

Servidor SOCKS-PROXY

Ricardo Soria

ricardo_soria AT yahoo.com

El objeto de este documento es proporcionar una guía práctica a los usuarios de Linux que quieran implementar un servidor proxy con SOCKS 4-5, a fin de ahorrarles tiempo y dolores de cabeza.

1. Introducción

Introducción

1.1. ¿Qué es un servidor proxy?

Imaginemos una situación que tarde o temprano se nos ha presentado a la mayoría de nosotros, Técnicos en Informática o Administradores de Red: La necesidad de compartir una conexión a Internet entre varios clientes o equipos que se encuentran conectados en un LAN. Eso de compartir recursos e intercambiar información entre los equipos integrantes de nuestra LAN, es una cosa, pero compartir la conexión a Internet, requiere ya otros tipos de configuración y en muchos casos, software adicional.

Particularmente hablando de Linux, disponemos de la conocida herramienta IPTABLES, equivalente a IPCHAINS e IPFWADM en versiones anteriores, la misma que, básicamente, nos permite ir configurando un firewall, pero por sobre este, una capa de NAT (Network Address Translation), lo cual nos permitiría compartir una conexión a Internet mediante reemplazo de direcciones IP fuente/destino.

Pero, ¿qué ocurre cuando, por ejemplo, el número de clientes entre los que deseamos compartir la conexión es demasiado alto? o supongamos otro caso típico (para mi): ¿Qué tal si en todos los equipos de nuestra LAN tenemos un antivirus que necesita ser actualizado periódicamente, y no nos permite descargar "una sola vez" la actualización, para luego instalarla en todos nuestros equipos? Habría que ir actualizando manualmente cada uno de ellos, lo que supondría una descarga directa desde Internet de varios MB por cada uno de nuestros equipos.

Todo este tipo de circunstancias nos obligaría a mantener constantemente una excesiva tasa de transferencia de información entre nuestra LAN e Internet, lo cual ocasiona los consiguientes

problemas de saturación, lentitud, etc. etc. etc., sin mencionar las miradas amenazadoras de los usuarios insatisfechos ;-)

En este punto es donde entra en juego la importancia de usar un servidor proxy con caché. Básicamente, un proxy es un software que abre un socket en determinado puerto en nuestro host, mediante el cual se escuchan peticiones de Internet del resto de máquinas de nuestra red local. Por otro lado, el proxy reserva un espacio de Disco de longitud variable, al que se denominará *caché*. El software toma una de estas peticiones, y lo siguiente que hace es buscar en el caché, para ver si ya existe una *copia* de la página u objeto que está solicitando el cliente. Si ya existe, ¡¡Bingo!! solo toma esa copia existente en nuestro disco duro, y se la envía al cliente solicitante. En cambio, si no existe dicha copia, el proxy tiene que bajar aquel contenido de Internet, para poder enviárselo al cliente. En este caso, además de hacer dicho envío, se encarga de añadir este nuevo objeto, página o archivo, al caché, para su posterior uso.

Considerando esto, es fácil darse cuenta de la gran utilidad que representa un caché: Al haber muchas peticiones cuyas respuestas viajan únicamente entre nuestro servidor y el respetivo cliente en nuestra red de área local, ahorraremos gran cantidad de tiempo y ancho de banda, proporcionando una navegación más rápida, segura y funcional a los usuarios finales. Lo único que tendremos que sacrificar será cierto porcentaje de recursos de memoria y de disco en nuestro servidor. Y Listo !!!

1.2. ¿Qué es un Socks Firewall?

