ICP and the Squid Web Cache*

Duane Wessels
k claffy

August 13, 1997

Abstract

We describe the structure and functionality of the Internet Cache Protocol (ICP) and its implementation in the Squid Web Caching software. ICP is a lightweight message format used for communication among Web caches. Caches exchange ICP queries and replies to gather information to use in selecting the most appropriate location from which to retrieve an object.

We present background on the history of ICP, and discuss issues in ICP deployment, efficiency, security, and interaction with other aspects of Web traffic behavior. We catalog successes, failures, and lessons learned from using ICP to deploy a global Web cache hierarchy.

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.

A common, albeit expensive way to alleviate such problems is to upgrade the overloaded resource: get a faster server, another E1, a bigger switch. However, this approach is not only often economically infeasible, but perhaps more importantly, it also fails to consider the numerous parties involved in even a single, simple Web transaction. In addition to the user and the information provider, typically several Internet Service Providers and exchange points participate in delivering requested information to the user. Like the proverbial chain, a transaction is only as good as its weakest network link. As an example, Microsoft could have a thousand Web servers and a thousand OC3 circuits, but downloading the latest version of MSIE[1] will still take 20 hours for some folks in Moscow who have a single 19.2 kbps link for all of their simultaneous TCP/IP connections.

Caching has proven a useful technique for reducing end user experienced latency on the Web [1, 2, 3, 4, 5, 6, 7]. 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 [8]. 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.

*This work is supported by a grant of the National Science Foundation (NCR-9521745).

[1]The MSIE 3.02 package for Windows 95 is 10.3 Mbytes.

World-Wide Web caches are implemented as *proxies*. Normally Web clients (browsers) make direct connections to Web servers. However, clients may instead be configured to connect to a proxy application which then forwards the request to the server on behalf of the client. For this reason Web caching is also occasionally referred to as *proxy caching*. Proxies are often used as a gateway between two sides of an Internet firewall, and are not necessarily used for caching.

1.1 HTTP and Caching

The Hypertext Transfer Protocol [9, 10] has several basic features relevant to Web caching. At the time of this writing, the majority of Web clients and servers use HTTP version 1.0. In many cases, HTTP/1.1 features are incrementally added to software products. Most of what we describe here pertains to HTTP/1.0, except for the *Cache-control* features, which are part of HTTP/1.1.

An HTTP request is comprised of three major parts: a request method, a Uniform Resource Locator (URL) [11], and a set of request headers. An HTTP reply consists of a numeric result code, a set of reply headers, and an optional reply body. The most common request method is GET, which is a request to retrieve the information indicated by the URL. A GET request is essentially a *download* operation. Another commonly used request method is POST, which is essentially an *upload* operation.

A special kind of GET request is a *conditional GET*, which is differentiated by the inclusion of an *If-Modified-Since* header, and whose reply depends upon the modification date of the URL compared to the date provided in this If-Modified-Since header. *Conditional GET* is an important feature for caching, as it allows the server to send a small *Not Modified* response if the client already holds a current copy of the requested resource. If the resource has changed since the If-Modified-Since timestamp, the server sends the current version.

The HTTP/1.1 *Cache-control* header allows both clients and servers to "override the default caching algorithms" [10]. For this article, we are primarily interested in the `Max-age` directive, which lets the client place an upper limit on how old an object can be and still satisfy the request without refreshing the document from the source. Here *age* refers to the elapsed time since the origin server provided the data. If the cached object is older than the client requires, the request must be forwarded to the origin server.

1.2 Hierarchical Caching and ICP

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. Like other wide-area, multi-administration Internet services such as the Domain Name Service (DNS) [12, 13], Usenet newsgroups, and Classless Inter-Domain Routing (CIDR) [14], the use of hierarchical structure is particularly auspicious for the scalability and manageability of Web caching. Figure 1 depicts a very simple cache hierarchy. A set of *child* caches share a common *parent* cache. Child caches forward requests to their parents for documents they do not have. Ideally, a hierarchy of Web caches is based on the underlying Internet topology, where an Internet service provider operates a parent cache and his customers operate the child caches.

However, such a simple hierarchy is not appropriate to all situations. For example, there might be multiple parent caches. In that case, to which parent should a cache forward requests, and what information might be available to aid the cache in this decision? Or perhaps there is no parent cache at all. Consider three caches operated by different departments of a university where,

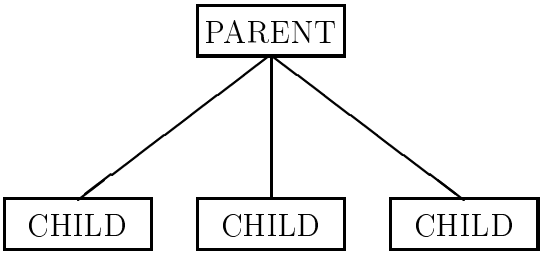

Figure 1: Simple cache hierarchy model. Child caches forward misses up to a parent cache.

for some reason, the networking organization is unwilling to operate a cache for the university as a whole. The departments may want to leverage each others' caches, by requesting from their counterpart caches the documents those caches have already retrieved. But how do they know which documents those are?

The Internet Cache Protocol (ICP), whose role is to provide a quick and efficient method of inter-cache communication, offers a mechanism for establishing complex cache hierarchies. ICP allows one cache to ask another if it has a valid copy of a named object, improving its chances of selecting a parent cache that would result in a cache hit. Section 4 describes these processes in detail. In addition, ICP allows requests to flow between child caches. We refer to this as a *sibling* relationship. The only difference between sibling and parent relationships is in their role during cache misses: parents can help resolve misses, and siblings do not. The parent/sibling distinction has a number of interesting consequences and issues which we discuss in section 6.1.

2 Related Work

Web caching is based on established techniques used for improving the performance of distributed (networked) filesystems [15, 16, 17, 18, 19]. Modern computer systems use caching for memory and local disk accesses as well. The most difficult part of any caching scheme is maintaining *cache consistency*, or keeping the cached data synchronized with the source. As mentioned above, HTTP has the *If-Modified-Since* feature for this purpose.

The CERN HTTPD [20] is the original proxy cache. Under heavy load, it suffers from two design flaws: 1) each request is handled by a separate process, and 2) the filesystem is used to index cached objects (i.e. each cache lookup requires a `stat()` system call). CERN caches can be arranged hierarchically, but every single cache miss is forwarded on to a single parent cache.

The Harvest cache [21, 22] set out to improve upon CERN by focusing on hierarchical caching and and an efficient process implementation (i.e. no forking and minimal disk access). However, Harvest fell short on properly implementing the HTTP protocol, most notably it does not support *If-Modified-Since* requests. When the Harvest project ended, development of the cache software continued by two groups. One is a commercial product called NetCache [23]. The other is called Squid, which we maintain as a part of our National Science Foundation grant to develop a prototype caching architecture [24]. Harvest, NetCache, and Squid all support ICP.

