento de polémica, actuar como servidor WINS o incluso como servidor de autentificación para los clientes Windows de la red.

6.4 – Servidor FTP: wuftp

Como una alternativa más estándar para la transmisión de ficheros, aunque quizá menos orientadas a redes pequeñas, podríamos ofrecer datos a través el protocolo FTP. Esto nos permitiría ofrecer datos, no sólo a clientes Windows sin o a cualquier máquina con una pila TCP/IP y clientes genéricos, utilizando además este servicio para otras funciones como la copia de seguridad de la configuración o la actualización de páginas web para el servicio de HTTP, como se describe en el siguiente punto.

Para ello deberíamos dar soporte a la creación y modificación de usuarios para poder dar estos servicios con un cierto nivel de seguridad. Además, así se podrían crear directorios de inicio personalizados para cada usuario para estructurar la información según los distintos tipos de usuarios. Esta gestión de usuarios podría utilizarse también para cualquier otro protocolo, como podría serlo SMB o WWW. Como un paso intermedio, podrían definirse varios usuarios genéricos, según los perfiles más comunes entre los clientes.

Pese a que hemos tomado *WU-ftpd [WUFTPD]* como servidor más representativo, la oferta de servidores libres se ha diversificado tanto que habría que mencionar otros más recientes como *proftpd [PROFTPD]*.

6.5 – Servidor WWW: apache

Uno de las funciones más ampliamente implementadas en sistemas libres es la de servidor web, gracias al rendimiento y a la estabilidad alcanzada por el software *apache [APACHE]*. Con el tiempo, este servicio ha pasado de servir como un mero escaparate para las llamadas *e-business*[23] a convertirse en la principal manera de ofrecer información interna y externa en cualquier sistema empresarial o educativo. Gracias a las facilidades de acceso a bases de datos implementadas en los servidores HTTP, las web internas han eliminado la necesidad de uso de programas cliente propietarios para cada aplicación.

Como funcionalidad añadida al servidor en estudio, cabría estudiar la posibilidad de añadir esta funcionalidad, pero para ello habría que incluir algún tipo de servicio de intercambio de ficheros seguro para actualizar el conjunto de páginas web a servir. Esto podría realizarse, bien a través de un servicio de FTP, bien integrando el servidor en una red Microsoft, como se describe en los puntos anteriores.

Un uso muy importante que se podría dar a un servidor HTTP interno sería el de ofrecer una alternativa al programa de configuración accesible por cualquier navegador. Esto permitiría que el administrador pueda configurar la máquina de una manera rápida y cómoda sin tener que depender de tener o no un cliente de telnet (cosa que, por ejemplo, no es común en ordenadores con sistema operativo MacOS).

Para poder realizar una configuración de este estilo, necesitamos añadir una capa de seguridad al protocolo HTTP. En estos casos encapsularemos sobre OpenSSL estos protocolos de tal manera que se pueda asegurar la confidencialidad en el transvase de informaición.

Este protocolo funciona a través de un sistemas de clave que precisa de una tercera entidad, además del servidor y el cliente, que gestione el sistema de claves de manera segura y confidencial. Así, nuestro servidor dispondrá de un identificador cifrado único que utilizará en todos los intercambios de información. De no usarse este organismo certificador, aún estando la información cifrada, otro organismo podría suplantar la identidad de nuestro servidor de cara a los usuarios que intentasen configurarlo.

[23] Término acuñado para las empresas que apoyan gran parte de su infraestructura en sistemas de Internet / Intranet

6.6 – Cliente PPP: pppd

Tal y como se ha definido este proyecto, para conexiones a través de la línea telefónica necesitaríamos un router externo que proporcionara una conexión de red Etherneth. En la pequeña empresa esto puede no ser demasiado común, así que sería interesante agregar al servidor una tarjeta modem o de RDSI para el acceso por línea telefónica a Internet.

Uno de los problemas inherentes a una conexión telefónica es que no podemos considerarla permanente. Por una parte suelen sufrir problemas de estabilidad, y por otra, no siempre hay disponible una tarifa plana con lo que hay que proporcional algún sistema de gasto telefónico. Debemos ser capaces de iniciar la llamada bajo demanda (por ejemplo, cuando un usuario abre su navegador), de aumentar el número de líneas activas cuando aumente el ancho de banda y de cortar la comunicación cuando cese el flujo de datos.

Para ello, el demonio estándar de acceso a la red por llamadas (*pppd*) tiene una serie de contadores que gestionan las líneas según el tráfico de la red. Activar las líneas cuando circule tráfico puede acarrear el problema de que procesos automáticos en la red que lancen paquetes hacia fuera de la organización (por ejemplo, servidores de nombres actualizándose) activen las líneas durante largos periodos. Para que esto no sea un problema habría que añadir la posibilidad de parametrizar completamente los niveles de carga de red en que se cambia el estado de las líneas y los tiempos de espera antes de realizarlos.

6.7 – Portabilidad a otras distribuciones y plataformas

Dado la rapidez con que Linux se ha ido extendiendo entre diferentes plataformas, es interesante considerar la posibilidad de utilizar distintas distribuciones para este proyecto. Quizá la arquitectura Intel sea la única suficientemente barata como para que la diferencia de precio con los routers comerciales sea significativa, pero en algunos casos podemos necesitar más potencia de cálculo para hacer frente a un tráfico de paquetes suficientemente alto. En esos casos, elegir una distribución compatible con otras plataformas (TurboLinux sólo tiene soporte para arquitectura Intel) podría ser interesante.

Por otra parte, también puede ser útil el migrar a una plataforma industrial como PC104 para diseñar routers de tamaño y consumo similar a los que se encuentran en el mercado. En el caso concreto de la arquitectura PC104, al ser de tipo PC, podemos seguir utilizando una distribución para plataformas Intel, pero con el inconveniente añadido de que tendremos un tamaño de disco más limitado. Para este caso habría que hacer una selección muy cuidadosa de paquetes a instalar, o si esto no es suficiente, migrar a una distribución menos exigente en tamaño de disco como puede ser Slackware.

El hecho de migrar a otra distribución puede ser bastante complicado en algunos casos, ya que es usual que cada una defina unos ficheros de configuración distintos a los estudiados aquí. Por ejemplo, en la distribución Debian 2.1, los parámetros de red se configuran directamente añadiendo líneas al script de configuración "`/etc/init.d/network`".

También es usual que, como es el caso de la distribución arriba indicada, cambie los lugares del sistema de ficheros donde se almacenan determinados scripts, o símplemente cambien las versiones de los demonios utilizados o incluso se preinstalen otros. Esto último puede ser corregido simplemente intstalando dichas versiones, pero esto puede ser problemático al generarse problemas en las interdependencias con otros paquetes instalados.

Un problema más serio que nos podemos encontrar el el hecho de que las nuevas distribuciones vengan con el núcleo del sistema 2.4.0 o posterior, lo cual nos obligaría a migrar toda la configuración al nuevo sistema de cortafuegos utilizado, usando el comando *iptables* en lugar de *ipchains*.

Capítulo 7 – Programa de configuración

A continuación se describe el programa de configuración que se ha diseñado para configurar automáticamente el servidor, de manera similar a como podría configurarse un router estándar. Se ha elegido como lenguaje de desarrollo el C, por ser el más usado en el desarrollo de aplicaciones sobre sistemas Unix, así como en el propio desarrollo del núcleo del sistema.

El código fuente presentado en este capítulo genera un ejecutable que se dejará accesible por telnet o por puerto serie. Este programa utiliza un fichero de configuración "/etc/ganso.conf" donde se definen todos los parámetros del router, guardando una copia de seguridad en el mismo directorio añadiendo la hora y la fecha en que deja de ser efectivo.

7.1 – Estructura general

El código fuente consta de los siguientes ficheros:

- *Makefile*

Fichero genérico para la compilación. Permite compilar el código cómodamente con los siguientes comandos:

- ✓ `make` : compila el código del programa generando un ejecutable llamado `configurar`.
- ✓ `make clean` : borra todos los ficheros temporales usados durante la compilación.
- ✓ `make mrproper` : borra los ficheros temporales y el ejecutable.

- *ganso.h*

Contiene enlaces a las librerías utilizadas, así como las variables globales que almacenan la configuración actual del sistema.

- *comprobacion.h / comprobacion.c*

Módulo para la comprobación de tipado de cadenas. Incluye las siguientes funciones

- ✓ `numerico(cadena)` : devuelve 0 si la cadena es estrictamente numérica, u otro valor en caso contrario.
- ✓ `haypuntos(cadena)` : devuelve 0 si la cadena contiene puntos, u otro valor en caso contrario.
- ✓ `numericopuntos(cadena)` : devuelve 0 si la cadena sólo contiene números o puntos, u otro valor en caso contrario.

- *leer_config.h / leer_config.c*

Módulo para leer la configuración del fichero `/etc/ganso.conf` y almacenarla en las variables globales definidas en `ganso.h`. Lo componen las siguientes funciones:

- ✓ `leer_fichero(cadena,fichero)` : lee una línea de de texto del fichero indicado.
- ✓ `leer_config ()` : lee la configuración completa.

- *guardar.h / guardar.c*

Módulo para guardar la configuración de las variables globales en `ganso.h`, haciendo antes una copia de seguridad de este fichero en el directorio `/etc` añadiendo la fecha y la hora actual a la cadena `ganso.conf`. Esta función también crea todos los ficheros de configuración según los parámetros dados para que, tras un reinicio del sistema, esté preparado para trabajar.

- *configurar.c*

Programa principal de configuración. Contiene las siguientes funciones:

- ✓ `tecla()`: espera a que se pulse la tecla 'Intro'.
- ✓ `leer(cadena1,cadena2)`: muestra el mensaje 'cadena1' en pantalla, y espera a que introduzca una línea de texto, que se guarda en 'cadena2'.
- ✓ `mirar()` : muestra la configuración actual.
- ✓ `cambiar()` : pide nuevos parámetros de configuración.
- ✓ `reiniciar()` : reinicia la máquina.
- ✓ `parar()` : para la máquina.
- ✓ `agregar_mapeos()` : añade nuevos mapeos de puertos.
- ✓ `quitar_mapeos()` : elimina mapeos existentes.
- ✓ `menu()` : presenta el menú principal, y llama a las funciones anteriores si es necesario.
- ✓ `main()` : función principal, que lee la configuración y presenta el menú principal.

Además, se han modificado los siguientes ficheros del sistema:

- *`/usr/local/bin/parametrosdhcp`*

Esta función lee el fichero donde se almacena las configuraciones IP válidas recibidas del servidor DHCP, seleccionando sólo la última válida. Este ejecutable se llama desde la función `mirar()`.

