Squid and ICP: Past, Present, and Future

Duane Wessels
wessels@nlanr.net

August 6, 1997

Abstract

Hierarchical Web caching has become prevalent throughout the Internet. The popular Squid software has played an important role. We present a short history of Squid and the Internet Cache Protocol, and offer our thoughts regarding its future evolution.

1 Introduction

Ever since the World-Wide Web rose to popularity around 1994, much effort has focused on reducing latency experienced by users. Surfing the Web can be slow for many reasons. Server systems become slow when overloaded, especially when hot spots suddenly appear. Congestion can also occur at network exchange points or across links, and is especially prevalent across trans-oceanic links that often cost millions of dollars per month.

Caching has proven a useful technique for improving end user experienced latency on the Web. The fundamental concept is the intermediate storage of copies of popular Web documents close to the end users. Caching is effective because many Web documents are requested (much) more than once[1]. Web browsers have local disk caches because individuals often browse the same pages repeatedly. Additionally, there is likely overlap in the set of documents requested by a large group of users. These users can benefit from a shared network cache.

Taking the proxy caching concept one step further, we might like to connect sets of caches together in a hierarchy. A group of Web caches can benefit from sharing another cache in the same way that a group of Web clients benefit from sharing a cache. In a simple cache hierarchy, a set of *child* caches share a common *parent* cache. Child caches forward to their parents requests for objects they do not have.

The Internet Cache Protocol (ICP) provides support for informed selection of a next-hop cache, including implicit indications of network congestion. Although the diagnostic functionality of ICP may be useful, simple cache topologies do not really require ICP to operate; other approaches can help recognize and deal with network failures. ICP is especially suited to enabling *sibling* relationships (i.e. queries between pairs of child caches). A group of sibling caches effectively acts as a large, distributed cache, a particularly useful configuration when no organization is willing to operate a parent cache for a large community with no explicit revenue stream for it.

The difference between sibling and parent relationships is in their role during cache misses: parents can help resolve misses, but siblings must not. Caches should not forward requests to sibling caches unless they know the sibling has the desired object. Without a mechanism to ascertain whether a cache has a given document, sibling relationships would not even be possible. ICP provides this functionality.

In this article we assume the reader is already familiar with HTTP/1.0[2] and at least aware of HTTP/1.1[3]. For additional information on ICP, please see our Internet Drafts (soon to be RFCs)[4, 5] and other work[6].

2 Squid

2.1 History

Squid was developed as a follow-on to one component of the Harvest project[7], the latter effort originally funded in 1994 by the U.S. Advanced Projects Research Agency (ARPA). Harvest focused on developing tools for effective use of Internet information, with emphasis on indexing and searching the Web. The project team developed a multi-faceted architecture for efficient cataloging and dissemination of popular information. One architectural component aimed at minimizing end user response time during Web navigation was an *object cache*, a platform strategically placed within the network topology that would store copies of popular documents to save users from having to wait for transmission of the document from the original source. The object cache component was named Harvest *cached*. The other Harvest components (the *gatherer* and the *broker*) were quite popular as well, and are still being used in a number of places. In fact, a group of volunteers has emerged to continue development of that code.

At that time, the only other Web caching (sometimes also called *proxy*) software in widespread use was the CERN HTTP Daemon[8]. The Harvest cache had two features that made it an attractive alternative to CERN's proxy. First, while CERN used a separate process for each request, Harvest serviced almost all proxy requests in a single process. The exception is FTP transfers, which are handled in an external process due to the complexity of the FTP protocol. Second, the Harvest cache software supported the notion of cache hierarchies.

The single-process nature of Harvest was attractive because many cache operators saw their machines slow to a crawl during busy periods. The operating system would spend a lot of time and resources forking and scheduling new processes, a price paid for the simplicity of the CERN server design. Harvest took this tradeoff in the opposite direction. As a single process, it has to worry about many operating system level tasks, in particular memory and file descriptor management.

Note, for the remainder of this article, unless stated otherwise, *Harvest* refers to the cache component of the Harvest research project. It does not in any way refer to the commercial software, which is also known as NetCache.

In late 1995 and early 1996, many members of the Harvest project moved on to industry jobs, bringing the project to a premature close. At the same time, Hans-Werner Braun, k claffy, and Michael Schwartz received U.S. National Science Foundation grant funding for their proposal to deploy a prototype Web caching system within the U.S. In March of 1996 I went to work for k and Hans-Werner on this project.

Shortly thereafter, Peter Danzig, primary technical architect for the original Harvest cache software itself, formed a company to sell a commercial version of the Harvest cache. In order to support the recently NSF-funded cache deployment and research project, we needed to have open and publicly available Web cache software. Since I had already worked for a year on the Harvest project, it made the most sense to take the last pre-commercial version of Harvest and enhance it to support research investigations. To avoid confusion between these two derivations of the Harvest software, we named our software Squid. We have spent the last year incorporating code contributions, suggestions, new feature requests, patches, error reports and other feedback into the Squid software.

After a few months of beta testing, the first official version of Squid (1.0.0) was released in July of 1996. Some of the more significant additional features and enhancements are described below.