La funcionalidad total de un servidor proxy no radica únicamente en el caché y la consiguiente mejora en la calidad de la navegación. Hay un factor de suma importancia que también debemos tomar en cuenta a la hora de decidir sobre la instalación de un servidor proxy: la **SEGURIDAD**. Para esto existen implementaciones como el protocolo SOCKS, que en su sentido más general, puede ser utilizado de la misma forma y para los mismos fines que un servidor proxy como el que describimos en párrafos anteriores (al que también se refiere comúnmente como *proxy http*, aunque se usen también para otros protocolos). Pero además , este protocolo Socks proporciona también la funcionalidad de un firewall , es decir, brinda una mayor seguridad para los usuarios internos de nuestra red, ocultando las verdaderas direcciones IP que formulan las solicitudes, y filtrando los paquetes que salen, pero sobre todo los que entran, para ser redirigidos hacia la máquina local que originalmente hizo la petición.

Actualmente, las versiones de Socks casi exclusivamente utilizadas son la 4 y la 5. Socks5 proporciona mayor seguridad aún, ya que añade un mecanismo de autenticación, donde los clientes deben además proporcionar nombre de usuario y contraseña para acceder al servicio.

Por cierto, sólo como un dato adicional, Socks es abreviatura de Sockets...

2. Descarga e Instalación

Descarga e Instalación:

2.1. Principales paquetes

Una vez que tomamos la decisión de instalar un servidor proxy, el siguiente paso será decidir qué software vamos a utilizar, descargarlo y ejecutar la respectiva instalación.

Lógicamente, a partir de este momento hablaremos exclusivamente dentro de entornos Linux. Hace pocos días, yo estuve justamente en esta situación: Buscar e instalar un servidor proxy en mi computador. Debo anotar que estoy trabajando con RedHat Linux 8.0, con la versión del kernel que vino con esta distribución, es decir, la 2.4.18-14.

Encontrar un servidor proxy con caché, no fue difícil. Entre los primeros resultados que produjo una búsqueda en google, apareció la dirección: <http://www.squid-cache.org> (<http://www.squid-cache.org/>)

Aquí podemos encontrar un software gratuito, re-distribuíble bajo los terminos de la licencia pública general GNU/GPL. La última versión estable que se encuentra disponible al momento de escribir esto es la 2.5-STABLE1, la misma que podemos descargar en formatos .tar.gz o .tar.bz2. Yo descargué el primero de ellos. En el siguiente punto hablaremos sobre la instalación.

Pero, Squid tenía un inconveniente grave, al menos para mí: No tiene soporte para el servidor Socks5. Yo necesitaba implmentarlo inevitablemente, ya que hay ciertos programas y ciertas páginas de Internet que utilizan applets que no funcionan correctamente si no es mediante el protocolo Socks4 o 5. Así que, lo siguiente que tuve que hacer fue comenzar a buscar un software que implementase este servicio adicional en mi servidor.

Esta búsqueda ya no fue tan fácil como la anterior: Parece que los servidores socks para Linux son muy escasos, y además, de los pocos que se encuentran, la documentación es casi nula. Los programas que encontré, con ayuda de Ricardo Cervera, si no se me escapa alguno, fueron:

- socks-d-0.1b (http://www.linux.org/apps/AppId_6666.html)
- dante-1.1.13 (<http://www.inet.no/dante>)
- nylon-1.1 (<http://monkey.org/~marius/nylon/>)

Descargué cada uno de ellos, pero al momento de instalar, tuve distintos tipos de problemas, tanto en la compilación, como en la instalación en sí. Al final, no pude utilizar ninguno de ellos. Desconozco las causas exactas por las que se me presentaron tantos problemas con estos programas. Lo único que sé es que los makefiles podrían no ser los correctos. Ya estaba por darme por vencido, cuando en algún lugar recóndito de Internet -ni si quiera he logrado volver a dar con él- encontré una referencia hacia otro servidor socks: DELEGATE. La url donde se puede encontrar es: <http://www.delegate.org/>

Delegate no es exclusivamente un servidor socks; también se puede utilizar como servidor proxy http común, aunque la verdad, no lo he probado. Fue el único programa que a la final puede utilizar para implementar socks5 en mi servidor.

