
netdb_ping_period

This directive specifies how long Squid must wait between sending consecutive ICMP pings to the same /24 network. The interval is relatively long so that Squid's ICMP traffic doesn't upset server administrators.

Syntax `netdb_ping_period time-specification`
 Default `netdb_ping_period 5 min`
 Example `netdb_ping_period 3 min`
 Related `pinger_program, query_icmp`

query_icmp

Enabling this directive instructs Squid to ask its neighbors for their ICMP measurements, which are included in ICP/HTCP replies. This, essentially, populates your *netdb* database with your neighbors' ICMP measurements. The bulk "netdb exchange" is another way to receive those measurements (see Chapter 10).

Squid uses the neighbors' *netdb* measurements when making forwarding decisions. If one of the parents is closer to the origin server, Squid forwards the request there and marks it with CLOSEST_PARENT_MISS in *access.log*.

Syntax `query_icmp on|off`
 Default `query_icmp off`
 Example `query_icmp on`
 Related `pinger_program, netdb_ping_period`

test_reachability

When you enable this directive, Squid looks at its *netdb* database while processing ICP queries. If Squid normally returns ICP_MISS, but the origin server isn't in the database or doesn't respond to ICMP pings, it returns ICP_MISS_NOFETCH instead. The ICP_MISS_NOFETCH reply signals the neighbor cache that Squid might not be able to communicate with the origin server.

Syntax `test_reachability on|off`
 Default `test_reachability off`
 Example `test_reachability on`
 Related `pinger_program, query_icmp, netdb_ping_period`

buffered_logs

While this directive used to affect multiple log files, it now only applies to *cache.log*. Squid uses the *stdio* library for *cache.log*. If this directive is enabled, Squid calls `fflush()` after every write. This allows you to see log file entries as they are written. You might want to

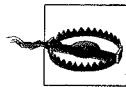
reload_into_ims

disable *buffered_logs* if you are debugging Squid in a way that creates a large number of *cache.log* entries.

Syntax	buffered_logs on off
Default	buffered_logs off
Example	buffered_logs on
Related	<i>cache_log</i>

reload_into_ims

If you enable this directive, Squid adds an If-Modified-Since header to requests that contain a no-cache directive. This is a global version of the reload-into-ims option for the *refresh_pattern* directive (see Chapter 7).



Altering the client's request in this manner is a violation of HTTP.

Syntax	reload_into_ims on off
Default	reload_into_ims off
Example	reload_into_ims on
Related	<i>refresh_pattern</i>

always_direct

The *always_direct* access rules define a class of requests that must always be forwarded directly to the origin server. For these, Squid doesn't query or otherwise consider any neighbor caches. See Chapter 10.

Syntax	always_direct allow deny [!]ACLname ...
Default	No default
Example	acl LocalServers dst 172.17.0.0/24 always_direct allow LocalServers
Related	<i>acl</i> , <i>never_direct</i> , <i>prefer_direct</i> , <i>nonhierarchical_direct</i> , <i>minimum_direct_hops</i> , <i>minimum_direct_rtt</i> , <i>cache_peer_access</i>

never_direct

The *never_direct* access rules define a class of requests that must never be forwarded to the origin server. For these, Squid must select an appropriate neighbor cache to handle the request. See Chapter 10.

Syntax	never_direct allow deny [!]ACLname ...
Default	No default

Example `acl SpecialServers dstdomain .example.com`
 `never_direct allow SpecialServers`

Related `acl, always_direct, prefer_direct, nonhierarchical_direct, minimum_direct_hops, minimum_direct_rtt,`
 `cache_peer_access`

header_access

This directive defines a set of access rules for filtering HTTP headers from both requests and responses. You can use it to remove headers that may violate your privacy, or that cause interoperation issues. For example, this configuration removes Cookie headers sent to a well-known web advertising company:

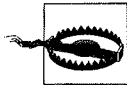
```
acl DC dstdomain .doubleclick.net
header_access Cookie deny DC
```

The *header-name* field must be one of the HTTP headers Squid knows about or one of the keywords *Other* or *All*. Squid currently knows the following HTTP headers:

Accept	Accept-Charset	Accept-Encoding
Accept-Language	Accept-Ranges	Age
Allow	Authentication-Info	Authorization
Cache-Control	Connection	Content-Base
Content-Encoding	Content-Language	Content-Length
Content-Location	Content-MD5	Content-Range
Content-Type	Cookie	Date
ETag	Expires	From
Host	If-Match	If-Modified-Since
If-None-Match	If-Range	Last-Modified
Link	Location	Max-Forwards
Mime-Version	Negotiate	Pragma
Proxy-Authenticate	Proxy-Authentication-Info	Proxy-Authorization
Proxy-Connection	Public	Range
Referer	Request-Range	Retry-After
Server	Set-Cookie	Title
Transfer-Encoding	Upgrade	User-Agent
Vary	Via	WWW-Authenticate
Warning	X-Accelerator-Vary	X-Cache
X-Cache-Lookup	X-Forwarded-For	X-Request-URI
X-Squid-Error		

Unfortunately, you can't refer to an unknown header individually. The best you can do is use the keyword *Other* to refer to all unknown HTTP headers. The keyword *All* refers to all (known and unknown) HTTP headers.

Note that if you deny the *Via* header, Squid can't detect forwarding loops (see Chapter 10).



Removing headers from requests and responses is a violation of HTTP.

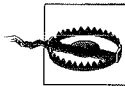
header_replace

Syntax `header_access header-name allow|deny [!]ACLname ...`
Default No default
Example `header_access From deny All`
Related `acl, header_replace`

header_replace

This directive works in conjunction with `header_access`. If you use `header_replace`, Squid replaces HTTP headers that are denied (removed) by an `header_access` rule. In other words, an HTTP header must be filtered out by `header_access` before it can be replaced by `header_replace`.

`header_replace` isn't especially flexible. You can only define one replacement value for each header. You can't, for example, use one value for some requests and a different value for others.



Changing HTTP request and response headers is a violation of HTTP.

Syntax `header_replace header-name string`
Default No default
Example `header_replace User-Agent Nutscape/1.0 (CP/M; 8-bit)`
Related `header_access`

icon_directory

This directive specifies the location of the icons Squid uses in FTP and Gopher directory listings. The icon filenames are defined in `mime.conf` (see the “`mime_table`” section). If you don't like Squid's icons, you can use your own, as long as the filenames found in `mime.conf` exist in the `icon_directory` directory.

Syntax `icon_directory directory`
Default `icon_directory $prefix/share/icons`
Example `icon_directory /usr/local/squid/share/myicons`
Related `error_directory, mime_table`

error_directory

This directive specifies the location of Squid's error message files. If you want to customize the error messages, you should put them into a nondefault directory. Otherwise, they may be overwritten if you run `make install` in the future.

Syntax `error_directory directory`
Default `error_directory $prefix/share/errors/$language`

Example `error_directory /usr/local/squid/share/my_errors`
 Related `icon_directory, err_html_text, deny_info`

maximum_single_addr_tries

This directive places a limit on the number of times Squid attempts to connect to a single IP address when forwarding a request. It can't be set higher than 10.

Syntax `maximum_single_addr_tries N`
 Default `maximum_single_addr_tries 3`
 Example `maximum_single_addr_tries 5`
 Related `connect_timeout`

snmp_port

This is the UDP port to which Squid listens for SNMP queries. SNMP support requires the `--enable-snmp` option to `./configure`. Set the SNMP port to 0 if Squid shouldn't accept any SNMP messages.

Syntax `snmp_port port-number`
 Default `snmp_port 3401`
 Example `snmp_port 3161`
 Related `snmp_access, snmp_incoming_address, snmp_outgoing_address`

snmp_access

The `snmp_access` rules apply to SNMP queries. Although this is a standard Squid access list rule, many ACL elements are undefined for SNMP. In fact, you can only use `src` and `snmp_community` ACLs.

Syntax `snmp_access allow|deny [!]ACLname ...`
 Default No default (all queries denied by default)
 Example `acl SNMPPasswd snmp_community sekret`
 `acl SNMPClients src 172.16.1.2 10.0.5.1`
 `acl All src 0/0`
 `snmp_access allow SNMPClients SNMPPasswd`
 `snmp_access deny All`
 Related `acl, snmp_port`

snmp_incoming_address

By default, Squid opens the SNMP socket to receive packets on all local interfaces. You can use this directive to bind the SNMP socket to a particular interface.

Syntax `snmp_incoming_address ip-address`
 Default `snmp_incoming_address 0.0.0.0`

`snmp_outgoing_address`

Example `snmp_incoming_address 172.16.0.1`
Related `snmp_port, snmp_access, udp_incoming_address`

snmp_outgoing_address

Squid uses a single SNMP socket by default. If you set this directive, however, Squid opens a separate socket for SNMP replies only. In most cases, you shouldn't use this directive because SNMP queries should come from the same address to which the queries are sent.

Syntax `snmp_outgoing_address ip-address`
Default No default
Example `snmp_outgoing_address 192.168.5.5`
Related `snmp_port, snmp_access, udp_outgoing_address`

as_whois_server

This is the hostname of the *whois* server Squid uses to resolve Autonomous System numbers into IP networks. You only need to worry about this if you use AS-based ACLs (*src_as*, *dst_as*).

The default server, *whois.ra.net*, seems to work relatively well. It may be too far away (and unreliable) for non-U.S. users. If you know of a local *whois* server that returns AS queries, feel free to use it instead.

Syntax `as_whois_server hostname`
Default `as_whois_server whois.ra.net`
Example `as_whois_server whois.host.name`
Related *acl*

wccp_router

This directive defines Squid's home router for WCCP. When you enter an IP address (or hostname) here, Squid sends WCCP "Here I Am" messages to the router. See Chapter 9 for more information.

Routers, by definition, have multiple network interfaces. You should probably use the address of the interface that is connected, or has the route, to Squid. Squid ignores WCCP messages that don't have the *wccp_router* value as their source address.

Syntax `wccp_router ip-address`
Default No default
Example `wccp_router 172.16.5.1`
Related `wccp_version, wccp_incoming_address, wccp_outgoing_address`

wccp_version

This particular version number refers to second field of the WCCP “Here I Am” message. It isn’t the same as WCCPv1 versus WCCPv2. Some users report that older installations of Cisco IOS only work when this directive is set to 3.

Syntax	<code>wccp_version <i>N</i></code>
Default	<code>wccp_version 4</code>
Example	<code>wccp_version 3</code>
Related	<code>wccp_router</code>

wccp_incoming_address

Squid listens for WCCP messages on all local interfaces by default. If you set this directive, Squid listens on only the specified address.

Syntax	<code>wccp_incoming_address <i>ip-address</i></code>
Default	<code>wccp_incoming_address 0.0.0.0</code>
Example	<code>wccp_incoming_address 10.1.2.3</code>
Related	<code>wccp_router</code> , <code>wccp_outgoing_address</code> , <code>udp_incoming_address</code>

wccp_outgoing_address

If, for some reason, you want Squid to send and receive WCCP messages on different interfaces, set this directive to the address of the outgoing interface. If this directive isn’t set, as is the default, Squid uses the same socket for incoming and outgoing messages.

Syntax	<code>wccp_outgoing_address <i>ip-address</i></code>
Default	No default
Example	<code>wccp_outgoing_address 172.16.1.1</code>
Related	<code>wccp_router</code> , <code>wccp_incoming_address</code> , <code>udp_outgoing_address</code>

delay_pools

This directive specifies the number of delay pools that you will later define with the `delay_class` and `delay_parameters` directives. It tells Squid the size of certain arrays used in the delay pools implementation. It must appear in the configuration file before the other delay pools directives.

Note that in order to use delay pools, you must give the `--enable-delay-pools` option to `./configure`.

Syntax	<code>delay_pools <i>N</i></code>
Default	<code>delay_pools 0</code>
Example	<code>delay_pools 4</code>
Related	<code>delay_class</code> , <code>delay_access</code> , <code>delay_parameters</code> , <code>delay_initial_bucket_level</code>

delay_class

This directive defines the class of each delay pool. The first argument is the delay pool index. Index values start at 1 and must be less than or equal to the *delay_pools* value. The second argument is the delay class, which has three possible values:

- A class 1 pool uses a single, aggregate bucket for all traffic that applies to the pool.
- A class 2 pool uses a single, aggregate bucket, as well as 256 individual buckets. The individual bucket is chosen by the last octet of the client's IPv4 address.
- A class 3 bucket uses a single, aggregate bucket, 256 network buckets, and 65,536 individual buckets. The network bucket is chosen based on the third octet of the client's IPv4 address. The individual bucket is chosen by the third and fourth octets.

Note that the class 2 and class 3 pools have multiple types of buckets (aggregate, network, individual). A client receives a traffic allocation from all relevant buckets, not just one of them. In other words, if any of the relevant buckets are empty, the client doesn't receive any traffic allocation.

Syntax	<code>delay_class pool-number class</code>
Default	No default
Example	<code>delay_class 1 2</code> <code>delay_class 2 3</code>
Related	<i>delay_pools, delay_access, delay_parameters, delay_initial_bucket_level</i>

delay_access

This directive maps a client request to a particular delay pool. A client's cache miss is delayed only if it is "allowed" by one of the *delay_access* rules. Squid checks the access rules for all pools in order. If a particular request is denied by all *delay_access* rules, it isn't delayed. You must define at least one rule to use delay pools.

Syntax	<code>delay_access pool-number allow deny [!]ACLname ...</code>
Default	No default
Example	<code>acl Dorms src 172.17.0.0/16</code> <code>delay_access 1 allow Dorms</code>
Related	<i>delay_pools, delay_class, delay_parameters, delay_initial_bucket_level</i>

delay_parameters

The *delay_parameters* directive determines the fill rate and capacity for each delay pools bucket. Following the pool number, you must write one, two, or three pairs of numbers. The number of pairs is the same as the pool's class. A class 1 pool takes one pair, a class 2 pool takes two pairs, and a class 3 pool takes three pairs.

Each pair of numbers specifies the fill rate and maximum bucket size. The fill rate should not be larger than the maximum size. The units are number of bytes. Thus, if you are thinking in terms of bits per second, you must divide by 8 to get bytes per second. For

example, if you want to define a bucket that refills at a rate of 100 Kbits/sec, and holds no more than 300 Kbits (3 seconds) of traffic, you would write 12500/37500.

Syntax	<code>delay_parameters pool-number aggr-rate/aggr-max [ind-rate/ind-max [net-rate/net-max]]</code>
Default	No default
Example	<code>delay_parameters 2 16000/32000 4000/8000</code>
Related	<code>delay_pools</code> , <code>delay_class</code> , <code>delay_access</code> , <code>delay_initial_bucket_level</code>

delay_initial_bucket_level

This directive determines the amount of traffic that Squid puts into newly created buckets. A bucket is created when Squid starts up or is reconfigured. For class 2 and 3 pools, individual and network buckets are created upon the first client request that uses the bucket. The `delay_initial_bucket_level` value is a percentage of the bucket's maximum size.

Syntax	<code>delay_initial_bucket_level percent</code>
Default	<code>delay_initial_bucket_level 50</code>
Example	<code>delay_initial_bucket_level 100</code>
Related	<code>delay_pools</code> , <code>delay_class</code> , <code>delay_access</code> , <code>delay_parameters</code>

incoming_icp_average

This directive controls the low-level routines that periodically check the ICP socket for incoming queries and replies. The algorithm is relatively complex to fully describe here. The idea is to make sure Squid checks the ICP socket frequently enough to handle the ICP load but not so often that it is a waste of time. This directive specifies the number of normal I/O events that should occur between checks to the ICP socket. A normal I/O event refers to reading from, and writing to, client- and server-side TCP sockets.

Unless you have a thorough understanding of the polling algorithms in the source code, I strongly recommend that you leave this directive set to its default value.

Syntax	<code>incoming_icp_average number</code>
Default	<code>incoming_icp_average 6</code>
Example	<code>incoming_icp_average 20</code>
Related	<code>incoming_http_average</code> , <code>incoming_dns_average</code>

incoming_http_average

This directive is similar to `incoming_icp_average`, except that it refers to the HTTP socket with which Squid accepts new client requests. Unless you have a thorough understanding of the polling algorithms in the source code, I strongly recommend that you leave this directive set to its default value.

Syntax	<code>incoming_http_average number</code>
Default	<code>incoming_http_average 4</code>

`incoming_dns_average`

Example `incoming_http_average 15`
Related `incoming_icp_average, incoming_dns_average`

incoming_dns_average

This directive is similar to `incoming_icp_average`, except that it refers to the UDP socket with which Squid receives DNS responses. Unless you have a thorough understanding of the polling algorithms in the source code, I strongly recommend that you leave this directive set to its default value.