Most Web caching systems in use today, including Squid, are demand-driven from the clients [25, 26, 27, 28, 29, 22, 20]. In other words, the caches are passive, and objects are only retrieved

or validated when requested by a client. This is similar to how Sun Microsystems' Network File System (NFS) [30] operates. NFS servers are stateless (as are HTTP servers), and NFS clients are tasked with maintaining cache consistency.

One alternative to demand-driven caching is known as *push-caching* [31, 32]. Gwertzman and Selzter [31, 33] propose that Web servers replicate popular data in advance based on geography.[2] Push-caching is similar to the Andrew File System (AFS) [34] because the servers become stateful and can invalidate stale cached data. However, an AFS server does not choose where data should be cached.

Replication is a technique similar to caching, but is generally considered to be more active. The process of setting up a new replica is often manually intensive (e.g. editing configuration files, installing cron jobs, and updating listings). Baentsch et. al. [35] have proposed a scheme for automatic replication of popular data, in which users rely on proxy caches that are aware of the replicated servers.

Prefetching can be effective at reducing latency at the expense of increased bandwidth usage. The inherent drawback is that some data will be prefetched but then never requested by the user. To be most effective, a prefetching proxy cache must accurately predict future requests. Padmanabhan and Mogul [36] propose that Web servers offer predictions to clients regarding the likelihood of future requests to the server, based on which clients could decide whether to prefetch specific resources.

Prefetching has been implemented in the Wcol proxy cache [37]. Chinen and Yamaguchi [38] examine hit rates, transfer times, and network traffic volume for prefetching compared to demand-driven caching and to no caching at all. Prefetching increased the amount of network traffic by a factor of 2.8 compared to no caching, and by a factor of 4.1 compared to demand-driven caching. The average retrieval time decreased by a factor of 1.6 compared to no caching, and by a factor of 2.5 for demand-driven caching. Finally, demand-driven caching resulted in a 39% document hit rate, but with prefetching it reached 64%.

Some Web sites, notably the Internet Movie Database (`www.imdb.com`) specifically ban prefetching and so-called Web accelerators based on IP address or the *User-agent* request header. A request to `www.imdb.com` without a *User-agent* header results in a page that includes the following text:

> your address or browser or proxy server has been banned for misuse of our service, e.g. overloading our servers with automated requests.
>
> All requests from NetJet, NetCarta, Autonomy, WebWhacker, FlashSite, Java102, infoGear, Teleport-Pro, MemoWeb, Microsoft's MS-Catapult/0.9, Netscape's Catalog-Robot are rejected because of persistent attempts to download huge numbers of URLs as fast as the networks permit.

Several efforts have focused on mechanisms for locating the best server to ask for a document. One approach is to add geographical coordinate information to DNS records [39]. However, topology does not generally match geography, so this technique has limited utility for Web caching.

Another approach is under development within the IETF. SONAR [40] is a simple message format for expressing the relative proximity of a set of server addresses. One implementation uses round-trip time as the proximity metric, but unfortunately the current version does not require a single

[2]This scheme requires mapping addresses to network administrator contacts with street addresses, using a file made available by Merit as a part of the NSFNet project, and then using the U.S. poastal ZIP code to get latitude and longitude coordinates from a geography server.

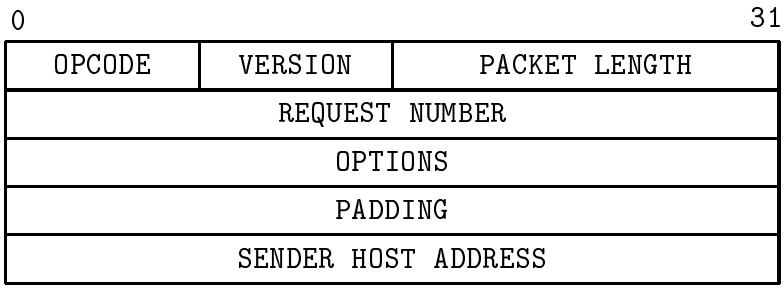

Figure 2: ICP header format.

standard metric, making it difficult to compare SONAR information from different SONAR servers. The proposed Host Proximity Service [41] is similar to SONAR, but more ambitious.

Carter and Crovella argue that dynamic server selection can outperform static schemes by as much as 50% [42]. Here a static policy would always assign the same client to the same replicated server. By measuring the Internet (i.e. throughput and latency) in near real-time, server selection can change relatively quickly in response to network congestion and other instabilities. They offer a pair of diagnostic tools to measure both the base bandwidth and available bandwidth of an Internet path.

Floyd, Zhang, and Jacobson are beginning a research project called Adaptive Web Caching [44]. Central to their approach is the use of reliable multicast to disseminate objects among proxy cache groups. They propose to use IP routing information to forward cache misses toward origin servers.

Some people believe that tree-like hierarchies have appealing scaling properties. Others argue that upper level nodes of the tree become excessively loaded and degrade performance, so distribution is the key to scaling. Povey describes a Distributed Internet Cache [4] and modifications to ICP to replace top-level caches with *pointer servers*. Instead of holding cached data, pointer servers remember which caches hold which objects, and direct requests accordingly. A similar project, known as CRISP [45] is underway at Duke University and AT&T.

We are not presently aware of any other protocols which provide the same functionality as ICP, although it might be possible to implement the same ideas with HTTP and the HEAD request method. URL hashing has been proposed [46] as an alternative method for load balancing and parent selection, however only ICP enables sibling relationships among caches.

3 The Internet Cache Protocol (ICP)

3.1 Message Format

An ICP message consists of a fixed, 20-octet header plus a variable-sized payload, the latter of which typically contains a URL. Figure 2 shows the ICP message format; all fields are represented in network byte order.[3]

[3] As of this writing the most current description of ICP is in Wessels' Internet Drafts [47, 48].

- OPCODE: indicates the type of message. Some common opcodes are ICP_QUERY, ICP_MISS, and ICP_HIT.
- VERSION: ICP protocol version, for maintaining backward compatibility.
- PACKET LENGTH: total size of the ICP message.
- REQUEST NUMBER: an opaque integer identifier to match queries and responses.
- OPTIONS: bitfield to support optional features and recent additions to the protocol.
- PADDING: currently unused. In the Harvest research code, the OPTIONS and PADDING fields were slated for authentication purposes.
- SENDER HOST ADDRESS: originally intended to hold the IPv4 address of sender. However, since the originating address is also available from the socket API, and more difficult to spoof, this field is redundant and often unused.

