Capítulo 5

El proxy – cache Squid

Introducción

Funcionamiento como proxy, como caché

Instalación y configuración

Ejemplos

Capítulo 5 El proxy – caché Squid

Tanto en las pequeñas como en las grandes organizaciones (en la pequeñas se acentúa el problema) se pretende conseguir dos objetivos, en cuanto al acceso a Internet se refiere.

Uno de ellos es reducir la carga de ancho de banda que se requiere y así disminuir el coste, o con el mismo presupuesto, obtener un mejor servicio.

El otro objetivo es limitar el acceso web a solamente los usuarios autorizados, para conseguir el mismo fin.

Para alcanzar dichos fines, el administrador de red cuenta con una herramienta muy robusta y a la vez versátil, el proxy – cache Squid.

5.1 Introducción a Squid

Squid es un servidor proxy – caché que puede ser usado en una red para distribuir una conexión a internet entre todas las máquinas que componen dicha red.

Podemos disponer de un ordenador central conectado a internet mediante algún medio (ADSL, cable, RDSI…), en el que ejecutaremos Squid y que actuará además como cortafuegos.

En el momento en el que un equipo haga una petición web, Squid la tomará y la realizará en nombre de este equipo (funcionamiento como proxy). El hecho de ser intermediario confiere a Squid la capacidad de poder realizar múltiples aplicaciones.

Una de las más importantes aplicaciones consiste en guardar la información que ha solicitado una máquina, durante un tiempo aceptable, de forma que si la pide otra máquina Squid se la ofrecerá sin recurrir a Internet (funcionamiento como caché).

Mediante este procedimiento encontramos ya varias ventajas. Por un lado el usuario obtiene su petición antes, dispone de un acceso más rápido, y por otro no ocupamos ese ancho de banda, que puede ser utilizado por otro usuario.

Otra aplicación que podemos dar a Squid es la de filtro de contenidos. Al hacer de proxy, pasan por delante de él todas las peticiones que se producen. Puede registrar todos los accesos, direcciones de origen, de destino, así como permitir o denegar aquellos que no se amolden a las reglas que mantiene y que podemos configurar.

Este es el punto que nos interesa. Vamos a controlar el acceso de los usuarios a la web mediante una contraseña.

Tenemos la posibilidad (entre otras) de mantener las contraseñas en una base de datos aparte. Cosa que no tendría mucho sentido. Los usuarios autorizados pueden compartirlas con los que no lo están sin mucho cargo de conciencia.

Una situación distinta se produciría si tuvieran que compartir la clave con la que acceden al sistema. A ningún usuario le gustaría que otro leyese su correo o manipulara su trabajo. Por este motivo vamos a usar los usuarios y contraseñas que ya existen en el sistema.

Esta información usuario – clave se encuentra ya en el servidor OpenLDAP, así que el siguiente paso consistirá en hacer que Squid se autentique contra él, o lo que es lo mismo, integrar la autenticación.

5.2 Funcionamiento (como proxy)

Hemos comentado ya que uno de las principales funciones de Squid es la de proxy. Un proxy no es más que un elemento que actúa en nombre de otro. En el caso de una red local que tiene acceso a Internet, el proxy será el intermediario. Se situará entre ambas redes y se encargará de realizar peticiones y devolver resultados, de forma genérica.

Uno de los conceptos fundamentales para entender la idea de proxy es la de enmascaramiento. Por razones principalmente económicas (aunque también se puede argumentar la escasez de direcciones públicas) las organizaciones suelen disponer de una única dirección con la que acceden al exterior, a Internet.

Supongamos que tenemos una red local compuesta por tres equipos conectados entre sí mediante un switch. Con esta configuración, teniendo en cuenta que disponemos de una sola dirección ip pública, es decir, de cara a Internet, sería imposible que todas las máquinas tuviesen acceso a ella. Sólo una de ellas podría.

Es necesario entonces algún elemento que se encargue de solucionar esta situación. Vamos a tener un equipo intermediario que recibirá todas las peticiones de los equipos de la red local con sus respectivas direcciones y las cambiará, de forma que todas saldrán a Internet usando la dirección pública que posee nuestra organización.

Evidentemente se tendrá que encargar también de deshacer el cambio,es decir, cuando las peticiones resueltas lleguen al equipo intermedio debe remitírsela concretamente al equipo que las hizo.

Podemos ver el proceso de enmascaramiento de una forma más clara en la siguiente figura.