Syntax `incoming_dns_average number`
Default `incoming_dns_average 4`
Example `incoming_dns_average 8`
Related `incoming_icp_average, incoming_http_average`

min_icp_poll_cnt

This directive controls the low-level routines that periodically check the ICP socket for incoming queries and replies. It specifies a lower limit on the number of normal I/O events that must occur between checks to the ICP socket. Unless you have a thorough understanding of the polling algorithms in the source code, I strongly recommend that you leave this directive set to its default value.

Syntax `min_icp_poll_cnt number`
Default `min_icp_poll_cnt 8`
Example `min_icp_poll_cnt 10`
Related `incoming_icp_average`

min_dns_poll_cnt

This directive is similar to `min_icp_poll_cnt`, except that it applies to the UDP socket with which Squid receives DNS replies. Unless you have a thorough understanding of the polling algorithms in the source code, I strongly recommend that you leave this directive set to its default value.

Syntax `min_dns_poll_cnt number`
Default `min_dns_poll_cnt 8`
Example `min_dns_poll_cnt 10`
Related `incoming_dns_average`

min_http_poll_cnt

This directive is similar to `min_icp_poll_cnt`, except that it applies to the TCP socket with which Squid accepts new client requests. Unless you have a thorough understanding of the

polling algorithms in the source code, I strongly recommend that you leave this directive set to its default value.

Syntax	<code>min_http_poll_cnt <i>number</i></code>
Default	<code>min_http_poll_cnt 8</code>
Example	<code>min_http_poll_cnt 12</code>
Related	<code>incoming_http_average</code>

max_open_disk_fds

This directive defines an upper limit on the number of file descriptors that Squid should open for reading and writing cache files on disk. It is relevant for only the *ufs* and *aufs* storage schemes. It is a relatively simple hack for measuring the level of Squid's disk activity. Experience shows that performance degrades significantly when Squid hits a file-system bottleneck.

If Squid reaches this limit, it doesn't attempt to store subsequent cachable responses. Each time that happens, Squid increments the `no_too_many_open_files` counter (see Chapter 14). Note that hitting this limit has a negative impact on your hit ratio. You can monitor the number of open disk files by requesting the *info* page from the cache manager (see Chapter 14).

If you set this directive to 0, Squid doesn't place any limits on the number of open disk file descriptors.

Syntax	<code>max_open_disk_fds <i>N</i></code>
Default	<code>max_open_disk_fds 0</code>
Example	<code>max_open_disk_fds 100</code>

offline_mode

When you enable *offline_mode*, Squid returns every cached response as an unvalidated cache hit. These are tagged with `TCP_OFFLINE_HIT` in *access.log*. When in this mode, Squid still attempts to forward cache misses. If your system truly is offline, some requests may hang while waiting for the DNS or HTTP transaction to timeout.

Syntax	<code>offline_mode on off</code>
Default	<code>offline_mode off</code>
Example	<code>offline_mode on</code>

uri_whitespace

This directive tells Squid what to do about URIs that contain whitespace characters (i.e., space and tab). The default action is to strip out the whitespace and shift the valid characters down as necessary. This is the behavior recommended by RFC 2396.

If you set this directive to *allow*, Squid doesn't change the URI. It is passed through to the origin server as is. This setting may cause some problems with redirectors and log file

broken_posts

parsers. Both use whitespace as a field delimiter, and a URI with whitespace adds an additional field (or fields) to the redirector input line and the *access.log* entry.

The deny setting instructs Squid to deny such a request, as though it were blocked by the access control rules. Note, however, that the URI is still written to *access.log* with the whitespace characters.

With the encode setting, Squid changes whitespace characters into their RFC 1738 equivalents. When some origin servers generate URIs that contain whitespace, this is what they should be doing in the first place.

Finally, the chop setting instructs Squid to simply cut off the URI at the first whitespace character.

Syntax	<code>uri_whitespace allow deny strip encode chop</code>
Default	<code>uri_whitespace strip</code>
Example	<code>uri_whitespace deny</code>
Related	<code>access_log, redirector_program</code>

broken_posts

Certain buggy HTTP servers expect two extra bytes, CR and LF characters, following an HTTP POST message body. It seems unlikely that such uncompliant servers are still in use today. Nonetheless, this access rule list exists to accommodate them. When a request matches a *broken_posts* rule, Squid appends the extra CRLF characters.

Syntax	<code>broken_posts allow deny [!]ACLname ...</code>
Default	No default
Example	<code>acl NeedsExtraCRLF dstdomain broken.server.com</code> <code>broken_posts allow NeedsExtraCRLF</code>
Related	<code>http_access, acl</code>

mcast_miss_addr

The *multicast miss stream* is a largely undocumented and unsupported Squid feature. The basic idea is to send a multicast message, containing a URI, for each cache miss. The messages are encrypted with a modest algorithm to prevent casual eavesdropping.

To use this feature, you must manually define the `MULTICAST_MISS_STREAM` preprocessor symbol before compiling Squid. To learn more about this feature, read the source code surrounded by `#if MULTICAST_MISS_STREAM` in *src/access_log.c*.

Syntax	<code>mcast_miss_addr multicast-address</code>
Default	No default
Example	<code>mcast_miss_addr 224.0.1.1</code>
Related	<code>mcast_miss_ttl, mcast_miss_port, mcast_miss_encode_key</code>

mcast_miss_ttl

This is the multicast TTL assigned to outgoing miss stream messages. See the discussion of multicast TTLs in Chapter 10.

Syntax	<code>mcast_miss_ttl <i>N</i></code>
Default	<code>mcast_miss_ttl 16</code>
Example	<code>mcast_miss_ttl 32</code>
Related	<code>mcast_miss_addr</code> , <code>mcast_miss_port</code> , <code>mcast_miss_encode_key</code>

mcast_miss_port

This is the UDP port number to which multicast miss stream messages are sent.

Syntax	<code>mcast_miss_port <i>port-number</i></code>
Default	<code>mcast_miss_port 3135</code>
Example	<code>mcast_miss_port 999</code>
Related	<code>mcast_miss_addr</code> , <code>mcast_miss_ttl</code> , <code>mcast_miss_encode_key</code>

mcast_miss_encode_key

Squid uses the Tiny Encryption Algorithm (TEA) to encrypt multicast miss messages. This directive specifies the encryption key, which should be 128 bits long.

Syntax	<code>mcast_miss_encode_key <i>string</i></code>
Default	<code>mcast_miss_encode_key XXXXXXXXXXXXXXXX</code>
Example	<code>mcast_miss_encode_key MySekRitPassWord</code>
Related	<code>mcast_miss_addr</code> , <code>mcast_miss_ttl</code> , <code>mcast_miss_port</code>

nonhierarchical_direct

A hierarchical request is one that looks like it might result in a cachable response, and therefore might be cached by one of Squid's neighbors. If your Squid doesn't have any neighbors, you don't need to worry about this directive.

By default, Squid prefers to skip the neighbor selection step for nonhierarchical requests (uncachable responses) because the request probably won't result in a cache hit. You can reverse this behavior by disabling the *nonhierarchical_direct* directive. See Chapter 10.

Syntax	<code>nonhierarchical_direct on off</code>
Default	<code>nonhierarchical_direct on</code>
Example	<code>nonhierarchical_direct off</code>
Related	<code>prefer_direct</code> , <code>never_direct</code> , <code>always_direct</code>

prefer_direct

prefer_direct

This directive affects Squid's neighbor selection algorithm for hierarchical requests (cacheable responses). It is only relevant if you have one or more neighbor caches. When Squid builds a list of next-hop locations for cache misses, it puts neighbor caches before the origin server by default. If you would rather have Squid put the origin server before neighbors, enable the *prefer_direct* directive. See Chapter 10.

Syntax	prefer_direct on off
Default	prefer_direct off
Example	prefer_direct on
Related	nonhierarchical_direct, never_direct, always_direct

strip_query_terms

When this directive is enabled, Squid doesn't log URI query terms in *access.log*. This feature is intended to give your users some privacy. It is enabled by default.

Syntax	strip_query_terms on off
Default	strip_query_terms on
Example	strip_query_terms off
Related	access_log, client_netmask

coredump_dir

Normally Squid doesn't change its current directory at startup. While this isn't usually a problem, it can be if Squid wants to leave a core-dump file. If the *core* file is very large, it might fill up a disk partition. Additionally, the *core* won't be created at all if Squid doesn't have permission to write in the current directory.

This directive changes Squid's current directory. You should set it to a location that has sufficient space, and appropriate permissions, for a large *core* file.

Note that the *coredump_dir* directive is used only when Squid starts up. If you change the value while Squid is running and then reconfigure, Squid doesn't change the current directory.

Syntax	coredump_dir <i>pathname</i>
Default	No default
Example	coredump_dir /squid/var

ignore_unknown_nameservers

Squid normally checks that DNS replies come from the same IP address to which the query was sent. If the addresses don't match, Squid writes a warning to *cache.log* and ignores the reply. Some installations use an */etc/resolv.conf* trick to query any local name server. If the name server IP address is 0.0.0.0, DNS queries are broadcast on the local area network.

The replies, however, come from specific addresses. If you want to use this trick, you must disable the *ignore_unknown_nameservers* directive.

Syntax	<code>ignore_unknown_nameservers on off</code>
Default	<code>ignore_unknown_nameservers on</code>
Example	<code>ignore_unknown_nameservers off</code>
Related	<i>dns_nameservers</i>

digest_generation

This directive controls whether or not Squid generates a Cache Digest for its own contents. It is enabled by default, when you give the `--enable-cache-digests` option to `./configure`. You may want to disable it if you know that you don't have any neighbors who request your digest.

Syntax	<code>digest_generation on off</code>
Default	<code>digest_generation on</code>
Example	<code>digest_generation off</code>
Related	<i>cache_peer, digest_bits_per_entry, digest_rebuild_period, digest_rewrite_period, digest_swapout_chunk_size, digest_rebuild_chunk_percentage</i>

digest_bits_per_entry

This directive affects the size of Squid's Cache Digest, based on the estimate for the total number of cache entries. Reducing the size of the digest results in lower memory usage but a higher false hit probability.

Syntax	<code>digest_bits_per_entry <i>number</i></code>
Default	<code>digest_bits_per_entry 5</code>
Example	<code>digest_bits_per_entry 4</code>
Related	<i>digest_generation, store_avg_object_size, cache_dir</i>

digest_rebuild_period

The digest rebuild period is how often Squid generates the digest of its own cache. This is a fairly CPU-intensive procedure, so you don't want to run it too often. On the other hand, the digest becomes less representative of Squid's contents as more time passes.

Syntax	<code>digest_rebuild_period <i>time-specification</i></code>
Default	<code>digest_rebuild_period 1 hour</code>
Example	<code>digest_rebuild_period 4 hours</code>
Related	<i>digest_generation, digest_rewrite_period, digest_swapout_chunk_size, digest_rebuild_chunk_percentage</i>

`digest_rewrite_period`

digest_rewrite_period

The digest rewrite period is how often Squid generates an on-disk cached HTTP response for its Cache Digest. This is the response sent to neighbors that request Squid's digest. In most cases *digest_rewrite_period* should be the same as *digest_rebuild_period*.

Syntax	<code>digest_rewrite_period</code> <i>time-specification</i>
Default	<code>digest_rewrite_period</code> 1 hour
Example	<code>digest_rewrite_period</code> 4 hours
Related	<i>digest_generation, digest_rebuild_period, digest_swapout_chunk_size, digest_rebuild_chunk_percentage</i>

digest_swapout_chunk_size

This directive controls the amount of data written to disk for each call to the digest swapout function. Squid services normal cache traffic (client requests, server responses, etc.) in between digest swapout calls. If the value is too large, Squid blocks on the disk I/O and delays normal cache traffic.

Syntax	<code>digest_swapout_chunk_size</code> <i>size-specification</i>
Default	<code>digest_swapout_chunk_size</code> 4 KB
Example	<code>digest_swapout_chunk_size</code> 16 KB
Related	<i>digest_generation, digest_rewrite_period, digest_rebuild_chunk_percentage</i>

digest_rebuild_chunk_percentage

This directive specifies the percentage of hash-table buckets Squid scans during each call to the digest rebuild procedure. Squid services normal cache traffic in between these calls. Since this scanning is CPU-intensive, user requests may be delayed for a small, but noticeable amount of time. If you suspect a performance problem during the rebuild phase, decrease the *digest_rebuild_chunk_percentage* value.

Syntax	<code>digest_rebuild_chunk_percentage</code> <i>percentage</i>
Default	<code>digest_rebuild_chunk_percentage</code> 10
Example	<code>digest_rebuild_chunk_percentage</code> 3
Related	<i>digest_generation, digest_rebuild_period, store_objects_per_bucket</i>

chroot

When you specify a value for this directive, Squid passes it to the `chroot()` system call. This provides an extra level of security by isolating the Squid process(es) from the rest of your filesystem. See Chapter 5 for more information.

Syntax	<code>chroot</code> <i>pathname</i>
Default	No default
Example	<code>chroot</code> /squid

client_persistent_connections

This directive controls whether or not Squid uses persistent HTTP connections to cache clients. When disabled, Squid sends `Connection: close` headers in its responses to clients. If you suspect problems caused by client-side persistent connections, disable this directive.

Syntax	<code>client_persistent_connections on off</code>
Default	<code>client_persistent_connections on</code>
Example	<code>client_persistent_connections off</code>
Related	<code>server_persistent_connections</code> , <code>pipeline_prefetch</code>

server_persistent_connections

This directive controls whether or not Squid uses persistent HTTP connections to origin servers and neighbors. When disabled, Squid sends `Connection: close` headers in forwarded requests. If you suspect problems caused by server-side persistent connections, disable this directive.

Syntax	<code>server_persistent_connections on off</code>
Default	<code>server_persistent_connections on</code>
Example	<code>server_persistent_connections off</code>
Related	<code>client_persistent_connections</code>

pipeline_prefetch

This directive controls whether or not Squid prefetches pipelined requests. It is disabled by default, so Squid acts only on one request at a time (per connection). If you enable this directive, Squid processes up to two client requests at once.

Note that the order of responses must match the order of requests. Thus, if the prefetched (second) request completes before the first, it is delayed until the first response is sent.

Squid doesn't implement pipelining on the server-side. It always opens a new connection to an origin server (or neighbor) if there are no idle persistent connections.

Syntax	<code>pipeline_prefetch on off</code>
Default	<code>pipeline_prefetch off</code>
Example	<code>pipeline_prefetch on</code>
Related	<code>client_persistent_connections</code>

extension_methods

HTTP (RFC 2616) allows clients and servers to use their own extension methods. If requests with nonstandard HTTP methods go through Squid, the client receives an "Invalid Request" error message. Squid also writes a *cache.log* entry, such as this:

```
2003/09/29 13:40:24| parseHttpRequest: Unsupported method 'XGET'
```

request_entities

If you want Squid to accept such requests, you must tell it about the nonstandard methods by listing them after the *extension_methods* directive.

Syntax	<code>extension_methods HTTP-method ...</code>
Default	No default
Example	<code>extension_methods XGET XPOST</code>

request_entities

This directive determines how Squid handles GET and HEAD requests that have message bodies (entities). Such requests normally don't contain bodies. There is some confusion about whether or not RFC 2616 allows entities in GET/HEAD requests. Squid denies such requests by default. If you would rather have Squid accept them, enable the *request_entities* directive.

Syntax	<code>request_entities on off</code>
Default	<code>request_entities off</code>
Example	<code>request_entities on</code>

high_response_time_warning

If you provide a non-zero value for this directive, Squid periodically checks the client-side median response time. If it's above this threshold, Squid prints a warning message in *cache.log*. The value is given in milliseconds.

Syntax	<code>high_response_time_warning milliseconds</code>
Default	<code>high_response_time_warning 0</code>
Example	<code>high_response_time_warning 2000</code>
Related	<i>high_page_fault_warning, high_memory_warning</i>

high_page_fault_warning

If you provide a nonzero value for this directive, Squid periodically checks the process page fault rate. Page faults generally occur when the Squid process doesn't fit entirely in memory. A moderate number of page faults can significantly degrade performance. If the one-minute average rate (page faults per second) exceeds this threshold, Squid prints a warning message in *cache.log*.

Syntax	<code>high_page_fault_warning number</code>
Default	<code>high_page_fault_warning 0</code>
Example	<code>high_page_fault_warning 5</code>
Related	<i>high_response_time_warning, high_memory_warning</i>

high_memory_warning

If you provide a nonzero value for this directive, Squid periodically checks process size. A large process size can lead to page faults and a significant performance degradation. Squid uses either `mstats()`, `mallinfo()`, or `sbrk()` to get the process size. If it exceeds the given threshold, Squid prints a warning message in *cache.log*.

Syntax	<code>high_memory_warning size-specification</code>
Default	<code>high_memory_warning 0</code>
Example	<code>high_memory_warning 400 MB</code>
Related	<i>high_response_time_warning, high_page_fault_warning</i>

ie_refresh

In Chapter 9, I explained that Internet Explorer versions prior to 5.5 SP1 have a bug that make it unable to force a validation of cached responses when using HTTP interception. This directive provides a partial workaround for the bug. When enabled, Squid pretends that the request contains a `no-cache` directive. Thus, Squid always forwards these requests on to the origin server or a neighbor.