En resumen, los programas con los que definitivamente trabajo hasta ahora son Squid-2.5STABLE1 (<http://www.squid-cache.org>) y Delegate-8.3.3 (<http://www.delegate.org>). Ambos los descargué en formato .tar.gz. En el caso de Squid, el desempaquetado quedaría así:

```
# tar -xvzf squid-2.5.STABLE1.tar.gz
```

Con lo que obtendremos el directorio `squid-2.5.STABLE1/`. Ingresamos a él procedemos a compilar e instalar el programa:

```
# cd squid-2.5.STABLE1
# ./configure
# make all
# make install
```

Si no hay ningún problema, el programa se instala y queda en la ubicación `/usr/local/squid/`. Esta ubicación se puede cambiar mediante el uso del parámetro `--prefix` en `configure`, y de igual modo, existen muchos otros parámetros que nos permiten especificar algunos detalles de compilación-instalación. Para una referencia completa, estando en el directorio de fuentes de Squid, podríamos digitar:

```
# ./configure --help
```

Ahora, hablemos sobre `delegate`...

Con lo que obtendremos el directorio `delegate8.3.3/`. Ingresamos a él, y lo único que tenemos que hacer es un `make`:

```
# cd delegate8.3.3
# make
```

En este punto, se nos preguntará nuestra dirección de e-mail. La ingresamos, y LISTO!!! el programa está completamente instalado... Mucho más fácil, no??? ;-)

3. Puesta en marcha de los servicios

Puesta en marcha de los servicios:

3.1. Preparando el archivo de configuración de Squid

Una vez instalado Squid, el siguiente paso es hacer algunas modificaciones mínimas necesarias en el archivo de configuración, el mismo que se encuentra en la carpeta `etc/` dentro del directorio en el que hayamos instalado Squid. Por ejemplo, `/usr/local/squid/etc/`. El archivo se llama `squid.conf`, y los parámetros que obligatoriamente hay que establecer son:

- `http_port`: Indica el puerto en el que se van a escuchar peticiones http por parte de los clientes. Comúnmente se utiliza el puerto 8080.
- `cache_mem`: Indica la cantidad de memoria que se utilizará para agilizar el caché. Es muy importante tomar en cuenta que squid usará considerablemente más memoria de la que nosotros especifiquemos aquí, por lo que no hay que establecer un valor muy alto. Yo utilizo 8 MB (por defecto), y trabaja sin ningún problema.
- `cache_dir`: Directorio y espacio máximo de disco para cache de squid. Por defecto tomamos los valores:

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

El primer valor (100) indica el espacio de disco en MB máximo para usarse como caché.

- `host_file`: Permite indicar el archivo de hosts de nuestro sistema, típicamente, `/etc/hosts`.
- `acl` y `http_access`: la sección `acl` nos permite definir listas de acceso, es decir, de qué direcciones IP se recibirán peticiones http, o cuáles puertos se bloquearán. Hay valores por defecto, pero nosotros debemos definir nuestra propia red. Por ejemplo, yo tengo añadida la siguiente línea en segundo lugar entre los valores por defecto:

```
acl lan src 192.168.0.0/255.255.255.0
```

Con lo cual estoy añadiendo una lista de acceso que inicia en la IP 192.168.0.0 y con netmask 255.255.255.0 (o sea, finalizaría en la IP 192.168.0.255). En el siguiente apartado, veremos otros tipos de ACLs que proporcionan mayor y más preciso control sobre quienes y a qué direcciones podrán acceder a través de nuestro proxy Squid.

Retomando el ejemplo anterior, debo habilitar esta lista, indicando que de todas estas IPs se recibirán peticiones. Esto se hace en la sección `http_access`. Yo lo tengo de la siguiente manera:

```
# INSERT YOUR OWN RULE(S) HERE TO ALLOW ACCESS FROM YOUR CLIENTS
```

```
http_access allow lan
```

Nótese que lo he insertado en la parte recomendada en el propio archivo de configuración, y además, que se utiliza el mismo nombre con el que se haya creado la `acl`, en este caso, `lan`.