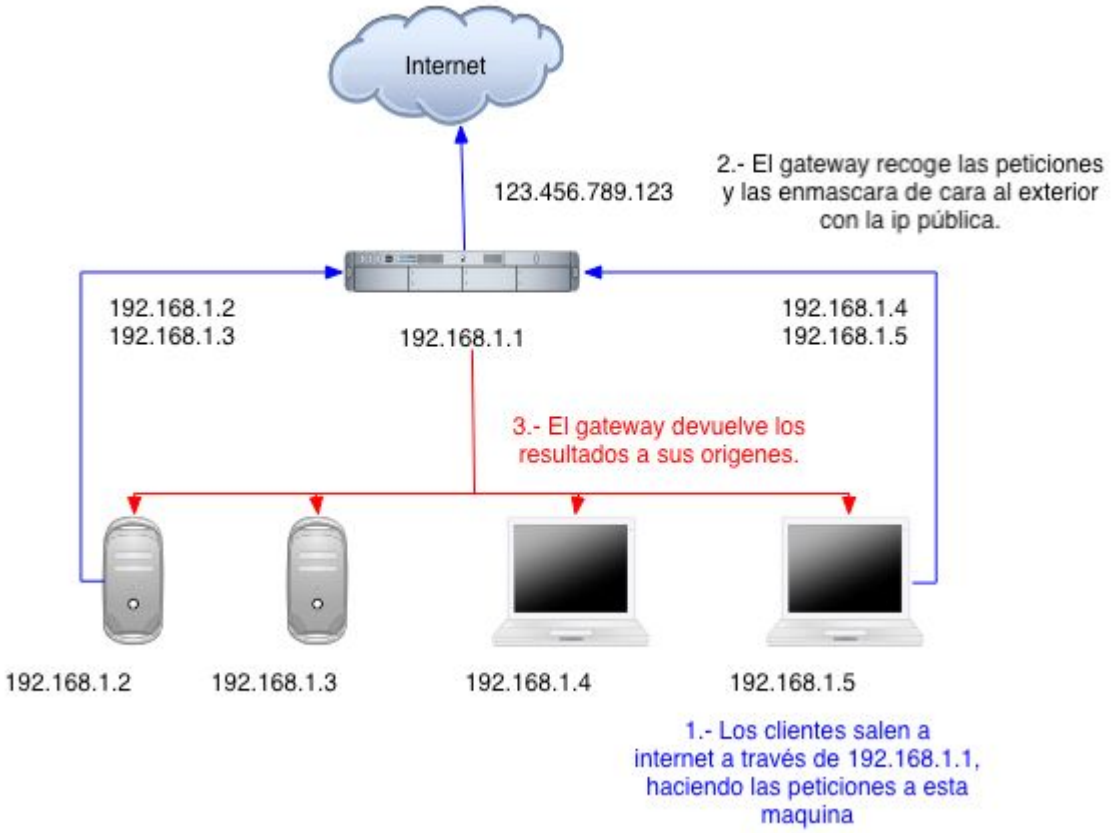

Figura 5.1 Funcionamiento como proxy

Vemos en la figura los pasos que antes hemos enumerado.

Squid hace de proxy para los siguientes protocolos: http, ftp, ssl, gopher y dns. Si necesitamos hacer uso de otros protocolos tales como smtp o pop, para el simple hecho de consultar un correo externo a la organización, no podríamos.

Este funcionamiento depende de la política de la empresa. Nosotros vamos a soportar estos protocolos y para ello debemos plantear otro esquema. Para ello utilizaremos Iptables[2] y, aunque no es el objeto de estudio de este proyecto, sí es necesario si queremos dar una solución global.

2 En realidad todo lo que hemos hablado sobre proxies lo podríamos haber hecho exclusivamente mediante Iptables, que de hecho, desarrolla su labor muy eficientemente.

5.2.1 Proxy transparente

El uso del proxy no depende sólo de la parte de servidor, sino también de la de cliente. Las máquinas de la subred deben configurar sus navegadores para que utilicen el proxy. Deben indicar la dirección donde se encuentra y el puerto por el que ofrece el servicio.

Existen situaciones que debemos tener en cuenta, como son:

- Imponer el uso del proxy. Los usuarios van a usarlo quieran o no.

- Los usuarios no deben darse cuenta que su tráfico está pasando por un proxy.

- Los usuarios van a usar el proxy pero no queremos configurar todos los clientes.

En estos casos se hace necesario el uso de un proxy transparente.

Cuando el proxy actúa de forma transparente intercepta las peticiones web y responde al usuario tal como lo haría el servidor al que está intentando conectarse.

Para que Squid actúe de este modo hace falta configurarlo pertinentemente, además de salvar algún escollo en la instalación de los paquetes adecuados. Ya hemos dicho que éste no era el objeto de nuestro estudio, así que nos limitaremos a dar unas ideas básicas e indicar cómo conseguir este funcionamiento.

5.3 Funcionamiento (como caché)

La otra función importante de Squid es la de mantener una caché. Esta caché está compuesta por los resultados de peticiones previas de forma que una nueva petición será servida desde la caché, con los beneficios que eso conlleva.

En la caché podemos almacenar diversos tipos de objetos que podemos encontrar en la red, tales como páginas web o peticiones dns. También podemos encontrarnos datos ftp y peticiones gopher, aunque no es normal.

Un ejemplo del funcionamiento como caché podemos encontrarlo en la siguiente figura. Vemos como tras realizar la petición web, Squid comprueba si ese contenido lo posee la caché y es válido. Por válido entendemos que no lleve demasiado tiempo almacenado. No tiene sentido que el tiempo de vida de un objeto en la caché sea de meses, por ejemplo. Debe estar ajustado. Tampoco debe ser demasiado pequeño ya que perderíamos la eficiencia prevista.

En la figura hemos supuesto que no encontramos el contenido que deseamos. Squid pasa entonces a solicitarlo a la web. Cuando lo recibe lo introduce en la caché y devuelve una copia al equipo que pidió dicha página.