- */etc/rc.d/rc.local*

Desde este fichero, que se ejecuta tras todos los scripts de arranque, llamamos al script `/etc/rc.d/mapeos`, que crea todos los mapeos de puertos que se hayan configurado en el sistema.

7.2 – Acceso desde telnet y por puerto serie

- *Acceso desde telnet*

Para este punto habrá que modificar e fichero `/etc/inetd.conf`, en el cual se definen los programas que responden a los accesos a los puertos controlados por el demonio `inetd`. En concreto, basta con modificar la línea que hace referencia al protocolo telnet para que quede de esta manera:

```
telnet stream tcp nowait root /usr/sbin/tcpd
/usr/sbin/in.telnetd -L /usr/local/bin/configurar
```

La sintaxis usada en esta línea es la siguiente:

- `telnet` : nombre del servicio, asociado a un número a través del fichero `/etc/services`.
- `stream` : especifica que vamos a usar un puerto para transferir un flujo de datos.
- `tcp` : marca el protocolo a usar por este puerto como TCP.
- `nowait` : indica que el demonio de internet no debe esperar a que finalice la ejecución de un hilo para lanzar el siguiente, de manera que se pueda atender a varias peticiones simultáneamente.
- `root` : señala el usuario que arrancará los servicios.
- `{demonio}` : selecciona el demoino estándar de telnet, pero especificando que usará como programa de inicio nuestro software de configuración. Así, cuando hagamos un *telnet* a la máquina desde cualquier lugar, obtendremos el programa de confiuguración en lugar de la petición de *login* que recibimos normalmente.

- *Acceso desde puerto serie*

En este caso, el fichero donde tenemos una definición del comportamiento de los puertos serie es el mismo donde se define la configuración de los distintos los niveles de arranque, es decir, `/etc/inittab`. Dado que la máquina tiene dos puertos serie, accesibles desde /dev/ttyS0 y /dev/ttyS1 respectivamente, tendremos que añadir las líneas que se indican a continuación a este fichero. Nótese que estamos utilizando el programa *agetty [agetty]* en lugar *getty*[24] o *mgetty*[25], debido a que el primero nos permite parametrizarlo según veremos a continuación. Dado que este programa no aparece en la distribución TurboLinux por defecto, deberemos bajarlo de la página web referenciada y compilarlo en el fichero ejecutable `/sbin/agetty`.

```
T0:2345:respawn:/sbin/agetty -Linl /usr/local/bin/configurar ttyS0 9600
vt100
```

[24] Programa que invoca por defecto el proceso inicial del sistema para permitir acceso a los usuarios. Este software es el que pide el nombre y la clave del usuario, ejecutando a continuación el proceso *login* que ejecuta a su vez la interfaz de comandos apropiada.

[25] Versión modificada de *getty* para tratar con modems y conexiones serie en lugar de los terminales virtuales asociados al sistema.

```
T1:2345:respawn:/sbin/agetty -Linl /usr/local/bin/configurar ttyS1 9600 vt100
```

De esta manera indicamos al sistema que en todos los sistemas de arranque multiusuario (2, 3, 4 y 5), arranquemos un processo agetty en el dispositivo adecuado (ttyS0 y ttyS1, correspondientes a los dos puertos serie por defecto del sistema), a 9600 baudios, con emulación de terminal vt100, y con las siguientes opciones:

- `L` : especifica que no tenemos que esperar a detectar una línea portadora, dado que no vamos a usar un modem, sin una conexión directa por cable.
- `i` : no muestra el contenido del fichero `/etc/issue`[26].
- `n` : no pide un nombre de usuario, ejecutando el fichero de arranque con el mismo usuario que inicia el proceso *agetty* (en este caso, el superusuario).
- `l {fichero}` : ejecuta el fichero especificado al arrancar, en nuestro caso, el fichero de configuración.

[26] Fichero que contiene el texto que se muestra a los usuarios antes de que se autentifiquen, informándoles del nombre y, a veces, de la funcion de la máquina.

7.3 – Código fuente

- */usr/local/bin/parametrosdhcp*

```
#!/bin/ksh
fichero=/var/lib/dhcp/dhclient.leases
typeset -i total=$(echo $(wc -l $fichero)|cut -f1 -d" ")
typeset -i primera=$(grep -n eth1 $fichero |tail -n1 |cut -d: -f1)
typeset -i numero=$total-$primera-1
typeset -i ultimo=$primera+$numero-1
cat $fichero|head -n $ultimo|tail -n $numero
```

- */etc/rc.d/rc.local*

Se añade la siguiente línea:

```
/etc/rc.d/mapeos
```

- *Makefile*

```
CC = cc
FLAGS = -Wall -O2

all: ejecutables

configurar.o: configurar.c
    $(CC) $(FLAGS) -c -o configurar.o configurar.c

comprobacion.o: comprobacion.c
    $(CC) $(FLAGS) -c -o comprobacion.o comprobacion.c

guardar.o: guardar.c
    $(CC) $(FLAGS) -c -o guardar.o guardar.c

leer_config.o: leer_config.c
    $(CC) $(FLAGS) -c -o leer_config.o leer_config.c

ejecutables: configurar.o comprobacion.o guardar.o leer_config.o
    $(CC) $(FLAGS) comprobacion.o guardar.o leer_config.o configurar.o -o
configurar
    strip configurar

clean:
    rm -f *.o
    rm -f core
    rm -f *~

mrproper: clean
    rm -f configurar
```

- *ganso.h*

```
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <unistd.h>
#include <time.h>
#include <pwd.h>

#define MAXCAD 8096
#define MAXPORT 256

struct Tmapeo {
 char tipo[MAXCAD];
 char ip1[MAXCAD];
 char puerto1[MAXCAD];
 char ip2[MAXCAD];
 char puerto2[MAXCAD];
 };

char Ghost[MAXCAD],Gip[MAXCAD],Gdominio[MAXCAD],Gmascara[MAXCAD];
char
Gsubred[MAXCAD],Gbroadcast[MAXCAD],Gip_proxy[MAXCAD],Gpuerto_proxy[MAXCAD];
struct Tmapeo Gmapeo[MAXPORT];
int Gnmapeos;
```

- *comprobacion.h*

```
int haypuntos(char *);
int numerico(char *);
int numericopuntos(char *);
```

- *comprobacion.c*

```
int haypuntos(char *cadena)
{
int f=0,hay=0;

while (cadena[f]!='\0') hay+=(cadena[f++]=='.')?1:0;
return(hay);
}


int numerico(char *cadena)
{
int f=0,num=1;

while (num && cadena[f]!='\0') num=(cadena[f]>='0' && cadena[f++]<='9');
return(num);
}
```

```
int numericopuntos(char *cadena)
{
int f=-1,num=1;

while (num && cadena[++f]!='\0')
 num=(cadena[f]=='.' || (cadena[f]>='0' && cadena[f]<='9'));

return(num);
}
```

- *leer_config.h*

```
void leer_fichero(char *,FILE *);
void leer_config(void);
```

- *leer_config.c*

```
#include "ganso.h"

void leer_fichero(char *cadena,FILE *fichero)
{
fgets(cadena,MAXCAD,fichero);
cadena[strlen(cadena)-1]='\0';
}


void leer_config(void)
{
char cadena[MAXCAD];
FILE *fichero;
int fin=0;

fichero=fopen("/etc/ganso.conf","r");
leer_fichero(Ghost,fichero);
leer_fichero(Gdominio,fichero);
leer_fichero(Gip,fichero);
leer_fichero(Gbroadcast,fichero);
leer_fichero(Gsubred,fichero);
leer_fichero(Gmascara,fichero);
leer_fichero(Gip_proxy,fichero);
leer_fichero(Gpuerto_proxy,fichero);

Gnmapeos=0;
while (!fin) {
 leer_fichero(cadena,fichero);
 if (!strcmp(cadena,"fin")) fin=1;
 else {
  strcpy(Gmapeo[Gnmapeos].tipo,cadena);
  leer_fichero(Gmapeo[Gnmapeos].ip1,fichero);
  leer_fichero(Gmapeo[Gnmapeos].puerto1,fichero);
  leer_fichero(Gmapeo[Gnmapeos].ip2,fichero);
  leer_fichero(Gmapeo[Gnmapeos].puerto2,fichero);
```

```
  Gnmapeos++;
  }
 }

fclose(fichero);
}
```

