

2. Squid looks at the *never_direct* list next. If the request matches this list, the direct flag is set to DIRECT_NO.
3. Squid has a special check for requests that appear to be looping. When Squid detects a forwarding loop, it sets the direct flag to DIRECT_YES to break the loop.
4. Squid checks the *minimum_direct_hops* and *minimum_direct_rtt* settings, but only if you've enabled *netdb*. If the measured hop count or round-trip time is lower than the configured values, Squid sets the direct flag to DIRECT_YES.
5. If none of the previous conditions are true, Squid sets the direct flag to DIRECT_MAYBE.

If the direct flag is set to DIRECT_YES, the selection process is complete. Squid forwards the request directly to the origin server and skips the remaining steps in this section.

Step 2: Neighbor Selection Protocols

Here Squid uses one of the hierarchical protocols to select a neighbor cache. As before, once Squid selects a neighbor in this step, it exits the routine and proceeds to Step 3. This step roughly corresponds to the `peerGetSomeNeighbor()` function:

1. Squid examines the neighbor's Cache Digests. If it indicates a hit, that neighbor is placed on the next-hop list.
2. Squid tries CARP if enabled. CARP always succeeds (i.e., selects a parent), unless the *cache_peer_access* or *cache_peer_domain* rules forbid communication with any of the parent caches for a particular request.
3. Squid checks *netdb* measurements (if enabled) for a "closest parent." If Squid knows that the round-trip time from one or more parents to the origin server is less than its own RTT to the origin server, Squid selects the parent with the least RTT. For this to happen, the following conditions must be met:
 - Both your Squid and the parent cache(s) must have enabled *netdb* measurements.
 - *query_icmp* must be enabled in your configuration file.
 - The origin server must respond to ICMP pings.
 - The parent(s) must have previously measured the RTT to the origin server and returned those measurements in ICP/HTCP replies, or through a *netdb* exchange.
4. Squid sends ICP/HTCP queries as the last resort. Squid loops through all neighbors and checks a number of conditions. Squid doesn't query a neighbor if:
 - The direct flag is DIRECT_MAYBE and the request is nonhierarchical (see the section "hierarchy_stoplist"). Because Squid is allowed to go directly to

the origin server, it doesn't bother the neighbor with this request, which is likely to be uncachable.

- The direct flag is `DIRECT_NO`, the neighbor is a sibling, and the request is nonhierarchical. Because Squid is forced to use a neighbor, it only queries parents, which can always handle a cache miss.
- The `cache_peer_access` or `cache_peer_domain` rules forbid sending this request to the neighbor.
- The neighbor's no-query flag is set, or its ICP/HTCP port number is zero.
- The neighbor is a multicast responder.

Squid counts how many queries it sends and calculates how many replies to expect. If it expects at least one reply, the rest of the next-hop selection procedure is postponed until the replies arrive, or a timeout occurs. Squid expects to receive replies from neighbors that are alive, but not neighbors that are dead (see the section "Neighbor State").

Step 2a: ICP/HTCP Reply Processing

If Squid sends out any ICP or HTCP queries, it waits for some number of replies. Just after transmitting the queries, Squid knows how many replies to expect and the maximum amount of time to wait for them. Squid expects a reply from every alive neighbor queried. If you're using multicast, Squid adds the current group size estimate to the expected reply count. While waiting for replies, Squid schedules a timeout, in case one or more of the replies don't arrive.

When Squid receives an ICP/HTCP reply from a neighbor, it takes the following actions:

1. If the reply is a hit, Squid forwards the request to that neighbor immediately. Any replies arriving after this point are ignored.
2. If the reply is a miss, and it is from a sibling, it is ignored.
3. Squid doesn't immediately act on ICP/HTCP misses from parents. Instead, it remembers which parents meet the following criteria:

The closest-parent miss

If the reply includes a *netdb* RTT measurement, Squid remembers the parent that has the least RTT to the origin server.

The first-parent miss

Squid remembers the parent that had the first reply. In other words, the parent with least RTT to your cache. Two *cache_peer* options affect this part of the algorithm: `weight=N` and `closest-only`.

The `weight=N` option makes a parent closer than it really is. When calculating RTTs, Squid divides the actual RTT by this artificial weight. Thus you

can give higher preference to certain parents by increasing their weight value.

The `closest-only` option disables the first-parent miss feature for a neighbor cache. In other words, Squid selects a parent (based on ICP/HTCP miss replies) only if that parent is the closest to the origin server.

4. If Squid receives the expected number of replies (all misses), or if the timeout occurs, it selects the closest-parent miss neighbor if set. Otherwise, it selects the first-parent miss neighbor if set.

Squid may not receive any ICP/HTCP replies from parent caches, either because they weren't queried or because the network dropped some packets. In this case, Squid relies on the secondary parent (or direct) selection algorithm described in the next section.

If the ICP/HTCP query timeout occurs before receiving the expected number of replies, Squid prepends the string `TIMEOUT_` to the result code in *access.log*.

Step 3: Secondary Parent Selection

This step is a little tricky. Remember that if the direct flag is `DIRECT_YES`, Squid never executes this step. If the flag is `DIRECT_NO`, Squid calls the `getSomeParent()` function (described subsequently) to select a backup parent, in case Step 2 failed to select one. Following that, Squid adds to the list all parents it believes are alive. Thus, it tries all possible parent caches before returning an error message to the user.

In the case of `DIRECT_MAYBE`, Squid adds both a parent cache, and the origin server. The order, however, depends on the *prefer_direct* setting. If *prefer_direct* is enabled, Squid inserts the origin server into the list first. Next, Squid calls `getSomeParent()` if the request is hierarchical or if the *nonhierarchical_direct* directive is disabled. Finally, Squid adds the origin server last if *prefer_direct* is disabled.

The `getSomeParent()` function selects one of the parents based on the following criteria. In each case, the parent must be alive and allowed to handle the request according to the *cache_peer_access* and *cache_peer_domain* rules:

- The first parent with the default *cache_peer* option
- The parent with the round-robin *cache_peer* option that has the lowest request count
- The first parent that is known to be alive

Retrying

Occasionally, Squid's attempt to forward a request to an origin server or neighbor may fail for one reason or another. This is why Squid creates a list of appropriate

next-hop locations during the neighbor selection procedure. When one of the following types of errors occurs, Squid can retry the request at the next server in the list:

- Network congestion or other errors can cause a “connection timeout.”
- The origin server or neighbor cache may be temporarily unavailable, causing a “connection refused” error.
- A sibling may return a 504 (Gateway Timeout) error if the request would cause a cache miss.
- A neighbor may return an “access denied” error message if the two caches have a mismatch in access control policies.
- A read error may occur on an established connection before Squid reads the HTTP message body.
- There may be race conditions with persistent connections.

Squid’s algorithm for retrying failed requests is relatively aggressive. It is better for Squid to keep trying (causing some extra delay), rather than return an error to the user.

How Do I ...

New Squid users often ask the same, or similar, questions about getting Squid to forward requests in the right way. Here I’ll show you how to configure Squid for some common scenarios.

Send All Requests Through Another Proxy?

You simply need to define a parent and tell Squid it isn’t allowed to connect directly to origin servers. For example:

```
cache_peer parent.host.name parent 3128 0
acl All src 0/0
never_direct allow All
```

The drawback to this configuration is that Squid can’t forward cache misses if the parent goes down. If that happens, your users receive the “cannot forward” error message.

Send All Requests Through Another Proxy Unless It’s Down?

Try this configuration:

```
nonhierarchical_direct off
prefer_direct off
cache_peer parent.host.name parent 3128 0 default no-query
```

Or, if you'd like to use ICP with the other proxy:

```
nonhierarchical_direct off
prefer_direct off
cache_peer parent.host.name parent 3128 3130 default
```

With this configuration, Squid forwards all cache misses to the parent as long as it is alive. Using ICP should cause Squid to detect a dead parent quickly, but at the same time may incorrectly declare the parent dead on occasion.

Make Sure Squid Doesn't Use Neighbors for Some Requests?

Define an ACL to match the special request:

```
cache_peer parent.host.name parent 3128 0
acl Special dstdomain special.server.name
always_direct allow Special
```

In this case, cache misses for requests in the *special.server.name* domain are always sent to the origin server. Other requests may, or may not, go through the parent cache.

Send Some Requests Through a Parent to Bypass Local Filters?

Some ISPs (and other organizations) have upstream providers that force HTTP traffic through a filtering proxy (perhaps with HTTP interception). You might be able to get around their filters if you can use a different proxy beyond their network. Here's how you can send only special requests to the far-away proxy:

```
cache_peer far-away-parent.host.name parent 3128 0
acl BlockedSites dstdomain www.censored.com
cache_peer_access far-away-parent.host.name allow BlockedSites
never_direct allow BlockedSites
```

Exercises

- Toggle your *prefer_direct* and/or *nonhierarchical_direct* settings and look for any changes in the *access.log*.
- Enable *netdb* and view the *netdb* cache manager page after Squid has been running for a while.
- If using ICP or HTCP, count the percentage of requests that experienced a timeout waiting for replies to arrive.
- If you used *--enable-cache-digests* and have a reasonably full cache, disable the *digest_generation* directive and note any change in memory usage.

- Use your operating system's packet filters to block ICP or HTCP messages to your neighbors. How quickly does Squid change their state from alive to dead, and back again?

Redirectors

A *redirector* is an external process that rewrites URIs from client requests. For example, although a user requests the page `http://www.example.com/page1.html`, a redirector can change the request to something else, such as `http://www.example.com/page2.html`. Squid fetches the new URI automatically, as though the client originally requested it. If the response is cachable, Squid stores it under the new URI.

The redirector feature allows you to implement a number of interesting things with Squid. Many sites use them for access controls, removing advertisements, local mirrors, or even working around browser bugs.

One of the nice things about using a redirector for access control is that you can send the user to a page that explains exactly why her request is denied. You may also find that a redirector offers more flexibility than Squid's built-in access controls. As you'll see shortly, however, a redirector doesn't have access to the full spectrum of information contained in a client's request.

Many people use a redirector to filter out web page advertisements. In most cases, this involves changing a request for a GIF or JPEG advertisement image into a request for a small, blank image, located on a local server. Thus, the advertisement just "disappears" and doesn't interfere with the page layout.

So in essence, a redirector is really just a program that reads a URI and other information from its input and writes a new URI on its output. Perl and Python are popular languages for redirectors, although some authors use compiled languages such as C for better performance.

The Squid source code doesn't come with any redirector programs. As an administrator, you are responsible for writing your own or downloading one written by someone else. The first part of this chapter describes the interface between Squid and a redirector process. I also provide a couple of simple redirector examples in Perl. If you're interested in using someone else's redirector, rather than programming your own, skip ahead to the "The Redirector Pool" section.

The Redirector Interface

A redirector receives data from Squid on *stdin* one line at a time. Each line contains the following four tokens separated by whitespace:

- Request-URI
- Client IP address and fully qualified domain name
- User's name, via either RFC 1413 ident or proxy authentication
- HTTP request method

For example:

```
http://www.example.com/page1.html 192.168.2.3/user.host.name jabroni GET
```

The Request-URI is taken from the client's request, including query terms, if any. Fragment identifier components (e.g., the # character and subsequent text) are removed, however.

The second token contains the client IP address and, optionally, its fully qualified domain name (FQDN). The FQDN is set only if you enable the *log_fqdn* directive or use a *srcdomain* ACL element. Even then, the FQDN may be unknown because the client's network administrators didn't properly set up the reverse pointer zones in their DNS. If Squid doesn't know the client's FQDN, it places a hyphen (-) in the field. For example:

```
http://www.example.com/page1.html 192.168.2.3/- jabroni GET
```

The client ident field is set if Squid knows the name of the user behind the request. This happens if you use proxy authentication, *ident* ACL elements, or enable *ident_lookup_access*. Remember, however, that the *ident_lookup_access* directive doesn't cause Squid to delay request processing. In other words, if you enable that directive, but don't use the access controls, Squid may not yet know the username when writing to the redirector process. If Squid doesn't know the username, it displays a -. For example:

```
http://www.example.com/page1.html 192.168.2.3/- - GET
```

Squid reads back one token from the redirector process: a URI. If Squid reads a blank line, the original URI remains unchanged.

A redirector program should never exit until end-of-file occurs on *stdin*. If the process does exit prematurely, Squid writes a warning to *cache.log*:

```
WARNING: redirector #2 (FD 18) exited
```

If 50% of the redirector processes exit prematurely, Squid aborts with a fatal error message.

Handling URIs That Contain Whitespace

If the Request-URI contains whitespace, and the *uri_whitespace* directive is set to allow, any whitespace in the URI is passed to the redirector. A redirector with a simple parser may become confused in this case. You have two options for handling whitespace in URIs when using a redirector.

One option is to set the *uri_whitespace* directive to anything except allow. The default setting, strip, is probably a good choice in most situations because Squid simply removes the whitespace from the URI when it parses the HTTP request. See Appendix A for information on the other values for this directive.

If that isn't an option, you need to make sure the redirector's parser is smart enough to detect the extra tokens. For example, if it finds more than four tokens in the line received from Squid, it can assume that the last three are the IP address, ident, and request method. Everything before the third-to-last token comprises the Request-URI.

Generating HTTP Redirect Messages

When a redirector changes the client's URI, it normally doesn't know that Squid decided to fetch a different resource. This is, in all likelihood, a gross violation of the HTTP RFC. If you want to be nicer, and remain compliant, there is a little trick that makes Squid return an HTTP redirect message. Simply have the redirector insert 301:, 302:, 303:, or 307:, before the new URI.

For example, if a redirector writes this line on its *stdout*:

```
301:http://www.example.com/page2.html
```

Squid sends a response like this back to the client:

```
HTTP/1.0 301 Moved Permanently
Server: squid/2.5.STABLE4
Date: Mon, 29 Sep 2003 04:06:23 GMT
Content-Length: 0
Location: http://www.example.com/page2.html
X-Cache: MISS from zoidberg
Proxy-Connection: close
```

Some Sample Redirectors

Example 11-1 is a very simple redirector written in Perl. Its purpose is to send HTTP requests for the *squid-cache.org* site to a local mirror site in Australia. If the requested URI looks like it is for *www.squid-cache.org* or one of its mirror sites, this script outputs a new URI with the hostname set to *www1.au.squid-cache.org*.

A common problem first-time redirector writers encounter is *buffered I/O*. Note that here I make sure *stdout* is unbuffered.

Example 11-1. A simple redirector in Perl

```
#!/usr/bin/perl -wl
$|=1; # don't buffer the output
while (<>) {
    ($uri,$client,$ident,$method) = ();
    ($uri,$client,$ident,$method) = split;
    next unless ($uri =~ m,^http://.*\.squid-cache\.org(\S*),);
    $uri = "http://www1.au.squid-cache.org$1";
} continue {
    print "$uri";
}
```

Example 11-2 is another, somewhat more complicated, example. Here I make a feeble attempt to deny requests when the URI contains “bad words.” This script demonstrates an alternative way to parse the input fields. If I don’t get all five required fields, the redirector returns a blank line, leaving the request unchanged.

This example also gives preferential treatment to some users. If the ident string is equal to “BigBoss,” or comes from the 192.168.4.0 subnet, the request is passed through. Finally, I use the 301: trick to make Squid return an HTTP redirect to the client. Note, this program is neither efficient nor smart enough to correctly deny so-called bad requests.

Example 11-2. A slightly less simple redirector in Perl

```
#!/usr/bin/perl -wl
$|=1; # don't buffer the output

$DENIED = "http://www.example.com/denied.html";
&load_word_list();

while (<>) {
    unless (m,(\S+) (\S+)/(\S+) (\S+) (\S+),) {
        $uri = '';
        next;
    }
    $uri = $1;
    $ipaddr = $2;
    $fqdn = $3;
    $ident = $4;
    $method = $5;
    next if ($ident eq 'TheBoss');
    next if ($ipaddr =~ /^192\.168\.4\../);
    $uri = "301:$DENIED" if &word_match($uri);
} continue {
    print "$uri";
}

sub load_word_list {
    @words = qw(sex drugs rock roll);
}
```

Example 11-2. A slightly less simple redirector in Perl (continued)

```
sub word_match {
    my $uri = shift;
    foreach $w (@words) { return 1 if ($uri =~ /$w/); }
    return 0;
}
```

For more ideas about writing your own redirector, I recommend reading the source code for the redirectors mentioned in the section “Popular Redirectors.”

The Redirector Pool

A redirector can take an arbitrarily long time to return its answer. For example, it may need to make a database query, search through long lists of regular expressions, or make some complex computations. Squid uses a pool of redirector processes so that they can all work in parallel. While one is busy, Squid hands a new request off to another.

For each new request, Squid examines the pool of redirector processes in order. It submits the request to the first idle process. If your request rate is very low, the first redirector may be able to handle all requests itself.

You can control the size of the redirector pool with the *redirect_children* directive. The default value is five processes. Note that Squid doesn’t dynamically increase or decrease the size of the pool depending on the load. Thus, it is a good idea to be a little liberal. If all redirectors are busy, Squid queues pending requests. If the queue becomes too large (bigger than twice the pool size), Squid exits with a fatal error message:

```
FATAL: Too many queued redirector requests
```

In this case, you need to increase the size of the redirector pool or change something so that the redirectors can process requests faster. You can use the cache manager’s *redirector* page to find out if you have too few, or too many redirectors running. For example:

```
% squidclient mgr:redirector
...
Redirector Statistics:
program: /usr/local/squid/bin/myredir
number running: 5 of 5
requests sent: 147
replies received: 142
queue length: 2
avg service time: 953.83 msec
```

#	FD	PID	# Requests	Flags	Time	Offset	Request
1	10	35200	46	AB	0.902	0	http://...
2	11	35201	29	AB	0.401	0	http://...
3	12	35202	25	AB	1.009	1	cache_o...

4	14	35203	25	AB	0.555	0 http://...
5	15	35204	21	AB	0.222	0 http://...

If, as in this example, you see that the last redirector has almost as many requests as the second to last, you should probably increase the size of the redirector pool. If, on the other hand, you see many redirectors with no requests, you can probably decrease the pool size.

Configuring Squid

The following five *squid.conf* directives control the behavior of redirectors in Squid.

redirect_program

The *redirect_program* directive specifies the command line for the redirector program. For example:

```
redirect_program /usr/local/squid/bin/my_redirector -xyz
```

Note, the redirector program must be executable by the Squid user ID. If, for some reason, Squid can't execute the redirector, you should see an error message in *cache.log*.^{*} For example:

```
ipcCreate: /usr/local/squid/bin/my_redirector: (13) Permission denied
```

Due to the way Squid works, the main Squid process may be unaware of problems executing the redirector program. Squid doesn't detect the error until it tries to write a request and read a response. It then prints:

```
WARNING: redirector #1 (FD 6) exited
```

Thus, if you see such a message for the first request sent to Squid, check *cache.log* closely for other errors, and make sure the program is executable by Squid.

redirect_children

The *redirect_children* directive specifies how many redirector processes Squid should start. For example:

```
redirect_children 20
```

Squid warns you (via *cache.log*) when all redirectors are simultaneously busy:

```
WARNING: All redirector processes are busy.
WARNING: 1 pending requests queued.
```

^{*} This message appears only in *cache.log*, and not on *stdout*, if you use the *-d* option, or in *syslog*, if you use the *-s* option.

If you see this warning, you should increase the number of child processes and restart (or reconfigure) Squid. If the queue size becomes twice the number of redirectors, Squid aborts with a fatal message.

Don't attempt to disable Squid's use of the redirectors by setting *redirect_children* to 0. Instead, simply remove the *redirect_program* line from *squid.conf*.

redirect_rewrites_host_header

Squid normally updates a request's Host header when using a redirector. That is, if the redirector returns a new URI with a different hostname, Squid puts the new hostname in the Host header. If you use Squid as a surrogate (see Chapter 15), you might want to disable this behavior by setting the *redirect_rewrites_host_header* directive to off:

```
redirect_rewrites_host_header off
```

redirector_access

Squid normally sends every request through a redirector. However, you can use the *redirector_access* rules to send certain requests through selectively. The syntax is identical to *http_access*:

```
redirector_access allow|deny [!]ACLname ...
```

For example:

```
acl Foo src 192.168.1.0/24
acl All src 0/0
redirector_access deny Foo
redirector_access allow All
```

In this case, Squid skips the redirector for any request that matches the *Foo* ACL.

redirector_bypass

If you enable the *redirector_bypass* directive, Squid bypasses the redirectors when all of them are busy. Normally, Squid queues pending requests until a redirector process becomes available. If this queue grows too large, Squid exits with a fatal error message. Enabling this directive ensures that Squid never reaches that state.

The tradeoff, of course, is that some user requests may not be redirected when the load is high. If that's all right with you, simply enable the directive with this line:

```
redirector_bypass on
```

Popular Redirectors

As I already mentioned, the Squid source code doesn't include any redirectors. However, you can find a number of useful third-party redirectors linked from the *Related Software* page on <http://www.squid-cache.org>. Here are some of the more popular offerings:

Squirm

<http://squirm.foote.com.au/>

Squirm comes from Chris Foote. It is written in C and distributed as source code under the GNU General Public License (GPL). Squirm's features include:

- Being very fast with minimal memory usage
- Full regular expression pattern matching and replacement
- Ability to apply different redirection lists to different client groups
- Interactive mode for testing on the command line
- Fail-safe mode passes requests through unchanged in the event that configuration files contain errors
- Writing debugging, errors, and more to various log files

Jesred

<http://www.linofee.org/~elkner/webtools/jesred/>

Jesred comes from Jens Elkner. It is written in C, based on Squirm, and also released under the GNU GPL. Its features include:

- Being faster than Squirm, with slightly more memory usage
- Ability to reread its configuration files while running
- Full regular expression pattern matching and replacement
- Fail-safe mode passes requests through unchanged in the event that configuration files contain errors
- Optionally logging rewritten requests to a log file

squidGuard

<http://www.squidguard.org/>

squidGuard comes from Pål Baltzersen and Lars Erik Håland at Tele Danmark Inter-Nordia. It is released under the GNU GPL. The authors also make sure squidGuard compiles easily on modern Unix systems. Their site contains a lot of good documentation. Here are some of squidGuard's features:

- Highly configurable; you can apply different rules to different groups of clients or users and at different times or days
- URI substitution, not just replacement, à la sed
- printf-like substitutions allow passing parameters to CGI scripts for customized messages
- Supportive of the 301/302/303/307 HTTP redirect status code feature for redirectors
- Selective logging for rewrite rule sets

At the squidGuard site, you can also find a blacklist of more than 100,000 sites categorized as porn, aggressive, drugs, hacking, ads, and more.

AdZapper

<http://www.zip.com.au/~cs/adzap/index.html>

AdZapper is a popular redirector because it specifically targets removal of advertisements from HTML pages. It is a Perl script written by Cameron Simpson. AdZapper can block banners (images), pop-up windows, flash animations, page counters, and web bugs. The script includes a list of regular expressions that match URIs known to contain ads, pop-ups, etc. Cameron updates the script periodically with new patterns. You can also maintain your own list of patterns.

Exercises

- Write a redirector that never changes the requested URI and configure Squid to use it.
- While running **tail -f cache.log**, kill Squid's redirector processes one by one until something interesting happens.
- Download and install one of the redirectors mentioned in the previous section.

CHAPTER 12

Authentication Helpers

I originally talked about proxy authentication in Chapter 6. However, I only explained how to write access control rules that use proxy authentication. Here, I'll show you how to select and configure the particular authentication helpers.

Recall that Squid supports three methods for gathering authentication credentials from users: Basic, Digest, and NTLM. These methods specify how Squid receives the username and password from a client. From a security standpoint, Basic authentication is extremely weak. Digest and NTLM are significantly stronger. For each method, Squid provides some authentication modules, or helper processes, which actually validate the credentials.

All of the authentication helpers that I mention here are included in the Squid source code distribution. You can compile them with `./configure` options that match their directory names. For example:

```
% ls helpers/basic_auth
LDAP          NCSA          getpwnam
MSNT          PAM           multi-domain-NTLM
Makefile      SASL          winbind
Makefile.am   SMB
Makefile.in   YP
```

```
% ./configure --enable-basic-auth-helpers=LDAP,NCSA ...
```

Helper programs are normally installed in the `$prefix/libexec` directory.

As with redirectors, Squid uses a pool of authentication helper processes. A request for authentication is sent to the first idle helper. When all authenticator processes are busy, Squid queues pending requests. If the queue becomes too large, Squid exits with a fatal error message. In most cases, Squid caches authentication results. This reduces the load on the helper processes and improves response time.

Configuring Squid

The `auth_param` directive controls every aspect of configuring Squid's authentication helpers. The different methods (Basic, Digest, NTLM) have some things in common, and some unique parameters. The first argument following `auth_param` must be one of `basic`, `digest`, or `ntlm`. I'll cover this directive in detail for each authentication scheme later in the chapter.

In addition to `auth_param`, Squid has two more directives that affect proxy authentication. You can use the `max_user_ip` ACL to prevent users from sharing their username and password with others. If Squid detects the same username coming from too many different IP addresses, the ACL is a match and you can deny the request. For example:

```
acl FOO max_user_ip 2
acl BAR proxy_auth REQUIRED
http_access deny FOO
http_access allow BAR
```

In this case, if a user submits requests from three or more different IP addresses, Squid denies the request. The `authenticate_ip_ttl` directive controls how long Squid remembers the source IP addresses for each user. A smaller TTL makes it easier for users with frequently changing IP addresses. You can use larger TTLs in an environment where users have the same IP address for long periods of time.

HTTP Basic Authentication

Basic authentication is the simplest and least secure that HTTP has to offer. It essentially transmits user passwords as cleartext, although they are encoded into printable characters. For example, if the user types her name as Fannie and her password as FuRpAnTsClUb, the user-agent first combines the two into a single string, with name and password separated by a colon:

```
Fannie:FuRpAnTsClUb
```

Then it encodes this string with base64 encoding, as defined in RFC 2045. It looks like this in the HTTP headers:

```
Authorization: Basic RmFubm110kZ1UnBBb1RzQ2xVYgo=
```

Anyone who happens to capture your users' HTTP requests can easily get both the username and password:

```
% echo RmFubm110kZ1UnBBb1RzQ2xVYgo= | /usr/local/lib/python1.5/base64.py -d
Fannie:FuRpAnTsClUb
```

As required by the HTTP/1.1 RFC, Squid doesn't forward "consumed" authorization credentials to other servers. In other words, if the credentials are for access to Squid, the Authorization header is removed from outgoing requests.*

You'll notice that some of the Basic authenticators can be configured to check the system password file. Because Basic credentials aren't encrypted, it is a bad idea to combine login passwords with cache access passwords. If you choose to use the *getpwnam* authenticator, make sure you fully understand the implications of having your users' passwords transmitted in the clear across your network.

HTTP Basic authentication supports the following *auth_param* parameters:

- *auth_param basic program command*
- *auth_param basic children number*
- *auth_param basic realm string*
- *auth_param basic credentialsttl time-specification*

The *program* parameter specifies the command, including arguments, for the helper program. In most cases, this will be the pathname to one of the authentication helper programs that you compiled. By default, they live in */usr/local/squid/libexec*.

The *children* parameter tells Squid how many helper processes to use. The default value is 5, which is a good starting point if you don't know how many Squid needs to handle the load. If you specify too few, Squid warns you with messages in *cache.log*.

The *realm* parameter is the authentication realm string that the user-agent should present to the user when prompting for a username and password. You can use something simple, such as "access to the Squid caching proxy."

The *credentialsttl* parameter specifies the amount of time that Squid internally caches authentication results. A larger value reduces the load on the external authenticator processes, but increases the amount of time until Squid detects changes to the authentication database. Note, this only affects positive results (i.e., successful validations). Negative results aren't cached inside Squid. The default TTL value is two hours.

Here is a complete example:

```
auth_param basic program /usr/local/squid/libexec/pam_auth
auth_param basic children 10
auth_param basic realm My Awesome Squid Cache
auth_param basic credentialsttl 1 hour

acl KnownUsers proxy_auth REQUIRED
http_access allow KnownUsers
```

Next I will discuss the Basic authentication helper programs that come with Squid.

* Unless you configure a peer with the *login=PASS* option.

NCSA

```
./configure --enable-basic-auth-helpers=NCSA
```

The NCSA authentication helper is relatively popular due to its simplicity and history. It stores usernames and passwords in a single text file, similar to the Unix */etc/passwd* file. This password file format was originally developed as a part of the NCSA HTTP server project.

You pass the path to the password file as the program's single command-line argument in *squid.conf*:

```
auth_param basic program /usr/local/squid/libexec/ncsa_auth
                /usr/local/squid/etc/passwd
```

You can use the *htpasswd* program that comes with Apache to create and update the password file. Also, you can download it from <http://www.squid-cache.org/htpasswd/>. From that page, you can also download the *chpasswd* CGI script, which allows users to change their own passwords if necessary.

LDAP

```
./configure --enable-basic-auth-helpers=LDAP
```

The LDAP helper interfaces to a Lightweight Directory Access Protocol server. The OpenLDAP libraries and header files must be installed before you can compile the *squid_ldap_auth* helper. You can find OpenLDAP at <http://www.openldap.org/>.

The *squid_ldap_auth* program requires at least two arguments: the base distinguished name (DN) and the LDAP server hostname. For example:

```
auth_param basic program /usr/local/squid/libexec/squid_ldap_auth
                -b "ou=people,dc=example,dc=com" ldap.example.com
```

The LDAP helper has a Unix manual page that describes all of its options and parameters. However, Squid's manual pages aren't normally installed when you run *make install*. You can read the manual page by locating it in the source tree and manually running *nroff*. For example:

```
% cd helpers/basic_auth/LDAP
% nroff -man squid_ldap_auth.8 | less
```

MSNT

```
./configure --enable-basic-auth-helpers=MSNT
```

The MSNT authenticator interfaces to a Microsoft NT domain database via the Server Message Block (SMB) protocol. It uses a small configuration file, named *msntauth.conf*, which must be placed in the *\$prefix/etc* or *--sysconfdir* directory.

You can specify up to five NT domain controllers in the configuration file. For example:

```
server pdc1_host bdc1_host my_nt_domain
server pdc2_host bdc2_host another_nt_domain
```

By default, the MSNT authenticator allows any user validated by the server. However, it also has the ability to allow or deny specific usernames. If you create an *allowusers* file, only the users listed there are allowed access to Squid. You might want to use this feature if you have a large number of users on the NT server, but only a small number who are allowed to use the cache. Alternatively, you can create a *denyusers* file. Any user listed in that file is automatically denied access, even before checking the *allowusers* file.

Alternatively, you can allow or deny specific usernames by placing them in the proxy_auth ACL as described in Chapter 6.

For additional documentation, see the *README.html* file in the *helpers/basic_auth/MSNT* directory.

Multi-domain-NTLM

```
./configure --enable-basic-auth-helpers=multi-domain-NTLM
```

The multi-domain-NTLM authenticator is similar to MSNT. Both send queries to a Windows NT domain database. Whereas MSNT queries up to five domain controllers, the multi-domain-NTLM authenticator requires users to insert the NT domain name before their username, like this:

```
ntdomain\username
```

The multi-domain-NTLM helper program is a relatively short Perl script. It relies on the Authen::SMB package from CPAN (<http://www.cpan.org>). If you don't hardcode the domain controller hostnames in the Perl script, it utilizes the nmblookup program from the Samba package (www.samba.org) to discover them automatically.

The Perl script is named *smb_auth.pl*. It might look like this in *squid.conf*:

```
auth_param basic program /usr/local/squid/libexec/smb_auth.pl
```

Documentation for multi-domain-NTLM is thin, but if you understand Perl, you should be able to figure it out by reading the code.

PAM

```
./configure --enable-basic-auth-helpers=PAM
```

In a sense, Pluggable Authentication Modules (PAM) are the glue between authentication methods (e.g., one-time passwords, kerberos, smart cards) and applications requiring authentication services (e.g., ssh, ftp, imap). Your system's */etc/pam.conf* file describes which methods to use for each application.

To use Squid's PAM authentication helper, you need to add "squid" as a service in the */etc/pam.conf* file and specify which PAM modules to use. For example, to use the Unix password file on FreeBSD, you might put this in *pam.conf*:

```
squid auth required pam_unix.so try_first_pass
```



To check the Unix password database, the `pam_auth` process must run as root. This is a security risk and you must manually make the executable `setuid` root. If `pam_auth` doesn't run as root, and it is configured to check the Unix password database, every request for authentication fails.

The PAM authenticator is documented with a manual page that you can find in the *helpers/basic_auth/PAM* directory.

SASL

```
./configure --enable-basic-auth-helpers=SASL
```

The Simple Authentication and Security Layer (SASL) is an IETF proposed standard, documented in RFC 2222. It is a protocol for negotiating security parameters for connection-based protocols (e.g., FTP, SMTP, HTTP). However, the SASL authenticator is similar to the PAM authenticator. It interfaces with a third-party library to query a number of different authentication databases.

Specifically, Squid's SASL authenticator requires the Cyrus SASL library developed by Carnegie Mellon University. You can find it at <http://asg.web.cmu.edu/sasl/>.

You can configure the SASL authenticator to check the traditional password file, the PAM system, or any of the other databases supported by CMU's library. For further information, see the *README* file in the *helpers/basic_auth/SASL* directory.

SMB

```
./configure --enable-basic-auth-helpers=SMB
```

SMB is another authenticator for Microsoft Windows databases. The authenticator itself is a C program. That program executes a shell script each time it talks to the Windows domain controller. The shell script contains commands from the Samba package. Thus, you'll need to install Samba before using the SMB authenticator.

The SMB authenticator program, `smb_auth` takes the Windows domain name as an argument. For example:

```
auth_param basic program /usr/local/squid/libexec/smb_auth -W MYNTDOMAIN
```

You can list multiple domains by repeating the `-W` option. For full documentation, see http://www.hacom.nl/~richard/software/smb_auth.html.

YP

```
./configure --enable-basic-auth-helpers=YP
```

The YP authenticator checks a system's "Yellow Pages" (a.k.a. NIS) directory. To use it with Squid, you need to provide the NIS domain name and the name of the password database, usually `passwd.byname` on the authenticator command line:

```
auth_param basic program /usr/local/squid/libexec/yp_auth my.nis.domain passwd.byname
```

The `yp_auth` program is relatively simple, but doesn't have any documentation.

getpwnam

```
./configure --enable-basic-auth-helpers=getpwnam
```

This authenticator is simply an interface to the `getpwnam()` function found in the C library on Unix systems. The `getpwnam()` function looks in the system password file for a given username. If you use YP/NIS, `getpwnam()` checks those databases as well. On some operating systems, it may also utilize the PAM system. You can use this authenticator if your cache users have login accounts on the system where Squid is running. Alternatively, you could set up "nologin" accounts in the password file for your cache users.

winbind

```
./configure --enable-basic-auth-helpers=winbind
```

Winbind is a feature of the Samba suite of software. It allows Unix systems to utilize Windows NT user account information. The *winbind* authenticator is a client for the Samba `winbindd` daemon. You must have Samba installed and the `winbindd` daemon running before you can use this authenticator.

The name of the *winbind* Basic authenticator is *wb_basic_auth*. It typically looks like this in *squid.conf*:

```
auth_param basic program /usr/local/squid/libexec/wb_basic_auth
```

The Basic Auth API

The interface between Squid and a Basic authenticator is quite simple. Squid sends usernames and passwords to the authenticator process, separated by a space and terminated by a newline. The authenticator reads the username and password pairs on *stdin*. After checking the credentials, the authenticator writes either OK or ERR to *stdout*.



Any “URL-unsafe” characters are encoded according to the RFC 1738 rules. Thus, the name “jack+jill” becomes “jack%2bjill”. Squid accepts usernames and passwords that contain whitespace characters. For example “a password” becomes “a%20password”. The authenticator program should be prepared to handle whitespace and other special characters after decoding the name and password.

You can easily test a Basic authenticator on the command line. Simply run the authenticator program in a terminal window and enter usernames and passwords. Or, you can do it like this:

```
% echo "bueller pencil" | ./ncsa_auth /tmp/passwd
OK
```

Here is a simple template authenticator written in Perl:

```
#!/usr/bin/perl -w

use URI::Escape;

$|=1;                                # don't buffer stdout
while (<>) {
    ($u,$p) = split;
    $u = uri_unescape($u);
    $p = uri_unescape($p);
    if (&valid($u,$p)) {
        print "OK";
    } else {
        print "ERR";
    }
}

sub valid {
    my $user = shift;
    my $pass = shift;
    ...
}
```

HTTP Digest Authentication

Digest authentication is designed to be significantly more secure than Basic. It makes extensive use of cryptographic hash functions and other tricks. Essentially, instead of sending a cleartext password, the user-agent sends a “message digest” of the password, username, and other information. (See RFC 2617 and O’Reilly’s *HTTP: The Definitive Guide* for more information.)

HTTP Digest authentication supports the following *auth_param* parameters:

- *auth_param* digest program *command*
- *auth_param* digest children *number*

- `auth_param digest realm` *string*
- `auth_param digest nonce_garbage_interval` *time-specification*
- `auth_param digest nonce_max_duration` *time-specification*
- `auth_param digest nonce_max_count` *number*
- `auth_param digest nonce_strictness` `on|off`

The `program`, `children`, and `realm` parameters are the same as for Basic authentication. All of the unique parameters relate to Digest authentication's use of something called *nonce*.

A nonce is a special string of data, which changes occasionally. During the authentication process, the server (Squid in this case) provides a nonce value to the client. The client uses the nonce value when generating the digest. Without the nonce data, an attacker could simply intercept and replay the digest values to gain access to Squid.

The `nonce_garbage_interval` parameter tells Squid how often to clean up the nonce cache. The default value is every 5 minutes. A very busy cache with many Digest authentication clients may benefit from more frequent nonce garbage collection.

The `nonce_max_duration` parameter specifies how long each nonce value remains valid. When a client attempts to use a nonce value older than the specified time, Squid generates a 401 (Unauthorized) response and sends along a fresh nonce value so the client can re-authenticate. The default value is 30 minutes. Note that any captured Authorization headers can be used in a replay attack until the nonce value expires. Setting the `nonce_max_duration` too low, however, causes Squid to generate 401 responses more often. Each 401 response essentially wastes the user's time as the client and server renegotiate their authentication credentials.

The `nonce_max_count` parameter places an upper limit on how many times a nonce value may be used. After the specified number of requests, Squid returns a 401 (Unauthorized) response and a new nonce value. The default is 50 requests.

Nonce counts are another feature designed to prevent replay attacks. Squid sends `qop=auth` in its 401 responses. This causes user-agents to include a nonce count in their requests, and to use the nonce count when generating the digest itself. Nonce count values must always increase over time. A decreasing nonce count indicates a replay attack. However, the counts may increase, but skip some values, for example: 5,6,8,9. The `nonce_strictness` parameter determines what Squid does in this case. If set to `on`, Squid returns a 401 response if a nonce count doesn't equal the previous nonce count plus one. If set to `off`, Squid allows gaps in the nonce count values.

Here is a complete example:

```
auth_param digest program /usr/local/squid/libexec/digest_pw
auth_param digest children 8
auth_param digest realm Access to Squid
auth_param digest nonce_garbage_interval 10 minutes
```

```
auth_param digest nonce_max_duration 45 minutes
auth_param digest nonce_max_count 100
auth_param digest nonce_strictness on
```

```
acl KnownUsers proxy_auth REQUIRED
http_access allow KnownUsers
```

Next I will discuss the Digest authentication helper programs that come with Squid.

password

```
./configure --enable-auth=digest --enable-digest-auth-helpers=password
```

This is a simple, reference implementation of Digest authentication for Squid. It demonstrates how to write a Digest-based authentication helper. This code simply reads usernames and passwords from a plaintext file. The format of this file is as follows:

```
username:password
```

The password file pathname is the single argument to the *digest_pw_auth* program. For example:

```
auth_param digest program /usr/local/squid/libexec/digest_pw_auth
                        /usr/local/squid/etc/digest_passwd
auth_param digest realm Some Nifty Realm
```

Squid doesn't provide any tools to maintain a password file in this format. If you choose to use Digest authentication, you must manage the file on your own, perhaps with a text editor or Perl scripts.

Digest Authentication API

If you'd like to write your own Digest authentication helper, you need to understand the communication between Squid and the helper process. The exchange is similar to that for Basic authentication, albeit a little more complicated.

The first difference is that Squid writes the username and realm string, rather than username and password, to the helper process. These strings are quoted and separated by a colon. For example:

```
"bobby": "Tom Landry Middle School"
```

The second difference is that the helper process returns an MD5 digest string, rather than OK, if the username is valid. As with Basic authentication, the helper process writes ERR if the user doesn't exist or if the input from Squid is unparseable for some reason.

The helper returns an MD5 digest with the username, realm, and password. The three strings are concatenated together and separated by colons:

```
username:realm:password
```

Remember that the password isn't sent in the HTTP request. Rather, the helper retrieves the user's password from a database (like the plaintext file used by the *password* helper). For example, let's say that Bobby's password is CapeRs. The helper process receives the username and realm from Squid, gets the password from its database, and calculates an MD5 checksum of this string:

```
bobby:Tom Landry Middle School:CapeRs
```

The Squid source code includes a library function, `DigestCalcHA1()`, which implements this calculation. We can test all this in a terminal window to see what the helper returns:

```
% echo 'bobby:CapeRs' > /tmp/pw
% echo bogus_input | digest_pw_auth /tmp/pw
ERR
% echo "nouser":"some realm" | digest_pw_auth /tmp/pw
ERR
% echo '"bobby":"Tom Landry Middle School"' | digest_pw_auth /tmp/pw
c7ca3efda238c65b2d48684a51baa90e
```

Squid stores this MD5 checksum and uses it in other parts of the Digest authentication algorithm. Note that the checksum only changes when the user changes his password. In Squid's current Digest implementation, these checksums are kept in memory as long as the user remains active. If the user is inactive for *authenticate_ttl* seconds, the MD5 checksum may be removed from Squid's memory. Upon the next request from that user, Squid asks the external helper process to calculate it again.

Microsoft NTLM Authentication

NTLM* is a proprietary connection authentication protocol from Microsoft. A number of groups, including the Squid developers, have reverse-engineered the protocol from what little information is available and by examining network traffic. You can find some technical details at <http://www.innovation.ch/java/ntlm.html>.

NTLM uses a three-way handshake to authenticate a connection. First, the client sends its request with a couple of identifiers. Second, the server sends back a challenge message. Third, the client sends its request again with a response to the challenge. At this point, the connection is authenticated and any further requests on the same connection don't require any challenge/response information. If the connection is closed, the client and server must repeat the entire three-way handshake. Persistent connections help reduce this overhead for NTLM.

NTLM uses cryptographic hash functions and nonce values, similar to Digest authentication, although experts believe NTLM is weaker.

NTLM authentication supports the following *auth_param* parameters:

* NTLM apparently stands for "NT LanMan" or perhaps "NT Lan Manager."

- `auth_param ntlm program command`
- `auth_param ntlm children number`
- `auth_param ntlm max_challenge_reuses number`
- `auth_param ntlm max_challenge_lifetime time-specification`

The `program` and `children` parameters are the same as for Basic and Digest authentication. The remaining parameters determine how often Squid may reuse a single challenge token.

The `max_challenge_reuses` parameter specifies how many times a challenge token may be reused. The default value is 0, so that challenges are never reused. Increasing this value may reduce the computational load on Squid and the NTLM helper processes, at the risk of weakening the protocol's security.

Similarly, the `max_challenge_lifetime` parameter places a time limit on challenge reuses, even if the `max_challenge_reuses` count has not been reached. The default value is 60 seconds.

Here is a complete example:

```
auth_param ntlm program /usr/local/squid/libexec/ntlm_auth foo\bar
auth_param ntlm children 12
auth_param ntlm max_challenge_reuses 5
auth_param ntlm max_challenge_lifetime 2 minutes

acl KnownUsers proxy_auth REQUIRED
http_access allow KnownUsers
```

Squid comes with the following NTLM authentication helper programs:

SMB

```
./configure --enable-auth=ntlm --enable-ntlm-auth-helpers=SMB
```

The Server Message Block (SMB) authenticator for NTLM is similar to those for Basic authentication. Your users can simply supply their Windows NT domain, username, and password. This authenticator can load balance between multiple domain controllers. The domain and controller names go on the command line:

```
auth_param ntlm program /usr/local/squid/libexec/ntlm_auth
domain\controller [domain\controller ...]
```

winbind

```
./configure --enable-auth=ntlm --enable-ntlm-auth-helpers=winbind
```

This authenticator is similar to *winbind* for Basic authentication. Both require that you have the Samba *winbindd* daemon installed and running. The name of the *winbind* Basic authenticator is *wb_ntlm_auth*. It typically looks like this in *squid.conf*:

```
auth_param basic program /usr/local/squid/libexec/wb_ntlm_auth
```

NTLM Authentication API

The communication between Squid and an NTLM authenticator is much more complicated than for Basic and Digest. One reason is that each helper process actually creates its own challenge. Thus, helpers become “stateful” and Squid must remember which connections belong to which helpers.

Squid and the helper processes use a handful of two-character codes to indicate what they are sending. Those codes are as follows:

YR Squid sends this to a helper when it needs a new challenge token. This is always the first communication between the two processes. It may also occur at any time that Squid needs a new challenge, due to the *auth_param* *max_challenge_lifetime* and *max_challenge_uses* parameters. The helper should respond with a **TT** message.

TT *challenge*

A helper sends this message back to Squid and includes a challenge token. It is sent in response to a **YR** request. The challenge is base64-encoded, as defined by RFC 2045.

KK *credentials*

Squid sends this to a helper when it wants to authenticate a user’s credentials. The helper responds with either **AF**, **NA**, **BH**, or **LD**.

AF *username*

The helper sends this message back to Squid when the user’s authentication credentials are valid. The helper sends the username with this message because Squid doesn’t try to decode the NTLM Authorization header.

NA *reason*

The helper sends this message back to Squid when the user’s credentials are invalid. It also includes a “reason” string that Squid can display on an error page.

BH *reason*

The helper sends this message back to Squid when the validation procedure fails. This might happen, for example, when the helper process is unable to communicate with a Windows NT domain controller. Squid rejects the user’s request.