Note this affects only requests that meet the following requirements:

- The User-Agent header indicates Internet Explorer Version 3, 4, 5.0, or 5.01.
- The If-Modified-Since header is present.
- The request contains a partial URI because it was intercepted (see Chapter 9) or Squid is a surrogate (see Chapter 15).

Squid versions prior to 2.5.STABLE3 contain a bug related to this feature. Although Squid behaves as though the client's request contains a `no-cache` directive, it doesn't add that directive to the outgoing request. This is a problem if you have one or more neighbor caches. Because the request received by the neighbor doesn't contain a `no-cache` directive, it may decide to return a cache hit, rather than forward it on to the origin server.

Later versions include the `no-cache` directive so that such requests should always reach the origin server.

Syntax	<code>ie_refresh on off</code>
Default	<code>ie_refresh off</code>
Example	<code>ie_refresh on</code>

vary_ignore_expire

When certain HTTP/1.1 origin servers receive an HTTP/1.0 request (e.g., from Squid), and the response contains a `Vary` header, they also add an `Expires` header set to the current time. This is to prevent HTTP/1.0 caches, which may not understand the `Vary` header, from incorrectly reusing a cached response.

Squid understands and implements the `Vary` header but still sends the string "HTTP/1.0" in its requests. You'll need to enable this directive if you want to get cache hits from

`sleep_after_fork`

responses with Vary and with Expires equal to Date. This directive is somewhat dangerous because the origin server may have its own reasons (other than maintaining backward compatibility) for setting the Expires header.

Syntax	<code>vary_ignore_expire on off</code>
Default	<code>vary_ignore_expire off</code>
Example	<code>vary_ignore_expire on</code>

sleep_after_fork

Squid uses the `fork()` system call to spawn helper processes, such as redirectors, authenticators, and DNS resolvers. On some systems, a rapid sequence of `fork()` calls consumes all available real and virtual memory. Thus, a `fork()` call may fail with an “out of memory” error. Note that this isn’t necessarily a fatal error. Squid continues running as long as at least 50% of helper processes are successfully started.

To alleviate this problem, you can instruct Squid to sleep for a small amount of time after each `fork()` call. This gives the recently forked process time to complete its `exec()` call and free up the memory.

Don’t set this value too high, especially if you have a large number of helper processes. Squid doesn’t service any client requests until all helpers have been started.

Syntax	<code>sleep_after_fork <i>microseconds</i></code>
Default	<code>sleep_after_fork 0</code>
Example	<code>sleep_after_fork 10000</code>

The Memory Cache

Squid stores some of its recently retrieved objects fully in memory. As you might expect, serving objects from memory is generally faster than reading the data from the disk. In some places, Squid calls this the *hot object cache*. The *cache_mem* directive specifies how much memory Squid should use for in-memory objects.

I usually recommend setting *cache_mem* to a small size, such as something between 8 and 32 MB. If you happen to have tons of extra memory, you can set it higher. In most cases, however, your extra memory is better used by increasing your disk cache size (see Chapter 7).

Many people misunderstand the *cache_mem* directive. They expect it to limit the total amount of memory that Squid uses. Unfortunately, for them, this assumption is incorrect. Squid doesn't have a directive that limits total memory consumption. See Chapters 7 and 16.

The current version of Squid (2.5) stores objects in memory only if they come from the network (origin server or neighbor cache). If Squid reads an object from disk, it doesn't also store it in memory. Older versions of Squid had that functionality. However, it was removed during a major rewrite to simplify the source code.

Only objects smaller than a certain size are held in memory. The *maximum_object_size_in_memory* directive controls this setting. Its default value is 8 KB, which is typically large enough to fit more than half of all responses Squid receives. This directive also limits the amount of memory used for each cache miss as the response is being received. If you have a high request rate but are low on memory, you may want to lower this value to 4 KB. Squid allocates memory for object data in 4-KB chunks. Thus, it makes sense to assign this directive a multiple of 4 KB. Other values end up wasting memory.

In-memory objects fall into one of two groups: *in-transit* or *complete*. Squid uses the memory cache for both types. Complete objects are held in memory only if there is some free space. They have lower priority than in-transit objects. If your cache is busy, the memory cache may contain nothing but in-transit objects (or, *maximum_*

object_size_in_memory chunks of in-transit objects, actually). Furthermore, Squid always allocates memory for in-transit objects, even if it must exceed the *cache_mem* limit. When an in-transit object becomes a complete object, it is kept in memory only if the memory cache size is below the limit.

The *memory_replacement_policy* directive is analogous to *replacement_policy*. It controls the replacement policy for objects cached in memory. Because the memory cache is typically much smaller than the disk cache, your choice of replacement policy may have a bigger impact. See Chapter 7 for a description of available replacement policies.

Delay Pools

Delay pools are Squid's answer to rate limiting and traffic shaping. They work by limiting the rate at which Squid returns data for cache misses. Cache hits are sent as quickly as possible, under the assumption that local bandwidth is plentiful.

Delay pools were written by David Luyer while at the University of Western Australia. The feature was designed for a LAN environment in which different groups of users (for example, students, instructors, and staff) are on different subnets. You'll see some evidence of this in the following descriptions.

Overview

The delay pools are, essentially "bandwidth buckets." A response is delayed until some amount of bandwidth is available from an appropriate bucket. The buckets don't actually store bandwidth (e.g., 100 Kbit/s), but rather some amount of traffic (e.g., 384 KB). Squid adds some amount of traffic to the buckets each second. Cache clients take some amount of traffic out when they receive data from an upstream source (origin server or neighbor).

The size of a bucket determines how much burst bandwidth is available to a client. If a bucket starts out full, a client can take as much traffic as it needs until the bucket becomes empty. The client then receives traffic allotments at the fill rate.

The mapping between Squid clients and actual buckets is a bit complicated. Squid uses three different constructs to do it: access rules, delay pool classes, and types of buckets. First, Squid checks a client request against the *delay_access* list. If the request is a match, it points to a particular delay pool. Each delay pool has a class: 1, 2, or 3. The classes determine which types of buckets are in use. Squid has three types of buckets: aggregate, individual, and network:

- A class 1 pool has a single aggregate bucket.
- A class 2 pool has an aggregate bucket and 256 individual buckets.

- A class 3 pool has an aggregate bucket, 256 network buckets, and 65,536 individual buckets.

As you can probably guess, the individual and network buckets correspond to IP address octets. In a class 2 pool, the individual bucket is determined by the last octet of the client's IPv4 address. In a class 3 pool, the network bucket is determined by the third octet, and the individual bucket by the third and fourth octets.

For the class 2 and 3 delay pools, you can disable buckets you don't want to use. For example, you can define a class 2 pool with only individual buckets by disabling the aggregate bucket.

When a request goes through a pool with more than one bucket type, it takes bandwidth from all buckets. For example, consider a class 3 pool with aggregate, network, and individual buckets. If the individual bucket has 20 KB, the network bucket 30 KB, but the aggregate bucket only 2 KB, the client receives only a 2-KB allotment. Even though some buckets have plenty of traffic, the client is limited by the bucket with the smallest amount.

Configuring Squid

Before you can use delay pools, you must enable the feature when compiling. Use the `--enable-delay-pools` option when running `./configure`. You can then use the following directives to set up the delay pools.

delay_pools

The *delay_pools* directive tells Squid how many pools you want to define. It should go before any other delay pool-configuration directives in *squid.conf*. For example, if you want to have five delay pools:

```
delay_pools 5
```

The next two directives actually define each pool's class and other characteristics.

delay_class

You must use this directive to define the class for each pool. For example, if the first pool is class 3:

```
delay_class 1 3
```

Similarly, if the fourth pool is class 2:

```
delay_class 4 2
```

In theory, you should have one *delay_class* line for each pool. However, if you skip or omit a particular pool, Squid doesn't complain.

delay_parameters

Finally, this is where you define the interesting delay pool parameters. For each pool, you must tell Squid the fill rate and maximum size for each type of bucket. The syntax is:

```
delay_parameters N rate/size [rate/size [rate/size]]
```

The *rate* value is given in bytes per second, and *size* in total bytes. If you think of *rate* in terms of bits per second, you must remember to divide by 8.

Note that if you divide the *size* by the *rate*, you'll know how long it takes (number of seconds) the bucket to go from empty to full when there are no clients using it.

A class 1 pool has just one bucket and might look like this:

```
delay_class 2 1
delay_parameters 2 2000/8000
```

For a class 2 pool, the first bucket is the aggregate, and the second is the group of individual buckets. For example:

```
delay_class 4 2
delay_parameters 4 7000/15000 3000/4000
```

Similarly, for a class 3 pool, the aggregate bucket is first, the network buckets are second, and the individual buckets are third:

```
delay_class 1 3
delay_parameters 1 7000/15000 3000/4000 1000/2000
```

delay_initial_bucket_level

This directive sets the initial level for all buckets when Squid first starts or is reconfigured. It also applies to individual and network buckets, which aren't created until first referenced. The value is a percentage. For example:

```
delay_initial_bucket_level 75%
```

In this case, each newly created bucket is initially filled to 75% of its maximum size.

delay_access

This list of access rules determines which requests go through which delay pools. Requests that are allowed go through the delay pools, while those that are denied aren't delayed at all. If you don't have any *delay_access* rules, Squid doesn't delay any requests.

The syntax for *delay_access* is similar to the other access rule lists (see Chapter 6), except that you must put a pool number before the allow or deny keyword. For example:

```
delay_access 1 allow TheseUsers
delay_access 2 allow OtherUsers
```

Internally, Squid stores a separate access rule list for each delay pool. If a request is allowed by a pool's rules, Squid uses that pool and stops searching. If a request is denied, however, Squid continues examining the rules for remaining pools. In other words, a deny rule causes Squid to stop searching the rules for a single pool but not for all pools.

cache_peer no-delay Option

The *cache_peer* directive has a no-delay option. If set, it makes Squid bypass the delay pools for any requests sent to that neighbor.

Examples

Let's start off with a simple example. Suppose that you have a saturated Internet connection, shared by many users. You can use delay pools to limit the amount of bandwidth that Squid consumes on the link, thus leaving the remaining bandwidth for other applications. Use a class 1 delay pool to limit the bandwidth for all users. For example, this limits everyone to 512 Kbit/s and keeps 1 MB in reserve if Squid is idle:

```
delay_pools 1
delay_class 1 1
delay_parameters 1 65536/1048576
acl All src 0/0
delay_access 1 allow All
```

One of the problems with this simple approach is that some users may receive more than their fair share of the bandwidth. If you want to try something more balanced, use a class 2 delay pool that has individual buckets. Recall that the individual bucket is determined by the fourth octet of the client's IPv4 address. Thus, if you have more than a /24 subnet, you might want to use a class 3 pool instead, which gives you 65536 individual buckets. In this example, I won't use the network buckets. While the overall bandwidth is still 512 Kbit/s, each individual is limited to 128 Kbit/s:

```
delay_pools 1
delay_class 1 3
delay_parameters 1 65536/1048576 -1/-1 16384/262144
acl All src 0/0
delay_access 1 allow All
```

You can also use delay pools to provide different classes of service. For example, you might have important users and unimportant users. In this case, you could use two

class 1 delay pools. Give the important users a higher bandwidth limit than everyone else:

```
delay_pools 2
delay_class 1 1
delay_class 2 1
delay_parameters 1 65536/1048576
delay_parameters 2 10000/50000
acl ImportantUsers src 192.168.8.0/22
acl All src 0/0
delay_access 1 allow ImportantUsers
delay_access 2 allow All
```

Issues

Squid's delay pools are often useful, but not perfect. You need to be aware of a few drawbacks and limitations before you use them.

Fairness

One of the most important things to realize about the current delay pools implementation is that it does nothing to guarantee fairness among all users of a single bucket. This is especially important for aggregate buckets (where sharing is high), but less so for individual buckets (where sharing is low).

Squid generally services requests in order of increasing file descriptors. Thus, a request whose server-side TCP connection has a lower file descriptor may receive more bandwidth from a shared bucket than it should.

Application Versus Transport Layer

Bandwidth shaping and rate limiting usually operate at the network transport layer. There, the flow of packets can be controlled very precisely. Delay pools, however, are implemented in the application layer. Because Squid doesn't actually send and receive TCP packets (the kernel does), it has less control over the flow of individual packets. Rather than controlling the transmission and receipt of packets on the wire, Squid controls only how many bytes to read from the kernel.

This means, for example, that incoming response data is queued up in the kernel. The TCP/IP stack can buffer some number of bytes that haven't yet been read by Squid. On most systems, the default TCP receive buffer size is usually between 32 KB and 64 KB. In other words, this much data can arrive over the network very quickly, regardless of anything Squid can do. On the one hand, it seems silly to read this data slowly even though it is already on your system. On the other hand, because the client doesn't receive the whole response right away, it is likely to postpone any future requests until the delayed responses are complete.

If you are concerned that the kernel buffers too much server-side data, you can decrease the TCP receive buffer size with the `tcp_recv_bufsize` directive. Even better, your operating system probably has a way to set this parameter for the whole system. On NetBSD/FreeBSD/OpenBSD, you can use the `sysctl` variable named `net.inet.tcp.recvspace`. For Linux, read about `/proc/sys/net/ipv4/tcp_rmem` in *Documentation/networking/ip-sysctl.txt*.

Fixed Subnetting Scheme

The current delay pools implementation assumes that your LAN uses /24 (class C) subnets, and that all users are in the same /16 (class B) subnet. This might not be so bad, depending on how your network is configured. However, it would be nice if the delay pools subnetting scheme were fully customizable.

If your address space is larger than a /24 and smaller than a /16, you can always create a class 3 pool and treat it as a class 2 pool (that is one of the examples given earlier).

If you use just one class 2 pool with more than 256 users, some users will share the individual buckets. That might not be so bad, unless you happen to have a bunch of heavy users fighting over one measly bucket.

You might also create multiple class 2 pools and use `delay_access` rules to divide them up among all users. The problem with this approach is that you can't have all users share a single aggregate bucket. Instead, each subgroup has their own aggregate bucket. You can't make a single client go through more than one delay pool.

Monitoring Delay Pools

You can monitor the delay pool levels with the cache manager interface. Request the `delay` page from the CGI interface or with the `squidclient` utility:

```
% squidclient mgr:delay | less
```

See Chapter 14 for a description of the output.

Filesystem Performance Benchmarks

You have a myriad of choices to make when installing and configuring Squid, especially when it comes to the way Squid stores files on disk. Back in Chapter 8, I talked about the various filesystems and storage schemes. Here, I'll provide some hard data on their relative performance.

These tests were done with Web Polygraph, a freely available, high-performance tool for benchmarking HTTP intermediaries (<http://www.web-polygraph.org/>). Over the course of many months, I ran approximately 40 different tests on 5 different operating systems.

The Benchmark Environment

The primary purpose of these benchmarks is to provide a number of measurements that allow you to compare different Squid configurations and features. In order to produce comparable results, I've taken care to minimize any differences between systems being tested.

Hardware for Squid

I used five identical computer systems—one for each of the following operating systems: FreeBSD, Linux, NetBSD, OpenBSD, and Solaris. The boxes are IBM Netfinity servers with one 500-MHz PIII CPU, 1 GB of RAM, an Intel fast-Ethernet NIC, and three 8-GB disk SCSI drives. I realize that these aren't particularly powerful machines by today's standards, but they are good enough for these tests. Anyway, it is more important that they be identical than powerful.

The requirement to use identical hardware means that I can't generate comparable results for other hardware platforms, such as Sun, Digital/Compaq/HP, and others.

Squid Version and Configuration

Except for the *cos*s tests, all results are from Squid Version 2.5.STABLE2. The *cos*s results are from a patched version of 2.5.STABLE3. Those patches have been committed to the source tree for inclusion into 2.5.STABLE4.

Unless otherwise specified, I used only the `--enable-storeio` option when running `./configure` before compiling Squid. For example:

```
% ./configure --enable-storeio=diskd,ufs,null,cos
```

In all cases, Squid is configured to use 7500 MB of each 8.2-GB disk. This is a total cache size of 21.5 GB. Additionally, *access.log* and *store.log* have been disabled in the configuration file. Here is a sample *squid.conf* file:

```
visible_hostname linux-squid.bench.tst
acl All src 0/0
http_access allow All

cache_dir aufs /cache0 7500 16 256
cache_dir aufs /cache1 7500 16 256
cache_dir aufs /cache2 7500 16 256

cache_effective_user nobody
cache_effective_group nobody
cache_access_log /dev/null
cache_store_log none
logfile_rotate 0
```

Web Polygraph Workload

All the tests in this appendix use the same Polygraph workload file.* Meeting this requirement was, perhaps, the hardest part of running these tests. Normally, the desired throughput is a configuration parameter in a Polygraph workload. However, because the sustainable throughput is different for each configuration, my colleague Alex Rousskov and I developed a workload that can be used for all tests.† We call this the “peak finder” workload because it finds the peak throughput for a device under test.