- *guardar.h*

```
void guardar(void);
```

- *guardar.c*

```
#include "ganso.h"

void guardar(void)
{
FILE *fichero;
char cadena[MAXCAD],sufijo[MAXCAD];
time_t fecha_sec;
struct tm *fecha;
int f;

time(&fecha_sec);
fecha=localtime(&fecha_sec);
strftime(sufijo,MAXCAD,"_%H:%M:%S_%d-%m-%Y",fecha);

fichero=fopen("/etc/HOSTNAME","w");
fprintf(fichero,"%s.%s\n",Ghost,Gdominio);
fclose(fichero);

fichero=fopen("/etc/hosts","w");
fprintf(fichero,"127.0.0.1    localhost\n");
fprintf(fichero,"%s    %s.%s  %s\n",Gip,Ghost,Gdominio,Ghost);
fclose(fichero);

fichero=fopen("/etc/sysconfig/network-scripts/ifcfg-eth0","w");
fprintf(fichero,"DEVICE=eth0\n");
fprintf(fichero,"IPADDR=%s\n",Gip);
fprintf(fichero,"NETMASK=%s\n",Gmascara);
fprintf(fichero,"NETWORK=%s\n",Gsubred);
fprintf(fichero,"BROADCAST=%s\n",Gbroadcast);
fprintf(fichero,"ONBOOT=yes\n");
fprintf(fichero,"BOOTPROTO=none\n");
fclose(fichero);

fichero=fopen("/etc/sysconfig/network-scripts/ifcfg-eth1","w");
fprintf(fichero,"DEVICE=eth1\n");
fprintf(fichero,"IPADDR=0.0.0.0\n");
fprintf(fichero,"NETMASK=0.0.0.0\n");
fprintf(fichero,"NETWORK=0.0.0.0\n");
fprintf(fichero,"BROADCAST=0.0.0.0\n");
```

```
   fprintf(fichero,"ONBOOT=yes\n");
   fprintf(fichero,"BOOTPROTO=dhcp\n");
   fclose(fichero);

   fichero=fopen("/etc/sysconfig/ipchains.rules","w");
   fprintf(fichero,":input ACCEPT\n");
   fprintf(fichero,":forward DENY\n");
   fprintf(fichero,":output ACCEPT\n");
   fprintf(fichero,"-A forward -j MASQ -s %s/%s -d 0.0.0.0/0.0.0.0 -i
eth1\n",Gsubred,Gmascara);
   fclose(fichero);

   fichero=fopen("/etc/rc.d/mapeos","w");
   for (f=0;f<Gnmapeos;f++) fprintf(fichero,"/usr/sbin/ipmasqadm portfw -a -P
%s -L %s %s -R %s
%s\n",Gmapeo[f].tipo,Gmapeo[f].ip1,Gmapeo[f].puerto1,Gmapeo[f].ip2,Gmapeo[f].p
uerto2);
   fclose(fichero);
   system("/bin/chmod 755 /etc/rc.d/mapeos\n");

   fichero=fopen("/etc/squid/squid.conf","w");
   fprintf(fichero,"http_port 8080\n");
   fprintf(fichero,"cache_peer %s parent %s
%s\n",Gip_proxy,Gpuerto_proxy,Gpuerto_proxy);
   fprintf(fichero,"cache_dir /cache 1024 16 256\n");
   fprintf(fichero,"acl all src 0.0.0.0/0.0.0.0\n");
   fprintf(fichero,"acl manager proto cache_object\n");
   fprintf(fichero,"acl localhost src 127.0.0.1/255.255.255.255\n");
   fprintf(fichero,"acl SSL_ports port 443 563\n");
   fprintf(fichero,"acl Safe_ports port 80 21 443 563 70 210 1025-65535\n");
   fprintf(fichero,"acl purge method PURGE\n");
   fprintf(fichero,"acl CONNECT method CONNECT\n");
   fprintf(fichero,"http_access allow manager localhost\n");
   fprintf(fichero,"http_access deny manager\n");
   fprintf(fichero,"http_access allow purge localhost\n");
   fprintf(fichero,"http_access deny purge\n");
   fprintf(fichero,"http_access deny !Safe_ports\n");
   fprintf(fichero,"http_access deny CONNECT !SSL_ports\n");
   fprintf(fichero,"http_access allow all\n");
   fprintf(fichero,"icp_access allow all\n");
   fprintf(fichero,"miss_access allow all\n");
   fprintf(fichero,"never_direct allow all\n");
   fprintf(fichero,"dead_peer_timeout 0 seconds\n");
   fclose(fichero);

   sprintf(cadena,"/etc/ganso.conf%s",sufijo);
   rename("/etc/ganso.conf",cadena);
   fichero=fopen("/etc/ganso.conf","w");
   fprintf(fichero,"%s\n%s\n%s\n%s\n%s\n%s\n%s\n%s\n%s\n",Ghost,Gdominio,Gip,Gbroa
dcast,Gsubred,Gmascara,Gip_proxy,Gpuerto_proxy);
```

```
   for                                                    (f=0;f<Gnmapeos;f++)
fprintf(fichero,"%s\n%s\n%s\n%s\n%s\n",Gmapeo[f].tipo,Gmapeo[f].ip1,Gmapeo[f].
puerto1,Gmapeo[f].ip2,Gmapeo[f].puerto2);
   fprintf(fichero,"fin\n");
   fclose(fichero);
   }
```

- *configurar.c*

```
   #include "ganso.h"
   #include "guardar.h"
   #include "leer_config.h"
   #include "comprobacion.h"

   int modificado=0;

   void tecla(void);
   int leer(char *,char *);

   void menu(void);
   void mirar(void);
   void cambiar(void);
   int reiniciar(void);
   int parar(void);
   void agregar_mapeos(void);
   void quitar_mapeos(void);

   void tecla(void)
   {
   char cadena[MAXCAD];
   fgets(cadena,MAXCAD,stdin);
   }




   int leer(char *texto,char *cadena)
   {
   printf("%s: ",texto);
   fgets(cadena,MAXCAD,stdin);
   cadena[strlen(cadena)-1]='\0';
   if (!strcmp(cadena,"")) return(0);
   return(1);
   }



   void menu(void)
   {
   int f,fin=0;
   char cadena[MAXCAD];
```

```
while (!fin) {
    for(f=0;f<50;f++) printf("\n");
    printf("*****************\n");
    printf("* Configuracion *\n");
    printf("*****************\n");

    if (modificado) printf("\n - Hay modificaciones no guardadas. -\n\n");
    if (modificado)     printf("G - Guardar los cambios\n");

    printf("V - Ver parametros\n");
    printf("P - Cambiar parametros\n");
    printf("M - Agregar mapeos de puertos\n");
    printf("Q - Quitar mapeos de puertos\n");
    printf("R - Resetear maquina\n");
    printf("A - Apagar maquina\n");
    printf("S - Salir\n\n");

    leer("Seleccione una opcion",cadena);

    if (cadena[1]=='\0') switch (cadena[0]) {
           case 'v':
           case 'V': mirar(); break;
           case 'p':
           case 'P': cambiar(); break;
           case 'r':
           case 'R': fin=reiniciar(); break;
           case 'a':
           case 'A': fin=parar(); break;
           case 'g':
           case 'G': if (modificado) guardar(); modificado=0; break;
           case 'm':
           case 'M': agregar_mapeos(); break;
           case 'q':
           case 'Q': quitar_mapeos(); break;
           case 's':
           case 'S': fin=1; break;
           default:
           }

    if (!fin) printf("\n\n");
    }
}



void mirar(void)
{
int f;

printf("\n\n* Configuracion:\n");
printf("Nombre: %s.%s\n",Ghost,Gdominio);
printf("Direccion: %s\n",Gip);
```

```
   printf("Subred: %s / %s\n",Gsubred,Gmascara);
   printf("Broadcast: %s\n",Gbroadcast);
   printf("Proxy: %s %s\n",Gip_proxy,Gpuerto_proxy);

   printf("\n--- Presione INTRO para continuar ---\n");
   tecla();

   if (Gnmapeos) {
    printf("Mapeos de puertos:\n\n");
    for      (f=0;f<Gnmapeos;f++)      printf("De      %s:%s      a      %s:%s
(%s)\n",Gmapeo[f].ip1,Gmapeo[f].puerto1,Gmapeo[f].ip2,Gmapeo[f].puerto2,Gmapeo
[f].tipo);
    printf("\n--- Presione INTRO para continuar ---\n");
    tecla();
    }

   printf("Datos de la interfaz externa tomados por DHCP:\n\n");

   system("/usr/local/bin/parametrosdhcp");

   printf("\n--- Presione INTRO para continuar ---\n");
   tecla();
   }



   void cambiar(void)
   {
   char cadena[MAXCAD],cadena2[MAXCAD];
   int cambio,valido;

   printf("\n\n* Teclee los parametros de configuracion.\n");
   printf("Si  deja  algun  campo  en  blanco  se  qudara  con  su  valor  por
defecto.\n");
   printf("Si introduce un punto como valor de algun campo volvera al menu
principal.\n\n\n");

   valido=0;
   while (!valido) {
    sprintf(cadena,"Nombre del servidor [%s]",Ghost);
    cambio=leer(cadena,cadena2);
    valido=!cambio;
    if (!strcmp(cadena2,".")) return;
    else if (cambio) {
       valido=!haypuntos(cadena2);
       if (valido) {
              strcpy(Ghost,cadena2);
              modificado=1;
              }
       }
    }
```

```
valido=0;
while (!valido) {
 sprintf(cadena,"Nombre del dominio [%s]",Gdominio);
 cambio=leer(cadena,cadena2);
 valido=!cambio;
 if (!strcmp(cadena2,".")) return;
 else if (cambio) {
    valido=haypuntos(cadena2);
    if (valido) {
           strcpy(Gdominio,cadena2);
           modificado=1;
           }
    }
 }

valido=0;
while (!valido) {
 sprintf(cadena,"Direccion IP [%s]",Gip);
 cambio=leer(cadena,cadena2);
 valido=!cambio;
 if (!strcmp(cadena2,".")) return;
 else if (cambio) {
    valido=(haypuntos(cadena2)==3 && numericopuntos(cadena2));
    if (valido) {
           strcpy(Gip,cadena2);
           modificado=1;
           }
    }
 }

valido=0;
while (!valido) {
 sprintf(cadena,"Direccion de la subred [%s]",Gsubred);
 cambio=leer(cadena,cadena2);
 valido=!cambio;
 if (!strcmp(cadena2,".")) return;
 else if (cambio) {
    valido=(haypuntos(cadena2)==3 && numericopuntos(cadena2));
    if (valido) {
           strcpy(Gsubred,cadena2);
           modificado=1;
           }
    }
 }

valido=0;
while (!valido) {
 sprintf(cadena,"Mascara de la subred [%s]",Gmascara);
 cambio=leer(cadena,cadena2);
 valido=!cambio;
 if (!strcmp(cadena2,".")) return;
 else if (cambio) {
```

```
    valido=(haypuntos(cadena2)==3 && numericopuntos(cadena2));
    if (valido) {
           strcpy(Gmascara,cadena2);
           modificado=1;
           }
    }
 }

valido=0;
while (!valido) {
 sprintf(cadena,"Direccion de broadcast [%s]",Gbroadcast);
 cambio=leer(cadena,cadena2);
 valido=!cambio;
 if (!strcmp(cadena2,".")) return;
 else if (cambio) {
    valido=(haypuntos(cadena2)==3 && numericopuntos(cadena2));
    if (valido) {
           strcpy(Gbroadcast,cadena2);
           modificado=1;
           }
    }
 }

valido=0;
while (!valido) {
 sprintf(cadena,"IP del servidor proxy [%s]",Gip_proxy);
 cambio=leer(cadena,cadena2);
 valido=!cambio;
 if (!strcmp(cadena2,".")) return;
 else if (cambio) {
    valido=(haypuntos(cadena2)==3 && numericopuntos(cadena2));
    if (valido) {
           strcpy(Gip_proxy,cadena2);
           modificado=1;
           }
    }
 }

valido=0;
while (!valido) {
 sprintf(cadena,"Puerto del servidor proxy [%s]",Gpuerto_proxy);
 cambio=leer(cadena,cadena2);
 valido=!cambio;
 if (!strcmp(cadena2,".")) return;
 else if (cambio) {
    valido=!haypuntos(cadena2);
    if (valido) {
           strcpy(Gpuerto_proxy,cadena2);
           modificado=1;
           }
    }
 }
```

```
  printf("--- Presione INTRO para continuar ---\n");
  tecla();
  }



  int reiniciar(void)
  {
  char cadena[MAXCAD];

  printf("\n\n* Reiniciar\n\n");
  leer("Esta seguro? [s/N] ",cadena);

  if ( !strcmp(cadena,"s") || !strcmp(cadena,"S") ) {
      printf("La maquina se esta reiniciando. Por favor espere.\n");
      sleep(1);
      system("/sbin/shutdown -r now\n");
      return(1);
      }

  return(0);
  }



  int parar(void)
  {
  char cadena[MAXCAD];

  printf("\n\n* Parar\n\n");
  leer("Esta seguro? [s/N] ",cadena);

  if ( !strcmp(cadena,"s") || !strcmp(cadena,"S") ) {
      printf("La maquina se esta apagando.\nProceda a apretar el boton de
apagado dentro de aproximadamente cinco minutos.\n");
      sleep(1);
      system("/sbin/shutdown -h now\n");
      return(1);
      }

  return(0);
  }



  void agregar_mapeos(void)
  {
  int relleno,valido;
  char cadena[MAXCAD];
  struct Tmapeo nuevo;
```

```
   if (Gnmapeos==MAXPORT) {
    printf("\n\nSe ha alcanzado el limite de numero de puertos
mapeados.\n\n");
    return;
    }

   printf("\n\n* Teclee los datos del nuevo mapeo estatico.\n");
   printf("Si deja en blanco cualquier campo volvera al menu
principal.\n\n\n");

   valido=0;
   while (!valido) {
    relleno=leer("Protocolo (tcp/udp)",cadena);
    if (!relleno) return;
    else {
       valido=(!strcmp(cadena,"tcp") || !strcmp(cadena,"udp"));
       if (valido) strcpy(nuevo.tipo,cadena);
       }
    }

   valido=0;
   while (!valido) {
    relleno=leer("Direccion IP a mapear",cadena);
    if (!relleno) return;
    else {
       valido=(haypuntos(cadena)==3 && numericopuntos(cadena));
       if (valido) strcpy(nuevo.ip1,cadena);
       }
    }

   valido=0;
   while (!valido) {
    relleno=leer("Puerto a mapear",cadena);
    if (!relleno) return;
    else {
       valido=numerico(cadena);
       if (valido) strcpy(nuevo.puerto1,cadena);
       }
    }

   valido=0;
   while (!valido) {
    relleno=leer("Direccion IP de destino",cadena);
    if (!relleno) return;
    else {
       valido=(haypuntos(cadena)==3 && numericopuntos(cadena));
       if (valido) strcpy(nuevo.ip2,cadena);
       }
    }

   valido=0;
   while (!valido) {
```

```
    relleno=leer("Puerto de destino",cadena);
    if (!relleno) return;
    else {
       valido=numerico(cadena);
       if (valido) strcpy(nuevo.puerto2,cadena);
       }
    }

   strcpy(Gmapeo[Gnmapeos].tipo,nuevo.tipo);
   strcpy(Gmapeo[Gnmapeos].ip1,nuevo.ip1);
   strcpy(Gmapeo[Gnmapeos].puerto1,nuevo.puerto1);
   strcpy(Gmapeo[Gnmapeos].ip2,nuevo.ip2);
   strcpy(Gmapeo[Gnmapeos].puerto2,nuevo.puerto2);
   Gnmapeos++;

   modificado=1;
   }



   void quitar_mapeos(void)
   {
   int f,numero=0,valido,relleno;
   char cadena[MAXCAD],cadena2[MAXCAD];

   if (!Gnmapeos) {
    printf("\n\nNo hay mapeos de puertos definidos.\n\n");
    printf("\n--- Presione INTRO para continuar ---\n");
    tecla();
    return;
    }

   printf("Mapeos de puertos:\n\n");
   for    (f=0;f<Gnmapeos;f++)    printf("%03i)    De    %s:%s    a    %s:%s
(%s)\n",f+1,Gmapeo[f].ip1,Gmapeo[f].puerto1,Gmapeo[f].ip2,Gmapeo[f].puerto2,Gm
apeo[f].tipo);

   valido=0;
   while (!valido) {
    sprintf(cadena2,"\nMapeo a borrar (1/%i)",Gnmapeos);
    relleno=leer(cadena2,cadena);
    if (!relleno) return;
    else {
     if (numerico(cadena)) {
      numero=atoi(cadena);
      if (numero>=1 && numero<=Gnmapeos) valido=1;
      }
     }
    }

   for (f=numero-1;f<Gnmapeos;f++) {
    strcpy(Gmapeo[f].tipo,Gmapeo[f+1].tipo);
```

```
 strcpy(Gmapeo[f].ip1,Gmapeo[f+1].ip1);
 strcpy(Gmapeo[f].puerto1,Gmapeo[f+1].puerto1);
 strcpy(Gmapeo[f].ip2,Gmapeo[f+1].ip2);
 strcpy(Gmapeo[f].puerto2,Gmapeo[f+1].puerto2);
 }

Gnmapeos--;
modificado=1;


printf("\nEl mapeo numero %i ha sido borrado.",numero);
printf("\n--- Presione INTRO para continuar ---\n");
tecla();
}



void main(void)
{
leer_config();
menu();
}
```