LD *username*

This helper-to-Squid response is similar to **BH**, except that Squid allows the user’s request. Like **AF**, it returns the username. To use this feature, you must compile Squid with the `--enable-ntlm-fail-open` option.

Since this protocol is relatively complicated, you’ll probably be better off to start with one of the two skeleton authenticators included in the Squid source distribution. The `no_check` helper is written in Perl, and `fakeauth` is written in C. You can find them in the `helpers/ntlm_auth` directory.

External ACLs

As of Version 2.5, Squid includes a new feature known as *external ACLs*. These are ACL elements that are implemented in external helper processes. You instruct Squid to write certain information to the helper, which then responds with either OK or ERR. Refer to Chapter 6 for a description of the *external_acl_type* syntax. Here, I'll only discuss the particular external ACL helper programs that come with the Squid source code.

ip_user

```
./configure --enable-external-acl-helpers=ip_user
```

This helper reads usernames and client IP addresses as input. It checks the two values against a configuration file to decide whether or not the combination is valid. To use this ACL helper, you would add lines like this to *squid.conf*:

```
external_acl_type ip_user_helper %SRC %LOGIN
    /usr/local/squid/libexec/ip_user -f /usr/local/squid/etc/ip_user.conf
acl AclName external ip_user_helper
```

%SRC is replaced with the client's IP address and %LOGIN is replaced with the username for each request. The *ip_user.conf* configuration file has the following format:

```
ip_addr[/mask]      user|@group|ALL|NONE
```

For example:

```
127.0.0.1           ALL
192.168.1.0/24      bob
10.8.1.0/24         @lusers
172.16.0.0/16       NONE
```

This configuration file causes *ip_user* to return OK for any request coming from 127.0.0.1, for Bob's requests coming from the 192.168.1.0/24 network, for any name in the *luser* group when the request comes from the 10.8.1.0/24 network, and returns ERR for any request from the 172.16.0.0/16 network. It also returns ERR for any address and username pair that doesn't appear in the list.

ldap_group

```
./configure --enable-external-acl-helpers=ldap_group
```

This helper determines whether or not a user belongs to a particular LDAP group. You specify the LDAP group names on the *acl* line. It might look like this in your configuration file:

```
external_acl_type ldap_group_helper %LOGIN /usr/local/squid/libexec/squid_ldap_group
    -b "ou=people,dc=example,dc=com" ldap.example.com
acl AclName external ldap_group_helper GroupRDN ...
```

Note that you must have the OpenLDAP (<http://www.openldap.org>) libraries installed on your system to compile the `squid_ldap_group` helper program.

unix_group

```
./configure --enable-external-acl-helpers=unix_group
```

This helper looks for usernames in the Unix group database (e.g., `/etc/group` file). You specify the groups to check on the helper command line as follows:

```
external_acl_type unix_group_helper %LOGIN  
    /usr/local/squid/libexec/check_group -g group1 -g group2 ...  
acl AclName external unix_group_helper
```

Alternatively, you can specify groups on the `acl` line. This allows you to use the same helper for different groups:

```
external_acl_type unix_group_helper %LOGIN /usr/local/squid/libexec/check_group  
acl AclName1 external unix_group_helper group1 ...  
acl AclName2 external unix_group_helper group2 ...
```

wbinfo_group

```
./configure --enable-external-acl-helpers=wbinfo_group
```

This helper is a short Perl script that utilizes the `wbinfo` program from the Samba package. `wbinfo` is a client for the `winbindd` daemon. The script expects a single Unix group name following the username on each request. Thus, you must put a group name on the `acl` line:

```
external_acl_type wbinfo_group_helper %LOGIN /usr/local/squid/libexec/wbinfo_group.pl  
acl AclName external wbinfo_group_helper group
```

winbind_group

```
./configure --enable-external-acl-helpers=winbind_group
```

This helper, written in C, also queries a `winbindd` server about group membership of Windows NT usernames. It is based on the `winbind` helpers for Basic and NTLM authentication. You can specify multiple group names on the `acl` command line:

```
external_acl_type winbind_group_helper %LOGIN /usr/local/squid/libexec/wb_check_group  
acl AclName external winbind_group_helper group1 group2 ...
```

Write Your Own

The external ACL interface offers a lot of flexibility. Chances are you can use it to implement almost any access control check not supported by the built-in methods. Writing an external ACL is a two-step process. First, you must decide what request information the helper program needs to make a decision. Place the appropriate

keywords on an *external_acl_type* line, along with the pathname to the helper program. For example, if you want to write an external ACL helper that uses the client's IP address, the user's name, and the value of the Host header, you would write something like:

```
external_acl_type MyAclHelper %SRC %LOGIN %{Host}  
/usr/local/squid/libexec/myaclhelper
```

The second step is to write the *myaclhelper* program. It must read the request tokens on stdin, make its decision, then write either OK or ERR to stdout. Continuing with the previous example, this Perl script illustrates how to do it:

```
#!/usr/bin/perl -wl  
require 'shellwords.pl';  
$|=1;  
while (<>) {  
    ($ip,$name,$host) = &shellwords;  
    if (&valid($ip,$name,$host)) {  
        print "OK";  
    } else {  
        print "ERR";  
    }  
}  
  
sub valid {  
    my $ip = shift;  
    my $name = shift;  
    my $host = shift;  
    ...  
}
```

Refer to Chapter 6 for the list of tokens (%SRC, %LOGIN, etc.) that you can pass from Squid to the helper. Note that when a token contains whitespace, Squid wraps it in double quotes. As the example shows, you can use Perl's *shellwords* library to parse quoted tokens easily.

Of course, to utilize the external ACL, you must reference it in an *acl* line. The ACL element is a match whenever the external helper returns OK.

The external ACL helper interface allows you to pass additional information from the helper to Squid (on the OK/ERR line). These take the form of *keyword=value* pairs. For example:

```
OK user=hank
```

Currently, the only keywords that Squid knows about are error and user. If the user value is set, Squid uses it in the *access.log*. The error value isn't currently used by Squid.

Exercises

- Write a fake helper for Basic authentication that always returns either OK or ERR.
- Use tcpdump or ethereal to capture some HTTP requests. Decode the authorization credentials.
- If you're using NTLM, capture some HTTP requests and attempt a replay attack.
- Kill Squid's authentication helper processes one-by-one while running **tail -f cache.log**.
- Find out what happens to your favorite NTLM-based authenticator when it can't communicate with the NT domain controller.

CHAPTER 13

Log Files

Log files are the primary sources of persistent information about Squid's operation. In other words, they provide a record of what Squid has been doing. This includes URIs requested by users, objects that have been saved to disk, and various warnings and errors. When Squid appears to be malfunctioning, you'll want to check the log files first. By the end of this chapter, you'll know how to interpret and manage all of Squid's various log files.

Depending on your configuration, Squid maintains, at most, seven log files. The three primary files are: *cache.log*, *access.log*, and *store.log*. Two optional log files, *useragent.log* and *referer.log*, are similar to *access.log* but contain additional information. I'll also talk about the *swap.state* and *netdb.state* files. These are databases, used by Squid when it restarts.

Note that the filenames, such as *access.log*, are the default values. You can change most of the log file names with various *squid.conf* directives.

The following list contains a brief description of each log file:

cache.log

This log file contains human-oriented, informational messages about Squid's operation. The filename is defined by the *cache_log* directive. Under normal conditions, the file grows by about 10–100 KB per day.

access.log

This log file contains an entry for every HTTP and (optionally) ICP transaction made by Squid's clients. The filename is defined by the *cache_access_log* directive. It grows at a rate of 100–200 bytes per transaction.

store.log

This log file contains low-level information about objects that enter and leave the cache. The filename is defined by the *cache_store_log* directive. It grows at a rate of about 150 bytes per transaction.

referer.log

This optional log file contains HTTP Referer* headers for each client request. You must enable referer logging with the `--enable-referer-log` option when running `./configure`. The filename is defined by the `referer_log` directive. It grows at a rate of about 80 bytes per transaction.

useragent.log

This optional log file contains HTTP User-Agent headers for each client request. You must enable user-agent logging with the `--enable-useragent-log` option when running `./configure`. The filename is defined by the `useragent_log` directive. It grows at a rate of about 75 bytes per transaction.

swap.state

These files contain internal metadata about the objects stored on disk. Squid uses them to reconstruct the cache upon startup. By default, they are located in the `cache_dir` directories. However, you can change the location with the `cache_swap_log` directive. They grow at a rate of 100 bytes per cache miss.

netdb_state

This file holds the contents of the Network Measurement Database (see Chapter 10). It is always located in the first `cache_dir` directory. Its size is determined by the `netdb_high` value.

If Squid receives an error while writing a log file, it doesn't silently continue. Instead, it exits with a fatal error message to get your attention. Make sure that you periodically rotate the log files, as described in the later section "Rotating the Log Files," to reduce the possibility of filling your disks. For the same reason, I also recommend placing your log files on a partition separate from any of your cache directories.

cache.log

cache.log contains various messages such as information about Squid's configuration, warnings about possible performance problems, and serious errors. Here is some sample *cache.log* output:

```
2003/09/29 12:09:45| Starting Squid Cache version 2.5.STABLE4 for i386-unknown-
freebsd4.8...
2003/09/29 12:09:45| Process ID 18990
2003/09/29 12:09:45| With 1064 file descriptors available
2003/09/29 12:09:45| Performing DNS Tests...
2003/09/29 12:09:45| Successful DNS name lookup tests...
2003/09/29 12:09:45| DNS Socket created at 0.0.0.0, port 1154, FD 5
2003/09/29 12:09:45| Adding nameserver 24.221.192.5 from /etc/resolv.conf
2003/09/29 12:09:45| Adding nameserver 24.221.208.5 from /etc/resolv.conf
2003/09/29 12:09:45| helperOpenServers: Starting 5 'redirector.pl' processes
```

* No, this isn't a typo. "Referer" has been historically misspelled by HTTP developers.

```

2003/09/29 12:09:45| Unlinkd pipe opened on FD 15
2003/09/29 12:09:45| Swap maxSize 10240 KB, estimated 787 objects
2003/09/29 12:09:45| Target number of buckets: 39
2003/09/29 12:09:45| Using 8192 Store buckets
2003/09/29 12:09:45| Max Mem size: 8192 KB
2003/09/29 12:09:45| Max Swap size: 10240 KB
2003/09/29 12:09:45| Rebuilding storage in /usr/local/squid/var/cache (CLEAN)
2003/09/29 12:09:45| Using Least Load store dir selection
2003/09/29 12:09:45| Set Current Directory to /usr/local/squid/var/cache
2003/09/29 12:09:45| Loaded Icons.
2003/09/29 12:09:45| Accepting HTTP connections at 0.0.0.0, port 3128, FD 16.
2003/09/29 12:09:45| Accepting ICP messages at 0.0.0.0, port 3130, FD 17.
2003/09/29 12:09:45| WCCP Disabled.
2003/09/29 12:09:45| Ready to serve requests.

```

Each *cache.log* entry starts with a timestamp showing when the message was generated. The very first entry in this sample reports the Squid version (2.5.STABLE4) and a string identifying the operating system for which Squid was configured (i386-unknown-freebsd4.8). The process ID (18990) follows. Many *cache.log* entries may look cryptic (Target number of buckets: 39). In most cases, under normal conditions, you can ignore entries you don't understand. On the other hand, you may want to look over essential configuration details such as name-server addresses or HTTP server address. This sample output ends with a statement that Squid is ready to serve requests. At this point, Squid can accept HTTP connections from clients.

Usually, the *cache.log* file grows slowly. However, an unusual HTTP transaction or similar event may cause Squid to emit a debugging message. If such an event happens often (e.g., a DoS attack, a new virus, or sudden disk failure), the log file may grow quickly. Rotating log files reduces the chance that you'll run out of disk space.

Major errors and abnormal conditions are likely to be reported in *cache.log*. I recommend archiving these logs so that it is possible to go back and find the first occurrence of an unusual event. When describing a particular Squid problem on the mailing list or a similar forum, the relevant lines from *cache.log* may be very useful. You may also want to increase debugging levels for some sections so that others can better understand and fix your problem.

Debugging Levels

The *debug_options* directive controls the level of detail for *cache.log* messages. The default value (ALL,1) is usually the best choice. At higher levels, the unimportant messages make it harder to find the important ones. Refer to Chapter 16 for a thorough description of the *debug_options* directive.

Note that debugging at the highest levels (9 or 10) may add thousands of lines for each request, quickly consuming disk space and significantly degrading Squid's performance.

You can use Squid's `-X` command-line option to enable full debugging for all sections. This mode is particularly useful if Squid refuses to start, and the debugging levels in *squid.conf* are insufficient to diagnose the problem. This is also a good way to enable full debugging of the configuration file parser, before it gets to the *debug_options* directive. You should never use the `-X` when Squid is operating properly.

You can use Squid's `-k debug` command-line option to enable full debugging immediately on a running Squid process. This command operates as a toggle: the first invocation turns on full debugging, and the second invocation turns it off. See Chapter 5 for a general discussion about the `-k` option.

As I already mentioned, full debugging generates an overwhelming amount of data. This can make Squid, and the operating system, very slow. In extreme cases, you may find your terminal session becomes unresponsive after executing the first `squid -k debug` command. Locking yourself out while Squid is spitting megabytes of *cache.log* entries per second is an unpleasant experience. I find the following trick useful to get a compact, five-second debugging snapshot with less risk:

```
% squid -k debug; sleep 5; squid -k debug
```

Forwarding *cache.log* Messages to the System Log

To have Squid send copies of *cache.log* messages to the system log, use the `-s` command-line option. Only messages with debugging levels 0 and 1 are forwarded. Level 0 messages are logged with syslog level `LOG_WARNING`. Level 1 messages use syslog level `LOG_NOTICE`. All messages use the `LOCAL4` syslog facility. Here is one way to configure `syslogd` so that these messages are saved:

```
local4.warning                                /var/log/squid.log
```

Using syslog in addition to *cache.log* is especially handy when you maintain several Squid boxes. You can configure each local syslog daemon to forward these messages to a central host and enjoy a unified view of all caches in one location. For example, you might use this entry in */etc/syslogd.conf*:

```
local4.notice                                @192.168.45.1
```

Dumping *cache.log* Messages to Your Terminal

The `-d` level command-line option instructs Squid to dump *cache.log* messages to your terminal (i.e., *stderr*). The level argument specifies the maximum level for messages that are dumped. Note that you'll see only messages that would appear in *cache.log*, subject to the *debug_options* setting. For example, if you have *debug_options ALL,1*, and run `squid -d2`, you won't see any level 2 debugging messages.

The `-d` level and `-N` options are most useful for debugging Squid problems or quickly testing a change to the configuration file. They allow you to start Squid easily and see the *cache.log* messages. This option may also be useful when Squid starts

from cron or a similar facility that automatically captures a program's standard error output and reports it back to the user. For example, you may have a cron job that automatically reconfigures the running Squid process:

```
15 */4 * * * /usr/local/squid/sbin/squid -d1 -k reconfigure
```

access.log

Squid saves key information about HTTP transactions in *access.log*. This file is line-based, such that each line corresponds to one client request. Squid records the client IP address (or hostname), requested URI, response size, and other information.

Squid records all HTTP accesses in *access.log*, except for those that disconnect before sending any data. Squid also records all ICP (but not HTCP) transactions unless you disable them with the *log_icp_queries* directive. The later section “Configuration Directives That Affect access.log” describes the other *squid.conf* directives that affect the access log.

The default *access.log* format contains 10 fields. Here are some examples, with long lines split and indented:

```
1066037222.011 126389 9.121.105.207 TCP_MISS/503 1055
    GET http://home.gigigaga.com/n8342133/Miho.DAT.019 -
    DIRECT/203.187.1.180 -
1066037222.011 19120 12.83.179.11 TCP_MISS/200 359
    GET http://ads.x10.com/720x300/Z2FtZ3JlZXRpbnmcxLmRhZ/7/AMG -
    DIRECT/63.211.210.20 text/html
1066037222.011 34173 166.181.33.71 TCP_MISS/200 559
    GET http://coursesites.blackboard.com:8081/service/collab/./1010706448190/ -
    DIRECT/216.200.107.101 application/octet-stream
1066037222.011 19287 41.51.105.27 TCP_REFRESH_MISS/200 500
    GET http://fn.yam.com/include/tsemark/show.js -
    DIRECT/210.59.224.59 application/x-javascript
1066037222.011 19395 41.51.105.27 TCP_MISS/304 274
    GET http://fnasp.yam.com/image/coin3.gif -
    DIRECT/211.72.254.133 -
1066037222.011 19074 30.208.85.76 TCP_CLIENT_REFRESH_MISS/304 197
    GET http://ads.icq.com/content/BO/0/..bC6GygEYNeHGjBUin5Azfe68m5hD1jLk$/aol -
    DIRECT/64.12.184.121 -
1066037222.011 19048 12.83.179.11 TCP_MISS/200 261
    GET http://ads.adsag.com/js.ng/...ne&cat=friendship&subcat=girltalk -
    DIRECT/209.225.54.119 application/x-javascript
1066037222.118 106 41.51.105.27 TCP_HIT/200 536
    GET http://rcm-images.amazon.com/images/G/01/rcm/privacy.gif -
    NONE/- image/gif
1066037222.352 19475 27.34.49.248 TCP_MISS/200 12387
    GET http://espanol.geocities.com/lebastias/divulgacion/budismo-tarot.html -
    DIRECT/209.1.225.139 text/html
1066037222.352 132 144.157.100.17 TCP_MISS/504 1293
    GET http://ar.atwola.com/image/93101912/aol -
    NONE/- -
```

Here are the definitions for all fields:

1: timestamp

The completion time of the request, expressed as the number of seconds since the Unix epoch (Thu Jan 1 00:00:00 UTC 1970), with millisecond resolution. Squid uses this format, instead of something more human-friendly, to simplify the work of various log file processing programs.

You can use a simple Perl command to convert the Unix timestamps into local time. For example:

```
perl -pe 's/^\d+\.\d+/localtime($&)/e;' access.log
```

2: response time

For HTTP transactions, this field indicates how much time it took to process the request. The timer starts when Squid receives the HTTP request and stops when the response has been fully delivered. The response time is given in milliseconds.

The response time is usually 0 for ICP queries. This is because Squid answers ICP queries very quickly. Furthermore, Squid doesn't update the process clock between receiving an ICP query and sending the reply.

While time values are reported with millisecond resolution, the precision of those entries is probably about 10 milliseconds. Timing becomes even less precise when Squid is heavily loaded.

3: client address

This field contains the client's IP address, or hostname if you enable *log_fqdn*. For security or privacy reasons, you may want to mask a part of client's address out using the *client_netmask* directive. However, that also makes it impossible to group requests coming from the same client.

4: result/status codes

This field consists of two tokens separated by a slash. The first token, result code, classifies the protocol and the result of a transaction (e.g., TCP_HIT or UDP_DENIED). These are Squid-specific codes, defined in the later section "access.log Result Codes." The codes that begin with TCP_ refer to HTTP requests, while UDP_ refers to ICP queries.

The second token is the HTTP response status code (e.g, 200, 304, 404, etc.). The status code normally comes from the origin server. In some cases, however, Squid may be responsible for selecting the status code. These codes, defined by the HTTP RFC, are summarized later in Table 13-1.

5: transfer size

This field indicates the number of bytes transferred to the client. Strictly speaking, it is the number of bytes that Squid told the TCP/IP stack to send to the client. Thus, it doesn't include overheads from TCP/IP headers. Also note that the transfer size is normally larger than the response's Content-Length. This value includes the HTTP response headers, while Content-Length does not.

These properties make the transfer size field useful for approximate bandwidth usage analysis but not for exact HTTP entity size calculations. If you need to know a response's Content-Length, you can find it in the *store.log* file.

6: *request method*

This field contains the request method. Because Squid clients may use ICP or HTTP, the request method is either HTTP- or ICP-specific. The most common HTTP request method is GET. ICP queries are always logged with ICP_QUERY. See Chapter 6 for a list of HTTP methods Squid knows about.

7: *URI*

This field contains the URI from the client's request. The vast majority of logged URIs are actually URLs (i.e., they have hostnames).

Squid uses a special format for certain failures. These are cases when Squid can't parse the HTTP request or otherwise determine the URI. Instead of a URI/URL, you'll see a string such as "error:invalid-request." For example:

```
1066036250.603 310 192.0.34.70 NONE/400 1203 GET error:invalid-request - NONE/- -
```

Also in this field look out for whitespace characters in the URI. Depending on your *uri_whitespace* setting, Squid may print the URI in the log file with whitespace characters. When this happens, the tools that read *access.log* files may become confused by the extra fields.

When logging, Squid strips all URI characters after the first question mark unless the *strip_query_terms* directive is disabled.

8: *client identity*

Squid can determine a user's identity in two different ways. One is with the RFC 1413 ident protocol; the other is from HTTP authentication headers.

Squid attempts ident lookups based on the *ident_lookup_access* rules, if any (see Chapter 6). Alternatively, if you use proxy authentication (or regular server authentication in surrogate mode), Squid places the given username in this field. If both methods provide Squid with a username, and you're using the native *access.log* format, the HTTP authentication name is logged, and the RFC 1413 name is ignored. The common log file format has separate fields for both names.

9: *peering code/peerhost*

The peering information consists of two tokens, separated by a slash. It is relevant only for requests that are cache misses. The first token indicates how the next hop was chosen. The second token is the address of that next hop. The peering codes are listed in the "access.log Peering Codes" section.

When Squid sends a request to a neighbor cache, the peerhost address is the neighbor's hostname. If the request is sent directly to the origin server, however, Squid writes the origin server's IP address or its hostname if *log_ip_on_direct* is disabled. The value NONE/- indicates that Squid didn't forward this request to any other servers.

10: content type

The final field of the default, native *access.log* is the content type of the HTTP response. Squid obtains the content type value from the response's Content-Type header. If that header is missing, Squid uses a hyphen (-).

If you enable the *log_mime_headers* directive, Squid appends two additional fields to each line:

11: HTTP request headers

Squid encodes the HTTP request headers and prints them between a pair of square brackets. The brackets are necessary because Squid doesn't encode space characters. The encoding scheme is a little strange. Carriage return (ASCII 13) and newline (ASCII 10) are printed as `\r` and `\n`, respectively. Other non-printable characters are encoded with the RFC 1738 style, such that Tab (ASCII 9) becomes `%09`.

12: HTTP response headers

Squid encodes the HTTP response headers and prints them between a pair of square brackets. Note that these are the headers sent to the client, which may be different from headers received from the origin server.

Squid writes to *access.log* only after the entire response has been sent to the client. This allows Squid to include both request and response information in the log file. However, transactions that take minutes, or even hours, to complete aren't visible in *access.log* at the time of the request. When these types of transactions present a performance or policy concern, the *access.log* may be unable help you. Instead, use the cache manager to view a list of pending transactions (see Chapter 14).

access.log Result Codes

The following labels may appear in the fourth field of the *access.log* file in response to HTTP requests:

TCP_HIT

Squid found a likely fresh copy of the requested resource and sent it immediately to the client.

TCP_MISS

Squid didn't have a cached copy of the requested resource.

TCP_REFRESH_HIT

Squid found a likely stale copy of the requested resource and sent a validation request to the origin server. The origin server sent a 304 (Not Modified) response, indicating that Squid's copy is still fresh.

TCP_REF_FAIL_HIT

Squid found a likely stale copy of the requested resource and sent a validation request to the origin server. However, the origin server failed to respond or sent

a response that Squid didn't understand. In any case, Squid sent the cached (and likely stale) copy to the client.

TCP_REFRESH_MISS

Squid found a likely stale copy of the requested resource and sent a validation request to the origin server. The server responded with new content, indicating the cached response was indeed stale.

TCP_CLIENT_REFRESH_MISS

Squid found a copy of the requested resource, but the client's request included a Cache-Control: no-cache directive. Squid forwarded the client's request to the origin server, forcing a cache validation.

TCP_IMS_HIT

The client sent a validation request, and Squid found a more recent, and likely fresh, copy of the requested resource. Squid sent the newer content to the client, without contacting the origin server.

TCP_SWAPFAIL_MISS

Squid found a valid copy of the requested resource but failed to load it from disk. Squid then sent the request to the origin server as though it were a cache miss.

TCP_NEGATIVE_HIT

When a request to an origin server results in an HTTP error, Squid may cache the response anyway. Repeated requests for these resources, within a short amount of time, result in negative hits. The *negative_ttl* directive controls the amount of time these errors may be cached. Also note that errors are cached only in memory and never written to disk. The following HTTP status codes may be negatively cached, subject to additional constraints: 204, 305, 400, 403, 404, 405, 414, 500, 501, 502, 503, 504.

TCP_MEM_HIT

Squid found a valid copy of the requested resource in the memory cache and sent it immediately to the client. Note that this doesn't accurately represent all responses served from memory. For example, responses that are cached in memory, but require validation, are logged with TCP_REFRESH_HIT, TCP_REFRESH_MISS, etc.

TCP_DENIED

The client's request was denied, due to either the *http_access* or *http_reply_access* rules. Note that requests denied by *http_access* have NONE/- in the ninth field, whereas those denied by *http_reply_access* have a valid entry.

TCP_OFFLINE_HIT

When *offline_mode* is enabled, Squid returns cache hits for almost any cached response, without considering its freshness.

TCP_REDIRECT

A redirector program told Squid to generate an HTTP redirect to a new URI (see Chapter 11). Normally, Squid doesn't log these redirects. To do so, you must manually define the `LOG_TCP_REDIRECTS` preprocessor directive before compiling Squid.

NONE

Unclassified result used for certain errors, such as invalid hostnames.

The following labels may appear in the fourth field of the *access.log* file in response to ICP queries:

UDP_HIT

Squid found a likely fresh copy of the requested resource in the cache.

UDP_MISS

Squid didn't find a likely fresh copy of the requested resource in the cache. If the same object is requested via HTTP, it would probably be a cache miss. Compare with `UDP_MISS_NOFETCH`.

UDP_MISS_NOFETCH

Like `UDP_MISS`, except that this also indicates Squid's reluctance to handle the corresponding HTTP request. If you use the `-Y` command-line option, Squid returns this, instead of `UDP_MISS`, while rebuilding its in-memory indexes at startup.

UDP_DENIED

The ICP query is denied due to the *icp_access* rules. If more than 95% of the ICP replies to a client are `UDP_DENIED`, and the client database is enabled (see Appendix A), Squid stops sending any ICP replies to the client for an hour. When this happens you'll also see a warning in *cache.log*.

UDP_INVALID

Squid received an invalid query (e.g., truncated message, invalid protocol version, whitespace in the URI, etc.). Squid sent an `ICP_INVALID` reply back to the client.

HTTP Response Status Codes

Table 13-1 lists the numerical HTTP response codes and reason phrases. Note that Squid and other HTTP agents care only about the numeric value. The reason phrase is purely informational and doesn't affect the meaning of the response. For each status code, I also provide a reference to the particular section in RFC 2616 that describes it. Note that status codes 0 and 600 are nonstandard values used by Squid, and aren't mentioned in the RFC.

Table 13-1. HTTP response status codes

Code	Reason phrase	RFC 2616 section
0	No Response Received (Squid-specific)	N/A
1xx	Informational	10.1

Table 13-1. HTTP response status codes (continued)

Code	Reason phrase	RFC 2616 section
100	Continue	10.1.1
101	Switching Protocols	10.1.2
2xx	Successful	10.2
200	OK	10.2.1
201	Created	10.2.2
202	Accepted	10.2.3
203	Non-Authoritative Information	10.2.4
204	No Content	10.2.5
205	Reset Content	10.2.6
206	Partial Content	10.2.7
3xx	Redirection	10.3
300	Multiple Choices	10.3.1
301	Moved Permanently	10.3.2
302	Found	10.3.3
303	See Other	10.3.4
304	Not Modified	10.3.5
305	Use Proxy	10.3.6
306	(Unused)	10.3.7
307	Temporary Redirect	10.3.8
4xx	Client Error	10.4
400	Bad Request	10.4.1
401	Unauthorized	10.4.2
402	Payment Required	10.4.3
403	Forbidden	10.4.4
404	Not Found	10.4.5
405	Method Not Allowed	10.4.6
406	Not Acceptable	10.4.7
407	Proxy Authentication Required	10.4.8
408	Request Timeout	10.4.9
409	Conflict	10.4.10
410	Gone	10.4.11
411	Length Required	10.4.12
412	Precondition Failed	10.4.13
413	Request Entity Too Large	10.4.14
414	Request-URI Too Long	10.4.15
415	Unsupported Media Type	10.4.16

Table 13-1. HTTP response status codes (continued)

Code	Reason phrase	RFC 2616 section
416	Requested Range Not Satisfiable	10.4.17
417	Expectation Failed	10.4.18
5xx	Server Error	10.5
500	Internal Server Error	10.5.1
501	Not Implemented	10.5.2
502	Bad Gateway	10.5.3
503	Service Unavailable	10.5.4
504	Gateway Timeout	10.5.5
505	HTTP Version Not Supported	10.5.6
6xx	Proxy Error	N/A
600	Unparseable Response Headers (Squid-specific)	N/A

You'll see status code 0 in the *access.log* if Squid doesn't receive any response from the origin server. You'll see status code 600 if Squid received a response but couldn't find any HTTP headers. In a small fraction of cases, certain origin servers send only the response body and omit any headers.

access.log Peering Codes

The following codes may appear in the ninth field of the *access.log*. Refer to Chapter 10 for a description of how Squid selects the next-hop for cache misses.

NONE

This indicates that Squid didn't communicate with any other servers (neighbors, origin) for this request. You'll see it in association with various types of cache hits, denied requests, cache manager requests, errors, and all ICP queries.

DIRECT

Squid forwarded the request directly to the origin server. The second half of the field shows the origin server's IP address, or hostname if you've disabled *log_ip_on_direct*.

SIBLING_HIT

Squid sent the request to this sibling cache after the sibling returned an ICP or HTCP hit.

PARENT_HIT

Squid sent the request to this parent cache after the parent returned an ICP or HTCP hit.

DEFAULT_PARENT

Squid selected this parent because it was marked as default on the *cache_peer* line in *squid.conf*.

FIRST_UP_PARENT

Squid forwarded the request to this parent because it is the first parent in the list known to be alive.

FIRST_PARENT_MISS

Squid forwarded the request to the parent cache that was first to respond with an ICP/HTCP miss message. In other words, for this particular ICP/HTCP query, at this particular time, the selected parent had the best round-trip time. Note that measured RTTs may be artificially adjusted by the *weight* option to the *cache_peer* directive.

CLOSEST_PARENT_MISS

Squid selected this parent because it reports the lowest RTT to the origin server. This occurs only if both caches have *netdb* enabled (see Chapter 10), and the origin server (or other servers on its subnet) returns ICMP pings.

CLOSEST_PARENT

This is similar to *CLOSEST_PARENT_MISS*, except that the RTT measurements don't come from the ICP/HTCP reply messages. Instead, they come from older measurements saved by Squid, such as the *netdb* exchange feature.

CLOSEST_DIRECT

Squid forwarded the request to the origin server based on *netdb* measurements. This happens if any of these conditions occur:

- The RTT between Squid and the origin server is less than the configured *minimum_direct_rtt* value.
- The measured number of router hops between Squid and the origin server is less than the configured *minimum_direct_hops* value.
- The RTT values returned in ICP/HTCP replies indicate that Squid is closer to the origin server than any of its neighbors.

ROUNDROBIN_PARENT

Squid forwarded the request to this parent because the round-robin option was set, and it had the lowest usage counter.

CD_PARENT_HIT

Squid forwarded the request to this parent based on the Cache Digest algorithm (see Chapter 10).

CD_SIBLING_HIT

Squid forwarded the request to this sibling based on the Cache Digest algorithm.

CARP

Squid selected this parent based on the Cache Array Routing Protocol algorithm (see Chapter 10).

ANY_PARENT

Squid selected this parent as a last resort because none of the other methods resulted in a viable next-hop.

Note that most of these codes may be preceded by `TIMEOUT_` to indicate that a timeout occurred while waiting for ICP/HTCP replies. For example:

```
1066038165.382    345 193.233.46.21 TCP_MISS/200 2836
GET http://www.caida.org/home/images/home.jpg
TIMEOUT_CLOSEST_DIRECT/213.219.122.19 image/jpeg
```

You can adjust the timeout with the `icp_query_timeout` directive.

Configuration Directives That Affect `access.log`

Following are the configuration file directives that affect the `access.log` in one way or another.

`log_icp_queries`

This directive, enabled by default, causes Squid to log all ICP queries. If you're running a busy parent cache, this may make your `access.log` files huge. To save space, disable this directive:

```
log_icp_queries off
```

If you disable ICP query logging, I suggest that you monitor the number of queries, either through the cache manager or with SNMP.

`emulate_httpd_log`

The `access.log` file has two formats: common and native. The common format is the same as most HTTP servers (e.g., Apache) use. It contains less information than Squid's native format. However, you might want to use the common log-file format if you use Squid as a surrogate (see Chapter 15). The common format may also be useful if you have log-file analysis tools that know how to parse it. Use this directive to enable the common format:

```
emulate_httpd_log on
```

See the site <http://www.w3.org/Daemon/User/Config/Logging.html#common-logfile-format>, for a description of this format.

`log_mime_hdrs`

Use the `log_mime_hdrs` directive to make Squid log the HTTP request and response headers:

```
log_mime_headers on
```

When enabled, Squid appends the request and response headers to `access.log`. This adds two fields to each line. Each field is surrounded by square brackets to make parsing easier. Certain characters are encoded to keep the log file readable. Table 13-2 shows the encoding scheme.

Table 13-2. Character encoding rules for HTTP headers in *access.log*

Character	Encoding
Newline	\n
Carriage return	\r
Backslash	\\
[	%5b
]	%5d
%	%25
ASCII 0–31	%xx (hexadecimal value)
ASCII 127–255	%xx (hexadecimal value)

log_fqdn

By default, Squid puts client IP addresses in the *access.log*. You can record hostnames, when available, by enabling this directive:

```
log_fqdn on
```

This causes Squid to make reverse DNS lookups for the client’s address when it receives a request. If an answer is available by the time the request is complete, Squid places it in the third field.

ident_lookup_access

This access rule list determines whether or not Squid makes an RFC 1413 ident query for the client’s TCP connection. By default, Squid doesn’t issue ident queries. To enable this feature, simply add one or more rules:

```
acl All src 0/0
ident_lookup_access allow All
```

If an answer is available by the time the request is complete, Squid places it in the eighth field. If you are also using HTTP authentication, that username is written instead of the ident answer.

log_ip_on_direct

When Squid forwards a cache miss to an origin server, it records the origin server’s IP address in the ninth field. You can disable this directive so that Squid writes the hostname instead:

```
log_ip_on_direct off
```

In this case, the hostname comes from the URI. If the URI contains an IP address, Squid doesn’t convert it to a hostname.

client_netmask

This directive exists to provide some level of privacy for your users. Rather than logging the entire client IP address, you can mask off some bits. For example:

```
client_netmask 255.255.255.0
```

With this setting, all client IP addresses in *access.log* have 0 as the last octet:

```
1066036246.918    35 163.11.255.0 TCP_IMS_HIT/304 266 GET http://...
1066036246.932    16 163.11.255.0 TCP_IMS_HIT/304 266 GET http://...
1066036247.616    313 140.132.252.0 TCP_MISS/200 1079 GET http://...
1066036248.598    44459 140.132.252.0 TCP_MISS/500 1531 GET http://...
1066036249.230    17 170.210.173.0 TCP_IMS_HIT/304 265 GET http://...
1066036249.752    2135 140.132.252.0 TCP_MISS/200 50230 GET http://...
1066036250.467     4 170.210.173.0 TCP_IMS_HIT/304 265 GET http://...
1066036250.762    102 163.11.255.0 TCP_IMS_HIT/304 265 GET http://...
1066036250.832    20 163.11.255.0 TCP_IMS_HIT/304 266 GET http://...
1066036251.026    74 203.91.150.0 TCP_CLIENT_REFRESH_MISS/304 267 GET http://...
```

strip_query_terms

This directive is another privacy feature. It removes query terms from URIs before logging them. If your log files somehow fall into the wrong hands, they won't be able to find any usernames and passwords. When this directive is enabled, all characters after a question mark (?) are removed. For example, a URI like this:

```
http://auto.search.msn.com/response.asp?MT=www.kimo.com.yw&srch=3&prov=&utf8
```

is logged like this:

```
http://auto.search.msn.com/response.asp?
```

uri_whitespace

Earlier, I mentioned the problem with whitespace appearing in some URIs. The RFCs state that URIs must not contain whitespace, but in reality it happens all too often. The *uri_whitespace* directive dictates how Squid should handle such cases. The allowed settings are: strip (default), deny, allow, encode, and chop. Of these, strip, encode, and chop ensure that the URI field doesn't contain any whitespace (thus adding more fields to *access.log*).

The allow setting allows the request to pass through Squid unmodified. It is likely to cause trouble for redirectors and log file parsers. The deny setting, on the other hand, causes Squid to deny the request. The user receives an error message, but the request is still written to *access.log* with the whitespace characters.

If you set it to encode, Squid changes the whitespace characters to their RFC 1738 equivalents. This is probably what the user-agent should have done in the first place. The chop setting causes Squid to cut off the URI at the first whitespace character.

The default setting is `strip`, which makes Squid remove the whitespace characters from the URI. It ensures that your log-file parsers and redirectors will be happy, but it might break certain things, such as improperly encoded search engine queries.

buffered_logs

By default, Squid disables buffering for the *cache.log* file, which allows you to run `tail -f` and watch log file entries appear in real time. If you think this will cause an unnecessary overhead, you can disable buffering:

```
buffered_logs off
```

However, it probably doesn't matter unless you are running Squid with full debugging. Note that this option affects only *cache.log*. The others always use unbuffered writes.

access.log Analysis Tools

The *access.log* file contains a wealth of information, much more than you can see by just browsing through it. In order to get the big picture view, you'll need to use a third-party log-file analysis package. You can find a long list of them linked from the Squid web page, or by going directly to <http://www.squid-cache.org/Scripts/>.

One of the most popular tools is Calamaris—a Perl script that parses the log file and generates either text or HTML-based reports. It provides a breakdown of traffic by request method, client IP address, origin server domain name, content types, file-name extensions, reply size, and more. Calamaris also reports on ICP query traffic and even understands log files from other caching products. Check it out by visiting <http://calamaris.cord.de/>.

Squeezer, and its derivative, Squeezer2, are Squid-specific analysis tools. They provide many statistics that can help you understand Squid's performance, especially when you have neighbors. Both generate HTML pages as output. Visit the *Logfile Analysis* page on the [squid-cache.org](http://www.squid-cache.org) site for links to these programs.

Webalyzer is another good utility. It is designed to be fast and produces HTML pages with tables and bar charts. It was originally designed for origin server access logs. Although it can parse Squid's logs, it doesn't report on such things as hit ratios and response times. It also uses some terms differently than I do. For example, Webalyzer calls any request a "hit," which isn't the same as a cache hit. It also makes a distinction between "pages" and "files." For more information, visit the Webalyzer home page at <http://www.mrunix.net/webalyzer/>.

store.log

The *store.log* is a record of Squid's decisions to store and remove objects from the cache. Squid creates an entry for each object it stores in the cache, each uncachable

object, and each object that is removed by the replacement policy. The log file covers both in-memory and on-disk caches.

The *store.log* provides the following you can't get from *access.log*:

- Whether or not a particular response was cached.
- The file number for cached objects. For UFS-based storage schemes, you can convert this to a pathname and examine the contents of the cache file.
- The response's content length: the Content-Length value, and the actual body length.
- Values for the Date, Last-Modified, and Expires headers.
- The response's cache key (i.e., MD5 hash value).

As you can see, this is mostly low-level information you won't need on a daily basis. Unless you do sophisticated analyses, or wish to debug a problem, you can probably get by without the *store.log*. You can disable it with a special setting:

```
cache_store_log none
```

As with other log files, Squid appends new *store.log* entries to the end of the file. A given URI may appear in the file more than once. For example, it gets cached, then released, then cached again. Only the most recent entry reflects the object's current status.

The *store.log* is text-based and looks something like this:

```
1067299212.411 RELEASE -1 FFFFFFFF A5964B32245AC98592D83F9B6EA10B8D 206
    1067299212 1064287906 -1 application/octet-stream 6840/6840
    GET http://download.windowsupdate.com/msdownload/update/v3-19990518/cab...
1067299212.422 SWAPOUT 02 0005FD5F 6F34570785CACABC8DD01ABA5D73B392 200
    1067299210 1057899600 -1 image/gif 1125/1125
    GET http://forum.topsportsnet.com/shfimages/nav_members1.gif
1067299212.641 RELEASE -1 FFFFFFFF B0616CB4B7280F67672A40647DD08474 200
    1067299212 -1 -1 text/html -1/67191
    GET http://www.tlava.com/
1067299212.671 RELEASE -1 FFFFFFFF 5ECD93934257594825659B596D9444BC 200
    1067299023 1034873897 1067299023 image/jpeg 3386/3386
    GET http://ebizo.ipixmedia.com/abc/ebiz/_EBIZ_3922eabf57d44e2a4c3e7cd234a...
1067299212.786 RELEASE -1 FFFFFFFF B388F7B766B307ADEC044A4099946A21 200
    1067297755 -1 -1 text/html -1/566
    GET http://www.evenflowrocks.com/pages/100303pic15.cfm
1067299212.837 RELEASE -1 FFFFFFFF ABC862C7107F3B7E9FC2D7CA01C8E6A1 304
    1067299212 -1 1067299212 unknown -1/0
    GET http://ebizo.ipixmedia.com/abc/ebiz/_EBIZ_3922eabf57d44e2a4c3e7cd234a...
1067299212.859 RELEASE -1 FFFFFFFF 5ED2726D4A3AD83CACC8A01CFDD6082B 304
    1066940882 1065063803 -1 application/x-javascript -1/0
    GET http://www.bellsouth.com/scripts/header_footer.js
```

Each entry contains the following 13 fields:

1: timestamp

The timestamp when the event took place, expressed as seconds since the Unix epoch with millisecond resolution.

2: action

The action taken on the object. This field has three possible values: SWAPOUT, RELEASE, and SO_FAIL.

- A SWAPOUT occurs when Squid successfully completes saving the object data to disk. Some objects, such as those that are negatively cached, are kept in memory, but not on disk. Squid doesn't make a *store.log* entry for them.
- A SO_FAIL entry indicates that Squid could not completely save the object to disk. Most likely it means that the storage scheme implementation refused to open a new disk file for writing.
- A RELEASE occurs when Squid removes an object from the cache, or decides that the response isn't cachable in the first place.

3: directory number

The directory number is a 7-bit index to the list of cache directories that's written as a decimal number. For objects that aren't saved to disk, this field contains the value -1.

4: file number

The file number is a 25-bit identifier used internally by Squid. It is written as an 8-character hexadecimal number. The UFS-based storage schemes have an algorithm for mapping file numbers to pathnames (see the later section "Mapping File Numbers to Pathnames").

Objects that aren't saved to disk don't have a valid file number. For these, the file number field contains FFFFFFFF. This value appears only for RELEASE and SO_FAIL entries.

5: cache key

Squid uses MD5 hash values for the primary index to locate cached objects. The key is based on the request method, URI, and possibly other information.

You might be able to use the cache key to match up *store.log* entries. Note, however, that an object's cache key can change. This happens, for example, whenever Squid logs a TCP_REFRESH_MISS request in *access.log*. It looks like this:

```
1065837334.045 SWAPOUT ... 554BACBD2CB2A0C38FF9BF4B2239A9E5 ... http://blah
1066031047.925 RELEASE ... 92AE17121926106EB12FA8054064CABA ... http://blah
1066031048.074 SWAPOUT ... 554BACBD2CB2A0C38FF9BF4B2239A9E5 ... http://blah
```

So what's going on? The object is originally cached under one key (554B...). Some time later, Squid receives another request for the object and forwards a validation request to the origin server. When the response comes back with new content, Squid changes the cache key of the old object (to 92AE...) so that it can give the new object the correct key (554B...). The old object is then removed, and the new object is saved to disk.

6: *status code*

This field shows the HTTP status code of the response, just like *access.log*. See Table 13-1 for a list of status codes.

7: *date*

The value of the Date header in the HTTP response, expressed as seconds since the Unix epoch. The value -1 indicates an unparseable Date header, and -2 means the header was entirely absent.

8: *last-modified*

The value of the Last-Modified header in the HTTP response, expressed as seconds since the Unix epoch. The value -1 indicates an unparseable Last-Modified header, and -2 means the header was entirely absent.

9: *expires*

The value of the Expires header in the HTTP response, expressed as seconds since the Unix epoch. The value -1 indicates an unparseable Expires header, and -2 means the header was entirely absent.

10: *content-type*

The value of the Content-Type header in the HTTP response, excluding any media-type parameters. Squid inserts the value unknown if the Content-Type is missing.

11: *content-length/size*

This field contains two numbers, separated by a slash. The first is the value of the Content-Length header. A -1 indicates the Content-Length header is absent. The second is the actual size of the HTTP message body. You can use these two numbers to identify partially received responses and origin servers that incorrectly calculate the content length. In most cases, the two numbers are the same.

12: *method*

The HTTP request method for the object, as in *access.log*.

13: *URI*

The final field is the requested URI, as in *access.log*. This field also has the whitespace problem mentioned in the previous section. However, it is less worrisome here because you can safely ignore any extra fields.

For many of the RELEASE entries, you'll see question marks (?) for the last eight fields. This is because most of those field values come from what Squid calls the MemObject structure. This structure is present only for objects that have just been received, or are being stored entirely in memory. Most of the objects in Squid's cache don't have a MemObject because they exist only on disk. For these, Squid puts question marks in the fields with missing information.