- `visible_hostname`: Podemos establecerlo al nombre de host que normalmente utilizemos en nuestro propio equipo, por ejemplo, `servidor`.
- `refresh_pattern`: Nos permite especificar qué tipos de objetos se van a almacenar en el caché, y además, con qué frecuencia se van a refrescar, es decir, se van a buscar nuevas versiones de los objetos. Resulta muy útil, desde el punto de vista de que por defecto, squid únicamente almacena objetos ftp, y gopher, pero nosotros podemos indicar que además se

almacenen objetos de otros tipos, como código html, imágenes jpg, gif, archivos swf de flash, etc.

Yo trabajo con la siguiente configuración:

```
refresh_pattern \.mid 10080 90% 10080
refresh_pattern \.wav 10080 90% 10080
refresh_pattern \.gif 2880 90% 10080
refresh_pattern \.jpg 2880 90% 10080
refresh_pattern \.swf 2880 90% 10080
refresh_pattern ^http: 2880 40% 4320
refresh_pattern ^ftp: 240 50% 1440
refresh_pattern ^gopher: 240 40% 1440
refresh_pattern . 240 20% 4320
```

Lo cual proporciona un rendimiento bastante rápido y aceptable. Note que el primer argumento es una expresión regular, así que si sabes manejarlas, podrás hacer referencia a URLs de objetos bastante específicas. Los valores que se encuentran encolumnados al final de cada línea, se encuentran por demás explicados en el mismo archivo de configuración, y en ciertas páginas de Internet referentes a Squid.

Cabe indicar que existen muchas opciones más que podemos configurar, pero las indicadas son las mínimas para un correcto funcionamiento del programa. El propio archivo de configuración contiene explicaciones y ejemplos en todas sus secciones.

3.2. ACLs de tiempo y de expresiones regulares

Si analizamos cuidadosamente los comentarios incluidos en el archivo de configuración de Squid (`squid.conf`), veremos que se reconocen diferentes tipos de ACLs, los mismos que nos permiten definir listas basándonos en ciertos parámetros que pueden resultar de gran utilidad en determinado momento. A continuación, hablaremos brevemente sobre los más importantes:

ACL de tiempo (`time`). Este tipo de lista nos permite definir cuándo se podrá acceder a Internet a través de nuestro proxy. Podemos definir los días de acceso, y además, las horas en las que es posible acceder. El formato para definir una ACL de tiempo es:

```
acl <nombre> time [abreviación-día] [h1:m1-h2:m2]
```

El primer parámetro (abreviación-día) nos permite indicar los días en los que se permitirá acceso a los clientes, de acuerdo con el siguiente esquema:

```
S -> Domingo (Sunday)
M -> Lunes (Monday)
T -> Martes (Tuesday)
W -> Miércoles (Wednesday)
H -> Jueves (Thursday)
F -> Viernes (Friday)
```

A -> Sábado (Saturday)

Además, como es lógico, existe la restricción de que h1:m1 (hora/minuto inicial) debe ser menor que h2:md2 (hora/minuto final).

Aquí tenemos un ejemplo simple:

```
acl dias time MTWHF 08:00-18:00
```

Esto nos permitiría crear una lista para posteriormente permitir o denegar el acceso a través de Squid, de Lunes a Viernes entre las 8 de la mañana y las 6 de la tarde.

Por otro lado, tenemos las ACLs relacionadas con *expresiones regulares*. Esto nos permite definir listas basadas en ciertas palabras, frases, o secuencias de caracteres presentes en alguna parte de la solicitud http formulada por el cliente.

Los tipos de ACLs referentes a *expresiones regulares* son:

- srcdom_regex
- dstdom_regex
- url_regex
- urpath_regex
- ident_regex
- proxy_auth_regex

Dado que este es un MINI-Howto, que pretende dar una guía rápida a los usuarios nuevos e inexpertos de Squid, no profundizaré en ninguno de ellos. Sin embargo, hablaré de uno que a mi juicio es muy útil, y que en lo personal, me ha servido mucho: `url_regex`. Este tipo de ACL nos permite crear una lista que intercepta peticiones de los clientes en cuya URL se hallen ciertas palabras o frases. Resulta especialmente útil cuando por ejemplo debemos bloquear el acceso a sitios pornográficos (especialmente necesario cuando lo que estamos manejando es un local público).