Anexo 1: Manual de usuario

A continuación se describe, a modo de ejemplo, un manual de usuario del software desarrollado pensado para el usuario final. Para una posible liberación al mercado de una caja negra[27] basada en éste, sería necesario añadir apartados para la configuración hardware de la misma, así como de la forma de conexión de esta a la red.

No se incluyen apartados sobre la funcionalidad de la solución o sobre futuras ampliaciones, dado que ya se han cubierto en anteriores puntos de este proyecto. Tampoco se incluye un apartado de solución de problemas, dado que éste dependería de la implementación física que se hiciera de la especificación propuesta.

[27] Se denominan así a productos que, pese a estar desarrollados sobre un ordenador concreto, presentan al usuario una apariencia personalizada que oculta su funcionamiento interno.

A1.1 – Configuración del servidor

- *Datos a conocer*

Antes de configurar el servidor, es necesario conocer una serie de parámetros que posteriormente habrá que indicar durante la configuración del sistema.

- ✓ `Configuración IP` : será necesario conocer la dirección IP del sistema, así como su ubicación dentro de la red a través de la máscara de subred y las direcciones de red y de broadcast (normalmente coincidentes con la primera y la última dirección del rango de la red respectivamente).
- ✓ `Servidor Proxy` : necesitamos conocer la dirección IP del servidor maestro al que se accede, así como el puerto por el que responde éste.
- ✓ `Dirección externa` : en el caso de recibir siempre la misma configuración por DHCP, es necesario conocerla para poder dar acceso desde el exterior a máquinas internas.
- ✓ `Servicios internos` : es necesario conocer tanto las direcciones de los servidores internos que van a ofrecer servicios al exterior como los puertos por los que ofrecen servicio. Asimismo, debemos conocer las direcciones y los puertos externos por los que los haremos accesibles.

- *Configuración inicial por puerto serie*

Dado que en un primer acceso no dispondremos de una dirección IP configurada en el servidor, necesitaremos conectar un cable serie de tipo *"Null-Modem"* desde un ordenador cercano al servidor hasta el puerto serie de éste. Podremos utilizar cualquier emulador de terminal como `Hyperterminal` en sistemas Windows o `minicom` en sistemas tipo Unix, configurándolo a 9600 baudios, 8 bits de datos, 0 bits de paridad y 1 bit de parada.

Una vez conectados, tras pulsar la tecla 'Intro', obtendremos el siguiente menú de configuración:

```
*****************
* Configuracion *
*****************
V - Ver parametros
P - Cambiar parametros
M - Agregar mapeos de puertos
Q - Quitar mapeos de puertos
R - Resetear maquina
A - Apagar maquina
S - Salir
```

Dado que ahora mismo la máquina se encuentra desconfigurada, pasaremos directamente a la configuración, a la cual se accede pulsando 'P' seguido de la tecla 'Intro', tras lo cual obtendremos esta información:

```
* Teclee los parametros de configuracion.
Si deja algun campo en blanco se qudara con su valor por defecto.
Si introduce un punto como valor de algun campo volvera al menu principal.
```

A continuación se nos pedirán una serie de parámetros que se detallan a continuación:

- `Nombre del servidor` : nombre no cualificado de la máquina a instalar. Conviene que haya una entrada estática en el DNS interno para este nombre. P.ej: "ganso"
- `Nombre del dominio` : dominio completo al que pertenecerá la máquina. P.ej: "fie.us.es"
- `Dirección IP` : dirección dentro de la red interna. P.ej: "192.168.0.1"
- `Dirección de la subred` : dirección de la red interna. P.ej: "192.168.0.0"
- `Mascara de la subred` : máscara de la red interna. P.ej: "255.255.255.0"
- `Dirección de broadcast` : dirección de broadcast de la red interna. P.ej: "192.168.0.255"
- `IP del servidor proxy` : dirección del proxy maestro. P.ej: "150.214.5.77"
- `Puerto del servidor proxy` : puerto de conexión del servicio de proxy en el servidor proxy maestro. P.ej: "3128"

Una vez hecho esto volveremos al menú principal, aunque ahora aparecerá una nueva opción.

```
******************
* Configuracion *
******************

 - Hay modificaciones no guardadas. -

G - Guardar los cambios
V - Ver parametros
P - Cambiar parametros
M - Agregar mapeos de puertos
Q - Quitar mapeos de puertos
R - Resetear maquina
A - Apagar maquina
S – Salir
```

Seleccionando esta nueva opción ('G') volveremos al menú inicial. Aunque ya se han guardado las últimas modificaciones, aún no se encuentran activas. Para ello tendremos que reiniciar la máquina, seleccionando la opción 'R'.

Una vez elegida esta opción, y tras unos 5 minutos de espera, volveremos a ver el menú inicial. En este punto, la máquina ya se encuentra configurada y podemos empezar a configurar los clientes según se ve en el siguiente punto.

- *Accesos posteriores por telnet*

Una vez realizada esta configuración inicial, ya podemos desconectar el cable serie y acceder cómodamente al servidor usando un cliente estándar de *telnet* desde cualquier máquina a ambos lados de

la red. En el caso de desconocer la dirección pública existente en la parte externa, podemos conectarnos desde una máquina interna y ver la configuración con la opción 'V'. Dentro de los datos obtenidos por DHCP encontramos una línea "fixed-address" que indica este dato.

En el caso de querer transladar físicamente el servidor, o de haber sido planeado un corte de alimentación eléctrica, tendremos que usar la opción 'A' para apagar la máquina. Tras confirmar el apagado, habrá que esperar 5 minutos para poder apagar físicamente el servidor. Una vez realizado el translado, o una vez recuperada la alimentación eléctrica, encendiendo físicamente la máquina y esperando otros 5 minutos podremos volver a usarla.

- *Mapeos de puertos*

En el caso de tener máquinas en la subred interna cuyos servicios queramos que sean visibles desde fuera, será necesario añadir mapeos estáticos de determinados puertos del router a otros de los servidores internos. Para ello usaremos la opción 'M' de menú principal.

A continuación habrá que dar una serie de parámetros sobre el nuevo mapeo a realizar. En cualquier momento se puede cancelar el proceso dejando un campo en blanco. Los parámetros a introducir son los siguentes:

- `Procolo` : será "tcp" o "udp" según el tipo de puerto.
- `Dirección IP a mapear` : dirección IP externa (normalmente la misma dirección del router).
- `Puerto` : puerto externo a mapear.
- `Dirección IP de destino` : dirección IP del servidor interno.
- `Puerto de destino` : puerto por el que ofrece servicio el servidor interno.