A cache will query its peers by sending each one an ICP_QUERY message. The payload of the ICP_QUERY message is a URL. Upon receipt of an ICP_QUERY, a cache will extract the URL, check for its existence locally, then reply with either an ICP_HIT or ICP_MISS message. An ICP_MISS indicates that the replying cache does not have the requested URL. Conversely, ICP_HIT indicates the cache does hold the URL. Later we will discuss other reply opcodes, such as ICP_DENIED.

The cache originating the ICP_QUERY collects the reply messages and then chooses a peer cache from which to retrieve the object. We describe this peer selection algorithm in section 4.3. ICP also defines two special opcodes to support the inclusion of the origin server and non-ICP caches in the selection algorithm. Since the Web server does not necessarily support ICP, in these cases we use the server's echo service (port 7) to determine host reachability. These opcodes are ICP_SECHO, for 'source ping,' and ICP_DECHO, for a 'dumb cache.' After selecting a source for the object, the cache makes a regular HTTP request to retrieve it.

3.2 Transport

ICP could use either TCP or UDP as the underlying delivery protocol, though it currently uses only UDP for a couple of reasons. First, a UDP version is simpler to implement because each cache needs to maintain only a single UDP socket. Second, ICP is intended as an unreliable protocol and TCP's retransmission would actually be detrimental. The ICP query/reply exchange must occur quickly, within a second or two. A cache cannot wait longer than that before beginning to retrieve an object. Failure to receive a reply message most likely means the network path is either congested or broken. In either case we would not want to select that peer.

3.3 ICP vs. HTTP

Note that ICP is extremely lightweight; even the fixed size header is in binary format for compactness. The payload is most often simply a URL. This simplicity has both a positive and a negative aspect.

The main advantage is that a cache can quickly parse and interpret an ICP message. A cache receiving an ICP_QUERY needs only to extract the URL, check for the existence of the object, and

then reply with hit or miss. It is important that the ICP turnaround time be extremely fast because a cache might handle many more ICP requests than HTTP requests (e.g. a factor of 2–6 for the NLANR caches[4]). We want a fast turnaround time so that ICP requests do not significantly burden the cache process, and to minimize the delays within a cache hierarchy.

A significant disadvantage to ICP's simplicity is that it does not match HTTP. Caches use ICP to locate objects, but must use HTTP to actually retrieve them. The recent advent of HTTP/1.1 introduces many parameters of an HTTP request that are not expressible in an ICP query. Section 6 gives specific failure mode examples. Another disadvantage, not related to the message format, is that ICP increases the request latency by at least the network round-trip time to a neighbor cache.

These differences between ICP and HTTP have recently become the subject of debate: whether to keep ICP lightweight or couple it more toward the evolving functionality of HTTP. Incorporating HTTP features into ICP would allow more sophisticated object location, since one could specify the entire HTTP request rather than just the URL. However, obviously adding such complexity would require additional CPU cycles to fully parse the HTTP request (in text), and the ominous size and complexity of the recent HTTP/1.1 RFC [10] lends concern to this methodology as a general direction. If we go this far then we can simply reduce ICP to 'HTTP over UDP,' with an additional request method (query) and a few more status codes (hit, miss, etc.).

3.4 Source Ping via ICMP

Although the source ping feature proved useful, complaints from paranoid system operators, opposed to having packets sent to their echo port, led us to discourage its use. The complaints were particularly loud following a CERT advisory regarding UDP-based denial-of-service attacks [49]. In conjunction with other work, Squid supports sending ICP_SECHO messages via ICMP instead of UDP. Unfortunately, ICMP packet transmission requires superuser privileges. Note that we cannot simply call the *ping* program because the ICP_SECHO messages have specific content (the requested URL) and the `fork()` and `exec()` system calls required to start the *ping* process would wreak havoc with Squid's performance.

4 Implementation of ICP in Squid

The following sections describe algorithms for sending ICP queries and collecting the replies as implemented in the Squid cache software.

4.1 ICP Query Algorithm

By default, Squid always sends ICP_QUERY messages to each peer. In certain configurations (e.g. with multiple parent caches in different directions) this practice can yield undesirable results, so Squid supports the ability to restrict the range of ICP_QUERY messages it will send to different peers. Specifically, one can configure Squid to only send ICP queries to certain peers for URLs inside specific DNS (usually top-level) domains. The `cache_host_domain` option lets one specify which domains to query for a given peer. As an example, a U.K. cache may have a peer relationship with a cache in Germany, and configure their U.K. border cache to only send ICP queries for `.de`

[4] `http://ircache.nlanr.net/Cache/Statistics/Vitals/`

URLs to the cache in Germany. The German cache is not interested in handling requests for `.jp` URLs from sites in the U.K.

Another Squid configuration parameter, `hierarchy_stoplist`, allows one to exclude certain requests from the ICP query algorithm. By default, Squid excludes any URLs containing the string `cgi-bin` or a question mark (`?`), since these URLs tend to be dynamic and/or uncachable, and so there is a low probability of receiving an ICP_HIT for them. More importantly, the hierarchy stoplist allows one to limit the workload passed to an upper level cache. The upper level caches should not handle objects unlikely to be requested more than once.

For every request not handled directly due to the hierarchy stoplist, Squid sends an ICP_QUERY message to each peer, unless:

- `cache_host_domain` rules prevent use of the peer for the given URL.
- a TCP connection to the peer has failed within the last minute.
- the peer has been configured locally with the `no-query` option.

To reduce bias from the ordered list of peers, the starting peer is offset by one each time. If Squid has the *source ping* option enabled, then it sends an ICP_SECHO message to the echo service of the origin server (i.e. HTTP daemon) for the requested URL. To its peer caches that do not speak ICP, Squid sends an ICP_DECHO message. After sending all the queries, Squid installs a timeout to make sure it begins retrieval of the object within a short time (default is 2 seconds), whether any replies arrive or not.

4.2 Processing an ICP query

When a Squid cache receives an ICP_QUERY, it processes the request as follows:

- Extracts and parses the URL. If the URL is not valid, return an ICP_INVALID message. This should only occur rarely since the querying site should have already checked the URL for correctness.
- Check local access controls. If access is denied, return an ICP_DENIED message. Receipt of an ICP_DENIED indicates a mismatch in configuration between peers.
- Lookup the given URL. If the object does not exist in the local cache, or exists but will become stale within 30 seconds, return an ICP_MISSmessage.
- If the object is small enough, return an ICP_HIT_OBJ message, including the entire object in the ICP payload. We discuss ICP_HIT_OBJ below.
- Otherwise, return an ICP_HIT message.