El formato básico para un `url_regex` ACL es:

```
acl <nombre> url_regex [-i] patrón
```

Donde `patrón` indica la palabra o palabras que deberán constar en la URL solicitada por el cliente. Podemos incluir la opción `-i`, que indica que no se diferenciará mayúsculas y minúsculas.

Por ejemplo, tenemos:

```
acl prohibidos url_regex -i gai sex lesb porn xxx nude
```

En este caso, es especialmente importante tener mucho cuidado de no bloquear accidentalmente ciertas palabras o frases de uso común. Por ejemplo, la expresión `sex` incluiría toda petición en

cuya URL se hallen secuencias como *sex*, *sexo* y *sexualidad*, lo cual podría provocar un efecto inverso al deseado en ciertos casos.

Entonces, la solución (que yo utilizo) es uncluir otra ACL del mismo tipo (`url_regex`), donde consten ciertos términos que sí vamos a aceptar, a pesar de incluir las expresiones que normalmente estamos bloqueando. Una vez creadas ambas listas, primero especificamos que vamos a aceptar los términos correspondientes a una lista, y luego, denegamos aquellos correspondientes a la otra.

Por ejemplo, podríamos tener algo así:

```
acl permitidos url_regex -i sexualidad sextante asexual
acl prohibidos url_regex -i sex
```

Y luego, en la parte correspondiente a `http_access`, tendríamos que incluir también lo siguiente:

```
http_access allow permitidos
http_access deny prohibidos
```

3.3. Creando el *Swap Directory* de Squid

El siguiente paso es construir el *swap directory*, que no es más que un árbol de directorios donde se guardarán los objetos y páginas de caché. Pero para esto, hay un detalle muy importante a tomar en cuenta: debemos cambiar el propietario del directorio `var/` dentro del directorio `squid/`. En nuestro ejemplo, sería el directorio `/usr/local/squid/var/`. Si estamos siguiendo todas las recomendaciones de la documentación de squid, habremos creado previamente en nuestro sistema el usuario-grupo `squid-squid`. Si así lo hicimos, debemos asignarle al directorio `var/` el propietario squid. Sería algo así:

```
chown -R squid.squid /usr/local/squid/var
```

En caso de que en nuestro sistema no exista el usuario y grupo squid, podemos utilizar otros, como por ejemplo, `nobody-grupo`, donde grupo puede ser cualquier grupo creado por nosotros mismos. El usuario `nobody` supuestamente ya existe en el sistema (al menos en mi caso, fue así).

Procedemos a crear el swap directory, de la siguiente manera:

```
# /usr/local/squid/sbin/squiz -z
```

Y listo

3.4. Arrancando el servicio

Hecho esto, estamos listos para utilizar squid. Par ainiciar el engine del programa, sólo debemos hacer:

```
# /usr/local/squid/sbin/squid
```

De esta forma, se crearán 2 procesos que corren en el background de nuestro sistema, o como se suele decir, corre como *servicio*. Una forma rápida de comprobar que squid está funcionando es mediante el comando:

```
# ps -A | grep squid
```

Y si todo va bien, deberíamos visualizar:

```
  875 ?          00:00:00 squid
 1040 ?          00:03:04 squid
```

Tomando en cuenta que los PID (ID de proceso, por ejemplo 875 y 1040) deberían ser diferentes. Y LISTO!!!

3.5. Ejecución de Delegated como servidor socks 4-5 y servidor http

Como ya habíamos dicho, delegated puede actuar tanto como servidor proxy http , como servidor socks 4-5.