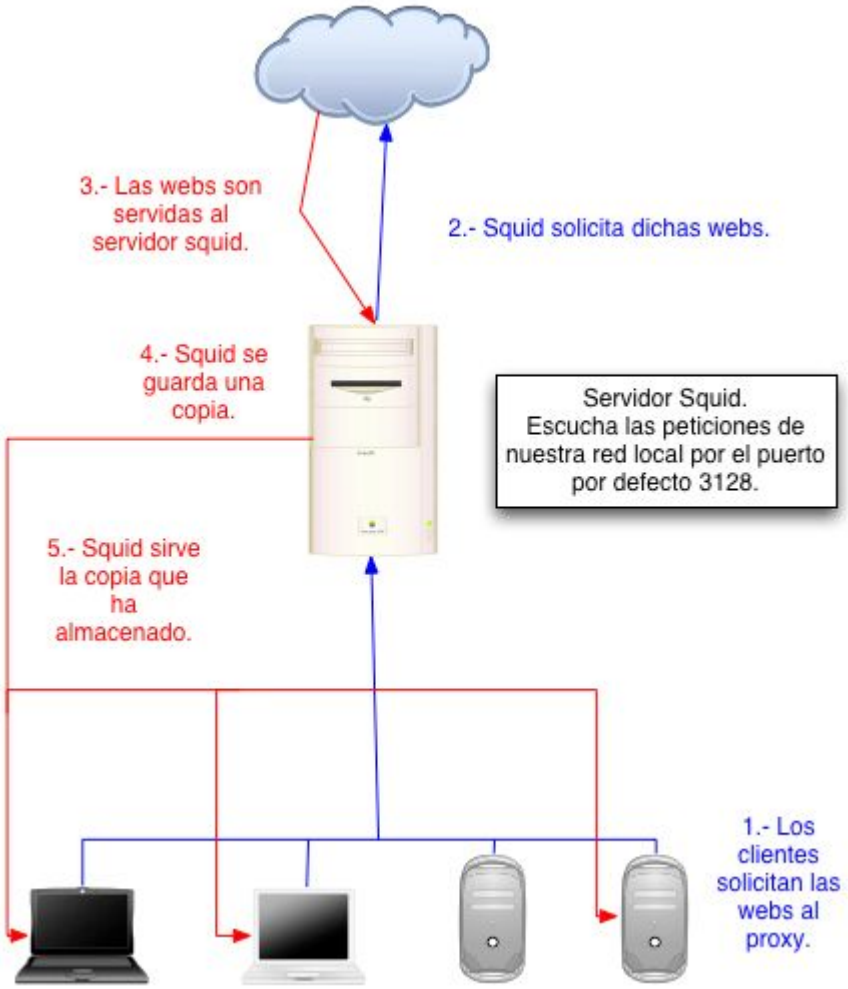

Figura 5.2 Funcionamiento como caché

Básicamente este es el funcionamiento como caché. Evidentemente debemos tener un control de los periodos de validez de los datos que contiene, aunque de estos detalles ya se encarga Squid (previa configuración).

5.4 Instalación y configuración

5.4.1 Obtener el software

El primer paso es obtener el software de la página oficial www.squid-cache.org. Existen dos versiones, una estable y otra más nueva pero que puede contener algún error. Actualmente tenemos las versiones 3.0 y 2.5, siendo esta última la estable. Trabajaremos con la versión estable.

5.4.2 Requerimientos mínimos

Hardware

En cuanto al hardware se refiere no precisamos un equipo demasiado potente. El hardware va a depender del tamaño de la caché y de la organización.

Entre los principales requisitos se encuentran una cantidad considerable de memoria RAM y un buen disco duro.

En cuanto a la memoria RAM podemos decir que cuanto más mejor. Esto no es ningún secreto, ocurre con todos los sistemas. Leer datos de RAM es mucho más rápido que tener que hacerlo desde disco. Squid mantiene en RAM una tabla con las localizaciones de los objetos en disco y su funcionamiento se ve claramente perjudicado cuando esta tabla se encuentra en memoria swap.

Quizás el factor más determinante en el rendimiento de Squid sea el disco duro. En párrafos anteriores indicábamos que necesitamos un buen disco duro para aprovechar al máximo Squid. Mediante “buen disco” hacemos referencia a un disco grande y sobre todo rápido. Por supuesto pedimos que tenga un índice de fallo pequeño, aunque este detalle lo resolveremos mediante redundancia, usando sistemas RAID.

Podemos obtener un rendimiento de la caché relativamente alto (un 25% aproximadamente) utilizando una caché pequeña (varios GBytes). Mediante este espacio estaríamos cacheando una buena parte de los sitios más habituales. Si doblamos el espacio no tiene porque doblarse también el rendimiento. Esto es debido a que ya estamos cacheando las páginas más solicitadas. Esto es, con más espacio capturamos más peticiones de las que no son utilizadas tan frecuentemente.

Una caché grande (por encima de los 20 GB) puede que no llegue al 50 % de rendimiento, a no ser que las páginas se conserven mucho tiempo en ella.

El otro aspecto importante del disco era la velocidad. Interesa que las peticiones se atiendan lo antes posible. Necesitamos un disco en el que las lecturas sean muy rápidas. Este era uno de los objetivos: acelerar el acceso al usuario. Para ello vamos a usar discos SCSI, que son los más rápidos. No se van a usar los discos IDE, aunque como ya hemos comentado, si se trata de una organización pequeña, podemos llegar a un compromiso.

En resumen, con un equipo actual tenemos cubiertas sobradamente todas las necesidades, en cuanto a Squid se refiere. Los componentes básicos de este equipo serán un disco duro SCSI de unos 80 GB y una memoria RAM de 1GB. En cuanto al procesador nos valdría cualquiera que se encuentre hoy en día en el mercado. De hecho, es raro encontrar alguno con un velocidad menor de 2 Ghz.