4.3 Collecting ICP replies

The peer selection algorithm is relatively simple. Squid collects replies until it receives an ICP_HIT, or until all ICP_MISS replies arrive. For this purpose ICP_SECHO and ICP_DECHO replies are always treated like hits. Immediately upon receiving an ICP_HIT, Squid begins retrieving the object from

that peer. Otherwise, Squid waits for all replies to arrive, up to the configurable timeout whose default is 2 seconds.

If an ICP_HIT_OBJ (section 4.6) reply is the first to arrive, then Squid has finished object retrieval without even needing to make an HTTP request, so it just takes the object data from the ICP message payload and adds it to the local cache.

If there are no hit replies, then Squid retrieves the object from the parent with the minimum weighted round-trip time. Normally, all parents are unweighted, which means that Squid will use the first parent to reply. We added a weighting capability to allow favoring some parents over others. If there are no miss replies from parents, then Squid fetches the object directly from the origin server (unless there is a firewall in place).

We essentially always choose the peer with the lowest RTT. While RTT is a reasonable metric for ICP_HIT's, its validity for resolution of ICP_MISS'es is debatable. Many people think that bandwidth or throughput would be better metrics, but these characteristics are more difficult to measure than RTT. Ideally, when choosing among a set of parents, we want to forward the request to a parent in the direction toward the origin server. In section 7.1 we describe some current work-in-progress to tackle this ultimately challenging task.

4.4 Detecting Unreachable Peers

Recall that the selection algorithm waits for all replies to arrive (unless one of them is an ICP_HIT). One of the peers becoming unreachable would significantly increase the chances of suffering the two-second timeout. To prevent this situation we need to detect when a peer becomes unreachable.

We designate a peer as dead when it fails to reply to 20 consecutive ICP queries. However, we continue to send ICP_QUERY messages to dead peers; we just don't expect to receive replies from them. As soon as the peer becomes reachable again, it is marked alive and we again include it in the count of sent messages to which we expect replies.

4.5 More Network Failures

UDP transport allows ICP to gracefully accommodate network failures, albeit only for failures between a pair of peers. Network failures have also been known to occur between the parent and the rest of the Internet. Consider figure 3, where the child caches have two ways to reach the global Internet, either via link A, or link B. Assume that link A is faster and therefore preferred over B.[5]

What happens when link A goes down? The child caches still have good connectivity to the parent, and will therefore receive ICP_HIT or ICP_MISS replies as usual. However, the parent cache will be unable to satisfy any miss requests because its path to the Internet is down. The users of the child caches will get many 'connection failed' error messages, even though they have an alternate way of reaching the Internet.

Squid keeps track of its failed requests to cope with this problem. When the ratio of failed to successful requests exceeds a threshold (i.e. 1) then Squid returns ICP_MISS_NOFETCH[6] instead of

[5] The NLANR caches [24] experienced this failure mode in September 1996. Specifically, the caches can all talk to each other over the vBNS, but cache MISS traffic primarily goes out on commodity networks. So when the commodity backbone fails, the caches can still send ICP queries to each other over the vBNS. Although the diagram shown here depicts a slightly different situation, the concept is the same.

[6] ICP_MISS_NOFETCH was previously called ICP_RELOADING.

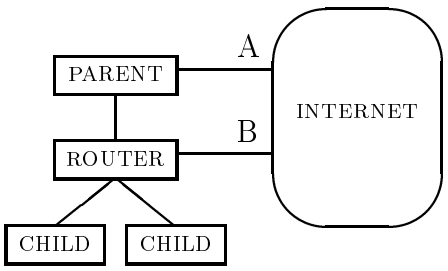

Figure 3: When good parents go bad. Normally, the child cache requests are satisfied via the parent and link A. But when link A fails, the child caches have an alternate path via link B. If the parent cache begins to encounter a large amount of failed requests, it should eliminate itself from the selection process for cache misses.

ICP_MISS replies. This feature allows a parent cache to continue serving hits, but take itself out of the peer selection process for misses.

4.6 ICP_HIT_OBJ

The ICP_HIT_OBJ idea derived from the realization that many Web objects are small (e.g. less than 4 kbytes) and could be piggybacked inside the ICP replies. Avoiding the subsequent HTTP request with the three-way handshake and other TCP overheads could prove to be very efficient, so we implemented the ICP_HIT_OBJ capability in Squid.

One problem with ICP_HIT_OBJ is that it makes the UDP packet quite a bit larger. The large UDP packet will likely undergo fragmentation and reassembly, bringing with it a higher chance of being dropped due to congestion. Increased packet loss is particularly bad for ICP since the two-second timeout would occur. Another problem with ICP_HIT_OBJ is they require more time to generate. Whereas caches can return standard hit and miss replies immediately, they must load the object data into memory before returning an ICP_HIT_OBJ reply. ICP_HIT_OBJ replies may thus be among the last to arrive, and are more likely to arrive after the two-second timeout. In some cases the tradeoff may not be worth it (avoiding the TCP connection versus losing the hit reply). In a cache hierarchy it may be better to have a normal ICP_HIT rather than no hit at all. Some cache administrators were also concerned about the lack of rate control for UDP datagrams. We do not believe this is an issue since the ICP messages are not sent in a continuous stream.

Fitting the object data inside the ICP message payload is somewhat ad hoc. The payload must actually consist of the URL followed by the object data. A NULL byte terminating the URL also marks the beginning of object data. Fortunately the PACKET_LENGTH field exists so we can verify receipt of the entire object data.

Backward compatibility was also an issue. We use one bit in the OPTIONS field to indicate support for the ICP_HIT_OBJ opcode, which also allows it to be disabled on a site-by-site basis.

We were originally quite enthusiastic about this feature, until we began to hear about associated complications, derived from the fact that the ICP request cannot convey everything from the HTTP request. In particular, the ICP query does not retain information from the HTTP/1.1 `Cache-control: Max-age` header. Some cache operators were noticing that ICP_HIT_OBJ was de-

livering older-than-requested objects. Another example is the 'Cookie' header, sometimes used to deliver customized content. For these reasons we now recommend against using ICP_HIT_OBJ.

4.7 Public vs. Private Objects

The Harvest cache (section 8.1) version 1.4 has an interesting mis-feature, which opened up a vulnerability that current versions of Squid have closed. In Harvest, 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. 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.

The Harvest cache 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.[7] 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 an ICP reply.

5 ICP Delays

We mentioned in section 3.3 that caches must handle ICP queries quickly. We have performed a series of measurements to assess the delay that ICP contributes to cache transactions. We do not claim these measurements prove that hierarchical caching with ICP gives improved performance; we consider this an important area for further experimentation. We suspect it depends on the regional and/or local network situation. For this article, we only wanted to find out how quickly ICP requests can be processed.