Para ejecutar el servidor socks, nada más sencillo:

```
# ./delegate8.3.3/src/delegated -P1080 SERVER=socks
```

Esto suponiendo que estamos en el directorio `delegate8.3.3`, por supuesto. Aparecerá un texto de varias líneas, lo que nos indica que está corriendo el programa, y a continuación presionamos cualquier tecla, y recuperamos el prompt `#`. Listo, para comprobar que el programa realmente está escuchando peticiones, pdremos de igual manera ejecutar:

```
# ps -A | grep delegate
```

Al igual que en el caso de squid, veremos algo similar a esto:

```
  902 ?          00:00:00 delegated
 1992 ?          00:00:00 delegated
```

Cabe anotar que cuando ejecutamos `delegated`, normalmente se iniciará un solo proceso correspondiente, como en el ejemplo, el 902. Sin embargo, una vez que inician las peticiones por parte de los clientes, se van generando más procesos de `delegated`, seguramente hijos del proceso original.

Para ejecutar el servidor proxy http, sólo hay que modificar levemente el comando para socks, quedando de la siguiente manera:

```
# ./delegate8.3.3/src/delegated -P8080 SERVER=http
```

Con lo que abriríamos un proceso que escuchará peticiones http de nuestros clientes en el puerto 8080. Tal vez ustedes se preguntarán por qué razón utilizo Squid y Delegate, pudiendo utilizar sólo Delegate, ya que también funciona como proxy http. La respuesta es que Squid me parece notablemente mejor que Delegate como servidor proxy. Cuando se utiliza el servicio http de Delegate, la navegación para los terminales se hace como más "lenta". Me parece que Delegate no se ayuda de un caché, aunque no estoy seguro de que se trate de eso. Pero en todo caso, con Squid se nota mayor velocidad de respuesta. Esa es la razón.

En realidad Delegate puede actuar, además de como servidor proxy http, y servidor socks, también como proxy ftp, nntp, smtp y otros. Para mayor información, se puede buscar en los manuales y documentación en la web de Delegate (<http://www.delegate.org/>). Más adelante, también se incluyen las páginas *man* tanto de Squid como de Delegate.

3.6. Arranque automático de estos servicios al iniciar Linux

Bien, ahora que tenemos instalados los servicios proxy http y socks que necesitábamos en nuestra red, y que sabemos cómo ejecutarlos, lo más lógico sería que estos programas arrancaran automáticamente cuando iniciamos nuestro sistema, en lugar de estar memorizando y digitando los comandos manualmente en cada ocasión, cierto???

Hacer esto, también es algo sencillo. Lo aprendí, una vez más, gracias a Ricardo Cervera, y la forma de hacerlo es:

Comencemos con Squid. Primero iniciamos sesión como root, y creamos un archivo de texto con el siguiente contenido:

```
#!/bin/bash

/usr/local/squid/sbin/squid
```

Y lo guardamos en la carpeta `/etc/init.d/` con un nombre significativo, como por ejemplo `mi_squid`. Luego, le damos al archivo el atributo de ejecutable, y creamos un enlace a este desde otra carpeta, como sigue:

```
# chmod +x /etc/init.d/mi_squid
# ln -s /etc/init.d/mi_squid /etc/rc5.d/S98squid
```

Y ya está!!! La próxima vez que iniciemos el equipo, estará automáticamente ejecutándose Squid, aún cuando no iniciemos ninguna sesión.

Para Delegate, hacemos exactamente lo mismo, solo que el archivo debería llamarse algo así como `mi_delegate`, el comando en el archivo de texto sería el adecuado para iniciar delegate (y no squid), y el nombre del enlace que crearemos podría ser, por ejemplo, `S99delegate`.