Con estas especificaciones cumplimos perfectamente (por exceso) los requerimientos.

Software

En cuanto a software necesitamos un compilador de C compatible con ANSI y el paquete Perl instalado en el sistema. Vamos a compilar las fuentes

así que no necesitamos una distribución en concreto. Cualquier versión servirá.

5.4.3 Pasos a seguir

Como ya dijimos anteriormente, actualmente tenemos una versión estable de Squid, la versión 2.5.STABLE7, y una en desarrollo la 3.0. Usaremos, por supuesto, la estable.

Los pasos a seguir son estos.

1) Descargar el archivo de la versión que hemos comentado. Se trata del fichero squid-2.5.STABLE7.tar.gz.
2) Desempaquetar y descomprimir. Para ello debemos ejecutar lo siguiente.
   ```
   tar –xvzf squid-*.tar.gz
   ```
   Lo cual creará un directorio squid
3) Compilar.
   Para ello debemos ejecutar el script configure. En este caso sólo tenemos que pasarle un parámetro para que se ajuste a nuestras exigencias. Debemos habilitar el uso de PAM para la autenticación.
   ```
   cd squid-*
   ./configure --enable-basic-auth-helpers="PAM"
   make all
   make install
   ```
   Por defecto se instalará en /usr/local/squid.

Tras estos pasos ya se encuentra instalado Squid, aunque todavía no puede arrancar. Necesita una configuración básica.

5.4.4 Configuración básica

A continuación presentamos la mínima configuración para que squid se ejecute. Este paso lo realizamos a modo de prueba para ver que el paquete

funciona realmente. En un paso posterior iremos depurando el fichero de configuración (/usr/local/squid/etc/squid.conf, por defecto) para obtener los resultados requeridos.

Los parámetros que tenemos que configurar son los siguientes:

- cache_dir

  Debemos establecer este parámetro a un área con suficiente espacio ya que será éste el que se dedique a la caché. Además indicamos el tamaño y el número de directorios que utilizaremos para dicho espacio.

  ```
  cache_dir ufs /usr/local/squid/cache 100 16 256
  ```

  En este caso usamos valores típicos que deben ser cambiados tras un estudio sobre la cantidad de caché necesaria.

  Lo único que no hemos comentado es el argumento ufs. Se refiere al formato que utiliza Squid para almacenar y tratar la caché.
- http_port

  Puerto por el que Squid presta su servicio. Por el momento vamos a dejarlo como viene por defecto.

  ```
  http_port 3128
  ```
- http_access

  Por defecto el acceso a la caché se deniega a todos. Somos nosotros los que debemos especificar quienes podrán acceder y para ello usaremos esta directiva. Debemos utilizar lo siguiente:

  Primero creamos las listas de acceso mediante la directiva acl. Estas son las que nos van a interesar:

  ```
  acl all src 0.0.0.0/0.0.0.0
  acl manager proto cache_object
  acl localhost src 127.0.0.1/255.255.255.255
  acl nuestrared src 192.168.123.0/255.255.255.0
  ```

  Vemos que son una especie de declaración. A continuación indicamos los accesos permitidos mediante allow y deny, dejando las opciones más restrictivas para el final.

  ```
  http_access allow manager localhost
  http_access deny manager
  http_access deny !nuestrared
  ```

```
http_access deny all
```

- cache_mgr

  Mediante esta directiva podemos indicar el correo electrónico de la persona a la que deben acudir los usuarios si la caché acaba cayendo. Por defecto es el webmaster. Vamos a personalizarlo con nuestro correo.

  ```
  cache_mgr gotche1981@yahoo.es
  ```

- visible_hostname

  No es necesaria, pero si nos va a ayudar a detectar errores en el caso de que los haya. Sería obligatorio su uso si tuviésemos distintas cachés en un cluster. Es ese caso cada una debería tener un nombre distinto.

  ```
  visible_hostname gotche
  ```

5.4.5 Configuración avanzada. Integración.

Vamos ahora a realizar algunos cambios en la configuración anterior para lograr nuestro objetivo de integrar la autenticación en Squid.

Debemos añadir las siguientes líneas, que pasaremos a comentar.

```
auth_param basic program /usr/local/squid/libexec/pam_auth
acl password proxy_auth REQUIRED
http_access allow password nuestrared
```

La primera de las líneas indica a squid que use un programa específico para la autenticación. En este caso vamos a usar el programa pam_auth. Su nombre es bastante descriptivo: consigue autenticar al usuario mediante el uso de un fichero PAM. Sólo nos quedaría enlazar con el servidor OpenLDAP, mediante el módulo pam_ldap que ya hemos utilizado, y tendríamos ya una parte importante de nuestro cometido.

Por defecto el fichero PAM que se utiliza reside en /etc/pam.d/squid, y contiene las siguientes entradas:

```
auth    required /lib/security/pam_ldap.so
account required /lib/security/pam_ldap.so
```

La siguiente línea se utiliza para definir el control de acceso (mediante acl). Indica que el password va a ser necesario.

La última línea sustituye a ésta,

```
http_access deny !nuestrared
```

que se incluye en la configuración básica. Al añadir la acl que definimos en la segunda línea estamos requiriendo el uso de la clave para el acceso de "nuestrared". A decir verdad mediante la tercera sentencia estamos discriminando entre aquellos usuarios que tengan clave válida o no.