We repeatedly measured two values over a four hour period: the network RTT and the *ICP RTT*. We used a special program to alternate between sending ICMP echo requests and ICP_QUERY

[7] Private objects are accessible only to the client originating the request; public objects are available to all clients.

Site	ICMP Median	ICMP 95% conf. int.	ICMP %Loss	ICP Median	ICP 95% conf. int.	ICP %Loss
IT	43.3	(43.2, 43.4)	0.2	54.7	(52.4, 57.6)	1.7
PB	37.2	(37.1, 37.3)	0.1	44.5	(44.1, 45.0)	0.2
UC	38.5	(38.4, 38.8)	15	41.9	(41.4, 42.6)	16
BO	2.0	(2.0, 2.0)	0.1	4.0	(3.8, 4.2)	0.1
SV	58.6	(58.4, 58.9)	22	61.0	(60.6, 61.2)	21
SD	78.2	(77.7, 79.0)	24	80.4	(79.6, 81.3)	25

Table 1: Summary of the distributions shown in figure 4. The median RTT values (in milliseconds) are shown, with 95% confidence intervals, for both ICMP and ICP measurements. The amount of packet loss is shown for each measurement as well.

messages, making a set of ICMP measurements, followed by a set of ICP measurements, and recording all RTTs between the local host and each of the six NLANR Squid caches [24]. Unlike an ICMP packet, the ICP request must go up to the application layer, inherently costing some extra delay. The goal of the study was to verify that Squid can process ICP requests very quickly, in which case the ICP RTT should be only slightly higher than the network RTT.

The top graph in figure 4 shows the cumulative percentiles for the network RTT data. The BO plot represents a cache co-located near the measurement source, so the measured values are small. Paths to two of the other caches (IT, PB) take the vBNS,[8] and the other three (SV, SD, UC) used MCI's commodity Internet on this particular day.

The bottom graph in figure 4 shows ICP RTT measurements. Visual inspection reveals that the ICP distributions have longer and heavier tails, but about 50–60% of the ICP requests seem to be only slightly higher than the network RTT.

Table 1 summarizes the data for all the caches. We show the median values with 95% confidence intervals, as well as the observed packet loss.[9] Because these distributions are extremely asymmetric, calculation of a mean value has little meaning. Taking instead the median as the average, we find that the ICP RTT is generally just a little larger than the network delay. We note that ICP packet loss closely follows ICMP packet loss, which is useful since it implies that *ping* measurements are reasonable predictors of ICP loss rates and the probability of suffering a timeout in the peer selection stage.

The graphs in figure 5 have implications regarding how cache load affects the ICP RTT. The two busiest caches, IT and PB, show the biggest difference between network and ICP RTTs. The least busy cache, SD, shows a stronger correlation between the two measurements. From this, we can qualitatively deduce that the ICP turnaround time is related to the request load. As a cache becomes busier, it takes longer to service the ICP requests, probably because they are not read from the ICP socket queue as frequently.

[8] `http://www.vbns.net`

[9] The measurements were made during the middle of the day, but these loss rates still seem unusually high.

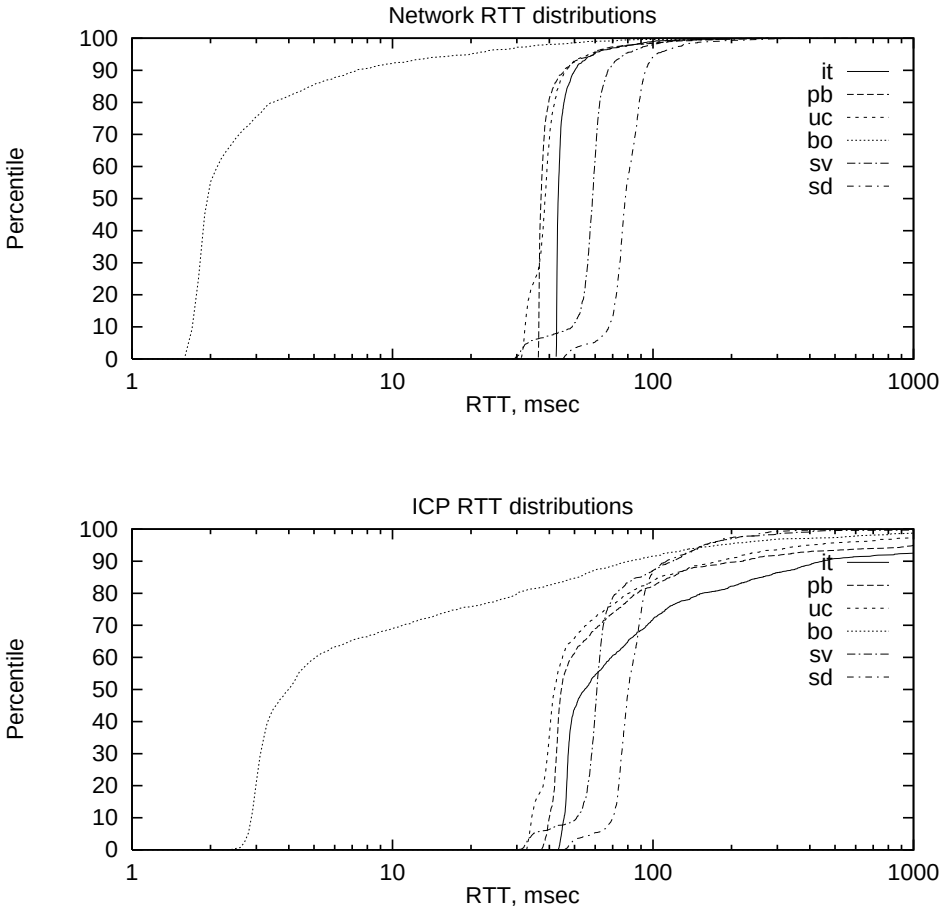

Figure 4: Network and ICP RTT measurements. Cumulative distributions of round-trip times between a single host and the six NLANR root caches. The top graph is based on ICMP data; the bottom graph on ICP data. We are looking to verify that an ICP request does not take much longer than the round-trip time since it will delay retrieval of the object. This data was collected over a four hour period on January 29th, 1997. The six NLANR caches are: IT-Ithaca, NY; PB-Pittsburgh, PA; UC-Urbana-Champaign, IL; BO-Boulder, CO; SV-Silicon Valley, CA; SD-San Diego, CA.