The name “peak finder” is somewhat misleading because, at least in Squid’s case, sustainable throughput decreases over time. The workload is designed to periodically adjust the offered load (throughput) subject to response time requirements. If the measured response time is below a given threshold, Polygraph increases the load. If response time is above the threshold, it decreases the load. Thus, at any point in

* Except for the number-of-spindles tests, in which the cache size depends on the number of disks in use.

† You can download this workload at <http://squidbook.org/extras/pf2-pm4.pg.txt>.

time during the test, we know the maximum throughput that still satisfies the response time requirements.

In order to reach a steady-state condition, the test runs until the cache has been filled twice. Polygraph knows the total cache size (21.5 GB) and keeps track of the amount of fill traffic pulled into the cache. These are responses that are cachable but not cache hits. The test duration, then, depends on the sustainable throughput. When the throughput is low, the test takes longer to complete. Some of these tests took more than 10 days to run.

General Comments

I show, for each test, how the sustainable throughput varies over time. The y-axis shows the throughput (responses per second). The x-axis is the ratio of fill-traffic volume to cache size. Because each test takes a different amount of time, this is a nice way to normalize all the results. The test is over when the cache has been filled twice.

In most traces, you'll see that sustainable throughput decreases over time. At the beginning of the test, the throughput is very high. Here, the disks are empty, and Squid doesn't need to replace old objects. The throughput for a full cache is usually worse than for an empty cache. This is a common characteristic of proxy benchmarks and emphasizes the importance of reaching steady-state conditions. Don't be fooled by impressive results from short tests.

The Throughput, Response Time, and Hit Ratio values given in the summary tables are taken from the last 25% of the test. Here, between 1.5 and 2.0 on the x-axis, the throughput is more or less stable and flat. I report the mean of the throughput, response time, and hit ratio values in this range from the trace data.

Throughput is the most interesting metric in these tests. It is given in responses per second. The rows in each summary table are sorted by throughput.

The response time numbers are less interesting because they are all about the same. I decided to report them to show that, indeed, the results stay within the response time window defined by the workload. The target response time is around 1.5 seconds, but the actual response time varies depending on the particular test.

The response hit ratio values are also not particularly interesting. The ideal hit ratio for this workload is about 58%. Due to an as-yet unresolved Polygraph bug, however, the hit ratio decreases slightly as the test progresses.

Keep in mind that these results are meant to demonstrate the relative performance of different options, rather than the absolute values. You'll get different numbers if you repeat the tests on different hardware.

Linux

Linux is obviously a popular choice for Squid. It supports a wide variety of filesystems and storage schemes. These results come from Linux kernel Version 2.4.19 (released August 2, 2002) with SGI's XFS patches Version 1.2.0 (released Feb 11, 2003) and ReiserFS Version 3.6.25.

The kernel's file descriptor limit is set to 8192. I used this command to configure Squid before compiling:

```
% ./configure --enable-storeio=diskd,ufs,aufs,null,coss --with-aufs-threads=32
```

The Linux results are summarized in Table D-1, and Figure D-1 shows the traces. You can see that *coss* is the best performer, with *aufs* coming in second and *diskd* third. As I'm writing this, *coss* is an experimental feature and not necessarily suitable for a production system. In the long run, you'll probably be better off with *aufs*.

Table D-1. Linux benchmarking results

Storage scheme	Filesystem	Mount options	Throughput (xact/sec)	Response time (sec)	Hit ratio (%)
<i>coss</i>			326.3	1.59	53.9
<i>aufs(1)</i>	<i>ext2fs</i>	<i>noatime</i>	168.5	1.45	56.3
<i>diskd(1)</i>	<i>ext2fs</i>	<i>noatime</i>	149.4	1.53	56.1
<i>aufs(2)</i>	<i>ext2fs</i>		110.0	1.46	55.6
<i>ufs(1)</i>	<i>ext2fs</i>		54.9	1.52	55.6
<i>ufs(2)</i>	<i>ext3fs</i>		48.4	1.49	56.8
<i>ufs(3)</i>	<i>xfs</i>		40.7	1.54	55.3
<i>ufs(4)</i>	<i>reiserfs</i>	<i>notail, noatime</i>	29.7	1.55	55.0
<i>ufs(5)</i>	<i>reiserfs</i>		21.4	1.55	55.1

Note that the *noatime* option gives a significant boost in performance to *aufs*. The throughput jumps from 110 to 168 transactions per second with the addition of this mount option. Linux also has an *async* option, but it is enabled by default. I did not run any tests with *async* disabled.

Of the many filesystem choices, *ext2fs* seems to give the best performance. *ext3fs* (*ext2* plus journaling) is only slightly lower, followed by *xfs*, and *reiserfs*.

FreeBSD

FreeBSD is another popular Squid platform, and my personal favorite. Table D-2 and Figure D-2 summarize the results for FreeBSD. Again, *coss* exhibits the highest throughput, followed by *diskd*. The *aufs* storage scheme doesn't currently run on

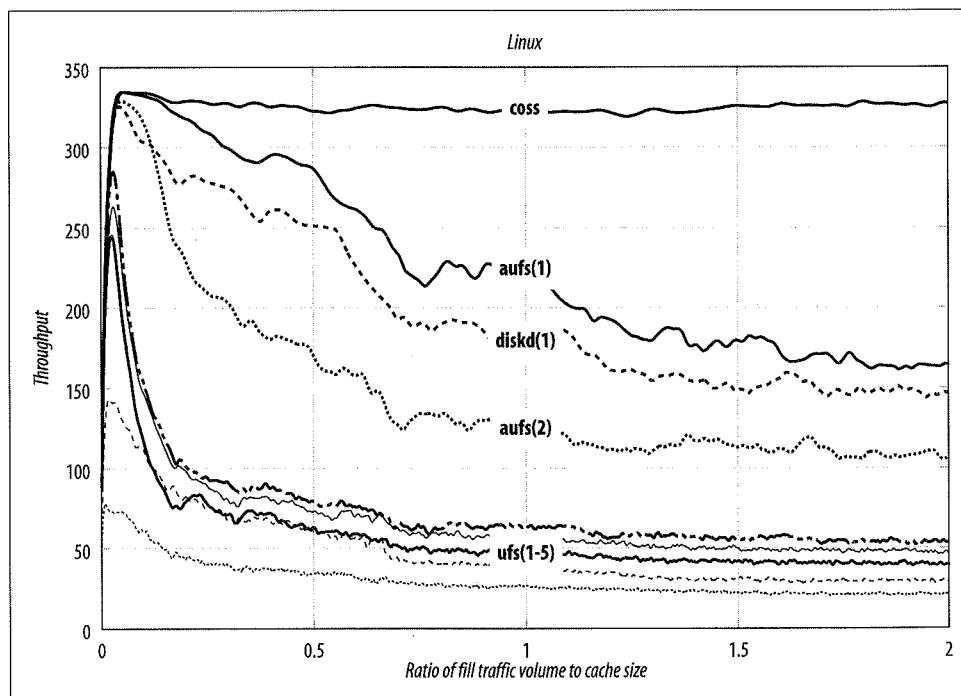


Figure D-1. Linux filesystem benchmarking traces

FreeBSD. These results come from FreeBSD Version 4.8-STABLE (released April 3, 2003). I built a kernel with the following noteworthy options:

```
options      MSGMNB=16384
options      MSGMNI=41
options      MSGSEG=2049
options      MSGSSZ=64
options      MSGTQL=512
options      SHMSEG=16
options      SHMMNI=32
options      SHMMAX=2097152
options      SHMALL=4096
options      MAXFILES=8192
options      NMBCLUSTERS=32768
options      VFS_AIO
```

Table D-2. FreeBSD benchmarking results

Storage scheme	Filesystem	Mount options	Throughput	Response time	Hit ratio
coss			330.7	1.58	54.5
diskd(1)	UFS	async, noatime, softupdate	129.0	1.58	54.1
diskd(2)	UFS		77.4	1.47	56.2
ufs(1)	UFS	async, noatime, softupdate	38.0	1.49	56.8

Table D-2. FreeBSD benchmarking results (continued)

Storage scheme	Filesystem	Mount options	Throughput	Response time	Hit ratio
ufs(2)	UFS	noatime	31.1	1.54	55.0
ufs(3)	UFS	async	30.2	1.51	55.9
ufs(4)	UFS	softupdate	29.9	1.51	55.7
ufs(5)	UFS		24.4	1.50	56.4

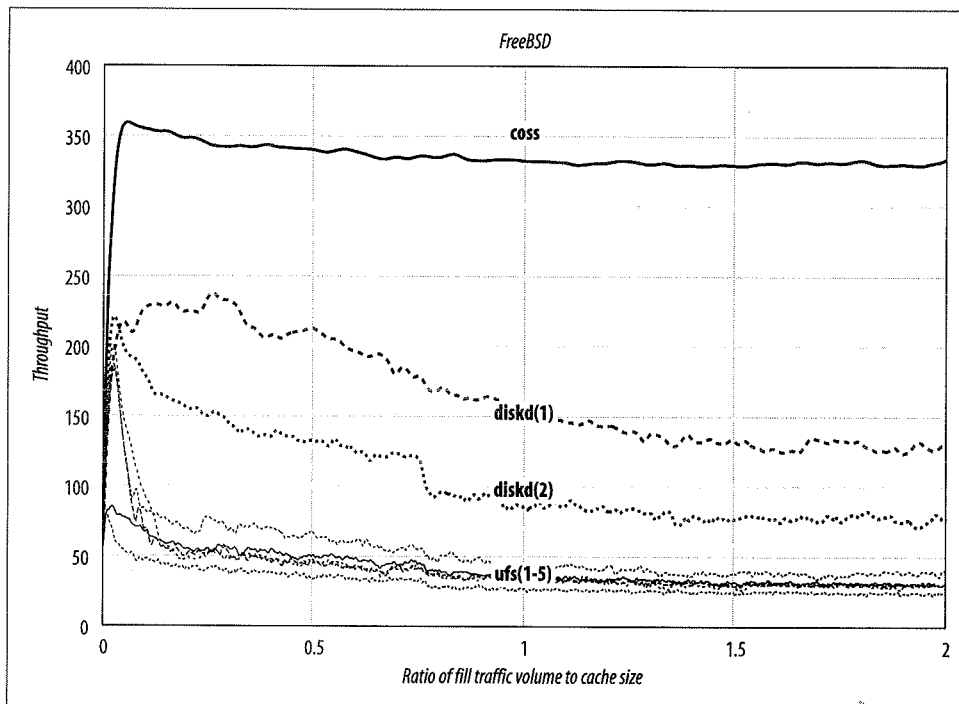


Figure D-2. FreeBSD filesystem benchmarking traces

Enabling the `async`, `noatime`, and `softupdate`* options boosts the standard `ufs` performance from 24 to 38 transactions per second. However, using one of the other storage schemes increases the sustainable throughput even more.

FreeBSD's `diskd` performance (129/sec) isn't quite as good as on Linux (169/sec), perhaps because the underlying filesystem (`ext2fs`) is better.

Note that the trace for `coss` is relatively flat. Its performance doesn't change much over time. Furthermore, both FreeBSD and Linux report similar throughput numbers: 326/sec and 331/sec. This leads me to believe that the disk system isn't a bottle-

* On FreeBSD, `softupdates` aren't a mount option, but must be set with the `tunefs` command.

neck in these tests. In fact, the test with no disk cache (see the section “Number of Disk Spindles”) achieves essentially the same throughput (332/sec).

OpenBSD

The results in this section are from OpenBSD Version 3.3 (released May 1, 2003). I built a kernel with the following notable configuration options:

```
option      MSGMNB=8192
option      MSGMNI=40
option      MSGSEG=512
option      MSGSSZ=64
option      MSGTQL=2048
option      SHMSEG=16
option      SHMMNI=32
option      SHMMAX=2048
option      SHMALL=4096
option      NMBCLUSTERS=32768
option      MAXFILES=8192
```

Table D-3 and Figure D-3 summarize the OpenBSD results. The choices for OpenBSD are similar to those for FreeBSD. Unfortunately, however, *coss* doesn’t run on OpenBSD, which lacks the `aio_read()` and `aio_write()` functions.

Table D-3. OpenBSD benchmarking results

Storage scheme	Filesystem	Mount options	Throughput	Response time	Hit ratio
<i>diskd(1)</i>	UFS	async, noatime, softupdate	91.1	1.45	56.3
<i>diskd(2)</i>	UFS		63.7	1.44	56.2
<i>ufs(1)</i>	UFS	softupdate	27.6	1.51	56.3
<i>ufs(2)</i>	UFS	noatime	25.1	1.52	56.3
<i>ufs(3)</i>	UFS		22.7	1.52	56.1
<i>ufs(4)</i>	UFS	async	22.1	1.51	56.6

In general, the OpenBSD results are slightly worse than FreeBSD. This isn’t too surprising, given that the OpenBSD project emphasizes security and perhaps spends less time on filesystem performance.

One odd result is that using the `async` option (alone) caused a slight decrease in performance for the *ufs* storage scheme.

NetBSD

These results come from NetBSD Version 1.6.1 (released April 21, 2003). Table D-4 and Figure D-4 summarize the NetBSD results. NetBSD actually performs almost the same as OpenBSD. The best configuration yields about 90 transactions per second.

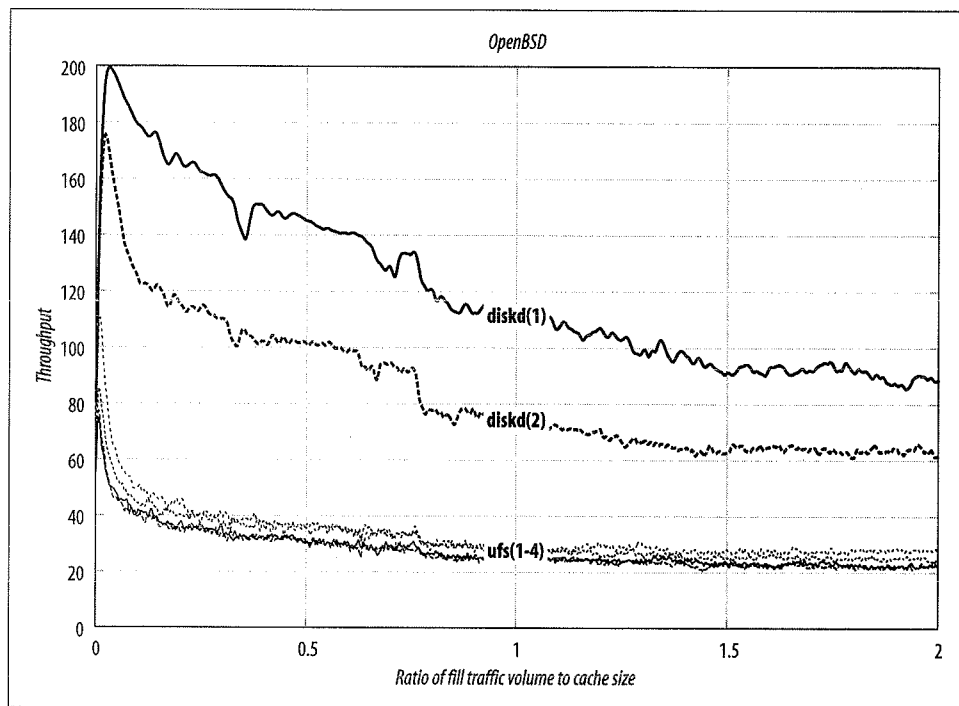


Figure D-3. OpenBSD filesystem benchmarking traces

Unfortunately, NetBSD doesn't support *coss* or *aufs*. I built a custom kernel with these options:

```
options      NMBCLUSTERS=32768
options      MAXFILES=8192
options      MSGSSZ=64
options      MSGSEG=512
options      MSGMNB=8192
options      MSGMNI=40
options      MSGTQL=2048
```

Table D-4. NetBSD benchmarking results

Storage scheme	Filesystem	Mount options	Throughput	Response time	Hit ratio
<i>diskd(1)</i>	UFS	softupdate,noatime,async	90.3	1.49	57.2
<i>diskd(2)</i>	UFS	softupdate	73.5	1.51	55.8
<i>diskd(3)</i>	UFS		60.1	1.48	55.9
<i>ufs(1)</i>	UFS	softupdate,noatime,async	34.9	1.51	56.2
<i>ufs(2)</i>	UFS	softupdate	31.7	1.52	55.5
<i>ufs(3)</i>	UFS		23.6	1.53	55.4