Este funcionamiento es así debido al orden de ejecución de las reglas. Squid comprueba una regla y si se cumple ya no sigue comprobando más. Es necesario por tanto tener cuidado si queremos que el comportamiento de Squid sea el previsto. De hecho el fichero de configuración quedará aproximadamente de este modo:

```
auth_param basic program /usr/local/squid/libexec/pam_auth
acl password proxy_auth REQUIRED
#En el siguiente espacio colocamos reglas como por ejemplo:
http_access deny !Safe_ports
...
#
http_access allow password nuestrared
```

En lugar de los puntos suspensivos colocamos las reglas, de forma que si se cumplen, Squid rechace las peticiones.

Veremos con más detalles estos aspectos en el siguiente punto.

5.4.6 Configuración de filtro de contenidos

Gracias a la configuración anterior disponemos ya de un proxy-cache capaz de:

- denegar el acceso a la caché y al exterior a los equipos que no pertenezcan a la subred especificada. En este caso es la 192.168.123.0/24

- denegar el acceso a aquellos usuarios que, aunque pertenezcan a la red anterior, no disponen de una contraseña válida.

- autenticar contra el directorio OpenLDAP que disponemos, consiguiendo así la integración que estábamos buscando.

Podemos dar un paso más y denegar el acceso a páginas determinadas o que contengan alguna palabra que consideremos tabú. Podemos restringir por franja horaria, día de la semana, incluso por tipos de archivos que se puedan descargar. Existen configuraciones para toda clase de políticas.

No vamos a elegir ninguna política en concreto para nuestra configuración, ya que es ésta una cuestión muy dependiente del tipo de organización. Podemos imaginar el caso de un colegio. Parece lógico que se restrinja el acceso a páginas con contenido pornográfico o violento.

Incluso puede depender la elección de unas reglas u otras de un criterio personal (del jefe o del administrador). Tampoco queremos establecer restricciones a título personal. Por ello, vamos a presentar distintas configuraciones a modo de ejemplo, sin que nos decidamos por alguna.

5.4.6.1 Ejemplo 1

Supongamos el caso de un colegio. Tenemos diversas aulas de informática que acceden a Internet a través de un proxy. La asociación de padres y madres de dicho colegio deciden que sus hijos no deben acceder a una serie de páginas y las especifican en el fichero sitios_prohibidos.txt. Tampoco desean que accedan a páginas con un contenido específico. Para

ello crean otro fichero, contenidos_prohibidos.txt, en el que listan las palabras claves que creen que suelen tener estos sitios.

Pues bien, haciendo uso de las ACLs adecuadas creamos el siguiente esqueleto para el fichero de configuración:

```
   # Ejemplo 1. Filtrado de contenidos.
   # A continuación las acl
   acl sitios_malos dstdomain "/usr/local/squid/etc/sitios_prohibidos.txt"
   acl palabras_malas url_regex
"/usr/local/squid/etc/contenidos_prohibidos.txt"
   ...
   #En el siguiente espacio colocamos las reglas:
   http_access deny sitios_malos
   http_access deny palabras_malas
   ...
   http_access allow nuestrared
```

Los ficheros de texto que antes hemos mencionado no son más que eso, ficheros que en cada línea aparece una página web o palabra prohibida. Los hemos guardado en el directorio /usr/local/squid/etc, que es además donde reside el fichero de configuración squid.conf.

El uso de estos ficheros es algo común. Es posible encontrar diversos ejemplos de ellos gracias a administradores que los ofrecen. En la bibliografía presentaremos algunas direcciones donde podemos encontrar estos ficheros y comentarios sobre su alcance y uso.

5.4.6.2 Ejemplo 2

Supongamos el caso de una biblioteca. En ella, se presenta la necesidad de bloquear el chat y la descargas masiva de archivos.

Pues bien, la solución pasa por configurar las reglas, tal como hicimos antes. Las presentamos a continuación.

```
   # Ejemplo 2. Filtrado de puertos y tipos de archivos.
   # A continuación las acl
   # Filtramos puertos 6667 (irc) y 1900 (Microsoft Messenger)
   acl puertos_malos port  6667
   acl puertos_malos port  1900
   ...
```

```
# Filtramos las descargas de archivos de audio y video
acl tipos_ficheros_malos url_regex -i \mp3$ \.avi$ \.mpg$ \.wma$
...
#En el siguiente espacio colocamos las reglas:
http_access deny puertos_malos
http_access deny ficheros_malas
...
http_access allow nuestrared
```

5.4.6.3 Ejemplo 3

Supongamos el caso de un centro de cálculo. En él se necesita restringir el acceso a internet a sólo unas horas determinadas. Se pretende que fuera de esas horas los alumnos encuentren en el centro más plazas libres para poder realizar sus prácticas.

De la misma forma que hicimos en los ejemplos anteriores definimos ACLs y configuramos reglas. En este caso necesitamos hacer uso de la directiva time.

```
# Ejemplo 3. Filtrado por horas
# A continuación las acl
# Vamos a permitir sólo conexiones en la franja de 12-16 horas.
# MTWHF se refiere a los días laborables de la semana(Monday,Tuesday...)
acl horas_permitidas MTWHF time  12:00-16:00
...
http_access deny !horas_permitidas
...
http_access allow nuestrared
```

5.4.6.4 Aclaración respecto a la nomenclatura

Puede parecer un tanto jocoso, o incluso infantil, haber nombrado a las reglas que hemos definido en los ejemplos anteriores mediante un nombre indicativo y el sufijo “_malos” (o “_buenos”), a modo de coletilla. Pero no es así, tiene una causa.