Indicar que en el nombre de los enlaces, `S` indica que iniciará un servicio, y `98` o `99`, el orden en que iniciará el servicio (respecto a los demás que inician automáticamente). En este caso, con `98` y `99` , Squid y Delegate inician en mi sistema al final, antes de ingresar al modo gráfico. Por otro lado, los enlaces los hemos incluido en la carpeta `rc5.d`, con lo cual, los servicios iniciarán siempre y cuando arranquemos en runlevel 5 (modo gráfico en mi caso), con lo cual hay que tener mucho cuidado, ya que dependiendo de algunos factores, como la distribución que estemos utilizando, o el modo en el que hayamos decidido instalar Linux, iniciaremos o no en runlevel 5. Podemos ver el runlevel que nuestro sistema arranca por defecto con:

```
# cat /etc/inittab | grep initdefault | cut -d\ : -f2
```

El último número que veamos en la salida del comando anterior será el runlevel que debemos utilizar si es distinto de 5 para crear los enlaces anteriores (sustituir el 5 por ese número).

Listo. Así es como yo lo tengo hasta ahora, y todo marcha de maravilla.

4. Páginas man y sinosys completas de Squid y Delegated

Páginas man:

4.1. Respecto a Squid-2.5STABLE-1

squid(8)

squid(8)

NAME

squid - proxy caching server

SYNOPSIS

```
squid [ -dhsvzCDFNRVYX ] [ -f config-file ] [ -[ au ] port ] [ -k sig-
nal ]
```

DESCRIPTION

squid is a high-performance proxy caching server for web clients, supporting FTP, gopher, and HTTP data objects. Unlike traditional caching software, squid handles all requests in a single, non-blocking, I/O-driven process.

squid keeps meta data and especially hot objects cached in RAM, caches

Servidor SOCKS-PROXY

DNS lookups, supports non-blocking DNS lookups, and implements negative caching of failed requests.

squid supports SSL, extensive access controls, and full request logging. By using the lightweight Internet Cache Protocol, squid caches can be arranged in a hierarchy or mesh for additional bandwidth savings.

squid consists of a main server program squid, a Domain Name System lookup program dnsserver, some optional programs for rewriting requests and performing authentication, and some management and client tools. When squid starts up, it spawns a configurable number of dnsserver pro-

cesses, each of which can perform a single, blocking Domain Name System (DNS) lookup. This reduces the amount of time the cache waits for DNS lookups.

squid is derived from the ARPA-funded Harvest Project <http://harvest.cs.colorado.edu/>

This manual page only lists the command line arguments. For details on how to configure squid see the file `/etc/squid/squid.conf`, the FAQ included with the distribution and the documentation at the squid home page <http://www.squid-cache.org>

OPTIONS

- a port
Specify HTTP port number (default: 3128).
- d level
Write debugging to stderr also.
- f file
Use the given config-file instead of `/etc/squid/squid.conf`
- h Print help message.
- k reconfigure | rotate | shutdown | interrupt | kill | debug | check | parse
Parse configuration file, then send signal to running copy (except -k parse) and exit.
- s Enable logging to syslog.

- u port
Specify ICP port number (default: 3130), disable with 0.

- v Print version.

- z Create swap directories

- C Do not catch fatal signals.

- D Disable initial DNS tests.

- F Don't serve any requests until store is rebuilt.
- N No daemon mode.
- R Do not set REUSEADDR on port.
- V Virtual host httpd-accelerator.
- X Force full debugging.
- Y Only return UDP_HIT or UDP_MISS_NOFETCH during fast reload.

FILES

/etc/squid/squid.conf
The main configuration file. You must initially make changes to this file for squid to work. For example, the default configuration does not allow access from any browser.

squid version 2.0

squid(8)

Nota: Est página man, como podemos observar, dice *squid version 2.0*, aunque es la que se incluye en la versión Squid-2.5STABLE-1.

4.2. Respecto a Delegate-8.3.3

En realidad, no he encontrado una página man sobre Delegate; en su lugar, tanto el paquete que bajamos, como en el sitio web , existe una página de manual de usuario, pero es realmente extensa, demasiado como para incluirla aquí.

Se puede ver aquí (<http://www.delegate.org/delegate/Manual.htm>).