Mapping File Numbers to Pathnames

If you find you need to examine a particular cache file, you can, with some effort, turn a file number into a pathname. You'll also need the directory number, and L1 and L2 values. In the Squid source code, the `storeUfsDirFullPath()` function does this. You can find it in the `src/fts/ufs/store_dir_ufs.c` file. This short Perl program mimics the current algorithm:

```
#!/usr/bin/perl
$L1 = 16;
$L2 = 256;
while (<>) {
    $filn = hex($_);
    printf("%02X/%02X/%08X\n",
        (($filn / $L2) / $L2) % $L1,
        ($filn / $L2) % $L2,
        $filn);
}
```

And here's how you can use it:

```
% echo 000DCD06 | ./fileno-to-pathname.pl
0D/CD/000DCD06
```

To find this file in the *N*th `cache_dir`, simply go to the corresponding directory and list or view the file:

```
% cd /cache2
% ls -l 0D/CD/000DCD06
-rw----- 1 squid squid 391 Jun  3 12:40 0D/CD/000DCD06
% less 0D/CD/000DCD06
```

referer.log

The optional `referer.log` contains Referer header values from client requests. To use this feature, you must run `./configure` with the `--enable-referer-log` option. You must also enter a pathname for the `referer_log` directive. For example:

```
referer_log /usr/local/squid/var/logs/referer.log
```

Set the filename to `none` if you want to disable referer logging.

The Referer header normally contains the URI from which the request was obtained (see Section 14.36 of RFC 2616). For example, when a web browser issues a request for an embedded image, the Referer header is set to the URI of the (HTML) page containing the images. It is also set when you click on an HTML link. Some web site operators use Referer values to find so-called dead links. You may find `referer.log` particularly useful if you use Squid as a surrogate.

The `referer.log` has a simple format, with only four fields. Here are a few examples:

```
1068047502.377 3.0.168.206
http://www.amazon.com/exec/obidos/search-handle-form/002-7230223-8205634
```

```

http://www.amazon.com/exec/obidos/ASIN/0596001622/qid=1068047396/sr=2-1/...
1068047503.109 3.0.168.206
http://www.amazon.com/exec/obidos/ASIN/0596001622/qid=1068047396/sr=2-1/...
http://g-images.amazon.com/images/G/01/gourmet/gourmet-segway.gif
1068047503.196 3.0.168.206
http://www.amazon.com/exec/obidos/ASIN/0596001622/qid=1068047396/sr=2-1/...
http://g-images.amazon.com/images/G/01/marketing/cross-shop/arnold/appar...
1068047503.198 3.0.168.206
http://www.amazon.com/exec/obidos/ASIN/0596001622/qid=1068047396/sr=2-1/...
http://g-images.amazon.com/images/G/01/marketing/cross-shop/arnold/appar...
1068047503.825 3.0.168.206
http://www.amazon.com/exec/obidos/ASIN/0596001622/qid=1068047396/sr=2-1/...
http://images.amazon.com/images/P/B00005R8BC.01.TZZZZZZZ.jpg
1068047503.842 3.0.168.206
http://www.amazon.com/exec/obidos/ASIN/0596001622/qid=1068047396/sr=2-1/...
http://images.amazon.com/images/P/0596001622.01._PE_PI_SCMZZZZZZZ_.jpg

```

Note that requests that lack a Referer header aren't logged. The four fields are as follows:

1: timestamp

The time of the request, expressed as the number of seconds since Unix epoch with millisecond resolution.

Note that, unlike *access.log*, a *referer.log* entry is made as soon as Squid receives the complete request. Thus, the *referer.log* entry occurs before the *access.log*, which waits for the end of the response.

2: client address

The same as the client address in *access.log*. The *log_fqdn* and *client_netmask* directives affect this log file as well.

3: referer

The value of the Referer header from the client's request. Note that the referer value might have whitespace (or any other) characters. Squid doesn't encode the value before writing to *referer.log*.

4: URI

The URI that the client is requesting. It matches the URI in *access.log*.

useragent.log

The optional *useragent.log* contains User-Agent header values from client requests. To use this feature, you must supply the `--enable-useragent-log` option when running `./configure`. You also must enter a pathname for the *useragent_log* directive. For example:

```
useragent_log /usr/local/squid/var/logs/useragent.log
```

The User-Agent header normally contains a description of the agent that made the request. In most cases, the description is simply a list of product names with optional

version information. You should be aware that applications can easily provide false user-agent information. Modern user-agents provide a way to customize the description. Even Squid can alter the User-Agent header in forwarded requests.

The *useragent.log* format is relatively simple. It looks like this:

```
3.0.168.206 [05/Nov/2003:08:51:43 -0700]
    "Mozilla/5.0 (compatible; Konqueror/3; FreeBSD)"
3.0.168.207 [05/Nov/2003:08:52:18 -0700]
    "Opera/7.21 (X11; FreeBSD i386; U) [en]"
4.241.144.204 [05/Nov/2003:08:55:11 -0700]
    "Mozilla/5.0 (Macintosh; U; PPC Mac OS X; en-us) AppleWebKit/103u (KHTML..."
3.0.168.206 [05/Nov/2003:08:51:43 -0700]
    "Java1.3.1_01"
64.68.82.28 [05/Nov/2003:08:52:50 -0700]
    "Googlebot/2.1 (http://www.googlebot.com/bot.html)"
3.0.168.205 [05/Nov/2003:08:52:50 -0700]
    "WebZIP/4.1 (http://www.spidersoft.com)"
4.241.144.201 [05/Nov/2003:08:52:50 -0700]
    "Mozilla/4.0 (compatible; MSIE 5.0; Windows 98; DigExt; Hotbar 3.0)"
3.0.168.206 [05/Nov/2003:08:54:40 -0700]
    "Bookmark Renewal Check Agent [http://www.bookmark.ne.jp/] (Version 2.0..."
```

Unlike the other log files, it has just three fields:

1: *client address*

The same as the client address in *access.log*. The *log_fqdn* and *client_netmask* directives affect this log file as well.

2: *timestamp*

Unlike the other log files, which represent the time as seconds since the Unix epoch, this one uses a human-readable format. It is the HTTP common log-file format timestamp, which looks like this:

```
[10/Jun/2003:22:38:36 -0600]
```

Note that the square brackets delimit the timestamp, which includes a space character. Also note that, like *referer.log*, these entries are created as soon as Squid receives the complete request.

3: *user-agent*

The value of the User-Agent header. These strings almost always contain whitespace. Squid doesn't encode User-Agent values before writing them in this log file.

swap.state

A *swap.state* file is a journal of objects that have been added to, and removed from, a cache directory. Each *cache_dir* has its own *swap.state* file. When Squid starts up, it reads the *swap.state* files to rebuild its in-memory indexes of cached objects. These files are a relatively critical part of Squid's operation.

By default, each *swap.state* file is located in its corresponding cache directory. Thus, each state file automatically stays with each *cache_dir*. This is important if you ever decide to reorder your *cache_dir* lines or if you remove one or more from the list.

If you prefer to put them in a different location, you can use the *cache_swap_log* directive:

```
cache_swap_log /usr/local/squid/var/logs/swap.state
```

In this case, Squid creates a *swap.state* file for each directory by appending a numeric suffix. For example, if you have four cache directories, Squid creates the following:

```
/usr/local/squid/var/logs/swap.state.00
/usr/local/squid/var/logs/swap.state.01
/usr/local/squid/var/logs/swap.state.02
/usr/local/squid/var/logs/swap.state.03
```

In this situation, if you add, remove, or rearrange *cache_dir* lines, you may need to rename the *swap.state* files manually to keep everything consistent.

Technically, the *swap.state* format is storage scheme-dependent. However, all storage schemes use the same format in the current versions of Squid. The *swap.state* file uses a fixed-size (48-byte) binary format. Fields are written in host-byte order and are thus not necessarily portable between different operating systems. Table 13-3 describes the fields of a *swap.state* entry.

Table 13-3. *swap.state* entry fields

Name	Size, in bytes	Description
op	1	Operation on the entry: added (1) or deleted (2).
file number	4	Same as the fourth field of <i>store.log</i> , except it is stored in binary.
timestamp	4	A timestamp corresponding to the time when the response was generated or last validated. Taken from the Date header for responses that have one. Stored as the number of seconds since the Unix epoch.
lastref	4	A timestamp corresponding to the most recent access to the object.
expires	4	The object's expiration time, taken from an Expires header or Cache-Control max-age directive.
last-modified	4	The object's Last-Modified value.
swap file size	4	The amount of space the object occupies on disk. This includes HTTP headers and other Squid-specific meta-information.
refcount	2	The number of times this object has been requested.
flags	2	Various internal flags used by Squid.
key	16	The MD5 hash of the corresponding URI. Same as the key in <i>store.log</i> , except this one is stored in binary.

Rotating the Log Files

Squid always appends new entries to its log files. If your cache is busy, some of these files can become very large after a few days. Some operating systems even place limits on the size of a file (e.g., 2 GB) and return an error for writes beyond that size. To keep your log files manageable, and Squid happy, you must regularly rotate them.

Squid has a built-in feature for rotating log files. You can invoke it with the `squid -k rotate` command. You then tell Squid how many old copies of each file to keep with the `logfile_rotate` directive. For example, if you set it to 7, you'll have eight versions of each log file: the current file and seven old ones.

Old log files are renamed with numeric extensions. For example, when you execute a rotation, Squid renames `log.6` to `log.7`, then `log.5` to `log.6`, and so on. The current `log` becomes `log.0`, and Squid creates a new, empty file named `log`.

Each time you execute `squid -k rotate`, Squid rotates the following files: `cache.log`, `access.log`, `store.log`, `useragent.log` (if enabled), and `referer.log` (if enabled). Squid also creates up-to-date versions of the `swap.state` files. Note, however, that `swap.state` isn't archived with numeric extensions.

Squid doesn't rotate the log files automatically. The best way to make it happen is with a daily cron job. For example:

```
0 0 * * * /usr/local/squid/sbin/squid -k rotate
```

If you'd rather write your own scripts to manage the log files, Squid has a special mode that you'll find useful. Simply set the `logfile_rotate` directive to 0. Then, when you run `squid -k rotate`, Squid simply closes the current log files and opens new ones. This is very useful when the operating system allows you to rename files opened by another process. The following shell script illustrates the idea:

```
#!/bin/sh
set -e

yesterday_secs=`perl -e 'print time -43200'`
yesterday_date=`date -r $yesterday_secs +%Y%m%d`

cd /usr/local/squid/var/logs

# rename the current log file without interrupting the logging process
mv access.log access.log.$yesterday_date

# tell Squid to close the current logs and open new ones
/usr/local/squid/sbin/squid -k rotate

# give Squid some time to finish writing swap.state files
sleep 60

mv access.log.$yesterday_date /archive/location/
gzip -9 /archive/location/access.log.$yesterday_date
```

Privacy and Security

Squid's log files, especially *access.log*, contain a record of users' activities and, hence, are subject to privacy concerns. As the Squid administrator, you should take every precaution to keep the log files safe and secure. One of the best ways to do that is limit the number of people who have access to the system on which Squid runs. If that isn't possible, carefully examine the file and directory permissions to make sure they can't be viewed by untrusted or unauthorized users.

You can also help protect your users' privacy by taking advantage of the *client_net_mask* and *strip_query_terms* directives. The former makes it harder to identify individual users in the *access.log*; the latter removes URI query terms that may contain personal information. See the earlier section "Configuration Directives That Affect *access.log*" for more information.

You may also want to develop a policy for keeping old log files. Obviously *access.log* helps keep users accountable for their activities, but how far back would you ever need to go searching for something? A week? A year? What would you do if presented with a court order to hand over your log files for the last three months?

If you like to keep historical data for a long time, perhaps you can make the log files anonymous or somehow reduce the dataset. If you are interested only in which URIs were accessed, but not by whom, you can extract only that field from *access.log*. This not only makes the file smaller, it also reduces the risk of a privacy violation. Another technique is to randomize the client IP addresses. In other words, create a filter that maps real IP addresses to fake ones, such that the same real address is always changed to the same fake address. If you are using RFC 1413 identification or HTTP authentication, consider making those fields anonymous as well.

Exercises

- Configure Squid so that it doesn't create any log files, except for the *swap.state* file(s).
- Write a simple Perl or awk script to calculate your cache hit ratio from *access.log*.
- How does an "access denied" response appear in the *access.log*?
- Does *store.log* have the same number of, more, or fewer, entries than *access.log*?
- Take a file number from *store.log* and find the corresponding file in the disk cache. Examine the file and make sure you've found the correct response.
- Develop and implement a policy for archiving old cache log files. Consider where and how they will be stored, for how long, and who has permission to access them.

Monitoring Squid

How can you tell if Squid is performing well? Does Squid have enough memory, bandwidth, and disk space? When the Internet seems slow, is it Squid's fault or a problem somewhere else? Is the operating system giving enough resources to Squid? Is someone trying to abuse or hack into my proxy? You can find the answers to these, and many more, questions in this chapter. Squid provides information about itself in three different ways: *cache.log* messages, the *cache manager*, and an SNMP MIB.

Squid writes various messages to *cache.log* as it runs. Most of these are abnormal events of one sort or another. Unfortunately, Squid isn't always smart enough to differentiate serious problems from those that can be safely ignored. Even so, *cache.log* is a good place to start when investigating a Squid problem.

The cache manager and SNMP interfaces allow you to query Squid for a variety of data. The cache manager, which has its own shortcomings, probably provides the most information in current versions of Squid. It has a TCP socket-based interface and tries to generate output suitable for both human and computer processing. The bulk of this chapter is devoted to explaining all the information available from the cache manager.

Squid supports SNMP as well. Unfortunately, the data available through SNMP is only a subset of the cache-manager information. Additionally, the Squid MIB has not evolved much over the years; it's essentially unchanged since its first incarnation. I'll explain how to make Squid process SNMP queries and describe all objects in the current MIB.

cache.log Warnings

This is one of the first places you should look whenever you perceive a problem with Squid. During normal operation, you'll find various warnings and informational messages that may or may not indicate a problem. I covered the mechanics

of *cache.log* back in Chapter 13. Here, I'll go over a few of the warning messages you might see in your log file.

The *high_response_time_warning* directive makes Squid print a warning whenever the median response time exceeds a threshold. The value is in milliseconds and is disabled by default. If you add this line to *squid.conf*:

```
high_response_time_warning 1500
```

Squid will print the following warning if the median response time, measured over a 1-minute interval, exceeds 1.5 seconds:

```
2003/09/29 03:17:31| WARNING: Median response time is 2309 milliseconds
```

Before setting this directive, you should have a good idea of Squid's normal response time levels. If you set it too low, you'll get false alarms. In this particular example, it means that half of your user's requests take more than 2.3 second to complete. High response times may be caused by local problems, such as running out of file descriptors, or by remote problems, such as a severely congested Internet link.

The *high_page_fault_warning* directive is similar. It causes Squid to emit a warning if the number of page faults per minute exceeds a given value. A high page-fault rate usually indicates that the Squid process can't fit entirely in memory and must be swapped out to disk. This swapping severely impacts Squid's performance, so you should remedy the situation as soon as possible, as I'll discuss in Chapter 16.

Squid uses the Unix `getrusage()` function to get page fault counts. On some operating systems (e.g., Solaris), the page fault counter represents something besides swapping. Therefore, the *high_page_fault_warning* may cause false alarms on those systems.

The *high_memory_warning* directive is also similar to the previously mentioned warnings. In this case, it checks the size of the Squid process; if it exceeds the threshold, you get the warning in *cache.log*. On some operating systems, the process size can only grow and never shrink. Therefore, you'll constantly get this warning until Squid shuts down.

Process size information comes from either the `mallinfo()`, `mstats()`, or `sbrk()` functions. If these are unavailable on your operating system, the *high_memory_warning* warning won't work.

Squid has a number of other hardcoded warnings you may see in *cache.log*:

```
DNS lookup for 'neighbor.host.name' failed!
```

This occurs whenever Squid fails to look up the IP address for a cache neighbor. Squid refreshes the neighbor addresses every hour or so. As long as the neighbor's address is unknown, Squid doesn't send any traffic there.

Detected DEAD Sibling: neighbor.host.name/3128/3130

Squid logs this message when it believes it can't communicate with a neighbor cache. This happens, for example, when too many consecutive ICP queries go unacknowledged. See Chapter 10 for more information.

95% of replies from 'neighbor.host.name' are UDP_DENIED

This message indicates that a neighbor cache is refusing to answer Squid's queries. It probably means that you are sending queries to the neighbor without their permission. If they are using address-based access controls, and you have recently changed your address, they won't know about the change. Squid refuses to send any more queries to the neighbor after detecting this condition.

Probable misconfigured neighbor at 192.168.121.5

This occurs when you have an unauthorized cache client sending you ICP or HTCP queries. The best thing to do in this case is try to find out the person or organization responsible for the given address. Ask why they are querying your cache.

Forwarding loop detected for:

Recall that a forwarding loop occurs when a single request flows through Squid a second time. The request's Via header contains a list of all proxies that have seen the request. If Squid detects its own name in the Via list, it emits the forwarding loop warning and sends the request directly to the origin server. See Chapter 10 for an explanation of forwarding loops.

Closing client 192.168.121.5 connection due to lifetime timeout

The *client_lifetime* directive places an upper limit on the duration for a single HTTP request. Squid warns you when such a request is terminated because it may indicate someone is abusing your cache with very long-lived connections, for example, by downloading infinitely long objects.

As you can see, *cache.log* provides only notification of abnormal events. For periodic monitoring, you need something else. The cache manager is perhaps the best choice, even though its interface is less than perfect.

The Cache Manager

The Cache Manager is an interface to Squid for receiving information about various components. It is accessed via normal HTTP requests with a special protocol name: *cache_object*. A full cache manager URL looks like *cache_object://cache.host.name/info*. Squid provides two easy ways to access the cache manager information: the command-line *squidclient* program* or the *cachemgr.cgi* CGI program.

* In older versions of Squid, it was called just *client*.

The `squidclient` utility is a simple HTTP client, with a few special features for use with Squid. For example, you can use a shortcut to request the cache manager pages. Rather than typing a long URL like this:

```
% squidclient cache_object://cache.host.name/info
```

you can use this shorter version:

```
% squidclient mgr:info
```

`squidclient` is a convenient way to quickly see some of the cache manager pages. It's also useful when you need to save the cache manager output to disk for later analysis. However, some pages, such as the memory utilization table, are difficult to read in a terminal window. They are really designed to be formatted as an HTML page and viewed with your web browser. In that case, you may want to use `cachemgr.cgi`.

To use `cachemgr.cgi`, you must have an HTTP server that can execute the program. You can use an existing server or install one alongside Squid if you prefer. Keep in mind that the cache manager has only weak security (cleartext passwords). If the HTTP server is on a different host, you need to add its IP address to a cache manager access list (see the section "Cache Manager Access Controls"). You may also want to add access controls to the HTTP server so that others can't access `cachemgr.cgi`.

If you use Apache, I recommend making a special *cgi-bin* directory so you can protect `cachemgr.cgi` with access controls. For example, create a new directory, and copy the binary to it:

```
# mkdir /usr/local/apache/squid-cgi
# cp /usr/local/squid/libexec/cachemgr.cgi /usr/local/apache/squid-cgi
# chmod 755 /usr/local/apache/squid-cgi/cachemgr.cgi
```

Now, add a *ScriptAlias* line to Apache's *httpd.conf*:

```
ScriptAlias /squid-cgi/ "/usr/local/apache/squid-cgi/"
```

Finally, create an *.htaccess* file in the *squid-cgi* directory that contains access controls. To allow requests from only one IP address, use something like this:

```
Allow from 192.168.4.2
Deny from all
```

Once `cachemgr.cgi` is installed, simply enter the appropriate URL into your web browser. For example:

```
http://www.server.name/squid-cgi/cachemgr.cgi
```

If the CGI program is working, you should see a page with four fields. See Figure 14-1 for an example. The *Cache Host* field contains the name of the host on which Squid is running—*localhost* by default. You can set it with the `--enable-cachemgr-hostname` option when running `./configure`. Similarly, *Cache Port* contains the TCP port number to which Squid listens for requests. It's 3128 by default and can be changed with

the `--enable-cachemgr-port` option. The *Manager name* and *Password* fields are for access to protected pages, which I'll talk about shortly.

Figure 14-1. The *cachemgr.cgi* login screen

After clicking on the *Continue...* button, you should see a list of all cache manager pages currently available. The following section describes the various pages, some of which are available only when you enable certain features at compile time.

Cache Manager Pages

This section describes the cache manager pages, in the same order in which they appear in the menu. Each section title has both the page name (for use with *squidclient*), followed by its description. Descriptions that contain an asterisk indicate pages that are disabled by default, unless you configure a password for them. Table 14-1 shows the table of contents and the section number for each page.

Table 14-1. *Cache manager pages*

Short name	Description
leaks	Memory Leak Tracking
mem	Memory Utilization
cbdata	Callback Data Registry Contents
events	Event Queue
squidaio_counts	Async IO Function Counters
diskd	DISKD Stats
config	Current Squid Configuration*
comm_incoming	comm_incoming() Stats

Table 14-1. Cache manager pages (continued)

Short name	Description
ipcache	IP Cache Stats and Contents
fqdn-cache	FQDN Cache Stats and Contents
idns	Internal DNS Statistics
dns	Dns-server Statistics
redirector	URL Redirector Stats
basic-authenticator	Basic User Authenticator Stats
digest-authenticator	Digest User Authenticator Stats
ntlm-authenticator	NTLM User Authenticator Stats
external_acl	External ACL Stats
http-headers	HTTP Header Statistics
via-headers	Via Request Headers
forw-headers	X-Forwarded-For Request Headers
menu	This Cache Manager Menu
shutdown	Shut Down the Squid Process*
offline-toggle	Toggle offline_mode Setting*
info	General Runtime Information
file-descriptors	Process File Descriptor Allocation
objects	All Cache Objects
vm-objects	In-Memory and In-Transit Objects
openfd-objects	Objects with Swapout Files Open
io	Server-Side Network read() Size Histograms
counters	Traffic and Resource Counters
peer-select	Peer Selection Algorithms
digest-stats	Cache Digest and ICP Blob
5min	5 Minute Average of Counters
60min	60 Minute Average of Counters
utilization	Cache Utilization
histograms	Full Histogram Counts
active-requests	Client-Side Active Requests
store-digest	Store Digest
storedir	Store Directory Stats
store-check-cachable-stats	storeCheckCachable() Stats
store-io	Store IO Interface Stats
pconn	Persistent Connection Utilization Histograms
refresh	Refresh Algorithm Statistics
delay	Delay Pool Levels

Table 14-1. Cache manager pages (continued)

Short name	Description
forward	Request Forwarding Statistics
client_list	Cache Client List
netdb	Network Measurement Database
asndb	AS Number Database
carp	CARP Information
server_list	Peer Cache Statistics
non_peers	List of Unknown Sites Sending ICP Messages

leaks: Memory Leak Tracking

This page is available only with the `./configure --enable-leakfinder` option and is intended for developers trying to track down memory leaks. The page shows each memory pointer being tracked and where and when it was most recently referenced. See the Squid Programmer's Guide (<http://www.squid-cache.org/Doc/Prog-Guide/>) for more information about Squid's leak-finder feature.

mem: Memory Utilization

The memory utilization page shows a large table of numbers. Each row corresponds to a different pool of memory. The pools have names like `acl_list` and `MemObject`. Much of this information is of interest to developers only. However, a few columns are worth mentioning here.



It is important to keep in mind that this table doesn't represent all the memory allocated by Squid. Some memory allocations aren't tracked and don't appear in the table. Thus, the *Total* row may be much less than Squid's actual memory usage.

The `impact` column shows each pool's contribution to the total amount of memory allocated. Usually, the `StoreEntry`, MD5 digest, and LRU policy node pools take up most of the memory.

If you are a developer, you can use this page to look for memory leaks. The column labeled `high (hrs)` shows the amount of time elapsed since the pool reached its maximum size. A small value in this column may indicate that memory for that pool isn't being freed correctly.

You can also use this page to find out if certain features, such as `netdb`, the `ipcache`, and `client_db` consume too much memory. For example, the `ClientInfo` pool is associated with the `client_db` feature. The memory utilization page shows you how much memory you can save if you disable `client_db` in `squid.conf`.

cbdata: Callback Data Registry Contents

The Callback Data Registry is an internal Squid programming feature for managing memory pointers. Currently, this cache manager page doesn't provide much useful information, apart from the number of active *cbdata* pointers being tracked. In earlier Squid versions, the *cbdata* feature was implemented differently and this page provided some information to developers debugging their code.

events: Event Queue

Squid maintains an event queue for a number of tasks that must occur separately from user requests. Perhaps the most important of these is the periodic task that maintains the disk cache size. Every second or so, this task runs and looks for cache files to remove. On this page, you can see all tasks currently scheduled for execution. Most likely, you'll not find this very interesting unless you are hacking the source code.

squidaio_counts: Async IO Function Counters

This page is available only with the `./configure --enable-storeio=aufs` option. It shows counters for the number of open, close, read, write, stat, and unlink requests received. For example:

```
ASYNC IO Counters:
Operation      # Requests
open           15318822
close          15318813
cancel         15318813
write          0
read           19237139
stat           0
unlink         2484325
check_callback 311678364
queue          0
```

The cancel counter is normally equal to the close counter. This is because the close function always calls the cancel function to ensure that any pending I/O operations are ignored.

The write counter is zero because this version of Squid performs writes synchronously, even for *aufs*.

The check_callback counter shows how many times the main Squid process has checked the done queue for completed operations.

The queue value indicates the current length of the request queue. Normally, the queue length should be less than the number of threads $\times$ 5. If you repeatedly observe a queue length larger than this, you may be pushing Squid too hard. Adding more threads may help, but only to a certain point.

diskd: DISK Stats

This page is available only with the `./configure --enable-storeio=diskd` option. It provides various statistics relating to the *diskd* storage scheme.

The `sent_count` and `recv_count` lines are counters for the number of I/O requests sent between Squid and the group of *diskd* processes. The two numbers should be very close to each other and could possibly be equal. The difference indicates how many requests are currently outstanding.

The `max_away` value indicates the largest number of outstanding requests. Similarly, the `max_shmuse` counter indicates the maximum number of shared memory blocks in use at once. These two values are reset (to zero) each time you request this page. Thus, if you wait longer between requests for this page, these maximum counters are likely to be larger.

The `open_fail_queue_len` counter indicates the number of times that the *diskd* code decided to return failure in response to a request to open a file because the message queue exceeded its configured limit. In other words, this is the number of times a *diskd* queue reached the Q1 limit. Similarly, `block_queue_len` shows how many times the Q2 limit has been reached. See the descriptions of Q1 and Q2 in Chapter 8.

The *diskd* page also shows how many requests Squid sent to the *diskd* processes for each of the six I/O operations: open, create, close, unlink, read, and write. It also shows how many times each operation succeeded or failed. Note, these counters are incremented only for requests sent. The `open_fail_queue_len` check occurs earlier, and in that case, Squid doesn't send a request to a *diskd* process.

config: Current Squid Configuration*

This option dumps Squid's current configuration in the *squid.conf* format. Thus, if you ever accidentally remove the configuration file, you can recover it from the running Squid process. By saving the output to a file, you can also compare (e.g., with the `diff` command) the running configuration to the saved configuration. Note, however, that comments and blank lines aren't preserved.

This option reveals potentially sensitive information, so it's available only with a password. You must add a cache manager password for the *config* option with the *cachemgr_passwd* directive. See the section "Cache Manager Access Controls" for specifics. Additionally, these cache manager passwords aren't displayed in this output.

comm_incoming: comm_incoming() Stats

This page provides low-level network I/O information to developers and Squid wizards. The loop that checks for activity on file descriptors is called `comm_poll()`. Over the years, this function has become increasingly complicated in order to improve Squid's performance. One of those performance improvements relates to how often Squid checks certain network sockets relative to the others.

For example, the incoming HTTP socket is where Squid accepts new client connections. This socket tends to be busier than a normal data socket because each new connection comes through the incoming socket. To provide good performance, Squid makes an extra effort to check the incoming socket more frequently than the others.

At the top of the *comm_incoming* page, you'll see three incoming interval numbers: one each for ICP, DNS, and HTTP. The interval is the number of normal I/O events that Squid handles before checking the incoming socket again. For example, if *incoming_dns_interval* is set to 140, Squid checks the incoming DNS socket after 140 I/Os on normal connections. Unless your Squid is very busy, you'll probably see 256 for all incoming intervals.

The page also contains three histograms that show how many events occur for each incoming function call. Normally, the majority of the histogram counts occur in the low values. In other words, functions such as *comm_select_http_incoming()* usually handle between one and four events.

ipcache: IP Cache Stats and Contents

The IP cache contains cached results of hostname-to-address lookups. This cache manager page displays quite a lot of information. At the top of this page you'll see a handful of statistics like these:

```
IPcache Entries: 10034
IPcache Requests: 1066445
IPcache Hits: 817880
IPcache Negative Hits: 6846
IPcache Misses: 200497
```

In this example, you can see that the IP cache contains slightly more than 10,000 entries (hostnames). Since Squid was started, there have been 1,066,445 name-to-address requests, 817,880 of which were cache hits. This is a cache hit ratio of 77%. An IP cache negative hit occurs when Squid receives a subsequent request for a hostname that it recently failed to resolve. Rather than retry the DNS lookup immediately, Squid assumes it will fail again and returns an error message to the user.

Following these brief statistics, you'll see a long list of the IP cache contents. For each hostname in the cache, Squid prints six fields:

- The hostname itself
- Flags: N for negatively cached entries and H if the addresses came from the local *hosts* file, rather than the DNS
- The number of seconds since the hostname was last requested or used
- The number of seconds until the cached entry expires

- The number of IP addresses known for the host, and, in parentheses, the number of BAD addresses
- A list of IP addresses and whether each is OK or BAD

Here is a short sample (formatted to fit the page):

Hostname	Flg	lstref	TTL	N	
ads.x10.com		9	110	1(0)	63.211.210.20-OK
us.rd.yahoo.com		640	-340	4(0)	216.136.232.150-OK
					216.136.232.147-OK
					216.136.232.149-OK
					216.136.232.148-OK
www.movielodge.com		7143	-2161	1(0)	66.250.223.36-OK
shell.windows.com		10865	-7447	2(1)	207.46.226.48-BAD
					207.46.248.237-OK
www.surf3.net		126810	-40415	1(0)	212.74.112.95-OK

The list is sorted by the time since last reference. Recently referenced names are at the top of the list, and unused (about to be removed) names are at the bottom.

IP addresses are marked OK by default. An address is marked BAD when Squid receives an error or timeout during a TCP connection attempt. Subsequent IP cache requests don't return BAD addresses. If all the host's addresses become BAD, Squid resets them all back to OK.

fqdn_cache: FQDN Cache Stats and Contents

The FQDN cache is similar to the IP cache, except that it stores address-to-hostname lookups. Another difference is that the FQDN cache doesn't mark hostnames as OK or BAD.

Your FQDN cache may be empty, unless you enable the *log_fqdn* directive, use domain-based ACLs (such as *srcdomain*, *dstdomain*, *srcdom_regex*, and *dstdom_regex*), or use a redirector.

idns: Internal DNS Statistics

Squid contains an internal DNS client implementation, which is enabled by default. Disabling internal DNS with the *--disable-internal-dns* option also disables this page. Here is some sample output:

Internal DNS Statistics:

The Queue:

ID	SIZE	SENDS	DELAY SINCE	
			FIRST SEND	LAST SEND
001876	44	1	0.010	0.010
001875	44	1	0.010	0.010

```

Nameservers:
IP ADDRESS      # QUERIES # REPLIES
-----
192.168.19.124    4889    4844
192.168.19.190    91      51
192.168.10.2      73      39

```

```

Rcode Matrix:
RCODE ATTEMPT1 ATTEMPT2 ATTEMPT3
0      6149      4      2
1       0       0      0
2       38      34     32
3       0       0      0
4       0       0      0
5       0       0      0

```

The Internal DNS page contains three tables. First, you'll see the queue of unanswered queries. Unfortunately, you can't see the contents of the query (the host-name or IP address). Instead, Squid prints the ID, size, number of transmissions, and elapsed times for each query. You should see relatively few queries in the queue. If you see a lot relative to your total traffic rate, make sure your DNS servers are functioning properly.

The second table (Nameservers) shows how many queries have been sent to, and replies received from, each DNS server. Squid always queries the first server in the list first. Second (and third, etc.) servers are queried only when the previous server times out for a given query. If you see zero replies from the first address, make sure a server is actually running at that address.

Finally, you'll see a table of DNS response codes versus number of attempts. The cell for response code 0 and ATTEMPT1 should have the highest count. Response code 0 indicates success, while others are different types of errors (see RFC 1035 for their descriptions). You may see some smaller numbers for response code 0 in the columns for ATTEMPT2 and ATTEMPT3. This shows the cases when retransmitting a query, after initially receiving an error, resulted in a successful reply. Note that Squid retries only response code 2 (server failure) errors.

dns: Dnserver Statistics

This cache manager page is available only when you use the `--disable-internal-dns` option. In this case, Squid uses a number of external `dnserver` processes to perform DNS lookups. The `dnserver` program is one of a number of helper processes Squid can use. The other types of helpers are redirectors, authenticators, and external ACLs. All Squid's helpers have cache manager pages that display the same statistics. For example:

```

Dnserver Statistics:
number running: 5 of 5
requests sent: 3001
replies received: 3001

```

queue length: 0
avg service time: 23.10 msec

#	FD	PID	# Requests	Flags	Time	Offset	Request
1	6	20110	128	AB	0.293	0	www.nlanr.net
2	7	20111	45	A	0.000	0	(none)
3	8	20112	4	A	0.000	0	(none)
4	9	20113	0	A	0.000	0	(none)
5	10	20114	0	A	0.000	0	(none)

The number running line shows how many helper processes are running and how many should be running. The *dns_children* directive specifies how many dnsserver processes to use. The two numbers should match, but they may not if a helper process dies unexpectedly or if some processes could not be started. Recall that when you reconfigure a running Squid instance, all the helpers are killed and restarted. See the discussion in Appendix A.

The requests sent and replies received values display the number of requests sent to (and responses received from) the helpers since Squid started. The difference between these two, if any, should correspond to the number of outstanding requests.

The queue length line shows how many requests are queued, waiting for one of the helpers to become free. The queue length should usually be zero. If not, you should add more helpers to reduce delays for your users.

The avg service time line shows the running average service time for all helpers. Your particular value may depend on numerous factors, such as your network bandwidth and processing power.

The next section displays a table of statistics for the running dnsserver processes. The FD column shows the file descriptor for the socket between Squid and each dnsserver process. Similarly, the PID column shows each helper's process ID number.

The # Requests column shows how many requests have been sent to each helper. These numbers are zeroed each time you reconfigure Squid, so they may not add up to the total number of requests sent, as shown earlier. Note that Squid always chooses the first idle helper in the list, so the first process should receive the largest number of requests. The last few processes may not receive any requests at all.

The Flags column shows a few flags describing the state of the helper process. You should normally see A (for Alive) in each column. Occasionally, when the helper process is handling a request, you'll see B (for Busy).

The Time column displays the amount of time elapsed (in seconds) for the current, or last, request. Offset shows how many bytes of the response message Squid has read on the socket. This is almost always zero. Finally, the Request column shows the request that was sent to the helper process. In this case, it is either a hostname or an IP address.

redirector: URL Redirector Stats

The Redirector Stats page is available only if you are using a redirector (see Chapter 11). The format of this page is identical to *Dnsserver Statistics*, described earlier.

basicauthenticator: Basic User Authenticator Stats

This page is available only with the `./configure --enable-auth=basic` option and when you define a Basic authenticator with the `auth_param basic program` directive. The format of this page is identical to *Dnsserver Statistics*, described earlier.

digestauthenticator: Digest User Authenticator Stats

This page is available only with the `./configure --enable-auth=digest` option and when you define a Digest authenticator with the `auth_param digest program` directive. The format of this page is identical to *Dnsserver Statistics*, described earlier.

ntlmauthenticator: NTLM User Authenticator Stats

This page is available only with the `./configure --enable-auth=ntlm` option and when you define a NTLM authenticator with then `auth_param ntlm program` directive. The format of this page is similar to *Dnsserver Statistics*, described earlier, with a few additions.

The table of helper processes includes an extra column: # Deferred Requests. NTLM requires “stateful” helpers because the helper processes themselves generate the challenges. Squid receives a challenge from a helper, sends that challenge to a user, and receives a response. Squid must send the user’s challenge response back to the same helper for validation. For this protocol to work, Squid must defer some messages to be sent to a helper until the helper is ready to accept them.

These helpers also have two new flags: R (reserved or deferred) and P (placeholder). The R flag is set when the helper has at least one deferred request waiting. The P flag is set when Squid is waiting for the NTLM helper to generate a new challenge token.

external_acl: External ACL Stats

This page displays helper statistics for your external ACLs. If you don’t have any `external_acl_type` lines in *squid.conf*, this page will be empty. Otherwise, Squid displays the statistics for each external ACL. The format is the same as for the *Dns-server Statistics*.

http_headers: HTTP Header Statistics

This page displays a number of tables containing statistics about HTTP headers. It contains up to four sections: HTCP reply stats (if HTCP is enabled), HTTP request stats, HTTP reply stats, and a final section called HTTP Fields Stats. The HTCP

reply statistics refer to HTTP replies received by your cache. The HTTP request section refers to HTTP requests either sent or received by your cache. Similarly, the HTTP reply section refers to replies either sent or received by Squid.

The first three sections have the same format. Each section contains three tables: Field type distribution, Cache-control directives distribution, and Number of fields per header distribution.

The Field type distribution table shows the number of times that each header value occurs and the percentage of cases in which it occurs. For example, in Table 14-2 you can see that the Accept header occurs in 98% of HTTP requests.

Table 14-2. Sample Field type distribution values for HTTP requests

ID	Name	Count	#/header
0	Accept	1416268	0.98
1	Accept-Charset	322077	0.22
2	Accept-Encoding	709715	0.49
3	Accept-Language	1334736	0.92
...	...	...	...

Unfortunately, these (and the following) statistics are tricky because they don't correspond one-to-one for client requests. For example, Squid may report 1,416,268 Accept headers in requests but only 800,542 client requests. This happens because Squid creates more than one HTTP header data structure for each request. In the case of HTTP replies, it seems that Squid may create up to four separate header structures, depending on the circumstances.

The Cache-Control directives distribution is similar, but applies only to the values of the Cache-Control header. Table 14-3 shows some of the possible field values.

Table 14-3. Sample Cache-Control directives distribution values for HTTP requests

ID	Name	Count	#/cc_field
0	public	6866	0.02
1	private	69783	0.24
2	no-cache	78252	0.27
3	no-store	9878	0.03
4	no-transform	168	0.00
5	must-revalidate	10983	0.04
6	proxy-revalidate	2480	0.01
7	max-age	165034	0.56
8	s-maxage	4995	0.02
9	max-stale	0	0.00

Table 14-3. Sample Cache-Control directives distribution values for HTTP requests (continued)

ID	Name	Count	#/cc_field
10	only-if-cached	0	0.00
11	Other	9149	0.03

The Number of fields per header distribution table shows how many headers occur in each request or reply. Usually, you should see something like a normal distribution with a peak around 10–13 headers per request or response.

Finally, this page ends with a table labeled `Http Fields Stats` (replies and requests). For each header, this table shows three values: `#alive`, `%err`, and `%repeat`.

The `#alive` column shows how many instances of this header are currently stored in memory. HTTP headers are kept in memory for both active requests/responses and for completed objects stored in the memory cache.

The `%err` column shows the percentage of times Squid encountered an error while parsing this header. Common errors include incorrect date formats for `Date`, `Expires`, `Last-Modified`, and similar headers. The value `-1` indicates no errors.

The `%repeat` column indicates the number of times that a particular header is repeated in a single request or response. These aren't errors because HTTP allows headers to be repeated.

via_headers: Via Request Headers

This page is available only with the `./configure --enable-forw-via-db` option. The information in this page is intended to help cache administrators understand where client requests come from. When enabled, Squid counts the number of times each unique Via header occurs in client requests.

The Via header contains a list of downstream proxies that have forwarded the request so far. When a proxy forwards a request, it should append its hostname and other identifying information to the Via header. With the information in this database, you can, in theory, reconstruct the hierarchy of proxies forwarding requests through yours.

Squid prints the Via database entries in a random order. The output may look something like this:

```

4 1.0 proxy.firekitten.org:3128 (squid/2.5.STABLE1)
1 1.0 xnsproxy.dyndns.org:3128 (squid/2.5.PRE3-20020125)
1751 1.0 nt04.rmtcc.cc.oh.us:3128 (Squid/2.4.STABLE6),
      1.0 tasksmart.rmtcc.cc.oh.us:3128 (Squid/2.4.STABLE7)
137 1.0 reg3.bdg.telco.co.id:8080 (Squid/2.2.STABLE5),
      1.0 c1.telco.co.id:8080 (Squid/2.4.STABLE6),
      1.0 cache2.telco.co.id:8080 (Squid/2.4.STABLE1)
```

```
53 1.0 IS_GW_312:3128 (Squid/2.4.STABLE6)
60 1.0 proxy.kilttron.net:3128 (Squid/2.4.STABLE7)
815 1.1 DORM
```

In this example, Squid received 1751 requests that previously passed through two other proxies (*nt04* and *tasksmart*). Note that only proxies add a *Via* header. Requests from user-agents usually don't have the header and, therefore, aren't counted in this database.

As you can see, the *Via* headers reveal some semiprivate information, such as host-names, port numbers, and software versions. Please take care to respect the privacy of your users if you enable this feature.

The *Via* database is stored entirely in memory and is lost if Squid restarts. The database is cleared whenever you rotate the log files (see Chapter 13).

forw_headers: X-Forwarded-For Request Headers

This page is available only with the `./configure --enable-forw-via-db` option. It is similar to the *via_headers* page, except that it displays the accumulation of X-Forwarded-For headers.

X-Forwarded-For is a nonstandard HTTP header that originated with the Squid project. Its value is a list of client IP addresses. In other words, when Squid receives and forwards a request, it appends the client's IP address to this header. It is similar to *Via* because the header grows each time a proxy passes the request on towards the origin server.

The *forw_headers* output is similar to *via_headers*. Each line begins with an integer, followed by a header value. The integer indicates how many times that particular X-Forwarded-For value was received. For example:

```
1 10.37.1.56, 10.1.83.8
3 10.3.33.77, 10.1.83.8
569 116.120.203.54
21 10.65.18.200, 10.1.83.120
31 116.120.204.6
5 10.1.92.7, 10.1.83.120
1 10.3.65.122, 10.3.1.201, 10.1.83.8
2 10.73.73.51, 10.1.83.120
1 10.1.68.141, 10.1.83.8
3 10.1.92.7, 10.1.83.122
```

As with *via_headers*, this database is also stored in memory and is lost if Squid exits. The database is cleared each time you rotate Squid's log files.

menu: This Cache Manager Menu

This page simply displays a listing of the other cache manager pages. You can use it if you forget the name of a page or if you want to know if certain optional pages are available. When using `cachemngr.cgi`, each item in the menu is a clickable link.

shutdown: Shut Down the Squid Process*

This is one of the few cache manager functions that doesn't simply display some information. Rather, this "page" allows you to shut down Squid remotely. To allow shutdown via the cache manager, you must assign it a password with the *cachemgr_passwd* (see the section "Cache Manager Access Controls") directive in *squid.conf*. Without a password, the shutdown operation is disabled (but you can still use *squid -k shutdown*).

Because the cache manager has very weak security—passwords are sent in cleartext—I don't recommend enabling this operation.

offline_toggle: Toggle offline_mode Setting*

This is another function that allows you to control Squid, rather than simply receive information. It also requires a password (see the section "Cache Manager Access Controls") in order to become active.

Each time you request this page, Squid toggles the *offline_mode* setting. Squid reports the new setting on your screen and in *cache.log*.

info: General Runtime Information

This page provides a lot of basic information about the way that Squid is operating. It is a good starting point for using the cache manager and for tracking down performance problems.

At the top, you'll see the release version (e.g., Version 2.5.STABLE4) and two timestamps: the starting and current times. For example:

```
Squid Object Cache: Version 2.5.STABLE4
Start Time:      Mon, 22 Sep 2003 03:10:37 GMT
Current Time:    Mon, 13 Oct 2003 10:25:16 GMT
```

Following that, you'll see seven different sections. The first section, Connection information, displays a few statistics about the number and rate of connections, and the number of cache clients:

```
Connection information for squid:
  Number of clients accessing cache:      386
  Number of HTTP requests received:       12997469
  Number of ICP messages received:        16302149
  Number of ICP messages sent:            16310714
  Number of queued ICP replies:           0
  Request failure ratio:                   0.00
  Average HTTP requests per minute since start:  423.7
  Average ICP messages per minute since start:   1063.2
  Select loop called: 400027445 times, 4.601 ms avg
```

Number of clients accessing cache

Here, "client" actually means IP address. Squid assumes that each client has a unique IP address.

Number of HTTP requests received

The total number of HTTP requests since Squid was started.

Number of ICP messages received

The total number of ICP messages received since Squid was started. Note, received messages includes both queries and responses. These values don't include HTCP messages, however.

Number of ICP messages sent

The total number of ICP messages sent since Squid was started. Note, received messages includes both queries and responses. Doesn't include HTCP messages. Most likely, your sent and received counts will be about the same.

Number of queued ICP replies

ICP messages are sent over UDP. The `sendto()` system call rarely fails, but if it does, Squid queues the ICP message for retransmission. This counter shows how many times an ICP message was queued for retransmission. Most likely, you'll see 0 here.

Request failure ratio

The failure ratio is a moving average ratio between the number of failed and successful requests. In this context, a failed request is caused by either a DNS error, TCP connection error, or network read error. When this ratio exceeds 1.0—meaning Squid returns more errors than successful responses—Squid goes into hit-only mode. In this mode, Squid returns `ICP_MISS_NOFETCH` instead of `ICP_MISS`. Thus, your neighbor caches that use ICP won't forward cache misses to you until the problem goes away.

Average HTTP requests per minute since start

This value is simply the number of HTTP requests divided by the amount of time Squid has been running. This average doesn't reflect short-term variations in load. To get a better instantaneous load measurement, use the *5min* or *60min* page.

Average ICP messages per minute since start

The number of ICP queries received by Squid divided by the amount of time that it has been running.

Select loop called

This number is probably meaningful only to Squid developers. It represents the number of times the `select()` (or `poll()`) function has been called and the average time between calls. During normal operation, the time between calls should be in the 1–100 millisecond range.