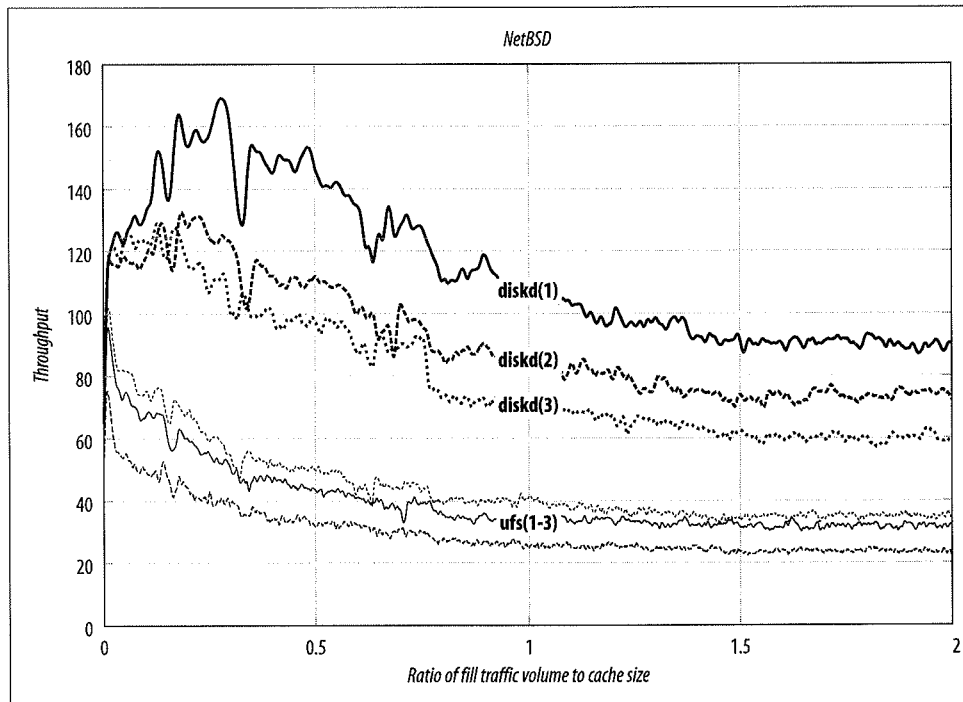


Figure D-4. NetBSD filesystem benchmarking traces

Solaris

These results come from Solaris Version 8 for Intel (released February 2002). Solaris 9 was available when I started these tests, but Sun no longer makes it freely available. I tweaked the kernel by adding these lines to `/etc/system`:

```
set rlim_fd_max = 8192
set msgsys:msginfo_msgmax=8192
set msgsys:msginfo_msgmnb=8192
set msgsys:msginfo_msgmni=40
set msgsys:msginfo_msgssz=64
set msgsys:msginfo_msgtql=2048
set shmsys:shminfo_shmmax=2097152
set shmsys:shminfo_shmmni=32
set shmsys:shminfo_shmseg=16
```

Table D-5 and Figure D-5 summarize the Solaris results. This is the only other operating system, in addition to Linux, in which the `aufs` storage scheme works well. Interestingly, both `aufs` and `diskd` have about the same performance on Solaris, although the actual numbers are much lower than on Linux.

Table D-5. Solaris benchmarking results

Storage scheme	Filesystem	Mount options	Throughput	Response time	Hit ratio
<i>diskd(1)</i>	UFS	noatime	56.3	1.53	55.7
<i>aufs(1)</i>	UFS	noatime	53.6	1.49	56.6
<i>diskd(2)</i>	UFS		37.9	1.53	55.5
<i>aufs(2)</i>	UFS		37.4	1.49	56.4
<i>coss</i>			32.4	1.47	54.6
<i>ufs(1)</i>	UFS	noatime	24.0	1.53	55.6
<i>ufs(2)</i>	UFS		19.0	1.50	56.3

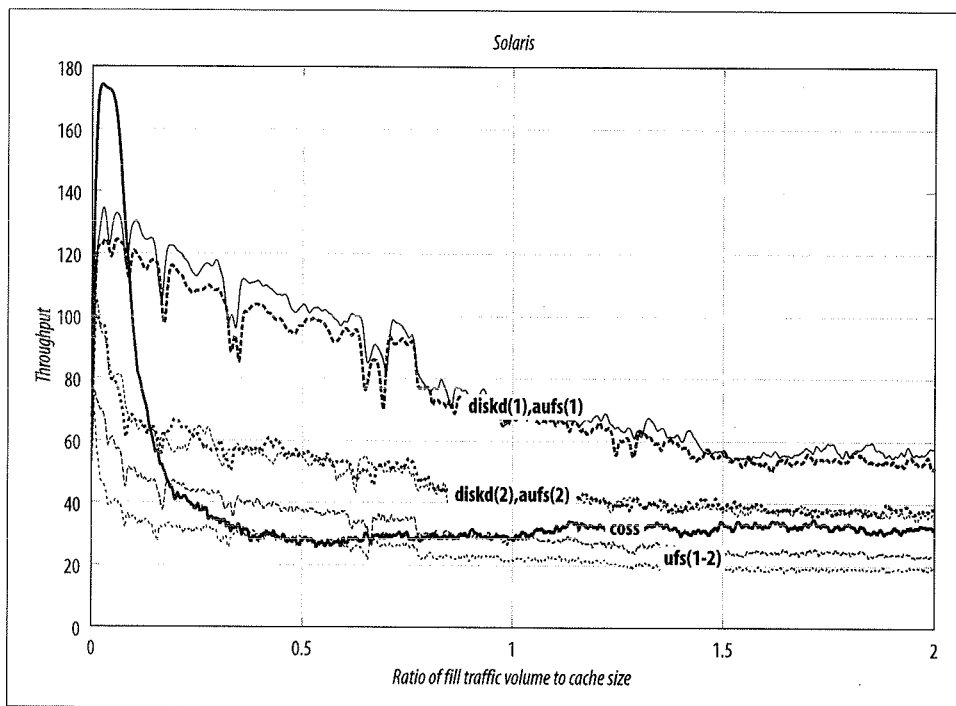


Figure D-5. Solaris filesystem benchmarking traces

Solaris also supports *coss*, but at nowhere near the rates for Linux and FreeBSD. For some unknown reason, *coss* on Solaris is limited to 32 transactions per second.

Number of Disk Spindles

In this section, I compare Squid's performance for different number of disk drives (spindles). These tests are from the Linux system with the *aufs* storage scheme and *ext2fs* filesystems.

Table D-6 and Figure D-6 summarize the results. The test with no disk drives has the best throughput, but the worst response time and hit ratio. Note that Squid does serve a few cache hits from memory, so the hit ratio isn't zero.

Table D-6. Comparison of 0–3 disk spindles on Linux with aufs

#Disks	Throughput	Response time	Hit ratio
0	332.1	2.99	0.4
3	109.6	1.44	56.2
2	85.3	1.49	53.9
1	66.0	1.50	53.5

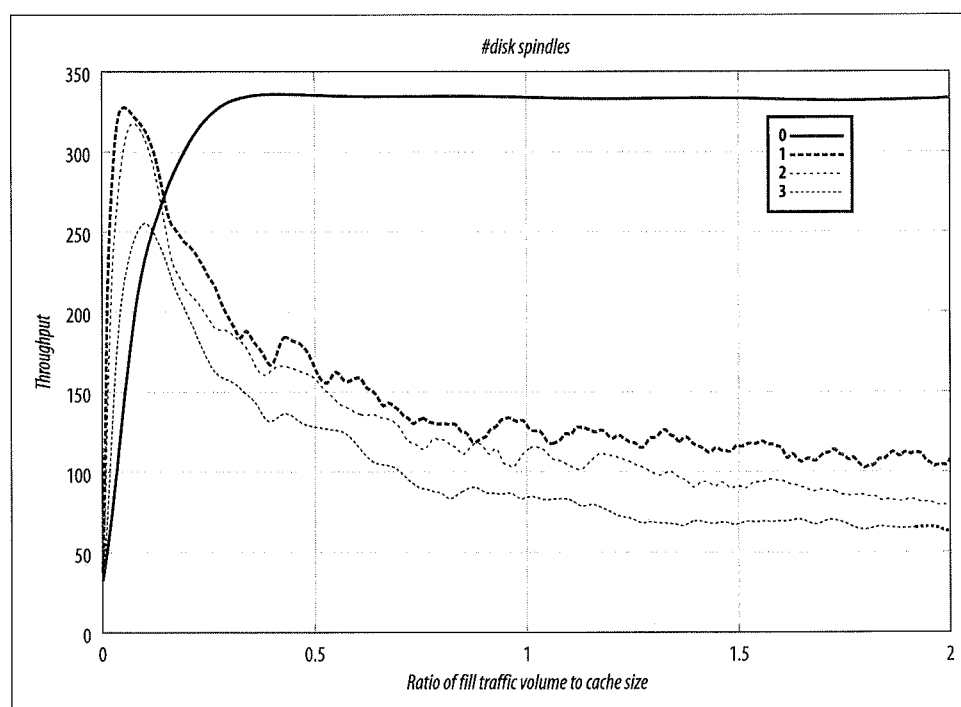


Figure D-6. Benchmarking results for 0, 1, 2, and 3 disk drives on Linux with aufs

The primary purpose of these tests is to show that Squid's performance doesn't increase in proportion to the number of disk drives. Excluding other factors, you may be able to get better performance from three systems with one disk drive each, rather than a single system with three drives.

APPENDIX E

Squid on Windows

Squid has been designed to run on Unix, but you can also get it to run on Microsoft Windows. Perhaps the easiest way is to use Red Hat's Cygwin emulation layer. It gives a Windows box everything it needs to run a variety of Unix applications. Another option is to use SquidNT. This is a version of the source code that has been modified to compile under a native Windows C compiler.

Cygwin

Cygwin is a Unix emulation package for Microsoft Windows. It provides an environment that allows you to build and run software primarily designed for Unix. You can also download and install a number of precompiled binary packages, including Squid.

Cygwin runs on Windows 95, 98, ME, NT, 2000, and XP. The Cygwin FAQ, however, makes this disclaimer:

Keep in mind that Cygwin can only do as much as the underlying OS supports. Because of this, Cygwin will behave differently, and exhibit different limitations, on the various versions of Windows.

When writing this appendix, I installed Cygwin Version 1.3.21 on Windows 2000.

Installing Cygwin

The first step is to install Cygwin on your Windows system. Visit the <http://www.cygwin.com/> site and click on the *Install Cygwin* link. After running Cygwin Setup, you'll have the base environment with a number of standard Unix tools. You might want to spend a little time playing with it to see how it works. Once you're comfortable with the Cygwin environment, decide if you'd like to use the precompiled package or compile Squid from its source.

The Squid Package

The Cygwin project provides a precompiled Squid binary. To download and install it, run the Cygwin Setup program again. When you see the *Select Packages* window, find the *Web* group and select *squid* for installation. Continue with the setup procedure as before.

When *Setup* completes, you should find the Squid binary at */usr/bin/squid* and the configuration file at */etc/squid.conf*.

Compiling Squid

You can also compile the Squid source code under Cygwin. This might be necessary if you want to run a more recent version than the precompiled binary available from the Cygwin site. To compile on Cygwin, you need to install at least the following packages:

- *Archive/sharutils*
- *Devel/make*
- *Devel/gcc*
- *Interpreters/Perl*

After installing those tools, you should be able to configure and compile Squid as described in Chapter 3.

Configuring and Running

Since Cygwin is essentially a Unix environment, you can run Squid as described throughout this book. Some special features may or may not work. For example, you won't be able to build certain authentication helpers without additional libraries and header files. Here are a few things to watch out for:

- The *cache_effective_user* directive is set to *nobody* by default. When you run Squid under Cygwin, you may get an error that the *nobody* doesn't exist. You can either create that user or set *cache_effective_user* to a username that does exist.
- Cygwin doesn't have a */etc/resolv.conf* by default, and Squid won't pick up your DNS server settings from the Windows registry. You can either create a fake */etc/resolv.conf* or list your name server addresses in *squid.conf* with a *dns_nameserver* directive.

SquidNT

Guido Serassio is maintaining a project called SquidNT. It is branch of Squid's development tree that contains changes necessary for a native port of Squid to Windows NT, 2000, XP, and 2003. In other words, you can compile and run this version of

Squid on Windows without any Unix emulation libraries. The code is known to compile with Microsoft's Visual C++ 6.0 compiler and under the MSYS+MinGW environment. Guido also provides some precompiled SquidNT binaries. You can find his work and more information on SquidNT by visiting <http://www.serassio.it/SquidNT.htm>.

Configuring Squid Clients

This appendix contains information on setting up various browsers and user-agents to use Squid. Although it is more extensively covered in my O'Reilly book *Web Caching*, I'll include some brief instructions here.

I have instructions for the following HTTP user-agents: Internet Explorer v6, Konqueror v3, Lynx v2.8, Netscape v7 a.k.a. Mozilla v5, Opera v7, libwww-perl v5, Python's urllib/urllib2, and Wget v1.8. If you think this is all a huge hassle, consider using HTTP interception, as described in Chapter 9.

Manually

Web browsers and other HTTP-based user-agents have methods for explicitly setting a proxy address. For large organizations, this is a real hassle. You may simply have too many desktops to visit one at a time. Additionally, this approach isn't as flexible as the others. For example, you can't temporarily stop the flow of requests to the proxy or easily bypass the cache for certain troublesome sites.

Browsers usually give you the option to send HTTPS URLs to a proxy. Squid can handle HTTPS requests, although it can't cache the responses. Squid simply tunnels the encrypted traffic. Thus, you should configure the browser to proxy HTTPS requests only if your firewall prevents direct connections to secure sites.

Netscape/Mozilla

To manually configure proxies with Netscape and Mozilla, follow this sequence of menus:

- *Edit*
- *Preferences*
- *Advanced*
- *Proxies*

- *Manual proxy configuration*
- Fill in the *HTTP Proxy* address and *Port* fields. Enter the same values for *FTP Proxy* if you like.

Explorer

To manually configure proxies in Internet Explorer, select the following sequence of menus:

- *View* from the main window menu
- *Internet Options*
- *Connections* tab
- *LAN Settings*
- Enable *Use a proxy server* and enter its address in the *Address* and *Port* fields

The *Advanced* button opens a new window in which you can enter different proxy addresses for different protocols (HTTP, FTP, etc.).

Konqueror

You can manually configure proxies in Konqueror by clicking on the following sequence of menus:

- *Settings*
- *Configure Konqueror*
- *Proxies & Cache*
- *Use Proxy*
- Fill in the address for *HTTP Proxy*, and *Port*. Use the same values for other protocols if you like.

Opera

Here's how to find the proxy configuration screen in Opera browsers:

- *File*
- *Preferences*
- *Network*
- *Proxy Servers*
- Enter an IP address (or hostname) and port number for HTTP, FTP, and other protocols as necessary.

Lynx

The Lynx browser uses a configuration file, typically `/usr/local/etc/lynx.cfg`. There you'll find a number of settings for proxies. For example:

```
http_proxy:http://proxy.example.com:3128/  
https_proxy:http://proxy.example.com:3128/  
ftp_proxy:http://proxy.example.com:3128/
```

Lynx also accepts proxy configuration via environment variables, as described in the next section.

Environment Variables

Some browsers and other user-agents look for proxy settings in environment variables. Note that the variable names are lowercase, unlike most environment variable names:

```
csh% setenv http_proxy http://proxy.example.com:3128/  
csh% setenv ftp_proxy http://proxy.example.com:3128/  
  
sh$ http_proxy=http://proxy.example.com:3128/  
sh$ ftp_proxy=http://proxy.example.com:3128/  
export http_proxy ftp_proxy
```

I've convinced myself that the following products and packages check for these environment variables:

- Opera
- Lynx
- Wget
- Python's `urllib` and `urllib2`
- `libwww-perl`

Proxy Auto-Configuration

Proxy Auto-Configuration is a technique that allows more control over the way user-agents select a proxy. The configuration file is simply a text file containing a JavaScript function. Browsers download the configuration file when they start up and then evaluate the function before each request. The function's return value determines where the request is sent.

Proxy Auto-Configuration is attractive because it gives the network administrator more control. For example, you can temporarily disable your caching service, implement load balancing, or migrate the service to new systems. Additionally, the function can return a list of proxy addresses, which the browser tries in sequence. If the first is unavailable, it tries the second, and so on.

The following browsers support Proxy Auto-Configuration:

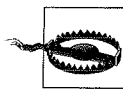
- Internet Explorer
- Opera
- Netscape
- Konqueror
- Mozilla

All these browsers have a place in which you can type in the Proxy Auto-Configuration URL. You'll find it in the same place as the manual proxy settings, earlier described in the section "Manually." Configuring hundreds or thousands of workstations is a real hassle, which is why a handful of companies came up with WPAD, described in the next section.