2.1.1 Support for If-Modified-Since

The Harvest research code does not support If-Modified-Since (IMS) requests. IMS requests are used when clients or caches have a copy of an object and want to make sure it is the most up-to-date version. Harvest simply ignored the IMS header which caused user agents to always receive full replies. The primary reason that Harvest did not support IMS is because the last modification time is not kept as metadata in memory. To implement this, Squid needs to parse the reply headers as the object was read in from disk. Much of the initial code was written by Henrik Nordstrom.

In Squid 1.0, IMS is only supported between Squid and the client. Squid forwards IMS headers for cache misses, and provides the client with the corresponding reply, but does not update its own state with information from any *Not Modified* replies. Squid 1.0 also never initiates an IMS request on its own.

2.1.2 Public and Private objects

Harvest 1.4 treats all objects it knows about as *public*. Multiple clients can simultaneously receive data as an object is being retrieved. Although often a huge savings for large objects, under certain conditions this approach might allow a second client to receive something that should have been given only to the first. For example, consider a request that includes some authentication information. Because this data requires authentication, other clients should not have access to it from the cache. Instead they should separately authenticate themselves with the origin server.

The implementation decisions allowing this vulnerability are:

- A single hash table is used to index all objects in the cache, including pending objects.
- The hash table key is simply the URL for GET requests.

One reason for keying on only the URL is that the cache uses the URL extracted from an ICP reply to look up the pending request and continue the thread of execution. During the time interval between sending the ICP queries and receiving the replies, additional clients could attach themselves to the reply stream. This is a problem because the reply headers may indicate that the object should only be given to the first client.

Harvest could likely have avoided this problem by using a separate hash table for pending requests, or alternatively by using the *Reqnum* field of the ICP message. In fact, Harvest always set the Reqnum field to zero in ICP replies.

Squid implements the Reqnum field properly and uses it to support both *private* and *public* objects. Private objects are accessible only to the client originating the request, and public objects are available to all clients. Squid indexes objects in the storage hash table with a key that includes an integer number prepended to the URL string. Squid places this number into the Reqnum field of outgoing ICP query messages, and a peer must use the same Reqnum value in its reply. This technique allows Squid to use a cache key so that pending requests can be located from the ICP replies, but not by new clients. All requests start out as private, and remain so during the peer selection stage. Upon receipt of the HTTP reply headers, the object will become public, unless the reply indicates otherwise. Only public objects remain in the cache; private ones are removed immediately after transfer. If Squid is configured with an old Harvest peer (which sets the Reqnum field to zero), then the private object feature must be disabled, because it will be unable to locate the pending requests from Harvest's ICP replies.

2.1.3 Metadata reloading in the background

Harvest 1.4 does not serve any requests until it has rebuilt the cache metadata from the swap log file. Shutting the cache down cleanly will allow a much faster reload upon restarting the cache, similar to the benefit of `fsck` on Unix filesystems. Depending on the cache size, a fast reload could take anywhere from one to ten minutes. A slow reload could take hours. In either case, it is unrealistic to deny service to end users while the cache rebuilds.

Henrik Nordstrom implemented a background processing task to reload the swap log while the cache was serving requests. During this phase, Squid gives `select()` a zero timeout value. Upon a timeout, Squid processes a small number of lines from the swap log.

2.1.4 New Access Control scheme

With Harvest 1.4, the only access control (ACL) mechanism is the client's IP address. While working on Squid, it quickly became apparent that administrators wanted a rich, flexible set of access controls. We implemented the following additional access controls:

- Request method
- Access protocol
- Destination domain name
- Destination IP address
- Port number
- Regular expression matching the URL-path.
- Time of day

However, Squid never postpones requests to make access control decisions. Specifically, the destination IP address is only available if the URL hostname is already present in the IP cache. Squid never delays the ACL decision to wait for a DNS lookup to complete. If the URL contains an IP address instead of a hostname, Squid 1.0 does not make a reverse DNS lookup and compare the answer against the destination domain name.

2.2 Present

The first version of Squid 1.1 was released in December 1996. The most recent (1.1.14) was released July 14, 1997. We describe its new features below.

2.2.1 Refresh Rules and If-Modified-Since

Squid 1.0 removes objects when they expire. Squid 1.1 keeps expired objects on disk and issues If-Modified-Since requests for them. This can result in a significant bandwidth savings since the origin server doesn't need to retransmit the entire object. Unfortunately, IMS does less to reduce latency than bandwidth, because the origin server must still be contacted.

2.2.2 URL Redirector

Several Squid users in the community requested the ability to rewrite requested URLs or perform HTTP redirection. Rather than implement this feature in Squid itself, where an elegant solution satisfying everyone would be difficult, we chose to make it an external process. If configured to use redirection, Squid sends every request to one of the *redirector* processes. The redirector program must be supplied (i.e. written) by the administrator. A redirector process reads URLs on stdin, possibly changes the request, then writes the result to stdout. Squid reads the redirector output and modifies the request data structures.