The Cache information section displays hit ratio and cache size statistics:

Cache information for squid:

Request Hit Ratios:	5min: 22.6%, 60min: 25.8%
Byte Hit Ratios:	5min: 24.6%, 60min: 38.7%
Request Memory Hit Ratios:	5min: 0.7%, 60min: 1.4%

```
Request Disk Hit Ratios:      5min: 6.0%, 60min: 12.4%
Storage Swap size:          41457489 KB
Storage Mem size:           10180 KB
Mean Object Size:           14.43 KB
Requests given to unlinkd:    0
```

Request Hit Ratios

Here, and on subsequent lines, you'll see two hit ratio numbers: one for the last five minutes, and one for the last hour. These values are simply the percentage of HTTP requests that result in a cache hit. Here, hits include cases in which Squid validates a cached response and receives a 304 (Not Modified) reply.

Byte Hit Ratios

Squid calculates byte hit ratio by comparing the number of bytes received from origin servers (or neighbors) to the number of bytes sent to clients. When received bytes are less than sent bytes, the byte hit ratio is positive. However, it is possible to see a negative byte hit ratio. This might occur, for example, if you have a lot of clients that abort their request before receiving the entire response.

Request Memory Hit Ratios

These values represent the percentage of all cache hits that were served from memory. Or, more accurately, the percentage of all hits (not requests!) logged as `TCP_MEM_HIT`.

Request Disk Hit Ratios

Similarly, these values represent the percentage of "plain" cache hits served from disk. In particular, these values are the percentage of all hits logged as `TCP_HIT`. You'll see that the memory and disk hit percentages don't add up to 100%. This is because the other cases (such as `TCP_IMS_HIT`, etc.) aren't included in either disk or memory hits.

Storage Swap size

The amount of data currently cached on disk. It is always expressed in kilobytes. To compensate for space wasted in partial blocks at the end of files, Squid rounds up file sizes to the nearest filesystem block size.

Storage Mem size

The amount of data currently cached in memory. It is always expressed in kilobytes and is always a multiple of Squid's memory page size: 4 KB.

Mean Object Size

Simply the storage swap size divided by the number of cached objects. You should set the configuration directive `store_avg_object_size` close to the actual value reported here. Squid uses the configured value for a number of internal estimates.

Requests given to unlinkd

The unlinkd process handles file deletion external to Squid (depending on your configuration). This value simply shows how many files Squid has asked unlinkd to remove. It is zero when unlinkd isn't used.

The Median Service Times section displays the median of various service time (or response time) distributions. You'll see a value for the last five minutes and for the last hour. All values are in seconds. Squid uses the median, rather than the mean, because these distributions often have heavy tails that can significantly skew the mean value. The output looks like this:

Median Service Times (seconds)	5 min	60 min:
HTTP Requests (All):	0.19742	0.15048
Cache Misses:	0.22004	0.17711
Cache Hits:	0.05951	0.04047
Near Hits:	0.37825	0.14252
Not-Modified Replies:	0.01309	0.01387
DNS Lookups:	0.05078	0.03223
ICP Queries:	0.00000	0.07487

HTTP Requests (All)

These are the median response times for all HTTP requests taken together. For an HTTP request, the timer starts as soon as Squid receives the request and ends when Squid writes the last byte of the response. Thus, this time also includes DNS lookups (if any), and ICP queries to upstream neighbors (if you have them) for cache misses.

Cache Misses

This line shows the response time for cache misses only. Unless your cache hit ratio is close to 50%, the cache miss response time is close to (but a little larger than) the overall response time.

Cache Hits

The cache hit response time includes only requests logged as TCP_HIT, TCP_MEM_HIT, and TCP_OFFLINE_HIT. These are unvalidated cache hits served directly from Squid, without any communication to the origin server. Thus, your cache hit response time should be significantly less than the miss time. You should keep track of this value over time; if it climbs too high, your disk filesystem may be a performance bottleneck.

Near Hits

A near hit is a validated cache hit. It corresponds to TCP_REFRESH_HIT in *access.log*. For these, Squid contacts the origin server (or parent cache), which adds some latency to the response time. The server's response is a small 304 (Not Modified) message. Thus, the near hit response time is typically in between cache hits and cache misses.

Not-Modified Replies

This line shows the response times for requests logged as TCP_IMS_HIT. This occurs when the client sends a conditional (a.k.a. validation) request, and Squid serves a response without contacting the origin server. The name "not-modified" is somewhat misleading for this category because the status code received by the client isn't necessarily 304. For example, the client may send an If-modified-since request, and Squid has a fresh, cached response with a more recent

modification time. Squid knows that its response is fresh and that the client's copy is stale. In this case, the client receives a 200 (OK) reply with the new object data.

DNS Lookups

The DNS service time shows how long it takes, on average, to query the DNS. This includes both name-to-address and address-to-name lookups. It doesn't include IP- and FQDN-cache hits, however. DNS queries can be a significant source of latency. If you experience performance problems with Squid, be sure to check this value. If you see a high median service time (i.e., around five seconds), make sure your primary DNS server (usually listed in */etc/resolv.conf*) is up and running.

ICP Queries

The ICP query time represents the elapsed time between an ICP query and response that causes Squid to select the corresponding neighbor as the next hop. Thus, it includes only requests logged as PARENT_HIT, SIBLING_HIT, FIRST_PARENT_MISS, and CLOSEST_PARENT_MISS. This value may not be a good estimate of the overall ICP response time because ICP query/response transactions that don't result in Squid selecting a neighbor are ignored. Due to a bug in Squid Versions 2.5.STABLE1 and earlier, ICP response time statistics aren't collected, and these values always appear as 0.

The Resource usage section includes a few statistics relating to CPU and memory usage:

```
Resource usage for squid:
  UP Time:      1840478.681 seconds
  CPU Time:     70571.874 seconds
  CPU Usage:    3.83%
  CPU Usage, 5 minute avg:    1.33%
  CPU Usage, 60 minute avg:   4.41%
  Process Data Segment Size via sbrk(): 342739 KB
  Maximum Resident Size: 345612 KB
  Page faults with physical i/o: 65375
```

UP Time

This line simply shows the amount of time this Squid process has been running. It is expressed in seconds.

CPU Time

The amount of CPU time used by Squid, also in seconds. This value comes from the `getrusage()` system call, which might not be available on all operating systems.

CPU Usage

This section has three CPU Usage lines. The first is the CPU Time value divided by the UP Time value. It is a long-term average CPU usage measurement. The next two lines show the CPU usage for the last five minutes and the last hour.

Process Data Segment Size via `sbrk()`

This line offers an estimate of Squid's process size. `sbrk()` is a low-level system call used by the memory allocation library (`malloc()`). The `sbrk()` technique provides only an estimate, which usually differs from values reported by programs such as `ps` and `top`. When the `sbrk()` value is greater than the Maximum Resident Size (discussed next), the Squid process is probably page faulting, and performance may be degrading.

Maximum Resident Size

This is another estimate of memory usage and process size. The maximum resident set size (RSS) value comes from the `getrusage()` system call. Although the definition of RSS may vary between operating systems, you can think of it as the maximum amount of physical memory used by the process at any one time. Squid's process size may be larger than the RSS, in which case some parts of the process are actually swapped to disk.

Page faults with physical i/o

This value also comes from `getrusage()`. A *page fault* occurs when the operating system must read a page of the process's memory from disk. This usually happens when the Squid process becomes too large to fit entirely in memory, or when the system has other programs competing for memory. Squid's performance suffers significantly when page faults occur. You probably won't notice any problems as long as the page-faults rate is an order of magnitude lower than the HTTP request rate.

You'll see a section called *Memory usage for squid* via `mstats()` if your system has the `mstats()` function. In particular, you'll have this function if the GNU `malloc` library (*libgnumalloc.a*) is installed. Squid reports two statistics from `mstats()`:

```
Memory usage for squid via mstats():
Total space in arena: 415116 KB
Total free:          129649 KB 31%
```

Total space in arena

This represents the total amount of memory allocated to the process. It may be similar to the value reported by `sbrk()`. Note that this value only increases over time.

Total free

This represents the amount of memory allocated to the process but not currently in use by Squid. For example, if Squid frees up some memory, it goes into this category. Squid can later reuse that memory, perhaps for a different data structure, without increasing the process size. This value fluctuates up and down over time.

The *Memory accounted for* section contains a few tidbits about Squid's internal memory management techniques:

```
Memory accounted for:
Total accounted:      228155 KB
```

```
memPoolAlloc calls: 2282058666
memPoolFree calls: 2273301305
```

Total accounted

Squid keeps track of some, but not nearly all, of the memory allocated to it. This value represents the total size of all data structures accounted for. Unfortunately, it is typically only about two-thirds of the actual memory usage. Squid uses a significant amount of memory in ways that make it difficult to track properly.

memPoolAlloc calls

`memPoolAlloc()` is the function through which Squid allocates many fixed-size data structures. This line shows how many times that function has been called.

memPoolFree calls

`memPoolFree()` is the companion function through which Squid frees memory allocated with `memPoolAlloc()`. In a steady-state condition, the two values should increase at the same rate and their difference should be roughly constant over time. If not, the code may contain a bug that frees pooled memory back to the `malloc` library.

The File descriptor usage section shows how many file descriptors are available to Squid and how many are in use:

```
File descriptor usage for squid:
Maximum number of file descriptors: 7372
Largest file desc currently in use: 151
Number of file desc currently in use: 105
Files queued for open: 0
Available number of file descriptors: 7267
Reserved number of file descriptors: 100
Store Disk files open: 0
```

Maximum number of file descriptors

This is the limit on open file descriptors for the squid process. This should be the same value reported by `./configure` when you compiled Squid. If you don't see at least 1024 here, you should probably go back and recompile Squid after reading Chapter 3.

Largest file desc currently in use

This is the highest file descriptor currently open. Its value isn't particularly important but should be within 15–20% of the next line (number currently in use). This value is more useful for developers because it corresponds to the first argument of the `select()` system call.

Number of file desc currently in use

The number of currently open descriptors is an important performance metric. In general, Squid's performance decreases as the number of open descriptors increases. The kernel must work harder to scan the larger set of descriptors for activity. Meanwhile, each file descriptor waits longer (on average) to be serviced.

Files queued for open

This value will always be zero, unless you are using the *aufs* storage scheme (see Chapter 8). It shows how many file-open requests have been dispatched to the thread processes but have not yet returned. *aufs* is the only storage scheme in which disk file descriptors are opened asynchronously.*

Available number of file descriptors

The number of available descriptors is the maximum, minus the number currently open and the number queued for open. It represents the amount of breathing room Squid has to handle more load. When the available number gets close to the reserved number (next line), Squid stops accepting new connections so that existing transactions continue receiving service.

Reserved number of file descriptors

The number of reserved file descriptors starts out at the lesser of 100 or 25% of the maximum. Squid refuses new client connections if the number of available (free) descriptors reaches this limit. It is increased if Squid encounters an error while trying to create a new TCP socket. In this case, you'll see a message in *cache.log*:

```
Reserved FD adjusted from 100 to 150 due to failures
```

Store Disk files open

This counter shows the number of disk files currently open for reading or writing. It is always zero if you are using the *diskd* storage scheme because disk files are opened by the *diskd* processes, rather than Squid itself. If you use the *max_open_disk_fds* directive in *squid.conf*, Squid stops opening more cache files for reading or writing when it reaches that limit. If your filesystem is a bottleneck, this is a simple way to sacrifice a few cache hits for stable performance.

The Internal Data Structures section gives a quick overview of how many objects are in the cache and how many are on disk or in memory. You can find more detail about Squid's data structure allocations in the *mem* page (see the previous section "mem: Memory Utilization"). This section has a few stats:

```
Internal Data Structures:
  2873586 StoreEntries
    1336 StoreEntries with MemObjects
    1302 Hot Object Cache Items
  2873375 on-disk objects
```

StoreEntries

This represents the number of objects cached by Squid. Each object in the cache uses one StoreEntry structure.

* *diskd* also opens files asynchronously, but those file descriptors belong to the *diskd* processes, not the *squid* process.

StoreEntries with MemObjects

MemObject is the data structure used for objects currently being requested and for objects stored in the memory cache.

Hot Object Cache Items

The Hot Object Cache is another name for the memory cache (see Appendix B). These objects are stored entirely in memory (as well as on disk). This number should always be less than the number of entries with MemObjects.

on-disk objects

This counter shows how many objects are currently stored on disk. The counter is incremented when the entire object has been successfully written. Thus, this number isn't necessarily equal to the number of StoreEntries minus the number of Hot Objects.

filedescriptors: Process File Descriptor Allocation

This page displays a table of all file descriptors currently opened by Squid. It looks like this:

File Type	Tout	Nread	* Nwrite	* Remote Address	Description
3 File	0	0	0		/usr/local/squid/logs/cache.log
6 File	0	0	2083739		/usr/local/squid/logs/access.log
12 Pipe	0	0	0		unlinkd -> squid
13 File	0	0	2485913		/usr/local/squid/logs/store.log
15 Pipe	0	0	0		squid -> unlinkd
16 Socket	24	220853*	1924	65.200.216.110.80	http://downloads.mp3.com/
18 Pipe	0	0	0		squid -> diskd
19 Socket	179	476*	1747	202.59.16.30.4171	http://ads.vesperexchange.com/
21 Pipe	0	0	0		squid -> diskd
22 Socket	20	158783*	998	210.222.20.8.80	http://home.hanmir.com/a
24 Pipe	0	0	0		squid -> diskd
25 Socket	1	0	0*	210.222.20.8.80	http://home.hanmir.com/b
26 Socket	0	9048307*	1578290	.0	DNS Socket
27 Pipe	0	0	0		squid -> diskd
28 Socket	0	0	0*	66.28.234.77.80	http://updates.hotbar.com/
29 Socket	0	0*	0	.0	HTTP Socket
30 Pipe	0	0	0		squid -> diskd
31 Socket	0	93	1126	127.0.0.1.3434	ncsa_auth #1
32 Socket	0	3	31	127.0.0.1.3438	ncsa_auth #3
33 Socket	0	0	0	127.0.0.1.3440	ncsa_auth #4
34 Socket	164	8835*	1070222*	212.47.19.52.2201	http://www.eyyubyaqubov.com/
35 Socket	177	6137*	249899*	212.47.19.25.3044	http://files10.rarlab.com/
36 Socket	0	0	0	127.0.0.1.3442	ncsa_auth #5
37 Socket	7	158783*	774	210.222.20.8.80	http://home.hanmir.com/c
38 Socket	166	1000*	148415*	202.17.13.8.5787	http://home.hanmir.com/d

The table has seven columns:

File

This is simply the file descriptor number. The list always starts with 3 because descriptors 0, 1, and 2 are reserved for *stdin*, *stdout*, and *stderr*. Any other gaps in the list represent closed descriptors.

Type

The type field contains one of the following values: *File*, *Pipe*, or *Socket*. The *File* type is used both for files storing cached responses and for log files, such as *cache.log* and *access.log*. The *Pipe* type represents kernel pipes used for interprocess communication. The *Socket* type is also occasionally used for interprocess communication, but it's mostly used for HTTP (and FTP) connections to clients and servers.

Tout

This is the general-purpose timeout value for the descriptor. It is expressed in minutes. *Files* and *Pipes* usually don't have a timeout, so this value is zero. For *Sockets*, however, if this number of minutes go by without any activity on the descriptor, Squid calls a timeout function.

Nread

This is where Squid reports the number of bytes read from the descriptor. An asterisk (*) after the number means Squid has a function (a read handler) registered to read additional data, if there is some available.

Nwrite

This column shows the number of bytes written to the descriptor. Again, the asterisk (*) indicates that a write handler is present for the descriptor. You can usually tell if a given socket is connected to a client or to a server by comparing the number of bytes read and written. Because requests are normally smaller than responses, a server connection has a higher Nread count than Nwrite. The opposite is true for client connections.

Remote Address

For *Sockets*, this field shows the remote TCP address of the connection. The format is similar to what you would find in `netstat -n` output: an IP address followed by the TCP port number.

Description

The description field indicates the descriptor's use. For *Files*, you'll see a path-name; for *Pipes*, a description to what the pipe is connected; and for *Sockets*, a URI, or at least the first part of it. A description such as `web.icq.com idle connection` indicates an idle persistent connection to an origin server. Similarly, `Waiting for next request` is an idle client-side persistent connection.

By default, the File Descriptor page isn't password-protected. However, you may want to give it a password because it contains some sensitive and, perhaps, personally identifiable information.

objects: All Cache Objects

Requesting this page results in a list of all objects in the cache. Be careful with this page because it can be extremely long. Furthermore, it contains low-level information that is probably useful only to developers.

For each cached object, Squid prints a sequence of lines, most of which look like this:

```
KEY FF1F6736BCC167A4C3F93275A126C5F5
STORE_OK      NOT_IN_MEMORY SWAPOUT_DONE PING_NONE
CACHABLE,DISPATCHED,VALIDATED
LV:1020824321 LU:1020824671 LM:1020821288 EX:-1
0 locks, 0 clients, 1 refs
Swap Dir 0, File 0X010AEE
```

The first line shows the cache key—a 128-bit MD5 checksum of the URI. The same MD5 checksum appears in *store.log* and in the metadata at the beginning of each response cached on disk.

The second line shows four state variables of the StoreEntry data structure: *store_status*, *mem_status*, *swap_status*, and *ping_status*. Refer to the Squid source code if you'd like more information about them.

The third line is a list of the StoreEntry flags that are set. Search the source code for *e->flags* for more information.

The fourth line shows the values of four timestamps: last-validation, last-use, last-modification, and expiration. The last-modification and expiration timestamps are taken from the origin server's HTTP response. The others are maintained by Squid.

The fifth line shows a few counters: locks, clients, and references. An entry with locks can't be removed. The clients counter shows how many clients are currently receiving data for this object. The refs counter shows how many times the object has been requested.

The sixth line shows the object's index to the on-disk storage. Each object has a 7-bit swap directory index and a 25-bit file number. Each storage scheme has a function to map these numbers into pathnames.

vm_objects: In-Memory and In-Transit Objects

This page is similar to All Cache Objects, except that it displays only objects that have a MemObject data structure. In other words, objects that are currently being requested or are stored in the memory cache. These objects are displayed like this:

```
KEY 5107D49BA7F9C6BA9559E006D6DDC4B2
GET http://www.rpgplanet.com/ac2hq/cartography/dynamic/LinvakMassif.jpg
STORE_PENDING NOT_IN_MEMORY SWAPOUT_WRITING PING_DONE
CACHABLE,DISPATCHED,VALIDATED
LV:1043286120 LU:1043286122 LM:1036015230 EX:-1
4 locks, 1 clients, 1 refs
```

```
Swap Dir 1, File 00X31BD9
inmem_lo: 184784
inmem_hi: 229840
swapout: 229376 bytes queued
swapout: 229509 bytes written
Client #0, 1533a1018
  copy_offset: 217552
  seen_offset: 217552
  copy_size: 4096
  flags:
```

As you can see, many of the lines are the same. However, the in-memory objects have a few additional lines. Directly following the cache key (MD5 checksum), Squid prints the request method and URI.

The `inmem_lo` and `inmem_hi` lines are byte offsets of the HTTP reply. They indicate the section of object data currently in memory. In most cases, the difference between these two should be less than the value of the *maximum_object_size_in_memory* directive.

The `swapout: bytes queued` line shows the offset for how many bytes have been given to the storage layer for writing. For objects in the `SWAPOUT_DONE` state, this value is the same as the object size. If the state is `SWAPOUT_WRITING`, Squid also shows the `bytes written` line, which indicates how many bytes have been successfully stored on disk.

If one or more clients are currently receiving the response, you'll see a section for each of them (Client #0 in this example). For each client, Squid reports another pair of offset values. The first, `copy_offset`, is the starting point for the last time the client-side asked for data from the storage system. The second, `seen_offset`, is the point at which the response data has been sent to the client. Note that `copy_offset` is always greater than or equal to `seen_offset`. The `copy_size` indicates the maximum amount of data the client can receive from the storage system.

openfd_objects: Objects with Swapout Files Open

The format of this page is the same as for In-Memory and In-Transit Objects. The objects reported on this page should all be in the `SWAPOUT_WRITING` state. The page is primarily useful to developers when trying to track down file-descriptor leaks.

io: Server-Side Network read() Size Histograms

This page displays a histogram for each of the following four server-side protocols: HTTP, FTP, Gopher, and WAIS. The histograms show how many bytes each `read( )` call received. The information is primarily useful to developers for tuning buffer sizes and other aspects of the source code.

The bins of the histogram are logarithmic to accommodate the large scale of read sizes. Here is an example:

```
HTTP I/O
number of reads: 9016088
Read Histogram:
 1-   1:   3082  0%
 2-   2:    583  0%
 3-   4:    905  0%
 5-   8:   2666  0%
 9-  16:  16690  0%
17-  32:  88046  1%
33-  64:  19712  0%
65- 128: 116655  1%
129- 256: 749259  8%
257- 512: 633075  7%
513-1024: 903145 10%
1025-2048: 3664862 41%
2049-4096: 1643747 18%
4097-8192: 789796  9%
8193-16384: 99476  1%
16385-32768: 30059  0%
```

In this case, you can see that the bin for 1025–2048 bytes is the most popular. When reading from an HTTP server, Squid got between 1025 and 2048 bytes per read 41% of the time.

counters: Traffic and Resource Counters

Squid maintains a data structure of counters. Actually, it is an array of counters. Squid shifts the array every 60 seconds and calculates 1-, 5-, and 60-minute averages from this array. This page is simply a dump of the current counter values in a format more suitable for computer processing than for reading by humans. The counters are as follows:

`sample_time`

The sample time is actually the time of the last shift, rather than the current time. The sample time is always within 60 seconds of the current time.

`client_http.requests`

The number of HTTP requests received from clients.

`client_http.hits`

The number of cache hits in response to client requests. A hit is any transaction logged with one of the TCP_HIT codes in *access.log*.

`client_http.errors`

The number of client transactions that resulted in an error.

`client_http.kbytes_in`

The amount of traffic (in kilobytes) received from clients in their requests. This is measured at the HTTP layer and doesn't include TCP, IP, and other packet headers.

`client_http.kbytes_out`

The amount of traffic (in kilobytes) sent to clients in responses. Also measured at the HTTP layer.

`client_http.hit_kbytes_out`

The amount of traffic sent to clients in responses that are cache hits. Keep in mind that some cache hits are 304 (Not Modified) responses.

`server.all.requests`

The number of requests forwarded to origin servers (or neighbor caches) for all server-side protocols (HTTP, FTP, Gopher, etc.).

`server.all.errors`

The number of server-side requests (all protocols) that resulted in some kind of error.

`server.all.kbytes_in`

The amount of traffic (in kilobytes) read from the server-side for all protocols.

`server.all.kbytes_out`

The amount of traffic written to origin servers and/or neighbor caches for server-side requests.

`server.http.requests`

The number of server-side requests to HTTP servers, including neighbor caches.

`server.http.errors`

The number of server-side HTTP requests that resulted in an error.

`server.http.kbytes_in`

The amount of traffic read from HTTP origin servers and neighbor caches.

`server.http.kbytes_out`

The amount of traffic written to HTTP origin servers and neighbor caches.

`server.ftp.requests`

The number of requests sent to FTP servers.

`server.ftp.errors`

The number of requests sent to FTP servers that resulted in an error.

`server.ftp.kbytes_in`

The amount of traffic read from FTP servers, including control channel traffic.

`server.ftp.kbytes_out`

The amount of traffic written to FTP servers, including control channel traffic.

`server.other.requests`
The number of “other” server-side requests. Currently, the other protocols are Gopher, WAIS, and SSL.

`server.other.errors`
The number of Gopher, WAIS, and SSL requests that resulted in an error.

`server.other.kbytes_in`
The amount of traffic read from Gopher, WAIS, and SSL servers.

`server.other.kbytes_out`
The amount of traffic written to Gopher, WAIS, and SSL servers.

`icp.pkts_sent`
The number of ICP messages sent to neighbors. This includes both queries and replies but doesn’t include HTCP messages.

`icp.pkts_rcv`
The number of ICP messages received from neighbors, including both queries and replies.

`icp.queries_sent`
The number of ICP queries sent to neighbors.

`icp.replies_sent`
The number of ICP replies sent to neighbors.

`icp.queries_rcv`
The number of ICP queries received from neighbors.

`icp.replies_rcv`
The number of ICP replies received from neighbors.

`icp.query_timeouts`
The number of times that Squid timed out waiting for ICP replies to arrive.

`icp.replies_queued`
The number of times Squid queued an ICP message after the initial attempt to send failed. See the section “info: General Runtime Information.”

`icp.kbytes_sent`
The amount of traffic sent in all ICP messages, including both queries and replies.

`icp.kbytes_rcv`
The amount of traffic received in all ICP messages, including both queries and replies.

`icp.q_kbytes_sent`
The amount of traffic sent to neighbors in ICP queries.

`icp.r_kbytes_sent`
The amount of traffic sent to neighbors in ICP replies.

`icp.q_kbytes_recv`
The amount of traffic received from neighbors in ICP queries.

`icp.r_kbytes_recv`
The amount of traffic received from neighbors in ICP replies.

`icp.times_used`
The number of times ICP resulted in the selection of a neighbor as the next-hop for a cache miss.

`cd.times_used`
The number of times Cache Digests resulted in the selection of a neighbor as the next-hop for a cache miss.

`cd.msgs_sent`
The number of Cache Digest messages sent to neighbors.

`cd.msgs_recv`
The number of Cache Digest messages received from neighbors.

`cd.memory`
The amount of memory (in kilobytes) used by enabling the Cache Digests' feature.

`cd.local_memory`
The amount of memory (in kilobytes) used to store Squid's own Cache Digest.

`cd.kbytes_sent`
The amount of traffic sent to neighbors in Cache Digest messages.

`cd.kbytes_recv`
The amount of traffic received from neighbors in Cache Digest messages.

`unlink.requests`
The number of unlink requests given to the (optional) `unlinkd` process.

`page_faults`
The number of (major) page faults as reported by `getrusage()`.

`select_loops`
The number of times Squid called `select()` or `poll()` in the main I/O loop.

`cpu_time`
The amount of CPU time (in seconds) accumulated, as reported by `getrusage()`.

`wall_time`
The amount of human time (in seconds) elapsed since Squid was started.

`swap.outs`
The number of objects (swap files) saved to disk.

`swap.ins`
The number of objects (swap files) read from disk.

swap.files_cleaned

The number of orphaned cache files removed by the periodic cleanup procedure.

aborted_requests

The number of server-side HTTP requests aborted due to client-side aborts.

peer_select: Peer Selection Algorithms

This page contains a lot of low-level detail about cache digests that I won't discuss. Most of the numbers are meaningful only to the developers that originally wrote the Cache Digest implementation.

However, at the end of this page is a little table that compares Algorithm usage:

Algorithm usage:	
Cache Digest:	27 (24%)
Icp:	84 (76%)
Total:	111 (100%)

In this example, you can see that Squid sent 111 requests to one of its neighbors: 27 are due to Cache Digests and 84 are due to ICP. In this context, ICP also includes HTCP.

digest_stats: Cache Digest and ICP Blob

This page is actually just a concatenation of the following other cache manager pages:

- Traffic and Resource Counters
- 5 Minute Average of Counters
- Full Histogram Counts
- Peer Selection Algorithms
- Store Digest

Its only purpose is to enable developers to take a snapshot of a number of statistics with a single request.

5min: 5 Minute Average of Counters

This page shows a five-minute average of the data in the Traffic and Resource Counters page. In addition to the counters mentioned in the section “counters: Traffic and Resource Counters,” this page also contains the following values:

client_http.all_median_svc_time

The median service (response) time for all client requests from the last five minutes.

client_http.miss_median_svc_time

The median service time for cache misses from the last five minutes.

`client_http.nm_median_svc_time`

The five-minute median service time for requests logged as `TCP_IMS_HIT`. See “Not-Modified Replies” in the “info: General Runtime Information” section.

`client_http.nh_median_svc_time`

The five-minute median service time for Near Hits (`TCP_REFRESH_HIT` requests).

`client_http.hit_median_svc_time`

The five-minute median service time for unvalidated cache hits.

`icp.query_median_svc_time`

The five-minute median service time for ICP queries sent by Squid (how long it takes for the neighbors to reply to our queries).

`icp.reply_median_svc_time`

The five-minute median service time for ICP queries received by Squid (how long it takes Squid to reply to its neighbor’s queries). ICP processing normally occurs faster than the process clock is updated, so this value is always zero.

`dns.median_svc_time`

The five-minute median service time for DNS queries.

`select_fds`

The mean rate at which the main I/O loop scans file descriptors with `select()` or `poll()`. Note: a low number doesn’t necessarily indicate poor performance. It may just be that Squid often has no work to do.

`average_select_fd_period`

The mean number of seconds required to scan a file descriptor in the main I/O loop.

`median_select_fds`

The five-minute median number of ready file descriptors each time Squid calls `select()` or `poll()` (the median of the `select()/poll()` return value). Unfortunately, this value is almost always zero because Squid’s functions for calculating the median don’t work very well with the `select_fds` histogram, in which 0 and 1 are the most common values.

`syscalls.selects`

The five-minute mean rate of calls to `select()/poll()`. If Squid is using `poll()` on your system, the variable is called `syscalls.polls`. This value may be a little larger than `select_loops`, because the latter only includes calls in the main I/O loop.

`syscalls.disk.opens`

The five-minute mean rate of `open()` calls for disk files.

`syscalls.disk.closes`

The five-minute mean rate of `close()` calls for disk files.

`syscalls.disk.reads`

The five-minute mean rate of `read()` calls for disk files.

`syscalls.disk.writes`

The five-minute mean rate of `write()` calls for disk files.

`syscalls.disk.seek`

The five-minute mean rate of `seek()` calls for disk files. Probably zero unless you are using *aufs*, which always calls `seek()` before reading.

`syscalls.disk.unlink`

The five-minute mean rate of `unlink()` (or, in some cases, `truncate()`) calls for disk files.

`syscalls.sock.accepts`

The five-minute mean rate of `accept()` calls for network sockets.

`syscalls.sock.sockets`

The five-minute mean rate of `socket()` calls for network sockets.

`syscalls.sock.connects`

The five-minute mean rate of `connect()` calls for network sockets.

`syscalls.sock.binds`

The five-minute mean rate of `bind()` calls for network sockets.

`syscalls.sock.closes`

The five-minute mean rate of `close()` calls for network sockets.

`syscalls.sock.reads`

The five-minute mean rate of `read()` calls for network sockets.

`syscalls.sock.writes`

The five-minute mean rate of `write()` calls for network sockets.

`syscalls.sock.recvfrom`

The five-minute mean rate of `recvfrom()` calls for network sockets. Used for UDP-based protocols, such as DNS, ICP, HTCP, and some interprocess communication.

`syscalls.sock.sendto`

The five-minute mean rate of `sendto()` calls for network sockets. Used for UDP-based protocols, such as DNS, ICP, HTCP, and some interprocess communication.

60min: 60 Minute Average of Counters

This page shows a 60-minute average of the data in the Traffic and Resource Counters page. The descriptions are identical to those for the 5 Minute Average of Counters page, except the measurements are taken over one hour.

utilization: Cache Utilization

This page displays averages of the counters (see Traffic and Resource Counters and 5 Minute Average of Counters) over various time spans. The same values are reported for 5-minute, 15-minute, 1-hour, 8-hour, 1-day, and 3-day intervals.

This page, with a poorly chosen name, exists primarily so that developers can take a quick snapshot of statistics for testing purposes.

histograms: Full Histogram Counts

This page displays the current histogram values (since Squid was started) for a number of measurements:

- client_http.all_svc_time
- client_http.miss_svc_time
- client_http.nm_svc_time
- client_http.nh_svc_time
- client_http.hit_svc_time
- icp.query_svc_time
- icp.reply_svc_time
- dns.svc_time
- select_fds_hist

These are the same measurements described in the section “5min: 5 Minute Average of Counters,” except that here Squid gives the full histogram, instead of the mean or median.

Depending on the type of histogram, you may see two or three columns. The first column is the bin number and lower bound on the bin value. The second column is the number of counts for that bin. The optional third column is the number of counts divided by the “size” of the bin. The last column is probably only interesting for log-based histograms, in which the bin size isn’t constant.

active_requests: Client-Side Active Requests

This page shows a list of currently active client-side requests. The list is sorted starting with the most recent, and ending with the oldest requests. The information given here is primarily useful to developers. A typical entry looks like this:

```
Connection: 0x84ecd10
FD 132, read 1273, wrote 12182
FD desc: http://www.squid-cache.org/Doc/FAQ/FAQ.html
in: buf 0xa063000, offset 0, size 4096
peer: 206.168.0.9:1058
me: 192.43.244.42:3128
```

```

nrequests: 3
defer: n 0, until 0
uri http://www.squid-cache.org/Doc/FAQ/FAQ.html
log_type TCP_MISS
out.offset 0, out.size 0
req_sz 392
entry 0x960c680/3B49762ABF444D80B6465552F6CFAD4C
old_entry 0x0/N/A
start 1066036250.669955 (2.240814 seconds ago)

```

Connection

The internal memory address of the connection structure.

FD

The file descriptor for the TCP connection, followed by the number of bytes read and written.

FD desc

A short description of the socket, usually a URI. This is the same as in the “file-descriptors: Process File Descriptor Allocation” section.

in

The internal memory location of the input buffer, the offset at which Squid will place data after the next `read( )` call, and the size of the input buffer.

peer

The remote socket address of the TCP connection. You can correlate this value with what you see in `netstat -n` output.

me

The local socket address of the TCP connection.

nrequests

The number of HTTP requests received on this connection. A value greater than 1 indicates persistent connection reuse.

defer

Indicates whether Squid is postponing reads on the socket.

uri

The URI from the client’s request. Unlike `FD desc`, this one isn’t truncated.

log_type

The cache status code that appears in *access.log* when this transaction is complete.

out.offset

The offset, relative to the start of the HTTP reply message, in which the client side has requested data from the storage system.

out.size

The number of response bytes written to the client.

req_sz

The size of the client's HTTP request. Note, for persistent connections, this refers only to the current request.

entry

The memory address and MD5 hash of the corresponding StoreEntry structure.

old_entry

For validation requests, this is the memory address and MD5 hash of the cached response StoreEntry.

start

The time at which Squid began processing this request.

store_digest: Store Digest

This page is available only with the `./configure --enable-cache-digests` option. It displays a few statistics about Squid's own cache digest. It looks like this:

```
store digest: size: 620307 bytes
               entries: count: 324806 capacity: 992490 util: 33%
               deletion attempts: 0
               bits: per entry: 5 on: 1141065 capacity: 4962456 util: 23%
               bit-seq: count: 1757902 avg.len: 2.82
               added: 324806 rejected: 611203 ( 65.30 %) del-ed: 0
               collisions: on add: 0.08 % on rej: 0.07 %
```

size

The number of bytes that the digest occupies in memory.

entries count

The number of cached objects entered into the digest.

entries capacity

The target capacity for the digest. Note, this isn't a hard limit, but rather an estimate for optimally sizing the digest.

entries util

The percentage of entries added compared to the capacity.

deletion attempts

Squid doesn't currently support deletion of cache digest entries, so this is always zero.

bits per entry

The number of bits that each item turns on. The same as the *digest_bits_per_entry* value from *squid.conf*.

bits on

The number of bits that have been turned on so far.

bits capacity

The total number of bits in the digest. Equal to the digest size multiplied by eight.

bit-seq count

The number of same-bit sequences in the digest. For example, the pattern 110100011111 has 5 sequences of 1s and 0s.

bit-seq avg.len

The mean length of same-bit sequences.

added

The number of entries added to the digest since it was created.

rejected

The number of entries not added to the digest. An entry may not be added because it isn't cachable, is too large, stale, or about to become stale, etc.

del-ed

Squid doesn't currently support deletion of cache digest entries, so this is always zero.

collisions on add

This is the percentage of additions that didn't turn on any new bits. Recall that Bloom filters have the property that two or more entries may turn on the same bit.

collisions on rej

This is the percentage of rejected additions that wouldn't have turned on any new bits.

storedir: Store Directory Stats

This page displays some statistics from the storage system. First, you'll see a few global values. For example:

```
Store Directory Statistics:
Store Entries           : 2873564
Maximum Swap Size      : 46080000 KB
Current Store Swap Size: 41461672 KB
Current Capacity       : 90% used, 10% free
```

Store Entries

The number of StoreEntry objects. Most, but not necessarily all, of these are for on-disk objects.

Maximum Swap Size

The sum of all *cache_dir* sizes.

Current Store Swap Size

The total amount of cached data currently stored on disk. Note that Squid rounds response sizes (e.g., 1722 bytes) up to the nearest multiple filesystem block size (e.g., 2048 bytes) when incrementing and decrementing this value.

Current Capacity

The percentage of the maximum disk space currently in use. The percentage in use should normally stay below the *cache_swap_high* value.

Next, you'll see a section for each *cache_dir*. It looks something like this:

```
Store Directory #1 (diskd): /cache1
FS Block Size 1024 Bytes
First level subdirectories: 16
Second level subdirectories: 64
Maximum Size: 15360000 KB
Current Size: 13823996 KB
Percent Used: 90.00%
Filemap bits in use: 958439 of 2097152 (46%)
Filesystem Space in use: 14030485/17370434 KB (81%)
Filesystem Inodes in use: 959440/4340990 (22%)
Flags: SELECTED
Pending operations: 0
Removal policy: lru
LRU reference age: 23.63 days
```

Store Directory

The directory number, type, and pathname.

FS Block Size

The filesystem block size, determined by the `statfs()` or `statvfs()` system calls. If these functions aren't available or return an error, the block size defaults to 2048 bytes.

The next few lines are actually storage scheme-dependent. For the most part, *ufs*, *aufs*, and *diskd* are very similar and all report the same statistics.

First level subdirectories

The number of first-level subdirectories you told Squid to use on the *cache_dir* line.

Second level subdirectories

The number of second-level subdirectories you told Squid to use on the *cache_dir* line.

Maximum Size

The maximum allowed size for this cache directory.

Current Size

The amount of disk space currently in use.

Percent Used

The percentage of *cache_dir* space currently in use.

Filemap bits in use

Squid uses a bitmap to keep track of file numbers that are allocated and free. This line shows the number and percentage of bits in use. The filemap grows automatically as needed, so don't worry if it shows up as 99% full.

Filesystem Space in use

These numbers come from the `statfs()/statvfs()` system calls. These should be the same values as you'd see from the `df` command. Squid doesn't use these numbers, other than to report them here for your information. Note that these

values may be larger than Current Size, especially if the partition is used for more than Squid's cache.

Filesystem Inodes in use

These numbers also come from `statfs()/statvfs()`. They are present to remind you that running out of inodes is just as bad as running out of free space. Unfortunately, if you run out of inodes, you'll probably be forced to `newfs` the partition.

Flags

Possible values include `SELECTED` and `READ-ONLY`. The `SELECTED` flag means that this particular `cache_dir` was most recently selected by the cache directory selection algorithm (see Chapter 7). The `READ-ONLY` flag means that the cache directory has been marked read-only in the configuration file (see Chapter 7).

Pending operations

This line appears only for `diskd` cache directories. It shows the number of I/O requests dispatched to the `diskd` process that have not yet been acknowledged.

That's the end of the scheme-specific data. The remaining lines are specific to the `cache_dir` replacement algorithm:

Removal policy

Possible values include `lru` (the default) or `heap`. Note that for `heap`, you won't see the algorithm name (`LFU`, `GDSF`, or `LRU`).

LRU reference age

If the removal policy is `lru`, you'll also see this line. It shows the age of the oldest object in the LRU list.

store_check_cachable_stats: storeCheckCachable() Stats

This page displays a table of counters from the `storeCheckCachable()` function. It is called for most responses, just before Squid attempts to open a disk file for writing.



Squid knows that some responses can't be cached, based entirely on the request. These responses aren't included in the `storeCheckCachable()` statistics.

The table includes the following lines:

`no.not_entry_cachable`

The `ENTRY_CACHABLE` flag was cleared for some reason.

`no.release_request`

The `RELEASE_REQUEST` flag was set while reading the response. This may be due to an error (such as receiving a partial response) or to the rules of the transfer protocol.

In some versions of Squid, this counter is always zero because the `storeReleaseRequest()` function always clears the `ENTRY_CACHABLE` bit, causing such objects to be counted as `no.not_entry_cachable` instead.

`no.wrong_content_length`

The actual content length doesn't match the Content-Length header value.

In some versions of Squid, this counter is always zero because `storeReleaseRequest()` is always called if the response size doesn't match the expected content length.

`no.negative_cached`

The `ENTRY_NEGCACHED` flag was set. See the description for `TCP_NEGATIVE_HIT` in Chapter 13.

`no.too_big`

The response body was larger than the *maximum_object_size* value.

`no.too_small`

The response body was smaller than the *minimum_object_size* value.

`no.private_key`

The response has a private cache key, indicating that it can't be shared with other users.

`no.too_many_open_files`

The Squid process was low on free file descriptors.

`no.too_many_open_fds`

Squid had more than *max_open_disk_fds* opened at one time.

`yes.default`

The response was cachable because it did not meet any of the preceding criteria.

store_io: Store IO Interface Stats

This short table contains four lines related to allocating disk storage for a new response. For example:

```
Store IO Interface Stats
create.calls 2825670
create.select_fail 0
create.create_fail 0
create.success 2825670
```

`create.calls`

The number of calls to the function that creates a new disk file.

`create.select_fail`

The number of times that the create operation failed because the *cache_dir* selection algorithm did not select a cache directory. The default selection algorithm, least-load, fails if it thinks all cache directories are too busy.

`create.create_fail`

The number of times that the create operation failed at the storage layer. This may happen if the `open()` call returns an error or if the storage system (e.g., *diskd*) elects to not open a disk file for some reason (e.g., overload condition).

`create.success`

The number of times the create operation succeeded.

pconn: Persistent Connection Utilization Histograms

This page displays two histograms. The first is for client-side persistent connection usage. For example:

Client-side persistent connection counts:

req/ conn	count
0	74292
1	14362705
2	3545955
3	2068486
4	1411423
5	1030023
6	778722
7	603649
8	474592
9	376154
10	301396

On the left is the number of requests per connection. On the right is the number of times a client connection had that many requests. Most likely, you'll see that one request/connection has the highest count and that the counts decrease as the number of requests/connection increases.

The second table has the same information, but for server-side HTTP connections. You should see the same sort of pattern here, with one request/connection having the highest count.

refresh: Refresh Algorithm Statistics

The *refresh* page shows a few tables relating to the freshness of cached objects. Internally, Squid keeps track of the way different modules use the refresh functions. The first table shows how many calls each module has made. The really interesting data is contained in the remaining tables, however.

The HTTP histogram shows the breakdown of freshness checks for client HTTP requests. For example:

HTTP histogram:

Count	%Total	Category
0	0.00	Fresh: request max-stale wildcard
0	0.00	Fresh: request max-stale value

173984	9.76	Fresh: expires time not reached
462757	25.97	Fresh: refresh_pattern last-mod factor percentage
42	0.00	Fresh: refresh_pattern min value
0	0.00	Fresh: refresh_pattern override expires
0	0.00	Fresh: refresh_pattern override lastmod
5521	0.31	Stale: response has must-revalidate
0	0.00	Stale: changed reload into IMS
0	0.00	Stale: request has no-cache directive
470912	26.43	Stale: age exceeds request max-age value
455073	25.54	Stale: expires time reached
65612	3.68	Stale: refresh_pattern max age rule
144706	8.12	Stale: refresh_pattern last-mod factor percentage
3274	0.18	Stale: by default
1781881	100.00	TOTAL

Note, the rules aren't necessarily evaluated in the order in which they appear in the table. Here's what each line means:

Fresh: request max-stale wildcard

Squid considers the cached response fresh because the request includes a max-stale directive without any value. For example:

```
GET /blah... HTTP/1.1
Cache-control: max-stale
```

According to RFC 2616: "If no value is assigned to max-stale, then the client is willing to accept a stale response of any age."

Fresh: request max-stale value

Squid considers the cached response fresh because the request includes a max-stale directive with a particular value, which is larger than the amount of time since the object expired.

Fresh: expires time not reached

Squid considers the cached response fresh because its expiration time has not yet been reached.

Fresh: refresh_pattern last-mod factor percentage

Squid considers the cached response fresh because it matches one of the refresh_pattern rules and has a last-modified factor (LM-factor) value that's less than that specified by the rule. See Chapter 7.

Fresh: refresh_pattern min value

Squid considers the cached response fresh because it matches one of the refresh_pattern rules and its age is less than the min value specified by the rule. See Chapter 7.

Fresh: refresh_pattern override expires

Squid considers the cached response fresh because it matched one of the refresh_pattern rules with the override-expire option. This option causes Squid to give precedence to the refresh_pattern minimum value over the object's expiration time. Note: using the override-expire option is a violation of RFC 2616.

Fresh: refresh_pattern override lastmod

Squid considers the cached response fresh because it matched one of the refresh_pattern rules with the override-lastmod option. This option causes Squid to give precedence to the refresh_pattern minimum value over the LM-factor value. Note: using the override-lastmod option is a violation of RFC 2616.

Stale: response has must-revalidate

Squid considers the cached response stale because it contains a Cache-Control: must-revalidate directive.

Stale: changed reload into IMS

Squid considers the cached response stale because it matches one of the refresh_pattern rules with the reload-into-ims option. With this option, Squid turns a request with Cache-Control: no-cache (or similar) into a cache validation. Note: using the reload-into-ims option is a violation of RFC 2616.

Stale: request has no-cache directive

Squid considers the cached response stale because the request contains a Cache-Control: no-cache directive.

Stale: age exceeds request max-age value

Squid considers the cached response stale because the request has a max-age directive, which is less than the response's age.

Stale: expires time reached

Squid considers the cached response stale because its expiration time has been reached.

Stale: refresh_pattern max age rule

Squid considers the cached response stale because it matches one of the refresh_pattern rules, and its age is greater than the max value specified by the rule.

Stale: refresh_pattern last-mod factor percentage

Squid considers the cached response stale because it matches one of the refresh_pattern rules, and its LM-factor value is greater than the factor specified by the rule.

Stale: by default

Squid considers the cached response stale by default, because it didn't meet any of the other criteria.

Following the HTTP histogram, you'll see the same data for ICP, HTCP, Cache Digests, and On Store.

The On Store table represents freshness checks for responses that are coming into Squid's cache (i.e., cachable misses). Note, however, that Squid does store stale responses (as long as they have a cache validator). Don't be alarmed if you see some stale responses in the On Store histogram.

delay: Delay Pool Levels

This page displays the Delay Pool statistics. Squid has three classes of pools (1, 2, 3) and three types of buckets (aggregate, individual, and network). A class 1 pool has only an aggregate bucket, a class 2 pool has both aggregate and individual, and a class 3 pool has all three.