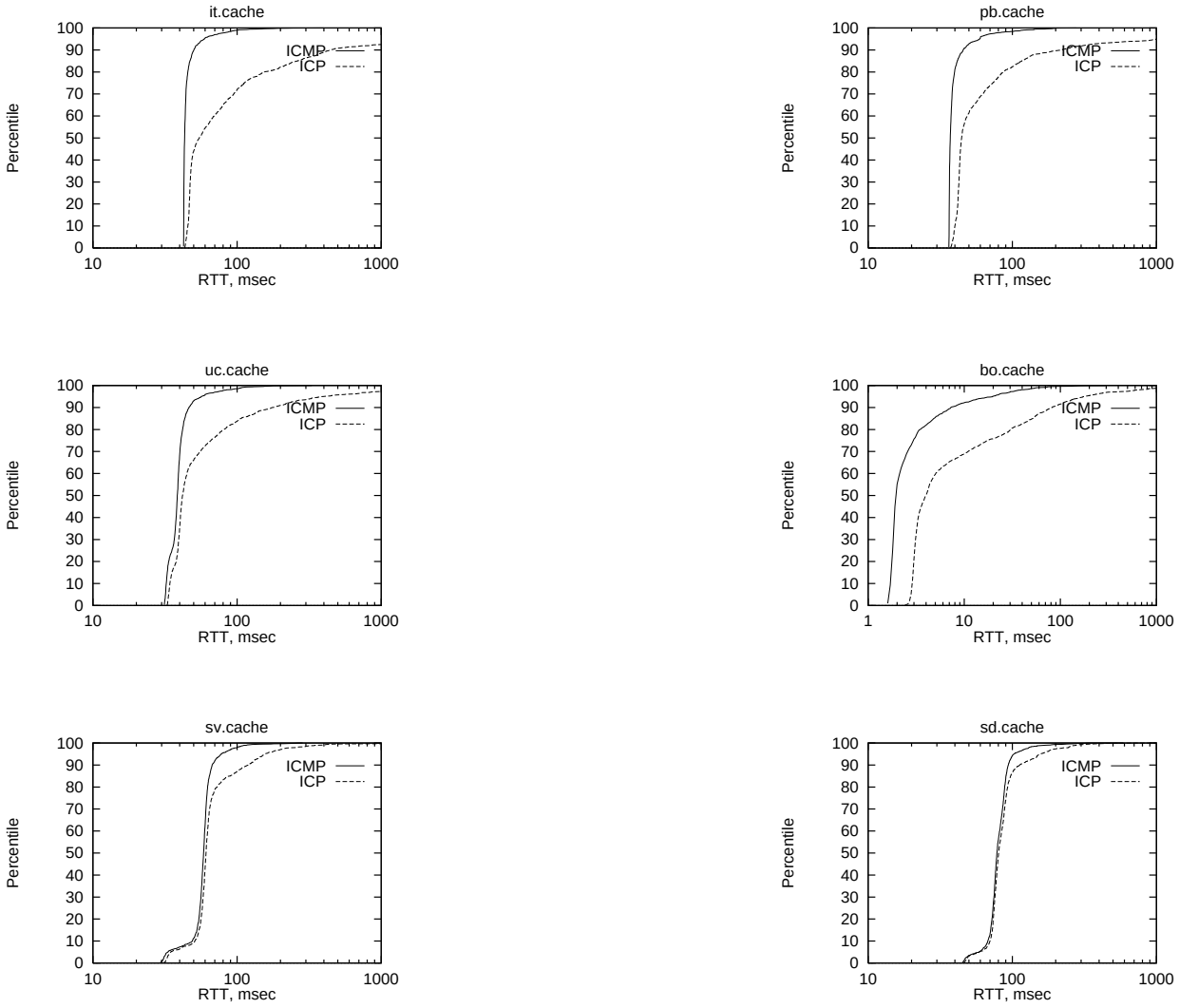

Figure 5: Network and ICP RTT cumulative distributions. Here the ICMP and ICP values are plotted for each host. Note that IT and PB are the busiest caches and show a greater difference between ICP and ICMP RTTs. Similarly, SV and SD are the least busy and have similar ICP and ICMP distributions. The BO cache is located close to the source so the x-axis has a slightly different range.

6 Experiences and Issues with ICP

We might claim that ICP is largely successful due to its widespread deployment. We estimate that there are on the order of 2000 Harvest and Squid caches in operation throughout the Internet. In most cases ICP serves its purpose well, both in load balancing among a group of parent caches, and in locating nearby alternative caches upon misses. However, there are numerous ways in which ICP fails to meet some of the demands being placed upon it.

6.1 Siblings vs. Parents

Recall that the desire to form sibling relationships is a significant factor in the motivation to use ICP. In the beginning, the parent/sibling distinction was clear (or so we thought). Our initial model, although never stated explicitly, was quite simple: sibling caches would all be a part of a single organization, perhaps an ISP or a company.

In reality, however, people want to create more complex relationships. In Europe we see quite a few bilateral cache peerings. When connecting a pair of national caches, should they be siblings or mutual parents? The mutual parent approach is appropriate in situations where using the other country's top-level cache is likely to yield better performance than fetching directly. In this case requests must be restricted to domains that each side is willing to field. But since a peering is already in place, it would be nice to take advantage of objects that the other cache holds, i.e. a sibling relationship. In essence, we want to be able to treat another cache as a parent for some requests, and as a sibling for others. Squid supports this functionality, where the relationship can depend on the DNS domain of the URL.

An ICP query does not include any parent or sibling designation, so the receiver really has no indication of how the peer cache is configured to use it. This issue becomes important when a cache is willing to serve cache hits to anyone, but only handle cache misses for its own customers. In other words, whether to allow the request depends on if the result is a hit or a miss. To support this functionality, Squid acquired the `miss_access` feature in October of 1996.

In addition to being awkward to implement, *miss access* brings its own complication: it requires that the ICP_QUERY reply be an extremely accurate prediction of the result of a subsequent HTTP request. This prediction is challenging if not impossible since the ICP request cannot convey the full HTTP request (section 3.3). Additionally, there are more types of HTTP request results than there are for ICP. The ICP query reply will either be a hit or miss, but an HTTP request might result in a `304 Not Modified` reply from the origin server. Such a reply is not strictly a hit since the peer needed to forward a conditional request to the source. At the same time, its not strictly a miss either since the local object data is still valid, and the Not-Modified reply required from the origin server is quite small.

6.2 Freshness Parameters

One serious problem for cache hierarchies is mismatched freshness parameters. Consider a cache $\mathcal{C}$ using strict freshness parameters so its users get maximally current data. $\mathcal{C}$ has a sibling $\mathcal{S}$ with less strict freshness parameters. When an object is requested at $\mathcal{C}$, $\mathcal{C}$ might find that $\mathcal{S}$ already has the object via an ICP query and ICP_HIT response. $\mathcal{C}$ then retrieves the object from $\mathcal{S}$.