One possible application of the redirector feature is additional access controls. If the redirector program has access to a list of forbidden URLs, it can redirect the request to a page describing exactly why the original request could not be allowed.

2.2.3 Reverse IP Lookups, client hostname ACLs.

Squid 1.1 includes support for access controls based on client domain name, previously impossible because Squid did not do reverse DNS lookups. Much like the IP cache, Squid caches reverse lookup results in the *FQDN cache*. We note that the domain lookups will incur some additional delay and we thus do not necessarily recommend their use.

2.2.4 Cache directory structure changes

The previous version of Squid used 100 subdirectories under each *cache_dir*. For large caches (e.g., more than 10^6 objects) this design would involve thousands of files per subdirectory. Such large directories can potentially cause unnecessary delays during filesystem operations.

A patch from Mark Treacy modified Squid to use a two-level directory structure, defaulting to 16 first-level directories and 256 second-level directories. In this configuration, a million objects would lead to a more sensible 256 objects per subdirectory.

2.2.5 Network Probe Database

Both Harvest and Squid allow requests to be routed based on the hostnames from URLs. However, there are a few problems with this approach. First, it requires a lot of manual configuration. The cache configuration file must list each domain with each peer; it is only practical to list top-level-domains (TLDs). In addition, domain restrictions don't work for URLs with IP addresses instead of fully qualified domain names.

The biggest problem is that the domain names are unrelated to network topology, apart from the rough mapping provided by the two-letter country code TLDs. These national TLDs can frame only a very coarse Web routing system, and international (top-level) domains, e.g., `.com`, `.net`, are even worse because they have no bearing at all to topology. For these reasons, any routing scheme based on domain names is inauspicious; like any other routing scheme, effective cache-level routing needs knowledge of the underlying network topology.

Squid 1.1 has an optional feature to acquire topology data with ICMP. When the *query_icmp* option is enabled, Squid builds up a table of hop counts and round-trip times for the origin servers it encounters. Squid aggregates this data by IP network under the assumption that two hosts on the same local network will have similar values. Via an external process, Squid caches send and receive ICMP echo requests to server hosts at a rate of no more than once per five minutes. Squid then includes the results of these network probe measurements in ICP reply messages. The cache collecting ICP replies uses the network measurements to select the best peer, ideally the one most toward the origin server.

As a cache collects network measurements from its peers, it adds them to its local table, learning over time which peers are good choices for which sources. The cache will be closer to some sources than any of its peers are; for these it can simply fetch directly and avoid the ICP querying. This approach complicates the peer selection algorithm, since instead of remembering a single best parent, Squid must now remember a list of possible parents until all ICP replies arrive.

2.2.6 Round-Robin IP

Squid's IP cache stores all addresses returned by the `gethostbyname()` function call. However, Squid 1.0 only uses the first address. Popular Web sites such as Microsoft require many servers, so an address lookup on `www.microsoft.com` returns 15–20 IP addresses. If the server at the first address fails, Squid 1.0 would never try any of the other addresses. This behavior is fixed in version 1.1. It cycles through all available addresses and deletes addresses for which TCP connections fail.

2.2.7 Neighbor features (neighbor_type_domain, miss_access).

When building a mesh of caches, having the same peering relationship (parent vs. sibling) for all requests might not be desirable. Consider, for example, a pair of upper-level caches in two different countries. Country A might want to forward requests for country B requests to the cache running in country B, in vice-versa. In other words, cache B would be a parent for B domains, and cache A would be a parent for A domains. But what about other requests? The cache administrators might want to leverage the fact that they are already peering with each other.

When some cache operators in Europe were building a high-level cache mesh, they were thwarted by the

Explore Litigation Insights

Docket Alarm provides insights to develop a more informed litigation strategy and the peace of mind of knowing you're on top of things.

Real-Time Litigation Alerts

Keep your litigation team up-to-date with **real-time alerts** and advanced team management tools built for the enterprise, all while greatly reducing PACER spend.

Our comprehensive service means we can handle Federal, State, and Administrative courts across the country.

Advanced Docket Research

With over 230 million records, Docket Alarm's cloud-native docket research platform finds what other services can't. Coverage includes Federal, State, plus PTAB, TTAB, ITC and NLRB decisions, all in one place.

Identify arguments that have been successful in the past with full text, pinpoint searching. Link to case law cited within any court document via Fastcase.

Analytics At Your Fingertips

Learn what happened the last time a particular judge, opposing counsel or company faced cases similar to yours.

Advanced out-of-the-box PTAB and TTAB analytics are always at your fingertips.

API

Docket Alarm offers a powerful API (application programming interface) to developers that want to integrate case filings into their apps.

LAW FIRMS
Build custom dashboards for your attorneys and clients with live data direct from the court.

Automate many repetitive legal tasks like conflict checks, document management, and marketing.

FINANCIAL INSTITUTIONS
Litigation and bankruptcy checks for companies and debtors.

E-DISCOVERY AND LEGAL VENDORS
Sync your system to PACER to automate legal marketing.

WHAT WILL YOU BUILD? | **sales@docketalarm.com** | **1-866-77-FASTCASE**

fastcase® Smarter legal research.