An aggregate bucket looks like this:

```
Aggregate:
  Max: 16384
  Restore: 4096
  Current: 6144
```

The values are all in bytes. Max is the size of the bucket, which is the number of bytes the bucket can hold. Restore is the number of bytes added to the bucket each second. Current is the number of bytes currently in the bucket. If nobody uses the bytes, the bucket fills until it reaches the maximum size.

An individual bucket is almost the same:

```
Individual:
  Max: 20000
  Restore: 5000
  Current: 1:18760 9:4475 14:20000
```

The only difference is that the Current line displays a number of different values, one for each host number. The *host number* is defined as the last octet of an IPv4 address. In this example, the host numbers are 1, 9, and 14. In a class 2 delay pool, the host numbers from different networks share the same bucket. For example, 192.168.0.1 and 192.168.44.1 both share the bucket for host number 1. In a class 3 pool, however, each network number (third octet) has its own array of individual buckets. Thus, for a class 3 pool, the individual buckets appear this way:

```
Individual:
  Max: 20000
  Rate: 5000
  Current [Network 0]: 1:12000
  Current [Network 44]: 1:17000
```

A network bucket (for class 3 pools only) is similar as well:

```
Network:
  Max: 30000
  Rate: 15000
  Current: 0:3912 7:30000
```

In this case, the Current line shows the current level for each network number (third octet). See Appendix C for more information about Delay Pools.

forward: Request Forwarding Statistics

The table on this page shows how many attempts were made to forward each request, with their results. Upon receiving some status codes, Squid gives up

immediately. For others, however, Squid keeps trying. Each row of the table is a different HTTP status code (200, 401, 404, etc.). Each column is the number of forwarding attempts. The value in each cell is how many requests were forwarded that many times, resulting in the corresponding status code. This information helps developers understand whether or not it makes sense to retry a request after receiving certain types of responses. Here is an example:

Status	try#1	try#2	try#3	try#4	try#5	try#6	try#7	try#8	try#9	try#10
0	1	0	0	0	0	0	0	0	0	0
200	3970083	111015	51185	29002	18242	12097	8191	6080	4490	6140
201	57	0	0	0	0	0	0	0	0	0
202	162	0	0	0	0	0	0	0	0	0
204	1321	11	0	0	0	0	0	0	0	0
206	624288	453	25	9	4	3	0	1	0	0
207	147	0	0	0	0	0	0	0	0	0
300	23	0	0	0	0	0	0	0	0	0
301	23500	25	2	0	0	0	1	0	0	0
302	339332	3806	153	26	6	4	2	3	0	1
303	101	1	0	0	0	0	0	0	0	0
304	772831	3510	125	21	7	8	8	5	3	2
307	7	0	0	0	0	0	0	0	0	0
400	529	1	0	0	0	0	0	0	0	0
401	1559	0	0	0	0	0	0	0	0	0
403	5098	30	1	1	0	0	0	0	0	0
404	100800	216	25	6	7	1	2	4	1	5
405	1	0	0	0	0	0	0	0	0	0
...										

A value of 29,002 in the cell under try#4 and in the row for status 200 means that there were 29,002 times when Squid finally got a successful response after 4 forwarding attempts. If you look at the table, you may see some unknown status codes. Squid keeps track of all status codes up to 600, even those it doesn't know about. See Table 13-1 for the list of codes that Squid does know about.

client_list: Cache Client List

The cache client list shows a handful of statistics for each client IP address accessing Squid, which looks like this:

```
Address: 206.168.0.9
Name: 206.168.0.9
Currently established connections: 0
  ICP Requests 59000
    UDP_HIT          1609  3%
    UDP_MISS        57388 97%
    UDP_INVALID       3    0%
  HTTP Requests 11281
    TCP_HIT          656   6%
    TCP_MISS        3464  31%
    TCP_REFRESH_HIT  4477 40%
    TCP_REFRESH_MISS 767   7%
    TCP_CLIENT_REFRESH_M 397  4%
```

TCP_IMS_HIT	1082	10%
TCP_SWAPFAIL_MISS	7	0%
TCP_NEGATIVE_HIT	13	0%
TCP_MEM_HIT	418	4%

The Address line, obviously, shows the client's IP address. Name is the same, unless you have *log_fqdn* enabled, and the DNS reports a name for the address. The Currently established connections line shows how many HTTP connections are currently open between the client and Squid.

If the client has sent any ICP queries, you'll see a breakdown of the results here. In this example, only 3% of this client's ICP queries were hits. Note, this page doesn't currently include HTCP result statistics. Finally, you'll see a breakdown of HTTP request result codes.

The client database consumes a fair amount of memory, especially if you have a large number of client IP addresses accessing Squid. You can disable the database entirely, thus conserving memory, with the *client_db* directive. Also note that there is no way to clear the counters or to remove entries while Squid is running.

netdb: Network Measurement Database

This page is available only with the *./configure --enable-icmp* option (see Chapter 10). On this page you'll find quite a lot of IP addresses, hostnames, packet counters, and RTT values. It looks something like this:

```
Network DB Statistics:
Network      rcv/sent    RTT  Hops  Hostnames
165.123.34.0    7/ 7    12.7  8.6  onlinebooks.library.upenn.edu
                7/ 7    12.7  8.6  www.library.upenn.edu
                7/ 7    12.7  8.6  digital.library.upenn.edu
rtp.us.ircache.net    17.0  11.0
sj.us.ircache.net     71.0  17.3
209.202.204.0    4/ 4    12.8  10.0  adbuyer3.lycos.com
rtp.us.ircache.net    20.6  15.0
sj.us.ircache.net     77.6  15.0
63.151.139.0    17/ 17   12.8   9.0  www.originlab.com
sj.us.ircache.net     80.0  12.0
209.68.20.0     23/ 23   12.8  11.7  www6.tomshardware.com www.guestbook.nu
rtp.us.ircache.net    34.9  15.1
sj.us.ircache.net     73.9  14.7
```

Each /24 network is listed, in order of increasing round-trip time. You can see how many ICMP pings have been sent and received, the average RTT, and the estimated router hop-count. The Hostnames field shows the hostnames that resolve to addresses within the /24 network. If Squid has ICMP measurements from its neighbors for the network, those are printed as well. In this example, the local cache is closer to all the networks than its neighbors (*rtp.us.ircache.net* and *sj.us.ircache.net*).

asndb: AS Number Database

Although this page is always available, it contains interesting data only if you are using one of the Autonomous System (AS) ACLs, such as *src_as* or *dst_as*.

When you use an AS-based ACL, Squid queries the Routing Arbiter database (*whois.ra.net*) to discover the IP networks associated with the AS number. The results of those queries are displayed on this page. The output looks like this:

Address	AS Numbers
128.98.0.0/16	7
146.80.0.0/16	7
192.5.28.0/24	7
192.5.29.0/24	7
192.5.30.0/24	7
192.107.178.0/24	7
192.135.183.0/24	5637
194.61.177.0/24	7
194.61.180.0/24	7
194.61.183.0/24	7
194.83.162.0/24	7

carp: CARP Information

This page is available only with the `./configure --enable-carp` option and if you have some CARP parents configured. Squid displays a table of all CARP parents, which looks like this:

Hostname	Hash	Multiplier	Factor	Actual
bo1.us.ircache.net	f142425b	0.894427	0.400000	0.527950
bo2.us.ircache.net	12180f04	1.118034	0.600000	0.472050

Hash is the neighbor's hash value from the CARP algorithm. Multiplier is another value used by the algorithm. Factor is taken from the `carp-load-factor` option on the `cache_dir` line in *squid.conf*. Actual is the actual distribution of requests among the CARP parents. Ideally, it should match the Factor value.

server_list: Peer Cache Statistics

This page displays various counters and statistics for your neighbor caches. For example:

```
Sibling      : pa.us.ircache.net/3128/4827
Flags        : htcp
Address[0]   : 192.6.19.203
Status       : Up
AVG RTT      : 14 msec
OPEN CONNS   : 19
LAST QUERY   :      4 seconds ago
LAST REPLY   :      4 seconds ago
PINGs SENT   :    9119
PINGs ACKED  :    9115 100%
FETCHES      :     109 1%
```

```

IGNORED      :      9114 100%
Histogram of PINGS ACKED:
    Misses      9114 100%
    Hits         1    0%
keep-alive ratio: 100%

```

Type

The first line shows the neighbor type (parent, sibling, or multicast group), followed by the hostname and port numbers. The first port number is for HTTP requests, while the second is for ICP or HTCP.

Flags

Here you'll see any of the *cache_peer* options that you may have specified, such as no-query, closest-only, and more. See Chapter 10 for the complete list.

Address[]

This line displays the IP address(es) associated with the hostname. The number in brackets is the number of addresses. Squid stores up to 10 addresses for each neighbor.

Status

The status line indicates whether Squid thinks the neighbor is Up or Down. See Chapter 10.

AVG RTT

This is the running average RTT for ICP/HTCP queries to the neighbor.

OPEN CONNS

This is the number of HTTP connections currently open to the neighbor.

LAST QUERY

This indicates the amount of time since Squid last sent an ICP/HTCP query to the neighbor.

LAST REPLY

This indicates the amount of time since Squid last received an ICP/HTCP reply from the neighbor.

PINGS SENT

The number of ICP/HTCP queries sent to the neighbor.

PINGS ACKED

The number of ICP/HTCP replies received back from the neighbor.

FETCHES

The number of HTTP requests sent to the neighbor. The percentage is based on the PINGS ACKED number. Unfortunately, the FETCHES number counts requests forwarded for any reason (ICP, HTCP, Cache Digests, default parent, etc.). Thus, the percentage doesn't always make sense and may be higher than 100%.

IGNORED

The number of ICP/HTCP replies ignored. The most common reason that Squid ignores an ICP/HTCP reply is that it is too late.

Histogram of PINGS ACKED

Here you'll see a breakdown of ICP/HTCP results. For ICP neighbors, Squid prints the ICP status codes (ICP_HIT, ICP_MISS, etc.). For HTCP neighbors, the only categories are Hits and Misses.

keep-alive ratio

This shows the percentage of times that Squid wanted an HTTP connection to be persistent, and the neighbor agreed. Note, this doesn't indicate anything about whether the connection was actually reused, only that both sides agreed that it could be.

non_peers: List of Unknown Sites Sending ICP messages

This page shows a list of clients that send unauthorized ICP (but not HTCP) queries. The list is the same format as the *Cache Client List* page.

Cache Manager Access Controls

The cache manager interface provides a lot of information. Much of it is sensitive and should be kept private. For example, the *Cache Client List* reveals the IP addresses of users, the *Process Filedescriptor Allocation* page shows URIs currently being requested, and the *Current Squid Configuration* displays the values from *squid.conf*, including passwords and access control rules. To keep unwanted visitors from browsing the cache manager pages, you must carefully configure access to it.

http_access

All cache manager requests use the pseudo-protocol scheme `cache_object`. The best way to protect the cache manager is restrict the IP addresses allowed to make `cache_object` requests. The default *squid.conf* contains these lines:

```
acl Manager proto cache_object
acl Localhost src 127.0.0.1/255.255.255.255
http_access allow Manager Localhost
http_access deny Manager
```

Thus, cache manager requests from the local host (127.0.0.1) are allowed, but all others are denied. If you have additional trusted hosts, you may want to add them to the access rules also. Make sure these lines are at the top of your *http_access* rules.

cachemgr_passwd

You may also want to modify the default *cachemgr_passwd* settings. Some of the cache manager pages require a password, so you won't be able to view those until you add one. For example, if you want to use the *Current Squid Configuration* page, you must assign it a password:

```
cachemgr_passwd JeckCy config
```

You can have a number of different passwords, but each action may have only one password. You may want to use a different password for less sensitive pages:

```
cachemgr_passwd byDroth filedescriptors client_list netdb
```

To disable a cache manager action, use `disable` as the password:

```
cachemgr_passwd disable netdb
```

To enable the sensitive actions without requiring a password, use `none`:

```
cachemgr_passwd none offline_toggle
```

If you want to give the same password to all actions, use the keyword `all`:

```
cachemgr_passwd Knoujush all
```

When using the command-line cache manager interface (e.g., `squidclient`), put an `@` sign and the password after the action name. For example:

```
squidclient mgr:objects@byDroth | less
```

Note that cache manager passwords aren't printed when you request the *Current Squid Configuration* page (see the section "config: Current Squid Configuration").

cachemgr.cgi

If you use `cachemgr.cgi`, the IP address of your HTTP server must be able to make cache manager requests to Squid. This opens up a back-door security hole. Anyone who can execute the CGI program on your server will be able to view the cache manager pages. The passwords described earlier can help, but you may also want to install access controls on your HTTP server so that only certain people can execute `cachemgr.cgi`.

The main `cachemgr.cgi` page has a form with *Username* and *Password* fields. The username is purely informational. If you have multiple administrators in your organization, each person can enter their own name for auditing purposes.

If you leave the password field blank, the password-protected pages are disabled. Entering a password activates links for those pages. `cachemgr.cgi` is stateless, so the password must be included as a URI parameter in links. Furthermore, the password encoding scheme isn't very sophisticated and trivial to break. Because many applications (such as Squid!) log the URIs of HTTP requests, your cache-manager password may be logged or even observed by an untrusted third party. If you really want to keep your cache manager passwords secret, never use them with `cachemgr.cgi` or from any remote system.

Reasons to Dislike the Cache Manager

The cache manager interface leaves much to be desired. It has a very unpolished feel. Novice administrators will probably find it difficult to use and understand. One of the first problems you might notice is that the menu (or table of contents) is

unorganized. There is no logical order or grouping. The first items in the list provide low-level information primarily meant for developers. Currently, the order is determined by the initialization sequence in the source code.

The output is often ugly. The `cachemgr.cgi` program renders very bland-looking HTML pages. There are no icons or graphics of any kind. Furthermore, many of the pages are simply presented as unformatted text. `cachemgr.cgi` doesn't do much more than format tab-delimited lines as HTML tables and put `<A>` tags around some URIs. Some of the cache manager pages are structured so that the output can be easily parsed by computer programs, rather than humans.

By today's standards, the cache manager has very weak security. You are essentially forced to use address-based controls and cleartext passwords. If you allow cache manager requests only from *localhost*, and your system security is good, you'll be relatively safe.

Squid-RRD

I personally use the cache manager to populate a number of RRDTool databases (<http://www.rrdtool.com/>). RRDTool is nice package for storing and displaying time-series data. It allows you to archive data at different time scales (e.g., days, weeks, months, years) in a database that doesn't grow in size over time.

I use a Perl script that runs every five minutes from cron. It issues cache manager requests for a number of pages and extracts the values that I am interested in. These values are stored in the RRD files.

RRDTool also generates nice-looking graphs, from either a CGI script or standalone program. I use the CGI program and check the graphs at least daily. See Figure 14-2 for some samples from one of my own Squid boxes.

You can find my scripts and instructions for integrating the cache manager and RRDTool at <http://www.squid-cache.org/~wessels/squid-rrd/>.

Using SNMP

Squid has a built-in SNMP agent that you can query with various SNMP client tools. It allows you to collect a few basic statistics from Squid. Unfortunately, the Squid MIB has not evolved much since its initial implementation. Many of the parameters that you'd like to monitor aren't available through the SNMP MIB. Perhaps this will be rectified in a future version.

To enable SNMP in Squid, use the `--enable-snmp` option when running `./configure` and recompile if necessary. Squid uses UDP port 3401 for SNMP by default. You can use a different port by setting the `snmp_port` directive.

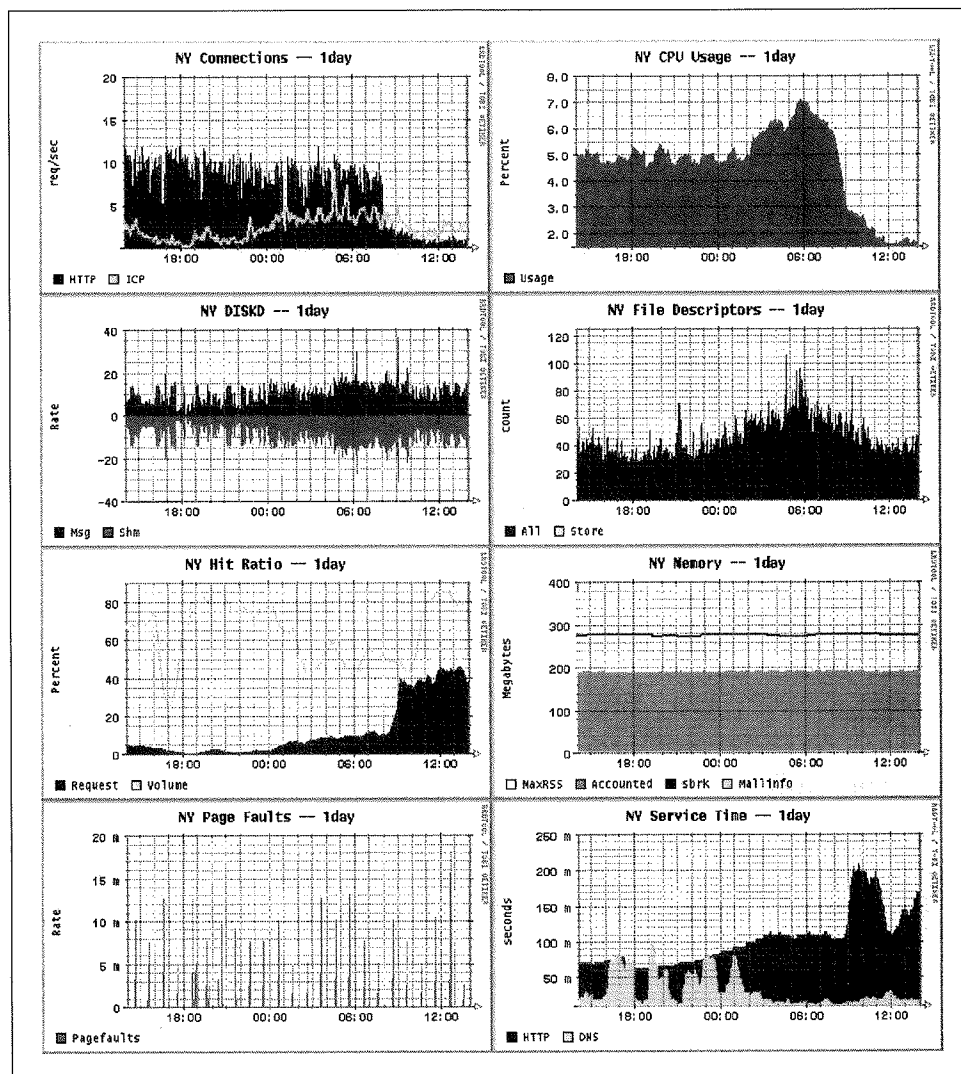


Figure 14-2. Some sample RRD graphs from RRDTool and cache manager data

Use the `snmp_access` access list and `snmp_community` ACL type to define an access policy for the SNMP agent. For example:

```
acl Snmppublic snmp_community public
acl Adminhost src 192.168.1.1
snmp_access allow Adminhost Snmppublic
```

In this case, Squid accepts SNMP requests from 192.168.1.1 with the community name set to public.

Using snmpwalk and snmpget

The NET-SNMP package (<http://net-snmp.sourceforge.net/>) provides a good implementation of the `snmpwalk` and `snmpget` command-line tools for Unix. The former walks through an SNMP MIB tree, displaying every value, while the latter prints the value for a single MIB object.

After installing NET-SNMP, copy the Squid MIB file to the directory where the utilities can find it. By default, this is the `/usr/local/share/snmp/mibs` directory:

```
# cp squid-2.5.STABLE4/src/mib.txt /usr/local/share/snmp/mibs/SQUID-MIB.txt
# chmod 644 /usr/local/share/snmp/mibs/SQUID-MIB.txt
```

You should then be able to use the `snmpget` command. Note that Squid is an SNMPv1 agent:

```
% snmpget -v 1 -c public -m SQUID-MIB localhost:3401 cacheDnsSvcTime.5
SQUID-MIB::cacheDnsSvcTime.5 = INTEGER: 44
```

If you want to see the entire Squid MIB tree, use `snmpwalk`. The `-Cc` option tells `snmpwalk` to ignore nonincreasing OIDs:

```
% snmpwalk -v 1 -c public -m SQUID-MIB -Cc localhost:3401 squid | less
```

If you can't get the Squid MIB installed so that `snmpwalk` sees it, you can use the numeric OID value instead:

```
% snmpwalk -v 1 -c public -m SQUID-MIB -Cc localhost:3401 .1.3.6.1.4.1.3495.1 | less
```

The Squid MIB

In this section, I provide a brief description for each OID in the Squid MIB, which lives in the global MIB tree under *iso.org.dod.internet.private.enterprises.nlanr.squid*, or *.1.3.6.1.4.1.3495.1*. The full MIB names, such as *cachePerf.cacheProtoStats.cache-MedianSvcTable.cacheMedianSvcEntry.cacheHttpMissSvcTime.60*, take up too much space on the page. Instead, I'll just use the last nonnumeric component of the OID name, which is unique.

cacheSysVMsize

The amount of memory (in kilobytes) currently used to store in-memory objects.
For example:

```
SQUID-MIB::cacheSysVMsize = INTEGER: 10224
```

cacheSysStorage

The amount of disk space (in kilobytes) currently used to store on-disk objects.
For example:

```
SQUID-MIB::cacheSysStorage = INTEGER: 19347723
```

cacheUptime

The amount of time (number of seconds) since Squid was started.

```
SQUID-MIB::cacheUptime = Timeticks: (33239630) 3 days, 20:19:56.30
```

cacheAdmin

The email address, or name, of the cache administrator. For example:

```
SQUID-MIB::cacheAdmin = STRING: wessels@bo2.us.ircache.net
```

cacheSoftware

The name of the application. For example:

```
SQUID-MIB::cacheSoftware = STRING: squid
```

cacheVersionId

The application's version identification. For example:

```
SQUID-MIB::cacheVersionId = STRING: "2.5.STABLE4"
```

cacheLoggingFacility

The current debugging levels, from the *debug_options* directive. For example:

```
SQUID-MIB::cacheLoggingFacility = STRING: ALL,1
```

cacheMemMaxSize

The value of the *cache_mem* directive, in megabytes. For example:

```
SQUID-MIB::cacheMemMaxSize = INTEGER: 10
```

cacheSwapMaxSize

The total amount of disk storage, in megabytes, taken from the sum of all *cache_dir* lines. For example:

```
SQUID-MIB::cacheSwapMaxSize = INTEGER: 21000
```

cacheSwapHighWM

The high watermark percentage for disk storage, taken from the *cache_swap_high* directive. For example:

```
SQUID-MIB::cacheSwapHighWM = INTEGER: 95
```

cacheSwapLowWM

The low watermark percentage for disk storage, taken from the *cache_swap_low* directive. For example:

```
SQUID-MIB::cacheSwapLowWM = INTEGER: 90
```

cacheSysPageFaults

The number of page faults for the Squid process since it was started. (See "Page faults with physical i/o" in the section "info: General Runtime Information.") For example:

```
SQUID-MIB::cacheSysPageFaults = Counter32: 9
```

cacheSysNumReads

The number of times this process called `read()` on HTTP sockets connected to origin servers and neighbor caches. For example:

```
SQUID-MIB::cacheSysNumReads = Counter32: 15941979
```

cacheMemUsage

The amount of memory allocated by the memory pooling routines. Not the same as the total memory used by Squid. (See "Total accounted" in the section "info: General Runtime Information.") For example:

```
SQUID-MIB::cacheMemUsage = INTEGER: 143709
```

cacheCpuTime

The amount of CPU time, in seconds, accumulated by the Squid process. For example:

```
SQUID-MIB::cacheCpuTime = INTEGER: 79313
```

cacheCpuUsage

The mean CPU utilization, as a percentage, since Squid was started. Unfortunately, since this value is an integer, any graphs that you make will be “quantized.” For example:

```
SQUID-MIB::cacheCpuUsage = INTEGER: 23
```

cacheMaxResSize

The maximum resident set size, in kilobytes, for the Squid process. (See “Maximum Resident Size” in the section “info: General Runtime Information.”) For example:

```
SQUID-MIB::cacheMaxResSize = INTEGER: 219128
```

cacheNumObjCount

The total number of objects currently in the cache. For example:

```
SQUID-MIB::cacheNumObjCount = Counter32: 1717181
```

cacheCurrentLRUExpiration

Current versions of Squid don’t have a global LRU expiration age value, so this is always reported as zero. For example:

```
SQUID-MIB::cacheCurrentLRUExpiration = Timeticks: (0) 0:00:00.00
```

cacheCurrentUnlinkRequests

The number of files given to the external unlinkd process for removal. Note that Squid doesn’t use unlinkd with the *diskd* and *aufs* storage schemes. For example:

```
SQUID-MIB::cacheCurrentUnlinkRequests = Counter32: 0
```

cacheCurrentUnusedFDescrCnt

The current number of available (unused) file descriptors. For example:

```
SQUID-MIB::cacheCurrentUnusedFDescrCnt = Gauge32: 7253
```

cacheCurrentResFileDescrCnt

The number of reserved file descriptors. (See “Reserved number of file descriptors” in the section “info: General Runtime Information.”) For example:

```
SQUID-MIB::cacheCurrentResFileDescrCnt = Gauge32: 100
```

cacheProtoClientHttpRequests

The total number of HTTP requests received from cache clients. For example:

```
SQUID-MIB::cacheProtoClientHttpRequests = Counter32: 7277019
```

cacheHttpHits

The number of client requests that were cache hits. For example:

```
SQUID-MIB::cacheHttpHits = Counter32: 2526484
```

cacheHttpErrors

The number of client requests that resulted in an error. For example:

SQUID-MIB::cacheHttpErrors = Counter32: 0

cacheHttpInKb

The amount of network traffic, in kilobytes, read from cache clients. For example:

SQUID-MIB::cacheHttpInKb = Counter32: 4231883

cacheHttpOutKb

The amount of network traffic, in kilobytes, written to cache clients. For example:

SQUID-MIB::cacheHttpOutKb = Counter32: 56894945

cacheIcpPktsSent

The number of ICP messages (both queries and replies) sent to neighbors. For example:

SQUID-MIB::cacheIcpPktsSent = Counter32: 5296120

cacheIcpPktsRecv

The number of ICP messages (both queries and replies) received from neighbors. For example:

SQUID-MIB::cacheIcpPktsRecv = Counter32: 5271238

cacheIcpKbSent

The amount of network traffic, in kilobytes, used for ICP messages sent to neighbors, not including UDP and IP headers. For example:

SQUID-MIB::cacheIcpKbSent = Counter32: 428112

cacheIcpKbRecv

The amount of network traffic, in kilobytes, used for ICP messages received from neighbors, not including UDP and IP headers. For example:

SQUID-MIB::cacheIcpKbRecv = Counter32: 447762

cacheServerRequests

The number of requests forwarded to origin servers and neighbor caches. For example:

SQUID-MIB::cacheServerRequests = INTEGER: 5338305

cacheServerErrors

The number of errors received from origin servers and neighbor caches. Currently unimplemented and always reported as zero. For example:

SQUID-MIB::cacheServerErrors = INTEGER: 0

cacheServerInKb

The amount of network traffic, in kilobytes, read from origin servers and neighbor caches. For example:

SQUID-MIB::cacheServerInKb = Counter32: 49196559

cacheServerOutKb

The amount of network traffic, in kilobytes, written to origin servers and neighbor caches. For example:

SQUID-MIB::cacheServerOutKb = Counter32: 3404717

cacheCurrentSwapSize

The amount of disk space, in kilobytes, currently in use by Squid. Compare to cacheSysStorage. For example:

```
SQUID-MIB::cacheCurrentSwapSize = Counter32: 19347723
```

cacheClients

The number of clients that sent HTTP requests to Squid since it was started. For example:

```
SQUID-MIB::cacheClients = Counter32: 498
```

cacheMedianTime.X

These OIDs report the time intervals, in minutes, over which median values are computed for subsequent OIDs. The value is the same as the last number of the OID. For example:

```
SQUID-MIB::cacheMedianTime.1 = INTEGER: 1  
SQUID-MIB::cacheMedianTime.5 = INTEGER: 5  
SQUID-MIB::cacheMedianTime.60 = INTEGER: 60
```

cacheHttpAllSvcTime.X

The 1-, 5-, and 60-minute median service time values, in milliseconds, for all client HTTP requests. For example:

```
SQUID-MIB::cacheHttpAllSvcTime.1 = INTEGER: 78  
SQUID-MIB::cacheHttpAllSvcTime.5 = INTEGER: 70  
SQUID-MIB::cacheHttpAllSvcTime.60 = INTEGER: 56
```

cacheHttpMissSvcTime.X

The 1-, 5-, and 60-minute median service time values for cache misses. For example:

```
SQUID-MIB::cacheHttpMissSvcTime.1 = INTEGER: 114  
SQUID-MIB::cacheHttpMissSvcTime.5 = INTEGER: 87  
SQUID-MIB::cacheHttpMissSvcTime.60 = INTEGER: 74
```

cacheHttpNmSvcTime.X

The 1-, 5-, and 60-minute median service time values for requests logged as TCP_IMS_HIT. (See “Not-Modified Replies” in the section “info: General Runtime Information.”) For example:

```
SQUID-MIB::cacheHttpNmSvcTime.1 = INTEGER: 12  
SQUID-MIB::cacheHttpNmSvcTime.5 = INTEGER: 34  
SQUID-MIB::cacheHttpNmSvcTime.60 = INTEGER: 32
```

cacheHttpHitSvcTime.X

The 1-, 5-, and 60-minute median service time values for cache hits, logged as TCP_HIT. For example:

```
SQUID-MIB::cacheHttpHitSvcTime.1 = INTEGER: 45  
SQUID-MIB::cacheHttpHitSvcTime.5 = INTEGER: 45  
SQUID-MIB::cacheHttpHitSvcTime.60 = INTEGER: 40
```

cacheIcpQuerySvcTime.X

The 1-, 5-, and 60-minute service time values for ICP queries sent by Squid (the time elapsed between sending your query and receiving a neighbor's reply). For example:

```
SQUID-MIB::cacheIcpQuerySvcTime.1 = INTEGER: 0
SQUID-MIB::cacheIcpQuerySvcTime.5 = INTEGER: 0
SQUID-MIB::cacheIcpQuerySvcTime.60 = INTEGER: 3563
```

cacheIcpReplySvcTime.X

The 1-, 5-, and 60-minute median service time values for ICP queries received by Squid. In current implementations, these are always zero because processing occurs faster than the process clock is updated. For example:

```
SQUID-MIB::cacheIcpReplySvcTime.1 = INTEGER: 0
SQUID-MIB::cacheIcpReplySvcTime.5 = INTEGER: 0
SQUID-MIB::cacheIcpReplySvcTime.60 = INTEGER: 0
```

cacheDnsSvcTime.X

The 1-, 5-, and 60-minute median service time values for Squid's DNS queries. For example:

```
SQUID-MIB::cacheDnsSvcTime.1 = INTEGER: 40
SQUID-MIB::cacheDnsSvcTime.5 = INTEGER: 42
SQUID-MIB::cacheDnsSvcTime.60 = INTEGER: 42
```

cacheRequestHitRatio.X

Squid's cache hit ratio (percentage) over the last 1, 5, and 60 minutes. For example:

```
SQUID-MIB::cacheRequestHitRatio.1 = INTEGER: 16
SQUID-MIB::cacheRequestHitRatio.5 = INTEGER: 18
SQUID-MIB::cacheRequestHitRatio.60 = INTEGER: 22
```

cacheRequestByteRatio.X

Squid's byte hit ratio (percentage) over the last 1, 5, and 60 minutes. For example:

```
SQUID-MIB::cacheRequestByteRatio.1 = INTEGER: 73
SQUID-MIB::cacheRequestByteRatio.5 = INTEGER: 43
SQUID-MIB::cacheRequestByteRatio.60 = INTEGER: 34
```

cacheIpEntries

The number of entries in Squid's IP (name-to-address) cache. For example:

```
SQUID-MIB::cacheIpEntries = Gauge32: 10033
```

cacheIpRequests

The number of requests received by Squid's IP cache. For example:

```
SQUID-MIB::cacheIpRequests = Counter32: 8195627
```

cacheIpHits

The number of lookups that were hits in the IP cache. For example:

```
SQUID-MIB::cacheIpHits = Counter32: 6040658
```

If the ratio of hits to requests is less than 60–75%, you may want to increase the size of your IP cache.

cacheIpPendingHits

Always zero in the current implementation. For example:

```
SQUID-MIB::cacheIpPendingHits = Gauge32: 0
```

Older versions of Squid had the notion of IP cache hits for outstanding queries.

cacheIpNegativeHits

The number of lookups that were negative hits in the IP cache. Certain failed queries may be negatively cached for an amount of time determined by the *negative_dns_ttl* directive. For example:

```
SQUID-MIB::cacheIpNegativeHits = Counter32: 49433
```

cacheIpMisses

The number of IP cache misses. For example:

```
SQUID-MIB::cacheIpMisses = Counter32: 1807438
```

cacheBlockingGetHostByName

Always zero in the current implementation. For example:

```
SQUID-MIB::cacheBlockingGetHostByName = Counter32: 0
```

Older versions occasionally called the `gethostbyname()` function if the IP cache couldn't provide an answer.

cacheAttemptReleaseLckEntries

Always zero in the current implementation. Older versions would, in some cases, want to release locked IP cache entries. For example:

```
SQUID-MIB::cacheAttemptReleaseLckEntries = Counter32: 0
```

cacheFqdnEntries

The number of entries in the FQDN (address-to-name) cache. For example:

```
SQUID-MIB::cacheFqdnEntries = Gauge32: 1
```

cacheFqdnRequests

The number of requests to the FQDN cache. For example:

```
SQUID-MIB::cacheFqdnRequests = Counter32: 0
```

cacheFqdnHits

The number of FQDN cache requests satisfied as hits. For example:

```
SQUID-MIB::cacheFqdnHits = Counter32: 0
```

cacheFqdnPendingHits

Always zero in the current implementation. For example:

```
SQUID-MIB::cacheFqdnPendingHits = Gauge32: 0
```

cacheFqdnNegativeHits

The number of FQDN requests satisfied as negative cache hits. For example:

```
SQUID-MIB::cacheFqdnNegativeHits = Counter32: 0
```

cacheFqdnMisses

The number of FQDN cache misses. For example:

```
SQUID-MIB::cacheFqdnMisses = Counter32: 0
```

cacheBlockingGetHostByAddr

Always zero in the current implementation. For example:

```
SQUID-MIB::cacheBlockingGetHostByAddr = Counter32: 0
```

cacheDnsRequests

The number of DNS queries made by Squid. This counter is reset each time you reconfigure the running Squid process. For example:

```
SQUID-MIB::cacheDnsRequests = Counter32: 3262
```

cacheDnsReplies

The number of DNS replies received by Squid. This counter is reset each time you reconfigure the running Squid process. For example:

```
SQUID-MIB::cacheDnsReplies = Counter32: 2440
```

cacheDnsNumberServers

When using internal DNS (the default), this OID reports the number of nameservers that Squid knows about. For external DNS, it reports the number of (running) dnsserver helper processes. For example:

```
SQUID-MIB::cacheDnsNumberServers = Counter32: 2
```

cachePeerName.A.B.C.D

This, and the next group of OIDs, come from the list of neighbor caches. (See the section “server_list: Peer Cache Statistics.”) These OIDs are indexed by the IPv4 address of the peer. This particular OID returns the neighbor cache’s host-name. For example:

```
SQUID-MIB::cachePeerName.192.203.230.19 = STRING: sv.us.ircache.net
```

cachePeerAddr.A.B.C.D

This is the IP address of the peer, which, of course, you already know from the OID itself. For example:

```
SQUID-MIB::cachePeerAddr.192.203.230.19 = IPAddress: 192.203.230.19
```

cachePeerPortHttp.A.B.C.D

This is the neighbor cache’s HTTP port number. For example:

```
SQUID-MIB::cachePeerPortHttp.192.203.230.19 = INTEGER: 3128
```

cachePeerPortIcp.A.B.C.D

This is the neighbor cache’s ICP or HTCP port number. For example:

```
SQUID-MIB::cachePeerPortIcp.192.203.230.19 = INTEGER: 3130
```

cachePeerType.A.B.C.D

The type of the neighbor: 1 for sibling, 2 for parent, and 3 for multicast. For example:

```
SQUID-MIB::cachePeerType.192.203.230.19 = INTEGER: 1
```

cachePeerState.A.B.C.D

The state of the peer: 1 for up, 0 for down. (See Chapter 10.) For example:

```
SQUID-MIB::cachePeerState.192.203.230.19 = INTEGER: 1
```

cachePeerPingsSent.A.B.C.D

The number of ICP/HTCP queries sent to the neighbor. For example:

SQUID-MIB::cachePeerPingsSent.192.203.230.19 = Counter32: 924

cachePeerPingsAcks.A.B.C.D

The number of ICP/HTCP queries received from the neighbor. For example:

SQUID-MIB::cachePeerPingsAcks.192.203.230.19 = Counter32: 901

cachePeerFetches.A.B.C.D

The number of HTTP requests sent to the neighbor. (See the discussion about FETCHES in “server_list: Peer Cache Statistics.”) For example:

SQUID-MIB::cachePeerFetches.192.203.230.19 = Counter32: 34

cachePeerRtt.A.B.C.D

The average round-trip time for ICP/HTCP queries to this peer. For example:

SQUID-MIB::cachePeerRtt.192.203.230.19 = INTEGER: 26

cachePeerIgnored.A.B.C.D

The number of ICP/HTCP replies that Squid ignored. (See the discussion about IGNORED in “server_list: Peer Cache Statistics.”) For example:

SQUID-MIB::cachePeerIgnored.192.203.230.19 = Counter32: 201

cachePeerKeepAlSent.A.B.C.D

The number of HTTP requests sent to the neighbor with a request to keep the connection open. For example:

SQUID-MIB::cachePeerKeepAlSent.192.203.230.19 = Counter32: 34

cachePeerKeepAlRecv.A.B.C.D

The number of HTTP replies received from the neighbor with a request to keep the connection open. For example:

SQUID-MIB::cachePeerKeepAlRecv.192.203.230.19 = Counter32: 34

cacheClientAddr.A.B.C.D

The cacheClientAddr OIDs come from the same database as the *Cache Client List* (see the section “client_list: Cache Client List”). This particular OID’s value is the IPv4 address, just like the last four octets of the OID itself. For example:

SQUID-MIB::cacheClientAddr.206.168.0.9 = IpAddress: 206.168.0.9

cacheClientHttpRequests.A.B.C.D

The number of HTTP requests received from this client. For example:

SQUID-MIB::cacheClientHttpRequests.206.168.0.9 = Counter32: 108281

cacheClientHttpKb.A.B.C.D

The amount of traffic, in kilobytes, sent to this client. For example:

SQUID-MIB::cacheClientHttpKb.206.168.0.9 = Counter32: 921447

cacheClientHttpHits.A.B.C.D

The number of cache hits sent to this client. For example:

SQUID-MIB::cacheClientHttpHits.206.168.0.9 = Counter32: 32365

cacheClientHTTPHitKb.A.B.C.D

The amount of traffic, in kilobytes, sent to this client for cache hits. For example:

SQUID-MIB::cacheClientHTTPHitKb.206.168.0.9 = Counter32: 141638

cacheClientIcpRequests.A.B.C.D

The number of ICP (but not HTCP) queries received from this client. For example:

SQUID-MIB::cacheClientIcpRequests.206.168.0.9 = Counter32: 79120

cacheClientIcpKb.A.B.C.D

The amount of traffic, in kilobytes, received from this client in ICP queries. For example:

SQUID-MIB::cacheClientIcpKb.206.168.0.9 = Counter32: 5986

cacheClientIcpHits.A.B.C.D

The number of ICP_HIT replies sent to this client. For example:

SQUID-MIB::cacheClientIcpHits.206.168.0.9 = Counter32: 21897

cacheClientIcpHitKb.A.B.C.D

The amount of traffic, in kilobytes, sent to this client for ICP_HIT messages. A somewhat silly measurement because ICP_HIT and ICP_MISS messages have the same size. However, old versions of Squid used the now-obsolete ICP_HIT_OBJ opcode, which included the object content. For example:

SQUID-MIB::cacheClientIcpHitKb.206.168.0.9 = Counter32: 1679

Exercises

- Write a shell script that uses `squidclient` to collect and save the total number of HTTP requests and the five-minute median overall response time.
- Write a shell script to periodically retrieve and archive the running configuration. It should also compare the current and most recent configurations and email you the changes, if any.
- Download, compile, and install the NET-SNMP package. Use `snmpwalk` to view Squid's entire MIB tree.
- Create and deploy a simple redirector (Chapter 11) that sleeps for 250 milliseconds on each request. Watch the cache manager's redirector page as Squid runs.

CHAPTER 15

Server Accelerator Mode

Throughout most of this book, I've been talking about Squid as a client-side caching proxy. However, with just a few special *squid.conf* settings, Squid is able to function as an origin server accelerator as well. In this mode, it accepts normal HTTP requests and forwards cache misses to the real origin server (or *backend server*). In the parlance of RFC 3040, Squid is operating as a *surrogate*. This configuration is similar to what I talked about in Chapter 9. The primary difference is that, as a surrogate, Squid accepts requests for one, or maybe a few, origin server(s), rather than any and all origins. HTTP interception isn't required for server acceleration.

As the name implies, server acceleration is generally used as a technique to improve the performance of slow, or heavily loaded, backend servers. It works well because origin servers tend to have a relatively small hot set. Most likely, the objects responsible for 90% of origin server traffic can fit entirely in memory. Depending on your particular backend server software and configuration, Squid may be able to serve requests much faster.

Security is another good reason to consider Squid as a surrogate. Think of Squid as a dedicated firewall in front of your origin server. The Squid source code is too large to be trusted as completely secure. However, you may sleep better with Squid protecting your backend server. It is simply a cache, so it doesn't permanently store the source of your data. If the Squid box is attacked or compromised, you won't lose any data. You may find it easier to secure a system running Squid than the system running your backend server application(s).

You might also be interested in server acceleration to implement load balancing. If your origin server runs on expensive boxes, you can save money by deploying Squid on a number of cheaper boxes. By placing Squid at a number of different locations, you can even build your own content delivery network (CDN).

Overview

Assuming that you already have an origin server in place, you need to move it to a different IP address or TCP port. For example, you can (1) install Squid on a separate machine, (2) give the origin server a new IP address, and (3) give Squid the origin server's old IP address. In the interest of security, you can use non-globally routable addresses (i.e., from RFC 1918) on the link between Squid and the backend server. See Figure 15-1.

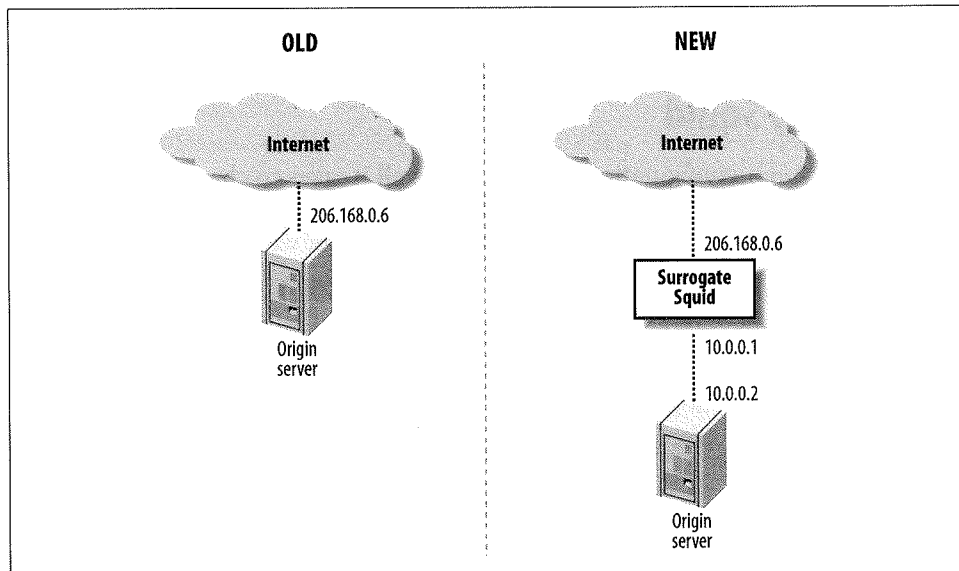


Figure 15-1. How to replace your origin server with Squid

Another option is to configure Squid for HTTP interception, as described in Chapter 9. For example, you can configure the origin server's nearest router or switch to intercept HTTP requests and divert them to Squid.

If you don't have the resources to put Squid on a dedicated system, you can run it alongside the HTTP server. However, both applications can't share the same IP address and port number. You need to make the backend server bind to a different address (e.g., 127.0.0.1) or move it to another port number. It might seem easiest to change the port number, but I recommend changing the IP address instead.

Changing the port number can be problematic. For example, when the backend server generates an error message, it may expose the "wrong" port. Even worse, if the server generates an HTTP redirect, it typically appends the nonstandard port number to the Location URI:

```
HTTP/1.1 301 Moved Permanently
Date: Mon, 29 Sep 2003 03:36:13 GMT
```

```
Server: Apache/1.3.26 (Unix)
Location: http://www.squid-cache.org:81/Doc/
```

If a client receives this response, it makes a connection to the nonstandard port (81), thus bypassing the server accelerator. If you must run Squid on the same host as your backend server, it is better to tell the backend server to listen on the loopback address (127.0.0.1). With Apache, you'd do it like this:

```
BindAddress 127.0.0.1
ServerName www.squid-cache.org
```

Once you've decided how to relocate your origin server, the next step is to configure Squid.