Tras responder a estas cuestiones volveremos al menú principal.

En el caso de haber añadido erróneamente un puerto, o de haber cambiado los datos de éstos, podemos eliminarlos con la opción 'Q' del menú. Tras elegir esta opción aparecerá una lista de los mapeos de puertos existentes, y se nos preguntará cuál de ellos queremos borrar.

De haber entrado por error en esta opción, podemos pulsar simplemente 'Intro' para volver al menú principal.

Tras agegar o eliminar cualquier puerto tendremos que seleccionar de nuevo las opciones 'G' y 'R' para guardar la configuración y reinicar la máquina respectivamente. En el caso de haber realizado cambios que no queramos guardar, pulsando 'S' saldremos del programa de configuración sin modificar ésta.

A1.2 – Configuración de los ordenadores cliente

Una vez configurado el servidor, en cada uno de los ordenadores clientes deberemos configurar los siguientes parámetros en su configuración:

- ✓ `Dirección IP` : tanto la dirección como la máscara de subred deben ser adecuadas para que sea visible el servidor de enrutado.
- ✓ `Ruta por defecto` : debe coincidir con la dirección IP del servidor.
- ✓ `Servidor DNS` : debe coincidir con la dirección IP del servidor.
- ✓ `Proxy` : debe configurarse que el navegador predeterminado acceda al servidor a través de su dirección IP, usando el puerto 8080.

Para un cliente Windows 2000 el proceso sería el siguente:

- **Configuración de la tarjeta de red**
  - ✓ Desde el menú de inicio, se selecciona el menú "Configuración" y se entra en "Panel de control".
  - ✓ Se selecciona el icono "Conexiones de red y de acceso telefónico" con un doble clic.
  - ✓ Se selecciona el icono "Conexión de área local" con un doble clic.
  - ✓ Se pulsa el icono "Aceptar".
  - ✓ Se selecciona "Protocolo Internet (TCP/IP) con un doble clic.
  - ✓ Se especifican direcciones IP y de DNS según lo especificado anteriormente.
  - ✓ Se pulsa el botón "Aceptar".
  - ✓ Se pulsa el botón "Aceptar".
  - ✓ Se pulsa el botón "Cerrar".

- **Configuración del navegador de Internet**
  - ✓ Se accede como antes al Panel de Control.
  - ✓ Se selecciona el icono "Opciones de Internet" con un doble clic.
  - ✓ Se selecciona la pestaña "Conexiones".
  - ✓ Se pulsa el botón "Configuración LAN".
  - ✓ Se selecciona la opción "Usar servidor proxy".
  - ✓ Se rellena la dirección y el puerto según lo especificado anteriormente.
  - ✓ Se pulsa el botón "Aceptar".
  - ✓ Se pulsa el botón "Aceptar".

Anexo 2: Páginas de manual y RFCs

En general, los sistemas Unix y, por extensión, los sistemas tipo GNU están especialente bien documentado electrónicamente. Así, cada comando del sistema (y cada programa externo que quiera cumplir los estándares) tiene una completa información de uso en un sistema centralizado de páginas de manual. Debido, además, a la expansión de Internet y su influencia en el desarrollo de GNU/Linux, la documentación existente en sistemas Unix tradicionales se ha revisado y completado gracias a la aportación de la comunidad internacional.

Paralelamente, y como una propuesta de estandarización del mundo Internet, se ha generado una serie de documentos llamados RFC (Request For Comment) *[RFC]* donde se proponen los nuevos estándares a usar en la Red. Gracias a estos documentos, que nacieron en 1969 durante el auge de la red Arpanet, se asegura que todas las aplicaciones que hagan uso de un protocolo concreto puedan intercambiar información sin peligro de incompatibilidades. De hecho, generalmente en la documentación de las aplicaciones diseñadas para Internet se suele referenciar al RFC correspondiente a los protocolos usados, para asegurar la compatibilidad con otras aplicaciones que puedan interactuar con ellas.

En este punto incluímos, a modo de referencia, las páginas de manual de Linux asociadas a los comandos descritos. También se anexa una lista de RFCs considerados interesantes para ampliar información sobre los protocolos utilizados, aunque en este caso no se ha decidido incluír su texto íntegro debido a su extensión.

A2.1 – Página de manual de ifconfig

```
IFCONFIG(8)         Linux Programmer's Manual         IFCONFIG(8)

NAME
       ifconfig - configure a network interface

SYNOPSIS
       ifconfig [interface]
       ifconfig interface [aftype] options | address ...

DESCRIPTION
       Ifconfig  is used to configure the kernel-resident network
       interfaces.  It is used at boot time to set up  interfaces
       as  necessary.  After that, it is usually only needed when
       debugging or when system tuning is needed.

       If no arguments are given, ifconfig displays the status of
       the  currently  active  interfaces.  If a single interface
       argument is given, it displays the  status  of  the  given
       interface  only; if a single -a argument is given, it dis-
       plays the status of all interfaces, even  those  that  are
       down.  Otherwise, it configures an interface.
Address Families
       If  the  first argument after the interface name is recog-
       nized as the name of  a  supported  address  family,  that
       address  family  is  used  for decoding and displaying all
       protocol addresses.  Currently supported address  families
       include  inet  (TCP/IP, default), inet6 (IPv6), ax25 (AMPR
       Packet Radio), ddp (Appletalk Phase 2), ipx  (Novell  IPX)
       and netrom (AMPR Packet radio).

OPTIONS
       interface
              The  name  of  the  interface.   This  is usually a
              driver name followed by a unit number, for  example
              eth0 for the first Ethernet interface.

       up     This flag causes the interface to be activated.  It
              is implicitly specified if an address  is  assigned
              to the interface.

       down   This  flag  causes the driver for this interface to
              be shut down.

       [-]arp Enable or disable the use of the  ARP  protocol  on
              this interface.

       [-]promisc
              Enable  or  disable  the  promiscuous  mode  of the
```

```
       interface.  If selected, all packets on the network
       will be received by the interface.

[-]allmulti
       Enable or disable all-multicast mode.  If selected,
       all  multicast  packets  on  the  network  will  be
       received by the interface.

metric N
       This parameter sets the interface metric.

mtu N  This parameter sets the Maximum Transfer Unit (MTU)
       of an interface.
dstaddr addr
       Set the remote IP address for a point-to-point link
       (such  as  PPP).  This keyword is now obsolete; use
       the pointopoint keyword instead.

netmask addr
       Set the IP network mask for this  interface.   This
       value defaults to the usual class A, B or C network
       mask (as derived from the  interface  IP  address),
       but it can be set to any value.

add addr/prefixlen
       Add an IPv6 address to an interface.

del addr/prefixlen
       Remove an IPv6 address from an interface.

tunnel aa.bb.cc.dd
       Create  a new SIT (IPv6-in-IPv4) device, tunnelling
       to the given destination.

irq addr
       Set the interrupt line used by  this  device.   Not
       all  devices  can dynamically change their IRQ set-
       ting.

io_addr addr
       Set the start address in I/O space for this device.

mem_start addr
       Set  the  start  address  for shared memory used by
       this device.  Only a few devices need this.

media type
       Set the physical port or medium type to be used  by
       the  device.   Not all devices can change this set-
       ting, and those that can vary in what  values  they
       support.  Typical values for type are 10base2 (thin
       Ethernet), 10baseT (twisted-pair 10Mbps  Ethernet),
```

```
              AUI  (external transceiver) and so on.  The special
              medium type of auto can be used to tell the  driver
              to  auto-sense  the  media.  Again, not all drivers
              can do this.

       [-]broadcast [addr]
              If the address argument is given, set the  protocol
              broadcast  address  for this interface.  Otherwise,
              set (or  clear)  the  IFF_BROADCAST  flag  for  the
              interface.

       [-]pointopoint [addr]
              This  keyword enables the point-to-point mode of an
              interface, meaning that it is a direct link between
              two machines with nobody else listening on it.
              If the address argument is also given, set the pro-
              tocol address of the other side of the  link,  just
              like the obsolete dstaddr keyword does.  Otherwise,
              set or  clear  the  IFF_POINTOPOINT  flag  for  the
              interface.

       hw class address
              Set  the hardware address of this interface, if the
              device driver supports this operation.  The keyword
              must  be followed by the name of the hardware class
              and the printable ASCII equivalent of the  hardware
              address.    Hardware  classes  currently  supported
              include ether (Ethernet), ax25 (AMPR AX.25), ARCnet
              and netrom (AMPR NET/ROM).

       multicast
              Set  the  multicast  flag  on  the  interface. This
              should not normally be needed as  the  drivers  set
              the flag correctly themselves.

       address
              The IP address to be assigned to this interface.

       txqueuelen length
              Set the length of the transmit queue of the device.
              It is useful to set this to small values for slower
              devices  with a high latency (modem links, ISDN) to
              prevent fast bulk transfers from disturbing  inter-
              active traffic like telnet too much.

FILES
       /proc/net/socket
       /proc/net/dev
       /proc/net/dev
       /proc/net/if_inet6
       /etc/init.d/network
```

```
BUGS
       While  appletalk  DDP  and IPX addresses will be displayed
       they cannot be altered by this command.

SEE ALSO
       route(8), netstat(8), arp(8), rarp(8)

AUTHORS
       Fred N. van Kempen, <waltje@uwalt.nl.mugnet.org>
       Alan Cox, <Alan.Cox@linux.org>
       Phil Blundell, <Philip.Blundell@pobox.com>
```

A2.2 – Página de manual de dhclient

dhclient(8) dhclient(8)

NAME

dhclient - Dynamic Host Configuration Protocol Client

SYNOPSIS

dhclient [-p port] [-d] [if0 [...ifN]]

DESCRIPTION

The Internet Software Consortium DHCP Client, dhclient, provides a means for configuring one or more network interfaces using the Dynamic Host Configuration Protocol, BOOTP protocol, or if these protocols fail, by statically assigning an address.

OPERATION

The DHCP protocol allows a host to contact a central server which maintains a list of IP addresses which may be assigned on one or more subnets. A DHCP client may request an address from this pool, and then use it on a temporary basis for communication on network. The DHCP protocol also provides a mechanism whereby a client can learn important details about the network to which it is attached, such as the location of a default router, the location of a name server, and so on.

On startup, dhclient reads the dhclient.conf for configuration instructions. It then gets a list of all the network interfaces that are configured in the current system. For each interface, it attempts to configure the interface using the DHCP protocol.