Ya hemos dicho lo importante que es la posición de las reglas en el fichero de configuración. En cuanto se encuentre la primera regla que se cumpla, ya no comprueba la siguiente. Por ello, debemos ser muy cuidadosos.

En los ejemplos anteriores no hay ningún problema. Tenemos pocas reglas y no hemos añadido ninguna excepción. Veamos que no es lo mismo ver esto:

```
http_access deny sitios
http_access deny palabras
http_access deny !puertos
...
http_access allow nuestrared
```

que esto:

```
http_access deny sitios_malos
http_access deny palabras_malos
http_access deny !puertos_buenos
...
http_access allow nuestrared
```

Claramente podemos distinguir de un vistazo la política que estamos siguiendo. Denegamos acceso a las páginas referenciadas por “sitios_malos” y que contengan alguna palabra contenida en “palabras_malos”. Deben usar un puerto que esté recogido en “puertos_buenos” y por supuesto pertenecer a “nuestrared”.

Fijémonos en la acl “puertos_buenos”. En ella hemos cambiado la forma de proceder. Ahora es más restrictiva. No estamos rechazando una serie de objetos, sino que estamos permitiendo sólo un conjunto específico y limitado de ellos.

Imaginemos que no estamos usando las coletillas (buenos/malos) y se nos olvida el signo de admiración en la directiva puertos. Y que por tanto Squid no funciona de forma prevista. Tendríamos que repasar entonces cada una de las reglas y también cada una de las ACLs. En cambio, si las usamos, viendo sólo las reglas podemos deducir claramente donde se encuentra el fallo.

Esta explicación viene a insistir en el hecho de que la claridad nunca es redundante. Toda ayuda es poca para la persona que se va a encargar del mantenimiento del proxy.

5.4.7 Dansguardian

Aunque ya hemos dejado patente el posible uso de Squid como filtro de contenidos, no es usual encontrarlo así.

La situación más común suele ser la combinación de Squid y Dansguardian. Squid funcionaría como proxy-cache, que es su función principal, como ya hemos visto, y Dansguardian que actuaría como filtro de contenido.

El porqué de esta solución es fácil de adivinar. Fiable, robusto, fácil de instalar y configurar y sobre todo que funciona a la perfección son sus características principales. De hecho ha ganado diversos premios que alaban su buen hacer. Si además añadimos que dispone de una licencia libre y que es usado en multitud de organizaciones (colegios, instituciones...), con la consecuente cantidad de información de la que podemos disponer, Dansguardian se nos antoja como la elección adecuada.

5.4.8 Otras prestaciones

5.4.8.1 Limitación del ancho de banda

El control del ancho de banda se lleva a cabo mediante las denominadas “delay_pools”. Su trabajo consiste en limitar la velocidad con la que Squid devuelve las pérdidas caché (cache miss). Los éxitos de la caché (caché hits) se devuelven lo antes posible. Asumimos que contamos con un elevado ancho de banda local.

Las delay pools pueden verse de forma sencilla como “cubos de ancho de banda”. Las respuestas se retrasan hasta que cierta cantidad de ancho de banda está disponible del cubo apropiado. Los cubos no almacenan un nivel de ancho de banda (por ejemplo 100Kbit/s) sino que almacenan cantidad de tráfico (por ejemplo 300 KBytes). Squid añade cierta cantidad de tráfico a los cubos cada segundo y finalmente los clientes de la caché sacan tráfico del cubo cuando reciben datos. Este es básicamente el mecanismo de control del que hablamos.

El tamaño del cubo determina cuanto ancho de banda está disponible para el cliente a modo de ráfaga. Partiendo de un cubo lleno, el cliente puede obtener tanto tráfico como necesite hasta que se vacíe. A partir de ese momento el cliente recibiría todo el tráfico a la tasa de llenado del cubo.

El mapeo que se realiza entre los clientes de Squid y los cubos reales puede resultar un poco complicado. Squid usa tres herramientas para lograrlo: reglas de acceso, clases de delay pools y tipos de cubos. La primera comprobación que realiza Squid es ver si el cliente se encuentra en la lista “delay_access”. Si es así, la petición de dicho cliente se asigna a una delay pool en concreto. Cada delay pool puede pertenecer a una clase: 1, 2 y 3. la clase determina que tipos de cubos se usan. Squid dispone de tres tipos de cubos: agregado, individual y de red.

La clase 1 tiene un único cubo agregado. La clase 2 tiene uno agregado y 256 cubos individuales. La clase 3 dispone de uno agregado, 256 cubos de red y 65.536 cubos individuales.

Podemos intuir que tanto los cubos individuales como los de red corresponden a octetos de direcciones IP. En una delay pool de clase 2, el cubo individual queda determinado por el último octeto de la dirección del cliente. En uno de clase 3, el cubo de tipo de red se determina mediante el

tercer octeto de la dirección, y el cubo individual por los octetos tercero y cuarto.