Configuring Squid

Technically, a single configuration file directive is all it takes to change Squid from a caching proxy into a surrogate. Unfortunately, life is never quite that simple. Due to the myriad of ways that different organizations design their web services, Squid has a number of directives to worry about.

http_port

Most likely, Squid is acting as a surrogate for your HTTP server on port 80. Use the *http_port* directive to make Squid listen on that port:

```
http_port 80
```

If you want Squid to act as surrogate and a caching proxy at the same time, list both port numbers:

```
http_port 80
http_port 3128
```

You can configure your clients to send their proxy requests to port 80 as well, but I strongly discourage that. By using separate ports, you'll find it easier to migrate the two services to separate boxes later if it becomes necessary.

https_port

You can configure Squid to terminate encrypted HTTP (SSL and TLS) connections. This feature requires the `--enable-ssl` option when running `./configure`. In this mode, Squid decrypts SSL/TLS connections from clients and forwards unencrypted requests to your backend server. The *https_port* directive has the following format:

```
https_port [host:]port cert=certificate.pem [key=key.pem] [version=1-4]
          [cipher=list] [options=list]
```

The `cert` and `key` arguments are pathnames to OpenSSL-compatible certificate and private key files. If you omit the `key` argument, the OpenSSL library looks for the private key in the certificate file.

The (optional) `version` argument specifies your requirements for various SSL and TLS protocols to support: 1=automatic, 2=SSLv2 only, 3=SSLv3 only, 4=TLSv1 only.

The (optional) `cipher` argument is a colon-separated list of ciphers. Squid simply passes it to the `SSL_CTX_set_cipher_list()` function. For more information, read the `ciphers(1)` manpage on your system or try running: `openssl ciphers`.

The (optional) `options` argument is a colon-separated list of OpenSSL options. Squid simply passes these to the `SSL_CTX_set_options()` function. For more information, read the `SSL_CTX_set_options(3)` manpage on your system.

Here are a few example `https_port` lines:

```
https_port 443 cert=/usr/local/etc/certs/squid.cert
https_port 443 cert=/usr/local/etc/certs/squid.cert version=2
https_port 443 cert=/usr/local/etc/certs/squid.cert cipher=SHA1
https_port 443 cert=/usr/local/etc/certs/squid.cert options=MICROSOFT_SESS_ID_BUG
```

httpd_accel_host

This is where you tell Squid the IP address, or hostname, of the backend server. If you use the loopback trick described previously, you write:

```
httpd_accel_host 127.0.0.1
```

Squid then prepends this value to partial URIs that get accelerated. It also changes the value of the `Host` header.* For example, if the client makes this request:

```
GET /index.html HTTP/1.1
Host: squidbook.org
```

Squid turns it into this request:

```
GET http://127.0.0.1/index.html HTTP/1.1
Host: 127.0.0.1
```

As you can see, the request no longer contains any information that indicates the request is for `squidbook.org`. This shouldn't be a problem as long as the backend server isn't configured for virtual hosting of multiple domains.

If you want Squid to use the origin server's hostname, you can put it in the `httpd_accel_host` directive:

```
httpd_accel_host squidbook.org
```

* Technically, the `Host` header is changed only in requests Squid forwards to the backend server (cache misses).

Then the request is as follows:

```
GET http://squidbook.org/index.html HTTP/1.1
Host: squidbook.org
```

Another option is to enable the *httpd_accel_uses_host_header* directive. Squid then inserts the Host header value into the URI for most requests, and the *httpd_accel_host* value is used only for requests that lack a Host header.

When you use a hostname, Squid goes through the normal steps to look up its IP address. Because you want the hostname to resolve to two different addresses (one for clients connecting to Squid and another for Squid connecting to the backend server), you should also add a static DNS entry to your system's */etc/hosts* file. For example:

```
127.0.0.1      squidbook.org
```

You might want to use a redirector instead. For example, you can write a simple Perl program that changes `http://squidbook.org/...` to `http://127.0.0.1/...`. See Chapter 11 for the nuts and bolts of redirecting client requests.

The *httpd_accel_host* directive has a special value. If you set it to `virtual`, Squid inserts the origin server's IP address into the URI when the Host header is missing. This feature is useful only when using HTTP interception, however.

httpd_accel_port

This directive tells Squid the port number of the backend server. It is 80 by default. You won't need to change this unless the backend server is running on a different port. Here is an example:

```
httpd_accel_port 8080
```

If you are accelerating origin servers on multiple ports, you can use the value 0. In this case, Squid takes the port number from the Host header.

httpd_accel_uses_host_header

This directive controls how Squid determines the hostname it inserts into accelerated URIs. If enabled, the request's Host header value takes precedence over *httpd_accel_host*.

The *httpd_accel_uses_host_header* directive goes hand in hand with virtual domain hosting on the backend server. You can leave it disabled if the backend server is handling only one domain. If, on the other hand, you are accelerating multiple origin server names, turn it on:

```
httpd_accel_uses_host_header on
```

If you enable *httpd_accel_uses_host_header*, be sure to install some access controls as described later in this chapter. To understand why, consider this configuration:

```
httpd_accel_host does.not.exist
httpd_accel_uses_host_header on
```

Because most requests have a Host header, Squid ignores the *httpd_accel_host* setting and rarely inserts the bogus *does.not.exist* name into URIs. This essentially turns your surrogate into a caching proxy for anyone smart enough to fake an HTTP request. If I know that you are using Squid as a surrogate without proper access controls, I can send a request like this:

```
GET /index.html HTTP/1.1
Host: www.mrcranky.com
```

If you've enabled *httpd_accel_uses_host_header* and don't have any destination-based access controls, Squid should forward my request to *www.mrcranky.com*. Read the "Access Controls" section and install access controls to ensure that Squid doesn't talk to foreign origin servers.

httpd_accel_single_host

Whereas the *httpd_accel_uses_host_header* directive determines the hostname Squid puts into a URI, this one determines where Squid forwards its cache misses. By default (i.e., with *httpd_accel_single_host* disabled), Squid forwards surrogate cache misses to the host in the URI. If the URI contains a hostname, Squid performs a DNS lookup to get the backend server's IP address.

When you enable *httpd_accel_single_host*, Squid always forwards surrogate cache misses to the host defined by *httpd_accel_host*. In other words, the contents of the URI and the Host header don't affect the forwarding decision. Perhaps the best reason to enable this directive is to avoid DNS lookups. Simply set *httpd_accel_host* to the backend server's IP address. Another reason to enable it is if you have another device (load balancer, virus scanner, etc.) between Squid and the backend server. You can make Squid forward the request to this other device without changing any aspect of the HTTP request.

Note that enabling both *httpd_accel_single_host* and *httpd_accel_uses_host_header* is a dangerous combination that might allow an attacker to poison your cache. Consider this configuration:

```
httpd_accel_single_host on
httpd_accel_host 172.16.1.1
httpd_accel_uses_host_header on
```

and this HTTP request:

```
GET /index.html HTTP/1.0
Host: www.othersite.com
```

Squid forwards the request to your backend server at 172.16.1.1 but stores the response under the URI *http://www.othersite.com/index.html*. Since 172.16.1.1 isn't actually *www.othersite.com*, Squid now contains a bogus response for that URI. If

you enable *httpd_accel_with_proxy* (next section) or your cache participates in a hierarchy or mesh, it may give out the bad response to unsuspecting users. To prevent such abuse, be sure to read the “Access Controls” section.

Server-side persistent connections may not work if you use the *httpd_accel_single_host* directive. This is because Squid saves idle connections under the origin server hostname, but the connection-establishment code looks for an idle connection named by the *httpd_accel_host* value. If the two values are different, Squid fails to locate an appropriate idle connection. The idle connections are closed after the timeout, without being reused. You can avoid this little problem by disabling server-side persistent connections with the *server_persistent_connections* directive (see Appendix A).

httpd_accel_with_proxy

By default, whenever you enable the *httpd_accel_host* directive, Squid goes into strict surrogate mode. That is, it refuses proxy HTTP requests and accepts only surrogate requests, as though it were truly an origin server. Squid also disables the ICP port (although not HTCP, if you have it enabled). If you want Squid to accept both surrogate and proxy requests, enable this directive:

```
httpd_accel_with_proxy on
```

Gee, That Was Confusing!

Yeah, it was for me too. Let’s look at it another way. The settings that you need to use depend on how many backend boxes you have and how many origin server names you are accelerating. Let’s consider the four separate cases in the following sections.

One Box, One Server Name

This is the simplest sort of configuration. Because you have only one box and one hostname, the *Host* header values don’t matter much. You should probably use:

```
httpd_accel_host www.example.com
httpd_accel_single_host on
httpd_accel_uses_host_header off
```

If you like, you can use an IP address for *httpd_accel_host*, although it will appear in URIs in your *access.log*.

One Box, Many Server Names

Because you have many origin server names being virtually hosted on a single box, the Host header becomes important. We want Squid to insert it into the URIs it generates from partial requests. Your configuration should be:

```
httpd_accel_host www.example.com
httpd_accel_single_host on
httpd_accel_uses_host_header on
```

In this case, Squid generates the URI based on the Host header. If absent, Squid inserts *www.example.com*. You can disable *httpd_accel_single_host* if you prefer. As before, you can use an IP address in *httpd_accel_host* to avoid DNS lookups.

Many Boxes, One Server Name

This sounds like a load-balancing configuration. One way to accomplish it is to create a DNS name for the backend servers with multiple IP addresses. Squid iterates between all addresses (a.k.a. round-robin) for each cache miss. In this situation, the configuration is the same as for the one box/one name case:

```
httpd_accel_host roundrobin.example.com
httpd_accel_single_host on
httpd_accel_uses_host_header off
```

The only difference is that the *httpd_accel_host* name resolves to multiple addresses. It might look like this in a Berkeley Internet Name Daemon (BIND) zone file:

```
$ORIGIN example.com.
roundrobin      IN      A      192.168.1.2
                IN      A      192.168.1.3
                IN      A      192.168.1.4
```

With this DNS configuration, Squid uses the next address in the list each time it opens a new connection to *roundrobin.example.com*. When it gets to the end of the list, it starts over at the top. Note that Squid caches these DNS answers internally according to their TTLs. You aren't relying on the name server to return the address list in a different order for each DNS query.

Another option is to use a redirector (see Chapter 11) to select the backend server. You can write a simple script to replace the URI hostname (e.g., *roundrobin.example.com*) with a different hostname or an IP address. You might even make the redirector smart enough to make its selection based on the current state of the backend servers. Use the following configuration with this approach:

```
httpd_accel_host roundrobin.example.com
httpd_accel_single_host off
httpd_accel_uses_host_header off
```

Many Boxes, Many Server Names

In this case, you want to use the `Host` header. You also want Squid to select the back-end server based on the origin server's name (i.e., a DNS lookup). The configuration is as follows:

```
httpd_accel_host www.example.com
httpd_accel_single_host off
httpd_accel_uses_host_header on
```

You might be tempted to set `httpd_accel_host` to `virtual`. However, that would be a mistake unless you are using HTTP interception.

Access Controls

A typically configured surrogate accepts HTTP requests from the whole Internet. This doesn't mean, however, that you can forget about access controls. In particular, you'll want to make sure Squid doesn't accept requests belonging to foreign origin servers. The exception is when you have `httpd_accel_with_proxy` enabled.

For a surrogate-only configuration, use one of the destination-based access controls. For example, the `dst` type accomplishes the task:

```
acl All src 0/0
acl TheOriginServer dst 192.168.3.2
http_access allow TheOriginServer
http_access deny All
```

Alternatively, you can use a `dstdomain` ACL if you prefer:

```
acl All src 0/0
acl TheOriginServer dstdomain www.squidbook.org
http_access allow TheOriginServer
http_access deny All
```

Note that enabling `httpd_accel_single_host` somewhat bypasses the access control rules. This is because the origin server location (i.e., the `httpd_accel_host` value) is then set after Squid performs the access control checks.

Access controls become really tricky when you combine surrogate and proxy modes in a single instance of Squid. You can no longer simply deny all requests to foreign origin servers. You can, however, make sure that outsiders aren't allowed to make proxy requests to random origin servers. For example:

```
acl All src 0/0
acl ProxyUsers src 10.2.0.0/16
acl TheOriginServer dst 192.168.3.2
http_access allow ProxyUsers
http_access allow TheOriginServer
http_access deny All
```

You can also use the local port number in your access control rules. It doesn't really protect you from malicious activity, but does ensure, for example, that user-agents send their proxy requests to the appropriate port. This also makes it easier for you to split the service into separate proxy- and surrogate-only systems later. Assuming you configure Squid to listen on ports 80 and 3128, you might use:

```
acl All src 0/0
acl ProxyPort myport 3128
acl ProxyUsers src 10.2.0.0/16
acl SurrogatePort myport 80
acl TheOriginServer dst 192.168.3.2
http_access allow ProxyUsers ProxyPort
http_access allow TheOriginServer SurrogatePort
http_access deny All
```

Unfortunately, these access control rules don't prevent attempts to poison your cache when you enable *httpd_accel_single_host*, *httpd_accel_uses_host_header*, and *httpd_accel_with_proxy* simultaneously. This is because the valid proxy request:

```
GET http://www.bad.site/ HTTP/1.1
Host: www.bad.site
```

and the bogus surrogate request:

```
GET / HTTP/1.1
Host: www.bad.site
```

have the same access control result but are forwarded to different servers. They have the same access control result because, after Squid rewrites the surrogate request, it has the same URI as the proxy request. However, they don't get sent to the same place. The surrogate request goes to the server defined by *httpd_accel_host* because *httpd_accel_single_host* is enabled.

You can take steps towards solving this problem. Make sure your backend server generates an error for unknown server names (e.g., when the Host header refers to a nonlocal server). Better yet, don't run Squid as a surrogate and proxy at the same time.

Content Negotiation

Recent versions of Squid support the HTTP/1.1 Vary header. This is good news if your backend server uses content negotiation. It might, for example, send different responses depending on which web browser makes the request (e.g., the User-Agent header), or based on the user's language preferences (e.g., the Accept-Language header).

When the response for a URI varies on some aspect of the request, the origin (backend) server includes a Vary header. This header contains the list of request headers used to select the variant. These are the *selecting* headers. When Squid receives a response with a Vary header, it includes the selecting header values when it generates

the internal cache key. Thus, a subsequent request with the same values for the selecting headers may generate a cache hit.

If you use the `--enable-x-accelerator-vary` option when running `./configure`, Squid looks for a response header named `X-Accelerator-Vary`. Squid treats this header exactly like the `Vary` header. Because this is an extension header, however, it is ignored by downstream agents. It essentially provides a means for private content negotiation between Squid and your backend server. In order to use it you must also modify your server application to send the header in its responses. I don't know of any situation in which this header would be useful. If you serve negotiated responses, you probably want to use the standard `Vary` header so that all agents know what's going on.

Gotchas

Using Squid as a surrogate may improve your origin server's security and performance. However, there are some potentially negative side effects as well. Here are a few things to keep in mind.

Logging

When using a surrogate, the origin server's access log contains only the cache misses from Squid. Furthermore, those log-file entries have Squid's IP address, rather than the client's. In other words, Squid's *access.log* is where all the good information is now stored.

Recall that, by default, Squid doesn't use the common log-file format. You should use the *emulate_httpd_log* directive to make Squid's *access.log* look just like Apache's default log-file format.

Ignoring Reloads

The *Reload* button found on most browsers generates HTTP requests with the `Cache-Control: no-cache` directive set. While this is usually desirable for client-side caching proxies, it may ruin the performance of a surrogate. This is especially true if the backend server is heavily loaded. A reload request forces Squid to purge the currently cached response while retrieving the new response from the origin server. If those origin server responses arrive slowly, Squid consumes a larger than normal number of file descriptors and network resources.

To help in this situation, you may want to use one of the *refresh_pattern* options. When the *ignore-reload* option is set, Squid pretends that the request doesn't contain the `no-cache` directive. The *ignore-reload* option is generally safe for surrogates, although it does, technically, violate the HTTP protocol.

To make Squid ignore reloads for all requests, use a line like this in *squid.conf*:

```
refresh_pattern . 0 20% 4320 ignore-reload
```

For a somewhat safer alternative, you can use the *reload-into-ims* option. It causes Squid to validate its cached response when the request contains *no-cache*. Note, however, that this works only for responses that have cache validators (such as Last-Modified timestamps).

Uncachable Content

As a surrogate, Squid obeys the standard HTTP headers for caching responses from your backend server. This means, for example, that certain dynamic responses might not be cached. You might want to use the *refresh_pattern* directive to force caching of these objects. For example:

```
refresh_pattern \.dhtml$ 60 80% 180
```

This trick only works for certain types of responses, namely, those without a Last-Modified or Expires header. By default, Squid doesn't cache such responses. However, using a nonzero minimum time in a *refresh_pattern* rule instructs Squid to cache the response, and serve it as a cache hit for that amount of time anyway. See Chapter 7 for the details.

If your backend server generates other types of uncachable responses, you may not be able to trick Squid into storing them.

Errors

With Squid as a surrogate in front of your origin server, you should be aware that visitors to your site may see an error message from Squid, rather than the origin server itself. In other words, your use of Squid may be “exposed” through certain error messages. For example, Squid returns its own error message when it fails to parse the client's HTTP request, which could happen if the request is incomplete or is malformed in some way. Squid also returns an error message if it can't connect to the backend server for some reason.

If your site is consistent and functioning properly, you probably don't need to worry about Squid's error messages. Nonetheless, you may want to take a close look at the *access.log* from time to time and see what sort of errors, if any, your users might be seeing.

Purging Objects

You may find the PURGE method particularly useful when operating a surrogate. Because you have a good understanding of the content being served, you are more

likely to know when a cached object must be purged. The technique for purging an object is the same as I mentioned previously. See Chapter 7 for a refresher.

Neighbors

Although I don't recommend it, you can configure Squid as a surrogate and as part of a mesh or hierarchy. If you choose to take on such an arrangement, note that, by default, Squid forwards cache misses to parents (rather than the backend server). Assuming that isn't what you really want, be sure to use the *cache_peer_access* directives so that requests for your backend server don't go to your neighbors instead.

Exercises

- Install and configure Squid as a surrogate on the same system where you run an HTTP server.
- Make a few test requests with *squidclient*. Pay particular attention to the reply headers and notice how the requests appear in both access logs.
- Try to poison your own surrogate with fake HTTP requests. It is probably easier with *httpd_accel_single_host* enabled.
- Estimate the size of your origin server's document set. What percentage of the data can fit into 1 GB of memory or disk space?

Debugging and Troubleshooting

No matter how hard the Squid developers try to be perfect, you may encounter some problems with Squid. These problems range from misbehaving clients and servers to fatal bugs in the Squid code. In this chapter, I'll talk about various ways you can debug these problems.

Some Squid problems may require you to turn on debugging. In most cases, you'll want to increase the debugging levels for specific parts of the code. I'll describe how to find out what those debugging sections are and how to change the settings. Also, I'll talk about the importance of providing detailed debugging when reporting bugs.

Finally, you may experience fatal bugs in the Squid code. These can result in segmentation violations, aborts, assertions, and core dumps. The core dump is a useful debugging aid. With a debugger, such as `gdb`, you can generate a process stack trace and send it to the developers for assistance.

If you suspect you have a Squid bug, but aren't sure, check with the *squid-users* mailing list or one of the other resources described in Chapter 1.

Some Common Problems

Before discussing debugging in general, I'll mention a few specific problems that commonly arise.

"Failed to make swap directory"

Failed to make swap directory /var/spool/cache: (13) Permission denied

This happens when you run `squid -z`, and the Squid user ID doesn't have write permission to the `/var/spool` directory. Remember that if you start Squid as *root* and don't add a *cache_effective_user* line, Squid runs as the user *nobody* by default. Thus, your solution may be to simply run:

```
# chown nobody:nobody /var/spool
```

“Address already in use”

```
commBind: Cannot bind socket FD 10 to *:3128: Address already in use
```

This message appears when the `bind()` system call fails because the requested port is already opened by another application. Usually, this happens when you try to start a second instance of Squid when the first one is still running. If you see this error message, use `ps` to see if Squid is already running.

Squid uses the `SO_REUSEADDR` socket option, so that the `bind()` call should succeed even if there are some leftover sockets in the `TIME_WAIT` state. If you get the message, even though Squid isn't already running, your operating system may be buggy or especially finicky. Rebooting your system is one way to get around this problem.

Another possibility to consider is that the port (e.g., 3128) is currently being used by a different application. If you suspect this, you can use the `lsof` program (<ftp://lsof.itap.purdue.edu/pub/tools/unix/lsof>) to find which application is listening on the port. FreeBSD users can use `sockstat` instead.

“Could not determine fully qualified hostname”

```
FATAL: Could not determine fully qualified hostname. Please set 'visible_hostname'
```

You'll see this if Squid can't figure out its own fully qualified domain name. Here is the algorithm Squid uses:

- If you told Squid to bind the HTTP port to a specific interface address, Squid attempts a reverse DNS lookup of that address. If successful, the answer is used.
- Squid calls the `gethostname()` function, and then attempts to resolve its IP address with `gethostbyname()`. If successful, Squid uses the official hostname string returned by the latter function.

If neither technique works, Squid exits with the fatal message shown earlier. In this case, you must tell Squid its hostname with the `visible_hostname` directive. For example:

```
visible_hostname my.host.name
```

“DNS name lookup tests failed”

By default, Squid makes a few DNS queries before starting. This ensures that your DNS servers are reachable and functioning properly. If these tests fail, you'll see the following message in `cache.log` and/or `syslog`:

```
FATAL: ipcache_init: DNS name lookup tests failed
```

If you use Squid on an intranet, Squid may be unable to query its standard list of hostnames. You can specify your own hostnames with the `dns_testnames` directive. Squid considers the DNS test successful as soon as it receives any reply.

If you want to skip the DNS tests altogether, simply use the `-D` command-line option when starting Squid:

```
% squid -D ...
```

“Illegal character in hostname”

```
urlParse: Illegal character in hostname 'super_bikes.tripod.com'
```

By default, Squid checks the characters in the hostname part of URLs and complains if it finds nonstandard characters. According to RFCs 1034 and 1035, names must consist of the letters A–Z, the digits 0–9, and a hyphen (-). The underscore () is one of the most problematic characters.

Squid validates hostnames because, in some cases, DNS resolvers behave differently with respect to illegal characters. For example:

```
% host super_bikes.tripod.com
super_bikes.tripod.com has address 209.202.196.70

% ping super_bikes.tripod.com
ping: cannot resolve super_bikes.tripod.com: Unknown server error
```

Rather than return the Unknown server error message, Squid checks the hostname first. It can then tell the user when the hostname contains illegal characters.

Some DNS resolvers do work with underscores and other nonstandard characters. If you’d prefer that Squid not check hostnames, use the `--disable-hostname-checks` option when running `./configure`. If you want to allow underscores as the only exception, use the `--enable-underscores` option.

“Running out of filedescriptors”

```
WARNING! Your cache is running out of filedescriptors
```

The above message appears when Squid uses up all available file descriptors. If this happens under normal conditions, you need to increase the kernel’s file-descriptor limits and recompile Squid. See Chapter 3.

You might also see this message if Squid is the target of a denial-of-service attack. Someone may be intentionally, or unintentionally, sending Squid hundreds or thousands of requests at once. If this is the case, you can probably add a packet-filtering rule to block incoming TCP connections from the offending address(es). If the attack is distributed or using a spoofed source address, you’ll have a harder time stopping it.

Forwarding loops (see Chapter 10) might also consume all of Squid’s file descriptors, but only if Squid can’t detect the loop. The `Via` header contains the hostname of all proxies that have seen a particular request. Squid looks for its own hostname in the header, and, if found, reports the loop. If, for some reason, the `Via` header is filtered from outgoing or incoming HTTP requests, Squid can’t detect the loop. In this

case, all file descriptors are quickly consumed by the same request going through Squid over and over.

“icmpRecv: Connection refused”

You’ll see the following message if the pinger program isn’t correctly installed:

```
icmpRecv: recv: (61) Connection refused
```

Most likely, the pinger program exits immediately because it doesn’t have permission to open a raw ICMP socket. Because the process isn’t running, Squid receives an I/O error when trying to talk to it. To alleviate the problem, go to the source directory and, as *root*, type:

```
# make install-pinger
```

If successful, you should find that the pinger program has the following file ownership and permission settings:

```
# ls -l /usr/local/squid/libexec/pinger
-rws--x--x 1 root  squid 140728 Sep 16 19:58 /usr/local/squid/libexec/pinger
```

Squid Becomes Slow After Running for Some Time

Most likely, Squid is competing with other processes, or with itself, for memory on your system. When the Squid process no longer fits entirely in memory, the operating system is forced to read and write areas of memory to and from the swap space. This has a drastic effect on Squid’s performance.

To validate this theory, check the Squid process size with utilities such as *top* and *ps*. Also check Squid’s own page fault counter, as described in Chapter 14. Once you’ve identified memory consumption as the problem, try the following steps to reduce Squid’s memory usage:

1. Reduce the value of *cache_mem* and read Appendix B.
2. Turn off memory pooling with this option:

```
memory_pools off
```
3. Reduce the size of the disk cache by lowering the size of one or more cache directories. See Chapter 7.

Debugging Access Controls

If you’re having no luck getting your access controls to work properly, here’s a little tip that might help. Edit your *squid.conf* file and set the *debug_options* line to this:

```
debug_options ALL,1 33,2
```

Then, reconfigure Squid:

```
% squid -k reconfigure
```

Now, Squid writes a message to *cache.log* for each client request and another for each reply. The messages contain the request method, URI, whether the request/reply is allowed or denied, and the name of the last ACL that matched it. For example:

```
2003/09/29 20:22:05| The request
GET http://images.slashdot.org:80/topics/topicprivacy.gif is ALLOWED,
because it matched 'localhost'
2003/09/29 20:22:05| The reply for
GET http://images.slashdot.org/topics/topicprivacy.gif is ALLOWED,
because it matched 'all'
```

Knowing the name of the ACL doesn't always tell you the corresponding *http_access* line, but it gets you pretty close. If necessary, you can replicate your *acl* lines and give them unique names so that a given ACL name appears on only one *http_access* rule.

Debugging via cache.log

You already know from Chapter 13 that *cache.log* contains various operational messages Squid thinks are important enough to tell you about. We also refer to these as debugging messages. You can use the *debug_options* directive to control the verbosity of messages that appear in *cache.log*. By increasing the debugging levels, you'll see more detailed messages that may help you understand what Squid is doing. For example:

```
debug_options ALL,1 11,3 20,3
```

Every debugging message in the Squid source code has two numeric attributes: a *section* and a *level*. Sections range from 0 to 100, and levels range from 0 to 10. In general, section numbers correspond to particular components of the source code. In other words, all the messages within a single source file have the same section number. In some cases, multiple files use the same debugging section. This tends to happen when a source file becomes too large and is split into smaller chunks.

The top of each source file has line that mentions the debugging section. It looks like this:

```
* DEBUG: section 9      File Transfer Protocol (FTP)
```

I don't expect you to look at the source files to find the section numbers. The same information appears here in Table 16-1.

Table 16-1. Debugging section numbers for the *debug_options* directive

Number	Description	Source file(s)
0	Client Database	<i>client_db.c</i>
1	Startup and Main Loop	<i>main.c</i>
2	Unlink Daemon	<i>unlinkd.c</i>
3	Configuration File Parsing	<i>cache_cf.c</i>

Table 16-1. Debugging section numbers for the `debug_options` directive (continued)

Number	Description	Source file(s)
4	Error Generation	<i>errorpage.c</i>
5	Socket Functions	<i>comm.c</i>
5	Socket Functions	<i>comm_select.c</i>
6	Disk I/O Routines	<i>disk.c</i>
7	Multicast	<i>multicast.c</i>
8	Swap File Bitmap	<i>filemap.c</i>
9	File Transfer Protocol (FTP)	<i>ftp.c</i>
10	Gopher	<i>gopher.c</i>
11	Hypertext Transfer Protocol (HTTP)	<i>http.c</i>
12	Internet Cache Protocol	<i>icp_v2.c</i>
12	Internet Cache Protocol	<i>icp_v3.c</i>
13	High Level Memory Pool Management	<i>mem.c</i>
14	IP Cache	<i>ipcache.c</i>
15	Neighbor Routines	<i>neighbors.c</i>
16	Cache Manager Objects	<i>cache_manager.c</i>
17	Request Forwarding	<i>forward.c</i>
18	Cache Manager Statistics	<i>stat.c</i>
19	Store Memory Primitives	<i>stmem.c</i>
20	Storage Manager	<i>store.c</i>
20	Storage Manager Client-Side Interface	<i>store_client.c</i>
20	Storage Manager Heap-Based Replacement	<i>repl/heap/store_heap_replacement.c</i>
20	Storage Manager Logging Functions	<i>store_log.c</i>
20	Storage Manager MD5 Cache Keys	<i>store_key_md5.c</i>
20	Storage Manager Swapfile Metadata	<i>store_swapmeta.c</i>
20	Storage Manager Swapin Functions	<i>store_swapin.c</i>
20	Storage Manager Swapout Functions	<i>store_swapout.c</i>
20	Store Rebuild Routines	<i>store_rebuild.c</i>
21	Misc Functions	<i>tools.c</i>
22	Refresh Calculation	<i>refresh.c</i>
23	URL Parsing	<i>url.c</i>
24	WAIS Relay	<i>wais.c</i>
25	MIME Parsing	<i>mime.c</i>
26	Secure Sockets Layer Proxy	<i>ssl.c</i>
27	Cache Announcer	<i>send-announce.c</i>
28	Access Control	<i>acl.c</i>
29	Authenticator	<i>auth/basic/auth_basic.c</i>

Table 16-1. Debugging section numbers for the `debug_options` directive (continued)

Number	Description	Source file(s)
29	Authenticator	<code>auth/digest/auth_digest.c</code>
29	Authenticator	<code>authenticate.c</code>
29	NTLM Authenticator	<code>auth/ntlm/auth_ntlm.c</code>
30	Ident (RFC 1413)	<code>ident.c</code>
31	Hypertext Caching Protocol	<code>htcp.c</code>
32	Asynchronous Disk I/O	<code>fs/aufs/async_io.c</code>
33	Client-Side Routines	<code>client_side.c</code>
34	Dnsserver Interface	<code>dns.c</code>
35	FQDN Cache	<code>fqdn_cache.c</code>
37	ICMP Routines	<code>icmp.c</code>
38	Network Measurement Database	<code>net_db.c</code>
39	Cache Array Routing Protocol	<code>carp.c</code>
40	Referer Logging	<code>referer.c</code>
40	User-Agent Logging	<code>useragent.c</code>
41	Event Processing	<code>event.c</code>
42	ICMP Pinger Program	<code>pinger.c</code>
43	AIOPS	<code>fs/aufs/aiops.c</code>
44	Peer Selection Algorithm	<code>peer_select.c</code>
45	Callback Data Registry	<code>cbdata.c</code>
45	Callback Data Registry	<code>leakfinder.c</code>
46	Access Log	<code>access_log.c</code>
47	Store COSS Directory Routines	<code>fs/coss/store_dir_coss.c</code>
47	Store Directory Routines	<code>fs/aufs/store_dir_aufs.c</code>
47	Store Directory Routines	<code>fs/diskd/store_dir_diskd.c</code>
47	Store Directory Routines	<code>fs/null/store_null.c</code>
47	Store Directory Routines	<code>fs/ufs/store_dir_ufs.c</code>
47	Store Directory Routines	<code>store_dir.c</code>
48	Persistent Connections	<code>pconn.c</code>
49	SNMP Interface	<code>snmp_agent.c</code>
49	SNMP Support	<code>snmp_core.c</code>
50	Log File Handling	<code>logfile.c</code>
51	File Descriptor Functions	<code>fd.c</code>
52	URN Parsing	<code>urn.c</code>
53	AS Number Handling	<code>asn.c</code>
54	Interprocess Communication	<code>ipc.c</code>
55	HTTP Header	<code>HTTPHeader.c</code>

Table 16-1. Debugging section numbers for the *debug_options* directive (continued)

Number	Description	Source file(s)
56	HTTP Message Body	<i>HttpBody.c</i>
57	HTTP Status-Line	<i>HttpStatusLine.c</i>
58	HTTP Reply (Response)	<i>HttpReply.c</i>
59	Auto-Growing Memory Buffer with <i>printf</i>	<i>MemBuf.c</i>
60	Packer: A Uniform Interface to Store Like Modules	<i>Packer.c</i>
61	Redirector	<i>redirect.c</i>
62	Generic Histogram	<i>StatHist.c</i>
63	Low Level Memory Pool Management	<i>MemPool.c</i>
64	HTTP Range Header	<i>HttpHdrRange.c</i>
65	HTTP Cache Control Header	<i>HttpHdrCc.c</i>
66	HTTP Header Tools	<i>HTTPHeaderTools.c</i>
67	String	<i>String.c</i>
68	HTTP Content-Range Header	<i>HttpHdrContRange.c</i>
69	HTTP Header: Extension Field	<i>HttpHdrExtField.c</i>
70	Cache Digest	<i>CacheDigest.c</i>
71	Store Digest Manager	<i>store_digest.c</i>
72	Peer Digest Routines	<i>peer_digest.c</i>
73	HTTP Request	<i>HttpRequest.c</i>
74	HTTP Message	<i>HttpMsg.c</i>
75	WHOIS Protocol	<i>whois.c</i>
76	Internal Squid Object handling	<i>internal.c</i>
77	Delay Pools	<i>delay_pools.c</i>
78	DNS Lookups; interacts with <i>lib/rfc1035.c</i>	<i>dns_internal.c</i>
79	Squid-Side DISK I/O Functions	<i>fs/diskd/store_io_diskd.c</i>
79	Storage Manager COSS Interface	<i>fs/coss/store_io_coss.c</i>
79	Storage Manager UFS Interface	<i>fs/ufs/store_io_ufs.c</i>
80	WCCP Support	<i>wccp.c</i>
82	External ACL	<i>external_acl.c</i>
83	SSL Accelerator Support	<i>ssl_support.c</i>
84	Helper Process Maintenance	<i>helper.c</i>

Debugging levels are assigned such that more important messages have smaller values and less important messages have higher values. Level 0 is for very important messages, while level 10 is for those that are relatively unimportant. Other than that, there are no strict guidelines or requirements. Developers are generally free to choose which debugging levels are appropriate.

The *debug_options* directive determines which messages appear in *cache.log*. Its syntax is:

```
debug_options section,level section,level ...
```

The default setting is ALL,1 such that Squid prints any debugging message with level 0 or 1. If you want to make even less output appear in *cache.log*, you can set *debug_options* to ALL,0.

If you want to see additional debugging for a particular component, simply add the appropriate section and level to the end of the *debug_options* list. For example, this line adds level 5 debugging for the FTP server-side code:

```
debug_options ALL,1 9,5
```

As with other configuration directives, you can change *debug_options*, then send Squid the reconfigure signal:

```
% squid -k reconfigure
```

Note that the *debug_options* parameters are processed sequentially, and a later value can override an earlier one. This is of particular concern if you use the ALL keyword. Consider this example:

```
debug_options 9,5 20,9 4,2 ALL,1
```

In this case, the final value overwrites all of the preceding settings because ALL,1 sets the debugging level to 1 for all sections.

Selecting appropriate debugging sections and levels is sometimes quite difficult, especially for novice Squid users. Many of the more detailed debugging messages are meaningful only to developers and those familiar with the source code. Inexperienced Squid users are likely to find many of the debugging messages meaningless and overwhelming. Furthermore, you may have difficulty isolating the debugging for a particular request or event if Squid is relatively busy. The higher debugging levels are often more useful if you can test Squid with one request at a time.

You must also be particularly careful about running Squid with high debugging levels for a long amount of time. If Squid is busy, the *cache.log* file grows very quickly and may eventually consume all free space on its partition. If this happens, Squid exits with a fatal message. Another concern is that performance may degrade significantly. Due to the high number of debugging messages, Squid devotes a lot of CPU resources to formatting and printing strings. It also consumes a lot of disk bandwidth writing them all to *cache.log*.

Core Dumps, Assertions, and Stack Traces

If you are unlucky, Squid may experience a fatal error while running. These sorts of errors come in three flavors: assertions, bus errors, and segmentation violations.

An *assertion* is a sanity check in the source code. It is a tool, used by developers, to make sure that some condition is always true before proceeding. If the condition is false, the program exits and creates a core file so that the developer can analyze the situation. Here is a typical example:

```
int some_array[100];

void
some_func(int idx)
{
    ...
    assert(idx < 100);
    some_array[idx]++;
    ...
}
```

Here, the assertion makes sure that the value of the array index is within the bounds of the array. It would be an error to access array elements greater than (or equal to) 100. If, somehow, the value of *idx* isn't less than 100, the program prints a message like this when it runs:

```
assertion failed: filename.c:123: "idx < 100"
```

If this happens with Squid, you'll see an "assertion failed" message in *cache.log*. In addition, your operating system should create a *core* file, which is helpful in the post-mortem analysis. I'll explain what to do with a *core* file at the end of this section.

A *bus error* is "a fatal failure in the execution of a machine language instruction resulting from the processor detecting an anomalous condition on its bus." They typically occur when the processor attempts an operation on a nonaligned memory address. You are, perhaps, more likely to see a bus error on a 64-bit processor system, such as the Alpha and some SPARC CPUs. Fortunately, they are easy to fix.

Segmentation violation errors are, unfortunately, more common and sometimes harder to fix. A "SEGV" usually occurs when the process tries to access an invalid memory area. It might be a NULL pointer or a memory address outside the scope of the process. They are particularly difficult to track down when the cause (the bug) and effect (the SEGV) are separated in time.

By default, Squid traps bus errors and segmentation violations, and attempts a clean shutdown when they occur. You'll see something like this in *cache.log*:

```
FATAL: Received Bus Error...dying.
2003/09/29 23:18:01| storeDirWriteCleanLogs: Starting...
```

In most cases, Squid is able to write clean versions of the *swap.state* files. Just before exiting, Squid calls `abort()` to create a *core* file. The *core* file may help you, or other developers, track down and fix the bug.

* From the Free On-line Dictionary of Computing (FOLDOC), <http://wombat.doc.ic.ac.uk/foldoc/>.

A *core* file is generally more useful when it is created immediately following the error, rather than calling the clean shutdown procedure first. You can tell Squid not to trap bus errors and segmentation violations with the `-C` command line option:

```
% squid -C ...
```

Note that some operating systems use the filename *core*, while others prepend the process name (i.e., *squid.core*). Once you have the *core* file, use a debugger to get a stack trace. `gdb` is the GNU debugger—a companion to the GNU C compiler. If you don't have `gdb`, try running `dbx` or `adb` instead. Here's how you can use `gdb` to get a stack trace:

```
% gdb /usr/local/squid/sbin/squid /path/to/squid.core
...
Core was generated by 'squid'.
Program terminated with signal 6, Abort trap.
...
```

Then, type **where** to print the stack trace:

```
(gdb) where
#0 0x28168b54 in kill () from /usr/lib/libc.so.4
#1 0x281aa0ce in abort () from /usr/lib/libc.so.4
#2 0x80a2316 in death (sig=10) at tools.c:301
#3 0xbfbfffac in ?? ()
#4 0x80abe0a in storeDiskdSend (mtype=4, sd=0x82101e0, id=1214000,
sio=0x9e90a10, size=4096, offset=-1, shm_offset=0)
at diskd/store_io_diskd.c:485
#5 0x80ab726 in storeDiskdWrite (SD=0x82101e0, sio=0x9e90a10,
buf=0x13e94000 "...", size=4096, offset=-1, free_func=0)
at diskd/store_io_diskd.c:251
#6 0x809d2fb in storeWrite (sio=0x9e90a10, buf=0x13e94000 "...",
size=4096, offset=-1, free_func=0) at store_io.c:89
#7 0x80a1c2d in storeSwapOut (e=0xc5a7800) at store_swapout.c:259
#8 0x809b667 in storeAppend (e=0xc5a7800, buf=0x810f9a0 "...", len=57344)
at store.c:533
#9 0x807873b in httpReadReply (fd=134, data=0xc343590) at http.c:642
#10 0x806492f in comm_poll (msec=10) at comm_select.c:445
#11 0x8084404 in main (argc=2, argv=0xbfbffa8c) at main.c:742
#12 0x804a465 in _start ()
```

As you can see, the stack trace prints the name of each function, its arguments, and the source code filenames and line numbers. This information is extremely useful when tracking down bugs. In some cases, however, it isn't sufficient. You might be asked to execute additional commands in the debugger, such as printing the value of a variable from within a certain function:

```
(gdb) frame 4
#4 0x80abe0a in storeDiskdSend (mtype=4, sd=0x82101e0, id=1214000,
sio=0x9e90a10, size=4096, offset=-1, shm_offset=0)
at diskd/store_io_diskd.c:485
485         x = msgsnd(diskdinfo->smsgid, &M, msg_snd_rcv_sz, IPC_NOWAIT);
(gdb) set print pretty
(gdb) print M
```

```
$2 = {
  mtype = 4,
  id = 1214000,
  seq_no = 7203103,
  callback_data = 0x9e90a10,
  size = 4096,
  offset = -1,
  status = -1,
  shm_offset = 0
}
```

After you've reported a bug, try to keep the *core* file around for a few days, in case you need additional information from it.

Can't Find the Core File?

core files are written in the process' current directory. By default, Squid doesn't change its current directory at startup. Thus, your *core* file, if any, should be written in the directory in which Squid was started. You won't find a *core* file if the filesystem doesn't have enough free space or if the process owner doesn't have write permission in the directory. You can use the *coredump_dir* directive to make Squid use a specific location—somewhere with plenty of space and sufficient permissions.

Process resource limits may also prevent the creation of a *core* file. One of the process limit parameters is the size of the core dump file. Usually, most systems set this to "unlimited" by default. You can check the current limit from your shell with the *limits* or *ulimit* commands. Note, however, that your shell's limit might be different than the Squid process limit, especially when Squid is started automatically at boot time. If you suspect process limits prevent generation of a *core* file, try this:

```
csh% limit coredumpsize unlimited
csh% squid -NCd1
```

On FreeBSD, a *sysctl* parameter controls whether or not the operating system generates a *core* file for processes that call *setuid()* and/or *setgid()*. Squid uses those functions if you start it as *root*. To get a core dump, then, you must tell the kernel to create the *core* file with this command:

```
# sysctl kern.sugid_coredump=1
```

See the *sysctl.conf* manpage for information on how to set the variable automatically when your system boots.

Replicating Problems

Occasionally you may encounter a certain request or origin server that seems not to work with Squid. You can use the following technique to determine if the problem lies with Squid, the client, or the origin server. The trick is to capture the HTTP request, then replay it in different ways until you identify the problem.

Capturing the HTTP request means getting more than just the URL. You also need the request method, HTTP version number, and all of the request headers. One way to capture the request is by enabling full debugging in Squid for a short time. On the Squid box, type:

```
% squid -kdebug
```

Then, go to the web browser and issue the request. Squid should receive the request almost immediately. After a few seconds, go back to the Squid box and issue the same command:

```
% squid -kdebug
```

Now your *cache.log* file should contain the client's request. If your Squid is busy, the *cache.log* will contain a lot of requests, so you'll have to search for it. It looks something like this:

```
2003/09/29 10:37:40| parseHttpRequest: Method is 'GET'
2003/09/29 10:37:40| parseHttpRequest: URI is 'http://squidbook.org/'
2003/09/29 10:37:40| parseHttpRequest: Client HTTP version 1.1.
2003/09/29 10:37:40| parseHttpRequest: req_hdr = {
User-Agent: Mozilla/5.0 (compatible; Konqueror/3)
Pragma: no-cache
Cache-control: no-cache
Accept: text/*, image/jpeg, image/png, image/*, */*
Accept-Encoding: x-gzip, gzip, identity
Accept-Charset: iso-8859-1, utf-8;q=0.5, */q=0.5
Accept-Language: en
Host: squidbook.org
```

Note that Squid prints the components of the first line separately. You'll have to manually reassemble them like this:

```
GET http://squidbook.org/ HTTP/1.1
```

Another way to capture the full request is with a utility such as netcat or socket (<http://www.jnickelsen.de/socket/>). Start the socket program listening on some port, then configure the browser to use that port as the proxy address. When you make the request again, socket prints the HTTP request:

```
% socket -s 8080
GET http://squidbook.org/ HTTP/1.1
User-Agent: Mozilla/5.0 (compatible; Konqueror/3)
Pragma: no-cache
Cache-control: no-cache
Accept: text/*, image/jpeg, image/png, image/*, */*
Accept-Encoding: x-gzip, gzip, identity
Accept-Charset: iso-8859-1, utf-8;q=0.5, */q=0.5
Accept-Language: en
Host: squidbook.org
```

Finally, you can also use a network packet capture utility, such as tcpdump or ethereal. After capturing a few packets with tcpdump, you can then use tcpshow to view them:

```
# tcpdump -w tcpdump.log -c 10 -s 1500 port 80
# tcpshow -noHostNames -noPortNames < tcpdump.log | less
...
Packet 4
TIME: 08:39:29.593051 (0.000627)
LINK: 00:90:27:16:AA:75 -> 00:00:24:C0:0D:25 type=IP
IP: 10.0.0.21 -> 206.168.0.6 hlen=20 TOS=00 dgramlen=304 id=4B29
MF/DF=0/1 frag=0 TTL=64 proto=TCP cksum=15DC
TCP: port 2074 -> 80 seq=0481728885 ack=4107144217
hlen=32 (data=252) UAPRSF=011000 wnd=57920 cksum=EB38 urg=0
DATA: GET / HTTP/1.0.
Host: www.ircache.net.
Accept: text/html, text/plain, application/pdf, application/
postscript, text/sgml, */*;q=0.01.
Accept-Encoding: gzip, compress.
Accept-Language: en.
Negotiate: trans.
User-Agent: Lynx/2.8.1rel.2 libwww-FM/2.14.
```

Note that tcpshow prints a period where the data contains a newline character.

Once you've captured a request, save it to a file. Then you can replay it through Squid with netcat or socket:

```
% socket squidhost 3128 < request | less
```

If the response looks normal, the problem might be with the user-agent. Otherwise, you can change various things to isolate the problem. For example, if you see some funny-looking HTTP headers, delete them from the request and try it again. You may also find it useful to try the request directly with the origin server, instead of going through Squid. To do that, remove the `http://host.name/` from the request and send it to the origin server:

```
% cat request
GET / HTTP/1.1
User-Agent: Mozilla/5.0 (compatible; Konqueror/3)
Pragma: no-cache
Cache-control: no-cache
Accept: text/*, image/jpeg, image/png, image/*, */*
Accept-Encoding: x-gzip, gzip, identity
Accept-Charset: iso-8859-1, utf-8;q=0.5, */q=0.5
Accept-Language: en
Host: squidbook.org
```

```
% socket squidbook.org 80 < request | less
```

When working with HTTP in this manner, you might find it useful to refer to RFC 2616 and O'Reilly's *HTTP: The Definitive Guide*.