In order to keep track of leases across system reboots and server restarts, dhclient keeps a list of leases it has been assigned in the dhclient.leases(5) file. On startup, after reading the dhclient.conf file, dhclient reads the dhclient.leases file to refresh its memory about what leases it has been assigned.

When a new lease is acquired, it is appended to the end of the dhclient.leases file. In order to prevent the file from becoming arbitrarily large, from time to time dhclient creates a new dhclient.leases file from its in-core lease database. The old version of the dhclient.leases file is retained under the name dhcpd.leases~ until the next time dhclient rewrites the database.

Old leases are kept around in case the DHCP server is unavailable when dhclient is first invoked (generally during the initial system boot process). In that event, old leases from the dhclient.leases file which have not yet expired are tested, and if they are determined to be valid, they are used until either they expire or the DHCP server becomes available.

A mobile host which may sometimes need to access a network on which no DHCP server exists may be preloaded with a lease for a fixed address on that network. When all attempts to contact a DHCP server have failed, dhclient will try to validate the static lease, and if it succeeds, will use that lease until it is restarted.

A mobile host may also travel to some networks on which DHCP is not available but BOOTP is. In that case, it may be advantageous to arrange with the network administrator for an entry on the BOOTP database, so that the host can boot quickly on that network rather than cycling through the list of old leases.

COMMAND LINE

The names of the network interfaces that dhclient should attempt to configure may be specified on the command line. If no interface names are specified on the command line dhclient will identify all network interfaces, elimininating non-broadcast interfaces if possible, and attempt to configure each interface.

If dhclient should listen and transmit on a port other than the standard (port 68), the -p flag may used. It should be followed by the udp port number that dhclient should use. This is mostly useful for debugging purposes. If the -p flag is specified, the client will transmit responses to servers at a port number that is one less than the one specified - i.e., if you specify -p 68, then the client will listen on port 68 and transmit to port 67. Datagrams that must go through relay agents are sent to the port number specified with the -p flag - if you wish to use alternate port numbers, you must configure any relay agents you are using to use the same alternate port numbers.

Dhclient will normally run in the foreground until it has configured an interface, and then will revert to running in the background. To run force dhclient to always run as a foreground process, the -d flag should be specified. This is useful when running dhclient under a debugger, or when running it out of inittab on System V systems.

CONFIGURATION

The syntax of the dhclient.conf(8) file is discussed seperately.

FILES

/etc/dhclient.conf, /var/lib/dhcp /dhclient.leases, /var/run/dhclient.pid, /var/lib/dhcp /dhclient.leases~.

SEE ALSO

dhcpd(8), dhcrelay(8), dhclient.conf(5), dhclient.leases(5)

AUTHOR

dhclient(8) has been written for the Internet Software Consortium by Ted Lemon <mellon@fugue.com> in cooperation with Vixie Enterprises. To learn more about the Internet Software Consortium, see http://www.vix.com/isc. To learn more about Vixie Enterprises, see http://www.vix.com.

This client was substantially modified and enhanced by Elliot Poger for use on Linux while he was working on the MosquitoNet project at Stanford.

The current version owes much to Elliot's Linux enhancements, but was substantially reorganized and partially rewritten by Ted Lemon so as to use the same networking framework that the Internet Software Consortium DHCP server uses. Much system-specific configuration code was moved into a shell script so that as support for more operating systems is added, it will not be necessary to port and maintain system-specific configuration code to these operating systems - instead, the shell script can invoke the native tools to accomplish the same purpose.

A2.3 – Página de manual de ipchains

IPCHAINS(8) IPCHAINS(8)

NAME

ipchains - IP firewall administration

SYNOPSIS

```
ipchains -[ADC] chain rule-specification [options]
ipchains -[RI] chain rulenum rule-specification [options]
ipchains -D chain rulenum [options]
ipchains -[LFZNX] [chain] [options]
ipchains -P chain target [options]
ipchains -M [ -L | -S ] [options]
```

DESCRIPTION

Ipchains is used to set up, maintain, and inspect the IP firewall rules in the Linux kernel. These rules can be divided into 4 different categories: the IP input chain, the IP output chain, the IP forwarding chain, and user defined chains.

For each of these categories, a separate table of rules is maintained, any of which might refer to one of the user-defined chains. See ipfw(4) for more details.

TARGETS

A firewall rule specifies criteria for a packet, and a target. If the packet does not match, the next rule in the chain is the examined; if it does match, then the next rule is specified by the value of the target, which can be the name of a user-defined chain, or one of the special values ACCEPT, DENY, REJECT, MASQ, REDIRECT, or RETURN.

ACCEPT means to let the packet through. DENY means to drop the packet on the floor. REJECT means the same as drop, but is more polite and easier to debug, since an ICMP message is sent back to the sender indicating that the packet was dropped. (Note that DENY and REJECT are the same for ICMP packets).

MASQ is only legal for the forward and user defined chains, and can only be used when the kernel is compiled with CONFIG_IP_MASQUERADE defined. With this, packets will be masqueraded as if they originated from the local host. Furthermore, reverse packets will be recognized as such and they will be demasqueraded automatically, bypassing the forwarding chain.

REDIRECT is only legal for the input and user-defined chains and can only be used when the Linux kernel is compiled with CONFIG_IP_TRANSPARENT_PROXY defined. With

this, packets will be redirected to a local socket, even if they were sent to a remote host. If the specified redirection port is 0, which is the default value, the destination port of a packet will be used as the redirection port. When this target is used, an optional extra argument (the port number) can be supplied.
If the end of a user-defined chain is reached, or a rule with target RETURN is matched, then the next rule in the previous (calling) chain is examined. If the end of a builtin chain is reached, or a rule in a builtin chain with target RETURN is matched, the target specified by the chain policy determines the fate of the packet.

OPTIONS

The options that are recognized by ipchains can be divided into several different groups.

COMMANDS

These options specify the specific action to perform; only one of them can be specified on the command line, unless otherwise specified below. For all the long versions of the command and option names, you only need to use enough letters to ensure that ipchains can differentiate it from all other options.

-A, --append

Append one or more rules to the end of the selected chain. When the source and/or destination names resolve to more than one address, a rule will be added for each possible address combination.

-D, --delete

Delete one or more rules from the selected chain. There are two versions of this command: the rule can be specified as a number in the chain (starting at 1 for the first rule) or a rule to match.

-R, --replace

Replace a rule in the selected chain. If the source and/or destination names resolve to multiple addresses, the command will fail. Rules are numbered starting at 1.

-I, --insert

Insert one or more rules in the selected chain as the given rule number. So, if the rule number is 1, the rule or rules are inserted at the head of the chain.

-L, --list

List all rules in the selected chain. If no chain is selected, all chains are listed. It is legal to

specify the -Z (zero) option as well, in which case no chain may be specified. The exact output is effected by the other arguments given.

-F, --flush
Flush the selected chain. This is equivalent to deleting all the rules one by one.

-Z, --zero
Zero the packet and byte counters in all chains. It is legal to specify the -L, --list (list) option as well, to see the counters immediately before they are cleared; if this is done, then no specific chain can be specified (they will all be displayed and cleared.

-N, --new-chain
Create a new user-defined chain of the given name. There must be no target of that name already.

-X, --delete-chain
Delete the specified user-defined chain. There must be no references to the chain (if there are you must delete or replace the referring rules before the chain can be deleted). If no argument is given, it will attempt to delete every non-builtin chain.

-P, --policy
Set the policy for the chain to the given target. See the section TARGETS for the legal targets. Only non-userdefined chains can have policies, and neither built-in nor user-defined chains can be policy targets.

-M, --masquerading
This option allows viewing of the currently masqueraded connections (in conjuction with the -L option) or to set the kernel masqerading parameters (with the -S option).

-S, --set tcp tcpfin udp
Change the timeout values used for masquerading. This command always takes 3 parameters, representing the timeout values (in seconds) for TCP sessions, TCP sessions after receiving a FIN packet, and UDP packets, respectively. A timeout value 0 means that the current timeout value of the corresponding entry is preserved. This option is only allowed in combination with the -M flag.

-C, --check

Check the given packet against the selected chain. This is extremely useful for testing, as the same kernel routines used to check "real" network packets are used to check this packet. It can be used to check user-defined chains as well as the builtin ones. The same arguments used to specify firewall rules are used to construct the packet to be tested. In particular, the -s (source), -d (destination), -p (protocol), and -i (interface) flags are compulsory.

-h Help. Give a (currently very brief) description of the command syntax.

PARAMETERS

The following parameters make up a rule specification (as used in the add, delete, replace, append and check commands).

-p, --protocol[!] protocol

The protocol of the rule or of the packet to check. The specified protocol can be one of tcp, udp, icmp, or all, or it can be a numeric value, representing one of these protocols or a different one. Also a protocol name from /etc/protocols is allowed. A "!" argument before the protocol inverts the test. The number zero is equivalent to all. Protocol all will match with all protocols and is taken as default when this option is omitted. All may not be used in in combination with the check command.

-s, --source [!] address[/mask] [!] [port[:port]]

Source specification. Address can be either a hostname, a network name, or a plain IP address. The mask can be either a network mask or a plain number, specifying the number of 1's at the left side of the network mask. Thus, a mask of 24 is equivalent to 255.255.255.0. A "!" argument before the address specification inverts the sense of the address.

The source may include a port specification or ICMP type. This can either be a service name, a port number, a numeric ICMP type, or one of the ICMP type names shown by the command

ipchains -h icmp Note that many of these ICMP names refer to both a type and code, meaning that an ICMP code after the -d flag is illegal. In the rest of this paragraph, a port means either a port specification or an ICMP type. An inclusive range is can also be specified, using the format port:port. If the first port is omitted, "0" is assumed; if the

```
       last is omitted, "65535" is assumed.
       Ports may only be specified in combination with the
       tcp, udp, or icmp protocols.  A "!" before the port
       specification  inverts  the  sense.  When the check
       command is specified, exactly one port is required,
       and  if  the  -f  (fragment)  flag is specified, no
       ports are allowed.  The flag --src is a convenience
       alias for this option.

--source-port [!] [port[:port]]
       This  allows  separate  specifiction  of the source
       port or port range.  See the description of the  -s
       flag above for details.The flag --sport is an alias
       for this option.

-d, --destination [!] address[/mask] [!] [port[:port]]
       Destination specification.  See the  desciption  of
       the  -s (source) flag for a detailed description of
       the syntax.  For ICMP, which does not have ports, a
       "destination port" refers to the numeric ICMP code.
       The flag --dst is  a  convenience  alias  for  this
       option.

--destination-port [!] [port[:port]]
       This  allows  separate  specifiction  of the ports.
       See the description of the  -s  flag  for  details.
       The flag --dport is an alias for this option.

--icmp-type [!] typename
       This allows specification of the ICMP type (use the
       -h icmp option to see valid ICMP type names).  This
       is  often  more  convenient  to appending it to the
       destination specification.

-j, --jump target
       This specifies the target of the rule; ie. what  to
       do  if  the packet matches it.  The target can be a
       user-defined chain (not the one this rule is in) or
       one of the special targets which decide the fate of
       the packet immediately.  If this option is  omitted
       in  a  rule,  then  matching  the rule will have no
       effect on the packet's fate, but  the  counters  on
       the rule will be incremented.

-i, --interface [!] name
       Optional name of an interface via which a packet is
       received (for packets entering the input chain), or
       via  which is packet is going to be sent (for pack-
       ets entering the forward or output  chains).   When
       this   option  is  omitted,  the  empty  string  is
       assumed, which has a special meaning and will match
       with any interface name.  When the "!"  argument is
```

used before the interface name, the sense is inverted. If the interface name ends in a "+", then any interface which begins with this name will match.