Cuando llega una petición a través de una delay pool que dispone de más de un tipo de cubo, toma el ancho de banda de todos los cubos. Por ejemplo, centrándonos en una delay pool de clase 3. Si el cubo individual dispone de 20 KB, el de red de 30 KB, pero el cubo agregado sólo tiene 2 KB, el cliente sólo recibe 2 KB. Incluso si algunos cubos están llenos. El cliente está limitado por el cubo más pequeño.

A continuación vamos a presentar unos ejemplos que ilustren el uso de estos conceptos.

Ejemplo 1

```
acl sitio_regulado url_regex –i www.marca.es

delay_pools 1
delay_class 1 1
delay_parameters 1 16384/16384
delay_access 1 allow sitio_regulado
```

En él estamos regulando el acceso a la página deportiva www.marca.es. Esto puede hacerse por mediante nombre, contenido, ip..., tal como hemos visto en los puntos anteriores. Aquí hemos decidido hacerlo así por comodidad.

Definimos un objeto delay_pool referenciado por un 1 y que será el único. Indicamos también que será de clase 1:

```
delay_class 1 1
```

Indicamos la máxima velocidad a la que nos vamos a poder conectar al sitio especificado (en bytes por segundo), en este caso 16384 Bps (16KBps). El otro número se refiere a la cantidad de datos (en bytes) que deberían ser regulados tras alcanzar la conexión.

Finalmente permitimos el acceso mediante delay_access.

Esta es la forma más básica de limitar en ancho de banda. No distingue entre usuarios. Todos “sufrirían” esta limitación.

Ejemplo 2

En este caso vamos a mostrar un ejemplo en el que limitamos sólo a un pequeño número de usuarios. Podríamos usar la primera clase que hemos tratado en el ejemplo 1, pero al crecer el número de usuarios se complica demasiado el asunto; habría que crear un delay_pool para cada nuevo usuario.

Supongamos que disponemos de una línea de 1024 Kbps y 20 usuarios y decidimos que cada uno de ellos no pueda usar más de 200 Kbps. La codificación de esta regla se realiza fácilmente mediante un delay_pool de clase 2. Veámoslo.

```
acl all src 0.0.0.0/0.0.0.0
delay_pools 1
delay_class 1 2
delay_parameters 1 128000/128000 25600/25600
delay_access 1 allow all
```

Recordemos que las velocidades se encuentran en bytes por segundo.

Ejemplo 3

La tercera clase puede utilizarse para regular rangos enteros de redes de clase C. Supongamos que disponemos de una línea de 2048 Kbps y de tres subredes con 100 máquinas cada una. Además vamos a limitar a 20 Kbps la velocidad máxima de cada equipo.

Este ejemplo se implementaría mediante las siguientes líneas.

```
acl all src 0.0.0.0/0.0.0.0
delay_pools 1
delay_class 1 3
```

```
delay_parameters 1 256000/256000 85333/85333 2560/2560
delay_access 1 allow all
```

5.4.8.2 Jerarquía de cachés

Una jerarquía de cachés es una colección de servidores proxy-caché organizados de forma que tienen relaciones padre e hijo. Esto es, aquellas cachés que se encuentran más cerca de las pasarelas a internet actúan como padres de las cachés que se encuentran más alejadas. Las cachés padres resuelven los “fallos” (“cache miss”) de los hijos. En otras palabras, cuando una caché pide un objeto a su caché padre, y el padre no tiene ese objeto en su caché, el padre busca ese objeto, lo mete en caché y se lo entrega al hijo.

Este esquema asegura la máxima reducción de ancho de banda en los enlaces a la red, ayuda a reducir la carga en los servidores fuera de ella, y construye una buena caché en los padres, de tal forma que las cachés hijos pueden obtener un mejor rendimiento (un mayor número de “caché hits”).

Además de las relaciones padre-hijo, Squid soporta el concepto de hermano. Se trata de cachés del mismo nivel jerárquico que se encargan de distribuir al carga de los servidores. Cada caché de la jerarquía decide independientemente si buscar un objeto en su ubicación real o por el contrario solicitarlo a un caché padre o hermano, usando para ello un protocolo de resolución muy simple. En ningún caso un hermano responderá a fallos de resolución de otra caché hermana.

5.4.8.3 Gestión de la caché

Una forma interesante de optimizar Squid es hacer un buen uso de la caché. Para ello disponemos de una herramienta que nos permite visualizar prácticamente todos los parámetros en los que Squid se encuentra involucrado. Mediante un análisis de estos parámetros podemos ajustar la

configuración para sacarle el máxima partido a la caché y por tanto lograr mejorar nuestros objetivos de rendimiento.

La herramienta en cuestión es un script cgi, llamado cachemgr.cgi, que se crea en el directorio /usr/local/squid/libexec cuando se realiza una instalación estándar. Para usarla debemos tener operativo un servidor web y copiar en la carpeta cgi-bin dicho script.

La ejecución de la aplicación se realiza mediante un navegador accediendo a la dirección http://localhost/cgi-bin/cachemgr.bin .