Reporting a Bug

If your Squid version is more than a few months old, you should probably update it before reporting any bugs. Chances are that others noticed the same bug, and it may already be fixed.

If you discover a legitimate bug in Squid, please enter it into the Squid bug tracking database: <http://www.squid-cache.org/bugs/>. This is currently a “bugzilla” database, which requires you to create an account. You will receive updates as the bug is processed by Squid developers.

If you are new at reporting bugs, please take the time to read “How to Report Bugs Effectively,” by Simon Tatham (<http://www.chiark.greenend.org.uk/~sgtatham/bugs.html>).

When reporting a bug, be sure to include the following information:

- Squid version number. If the bug happens with more than one version, include the other versions as well.
- Your operating system name and version.
- Whether the bug happens every time or occasionally.
- A good description of exactly what happens. Phrases such as “it doesn’t work,” and “the request fails” are essentially useless to bug fixers. Be very specific.
- A stack trace in the case of an assertion, bus error, or segmentation violation.

Remember that Squid developers are generally unpaid volunteers, so be patient. Critical bugs have more priority over minor annoyances.

Exercises

- Use `tcpdump` or `ethereal` to capture some real HTTP requests. Save them to a file and replay the requests through Squid. Feel free to modify or delete some of the HTTP headers.
- Try to make Squid run out of file descriptors.
- Run `tail -f cache.log` and start Squid with `debug_options` set to `ALL,3`. If that is too overwhelming, try `ALL,2`.
- Force Squid to generate a `core` file by sending each of the following signals: `SIGBUS`, `SIGSEGV`, and `SIGABRT`. Find the `core` file and use `gdb` or another debugger to get a stack trace.

Config File Reference

This appendix contains descriptions and examples for every *squid.conf* directive. I present them here in the same order they appear in the default *squid.conf*. That means certain related directives are grouped together, and some recently added directives are at the end. You may want to use the book's index to locate a particular directive by name.

In the following sections, the descriptive text is followed by a table that contains the directive's syntax, a default value, an example, and related directives.

http_port

This is the port, or ports, Squid uses to listen for HTTP requests from cache clients. If your system has more than one network interface, you can use the optional hostname prefix to make Squid bind the socket to a specific IP address. The hostname must correspond to one of your interface addresses. I recommend using an IP address here, instead of a hostname, to avoid DNS lookup delays at startup.

If you run Squid as a surrogate (accelerator), you probably want to accept HTTP connections on port 80. Binding to privileged ports requires root permissions.

Syntax	<code>http_port [hostname:]port [[hostname:]port] ...</code>
Default	<code>http_port 3128</code>
Example	<code>http_port 8080</code> <code>http_port 3128 3129 3130 3131</code> <code>http_port 192.168.1.1:3128</code>
Related	<i>https_port, icp_port, htcp_port, snmp_port, httpd_accel_port, http_access</i>

https_port

This directive allows Squid to accept encrypted (SSL or TLS) connections. It is available only when you use the `/configure --enable-ssl` option.

The mandatory `cert=` argument specifies the pathname to an SSL certificate file in PEM format. This is the format commonly used by OpenSSL and other security software for portable representation of encryption keys.

ssl_unclean_shutdown

The optional `key=` argument is the path to a private key file. If you omit this option, Squid assumes the former key file also contains a private key.

You can use the `version=` argument to tell Squid which protocol versions are allowed: 1=automatic, 2=SSLv2 only, 3=SSLv3 only, 4=TLSv1 only.

The `cipher=` argument is an optional colon-separated list of allowed ciphers. Squid simply passes this list to the `SSL_CTX_set_cipher_list()` function.

Lastly, the `options=` argument allows you to pass additional configuration parameters to the OpenSSL library. For example, `NO_SSLv2`, `NO_SSLv3`, and `NO_TLSv1` disable the use of those particular protocols. Additional option keywords are defined in Squid's `src/ssl_support.c` file.

Syntax	<code>https_port [hostname:]port cert=certificate.pem [key=key.pem] [version=N] [cipher=list] [options=SSL_Options]</code>
Default	No default
Example	<code>https_port 443 cert=/etc/squid-cert.pem key=/etc/squid-privkey.pem</code>
Related	<code>http_port</code> , <code>http_access</code>

ssl_unclean_shutdown

This is a hack borrowed from `mod_ssl` for Apache. Certain user-agents, notably Microsoft Internet Explorer, may not execute the SSL shutdown procedure correctly, especially when persistent connections are involved. Enabling this directive violates the SSL/TLS standard but may eliminate error messages from broken clients.

Syntax	<code>ssl_unclean_shutdown on off</code>
Default	<code>ssl_unclean_shutdown off</code>
Example	<code>ssl_unclean_shutdown on</code>
Related	<code>https_port</code>

icp_port

This is the UDP port Squid uses for ICP messages. In particular, it is used both for sending and receiving queries and replies. Your Squid receives ICP queries from other caches on this port. It also receives ICP replies from other caches, in response to its own queries, on this port.

Unlike `http_port`, you can't specify a list of ICP port numbers. Furthermore, you must use the `udp_incoming_address` and `udp_outgoing_address` directives if you want to restrict ICP traffic to a specific interface address.

Setting `icp_port` to 0 disables ICP.

Syntax	<code>icp_port port</code>
Default	<code>icp_port 3130</code>
Example	<code>icp_port 4130</code>
Related	<code>icp_query_timeout</code> , <code>icp_access</code> , <code>log_icp_queries</code> , <code>icp_hit_stale</code> , <code>udp_incoming_address</code> , <code>http_port</code> , <code>http_port</code> , <code>cache_peer</code>

htcp_port

The Hypertext Caching Protocol is an alternative to ICP. It provides better security and better cache hit predictions. However, HTCP messages are larger and more complicated. HTCP must be enabled at compile-time with the `--enable-htcp` option.

This directive specifies the UDP port Squid uses to send and receive HTCP queries and replies. You may only specify one HTCP port number. As with ICP, the `udp_incoming_address` and `udp_outgoing_address` directives also control HTCP packets.

You may configure Squid to receive both ICP and HTCP queries at the same time. Setting `htcp_port` to 0 disables HTCP.

Syntax	<code>htcp_port port</code>
Default	<code>htcp_port 4827</code>
Example	<code>htcp_port 9999</code>
Related	<code>icp_port</code> , <code>http_port</code> , <code>udp_incoming_address</code> , <code>udp_outgoing_address</code> , <code>cache_peer</code>

mcast_groups

As discussed in Chapter 10, Squid supports receiving ICP queries via multicast. This option specifies a list of multicast addresses Squid should join to receive these ICP queries.



IP multicast is a very tricky and often fragile feature of the Internet. I strongly recommend you avoid using multicast for ICP unless you are already familiar with it. Don't try to guess appropriate values for these directives, and don't expect it to work the first time.

Syntax	<code>mcast_groups multicast-address [multicast-address] ...</code>
Default	No default
Example	<code>mcast_groups 239.128.16.128</code>
Related	<code>cache_peer</code> , <code>mcast_icp_query_timeout</code>

udp_incoming_address

This directive causes Squid to bind all UDP sockets to a specific interface address. The IP address must correspond to one of the system's network interfaces. This directive affects the DNS (when using the internal implementation), ICP, HTCP, and SNMP sockets.

If your system has just one IP address, you probably shouldn't use this directive.

If you set `udp_outgoing_address` to one of your other network interface addresses, Squid can receive UDP datagrams on that interface as well.

Syntax	<code>udp_incoming_address ip-address</code>
Default	<code>udp_incoming_address 0.0.0.0</code>
Example	<code>udp_incoming_address 192.168.4.5</code>
Related	<code>udp_outgoing_address</code> , <code>icp_port</code> , <code>htcp_port</code> , <code>snmp_port</code>

udp_outgoing_address

This directive specifies the source address for UDP messages that Squid sends. It affects DNS (when using the internal implementation), ICP, HTCP, and SNMP messages. The specified address must correspond to one of the system's network interfaces. You should use this directive only if your system has multiple IP addresses.

The default value of 255.255.255.255 causes Squid to use the incoming address for sending, as well as receiving. In other words, rather than creating a separate UDP socket for sending, Squid sends and receives messages through a single socket.

If you use this directive, it must have a different value than *udp_incoming_address*. Squid can't create two UDP sockets bound to the same IP address and port number.

Syntax	<code>udp_outgoing_address ip-address</code>
Default	<code>udp_outgoing_address 255.255.255.255</code>
Example	<code>udp_outgoing_address 192.168.5.6</code>
Related	<code>udp_outgoing_address</code> , <code>icp_port</code> , <code>htcp_port</code> , <code>snmp_port</code>

cache_peer

Okay, this one's long, so hang on...

This directive defines your neighbor caches and tells Squid how to communicate with them. See Chapter 10 for the lowdown on neighbor caches.

The first argument is the neighbor cache's hostname, or IP address. You can safely use hostnames here because Squid doesn't block while resolving them. In fact, Squid periodically re-resolves the hostname so that if the address changes, you won't need to restart. Neighbor hostnames must be unique; you can't have two neighbors with the same name, even if they have different ports.

The second argument specifies the type of neighbor cache. The choices are *parent*, *sibling*, or *multicast*. Recall from Chapter 10 that for a multicast neighbor, Squid sends ICP queries only to the neighbor's IP address, which must be a valid multicast address. Squid makes HTTP requests to parents and siblings but never to a multicast neighbor.

The third and fourth arguments are HTTP and ICP/HTCP port numbers. The HTTP port number corresponds to the neighbor cache's *http_port* (or equivalent) setting. A value of 0 for the ICP/HTCP port disables those protocols for the neighbor. If you add the *htcp* option (described in the subsequent paragraphs), Squid sends HTCP queries to the neighbor. Otherwise, Squid sends ICP queries. If you choose not to use ICP or HTCP, you must specify the neighbor as a parent cache.

This brings us to the options field. The *cache_peer* directive has numerous options, which can be very confusing:

proxy-only

Instructs Squid to not store any responses received from the neighbor. This is often useful when you have a cluster and don't want a resource to be stored on more than one cache.

weight=*n*

Allows you to weight parent caches artificially when using ICP/HTCP and all parents report a cache miss. Normally Squid selects the parent whose reply arrived first. In fact, it remembers which parent has the best round-trip time for the query. Squid actually divides the RTT by the weight, so that a parent with weight=2 has lower (better) round-trip times and should be selected more often.

ttl=*n*

An option for multicast neighbors only. It is the multicast TTL value to use for ICP queries and it controls how far away the ICP queries can travel. The valid range is 0–128. A larger value allows the multicast queries to travel farther and possibly be intercepted by outsiders. Use a lower number to keep the queries close to the source and within your network.

no-query

Disables ICP/HTCP for the neighbor. That is, your cache won't send any queries to the neighbor for cache misses. It is often used with the default option.

default

Specifies the neighbor as a suitable choice in the absence of other hints. Squid would prefer to forward a cache miss to a parent that is likely to have a cached copy of the particular resource. Sometimes Squid won't have any clues (e.g., if you disable ICP/HTCP with no-query). In these cases, Squid looks for a parent that has been marked as a default choice.

round-robin

A simple load-sharing technique. It only makes sense when you mark two or more parent caches as round-robin. Squid keeps a counter for each parent. When it needs to forward a cache miss, Squid selects the parent with the lowest counter.

multicast-responder

Tells Squid to expect ICP replies from the neighbor in response to multicast queries.

closest-only

Refers to Squid's *netdb* features. When your neighbor has enabled the network database, it may return ICMP RTT measurements in ICP miss replies. This option instructs Squid to select a parent based on the RTT between the parent and the origin server, rather than the RTT between your cache and the parent.

no-digest

Tells Squid not to request a Cache Digest from the neighbor. See Chapter 10.

no-netdb-exchange

Tells Squid not to request the neighbor's *netdb* database. Note, this refers to the bulk transfer of the RTT measurements, not the inclusion of these measurements in ICP miss replies.

no-delay

Tells Squid to ignore any delay pools settings for requests to the neighbor. See Appendix C.

login=*credentials*

Instructs Squid to send authentication credentials to the neighbor. This option has three different formats, which I've fully described in Chapter 10.

`cache_peer_domain`

`connect-timeout=n`

Specifies how long Squid should wait when establishing a TCP connection to the neighbor. Without this option, the timeout is taken from the global `connect_timeout` directive. By using a lower timeout, Squid gives up on the neighbor quickly and tries forwarding the request elsewhere.

`digest-url=url`

Specifies the URL for the neighbor's Cache Digest. Without this option, Squid assumes the digest URL is `http://neighbor.host.name:port/squid-internal-periodic/store_digest`.

`allow-miss`

Instructs Squid to omit the `Cache-control: only-if-cached` directive for requests sent to a sibling. You should use this only if the neighbor is using the `icp_hit_stale` and isn't using a `miss_access` list.

`max-conn`

Places a limit on the number of simultaneous connections that Squid can open to the neighbor. When this limit is reached, Squid excludes the neighbor from its selection algorithm.

`htcp`

Tells Squid to send HTCP, instead of ICP, queries to this neighbor. If you add this option, don't forget to also change the port number. Squid uses 4827 as the default HTCP port. See Chapter 10.

`carp-load-factor=f`

Tells Squid that this neighbor is a member of a CARP array. The load factor value specifies the fraction of requests that this neighbor will receive. The load factor values for all neighbors must add up to 1.0. See Chapter 10.

Syntax `cache_peer hostname type http-port icp-port [options]`

Default No default

Example `cache_peer bigcache.isp.net parent 3128 3130`
`cache_peer medcache.isp.net sibling 3128 4827 htcp`
`cache_peer 172.16.45.111 parent 3128 0 no-query default`

Related `cache_peer_access`, `http_port`, `icp_port`, `htcp_port`, `icp_query_timeout`, `dead_peer_timeout`, `peer_connect_timeout`, `cache_peer_domain`, `neighbor_type_domain`

cache_peer_domain

This directive allows you to restrict forwarded requests by their domain names. For example, you can make sure that URIs in a certain domain never go to your parent cache. Similarly, you can make sure that requests for only a few specific domain names are sent to a neighbor. The `cache_peer_domain` directive has been largely superseded by `cache_peer_access`, which is much more flexible.

Following the neighbor's hostname, you can specify a list of domain names. These are searched in order, until Squid finds a match. A match means that the request can be sent to the neighbor, unless you prefix the domain name with ! ("not"). For example, `.foo.com` means "allow .foo.com," while `!.bar.net` means "disallow .bar.net." If none of the listed

domains match the URL, the default action (allow or deny) is the opposite of the last one in the list.

Note, the domain name matching algorithm is somewhat tricky. See the description in Chapter 6.

Syntax	<code>cache_peer_domain hostname domain ...</code>
Default	No default
Example	<code>cache_peer_domain bigcache.isp.net .net .org</code> <code>cache_peer_domain aol.web-cache.net !ads.aol.com .aol.com</code>
Related	<code>cache_peer</code> , <code>cache_peer_access</code> , <code>neighbor_type_domain</code>

neighbor_type_domain

You can use this directive to modify the relationship for a neighbor cache selectively. For example, you may have a sibling neighbor that allows you to fetch misses for certain, nearby domains. The `neighbor_type_domain` option overrides the type given in the `cache_peer` line for requests that match the listed domains.

The syntax and algorithms for matching domain names are identical to the `cache_peer_domain` directive.

Syntax	<code>neighbor_type_domain parent sibling hostname domain ...</code>
Default	No default
Example	<code>neighbor_type_domain bigcache.isp.net parent .customer.isp.net</code>
Related	<code>cache_peer</code> , <code>cache_peer_domain</code>

icp_query_timeout

When Squid sends an ICP/HTCP query to one or more neighbors, it waits some amount of time for the replies to arrive. Because the messages are unreliable UDP datagrams, the queries and/or replies may never arrive. Squid automatically figures out how long to wait for ICP/HTCP replies. For a particular query, the timeout is twice the mean of how long it took for recent replies to arrive. In other words, Squid averages the query RTT values from previous requests, doubles it, and waits that amount of time. This algorithm works best when all your neighbors have about the same RTT, and when network conditions are consistent.

You can override this algorithm with the `icp_query_timeout` directive. Instead of dynamically calculating the timeout, Squid waits a fixed amount of time for every ICP/HTCP query.

Syntax	<code>icp_query_timeout milliseconds</code>
Default	No default
Example	<code>icp_query_timeout 1500</code>
Related	<code>icp_port</code> , <code>htcp_port</code> , <code>maximum_icp_query_timeout</code> , <code>mcast_icp_query_timeout</code> , <code>dead_peer_timeout</code>

maximum_icp_query_timeout

maximum_icp_query_timeout

I described Squid's dynamic ICP/HTCP timeout algorithm under *icp_query_timeout*. If you'd like to use that algorithm, but wish to place an upper limit on the timeout, use the *maximum_icp_query_timeout* directive instead. Rather than a fixed timeout, Squid uses the dynamic timeout but makes sure it doesn't exceed the limit that you specify.

Syntax	<code>maximum_icp_query_timeout milliseconds</code>
Default	No default
Example	<code>maximum_icp_query_timeout 3000</code>
Related	<code>icp_port, http_port, icp_query_timeout</code>

mcast_icp_query_timeout

When you use multicast ICP, Squid doesn't know in advance how many multicast-capable neighbors are listening for its messages. Squid determines this by sending periodic probes to the multicast group and counting the number of replies. Squid uses this count when waiting for replies to real multicast queries.

The *mcast_icp_query_timeout* directive specifies how long Squid should wait when counting replies to its fake probe queries. Why not just use this timeout when sending real multicast ICP queries? The reason is that Squid might be sending queries to both multicast and unicast neighbors.

The *mcast_icp_query_timeout* directive essentially controls how long Squid waits for replies to real multicast queries. Let's say you have an ICP multicast group with 10 neighbor caches, and that it typically takes 3000 msec for all 10 replies to arrive but only takes 1000 msec to receive 5 replies. If you set *mcast_icp_query_timeout* to 1000 msec, Squid's periodic probes will count 5 neighbors. Then, for a real multicast ICP query, Squid waits for only 5 replies from multicast responders. On average, this should take only 1000 milliseconds.

Another nice feature of this algorithm is that Squid does the right thing if, for some reason, all your multicast neighbors stop responding. In that case, Squid counts zero neighbors and doesn't wait for any replies from multicast responders.



Squid doesn't send multicast HTCP queries.

Syntax	<code>mcast_icp_query_timeout milliseconds</code>
Default	<code>mcast_icp_query_timeout 2000</code>
Example	<code>mcast_icp_query_timeout 750</code>
Related	<code>icp_port, icp_query_timeout</code>

dead_peer_timeout

This is another directive that controls the way Squid waits for ICP/HTCP replies. Squid marks each of its peers as either dead (down) or alive (up). Squid uses ICP/HTCP replies (and other techniques) to determine a peer's state. If Squid doesn't receive any replies for the time specified by *dead_peer_timeout*, the peer is declared dead.

When a peer is declared dead, Squid continues to send it ICP/HTCP queries. However, it doesn't expect to receive replies. That is, a dead peer isn't included in the algorithm that decides when all ICP replies have been received. As soon as Squid receives an ICP/HTCP reply from a dead peer, its state is changed to alive.

Squid tends to be paranoid about the state of its peers. Additionally, Squid doesn't proactively monitor the peers when there are no client requests. When Squid has no occasion to send ICP/HTCP queries, the state of the peer is unknown. If Squid doesn't send any ICP/HTCP queries for an amount of time longer than *dead_peer_timeout*, Squid treats the peer as dead.

Syntax	<i>dead_peer_timeout time-specification</i>
Default	<i>dead_peer_timeout 10 seconds</i>
Example	<i>dead_peer_timeout 30 seconds</i>
Related	<i>icp_port, http_port, icp_query_timeout</i>

hierarchy_stoplist

Every HTTP request that Squid receives is marked as either hierarchical or nonhierarchical. This terminology is somewhat confusing. A request is hierarchical when there is a possibility it could be a cache hit in one of the neighbors. In other words, if the information in the request indicates that the response may be cachable, the request is hierarchical. A request is marked nonhierarchical when Squid thinks there is no chance of getting a hit from a neighbor.

Squid uses the hierarchical flag to decide whether or not it should query neighbors for the request. If the request is hierarchical, Squid may perform ICP/HTCP queries, or use Cache Digests, to locate cache hits in neighbors. Otherwise, Squid may forward the request directly to the origin server or select a parent based on some other technique.

Squid has a few hardcoded rules that determine if a request is hierarchical. For example, only GET requests are hierarchical. Squid never expects cache hits on non-GET requests. Another rule is that requests including authentication information are nonhierarchical. The *hierarchy_stoplist* directive allows you to customize the algorithm further. The stoplist is simply a list of strings. Squid searches the requested URL for these strings. The string comparison is case-sensitive. In the case of a match, the request becomes nonhierarchical. The default configuration is to search for *cgi-bin* and *?* so that queries and other CGI responses aren't hierarchical.

Note that the hierarchical flag determines only whether or not Squid queries its neighbor caches. It doesn't determine which requests must, or must not, be sent to parent caches. The *always_direct* and *never_direct* access lists have that responsibility.

no_cache

Syntax	<code>hierarchy_stoplist string ...</code>
Default	<code>hierarchy_stoplist cgi-bin ?</code>
Example	<code>hierarchy_stoplist .cgi</code> <code>hierarchy_stoplist http://www.mysite.org</code>
Related	<i>always_direct, never_direct</i>

no_cache

no_cache is a sequence of access control rules (see Chapter 6) that specify responses that must not be cached by Squid. Of course, Squid has some hardcoded rules for responses that must not be cached according to the HTTP RFC. The *no_cache* rules are in addition to those.

The *no_cache* syntax is a little tricky. You must use *deny* for rules where the response must not be cached. Consider this example:

```
acl GoodStuff url_regex /foo/bar/
acl BadStuff url_regex /bar/
no_cache allow GoodStuff
no_cache deny BadStuff
```

Here, a URL containing `/foo/bar/` may be cached, but any other URL containing only `/bar/` isn't cached. The meaning of the *allow* and *deny* might be the opposite of what you expect. Just remember that *deny* carries the same negative connotation as "not caching" something.

Syntax	<code>no_cache allow deny [!]ACLname ...</code>
Default	No default
Example	<code>acl LocalServers dst 192.168.8.0/24</code> <code>no_cache deny LocalServers</code>
Related	<i>always_direct, never_direct, http_access</i>

cache_access_log

This is the location of Squid's *access.log*, which contains one entry for each client request. See Chapter 13 for the details. If you want to disable the access log, set this to */dev/null*.

Syntax	<code>cache_access_log pathname</code>
Default	<code>cache_access_log \$prefix/var/logs/access.log</code>
Example	<code>cache_access_log /var/log/squid-access.log</code>
Related	<i>emulate_httpd_log, cache_log, cache_store_log, log_ip_on_direct, logfile_rotate</i>

cache_log

This log file contains various operational and debugging messages from Squid. See Chapter 13 for more information. If you want to disable *cache.log*, set this directive to */dev/null*.

Syntax	<code>cache_log pathname</code>
Default	<code>cache_log \$prefix/var/logs/cache.log</code>
Example	<code>cache_log /var/log/squid.log</code>
Related	<code>debug_options</code> , <code>cache_access_log</code> , <code>cache_store_log</code> , <code>logfile_rotate</code>

cache_store_log

The *store.log* contains details about Squid's interaction with the disk cache. You'll see entries as objects are stored to disk, read from disk, and removed from the cache. See Chapter 13 for the details. You can disable this log by setting it to none.

Syntax	<code>cache_store_log pathname</code>
Default	<code>cache_store_log \$prefix/var/logs/store.log</code>
Example	<code>cache_store_log /var/log/squid-store.log</code>
Related	<code>cache_access_log</code> , <code>cache_log</code> , <code>logfile_rotate</code>

cache_swap_log

Each cache directory has its own swap log file. These are binary-format journal files Squid uses to rebuild the in-memory indexes when it starts up. Each swap log file is located in the corresponding cache directory by default. If you use this option, Squid puts all swap log files in one directory. See Chapter 13 for more information.

Syntax	<code>cache_swap_log pathname</code>
Default	swap.state in each <i>cache_dir</i>
Example	<code>cache_swap_log /var/log/squid-swap-state</code>
Related	<code>cache_store_log</code> , <code>logfile_rotate</code> , <code>cache_dir</code>

emulate_httpd_log

Squid uses its own native format for the *access.log* by default. If you enable this directive, the access log is written in the HTTPD common log file format. Often useful when Squid is accelerating an origin server site.

Syntax	<code>emulate_httpd_log on off</code>
Default	<code>emulate_httpd_log off</code>
Example	<code>emulate_httpd_log on</code>
Related	<code>cache_access_log</code> , <code>httpd_accel_host</code>

log_ip_on_direct

By default, Squid puts origin server IP addresses into the ninth field of the *access.log*. If you enable this directive, Squid puts the origin server hostname there instead.

cache_dir

Syntax	log_ip_on_direct on off
Default	log_ip_on_direct on
Example	log_ip_on_direct off
Related	cache_access_log

cache_dir

This directive instructs Squid where, and how, to store cached objects on disk. See Chapter 7 for the details on cache directories.

The second parameter selects the storage scheme. Your choices are ufs, aufs, diskd, coss, and null. To use any scheme other than ufs, you must use the `--enable-storeio` option with `./configure`. See Chapter 3.

The third parameter is the amount of disk space to use for the cache. The units are in megabytes.

The fourth and fifth parameters are the number of L1 and L2 directories. Don't change these values for directories that already contain cached objects.

Some *cache_dir* schemes have additional, optional parameters. Refer to the scheme-specific sections in Chapter 8.

Syntax	cache_dir <i>scheme directory size-MB L1 L2 [options...]</i>
Default	cache_dir ufs <i>\$prefix/var/cache</i> 100 16 256
Example	cache_dir ufs /cache0 3072 16 128
Related	cache_replacement_policy, cache_mem

cache_mem

Squid uses memory to store recently received objects and to buffer active responses. This directive specifies the amount of memory to use for storing these objects.



This directive doesn't entirely control the size of the Squid process. See Appendix B for additional information.

Syntax	cache_mem <i>bytes-specification</i>
Default	cache_mem 8 MB
Example	cache_mem 16 MB
Related	cache_dir, maximum_object_size_in_memory, memory_replacement_policy

cache_swap_low

This directive, along with *cache_swap_high* controls the replacement of objects stored on disk. It is a percentage of the maximum cache size, which comes from the sum of all *cache_dir* sizes. See Chapter 7 for additional information.

Syntax	<code>cache_swap_low percent</code>
Default	<code>cache_swap_low 90</code>
Example	<code>cache_swap_low 85</code>
Related	<code>cache_swap_high</code> , <code>cache_dir</code>

cache_swap_high

See the description for `cache_swap_low`. Note that changing `cache_swap_high` probably won't have a big impact on Squid's disk usage. See Chapter 7 for additional information.

Syntax	<code>cache_swap_high percent</code>
Default	<code>cache_swap_high 95</code>
Example	<code>cache_swap_high 99</code>
Related	<code>cache_swap_low</code> , <code>cache_dir</code>

maximum_object_size

This directive places a limit on the largest object that Squid can store on disk. Responses larger than this size aren't cached. See Chapter 7 for additional information.

Syntax	<code>maximum_object_size bytes-specification</code>
Default	<code>maximum_object_size 4096 MB</code>
Example	<code>maximum_object_size 250 MB</code>
Related	<code>minimum_object_size</code> , <code>maximum_object_size_in_memory</code> , <code>reply_body_max_size</code>

minimum_object_size

With this directive, you can also place lower limits on the size of cached objects. Responses smaller than this size aren't stored on disk or in memory. See Chapter 7 for additional information.

Syntax	<code>minimum_object_size bytes-specification</code>
Default	<code>minimum_object_size 0 bytes</code>
Example	<code>minimum_object_size 300 bytes</code>
Related	<code>maximum_object_size</code>

maximum_object_size_in_memory

This directive allows you to control the size of objects stored in memory. Objects that are larger than this value aren't kept in memory. See Chapter 7 for additional information.

Syntax	<code>maximum_object_size_in_memory bytes-specification</code>
Default	<code>maximum_object_size_in_memory 8 KB</code>
Example	<code>maximum_object_size_in_memory 12 KB</code>
Related	<code>cache_mem</code> , <code>maximum_object_size</code>

cache_replacement_policy

This directive controls the replacement policy for Squid's disk cache. Version 2.5 offers three different replacement policies: least recently used (LRU), greedy dual-size frequency (GDSF), and least frequently used with dynamic aging (LFUDA). Note that the keywords (lru, GDSF, etc.) are case-sensitive! See Chapter 7 for additional information.

Syntax `cache_replacement_policy lru`
 `cache_replacement_policy heap GDSF|LFUDA|LRU`
Default `cache_replacement_policy lru`
Example `cache_replacement_policy heap GDSF`
Related *memory_replacement_policy, cache_dir*

memory_replacement_policy

This directive controls the replacement policy for objects cached in memory. See Chapter 7 for additional information.

Syntax `memory_replacement_policy lru`
 `memory_replacement_policy heap GDSF|LFUDA|LRU`
Default `memory_replacement_policy lru`
Example `memory_replacement_policy heap LFUDA`
Related *cache_replacement_policy, cache_mem*

store_dir_select_algorithm

This directive controls the algorithm Squid uses when selecting a *cache_dir* for a new cache file. The possible choices are: least-load and round-robin. See Chapter 7 for additional information.

Syntax `store_dir_select_algorithm round-robin|least-load`
Default `store_dir_select_algorithm least-load`
Example `store_dir_select_algorithm round-robin`
Related *cache_dir*

mime_table

Squid uses the information in this file for FTP and Gopher requests. Unlike HTTP, these protocols don't inform clients about the type of data they transfer. When Squid gateways the response from an FTP server to an HTTP client, it must insert Content-Type and other headers. Squid uses the MIME table file to convert filename extensions into:

- Values for the Content-Type header
- Icons that are displayed for directory listings

- Content-Encoding header values for compressed data
- Transfer type options for FTP servers, either *image* or *ascii*; this corresponds to the TYPE command in the FTP protocol

Please refer to the sample *mime.conf* for an explanation of the format of this file.

Syntax `mime_table pathname`
 Default `mime_table $prefix/etc/mime.conf`
 Example `mime_table /usr/local/squid/etc/my-mime-types.txt`

ipcache_size

Squid's IP cache holds recent DNS name-to-address lookups. This directive limits the number of names in the cache. Each IP cache entry uses a relatively small amount of memory, so you can safely increase this limit to 10,000 or more.

Syntax `ipcache_size count`
 Default `ipcache_size 1024`
 Example `ipcache_size 5000`
 Related `ipcache_low`, `ipcache_high`, `fqdn_cache_size`

ipcache_low

This directive controls the IP cache LRU replacement algorithm. The replacement function runs periodically and removes the least recently used IP cache entries until reaching this low watermark. You should have almost no reason to change this value. You'd be better off changing *ipcache_size* instead.

Syntax `ipcache_low percent`
 Default `ipcache_low 90`
 Example `ipcache_low 95`
 Related `ipcache_size`, `ipcache_high`

ipcache_high

This directive is essentially unused in current versions of Squid. The LRU replacement routine uses only *ipcache_low*. The only time that Squid uses *ipcache_high* is when calculating the hash table size for the IP cache at startup.

Syntax `ipcache_high percent`
 Default `ipcache_high 95`
 Example `ipcache_high 99`
 Related `ipcache_size`, `ipcache_low`

`fqdn_cache_size`

fqdn_cache_size

Squid's FQDN cache holds recent DNS address-to-name lookups. However, Squid makes these reverse DNS lookups only when you enable the *log_fqdn* directive or use a *dstdomain* ACL. This directive limits the number of names in the cache. Each FQDN cache entry uses a relatively small amount of memory, so you can safely increase this limit to 10,000 or more.

Syntax	<code>fqdn_cache_size count</code>
Default	<code>fqdn_cache_size 1024</code>
Example	<code>fqdn_cache_size 6000</code>
Related	<i>ipcache_size, log_fqdn</i>

log_mime_hdrs

When you enable this directive, Squid writes the HTTP request and response headers to the *access.log* file. The headers appear as two additional fields on each line. All whitespace and other special characters are encoded with URL-style escape codes. Enabling this option may assist in tracking down certain problems. Note that HTTP headers are relatively large (a few hundred bytes each). Logging them dramatically increases the size of your *access.log* file.

Syntax	<code>log_mime_hdrs on off</code>
Default	<code>log_mime_hdrs off</code>
Example	<code>log_mime_hdrs on</code>
Related	<i>cache_access_log</i>

useragent_log

This directive causes Squid to create a log file of User-Agent strings. The file contains three fields: client identifier, timestamp, and user-agent string. The client identifier is an IP address, unless you enable the *log_fqdn* directive, in which case it is a hostname if one is available. Squid writes an entry for every HTTP request that has a User-Agent header. Unlike *access.log*, entries are written to this file when the request is received.

Syntax	<code>useragent_log pathname</code>
Default	No default
Example	<code>useragent_log /usr/local/squid/var/logs/useragent.log</code>
Related	<i>log_fqdn, cache_access_log, referer_log</i>

referer_log

This directive causes Squid to create a log file of Referer values from client requests. The file contains four fields: time, client identifier, Referer value, and the URI request. For

example, when a client requests the image *foo.png* embedded in an *index.html*, the referer log contains:

```
1068047502.377 192.168.1.2 /index.html /foo.png
```

Squid writes an entry for every HTTP request that has a Referer header. Unlike *access.log*, entries are written to this file when the request is received.

Syntax	referer_log <i>pathname</i>
Default	No default
Example	referer_log /usr/local/squid/var/logs/referer.log
Related	log_fqdn, cache_access_log, useragent_log

pid_filename

This is the file in which Squid writes its process ID (PID) number. Squid uses the PID file in a couple of ways. First, it looks for and reads this file when starting. If the file exists and contains a valid PID, Squid reports it is already running under that PID so that you don't accidentally start Squid twice. The PID file is also read when you use one of the *-k* commands such as *squid -k rotate*.

You probably don't need to worry about this directive unless you actually do want to run two (or more) Squid processes on the same machine. Each instance of Squid requires a unique PID filename.

Syntax	pid_filename <i>pathname</i>
Default	pid_filename <i>\$prefix</i> /var/logs/squid.pid
Example	pid_filename /var/run/squid.pid

debug_options

This directive controls the amount of debugging information written to *cache.log*. Each source code module has a section number. Individual debugging statements in the code have a level. Higher debugging levels correspond to more verbose debugging. For a list of section numbers, refer to Chapter 16 or the *doc/debug-sections.txt* file in the source distribution.

Syntax	debug_options <i>section,level</i> ...
Default	debug_options ALL,1
Example	debug_options ALL,1 42,5
Related	cache_log

log_fqdn

This directive controls whether or not Squid places client IP addresses or hostnames in the log files. By default Squid writes the IP address. If you enable this feature, Squid queries the DNS for client hostnames or fully qualified domain names (FQDN). These address-to-name lookups sometimes take a long time. Squid never postpones logging to wait for an

client_netmask

answer. If the FQDN isn't available when Squid is ready to write the log entry, it uses the IP address.

Syntax	<code>log_fqdn on off</code>
Default	<code>log_fqdn off</code>
Example	<code>log_fqdn on</code>
Related	<code>cache_access_log</code> , <code>useragent_log</code> , <code>referer_log</code> , <code>fqdn_cache_size</code> , <code>client_netmask</code>

client_netmask

This directive is available to provide privacy for users. When Squid writes *access.log* and other log files, it applies this mask to the client's IP address. For example, if you set the netmask to 255.255.255.0, Squid logs a request from 1.2.3.0 instead of 1.2.3.4. Thus, if someone manages to read the log file, they know only approximately, not exactly, which host (or user) made each request.

If you use *log_fqdn*, Squid applies the *client_netmask* before issuing the DNS lookup. For example, Squid will try to find a hostname record for 1.2.3.0 instead of 1.2.3.4.

Syntax	<code>client_netmask IPv4-netmask</code>
Default	<code>client_netmask 255.255.255.255</code>
Example	<code>client_netmask 255.255.255.0</code>
Related	<code>cache_access_log</code> , <code>useragent_log</code> , <code>referer_log</code> , <code>log_fqdn</code>

ftp_user

This directive contains the password Squid sends when logging in to anonymous FTP servers. Convention dictates that anonymous FTP clients send the user's email address as the login password. Most anonymous FTP servers accept an abbreviated form with only a username followed by @ (e.g., joe_blow@). You probably won't need to change this directive unless you encounter a very picky FTP server.

Syntax	<code>ftp_user email-address</code>
Default	<code>ftp_user Squid@</code>
Example	<code>ftp_user joe_blow@company.com</code>
Related	<code>ftp_list_width</code> , <code>ftp_passive</code>

ftp_list_width

This directive controls the width of the filename column in FTP directory listings that Squid generates. The default value is chosen so that the listings fit inside a typical browser window. This also means that long filenames may be truncated. If you'd like to see more characters in long filenames, increase this value.

Syntax	<code>ftp_list_width character-count</code>
Default	<code>ftp_list_width 32</code>

Example `ftp_list_width 64`
 Related `ftp_user`

ftp_passive

Squid normally uses FTP's so-called passive mode for file transfers. This means that the FTP server creates a TCP socket for data transfer and waits for the client to connect. Passive mode works much better through most Internet firewalls. The alternative is to have the FTP client (Squid in this case) create a TCP socket and wait for a connection from the server. Most likely, you'll never have problems with FTP passive mode. However, you can force nonpassive operation by turning off this directive.

Syntax `ftp_passive on|off`
 Default `ftp_passive on`
 Example `ftp_passive off`
 Related `ftp_user, ftp_list_width, ftp_sanitycheck`

ftp_sanitycheck

When using FTP passive mode (the default), the FTP server tells Squid the IP address and port number for each data connection. Squid normally checks the given values to make sure they match the server's IP address. In other words, an FTP server should always use its own IP address in the PASV reply message. If it doesn't, Squid complains to *cache.log* and attempts a data connection with the PORT command. Disable the *ftp_sanitycheck* directive if you want Squid to skip the IP address sanity check.

Syntax `ftp_sanitycheck on|off`
 Default `ftp_sanitycheck on`
 Example `ftp_sanitycheck off`
 Related `ftp_passive`

cache_dns_program

Recall that, by default, Squid uses an internal DNS client implementation. However, you also have the choice of using an external helper program to perform DNS lookups. This choice must be made when you run `./configure`, with the `--disable-internal-dns` option.

If you elect to use the external DNS, this directive specifies the pathname to the `dnsserver` program. This is a misleading name in that the program isn't really a DNS server. It is more like a DNS proxy. The program reads hostnames (or IP addresses) from Squid, executes the necessary lookup, and writes IP addresses (or hostnames) back.

You probably won't need to use this directive, unless you move the Squid binaries after running `make install` or you're inclined to experiment with the external DNS program.

Syntax `cache_dns_program pathname`
 Default `cache_dns_program $prefix/libexec/dnsserver`

dns_children

Example `cache_dns_program /usr/local/squid/libexec/better_dnsserver`
Related `dns_children`

dns_children

This directive is meaningful only with the `--disable-internal-dns` option.

The interface between Squid and the external DNS program is built around the `gethostbyname()` function. Squid writes a request to a `dnsserver` process, which performs the query. The `gethostbyname()` call blocks the process until the reply arrives. This is why Squid can't use the function internally.

Each `dnsserver` handles only one request at a time, so you need enough of them to handle the load from your cache. Unfortunately, you may need to experiment with different values to discover the appropriate setting for your particular situation. In theory, you can calculate the number of child processes if you know the rate of DNS lookups and how long lookups take on average. Unfortunately, both values can vary significantly over time.

Squid writes a warning into `cache.log` if you have too few `dnsserver` child processes. If all helper processes are busy, Squid queues up new lookups. If the queue grows too large, Squid emits an error message and exits. Thus, too many child processes are better than too few.

You can use the `dns` entry in the cache manager menu to see `dnsserver` utilization information. Requests are always sent to the first idle process, so you can see if some processes never receive any DNS lookup requests. In that case you may want to lower the `dns_children` value.

Why doesn't Squid just create and destroy child processes as necessary? The primary reason is that the creation of a child process, via `fork()`, is a relatively "heavy" operation. It may introduce significant delays for active HTTP requests. A Squid process typically consumes a lot of memory. In some cases, `fork()` may fail due to lack of available memory or swap space. Rather than try to fix all these issues with the external DNS implementation, Squid can read and write DNS messages internally.

Syntax `dns_children number`
Default `dns_children 5`
Example `dns_children 16`
Related `cache_dns_program`

dns_retransmit_interval

This directive is meaningful only when you use the internal DNS implementation (the default).

This directive is the initial retransmission interval for unacknowledged DNS queries. Each time Squid retransmits a DNS query, it's sent to the next DNS server in the list. If none of the servers answer, Squid starts at the top of the list again and doubles the retransmit interval.

Syntax `dns_retransmit_interval time-specification`
 Default `dns_retransmit_interval 5 seconds`
 Example `dns_retransmit_interval 10 seconds`
 Related `dns_timeout`

dns_timeout

This directive is meaningful only when you use the internal DNS implementation (the default).

This directive is the total amount of time that Squid waits for a DNS answer. If the timeout occurs, Squid returns an error message to the user.

Syntax `dns_timeout time-specification`
 Default `dns_timeout 5 minutes`
 Example `dns_timeout 2 minutes`
 Related `dns_retransmit_interval`

dns_defnames

This directive is meaningful only with the `--disable-internal-dns` option.

By default, Squid's `dnsserver` program doesn't attempt to expand single-word hostnames (such as `www`) into fully qualified domain names. If your users are accustomed to using single-word hostnames, you may want to enable this directive.

Syntax `dns_defnames on|off`
 Default `dns_defnames off`
 Example `dns_defnames on`
 Related `append_domain`

dns_nameservers

By default, Squid sends DNS queries to the name servers listed in the `/etc/resolv.conf` file. If you want Squid to use a different set of name servers, you can specify them with this directive. Of course, you can also just change your `resolv.conf` file.

Syntax `dns_nameservers ip-address ...`
 Default `No default`
 Example `dns_nameservers 127.0.0.1 192.168.0.1`

hosts_file

When you use the internal DNS implementation (the default), Squid always uses the DNS name servers to resolve names and addresses. The external `dnsserver` program, on the other hand, may check a local database—the *hosts file*—before querying the DNS. With

diskd_program

this directive, you can make Squid preload the contents of a *hosts* file into its IP and FQDN caches.

Squid rereads the *hosts* file when you send it the reconfigure signal (`squid -k reconfigure`).

If you configure the *append_domain* directive, it's appended to any single-component names in the *hosts* file.

Syntax	<code>hosts_file pathname</code>
Default	No default
Example	<code>hosts_file /usr/local/squid/etc/hosts</code>
Related	<i>dns_defnames, append_domain</i>

diskd_program

This is the pathname to the diskd helper program. It gets executed for each *cache_dir* of type *diskd*.

Syntax	<code>diskd_program pathname</code>
Default	<code>diskd_program \$prefix/libexec/diskd</code>
Example	<code>diskd_program /usr/local/squid-2.4/libexec/squid/diskd</code>
Related	<i>cache_dir</i>

unlinkd_program

This is the pathname to the unlinkd program. By executing the unlink operations in this external process, Squid's performance improves significantly. You can disable the external unlinker with the `--disable-unlinkd` option to `./configure`.

Syntax	<code>unlinkd_program pathname</code>
Default	<code>unlinkd_program \$prefix/libexec/unlinkd</code>
Example	<code>unlinkd_program /usr/local/squid-2.4/libexec/unlinkd</code>

pinger_program

Squid uses the pinger program to send ICMP pings to origin server sites. Squid uses these ICMP measurements to estimate network proximity. Note that the pinger program must be installed as `setuid root` because it opens a raw ICMP socket. To enable the ICMP measurement features, use the `./configure --enable-icmp` option.

Syntax	<code>pinger_program pathname</code>
Default	<code>pinger_program \$prefix/libexec/pinger</code>
Example	<code>pinger_program /usr/local/squid-2.4/libexec/pinger</code>
Related	<i>netdb_low, netdb_high, netdb_ping_period</i>

redirect_program

This directive specifies the pathname of a redirector program. It must be executable by the Squid user ID. See Chapter 11.

Syntax	<code>redirect_program <i>pathname</i></code>
Default	No default
Example	<code>redirect_program /usr/local/squid/libexec/my_redirector</code>
Related	<code>redirect_children</code> , <code>redirect_rewrites_host_header</code> , <code>redirector_access</code> , <code>redirector_bypass</code>

redirect_children

This directive specifies how many redirector processes Squid should start. Client requests are written to the first idle redirector process. Squid warns you (via *cache.log*) when all processes are simultaneously busy. If you see this warning, you should increase the number of child processes and restart Squid.

Syntax	<code>redirect_children <i>number</i></code>
Default	<code>redirect_children 5</code>
Example	<code>redirect_children 20</code>
Related	<code>redirect_program</code> , <code>sleep_after_fork</code> , <code>redirector_bypass</code>

redirect_rewrites_host_header

Squid normally updates a request's Host header when using a redirector. If you use Squid as a surrogate (HTTP accelerator), you might want to disable this behavior by setting this directive to off.

Syntax	<code>redirect_rewrites_host_header on off</code>
Default	<code>redirect_rewrites_host_header on</code>
Example	<code>redirect_rewrites_host_header off</code>
Related	<code>httpd_accel_single_host</code>

redirector_access

If you use this directive, only the requests that match the access list rules are sent to the redirector processes. Without any *redirector_access* rules, all requests are sent to the redirector processes.

Syntax	<code>redirector_access allow deny [!]ACLname ...</code>
Default	No default
Example	<code>acl Foo src 192.168.1.0/24</code> <code>acl All src 0/0</code> <code>redirector_access deny Foo</code> <code>redirector_access allow All</code>
Related	<code>acl</code> , <code>http_access</code>

redirector_bypass

Squid uses a pool of redirectors to service client requests. This directive determines Squid's behavior when all redirectors in the pool are busy. Normally, Squid queues subsequent requests, waiting for one of the redirectors to become free. If the queue becomes too large, Squid exits with a fatal message. If you enable this directive, however, Squid simply skips the redirection step if all redirectors are busy.