Writing a Proxy Auto-Configuration function is relatively straightforward. The function, named `FindProxyForURL`, takes two arguments and returns a list of proxy addresses, separated by semicolons. The word `DIRECT` instructs the browser to forward the request directly to the origin server, rather than to a proxy. Here is a simple example:

```
function FindProxyForURL(url, host) {  
    if (isPlainHostName(host))  
        return "DIRECT";  
    if (!isResolvable(host))  
        return "DIRECT";  
    if (url.substring(0, 5) == "http:")  
        return "PROXY 172.16.5.1:3128; DIRECT";  
    if (url.substring(0, 4) == "ftp:")  
        return "PROXY 172.16.5.1:3128; DIRECT";  
    return "DIRECT";  
}
```

The first `if` statement makes the browser connect directly to the origin server if the user types a single-component hostname, such as *www*. This is generally a good idea because the browser's interpretation of the hostname might be different from the proxy's. The second `if` statement ensures that the hostname exists in the DNS. If not, the user sees an error message from the browser itself, rather than from Squid. The next two `if` statements return a proxy address, followed by `DIRECT` for HTTP and FTP URLs. If the proxy doesn't respond, the browser attempts to make a direct connection to the origin server.



If you have a firewall in place, the browser probably won't be able to make a direct connection.

After writing the function, save it somewhere in your web server's data directory. Next, you need to configure the server to return a specific content type for the file.

The convention is to give the file a *.pac* extension, such as *proxy.pac*. Then, ensure that the HTTP server returns the content type `application/x-ns-proxy-autoconfig`. With Apache, you can add this line to your server config file:

```
AddType application/x-ns-proxy-autoconfig .pac
```

Refer to Section 4.3 of *Web Caching* (O'Reilly), for more information on Proxy Auto-Configuration files, including more complicated `FindProxyForURL` ideas and examples.

WPAD

The Web Proxy Auto Discovery (WPAD) protocol is a technique for user-agents to find a nearby caching proxy automatically. The idea is relatively simple. The protocol provides a number of methods for generating a URL that refers to a Proxy Auto-Configuration file. Those methods include DHCP, DNS lookups, and SLP (the Service Location Protocol).

DHCP is the first method the user-agent should try. It sends a query for “option 252” to a local DHCP server. The response is a string: the URL. Here’s how to configure ISC’s DHCP server for WPAD:

```
option wpad code 252 = text;  
option wpad "http://172.16.1.1/proxy.pac";
```

The second method is SLP. However, its implementation is optional. I do not know if any user-agents actually support WPAD via SLP.

DNS is the last resort. The protocol specification outlines a number of DNS techniques a user-agent might use to find a *wpad.dat* URL. The most straightforward technique is to perform an address lookup for the hostname *wpad* in the local domain. For example, if the system’s hostname is *orion.example.com*, the agent requests the IP address of *wpad.example.com*. If the lookup is successful, the agent makes a TCP connect to that address on port 80 and requests */wpad.dat*.

To make this work in Apache, you need to set the content type for the *wpad.dat* file like this:

```
AddType application/x-ns-proxy-autoconfig .dat
```

This may have negative side effects if your server has other files that end with *.dat*. One trick some people use is to redirect requests for *wpad.dat* to *proxy.pac*, with commands like this in *httpd.conf*:

```
Redirect /wpad.dat http://wpad.example.com/proxy.pac
```

Note that you probably won’t be able to set up a separate virtual host for the *wpad* name in your domain. This is because some user-agents set the `Host` header to the IP address, rather than the hostname. The following is an example.

```
GET /wpad.dat HTTP/1.1
Accept: */*
User-Agent: Mozilla/4.0 (compatible; MSIE 6.0; Win32)
Host: 206.168.0.13
```

WPAD is enabled by default in Microsoft Internet Explorer. Konqueror also supports WPAD but disables it by default. You can enable WPAD in Konqueror by visiting the proxy configuration page (described in the “Konqueror” section) and selecting *Auto Configure Proxy*. Although the current stable versions of Netscape (v7.02) and Mozilla (v5.0) don’t implement WPAD, future versions will.

Summary

Table F-1 summarizes the various proxy configuration options for the user-agents mentioned in this appendix.

Table F-1. Proxy configuration techniques for popular user-agents

User agent	Manual	Environment	PAC	WPAD
Explorer	Yes	No	Yes	Yes
Konqueror	Yes	No	Yes	Yes
libwww-perl	N/A	Yes	No	No
Lynx	Yes	Yes	No	No
Netscape/Mozilla	Yes	No	Yes	No
Opera	Yes	Yes	Yes	No
Wget	N/A	Yes	No	No

Index

Numbers

5min, cache manager page, 270
60min, cache manager page, 272

A

-a option, command-line, 43
access controls, 54
 cache manager, 288
 client blocking, 85
 debugging, 318
 delays due to, 83
 ICP queries, 89
 IP addresses, request denial and, 89
 local clients, 85
 non-HTTP servers and, 87
 pornography denials, 85
 redirectors and, 185
 rules
 checks, 84
 matching, 81
 syntax, 80
 server acceleration, 310
 squid.conf file, 39
 surrogate mode and, 310
 syntax, 82
 testing, 91
 usage restriction, 86
 user access, 88
access.log, 39, 211, 215–227
 Calamaris, 227
 configuration directives, 224–227
 fields, 215
 HTTP response field, status codes, 220

 peering codes, 222
 result codes, 218
 Squeezer, 227
 store.log comparison, 228
 Webalyzer, 227
acl directive, 54, 365
ACL elements
 arp type, 72
 AS numbers, 59
 browser type, 73
 domain names, 57
 dst type, 61
 dst_as type, 70
 dstdomain type, 62
 dstdom_regex type, 72
 external, 74
 authentication helpers and, 207
 writing, 208
 ident type, 68
 ident_regex type, 74
 IP addresses, 55
 long lists, 76
 matching, 77
 maxconn type, 71
 method type, 65
 myip type, 61
 myport type, 65
 port type, 63
 proto type, 66
 proxy_auth type, 69
 proxy_auth_regex type, 74
 regular expressions, 59
 reply_body_max_size, 80
 req_mime_type, 74

We'd like to hear your suggestions for improving our indexes. Send email to index@oreilly.com.

- ACL elements (*continued*)
 - snmp_community type, 71
 - src type, 60
 - src_as type, 70
 - srcdomain type, 63
 - srcdom_regex type, 72
 - syntax, 54
 - TCP port numbers, 59
 - usernames, 58
 - values, 54
- ACL rules, 78
- ACLs (access control lists)
 - always_direct, 79
 - broken_posts, 79
 - cachemgr.cgi rule, 289
 - cachemgr_passwd rule, 288
 - cache_peer_access, 80
 - delay_access, 80
 - elements, 54
 - header_access, 80
 - header_replace, 80
 - http_access, 78
 - http_access rule, cache manager and, 288
 - http_reply_access rule, 78
 - example, 90
 - icp_access, 78
 - ident_lookup_access, 79
 - miss_access, 79
 - never_direct, 79
 - no_cache, 78
 - redirector_access, 79
 - rep_mime_type type, 74
 - snmp_access, 79
 - tcp_outgoing_address, 80
 - tcp_outgoing_tos, 80
 - time type, 67
 - urlpath_regex type, 73
 - url_regex type, 73
- action field, store.log, 229
- active_requests, cache manager page, 273
- Address already in use message, 316
- addresses, ACLs and, 55
- administrator contact information, 42
- AdZapper redirector, 193
- Alteon/Nortel, interception caching and, 131
- always_direct ACL, 79
- always_direct directive, 380
 - neighbor caches, 161
- announce_file directive, 371
- announce_host directive, 371
- announce_period directive, 370
- announce_port directive, 371
- APIs
 - Basic Auth API, 200
 - Digest authentication, 203
 - NTLM authentication, 206
- append_domain directive, 373
- application layer, transport layer and, 405
- application-layer routing, 153
- arp ACL type, 72
- arrowpoint (Cisco), interception caching and, 136
- AS (Autonomous System) numbers,
 - ACLs, 59
- asndb, cache manager page, 286
- assertions, debugging and, 323–326
- as_whois_server directive, 384
- aufs storage scheme, 112, 115
 - issues with, 113
 - monitoring, 114
- authenticate_cache_garbage_interval directive, 358
- authenticate_ip_ttl directive, 195, 358
- authenticate_ttl directive, 357
- authentication, 194
 - Basic, 194, 195
 - Basic Auth API, 200
 - Digest, 194
 - Digest API, 203
 - helpers, 194
 - configuration, 195
 - external ACLs and, 207
 - getpwnam (Basic authentication), 196, 200
 - LDAP (Basic authentication), 197
 - MSNT (Basic authentication), 197
 - multi-domain-NTLM (Basic authentication), 198
 - NCSA (Basic authentication), 197
 - PAM (Basic authentication), 198
 - SASL (Basic authentication), 199
 - SMB (Basic authentication), 199
 - SMB (NTLM authentication), 205
 - winbind (Basic authentication), 200
 - winbind (NTLM authentication), 205
 - YP (Basic authentication), 200
 - HTTP Digest, 201
 - NTLM, 194, 204
 - API, 206
 - proxy authentication, directives, 195
- auth_param directive, 195, 354–357
 - arguments, 195
 - Basic authentication, parameters supported, 196

- Digest authentication, parameters supported, 201
- NTLM authentication support, parameters supported, 204

B

- backend servers
 - boxes, surrogate configuration and, 308
 - content negotiation, 311
 - IP addresses and, 303
 - server acceleration and, 302
- bandwidth
 - application layer, 405
 - transport layer, 405
- bandwidth buckets, 401
- Basic authentication, 194, 195
 - auth_param directive, parameters supported, 196
 - Basic Auth API, 200
 - getpwnam helper, 200
 - LDAP helper, 197
 - MSNT helper, 197
 - multi-domain NTLM helper, 198
 - NCSA helper, 197
 - PAM helper, 198
 - SASL helper, 199
 - SMB helper, 199
 - winbind helper, 200
 - YP helper, 200
- basicauthenticator, cache manager page, 250
- benchmarks
 - filesystem
 - disk spindles, 416
 - FreeBSD, 410
 - Linux, 410
 - NetBSD, 413
 - Solaris, 415
 - filesystem performance
 - configuration, 408
 - hardware, 407
 - Squid versions, 408
 - I/O bottleneck and, 109
- blocking clients, access controls and, 85
- Bloom filters, Cache Digests, 172
- boot scripts, 48
 - /etc/inittab scheme, 50
 - /etc/rc.local, 49
 - init.d scheme, 49
 - rc.d scheme, 49
- broken_posts ACL, 79
- broken_posts directive, 390
- browser ACL type, 73

- BSD-based code, mbufs and, 17
- buffered I/O, redirectors, 187
- buffered_logs directive, 379
 - access.log and, 227
- bus errors, debugging, 323–326
- byte hit ratio, 2

C

- C option, command-line, 44
- cache
 - hits on local sites, access controls and, 90
 - Squid as, 1
- Cache Digests, 152, 172
 - Bloom filter, 172
 - configuration for, 173
- cache directories, initialization, 46
- cache hierarchies, 152
 - forwarding loops, 154
 - neighbor caches, cache_peer directive, 155–159
- cache hit ratio, 2
- cache hits, 2
- cache key field, store.log, 229
- cache manager, 237, 239
 - access controls, 288
 - disadvantages, 289
 - http_access ACL rule, 288
 - squidclient utility and, 240
 - Squid-RRD and, 290
- cache manager pages, 241
 - 5min, 270
 - 60min, 272
 - active_requests, 273
 - asndb, 286
 - basicauthenticator, 250
 - carp, 286
 - cbdata, 244
 - client_list, 284
 - comm_incoming, 245
 - config, 245
 - counters, 266–270
 - delay, 283
 - digestauthenticator, 250
 - digest_stats, 270
 - diskd, 245
 - dns, 247
 - events, 244
 - external_acl, 250
 - filedescriptors, 262
 - forward, 283
 - forw_headers, 253
 - fqdn_cache, 247

- cache manager pages (*continued*)
 - histograms, 273
 - http_headers, 250
 - info, 254–262
 - io, 265
 - ipcache, 246
 - leaks, 243
 - mem, 243
 - menu, 253
 - netdb, 285
 - non_peers, 288
 - ntlmauthenticator, 250
 - objects, 264
 - offline_toggle, 254
 - openfd_objects, 265
 - pconn, 280
 - peer_select, 270
 - redirector, 250
 - refresh, 280
 - server_list, 286
 - shutdown, 254
 - squid aio_counts, 244
 - store_check_cachable_stats, 278
 - store_digest, 275
 - storedir, 276
 - store_io, 279
 - utilitization, 273
 - via_headers, 252
 - vm_objects, 264
- cache misses, 2
- cache validation, 3
- cache_access_log directive, 340
- cache_dir directive, 94, 342
 - options, 98
- cache_dns_program directive, 349
- cache_effective_group directive, 369
- cache_effective_user directive, 369
- cache.log, 39, 211, 212–215, 237
 - debugging and, 213, 319–323
 - messages
 - system log, 214
 - terminal, 214
 - warnings, 237
 - hardcoded, 238
 - page fault excesses, 238
 - process size, 238
 - response time, 238
- cache_log directive, 340
- cache_mem directive, 342, 399
- cache_mgr directive, 368
 - squid.conf (administrator contact), 42
- cachemgr.cgi ACL rule, 289
- cachemgr_passwd directive, 377
- cache_peer directive, 334
 - neighbor caches and, 155–159
 - no-delay option, 404
 - options, 155–158
- cache_peer_access directive, 80, 366
 - loops and, 154
 - neighbor cache access list, 159
- cache_peer_domain directive, 336
 - neighbor cache access, 160
- cache_replacement_policy directive, 101, 344
- caches
 - memory cache, 399
 - retrying requests, 181
 - Web caching, 2
- cache_store_log directive, 341
- cache_swap_high directive, 99, 343
- cache_swap_log directive, 341
- cache_swap_low directive, 342
 - disk cache, 99
- Calamaris, 227
- Cannot determine fully qualified hostname, debugging and, 316
- CARP (Cache Array Routing Protocol), 152, 176
 - configuration for, 177
- carp, cache manager page, 286
- case sensitivity, squid.conf file, 36
- cbdata page, cache manager pages, 244
- CDN (content delivery network), 302
- children parameter, auth_param directive, 196
- chroot directive, 394
- chroot environment, 50
- Cisco
 - arrowpoint, interception caching and, 136
 - policy routing, interception caching and, 137
 - WCCP, interception caching and, 138
- client address field
 - access.log, 216
 - referer.log, 232
 - useragent.log, 233
- client identity field, access.log, 217
- client_db directive, 378
- client_lifetime directive, 364
- client_list, cache manager page, 284
- client_netmask directive, 348
 - access.log and, 226
- client_persistent_connections directive, 395

- clients
 - blocking, access controls and, 85
 - configuration, manual, 421–423
 - local, access controls, 85
- client-side of Squid, 1
- command-line options, 43
 - a, 43
 - C, 44
 - D, 44
 - d, 44
 - F, 45
 - f, 44
 - h, 44
 - k, 44
 - N, 45
 - R, 45
 - s, 44
 - u, 44
 - V, 45
 - v, 44
 - X, 45
 - Y, 45
 - z, 44
- commands
 - make, ./configure script, 27
 - squid -k shutdown, 51
- comm_incoming, cache manager page, 245
- compiling
 - ./configure script, 18
 - options, 19–26
 - coss, 120
 - diskd, 116
 - file descriptors, 15
 - installing programs, 29
 - kernel and, 14
 - preparations, 13
 - unpacking source, 14
- complete objects, memory, 399
- config, cache manager page, 245
- configuration
 - authentication helpers, 195
 - automatic, Proxy
 - Auto-Configuration, 423
 - Cache Digests and, 173
 - CARP and, 177
 - coss storage scheme, 120
 - delay pools, 402
 - devices, interception caching
 - and, 130–141
 - directives, access.log and, 224–227
 - diskd, 116
 - filesystem benchmarks, 408
 - HTCP and, 175
 - interception caching and, 146
 - WCCP, 147
 - redirectors, 190
 - running processes, reconfiguring, 52
 - surrogate mode and, 304
 - (see also squid.conf)
 - ./configure script, 18
 - make command, 27
 - options, 19–26
 - rerunning, 33
 - running, 26
 - connect_timeout directive, 362
 - contact info for administrator, 42
 - content routers, 153
 - Squid as, 152
 - content type field, access.log, 218
 - content, uncacheable, 313
 - content-length/size field, store.log, 230
 - content-type field, store.log, 230
 - controls (see access controls)
 - core dumps, debugging and, 323–326
 - coredump_dir directive, 392
 - coss (Cyclic Object Storage Scheme), 119–122
 - counters, cache manager page, 266–270
 - credentialsttl parameter, auth_param
 - directive, 196
 - CVS (Concurrent Versioning System), 11
 - Cygwin, 418