[!] -f, --fragment
This means that the rule only refers to second and furthur fragments of fragmented packets. Since there is no way to tell the source or destination ports of such a packet (or ICMP type), such a packet will not match any rules which specify them. When the "!" argument precedes the "-f" flag, the sense is inverted.

OTHER OPTIONS
The following additional options can be specified:

-b, --bidirectional
Bidirectional mode. The rule will match with IP packets in both directions; this has the same effect as repeating the rule with the source & destination reversed.

-v, --verbose
Verbose output. This option makes the list command show the interface address, the rule options (if any), and the TOS masks. The packet and byte counters are also listed, with the suffix 'K', 'M' or 'G' for 1000, 1,000,000 and 1,000,000,000 multipliers respectively (but see the -x flag to change this). When used in combination with -M, information related to delta sequence numbers will also be listed. For appending, insertion, deletion and replacement, this causes detailed information on the rule or rules to be printed.

-n, --numeric
Numeric output. IP addresses and port numbers will be printed in numeric format. By default, the program will try to display them as host names, network names, or services (whenever applicable).

-l, --log
Turn on kernel logging of matching packets. When this option is set for a rule, the Linux kernel will print some information of all matching packets (like most IP header fields) via printk().

-o, --output [maxsize]
Copy matching packets to the userspace device. This is currently mainly for developers who want to play with firewalling effects in userspace. The

optional maxsize argument can be used to limit the maximum number of bytes from the packet which are to be copied. This option is only valid if the kernel has been compiled with CONFIG_IP_FIREWALL_NETLINK set.

-m, --mark markvalue
Mark matching packets. Packets can be marked with a 32-bit unsigned value which may (one day) change how they are handled internally. If you are not a kernel hacker you are unlikely to care about this. If the string markvalue begins with a + or -, then this value will be added or subtracted from the current marked value of the packet (which starts at zero).

-t, --TOS andmask xormask
Masks used for modifying the TOS field in the IP header. When a packet matches a rule, its TOS field is first bitwise and'ed with first mask and the result of this will be bitwise xor'ed with the second mask. The masks should be specified as hexadecimal 8-bit values. As the LSB of the TOS field must be unaltered (RFC 1349), TOS values which would cause it to be altered are rejected, as are any rules which always set more than TOS bit. Rules which might set multiple TOS bits for certain packets result in warnings (sent to stdout) which can be ignored if you know that packets with those TOS values will never reach that rule. Obviously, manipulating the TOS is a meaningless gesture if the rule's target is DENY or REJECT.

-x, --exact
Expand numbers. Display the exact value of the packet and byte counters, instead of only the rounded number in K's (multiples of 1000) M's (multiples of 1000K) or G's (multiples of 1000M). This option is only relevent for the -L command.

[!] -y, --syn
Only match TCP packets with the SYN bit set and the ACK and FIN bits cleared. Such packets are used to request TCP connection initiation; for example, blocking such packets coming in an interface will prevent incoming TCP connections, but outgoing TCP connections will be unaffected. This option is only meaningful when the protocol type is set to TCP. If the "!" flag precedes the "-y", the sense of the option is inverted.

--line-numbers

```
              When listing rules, add line numbers to the  begin-
              ning  of  each  rule,  corresponding to that rule's
              position in the chain.

       --no-warnings
              Disable all warnings.

FILES
       /proc/net/ip_fwchains
       /proc/net/ip_masquerade

DIAGNOSTICS
       Various error messages are printed to standard error.  The
       exit  code  is  0  for  correct functioning.  Errors which
       appear to be caused by  invalid  or  abused  command  line
       parameters cause an exit code of 2, and other errors cause
       an exit code of 1.

BUGS
       If input is a terminal, and a  rule  is  inserted  in,  or
       appended to, the forward chain, and IP forwarding does not
       seem to be enabled, and --no-warnings is not specified,  a
       message  is  printed  to  standard output, warning that no
       forwarding will occur until this is rectified.  This is to
       help users unaware of the requirement (which did not exist
       in the 2.0 kernels).

       There is no way to reset the packet and byte  counters  in
       one chain only.  This is a kernel limitation.

       Loop  detection is not done in ipchains; packets in a loop
       get dropped and logged, but that's the first  you'll  find
       out about it if you inadvertantly create a loop.

       The  explanation  of  what  effect marking a packet has is
       intentionally vague until documentation describing the new
       2.1 kernel's packet scheduling routines is released.

       There  is no way to zero the policy counters (ie. those on
       the built-in chains).

NOTES
       This ipchains is very different from the  ipfwadm  by  Jos
       Vos,  as it uses the new IP firewall trees.  Its function-
       ality is a superset of ipfwadm, and there is  generally  a
       1:1  mapping of commands.  I believe the new command names
       are more rational.  There are, however, a few  changes  of
       which you should be aware.

       Fragments  are  handled  differently.  All fragments after
       the first used to be let through (which is usually  safe);
       they  can  now  be  filtered.   This means that you should
```

probably add an explicit rule to accept fragments if you are converting over. Also, look for old accounting rules which check for source and destination ports of 0xFFFF (0xFF for ICMP packets) which was the old way of doing accounting on fragments.

Accounting rules are now simply integrated into the input and output chains; you can simulate the old behaviour like so:

```
 ipchains -N acctin
 ipchains -N acctout
 ipchains -N acctio
 ipchains -I input -j acctio
 ipchains -I input -j acctin
 ipchains -I output -j acctio
 ipchains -I output -j acctout
```

This creates three user-defined chains, acctin, acctout and acctio, which are to contain any accounting rules (these rules should be specified without a -j flag, so that the packets simply pass through them unscathed).

A MASQ or REDIRECT target encountered by the kernel out of place (ie. not during a forward or input rule respectively) will cause a message to the syslog and the packet to be dropped.

The old behaviour of SYN and ACK matching (which was previously ignored for non-TCP packets) has changed; the SYN option is not valid for non-TCP-specific rules.

The ACK matching option (-k) is no longer supported; the combination of ! and -y will give the equivalent).

It is now illegal to specify a TOS mask which will set or alter the least significant TOS bit; previously TOS masks were silently altered by the kernel if they tried to do this.

The -b flag is now handled by simply inserting or deleting a pair of rules, one with the source and destination specifications reversed.

There is no way to specify an interface by address: use its name.

SEE ALSO

ipfw(4)

AUTHOR

Rusty Russell <ipchains@rustcorp.com>

A2.4 – Página de manual de ipmasqadm

```
IPMASQADM(8)                                                IPMASQADM(8)


NAME
       ipmasqadm - IP Masquerading additional modules administra-
       tion

SYNOPSIS
       ipmasqadm <module> [module-specific-options]
       ipmasqadm <module> -h

       ipmasqadm autofw options
       ipmasqadm portfw options
       ipmasqadm mfw options

DESCRIPTION
       Ipmasqadm is used to  configure  extra  masquerading  fun-
       cionality,  usually provided by additional kernel modules.

       All in-firewall forwarding  takes  place  by  reverse-mas-
       querading  so  you  must  create  firewall rules that must
       match desired forwarding as-is  the  connection  had  been
       outgoing (instead of incoming).

       Kernel must have been compiled with
       CONFIG_EXPERIMENTAL=y
       CONFIG_IP_MASQUERADE=y
       CONFIG_IP_MASQUERADE_MOD=y
       and
       CONFIG_IP_MASQUERADE_IPAUTOFW=y/m
       CONFIG_IP_MASQUERADE_IPPORTFW=y/m
       CONFIG_IP_MASQUERADE_MFW=y/m
       for respective modules.

       If  you  need  to  forward one (or more) ports to internal
       hosts, consider using mfw module.

       In short:
       Short    ipmasqadm          kernel                    kernel
       descr.   module             module                    option
       ------------------------------------------------------------------
-----
       Auto                      autofw.so                 ip_masq_autofw.o
CONFIG_IP_MASQUERADE_IPAUTOFW
       Port                      portfw.so                 ip_masq_portfw.o
CONFIG_IP_MASQUERADE_IPPORTFW
       Fwmark   mfw.so       ip_masq_mfw.o        CONFIG_IP_MASQUERADE_MFW
```

```
MODULE autofw - Auto-forwarding
       This module is, under some circustances, capable  of  han-
       dling  application  protocolos  that don't have support as
       specific masq modules.  Kernel  must  have  been  compiled
       with


   autofw -h
       Command help. By now please refer to it.

       For  lot  of  useful  info about using autofw please visit
       http://ipmasq.home.ml.org


MODULE portfw - Port-forwarding
       This module is able  to  forward  to-firewall  packets  to
       internal hosts, based on address and port specification.


   portfw -h
       Command help. By now please refer to it.


MODULE mfw - fwmark-forwarding
       This  module  allows  forwarding  to-firewall  packets  to
       internal hosts, based on fwmark matching.  See ipchains(8)
       for setting up firewall rules with fwmarking.  Also please
       note that because this module acts only  in  first  packet
       connection,  it  makes  sense to add -y ipchains switch to
       TCP fwmark rules.



   COMMANDS
       mfw -A -m fwmark -r address [port] [-p pref]
              Append one rule to the end of fwmark list  of  for-
              warding hosts.
              Packets  fwmarked  will  create  a  masq-tunnel for
              redirecting further connection traffic  to  address
              port.   This  will happen at most pref times before
              scheduling another entry with same fwmark value.
              If no  port  is  specified,  redirection  will  use
              original packet destination port.


       mfw -I -m fwmark -r address [port] [-p pref]
              Same as -A option, except that the rule is inserted
              at the head.
```

```
       mfw -D -m fwmark [-r address [port] ]
              Delete specified rule(s).


       mfw -E -m fwmark [-r address [port] ] -p pref
              Edit specified rule(s), currently -p value  can  be
              changed.


       mfw -S -m fwmark
              Force scheduling in fwmark redirect entries.


       mfw -F Flush all rules.

       mfw -L [-n]
              List  rules,  optionally showing only addresses (no
              names).


   EXAMPLES
       Redirect all web traffic to  internals  hostA  and  hostB,
       where  hostB will serve 2 times hostA connections. Forward
       rules already masq internal hosts to outside (typical).

              ipchains -I input -p tcp -y -d yours.com/32 80 -m 1
              ipmasqadm mfw -I -m 1 -r hostA 80 -p 10
              ipmasqadm mfw -I -m 1 -r hostB 80 -p 20

       Redirect  ssh  traffic  from  external clientA to internal
       hostB, also show forward masq rule  to  allow  only  hostB
       incoming connections to ssh port.

              ipchains   -I  forward  -p  tcp  -d  clientA/32  -s
              hostB/32 22
              ipchains -I input -p tcp -y -s clientA/32 -d 0/0 22
              -m 2
              ipmasqadm mfw -I -m 2 -r hostB 22


       Redirect  all  traffic  from  external clientA to internal
       hostB, also show forward masq rule to allow this for hostB
       only (clean, simple ... just *grin*)

              ipchains -I forward -d clientA/32 -s hostB/32
              ipchains -I input -s clientA/32 -m 3
              ipmasqadm mfw -I -m 3 -r hostB
FILES
       /usr/lib/ipmasqadm/*.so
                           Modules   used  for  ipmasqadm  kernel
                           interfacing.
```

```
       /proc/net/ipmasq/*  Masquerading  modules  internal  state
                           files.


BUGS
       By  2.2, there is no way to share port numbers with normal
       sockets. Currently masq  modules  take  precedence  before
       sockets.

       Also  because  redirections are actually masq tunnels they
       have same  propierties:  idle  timeouts,  max.  number  of
       entries, etc.

       Kernel  module  autoloading  will  work  for  -A  and  -I
       switches, and not for -L, so you will see  warnings  about
       missing  /proc/net/ip_masq/...   if  you list entries when
       module is not (auto)loaded.  This  will  change  in  futur
       releases.


CAVEATS
       Protocols that use control and data connections are always
       a headache when crossing firewalls. Examples of these  are
       ftp,  irc,  real  audio, etc.  Because we are reverse-masq
       forwarding problems get reversed; for  example:  ftp  from
       outside  to  an internal forwarded server will not work in
       PASV mode because server will send its internal address to
       outside  client, in contrast, traditional non-passive con-
       nections will success (think about this a little, please).
       Support  for bidirectional helper modules is in the works.


NOTES
       This is my first man page, just in case you didn't  notice
       ... ;)

       Consider it pre-alpha quality.


SEE ALSO
       ipchains(8)


AUTHOR
       Juan Jose Ciarlante <jjciarla@raiz.uncu.edu.ar>
```