In an HTTP/1.0 world, $\mathcal{C}$ (and $\mathcal{C}$'s client) will receive an object that was never subject to its local

freshness rules. Neither HTTP/1.0 nor ICP provides any way to ask only for objects less than a certain age. If the retrieved object is stale by $\mathcal{C}$'s rules, it will be removed from $\mathcal{C}$'s cache, but subsequently fetched from $\mathcal{S}$ so long as it remains fresh there. This configuration miscoupling problem is a significant deterrent to establishing both parent and sibling relationships.

HTTP/1.1 provides numerous request headers to specify freshness requirements, which actually introduces a different problem for cache hierarchies: ICP still does not include any age information, neither in query nor reply. So $\mathcal{S}$ may return an ICP_HIT if its copy of the object is fresh by its configuration parameters, but the subsequent HTTP request may result in a cache miss due to `Cache-control:` headers originated by $\mathcal{C}$ or by $\mathcal{C}$'s client. Situations now emerge where the ICP reply no longer matches the HTTP request result.

6.3 Hit or Miss?

In the end, the fundamental problem is that the ICP query does not provide enough information to accurately predict whether the HTTP request will be a hit or miss. In fact, the current ICP Internet Draft is very vague on this subject. What does ICP_HIT really mean? Does it mean "I know a little about that URL and have some copy of the object?" Or does it mean "I have a valid copy of that object and you are allowed to get it from me?"

There are a couple of quick-fix modifications to ICP that could fix the freshness problem.

- Include freshness requirements in the ICP_QUERY. The peer cache would then be responsible for returning ICP_HIT for fresh objects and ICP_MISS for stale ones. Unfortunately HTTP/1.1 has various cache-control options (max-age, max-stale, min-fresh, etc.) that complicate this task, and there is not much unused space in the ICP header.
- Include timestamps in the ICP_HIT reply. This seems to be a better solution than the first. Hopefully we could get away with including only two 32-bit timestamps: the last-modified and expires times. The local cache would take responsibility for selecting the peer that best fulfills the freshness requirements.

Both of these solutions address only the freshness problem. There are in fact many other HTTP request headers that an ICP query can not accommodate, e.g., *ETag*, language encodings, acceptable content types, others we have not yet realized or invented. If ICP's role is to accurately predict hit versus miss, then it should include the entire HTTP request.

6.4 Other Issues

6.4.1 ICP_DECHO

Integrating non-ICP-speaking caches into an ICP-based hierarchy relies on the UDP echo service of peer hosts. This technique can somewhat adequately gauge network connectivity, but provides no indication that the cache is actually running and accepting HTTP requests. Non-ICP cache products could offer minimal ICP support by simply providing a UDP socket that echos datagrams back to their source.

6.4.2 Security

As with all networking applications, security is often an issue. ICP assumes the application receives an accurate IP address for the message datagram. Therefore, it is susceptible to IP spoofing attacks. Falsifying, altering, inserting, or blocking ICP messages could cause HTTP requests to be forwarded (or not forwarded) to certain neighbors. If the neighbor cache has also been compromised, then it could serve bogus content and pollute a cache hierarchy.

A likely method of attack would be to attempt to poison a cache with false data. Generating fake ICP ICP_HIT or ICP_MISS messages will not succeed in poisoning since the object data is retrieved via HTTP. However, the ICP_HIT_OBJ feature does pose a potential problem if the source IP address can be spoofed (another reason to avoid using ICP_HIT_OBJ). Nonetheless, a cache should verify the address and port number of every ICP message it receives, and only accept ICP replies from known peers.

The ICP Internet Draft [48] has additional details on security concerns.

6.4.3 ICP Scales Poorly

ICP has poor scaling characteristics. It is not really practical to send more than about five ICP queries for each HTTP request, if only because it increases the chance of losing at least one ICP reply and therefore frequently incurring a timeout. Another reason is that (with unicast) the number of exchanged ICP messages is directly proportional to the number of peers. Multicast (section 7.2) could help alleviate this problem, although it will not reduce the number of ICP reply messages. Finally, caches should spend most of their time handling HTTP requests, not ICP. A peer relationship that yields an HTTP to ICP request ratio of 1:100 probably merits re-evaluation.

Operational Web caches for large ISPs involve a considerable number of other scaling issues as well, beyond those of ICP. The most critical parameters are the request rate and cache size. The latter parameter reflects an upper limit on the amount of data that a cache can manage, beyond which the cache will spend too much time doing administrative tasks (e.g. deleting old cache files). Also, the relationship between cache size and hit rate is not linear. Increases in cache disk space eventually reach a point of diminishing returns, and this point depends upon the workload profile.

Another interesting scaling parameter is the acceptable number of levels in a cache hierarchy (or mesh). Empirical results by Baentsch *et al.* [50] indicate that three levels (or cache-hops) is acceptable, but four gives noticeable delay.

6.5 URLs

Squid and Harvest use URLs in ICP messages because they also use the URL as a part of the cache key. This allows the ICP queries to be handled very quickly and efficiently. However, using URLs has a negative aspect as well: URLs can vary greatly in length. Usually they are less than 100 bytes, but occasionally they can grow to 4 kbytes and larger (e.g. for complex CGI scripts). ICP does not currently impose any limits on URL length. For that matter, it does not limit the size of the ICP message. In the future, ICP may support sending an MD5 [51] hash of the URL instead of the URL itself.

7 Recent Work

7.1 Probing the Network

Earlier we alluded to some new techniques to improve the peer selection process. Squid supports domain restrictions on peers to route requests in the right general direction. 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 doomed. To do the job properly, a cache must have knowledge of the underlying network topology.

We have recently implemented an approach where Squid acquires topology data with ICMP. Over time, Squid builds a table of hop counts and round-trip times for the server hosts 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 (in the unused PADDING field).[10] The cache collecting ICP replies uses the network measurements to select the best peer, ideally the one most toward the origin server.

Initially we thought that hop count would make a good metric for peer selection; the optimal peer would be that with the lowest hop count to the origin server. Unfortunately, we can only estimate the hop count, using the `ip->ttl` field from the ICMP echo reply and guessing at likely starting TTL values. We might consider a technique similar to the one used by *traceroute*, but at the outset this seems to generate an excessive amount of ICMP traffic and considerably more difficult to implement.

Instead, we are now using the RTT (averaged over time) as the selection metric, which also provides an indication of congestion from peer to source. Whereas the hop count is likely to remain constant over time (and would therefore be termed *static* by Carter and Crovella [42]), the round-trip time can vary widely between measurements. We use decaying averaging to achieve some stability while still adjusting to changing network conditions.