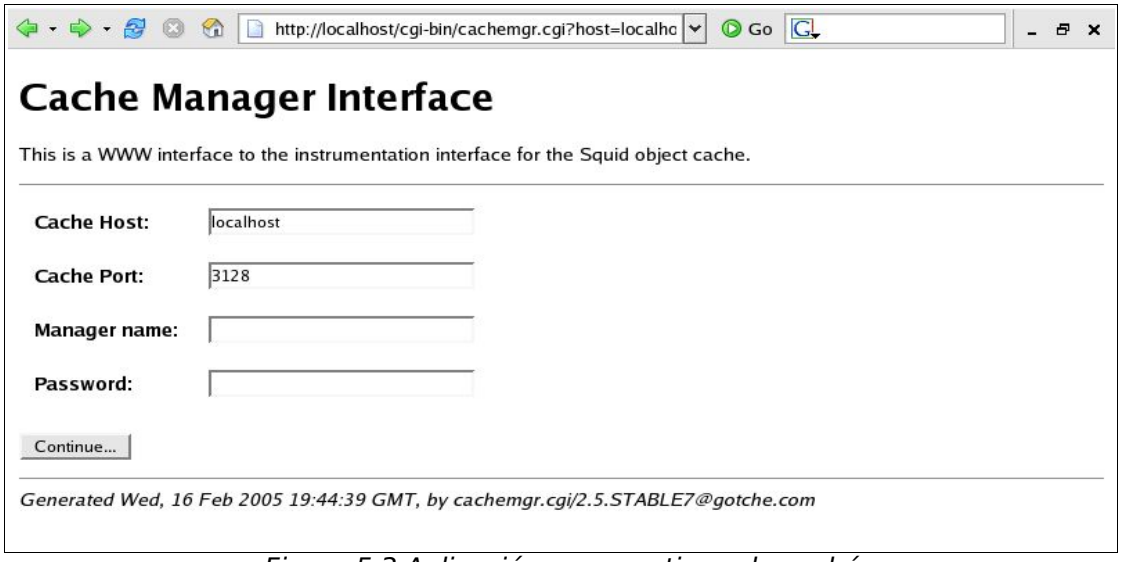

Figura 5.3 Aplicación para gestionar la caché

Por defecto no hemos usado contraseña, aunque deberíamos hacerlo. Para ello debemos añadir al fichero squid.conf la siguiente línea:

```
cachemgr_passwd 'password'
```

y sustituir 'password' por una clave apropiada. Debemos tener en cuenta que en este caso la contraseña se encuentra en texto plano, por lo que debemos proteger el archivo donde se halla de manos inapropiadas. Basta con permitir el acceso al archivo únicamente su dueño, que en este caso es el root.

La aplicación en sí tiene numerosas opciones y parámetros. Disponemos de estadísticas de prácticamente cualquier dato, muy útiles para conseguir un mayor rendimiento.

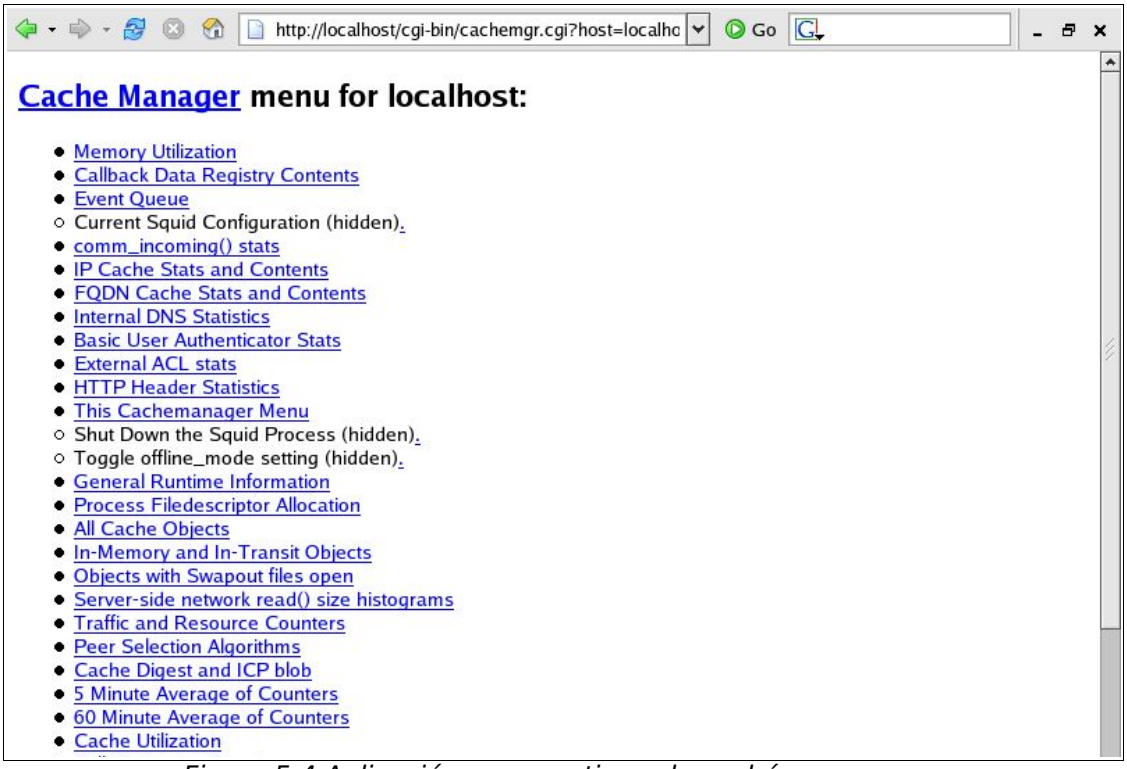

Figura 5.4 Aplicación para gestionar la caché