D

- D option, command-line, 44
- d option, command-line, 44
- daemon processes, Squid as, 47
 - squid_start script, 48
- date field, store.log, 230
- dead_peer_timeout directive, 339
- debugging
 - access controls, 318
 - Address already in use, 316
 - assertions and, 323–326
 - bus errors and, 323–326
 - cache.log, 213
 - cache.log and, 319–323
 - core dumps and, 323–326
 - DNS name lookup and, 316
 - filedescriptors and, 317
 - fully qualified hostname message, 316
 - hostnames and, 317
 - HTTP interception, 148
 - icmpRecv and, 318

- debugging (*continued*)
 - replicating problems and, 326–328
 - reporting bugs, 329
 - segmentation violations and, 323–326
 - stack traces and, 323–326
 - swap directory error, 315
 - system speed, 318
 - debug_options directive, 347
 - cache.log, 213
 - delay, cache manager page, 283
 - delay pools
 - configuration, 402
 - monitoring, 406
 - overview, 401
 - subnetting scheme and, 406
 - delay_access ACL, 80
 - delay_access directive, 386, 403
 - delay_class directive, 386, 402
 - delay_initial_bucket_level directive, 387, 403
 - delay_parameters directive, 386, 403
 - delay_pools directive, 385, 402
 - deny_info directive, 374
 - DEVEL releases, 9
 - developers, devel.squid-cache.org site, 12
 - devices, interception caching and, 130–141
 - diffs, applying, 32
 - Digest authentication, 194
 - API, 203
 - auth_param directive, parameters supported, 201
 - digestauthenticator, cache manager page, 250
 - digest_bits_per_entry directive, 393
 - digest_generation directive, 393
 - digest_rebuild_chunk_percentage directive, 394
 - digest_rebuild_period directive, 393
 - digest_rewrite_period directive, 394
 - digest_stats, cache manager page, 270
 - digest_swapout_chunk_size directive, 394
 - direct options, 178
 - directories
 - cache directories, initialization, 46
 - disk cache, object allocation, 100
 - directory argument, cache_dir directive, 94
 - directory number field, store.log, 229
 - disk cache
 - cache_dir directive, 94
 - cache_replacement_policy directive, 101
 - cache_swap_high directive, 99
 - cache_swap_low directive, 99
 - directories, object allocation, 100
 - I/O bottleneck, 108
 - object removal, 102–104
 - object size, 99
 - refresh_pattern directive, 104
 - replacement policy, 101
 - usage, 99
 - disk space, process size and, 96
 - disk spindles, benchmarks and, 416
 - diskd, cache manager page, 245
 - diskd storage scheme, 115–119
 - diskd_program directive, 352
 - dns, cache manager page, 247
 - DNS name lookup tests failed message, debugging and, 316
 - dns_children directive, 350
 - dns_defnames directive, 351
 - dns_nameservers directive, 351
 - dns_retransmit_interval directive, 350
 - dns_testnames directive, 373
 - dns_timeout directive, 351
 - domain names, ACLs, 57
 - dst ACL type, 61
 - dst_as ACL type, 70
 - dstdomain ACL type, 62
 - dstdom_regex ACL type, 72
- ## E
- emulate_httpd_log directive, 341
 - access.log and, 224
 - environment variables, proxy manual configuration, 423
 - environments, chroot, 50
 - ephemeral ports, 17
 - err_html_text directive, 374
 - error checking, squid.conf file, 45
 - error messages, surrogate mode and, 313
 - error_directory directive, 382
 - ESI (Edge Side Includes), 4
 - /etc/inittab scheme, 50
 - /etc/rc.local script, 49
 - events page, cache manager pages, 244
 - expires field, store.log, 230
 - Explorer, manual configuration, 422
 - ext2fs, 93
 - extension_methods directive, 395
 - external ACLs, 74
 - authentication helpers and, 207
 - writing, 208
 - external_acl, cache manager page, 250

external_acl_type directive, 359
Extreme Networks, interception caching
and, 135

F

-F option, command-line, 45
-f option, command-line, 44
false hits, sibling caches, 154
FAQ document, 6
FFS (Fast File System), 93
fields, access.log, 215
file descriptors
 compiling and, 15
 FreeBSD and, 16
 Linux and, 16
 NetBSD and, 16
 OpenBSD and, 16
 Solaris and, 17
file number field, store.log, 229
file numbers, mapping to pathnames, 231
filedescriptors
 cache manager page, 262
 debugging and, 317
filesystems, 93
 alternative, 111
 aufs storage scheme, 112–115
 benchmarks
 configuration, 408
 disk spindles, 416
 FreeBSD, 410
 hardware, 407
 Linux, 410
 NetBSD, 413
 OpenBSD, 413
 Solaris, 415
 Squid versions, 408
 coss (Cyclic Object Storage
 Scheme), 119–122
 disk cache, object size, 99
 disk space, process size and, 96
 disk usage, 99
 diskd storage scheme, 115–119
 ext2fs, 93
 FFS, 93
 inodes, 95
 I/O bottleneck, 108
 journaling systems, 111
 null storage scheme, 122
 soft updates, 110
 storage schemes, 93
 system calls, 93

 tuning options, 110–111
 UFS, 93
filters, redirectors and, 185
forward, cache manager pages, 283
forwarded_for directive, 376
forwarding loops, cache hierarchies, 154
forw_headers, cache manager page, 253
Foundry, interception caching and, 134
FQDN (fully qualified domain name), 186
fqdn_cache, cache manager page, 247
fqdn_cache_size directive, 346
FreeBSD
 file descriptors and, 16
 filesystem benchmarks, 410
 interception caching and, 143
FTP (File Transfer Protocol) servers, 2
ftp_list_width directive, 348
ftp_passive directive, 349
ftp_sanitycheck directive, 349
ftp_user directive, 348
fully qualified hostname, debugging, 316

G

GDSF (greedy dual-size frequency)
 replacement policy, 101
getpwnam authentication helper (Basic
 authentication), 200
getrusage() function, 238
GNU (General Public License), 5
Gopher servers, 2
GRE (Generic Routing Encapsulation),
 interception caching and, 138
groups, 37

H

-h option, command-line, 44
half_closed_clients directive, 364
hardcoded warnings, cache.log, 238
hardware, filesystem benchmarks, 407
hardware requirements, 4
header_access ACL, 80, 381
header_replace ACL, 80, 382
health checks, interception caching and, 137
hierarchies
 cache hierarchies, 152
 surrogate mode and, 314
hierarchy_stoplist directive, 339
 neighbor caches, 161
high_memory_warning directive, 238, 397
high_page_fault_warning directive, 238, 396

- high_response_time_warning directive, 238, 396
 - histograms, cache manager page, 273
 - hostname, debugging and, 317
 - hostname_aliases directive, 370
 - hosts_file directive, 351
 - hot object cache, 399
 - HTCP (Hypertext Caching Protocol), 152, 333
 - configuring for, 175
 - ICP and, 174
 - reply processing, 180
 - httpd_port directive, 333
 - HTTP (Hypertext Transfer Protocol)
 - Basic authentication, 195
 - auth_param directive parameters, 196
 - Digest authentication, 201
 - interception, debugging, 148
 - proxy authentication, interception
 - caching and, 128
 - redirect messages, 187
 - servers, 2
 - servers, interception caching and, 137
 - HTTP interception, 124, 303
 - layer four switches and, 131
 - HTTP request headers field, access.log, 218
 - HTTP response field, access.log, status codes, 220
 - HTTP response headers field, access.log, 218
 - HTTP servers, access controls and, 87
 - http_access ACL rule, cache manager and, 288
 - http_access ACL type, 78
 - http_access directive, 365
 - httpd_accel_host directive, 305, 371
 - httpd_accel_port directive, 306, 372
 - httpd_accel_single_host directive, 307, 372
 - httpd_accel_uses_host_header directive, 306, 373
 - httpd_accel_with_proxy directive, 308, 372
 - http_headers, cache manager page, 250
 - http_port directive, 38, 304, 331
 - http_reply_access ACL, 78
 - http_reply_access ACL rule, example, 90
 - http_reply_access directive, 366
 - https_port directive, 304, 331
- I**
- ICAP (Internet Content Adaptation Protocol), 4
 - icmpRecv, debugging and, 318
 - icon_directory directive, 382
 - ICP (Internet Cache Protocol), 152, 164
 - clients
 - cache_peer directive, 169
 - icp_port directive, 168
 - multicast ICP, 171
 - HTCP and, 174
 - multicast, 170
 - clients, 171
 - example, 172
 - servers, 170
 - neighbor caches and, 166
 - netdb and, 169
 - reply processing, 180
 - requests, 165
 - server, Squid as, 165
 - stale responses, icp_hit_stale directive, 167
 - ICP queries, access controls and, 89
 - icp_access ACL, 78
 - icp_access directive, 366
 - icp_hit_stale directive, 167, 376
 - ICP_MISS_NOFETCH, 168
 - icp_port directive, 168, 332
 - Squid as ICP server, 165
 - icp_query_timeout directive, 337, 338
 - ident ACL elements, 186
 - ident ACL type, 68
 - ident_lookup_access directive, 79, 186, 367
 - access.log and, 225
 - ident_regex ACL type, 74
 - ident_timeout directive, 365
 - ie_refresh directive, 397
 - ignore_unknown_nameservers directive, 392
 - incoming_dns_average directive, 388
 - incoming_http_average directive, 387
 - incoming_icp_average directive, 387
 - info, cache manager page, 254–262
 - init.d scheme, boot scripts, 49
 - initializing cache directories, 46
 - inodes, 93, 95
 - installation
 - compiled programs, 29
 - Cygwin, 418
 - pinger program, 30
 - intercache communication, 152
 - interception caching, 124
 - benefits and disadvantages, 127
 - configuration and, 146
 - device configuration, 130–141
 - FreeBSD and, 143
 - HTTP, 124

- debugging, 148
- proxy authentication and, 128
- IPFilter, NetBSD, 146
- layer four switches and, 131
- Linux systems, 141
- OpenBSD, 145
- operating systems and, 141–146
- in-transit objects, memory, 399
- I/O
 - aufs storage system and, 112
 - bottleneck, 108
 - buffered I/O, redirectors and, 187
- io, cache manager page, 265
- IP addresses
 - access controls and, request denial and, 89
 - ACLs, 55
- IP packet filtering software, 124
- ipcache, cache manager page, 246
- ipcache_high directive, 345
- ipcache_low directive, 345
- ipcache_size directive, 345
- IPFilter, 124
 - interception caching, NetBSD, 146
- ipfw (filtering software), 124
- iptables (filtering software), 124
- IRCache (Information Resource Caching), 3

J

- Jesred redirector, 192
- journaling filesystems, 111

K

- k option, command-line, 44
- kernel
 - compiling and, 14
 - precompiled binaries and, 15
- Konqueror, manual configuration, 422

L

- L1 argument, cach_dir directive, 96
- L2 argument, cache_dir directive, 96
- last-modified field, store.log, 230
- last-modified timestamps, 3
- layer four switches, HTTP interception and, 131
- LDAP authentication helper (Basic authentication), 197
- leaks, cache manager page, 243
- libraries, shared, 10

Linux

- file descriptors and, 16
- filesystem benchmarks, 410
- interception caching and, 141
- local clients, access controls, 85
- local site cache hits, preventing with access controls, 90

log files

- access.log, 39, 211, 215–227
- cache.log, 39, 211, 212–215
- netdb_state file, 211
- pathnames, 38
- privacy issues, 236
- referer.log, 211, 231
- rotating, 53, 235
- security, 236
- storage space, 38
- store.log, 39, 211, 227–231
- surrogate mode and, 312
- swap.state file, 211, 233
- useragent.log, 211, 232
- logfile_rotate directive, 373
- log_fqdn directive, 347
 - access.log and, 225
- log_icp_queries directive, 376
 - access.log and, 224
- log_ip_on_direct directive, 341
 - access.log and, 225
- log_mime_hdrs directive, 346
 - access.log and, 224
- loops, cache_peer_access directive and, 154
- LRU (least recently used) replacement policy, 101
- Lynx proxies, manual configuration, 423

M

- mailing lists
 - squid-announce, 7
 - squid-dev, 7
 - squid-users, 5, 6
- make command, ./configure script, 27
- max_challenge_lifetime parameter, auth_param directive, 205
- max_challenge_reuses parameter, auth_param directive, 205
- maxconn ACL type, 71
- maximum_object_size directive, 343
- maximum_object_size_in_memory directive, 343
- maximum_single_addr_tries directive, 383
- max_open_disk_fds directive, 389
- max-size option, cache_dir directive, 98

- max_user_ip ACL, proxy authentication and, 195
 - mbufs (BSD-based code), 17
 - mcast_groups directive, 333
 - multicast groups, 170
 - mcast_icp_query_timeout directive, 338
 - mcast_miss_addr directive, 390
 - mcast_miss_encode_key directive, 391
 - mcast_miss_port directive, 391
 - mcast_miss_ttl directive, 391
 - mem, cache manager page, 243
 - memory
 - netdb and, 163
 - requirements, 4
 - memory cache, 399
 - memory_pools directive, 375
 - memory_pools_limit directive, 375
 - memory_replacement_policy directive, 344, 400
 - menu, cache manager page, 253
 - method ACL type, 65
 - method field, store.log, 230
 - Microsoft NTLM authentication (see NTLM authentication)
 - mime_table directive, 344
 - min_dns_poll_cnt directive, 388
 - min_http_poll_cnt directive, 388
 - min_icp_poll_cnt directive, 388
 - minimum_direct_hops directive, 376
 - minimum_direct_rtt directive, 377
 - minimum_object_size directive, 343
 - miss_access ACL, 79
 - miss_access directive, 366
 - monitoring
 - Cache Manager, 237, 239
 - cache manager pages, 241
 - cache.log and, 237
 - warnings, 237
 - delay pools, 406
 - SNMP and, 290
 - snmpget and, 292
 - snmpwalk and, 292
 - SNMP MIB, 237
 - Squid MIB, 292–301
 - Mozilla proxies, manual configuration, 421
 - MSNT authentication helper (Basic authentication), 197
 - multicast ICP, 170
 - clients, 171
 - example, 172
 - servers, 170
 - myip ACL type, 61
 - myport ACL type, 65
- ## N
- N option, command-line, 45
 - NCSA authentication helper (Basic authentication), 197
 - negative_dns_ttl directive, 362
 - negative_ttl directive, 361
 - neighbor caches, 152, 153
 - always_direct directive, 161
 - cache_peer directive and, 155–159
 - cache_peer_access directive, 159
 - cache_peer_domain directive, 160
 - hierarchy_stoplist directive, 161
 - ICP and, 166
 - limiting requests, 159–162
 - never_direct directive and, 160
 - nonhierarchical_direct directive and, 162
 - prefer_direct directive, 162
 - selection algorithm, 179
 - up/down state, 158
 - neighbor_type_domain directive, 159, 337
 - NetBSD
 - file descriptors and, 16
 - filesystem benchmarks, 413
 - interception caching, IPFilter, 146
 - netdb, cache manager page, 285
 - netdb (network measurement database), 162
 - enabling, 163
 - ICP and, 169
 - memory requirements, 163
 - test_reachability directive, 168
 - netdb_high directive, 378
 - netdb_low directive, 378
 - netdb_ping_period directive, 379
 - netdb_state file, 211
 - netmasks, 55
 - Netscape proxies, manual configuration, 421
 - network hardware requirements, 4
 - never_direct ACL, 79
 - never_direct directive, 380
 - neighbor cache access list, 160
 - NLANR (National Laboratory for Applied Network Research), 3
 - no_cache ACL, 78
 - no_cache directive, 340
 - nonce_garbage_interval parameter, auth_param directive, 202
 - nonce_max_count parameter, auth_param directive, 202

- nonce_max_duration parameter, auth_param directive, 202
- nonhierarchical_direct directive, 391
 - neighbor caches and, 162
- non_peers, cache manager page, 288
- NTLM authentication, 194, 204
 - API, 206
 - auth_param directive, parameters supported, 204
 - SMB helper, 205
- NTLM authentication helper (Basic authentication), multi-domain, 198
- ntlmauthenticator, cache manager page, 250
- null storage scheme, 122

O

- object size, disk cache, 99
- objects
 - allocating to cache directories, 100
 - disk cache, removing from, 102–104
 - memory
 - complete, 399
 - in-transit, 399
- objects, cache manager page, 264
- offline_mode directive, 389
- offline_toggle, cache manager page, 254
- open source code, 5
 - precompiled binaries, 9
- OpenBSD
 - file descriptors and, 16
 - filesystem benchmarks, 413
 - interception caching and, 145
- openfd_objects, cache manager page, 265
- Opera, manual configuration, 422
- operating system
 - interception caching and, 141–146
 - requirements, 4
- origin servers, 2