A2.5 – RFCs de interés

- **Capítulo 2: Direccionamiento IP**
  - *RFC 1918:* Address Allocation for Private Internets.
  - *RFC 2131:* Dinamic Host Configuration Protocol.
- **Capítulo 3: Enrutado y cortafuegos**
  - *RFC 1058:* Routing Information Protocol.
  - *RFC 2993:* Architectural Implications of NAT.
  - *RFC 3022:* Traditional IP Network Address Translator.
- **Capítulo 4: Proxy**
  - *RFC 2186:* Internet Cache Protocol (ICP), version 2.
  - *RFC 2187:* Application of Internet Cache Protocol (ICP), version 2.
- **Capítulo 5: DNS**
  - *RFC 1034:* Domain names - concepts and facilities.
  - *RFC 1035:* Domain names - implementation and specification
  - *RFC 2181:* Clarifications to the DNS Specification.
- **Capítulo 6: Futuras ampliaciones**
  - *RFC 959:* File Transfer Protocol (FTP).
  - *RFC 1123:* Requirements for Internet Hosts -- Application and Support.
  - *RFC 1661:* The Point-to-Point Protocol (PPP).
  - *RFC 2570:* Introduction to Version 3 of the Internet-standard Network Management Framework.
  - *RFC 2616:* Hypertext Transfer Protocol – HTTP/1.1.

Anexo 3: Bibliografía

En este anexo encontramos tanto referencias a libros impresos como a páginas web u otros medios electrónicos, primando estos últimos dado el carácter del estudio. La razón por lo que no se ha considerado necesario referenciar más libros, como sería habitual, es que la filosofía del software libre ha hecho posible que la documentación técnica se encuentre disponible de manera gratuita. Así, en cualquier distribución de Linux, al igual que en las páginas web de proyectos como el Linux Documentation Project *[LDP]*, encontramos todo lo necesario para, partiendo de unos conocimientos mínimos, afrontar éste y otros desarrollos similares.

Es importante también recalcar que existen puntos de la bibliografía no referenciados en el texto pero que se han considerado de relevancia para ampliar la información aquí presentada.

La nomenclatura utilizada se basa en la Guía para Referencias Bibliográficas publicada en la página web de la Universidad de Sevilla *[REFERENCIAS]* (basados a su vez en la Norma UNE 50-104-94), aunque se ha simplificado el número de campos y su contenido para adaptarlo a este caso particular.

Concretamente, para páginas web se ha utilizado el siguiente formato:

- [CÓDIGO DE REFERENCIA] Nombre de la empresa o persona responsable del contenido. Título del contenido. *URL*[28]

Para referencias a documentos escritos, el formato es:

- [CÓDIGO DE REFERENCIA] Autores. *Título del documento*. Año de edición. ISBN.

Se ha decidido no incluir más datos acerca de la publicación de los documentos al ser el código ISBN referencia unívoca de estos.

Para páginas web empresariales se ha tomado como código de referencia una palabra descriptiva del título del documento web. Para libros o artículos publicados se toma el nombre del autor seguido del año de publicación del documento (p.ej. *[STALLMAN-1994]*). En caso de haber varios libros de un autor en un año se añade un número ordinal para evitar confusiones.

[28] Universal Resource Locator: formato estándar para referenciar documentos en Internet

A3.1 – Documentación electrónica

• *Linux y software libre*

- [FRESHMEAT] Freshmeat. Inventario de software libre Freshmeat. *http://www.freshmeat.net*
- [FSF-GNU] Free Software Fundation. GNU No es Unix! - El Proyecto GNU y la Fundación para el Software Libre (FSF). *http://www.gnu.org/home.es.html*
- [LDP] Linux Documentation Project. *http://www.linuxdoc.org*
- [STALLMAN-1994] Richard Stallman. Por Qué El Software No Debe Tener Propietarios. 1994. *http://www.gnu.org/philosophy/why-free.es.html*

• *Distribuciones de Linux*

- [DEBIAN] Debian GNU/Linux. *http://www.debian.org*
- [REDHAT] RedHat Linux. *http://www.redhat.com*
- [SLACKWARE] The Slackware Linux Project. *http://www.slackware.com*
- [SUSE] SuSE Inc. *http://www.suse.com*
- [TURBOLINUX] Turbolinux. *http://www.turbolinux.com*

• *Routers hardware*

- [CABLETRON] Cabletron Systems. Enterasys Product Families. *http://www.enterasys.com/smartswitch-router/*
- [CISCO] Cisco Systems. Cisco - Hojas de datos en español: productos de acceso. *http://www.redhat.com*
- [3COM] 3Com Corporation. 3Com Routers. *http://www.3com.com/products/routers/index.html*

• *Software de Proxy*

- [GUERRERO-1999] Caching The Web: Improve your users browsing and save your bandwidth by using proxy servers to cache web pages. Enero 1999. *http://www.david-guerrero.com/papers/squid/squid.htm*
- [MS-ISA] Microsoft Internet Security and Acceleration Server. *http://www.microsoft.com/isaserver*
- [MS-PROXY] Microsoft Proxy Server: *http://www.microsoft.com/proxy*
- [SQUID] Squid: *http://www.squid-cache.org*
- [TUCOWS] Listado de los servidores proxy más comunes para Windows 95 en TuCows, un inventario de software shareware y freeware: *http://tucows.arrakis.es/proxyserver95.html*
- [WINGATE] Wingate: *http://www.wingate.com*

• *Otros programas GNU utilizados*

- [AGETTY] Wietse's collectgion of tools and papers. *ftp://ftp.porcupine.org/pub/security/index.html*

- [APACHE] The Apache Software Foundation. *http://www.apache.org*
- [BIND] Internet Software Consortium – BIND. *http://www.isc.org/products/BIND*
- [DHCPCD] Internet Software Consortium – DHCP. *http://www.isc.org/products/DHCP*
- [IPCHAINS] Linux IP Firewalling Chains. *http://netfilter.filewatcher.org/ipchains*
- [NETSNMP] The NET-SNMP Project Home Page. *http://www.net-snmp.org*
- [OPENSSL] OpenSSL: The Open Source toolkit for SSL/TLS. *http://www.openssl.org*
- [PROFTPD] The ProFTPD Project. *http://www.proftpd.org*
- [SAMBA] SAMBA Web Pages. *http://www.samba.org*
- [WUFTPD] WU-FTPD Development Group. *http://www.wu-ftpd.org*

- *Varios*

- [OPTIMA] Optima Technologies, S.L. *http://www.optimat.com*
- [REFERENCIAS] Biblioteca de la Universidad de Sevilla. Referencias Bibliográficas. *http://bib.us.es/guias/referenciabib.asp*
- [RFC] Request for Comments (RFC) Editor Homepage. *http://www.rfc-editor.org*

A3.2 – Documentación Impresa

- *Sistemas operativos y programación*

- [STEVENS-1998-1] W. Richard Stevens. *Unix Network Programming, Second Edition, Vol. 1*. 1998. ISBN: 0-13-490012-X.
- [STEVENS-1998-2] W. Richard Stevens. *Unix Network Programming, Second Edition, Vol. 2*. 1998. ISBN: 0-13-081081-9.
- [ZIEGLER-2000] Robert L. Ziegler. *Guía avanzada Firewalls Linux.* 2000. ISBN: 84-205-2949-4.

- *Redes*

- [ALBITZ-1994] Paul Albitz, Cricket Liu. *DNS and BINDin.*1994. ISBN: 1-56592-010-4.
- [ISO-1993] ISO/IEC. *Carrier sense multiple access with collision detection (CSMA/CD) access method and physical layer specifications. Std 802.3.* 1998. ISBN: 1-55937-324-5.
- [KERCHEVAL-1999] Berry Kercheval. *DHCP: A Guide to Dynamic TCP/IP Network Configuration.* 1999. ISBN: 0-13-099721-8.
- [SALAMONE-1998] Salvatore Salamone. *Guía de Gestión de Conectividad Remota.* 1999. ISBN: 0-07-882267-X.
- [STALLINGS-1997] William Stallins. *SNMP, SNMPv2 and RMON.* 1997. ISBN: 0-201-63479-1.