Syntax	<code>redirector_bypass on off</code>
Default	<code>redirector_bypass off</code>
Example	<code>redirector_bypass on</code>
Related	<code>redirect_program</code> , <code>redirector_access</code>

auth_param

The `auth_param` directive controls almost every aspect of Squid's external user authentication interface. Squid currently supports three authentication schemes: Basic, Digest, and NTLM. Basic authentication support is compiled by default. For the others, you must use the `--enable-auth` option with `./configure`.

Since the `auth_param` directive is very complex, I'm presenting it here as a separate directive for each combination of parameters.

Syntax	See the following subsections
Default	See the following subsections
Example	See the following subsections
Related	<code>authenticate_cache_garbage_interval</code> , <code>authenticate_ttl</code> , <code>authenticate_ip_ttl</code>

auth_param basic program

The command for the HTTP Basic authentication helper. You need to specify the full path-name to the program, plus any command-line options.

Syntax	<code>auth_param basic program <i>command</i> ...</code>
Default	No default
Example	<code>auth_param basic program /usr/local/squid/libexec/ncsa_auth /usr/local/squid/etc/ncsa_passwd</code>
Related	<code>auth_param basic children</code> , <code>auth_param basic realm</code> , <code>auth_param basic credentialsttl</code>

auth_param basic children

This is the number of Basic authentication helper processes Squid uses.

Syntax	<code>auth_param basic children <i>count</i></code>
Default	<code>auth_param basic children 5</code>
Example	<code>auth_param basic children 10</code>
Related	<code>auth_param basic program</code> , <code>auth_param basic realm</code> , <code>auth_param basic credentialsttl</code>

auth_param basic realm

This is the Basic authentication realm Squid sends in 407 (Proxy Authentication Required) responses. User agents typically display the realm string to the user when requesting a username and password. Refer to RFC 2617, Section 2.

Syntax	<code>auth_param basic realm <i>string</i></code>
Default	No default
Example	<code>auth_param basic realm Squid proxy-caching web server</code>
Related	<code>auth_param basic program</code> , <code>auth_param basic children</code> , <code>auth_param basic credentialsttl</code>

auth_param basic credentialsttl

To reduce load on the external authentication processes, Squid caches successful answers for this amount of time. In other words, once a user is authenticated, Squid doesn't query the helper program again until this TTL expires. If you change the external database (e.g., password file), Squid may not notice the change until the cached credentials time out.

Syntax	<code>auth_param basic credentialsttl <i>time-specification</i></code>
Default	<code>auth_param basic credentialsttl 5 minutes</code>
Example	<code>auth_param basic credentialsttl 15 minutes</code>
Related	<code>auth_param basic program</code> , <code>auth_param basic children</code> , <code>auth_param basic realm</code>

auth_param digest program

As with Basic authentication, this specifies the command to execute for the external Digest authentication program.

Syntax	<code>auth_param digest program <i>command ...</i></code>
Default	No default
Example	<code>auth_param digest program /usr/local/squid/libexec/digest_auth /usr/local/squid/etc/digest_passwd</code>
Related	<code>auth_param digest children</code> , <code>auth_param digest realm</code> , <code>auth_param digest nonce_garbage_interval</code> , <code>auth_param digest nonce_max_duration</code> , <code>auth_param digest nonce_max_count</code>

auth_param digest children

This is the number of Digest authentication helper processes that Squid uses.

Syntax	<code>auth_param digest children <i>count</i></code>
Default	<code>auth_param digest children 5</code>
Example	<code>auth_param digest children 11</code>
Related	<code>auth_param digest program</code> , <code>auth_param digest realm</code> , <code>auth_param digest nonce_garbage_interval</code> , <code>auth_param digest nonce_max_duration</code> , <code>auth_param digest nonce_max_count</code>

auth_param digest realm

This is the Digest authentication realm that Squid sends in 407 (Proxy Authentication Required) responses. User agents typically display the realm string to the user when requesting a username and password. Refer to RFC 2617, Section 3.2.1.

auth_param

Syntax	<code>auth_param digest realm string</code>
Default	No default
Example	<code>auth_param digest realm Squid proxy-caching web server</code>
Related	<code>auth_param digest program</code> , <code>auth_param digest children</code> , <code>auth_param digest nonce_garbage_interval</code> , <code>auth_param digest nonce_max_duration</code> , <code>auth_param digest nonce_max_count</code>

auth_param digest nonce_garbage_interval

As I explained in Chapter 12, a nonce is a special string of data that changes from time to time. Its purpose is to prevent replay attacks with captured digest authentication data.

Squid maintains a cache of nonce values it has sent to clients requiring authentication. This cache must be pruned occasionally because nonce strings expire. This directive specifies how often Squid executes the garbage collection procedure for the nonce cache.

If Squid is very busy, you may want to clean the nonce cache more frequently to reduce the amount of time spent in the garbage collection function each time it runs.

Syntax	<code>auth_param digest nonce_garbage_interval time-specification</code>
Default	<code>auth_param digest nonce_garbage_interval 5 minutes</code>
Example	<code>auth_param digest nonce_garbage_interval 5 minutes</code>
Related	<code>auth_param digest program</code> , <code>auth_param digest children</code> , <code>auth_param digest realm</code> , <code>auth_param digest nonce_max_duration</code> , <code>auth_param digest nonce_max_count</code>

auth_param digest nonce_max_duration

This directive specifies how long a Digest nonce value remains valid. It is similar to the `credentialsttl` directive for Basic authentication.

If an attacker captures the client's digest authentication headers from an HTTP request, a simple replay attack provides authenticated access to Squid until the nonce value times out or until the maximum usage count is reached. Decrease this value to reduce that risk.

Syntax	<code>auth_param digest nonce_max_duration time-specification</code>
Default	<code>auth_param digest nonce_max_duration 5 minutes</code>
Example	<code>auth_param digest nonce_max_duration 30 minutes</code>
Related	<code>auth_param digest program</code> , <code>auth_param digest children</code> , <code>auth_param digest realm</code> , <code>auth_param digest nonce_garbage_interval</code> , <code>auth_param digest nonce_max_count</code> , <code>auth_param basic credentialsttl</code>

auth_param digest nonce_max_count

This directive specifies a limit on the number of requests for a Digest nonce value. If a client issues this many requests with the same nonce value, Squid invalidates it and causes a new one to be generated. See Section 4.3 of RFC 2617.

Syntax	<code>auth_param digest nonce_max_count count</code>
Default	<code>auth_param digest nonce_max_count 50</code>
Example	<code>auth_param digest nonce_max_count 50</code>
Related	<code>auth_param digest program</code> , <code>auth_param digest children</code> , <code>auth_param digest realm</code> , <code>auth_param digest nonce_garbage_interval</code> , <code>auth_param digest nonce_max_duration</code>

auth_param ntlm program

This directive specifies the command, including options, to execute for the external NTLM authentication program.

Syntax	<code>auth_param ntlm program <i>command</i></code>
Default	No default
Example	<code>auth_param ntlm program /usr/local/squid/libexec/ntlm_auth /usr/local/squid/etc/ntlm_db</code>
Related	<code>auth_param ntlm children</code> , <code>auth_param ntlm max_challenge_reuses</code> , <code>auth_param ntlm max_challenge_lifetime</code>

auth_param ntlm children

Specifies the number of NTLM authentication helper process that Squid uses.

Syntax	<code>auth_param ntlm children <i>count</i></code>
Default	<code>auth_param ntlm children 5</code>
Example	<code>auth_param ntlm children 14</code>
Related	<code>auth_param ntlm program</code> , <code>auth_param ntlm max_challenge_reuses</code> , <code>auth_param ntlm max_challenge_lifetime</code>

auth_param ntlm max_challenge_reuses

In Squid's NTLM implementation, the NTLM challenge token comes from the external helper process, rather than Squid itself. Each helper process generates its own challenge token. This directive specifies how many times each token may be reused. By default, the tokens are never reused. Challenge reuse is also subject to the `max_challenge_lifetime` restriction.

Syntax	<code>auth_param ntlm max_challenge_reuses <i>count</i></code>
Default	<code>auth_param ntlm max_challenge_reuses 0</code>
Example	<code>auth_param ntlm max_challenge_reuses 5</code>
Related	<code>auth_param ntlm program</code> , <code>auth_param ntlm children</code> , <code>auth_param ntlm max_challenge_lifetime</code>

auth_param ntlm max_challenge_lifetime

This directive also controls whether the external NTML helper processes can reuse their challenge tokens. It specifies the maximum amount of time a single challenge can be used.

Syntax	<code>auth_param ntlm max_challenge_lifetime <i>time-specification</i></code>
Default	<code>auth_param ntlm max_challenge_lifetime 1 minute</code>
Example	<code>auth_param ntlm max_challenge_lifetime 2 minutes</code>
Related	<code>auth_param ntlm program</code> , <code>auth_param ntlm children</code> , <code>auth_param ntlm max_challenge_reuses</code>

authenticate_ttl

Squid maintains a cache of proxy authentication usernames and credentials. Squid periodically removes unused entries to keep memory usage down. This directive specifies how

authenticate_cache_garbage_interval

long Squid keeps entries in the proxy authentication username cache. A user's TTL is extended each time Squid receives a request from that user.



This directive doesn't determine how long credentials remain valid. It only affects whether or not an entry is removed from the username cache. Squid may decide to revalidate the credentials of a user that is in the cache. Each authentication scheme has its own way of determining when to revalidate credentials with the external helper.

Syntax	<code>authenticate_ttl <i>time-specification</i></code>
Default	<code>authenticate_ttl 1 hour</code>
Example	<code>authenticate_ttl 30 minutes</code>
Related	<code>authenticate_cache_garbage_interval</code> , <code>auth_param</code>

authenticate_cache_garbage_interval

This directive specifies how often Squid executes the function to clean up the proxy authentication username cache. During this process, usernames that have been inactive for some amount of time (defined by `authenticate_ttl`) are purged.

Syntax	<code>authenticate_cache_garbage_interval <i>time-specification</i></code>
Default	<code>authenticate_cache_garbage_interval 1 hour</code>
Example	<code>authenticate_cache_garbage_interval 8 hours</code>
Related	<code>authenticate_ttl</code> , <code>auth_param</code>

authenticate_ip_ttl

This directive causes Squid to deny requests if the same proxy authentication username comes from more than one IP address within a given amount of time. It's designed to discourage users from sharing their username and password with others. When Squid detects the same username from multiple IP addresses, it forces the user to reauthenticate by denying the request.

This feature is disabled by default (0 seconds). If your users normally have the same IP address (e.g., static addressing or DHCP with long leases), you can set `authenticate_ip_ttl` to a large value such as 1 hour. However, if your users are on dial-up connections, they may be more likely to change IP addresses within a short period of time. To make their lives easier, use a small `authenticate_ip_ttl` value, such as 1 minute.

Syntax	<code>authenticate_ip_ttl <i>time-specification</i></code>
Default	<code>authenticate_ip_ttl 0 seconds</code>
Example	<code>authenticate_ip_ttl 1 minute</code>
Related	<code>auth_param</code>

external_acl_type

This directive defines new ACL types implemented as external programs. See Chapter 6.

Syntax	<code>external_acl_type type-name [options] format helper-command</code>
Default	No default
Example	<code>external_acl_type MyAcltype %LOGIN /usr/local/squid/libexec/my-acl-prog.pl</code>
Related	<code>acl</code> , <code>http_access</code>

wais_relay_host

The Wide Area Information Service (WAIS) is an obsolete protocol that predates the Web. This directive is largely historical. Its purpose is to make Squid forward all WAIS requests to another proxy, perhaps a dedicated WAIS gateway. You can accomplish the same effect with ACLs and `cache_peer_access`.

Syntax	<code>wais_relay_host hostname</code>
Default	No default
Example	<code>wais_relay_host some.host.name</code>
Related	<code>wais_relay_port</code>

wais_relay_port

If, for some reason, you use `wais_relay_host`, you must set the WAIS relay port number with this directive. Arguably you should be able to specify both with a single directive. However, they were split some time ago to simplify Squid's parsing code.

Syntax	<code>wais_relay_port port-number</code>
Default	No default
Example	<code>wais_relay_port 8001</code>
Related	<code>wais_relay_host</code>

request_header_max_size

This directive places an upper limit on the size of headers in an HTTP request. When Squid receives an HTTP request with headers that exceed this value, it returns a 413 (Request Entity Too Large) error response. In most cases, request headers are smaller than 512 bytes. This directive exists to catch certain abnormal conditions, such as persistent connection bugs, buffer overflow attempts, and denial-of-service attacks.

Syntax	<code>request_header_max_size size-specification</code>
Default	<code>request_header_max_size 10 KB</code>
Example	<code>request_header_max_size 35 KB</code>
Related	<code>request_body_max_size</code> , <code>reply_body_max_size</code>

`request_body_max_size`

request_body_max_size

This directive, if nonzero, places an upper limit on the size of a client's HTTP request body. Most requests (i.e., GET requests) don't have request bodies. This directive applies to PUT and POST requests. A request that exceeds this limit generates a 413 (Request Entity Too Large) error response.

Syntax `request_body_max_size size-specification`
Default No limit
Example `request_body_max_size 100 KB`
Related `request_header_max_size, reply_body_max_size`

refresh_pattern

This directive provides a way to customize Squid's algorithm for validating cached responses. HTTP has a relatively complex procedure for determining whether or not a cached response is fresh or stale. In some cases, origin servers provide an explicit expiration time. However, the majority of responses don't have this information. For these, Squid applies some heuristics to the response. See Chapter 7 for more information.

Syntax `refresh_pattern regex mtime percent maxtime [options]`
Default `refresh_pattern . 0 20% 4320`
Example `refresh_pattern \.jpg$ 0 75 7200`

quick_abort_min

This directive controls Squid's behavior for requests aborted by the user. In some cases, Squid continues reading data from the origin server so that future requests may be satisfied as cache hits. If Squid knows that the transfer (between itself and the origin server) has no more than this many bytes remaining, it continues receiving the object. Otherwise, Squid checks the `quick_abort_max` setting next.

Syntax `quick_abort_min size-specification`
Default `quick_abort_min 16 KB`
Example `quick_abort_min 50 KB`
Related `quick_abort_max, quick_abort_pct`

quick_abort_max

After checking `quick_abort_min`, Squid checks the value of this directive. If an aborted request has more than this many bytes remaining in the transfer, Squid terminates the connection to the origin server. Otherwise, it checks the `quick_abort_pct` setting.

Syntax `quick_abort_max size-specification`
Default `quick_abort_max 16 KB`

Example `quick_abort_max 1 MB`
 Related `quick_abort_min, quick_abort_pct`

quick_abort_pct

Squid checks this value last, after checking `quick_abort_max`, for a transfer aborted by the user. If Squid has already received at least this percentage of the response, it continues reading the data from the origin server so the entire response is cached.

Syntax `quick_abort_pct percentage`
 Default `quick_abort_pct 95%`
 Example `quick_abort_pct 75%`
 Related `quick_abort_min, quick_abort_max`

negative_ttl

Squid takes the liberty of caching certain error responses, such as “connection refused” and 404 (Not Found) messages. In most cases, repeating the request again immediately is likely to result in the same error. This directive specifies how long Squid caches these errors. Cache hits for negatively cached responses are logged with `TCP_NEGATIVE_HIT` in `access.log`.

Syntax `negative_ttl time-specification`
 Default `negative_ttl 5 min`
 Example `negative_ttl 1 minute`
 Related `refresh_pattern`

positive_dns_ttl

Each and every DNS resource record carries an explicit TTL that specifies how long the information may be cached. In most situations, Squid has access to the TTL values and doesn't store DNS answers longer than allowed. This is certainly true when you use Squid's internal DNS implementation, which is enabled by default.

However, if you elect to use the (external) `dnsserver` processes, Squid may not receive TTL values for DNS answers. In this case, successful DNS answers are cached for the amount of time specified by this directive.

Syntax `positive_dns_ttl time-specification`
 Default `positive_dns_ttl 6 hours`
 Example `positive_dns_ttl 1 hour`
 Related `negative_dns_ttl`

`negative_dns_ttl`

negative_dns_ttl

This is similar to *positive_dns_ttl*, except that it applies only to failed DNS queries. That is, when Squid receives an error for a DNS lookup, it negatively caches the error for this amount of time. It doesn't retry the query until the negative TTL expires. This applies to both internal and external DNS implementation choices.

Syntax	<code>negative_dns_ttl <i>time-specification</i></code>
Default	<code>negative_dns_ttl 5 minutes</code>
Example	<code>negative_dns_ttl 1 minute</code>
Related	<i>positive_dns_ttl</i>

range_offset_limit

A *range request* comes from a client that wants only some subset of an HTTP response. They are sometimes used to resume a failed transfer of a large file. Squid isn't yet able to cache partial responses and thus must make a decision when forwarding a range request: either remove the Range header or leave it in.

If Squid leaves the Range header in, the origin server sends only the subset that the client wants, and the client receives the response immediately. However, this partial response isn't cached.

On the other hand, if Squid removes the header before forwarding, it receives the entire response, which may be cached. Squid is then responsible for ensuring that the client receives only the subset it needs. The origin server may send a lot of data the client doesn't want. Depending on the speed of your connection, the client may be forced to wait a long time until its range is available.

If the beginning of the requested range is larger than the *range_offset_limit* value, Squid forwards the Range header and doesn't cache the response. Setting *range_offset_limit* to 0 causes Squid to always forward the Range header (the default). Setting it to -1 causes Squid to never forward the header.

Syntax	<code>range_offset_limit <i>size-specification</i></code>
Default	<code>range_offset_limit 0 KB</code>
Example	<code>range_offset_limit 100 KB</code>

connect_timeout

This directive tells Squid how long to wait when trying to connect to an origin server. After this amount of time, Squid gives up and tries another location or returns an error to the user. Your operating system's TCP implementation has its own connection timeout. If the TCP timeout occurs before *connect_timeout*, Squid creates a new TCP connection and tries again.

Syntax	<code>connect_timeout <i>time-specification</i></code>
Default	<code>connect_timeout 2 minutes</code>

Example `connect_timeout 30 seconds`
 Related `peer_connect_timeout, read_timeout, write_timeout, request_timeout, pconn_timeout, minimum_retry_timeout`

peer_connect_timeout

This is similar to `connect_timeout`, except that it applies to connections to your neighbors. Most likely, you'll want a smaller timeout for neighbor connections because they should be closer to you than most origin servers. If a neighbor is down, you want the connection to time out quickly so that you can try another source. Note that you can also specify individual neighbor timeouts with the `connect-timeout` option of the `cache_peer` directive.

Syntax `peer_connect_timeout time-specification`
 Default `peer_connect_timeout 30 seconds`
 Example `peer_connect_timeout 15 seconds`
 Related `connect_timeout`

read_timeout

This timeout applies to server connections (between Squid and origin servers or neighbor caches). If Squid doesn't receive any data for this amount of time, it closes the connection. If the user hasn't yet received any part of the response, Squid generates a "read timeout" error message.

Syntax `read_timeout time-specification`
 Default `read_timeout 15 minutes`
 Example `read_timeout 1 hour`
 Related `connect_timeout, write_timeout, request_timeout, client_lifetime`

request_timeout

This timeout applies to client connections. Once a client establishes a connection, Squid waits this long to receive the client's HTTP request. If the client fails to send a complete request, Squid simply closes the connection without sending any error message.

Syntax `request_timeout time-specification`
 Default `request_timeout 5 minutes`
 Example `request_timeout 30 seconds`
 Related `read_timeout, connect_timeout`

persistent_request_timeout

This timeout is similar to `request_timeout`, except that it applies only to idle, persistent connections.

`client_lifetime`

Syntax	<code>persistent_request_timeout</code> <i>time-specification</i>
Default	<code>persistent_request_timeout</code> 1 minute
Example	<code>persistent_request_timeout</code> 30 seconds
Related	<code>request_timeout</code>

`client_lifetime`

This timeout specifies the maximum amount of time for a client connection. In most cases, client connections should never last longer than a few hours. Long-lived client connections may be the result of a network outage, user-agent bugs, or mischievous activity.

Syntax	<code>client_lifetime</code> <i>time-specification</i>
Default	<code>client_lifetime</code> 1 day
Example	<code>client_lifetime</code> 3 hours
Related	<code>read_timeout</code>

`half_closed_clients`

TCP allows applications to close connections in one direction. That is, a client may close its connection for writing but keep it open for reading. These half-closed connections are confusing because Squid can't easily tell the difference between a client that intentionally closed half the connection and a client that simply aborted the entire connection. The only way Squid knows for sure is when its attempt to write some data returns an error. Most user-agents don't use the TCP half-close, but some may.

When the `half_closed_clients` directive is enabled (the default), Squid keeps these connections open until a write error (or some other error) occurs. When disabled, Squid fully closes the connection. Thus, if you disable this directive and have clients that use the TCP half-close, they can't receive any data from Squid.

Syntax	<code>half_closed_clients</code> on off
Default	<code>half_closed_clients</code> on
Example	<code>half_closed_clients</code> off
Related	<code>client_lifetime</code> , <code>read_timeout</code>

`pconn_timeout`

This timeout applies to idle server persistent connections (i.e., connections between Squid and origin servers or neighbors). If the idle connection isn't reused within this amount of time, Squid closes it to conserve resources.

Syntax	<code>pconn_timeout</code> <i>time-specification</i>
Default	<code>pconn_timeout</code> 2 minutes
Example	<code>pconn_timeout</code> 45 seconds
Related	<code>persistent_request_timeout</code> , <code>connect_timeout</code> , <code>read_timeout</code>

ident_timeout

This timeout applies to ident (RFC 1413) requests made to client hosts. Squid makes ident lookups for one of two reasons: to satisfy an ACL check or for logging in *access.log*. In the ACL case, Squid blocks the request until the ident lookup returns, or this timeout occurs. When only logging, Squid doesn't block on the ident lookup.

Syntax	<code>ident_timeout <i>time-specification</i></code>
Default	<code>ident_timeout 10 seconds</code>
Example	<code>ident_timeout 1 minute</code>
Related	<i>ident_lookup_access, acl ident</i>

shutdown_lifetime

When you shut down the Squid process, some user requests will still be active. This directive specifies how long to wait until all client requests are complete. Squid finally exits when all client connections have been closed or when this timeout occurs.

Syntax	<code>shutdown_lifetime <i>time-specification</i></code>
Default	<code>shutdown_lifetime 30 seconds</code>
Example	<code>shutdown_lifetime 60 seconds</code>

acl

The *acl* directive defines an access control element, such as a client IP address, origin server hostname, or server port number. The syntax depends on the particular ACL type you wish to define. See Chapter 6 for the full-blown explanation.

Syntax	<code>acl <i>name type data...</i></code>
Default	No default
Example	<code>acl MyClients src 172.16.1.0/24</code>
Related	<i>http_access, icp_access, miss_access, no_cache, redirector_access, http_reply_access, ident_lookup_access, always_direct, never_direct, snmp_access, broken_posts</i>

http_access

The *http_access* directive is one of the most important aspects of your configuration. It determines whether or not Squid allows or denies a client's request. If you don't get your access-control rules just right, savvy Internet users can abuse your resources (e.g., bandwidth, disk storage, address space). Some people find the access control rule syntax confusing. Be sure to read Chapter 6 closely.

Syntax	<code>http_access allow deny [!]<i>ACLname ...</i></code>
Default	<code>http_access deny all</code>

`http_reply_access`

Example `http_access allow MyClients`
Related `acl, http_reply_access, miss_access, icp_access`

`http_reply_access`

The `http_reply_access` rules are similar to `http_access`, except that they are checked after Squid receives the HTTP response headers for a cache miss. You might want to use this access list to deny requests based on some characteristic of the response, such as the content type.

Syntax `http_reply_access allow|deny [!]ACLname ...`
Default `http_reply_access allow all`
Example `http_reply_access deny MP3Files`
Related `acl, http_access`

`icp_access`

This access list applies to ICP queries. If a particular ICP query is denied by the `icp_access` rules, Squid returns an ICP_DENIED message to the neighbor.

Syntax `icp_access allow|deny [!]ACLname ...`
Default `icp_access deny all`
Example `icp_access allow Neighbor1`
Related `acl, http_access`

`miss_access`

The `miss_access` rules are similar to `http_access`. However, they are applied to cache misses only. This allows you to enforce sibling relationships with your neighbor caches. See Chapter 6.

Syntax `miss_access allow|deny [!]ACLname ...`
Default `miss_access allow all`
Example `miss_access deny MySiblings`
Related `acl, http_access`

`cache_peer_access`

The `cache_peer_access` rules determine which requests Squid will forward to a particular neighbor. If a particular request is denied by a `cache_peer_access` list, Squid doesn't forward the request to that neighbor. See Chapter 10.

Syntax `cache_peer_access peername allow|deny [!]ACLname ...`
Default No default

Example `cache_peer_access neighbor.host.name allow SomeOriginDomains`
 Related `acl, cache_peer, cache_peer_domain, http_access`

ident_lookup_access

The *ident_lookup_access* rules determine whether or not Squid performs an RFC 1413 username lookup for a client's TCP connection. These rules are checked before Squid reads any part of the HTTP request. Thus, only TCP/IP-based ACL elements (e.g., client address, port number) should be used in these rules.

Syntax `ident_lookup_access allow|deny [!]ACLname ...`
 Default `ident_lookup_access deny all`
 Example `ident_lookup_access allow TheseClients`
 Related `acl, ident_timeout`

tcp_outgoing_tos

This directive allows you to set specific DSCP (differential services code point) values for outgoing TCP connections—those made to origin servers and neighbors. The differential services protocol is quite complex. Simply using the example in the following table will get you nowhere. Make sure that you understand what you are doing before using this directive. See RFCs 2474, 2475, and 3140 for additional information on differential services.

Syntax `tcp_outgoing_tos byte-value [!]ACLname ...`
 Default `No default`
 Example `acl NormalService src 10.0.0.0/255.255.255.0`
`acl BetterService src 10.0.1.0/255.255.255.0`
`tcp_outgoing_tos 0x00 NormalService`
`tcp_outgoing_tos 0x20 BetterService`

tcp_outgoing_address

You can use this access list-based directive to bind outgoing TCP connections to specific local addresses. It might be useful if your system has multiple network interfaces, and you want to make sure all of Squid's traffic leaves through one and not the other. Another possibility is that you have two or more interfaces with different costs or characteristics. You may want to send privileged user's traffic through the expensive, uncongested link, while other users go out the cheap, low-quality connection. Don't use this directive if your system has only one network interface.

If you have an *tcp_outgoing_address* rule with no ACLs, that address is used for requests that don't match any of the other rules.

Syntax `tcp_outgoing_address ipaddr [!]ACLname ...`
 Default `No default`

`reply_body_max_size`

Example `acl SomeUsers src 10.0.0.0/24`
 `acl OtherUsers src 10.0.1.0/24`
 `tcp_outgoing_address 172.16.0.1 SomeUsers`
 `tcp_outgoing_address 192.168.0.1 OtherUsers`
 `tcp_outgoing_address 172.16.5.1`
Related `udp_incoming_address, udp_outgoing_address`

reply_body_max_size

This directive allows you to limit the size of HTTP reply bodies based on ACL elements. When a request matches one of the `reply_body_max_size` rules, Squid places a limit on the size of the HTTP response. A value of 0 indicates no limit. Squid checks the reply size first when all HTTP headers have been received. If the headers contain a Content-Length value that exceeds the specified limit, the user receives a message that states “the request or reply is too large.” If the content length is unavailable, Squid continues checking the limit as data comes in from the server. If the reply size exceeds the limit, Squid closes the client’s connection, which causes the client to receive a partial reply.

Downstream caches often can’t detect partial replies. Because the headers lack a content length value, the downstream cache (or user-agent) doesn’t know that additional data is missing. Thus, you shouldn’t use `reply_body_max_size` if you have child or sibling caches.

The code that checks the `reply_body_max_size` list ignores deny rules. In other words, it is pointless to include deny rules in this list.

Make sure that the maximum reply size is large enough for a Squid error message (typically 1K–2K bytes). An error message that is larger than the maximum reply body size causes Squid to crash.

Syntax `reply_body_max_size bytes allow [!]ACLname ...`
Default `reply_body_max_size 0 allow all`
Example `acl WorkingHours time 08:00-17:00`
 `reply_body_max_size 10485760 allow WorkingHours`
Related `maximum_object_size, request_body_max_size, request_header_max_size`

cache_mgr

This email address is printed in error messages generated by Squid. Set this as an address to which your users should send support messages and problem reports. This address also receives a notification message if Squid dies unexpectedly.

Syntax `cache_mgr email@address`
Default `cache_mgr webmaster`
Example `cache_mgr support@example.com`

cache_effective_user

In the interest of security, Squid doesn't allow itself to run as *root*. If you start the process as *root*, Squid changes its *effective userid* to a nonprivileged user. This user ID must have write permission to the cache directories and log file directory.

You need to set this directive only if you're starting Squid as *root*. If you start Squid as a non-*root* user, this directive is ignored.

Syntax	cache_effective_user <i>username</i>
Default	cache_effective_user nobody
Example	cache_effective_user squid
Related	cache_effective_group

cache_effective_group

If you start Squid as *root*, it changes the process' user ID to the username specified by *cache_effective_user*. By default, Squid sets the process' group ID to the group associated with the *cache_effective_user*. You can set the *cache_effective_group* directive if you want Squid to use some other group ID.

You only need to set this directive if you're starting Squid as *root*. If you start Squid as a non-*root* user, this directive is ignored.

Syntax	cache_effective_group <i>groupname</i>
Default	No default
Example	cache_effective_group squid
Related	cache_effective_user

visible_hostname

Use this directive when Squid can't determine the fully qualified domain name on its own or if you want to present a special, external name to the world. Squid uses this name in error messages, FTP directory listings, X-Cache header values, cache announcements, and for internal URLs.

Squid also puts the visible hostname into HTTP Via headers, unless you also define the *unique_hostname* directive. Note that you must use *unique_hostname* if you have a cluster of caches that have the same visible hostname.

Syntax	visible_hostname <i>hostname</i>
Default	No default
Example	visible_hostname my.host.name
Related	unique_hostname, hostname_aliases, announce_period

unique_hostname

unique_hostname

If you have a cluster of caches talking to each other and sharing a single *visible_hostname* value, you must use this directive to give each a unique name. Squid uses the unique name in HTTP Via headers to detect forwarding loops (see Chapter 10).

Syntax	<code>unique_hostname hostname</code>
Default	No default
Example	<code>unique_hostname cache1.host.name</code>
Related	<i>visible_hostname, hostname_aliases</i>

hostname_aliases

You may find yourself in a situation where more than one hostname resolves to Squid's IP address. For example, both *sv.us.ircache.net* and *sv.cache.nlanr.net* resolve to 192.203.230.19. If you have neighbors, they may send requests for certain Squid-specific internal URLs, as in the case of Cache Digests. These URLs might contain either hostname. You must use this directive to tell Squid that it is known by names other than its *visible_hostname*.

Syntax	<code>hostname_aliases hostname ...</code>
Default	No default
Example	<code>hostname_aliases this.host.name that.host.name</code>
Related	<i>visible_hostname, unique_hostname</i>

announce_period

Squid's announcement feature allows Squid administrators to find nearby caches that might be interested in joining a cache hierarchy. When you enable this directive, Squid periodically sends a small announcement message to a central server. By default, the announcement message contains five fields:

- The IP address and hostname that sent the announcement
- The Squid version
- The hostname Squid uses internally—either your hostname if Squid can figure it out or the value of the *visible_hostname* directive
- The value of the *cache_mgr* directive
- The date and time of the announcement

Setting *announce_period* to 0 disables the announcement feature.

Syntax	<code>announce_period time-specification</code>
Default	<code>announce_period 0</code>
Example	<code>announce_period 4 hours</code>
Related	<i>announce_host, announce_file, announce_port</i>

announce_host

This is the host setup to receive Squid's announcement messages. The default value, *tracker.ircache.net* is the only server I know about. You can search the *tracker.ircache.net* database by visiting <http://www.ircache.net/Tracker/>.

Note that if you set *cache_mgr*, your email address may be available to random people. On more than one occasion I have seen commercial caching vendors target Squid users by collecting their email addresses from this database.

Syntax	<code>announce_host hostname</code>
Default	<code>announce_host tracker.ircache.net</code>
Example	<code>announce_host some.host.name</code>
Related	<i>announce_period, announce_file, announce_port</i>

announce_file

You can customize your cache announcement message by setting this directive to a file containing additional information. For example, you can include information about your upstream service provider, telephone number, other caches that you peer with, etc.

Announcement messages are sent via UDP, so this file shouldn't be too large. Some systems can't send or receive UDP messages larger than 9 KB. Furthermore, larger messages are more likely to be dropped before reaching their destination.

Syntax	<code>announce_file pathname</code>
Default	No default
Example	<code>announce_file /usr/local/squid/etc/announce.txt</code>
Related	<i>announce_period, announce_host, announce_port</i>

announce_port

This is the UDP port number to which the announcement messages are sent.

Syntax	<code>announce_port port-number</code>
Default	<code>announce_port 3131</code>
Example	<code>announce_port 1234</code>
Related	<i>announce_period, announce_host, announce_file</i>

httpd_accel_host

This directive enables HTTP server acceleration (see Chapter 15) and HTTP interception (see Chapter 9). When Squid is configured for server acceleration, this directive specifies the hostname or IP address of the backend server. When used in an interception configuration, you should probably use the keyword *virtual* here.

httpd_accel_port

When this directive is set, Squid disables ICP and rejects proxy-HTTP requests unless you also enable *httpd_accel_with_proxy*.

Syntax	<code>httpd_accel_host hostname virtual</code>
Default	No default
Example	<code>httpd_accel_host virtual</code>
Related	<i>httpd_accel_port, httpd_accel_single_host, httpd_accel_with_proxy, httpd_accel_uses_host_header, emulate_httpd_log</i>

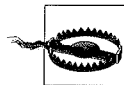
httpd_accel_port

This is the TCP port number to which accelerated/intercepted requests are sent. In most cases, you should leave it set to port 80. If you are accelerating/intercepting more than one port, set it to 0. That is similar to the virtual setting for *httpd_accel_host*.

Syntax	<code>httpd_accel_port port-number</code>
Default	<code>httpd_accel_port 80</code>
Example	<code>httpd_accel_port 0</code>
Related	<i>httpd_accel_host, httpd_accel_single_host, httpd_accel_with_proxy, httpd_accel_uses_host_header</i>

httpd_accel_single_host

When enabled, this directive makes Squid forward all accelerated/intercepted requests to the *httpd_accel_host* address. See Chapter 15.



If you enable this directive and *httpd_accel_with_proxy*, Squid may become susceptible to cache poisoning. Please read Chapter 15 thoroughly before running such a configuration.

Syntax	<code>httpd_accel_single_host on off</code>
Default	<code>httpd_accel_single_host off</code>
Example	<code>httpd_accel_single_host on</code>
Related	<i>httpd_accel_host, httpd_accel_port, httpd_accel_with_proxy, httpd_accel_uses_host_header</i>

httpd_accel_with_proxy

Enabling HTTP acceleration/interception normally disables proxy-HTTP caching. That is, Squid refuses to handle proxy requests (with a full URI) when in HTTP server accelerator mode. Although I don't recommend it, you can force Squid to accept both types of requests by enabling this directive.

Syntax	<code>httpd_accel_with_proxy on off</code>
Default	<code>httpd_accel_with_proxy off</code>
Example	<code>httpd_accel_with_proxy on</code>
Related	<i>httpd_accel_host, httpd_accel_port, httpd_accel_single_host, httpd_accel_uses_host_header</i>

httpd_accel_uses_host_header

When this directive is enabled, Squid uses a request's Host header when rewriting accelerated/intercepted requests. When disabled, Squid uses either the origin server's IP address or the *httpd_accel_host* value.

You should probably enable *httpd_accel_uses_host_header* when running Squid as an HTTP-intercepting proxy. If Squid is a surrogate (accelerator), you only need to enable this directive if the backend server is configured for virtual hosting.

Syntax	<code>httpd_accel_uses_host_header on off</code>
Default	<code>httpd_accel_uses_host_header off</code>
Example	<code>httpd_accel_uses_host_header on</code>
Related	<i>httpd_accel_host, httpd_accel_port, httpd_accel_single_host, httpd_accel_with_proxy</i>

dns_testnames

Squid uses these hostnames to test the DNS before starting. If Squid can't resolve any of these names, it prints an error and refuses to run. If the default list doesn't seem to work on your network, try listing some local hostnames instead.

Syntax	<code>dns_testnames hostname ...</code>
Default	<code>dns_testnames netscape.com internic.net nlanr.net microsoft.com</code>
Example	<code>dns_testnames yahoo.com example.com squid-cache.org</code>

logfile_rotate

You must periodically signal Squid to rotate its log files. If you don't, they will increase in size and eventually fill up the disk partition. This directive specifies how many old copies of each log file to keep around. See Chapter 13 for more information.

Syntax	<code>logfile_rotate N</code>
Default	<code>logfile_rotate 10</code>
Example	<code>logfile_rotate 5</code>
Related	<i>cache_access_log, cache_log, cache_store_log, cache_swap_log, useragent_log, referer_log</i>

append_domain

This directive helps Squid turn single-component hostnames into fully qualified domain names. For example, *http://www/* becomes *www.example.com/*. This is especially important if you are participating in a cache hierarchy.

Syntax	<code>append_domain .domain.name</code>
Default	No default
Example	<code>append_domain .example.com</code>
Related	<i>dns_defnames, hosts_file</i>

tcp_recv_bufsize

tcp_recv_bufsize

If you use this directive, Squid sets the receive buffer size for each TCP socket that it creates. This value refers to the amount of data that the TCP/IP stack will buffer on behalf of the application. You can see how much data is being buffered at any given time by looking at the Recv-Q column of `netstat -n` output. Larger TCP buffers lead to increased memory usage and better performance.

In general, you shouldn't need to use this directive. Most operating systems in use today have default TCP buffer sizes greater than 32 KB. Empirical evidence suggests that fewer than 5% of typical web objects are larger than 32 KB.

When `tcp_recv_bufsize` is set to 0, Squid doesn't change the TCP buffer size from its default value.

Syntax	<code>tcp_recv_bufsize <i>size-specification</i></code>
Default	<code>tcp_recv_bufsize 0</code>
Example	<code>tcp_recv_bufsize 8 kb</code>

err_html_text

This directive is one way to customize Squid's error messages. The error message files contain printf-like tokens. Squid dynamically replaces the tokens with appropriate values for each error. If Squid encounters the token `%L`, it inserts the contents of this directive. Note that none of the default error messages contain a `%L`. Thus, to use this feature, you must modify the default error files.

Syntax	<code>err_html_text <i>character string</i></code>
Default	No default
Example	<code>err_html_text Call 555-1234 to report problems with Squid.</code>
Related	<i>error_directory</i>

deny_info

This directive allows you to show specific error messages to users when a request matches certain ACL elements. This is more informative than sending a generic "access denied" error message, as happens by default.

When Squid checks its access control rules to see whether or not a particular request is allowed or denied, it remembers the ACL element that causes the search to terminate. You can use these ACL element names in a `deny_info` line to correlate error messages with a specific request characteristic. Consider, for example, this configuration:

```
acl Unsafe_Ports 7 9 19 22 23 25 53 109 110 119
...
http_access deny Unsafe_Ports
...
deny_info ERR_PORT_IS_UNSAFE Unsafe_Ports
```

When a user makes a request to an origin server on one of the ports listed in the *Unsafe_Ports* ACL, Squid denies the request. Furthermore, Squid generates an error message from the *ERR_PORT_IS_UNSAFE* file, found in the *error_directory* directory.

Alternatively, you can specify a URI instead of an error message template. In this case, Squid sends an HTTP 302 (Moved Temporarily) redirect to the given URI.

Finally, if you specify *TCP_RESET* as the error message template, Squid closes the client's connection in a way that generates a TCP reset.

Syntax	<code>deny_info error-page-name URI acl-name</code>
Default	No default
Example	<code>deny_info ERR_PORT_IS_UNSAFE Unsafe_Ports</code>
Related	<i>error_directory, acl</i>

memory_pools

Squid's memory pools are an attempt to optimize the way Squid allocates and frees memory. Certain data structures inside Squid are *pooled*. This means that rather than freeing unused memory, Squid holds onto it for future use. It also means that a particular chunk of memory is normally used for the same type of data structure. Memory pools may improve Squid's performance by avoiding frequent calls to `malloc()` and `free()`. The downside, however, is that the overall memory usage may be higher. If memory is a precious resource on your system, you might want to disable memory pools.

Syntax	<code>memory_pools on off</code>
Default	<code>memory_pools on</code>
Example	<code>memory_pools off</code>
Related	<i>cache_mem, memory_pools_limit</i>

memory_pools_limit

This directive specifies an upper limit on the amount of unused memory to hold onto. If the total size of all unused, pooled memory exceeds this value, Squid begins returning unused memory to the *malloc* library by calling `free()`.

If set to 0 (the default), Squid doesn't place any limit on the amount of unused memory to keep in the pools.

Syntax	<code>memory_pools_limit size-specification</code>
Default	<code>memory_pools_limit 0</code>
Example	<code>memory_pools_limit 100 MB</code>
Related	<i>memory_pools</i>

forwarded_for

forwarded_for

Squid appends an item to the X-Forwarded-For header in requests sent to origin servers and neighbors. When this directive is enabled, Squid places the client's IP address there. When it is disabled, Squid prints the word *unknown* instead. Thus, disabling *forwarded_for* increases your user's privacy.

Syntax	forwarded_for on off
Default	forwarded_for on
Example	forwarded_for off

log_icp_queries

By default, ICP queries appear in Squid's *access.log*. If Squid receives a large amount of ICP queries from neighbors, your *access.log* file may become too large to effectively manage. If you disable this directive, ICP queries are never logged.

Syntax	log_icp_queries on off
Default	log_icp_queries on
Example	log_icp_queries off
Related	<i>access_log, icp_port</i>

icp_hit_stale

Squid normally returns ICP_MISS for queries to stale objects. This causes an annoying problem described in Chapter 10. If you enable this directive, Squid returns ICP_HIT messages instead.

Syntax	icp_hit_stale on off
Default	icp_hit_stale off
Example	icp_hit_stale on
Related	<i>cache_peer, miss_access</i>

minimum_direct_hops

If you're using *netdb* (see Chapter 10), and a cache hierarchy, Squid forwards requests directly to origin servers that are within this many router hops. Such requests are marked with CLOSEST_DIRECT in *access.log*.

Syntax	minimum_direct_hops <i>N</i>
Default	minimum_direct_hops 4
Example	minimum_direct_hops 6
Related	<i>minimum_direct_rtt, always_direct</i>

minimum_direct_rtt

Similar to *minimum_direct_hops*. If Squid is within *minimum_direct_rtt* milliseconds (as measured by ICMP pings) to the origin server, the request is sent there directly. These requests are marked with CLOSEST_DIRECT in *access.log*.

Syntax *minimum_direct_rtt milliseconds*
 Default *minimum_direct_rtt 400*
 Example *minimum_direct_rtt 100*
 Related *minimum_direct_hops, always_direct*

cachemgr_passwd

This directive allows you to protect cache manager pages with a password. Unfortunately, this is an extremely weak authorization scheme, because passwords are sent as cleartext in the cache manager HTTP request. See Chapter 14 for a discussion of cache manager passwords.

Syntax *cachemgr_passwd password cachemgr-page ...*
 Default No default
 Example *cachemgr_passwd SekrIt config objects vm_objects*
 Related *http_access*

store_avg_object_size

Squid uses this value as a hint for estimating the size of certain data structures. In particular, Squid calculates an estimate for the total number of objects in the cache, based on this value and the sum of all *cache_dir* sizes. This estimate is, in turn, used to calculate the number of hash buckets for the primary index to cached objects. Additionally, it can estimate the cache digest size, if that feature is enabled.

In most cases the default should be sufficient. You can find the actual value for your cache by querying the cache manager. Look for “Mean Object Size” on the *info* page (see Chapter 14).

Syntax *store_avg_object_size size-specification*
 Default *store_avg_object_size 13 KB*
 Example *store_avg_object_size 10 KB*
 Related *cache_dir, digest_bits_per_entry, store_objects_per_bucket*

store_objects_per_bucket

This directive allows you to tune the tradeoff between increased memory usage and longer searching times. Squid calculates the number of hash table buckets, depending on this directive, the average object size, and the total cache size. Squid’s goal is to have this many objects in each bucket of the hash table.

client_db

A larger value here leads to reduced memory usage but longer search times. Conversely, a smaller value leads to faster search times, at the expense of increased memory usage.

Syntax	<code>store_objects_per_bucket N</code>
Default	<code>store_objects_per_bucket 20</code>
Example	<code>store_objects_per_bucket 15</code>
Related	<code>store_avg_object_size</code>

client_db

Squid keeps a number of statistics for each cache client (IP address). You can view them by visiting the cache manager *client_list* page. The ClientInfo data structure is about 240 bytes on 32-bit systems and 300 bytes on 64-bit systems. If you have thousands of clients, this database can consume a significant amount of memory. You can disable this directive and free up that memory for other uses.

Syntax	<code>client_db on off</code>
Default	<code>client_db on</code>
Example	<code>client_db off</code>

netdb_low

The *netdb* database contains round-trip time and hop-count measurements derived from ICMP pings. This directive specifies the lower limit for the *netdb* replacement policy. In other words, when Squid is removing *netdb* entries, it stops when the total number reaches *netdb_low*.

Syntax	<code>netdb_low N</code>
Default	<code>netdb_low 900</code>
Example	<code>netdb_low 9900</code>
Related	<code>netdb_high</code> , <code>query_icmp</code>

netdb_high

The *netdb* database contains round-trip time and hop-count measurements derived from ICMP pings. This directive specifies an upper limit on the number entries in the database. When Squid finds more than *netdb_high* entries, it removes least-recently used networks until the size reaches *netdb_low*.

Syntax	<code>netdb_high N</code>
Default	<code>netdb_high 1000</code>
Example	<code>netdb_high 10000</code>
Related	<code>netdb_low</code> , <code>query_icmp</code>