P

- page faults
 - excessive, 238
 - getrusage() function, 238
- PAM authentication helper (Basic authentication), 198
- parent caches, 153
 - secondary selection, 181
- patches, applying, 32
- pathnames
 - log files, 38
 - mapping file numbers to, 231

- pconn, cache manager page, 280
- pconn_timeout directive, 364
- peer caches, 152
- peer_connect_timeout directive, 363
- peering code/peerhost field, access.log, 217
- peering codes, access.log, 222
- peer_select, cache manager page, 270
- persistent connections
 - client_persistent_connections directive, 395
 - pconn_timeout directive, 364
 - persistent_request_timeout directive, 363
 - pipeline_prefetch directive, 395
 - server_persistent_connections directive, 395
- persistent_request_timeout directive, 363
- pf (filtering software), 124
- pid_filename directive, 347
- pinger program, installation, 30
- pinger_program directive, 352
- pipeline_prefetch directive, 395
- policy routing (Cisco), interception caching and, 137
- pornography, access controls and, 85
- port ACL type, 63
- port number, changing, 303
- ports
 - ephemeral ports, 17
 - squid.conf directives, 38
 - TCP port numbers, ACLs, 59
- positive_dns_ttl directive, 361
- precompiled binaries, 9, 11
 - kernel and, 15
- prefer_direct directive, 392
 - neighbor caches, 162
- privacy issues, log files, 236
- processes, reconfiguring running, 52
- program parameter, auth_param directive, 196
- proto ACL type, 66
- proxies
 - all requests, 182
 - request through different, 182
 - Squid as, 1
- proxy authentication
 - directives, 195
 - HTTP interception caching and, 128
- Proxy Auto-Configuration, 423
- proxy_auth ACL type, 69
- proxy_auth_regex ACL type, 74
- pthreads library, aufs storage scheme, 112
- purging, surrogate mode and, 313

Q

- query_icmp directive, 379
- quick_abort_max directive, 360
- quick_abort_min directive, 360
- quick_abort_pct directive, 361

R

- R option, command-line, 45
- range requests
 - range_offset_limit directive, 362
- range_offset_limit directive, 362
- rc.d scheme, 49
- read-only option, cache_dir directive, 98
- read_timeout directive, 363
- Ready to serve requests message, 47
- realm parameter, auth_param directive, 196
- redirect_children directive, 189, 190, 353
- redirector, cache manager page, 250
- redirector pool, 189
- redirector_access ACL, 79
- redirector_access directive, 191, 353
- redirector_bypass directive, 191, 354
- redirectors
 - access controls and, 185
 - AdZapper, 193
 - buffered I/O, 187
 - configuration for, 190
 - definition, 185
 - filters and, 185
 - interface, 186
 - Jesred, 192
 - samples, 187
 - squidGuard, 192
 - Squirm, 192
- redirect_program directive, 190, 353
- redirect_rewrites_host_header
 - directive, 191, 353
- referer field, referer.log, 232
- referer.log, 231
- referer_log directive, 346
- referer.log file, 211
- refresh, cache manager page, 280
- refresh_pattern directive, 360
- refresh_pattern directive, disk cache, 104
- regular expressions, ACLs type, 59
- releases of Squid, 9
- reload_into_ims directive, 380
- reloads, surrogate mode and, 312
- removing objects
 - entire cache directories, 104
 - groups of, 103
 - individually, 102
- replacement policy, disk cache, 101
- replicating problems, debugging
 - and, 326–328
- reply_body_max_size ACL, 80
- reply_body_max_size directive, 368
- rep_mime_type ACL type, 74
- reporting bugs, 329
- req_mime_type ACL type, 74
- request method field, access.log, 217
- request_body_max_size directive, 360
- request_entities directive, 396
- request_header_max_size directive, 359
- requests
 - denying, access controls and, 89
 - different proxy, 182
 - single proxy, 182
- request_timeout directive, 363
- Request-URI
 - FQDN, 186
 - HTTP redirect messages, 187
 - ident_lookup_access directive, 186
 - whitespace, 187
- resources for support, 5
- response time field, access.log, 216
- responses, median time, 238
- restricting usage, access controls and, 86
- result code, access.log, 218
- result/status codes field, access.log, 216
- root, starting as, 37
- rotating log files, 53, 235
- routers
 - application-layer, 153
 - content routers, 153
- RTT (round-trip time), netdb and, 163
- rules
 - access controls
 - checks, 84
 - matching, 81
 - syntax, 80
 - ACLs, 78
- running processes, reconfiguring, 52

S

- s option, command-line, 44
- samples, redirectors, 187
- SASL (Simple Authentication and Security Layer), authentication and, 199
- scaling, ICP, 165

- scheme argument, `cache_dir` directive, 94
- security
 - log files, 236
 - surrogate mode and, 302
- segmentation violations,
 - debugging, 323–326
- server acceleration, 302
 - content negotiation, 311
 - overview, 303
 - (see also surrogate mode)
- `server_list`, cache manager page, 286
- `server_persistent_connections` directive, 395
- servers
 - FTP servers, 2
 - Gopher, 2
 - HTTP, 2
 - multicast ICP, 170
 - origin servers, 2
- server-side of Squid, 1
- sever acceleration, access controls, 310
- shells, file-descriptor limits, 15
- shutdown, 51
- shutdown, cache manager page, 254
- `shutdown_lifetime` directive, 365
- sibling caches, 153
 - false hits, 154
- size argument
 - `cache_dir` directive, 95
- `sleep_after_fork` directive, 398
- slow speed, debugging and, 318
- SMB (authentication helper)
 - Basic authentication, 199
 - NTLM authentication, 205
- SNMP
 - monitoring and, 290
 - `snmpget` and, 292
 - `snmpwalk` and, 292
- SNMP MIB, 237
- `snmp_access` ACL, 79
- `snmp_access` directive, 383
- `snmp_community` ACL type, 71
- `snmpget`, 292
- `snmp_incoming_address` directive, 383
- `snmp_outgoing_address` directive, 384
- `snmp_port` directive, 383
- `snmpwalk`, 292
- soft updates, filesystem tuning and, 110
- Solaris
 - file descriptors and, 17
 - filesystem benchmarks, 415
- source code
 - CVS and, 11
 - patches, applying, 32
 - precompiled binaries, 9, 11
 - shared libraries and, 10
 - Squid ports, 9
 - unpacking, 14
- speed, debugging and, 318
- Squeezer, 227
- Squid
 - as daemon process, 47
 - `squid_start` script, 48
 - history of, 3
- `squid -k` shutdown command, 51
- Squid MIB, 292–301
- Squid ports, 9
- Squid RPMs, 11
- `squidaido_counts`, cache manager page, 244
- `squid-announce` mailing list, 7
- `squid-cache` Web site, 5
- `squidclient` utility, Cache Manager and, 240
- `squid.conf`
 - access controls, 39
 - `cache_mgr` directive, 42
 - case sensitivity, 36
 - directives, 35, 331–398
 - error checking, 45
 - `http_port` directive, 38
 - syntax, 35
 - `visible_hostname` directive, 40
- `squid-dev` mailing list, 7
- `squidGuard` redirector, 192
- SquidNT, 419
- `squid.pid` file, shutdown and, 51
- Squid-RRD, cache manager and, 290
- `squid_start` script, Squid as daemon
 - process, 48
- `squid-users` mailing list, 5, 6
- Squirm redirector, 192
- `src` ACL type, 60
- `src_as` ACL type, 70
- `srcdomain` ACL type, 63
- `srcdom_regex` ACL type, 72
- SSL connections
 - `https_port` directive, 331
 - surrogate mode, 304
- `ssl_unclean_shutdown` directive, 332
- STABLE releases, 9
- stack traces
 - debugging, 323–326
- status code field, `store.log`, 230
- `stderr`, terminal window and, 46

- storage
 - aufs storage scheme, 112–115
 - coss storage scheme, 120
 - diskd storage scheme, 115–119
 - filesystems, 93
 - I/O bottleneck, 108
 - null storage scheme, 122
 - store_avg_object_size directive, 377
 - store_check_cachable_stats, cache manager page, 278
 - store_digest, cache manager page, 275
 - storedir, cache manager page, 276
 - store_dir_select_algorithm directive, 344
 - store_io, cache manager page, 279
 - store.log, 227–231
 - access.log comparison, 228
 - file numbers, mapping to pathnames, 231
 - store.log file, 39, 211
 - store_objects_per_bucket directive, 377
 - strip_query_terms directive, 392
 - access.log and, 226
 - subnets, delay pools and, 406
 - surrogate mode
 - access controls, 310
 - configuration, 304
 - content negotiation, 311
 - directives, 304–308
 - error messages and, 313
 - hierarchies and, 314
 - log files and, 312
 - purge operations and, 313
 - reloads and, 312
 - uncachable content, 313
 - swap directory error, debugging and, 315
 - swap.state file, 211, 233
 - syntax
 - access control rules, 80
 - access controls, 82
 - ACL elements, 54
 - squid.conf, 35
 - system calls, filesystems and, 93
- T**
- tar command, unpacking source, 14
 - TCP (Transmission Control Protocol), port numbers, 59
 - tcp_outgoing_address directive, 80, 367
 - tcp_outgoing_tos directive, 80, 367
 - tcp_rcv_bufsize directive, 374
 - technical support, 7
 - terminal window
 - cache.log messages, 214
 - testing and, 46
 - testing
 - access controls, 91
 - terminal window and, 46
 - test_reachability directive, 379
 - netdb and, 168
 - time ACL type, 67
 - timestamp field
 - access.log, 216
 - referer.log, 232, 233
 - store.log, 229
 - timestamps, 3
 - traffic shaping (see delay pools)
 - transfer size field, access.log, 216
 - transparent caching (see HTTP interception)
 - transport layer, application layer and, 405
- U**
- u option, command-line, 44
 - udp_incoming_address directive, 333
 - udp_outgoing_address directive, 334
 - UFS (Unix File System), 93
 - performance tuning, 110
 - uncacheable content, surrogate mode and, 313
 - unique_hostname directive, 370
 - Unix, Cygwin, 418
 - unlinkd_program directive, 352
 - unpacking source, 14
 - URI field
 - access.log, 217
 - referer.log, 232
 - store.log, 230
 - uri_whitespace directive, 187, 389
 - access.log and, 226
 - urlpath_regex ACL type, 73
 - url_regex ACL type, 73
 - usage restriction, access controls and, 86
 - user-agent field, useragent.log, 233
 - useragent.log, 211, 232
 - useragent_log directive, 346
 - usernames, ACLs type, 58
 - users, 37
 - access controls and, 88
 - utilization, cache manager page, 273

V

- V option, command-line, 45
- v option, command-line, 44
- vary_ignore_expire directive, 397
- versions of Squid, 9
- via_headers, cache manager page, 252
- visible_hostname directive, 40, 369
- vm_objects, cache manager page, 264

W

- wais_relay_host directive, 359
- wais_relay_port directive, 359
- WCCP (Web Cache Coordination Protocol)
 - interception caching, 138
 - configuration, 147
 - FreeBSD, 144
 - Linux systems, 142
 - NetBSD, 146
 - OpenBSD, 145
- wccp_incoming_address directive, 385
- wccp_outgoing_address directive, 385
- wccp_router directive, 384
- wccp_version directive, 385

- web caching, 2
- Web Polygraph, workload file
 - filesystem benchmarks, 408
- Webalyzer, 227
- whitespace, Request-URI, 187
- winbind authentication helper
 - Basic authentication, 200
 - NTLM authentication, 205
- Windows, Cygwin, 418
- workload files, Polygraph, 408
- WPAD (Web Proxy Auto Discovery) proxy
 - configuration, 425

X

- X option, command-line, 45

Y

- Y option, command-line, 45
- YP authentication helper (Basic authentication), 200

Z

- z option, command-line, 44

About the Author

Duane Wessels has been active in the field of web caching since 1994 when he worked on the Harvest project. Later he joined the National Laboratory for Applied Network Research to work on IRCache and Squid. His first book, *Web Caching*, was published by O'Reilly & Associates in 2001. He is currently co-owner of The Measurement Factory, Inc., a company that specializes in evaluating the performance, compliance, and security of HTTP intermediaries.

Colophon

Our look is the result of reader comments, our own experimentation, and feedback from distribution channels. Distinctive covers complement our distinctive approach to technical topics, breathing personality and life into potentially dry subjects.

The animal on the cover of *Squid: The Definitive Guide* is a giant squid (*Architeuthis dux*). Of the class *Cephalopoda*, which means “head foot,” the giant squid holds much fascination for humans, part of which has to do with the fact that it has never been observed alive in its natural habitat. Scientists have only been able to study specimens that have been caught or found washed up on beaches. This invertebrate can grow to 60 feet in length and weigh as much as a ton. It's a deep-sea dweller (660–2,300 feet) that is found throughout the world's oceans.

A giant squid consists of seven parts. Its head houses a complex brain. Its eyes are the largest in the animal kingdom—up to 10 inches in diameter. (Most deep-sea animals have very large eyes so they can gather the small amounts of light available in the depths of the ocean.) Its fins are relatively small and help it to balance and maneuver as it swims. Its main body is called a mantle: it's a muscular sac that contains most of the organ systems. Its eight arms are studded with two rows of suckers; it also has two much longer feeding tentacles, the ends of which also have suckers and are called clubs. Finally, its funnel is a multipurpose tube used to breathe, squirt ink, lay eggs, expel waste, and propel itself.

To eat, a giant squid captures its prey with its two long feeding tentacles. Holding the intended dinner with its shorter arms, its sharp horny beak cuts the food up, and a file-like radula sends it down the throat and esophagus; the food then passes directly through the brain to the stomach. Scientists believe giant squid may be solitary hunters because no more than one has ever been caught in the same fishing net.

Mary Anne Weeks Mayo was the production editor and copyeditor for *Squid: The Definitive Guide*. Sada Preisch proofread the book, and Marlowe Shaeffer and Claire Cloutier provided quality control. Jamie Peppard and Mary Agner provided production assistance. Johnna Dinse wrote the index.

Ellie Volckhausen designed the cover of this book, based on a series design by Edie Freedman. The cover image is a 19th-century engraving from the Dover Pictorial Archive. Emma Colby produced the cover layout with QuarkXPress 4.1 using Adobe's ITC Garamond font.

Melanie Wang designed the interior layout, based on a series design by David Futato. This book was converted by Joe Wizda to FrameMaker 5.5.6 with a format conversion tool created by Erik Ray, Jason McIntosh, Neil Walls, and Mike Sierra that uses Perl and XML technologies. The text font is Linotype Birka; the heading font is Adobe Myriad Condensed; and the code font is LucasFont's TheSans Mono Condensed. The illustrations that appear in the book were produced by Robert Romano and Jessamyn Read using Macromedia FreeHand 9 and Adobe Photoshop 6. The tip and warning icons were drawn by Christopher Bing. This colophon was compiled by Mary Anne Weeks Mayo.

Squid: The Definitive Guide

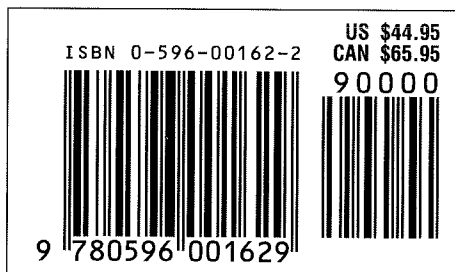


Squid is the most popular web caching software in use today. It works on a variety of platforms, including Linux, FreeBSD, and Microsoft Windows. Squid improves network performance by reducing the amount of bandwidth used when surfing the Web. It makes web pages load faster and can even reduce the load on your web server. By caching and reusing popular web content, Squid allows you to get by with smaller network connections. It also protects the hosts on your internal network by acting as a firewall and proxying your internal web traffic. You can use Squid to collect statistics about the traffic on your network, prevent users from visiting inappropriate web sites at work or school, ensure that only authorized users can surf the Internet, and enhance your privacy by filtering sensitive information from web requests. Companies, schools, libraries, and organizations that use web-caching proxies can look forward to a multitude of benefits.

Written by Duane Wessels, the creator of Squid, *Squid: The Definitive Guide* will help you configure and tune Squid for your particular situation. Newcomers to Squid will learn how to download, compile, and install code. Seasoned users of Squid will be interested in the chapters that tackle advanced topics such as high-performance storage options, rewriting requests, HTTP server acceleration, monitoring, debugging, and troubleshooting Squid.

Topics covered include:

- Compiling and installing Squid
- Running Squid
- Using Squid's sophisticated access controls
- Tuning disk storage for optimal performance
- Configuring your operating system for HTTP interception
- Forwarding requests to other web caches
- Using redirectors to rewrite user requests
- Monitoring Squid with the cache manager and SNMP
- Using Squid to accelerate and protect HTTP servers
- Managing bandwidth consumption with Delay Pools



Visit O'Reilly on the Web at www.oreilly.com



Bright Data Ltd. v. Oxylabs, UAB
IPR2023-00840, EX. 1009
468 of 468