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 than any peers to some sources; 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. The selection process also becomes more complex as the number of selection criteria increases. Currently we have: type of peer, hit or miss, peer-to-peer RTTs, artificial weights, DNS domain restrictions, and the most recent additions of peer-to-source RTTs and hop counts. Meeting the vast range of

[10] Since we are deviating from the standard interpretation of header fields, support for this feature must be indicated by setting an OPTIONS bit.

configuration requirements with a single peer selection policy is growing even more challenging. Developing mechanisms to make the process more configurable and intuitive is an important topic for future study.

This approach is different than the one proposed by Floyd, Zhang, and Jacobson [44], although the goal is the same. In both cases we want to know which of a set of possible parent caches is closer to the origin server. They propose to base this decision on an IP routing table, perhaps from a nearby router. Unlike our measurement-based approach, their technique does not account for path characteristics, such as bandwidth and congestion; rather cache request routing just follows IP routing. The additional data gleaned from ICMP measurements can allow caches to avoid trouble areas along what may be default IP routing paths. The correctness of either approach is debatable.

7.2 Multicast

Multicast has been proposed by numerous individuals [44, 52] as a solution to some of ICP's problems, such as scaling and configuration. Ideally multicast can reduce the amount of ICP traffic a cache must send,[11] and also eliminate duplicate messages traversing a single link. While multicast purports to solve some problems, it also exposes some new ones.

In a group of cooperating caches, each member must currently be specified in every other configuration file. It would be nice if we could simply specify one multicast group address and group members could join or leave at will. Unfortunately, joining a multicast group does not require any special privileges or authentication. Should we implicitly trust any member of a multicast group? For Web caching, we can not, so even when multicast is used, all (trusted) group members must still be specified. A similar problem is that multicast allows others to very easily snoop on ICP requests, providing an easy way to receive a list of URLs actively being retrieved.

Initially we were unsure whether ICP replies should be returned via unicast or multicast, or whether to send ICP_MISS replies at all. Recall that Squid counts the number of replies received in response to a query, so a lack of ICP_MISS replies would complicate that process. We believe that it is best to return ICP replies via unicast. This prevents other group members from receiving the reply messages, and gives the querying cache an some information about the unicast path between the pair.

Multicast also brings difficulties with DNS domain restrictions on peers. With unicast it is trivial to query only a specific subset of peer caches. Doing so with multicast either requires each potential subset of peers to join separate multicast groups, or use only one group and ignore replies from peers based on the request and domain restrictions. Squid has used the latter approach.

A final disadvantage to multicast is the lack of widely deployed and stable infrastructure. In most cases, Web caches operate as mission critical services. Many cache operators are unwilling to rely on the current, operationally tenuous MBone[12] for Web caching.

[11] To be specific, the number of times the application must call `sendto()`

[12] `http://www.best.com/~prince/techinfo/mbone.html`

8 History of ICP

8.1 Harvest Research Software

ICP was originally developed as a component of the Harvest cache project, funded primarily by an ARPA research grant during 1993–1995. Researchers at the University of Southern California and the University of Colorado developed the first version of the hierarchical Web caching software in 1994 [22]. The Harvest Web cache software (called *Cached*) began to acquire real users during 1995. Late that year, development slowed considerably as project members migrated toward industry jobs. A few project members formed a company to sell a commercial-strength version of the Harvest cache [23].

The early Harvest cache used a very simple caching model:

- Only GET requests should be cached. There is no need to query peers for non-GET requests (i.e. POST) since other caches won't have them either. Because of this, an ICP message does not include the request method, only a URL.
- The research version did not support *If-Modified-Since* requests. As the cache added a new object, it calculated a Time-To-Live for the object, and released the object when the TTL expired. Until an object expired, subsequent requests would always result in a hit. In other words, any non-expired object in the cache was considered valid. An expired object was always purged from the cache, regardless of whether or not it had actually changed.
- Only HTTP/1.0 servers were in use. The only mechanism for maintaining cache consistency was the *If-Modified-Since* conditional request.

During the Harvest research project, ICP remained largely unchanged. At that time, the ICP message format included an eight-byte authentication field, but the authentication mechanism was never implemented. The Harvest research cache software always set the ICP version number to two (2).

8.2 Squid

Because the NSF-funded NLANR caching project [24] required a research version of the software for code experimentation, project members continued development on a derivative of the Harvest cache software, Squid [25], with considerable assistance from the user community. Our description and discussion of ICP in this paper derives from our experiences with Squid.

In the time we have been working on Squid, a number of new features have been added to its ICP implementation. These include the notion of private objects, ICMP support, ICP_HIT_OBJ, and ICP_MISS_NOFETCH. Like the early Harvest cache, Squid also uses two (2) as the ICP version number.

8.3 Netcache

The commercial Harvest cache has made some modifications to ICP as well. For differentiation, it uses three (3) as the ICP version number. For efficient support of *If-Modified-Since* requests in a cache hierarchy, it uses the former eight-byte authentication field to hold two object timestamps. We are not aware of any other ICP-related changes to the commercial version of Harvest.

8.4 ICP Compatibility

Current versions of Harvest and Squid interoperate very well, although previous versions have had problems. Since the two use different ICP version identifiers, it is simple to account for the differences. An IETF working group is emerging to standardize ICP for interoperability among multiple cache implementations.

9 Conclusions

The Internet Cache Protocol has been in use in a global Internet caching hierarchy for approximately two years. As originally conceived, ICP provides *hints* about the location of Web objects. At the same time, it probes the network for good connectivity, and it performs this task quite well. However, many administrators now seek to expand the role of ICP into *cache policy* expression and enforcement, which turns out to be difficult with such an (intentionally) simple protocol. To properly implement ICP as a cache policy protocol will inevitably require ICP messages to exchange full HTTP requests.

Our measurements on ICP delays merit further study. We observe a linear (or perhaps quadratic) relationship between cache load and average ICP delays. We need to make these measurements against caches operating near full capacity to see if the trend holds.

In addition to evolving the protocol itself, there are numerous interesting avenues to explore in the application of ICP. Despite the various obstacles preventing multicast distribution of ICP, we feel that it still holds great promise for alleviating configuration and scaling problems.

9.1 Is it all worth it?

We will be first to admit that configuring hierarchical Web caches is time-consuming and sometimes difficult to coordinate. So what is the benefit from all this trouble? In addition to the 30-50% local hit rates typically seen by most caches [53, 54, 1], we find that approximately another 10% of requests will be cache hits in neighbor caches.[13] Does this extra margin justify joining a cache hierarchy? We believe the answer to this depends on where you are and on the quality of your Internet service. In the U.S. we are lucky to enjoy relatively high amounts of bandwidth and overall good connectivity. In other countries, the situation is quite different.

In interacting on a daily basis with people from all over the world, the feedback we receive indicates that cache administrators are generally pleased that software such as Harvest and Squid allows them to unite in cooperative meshes of caches. An additional 10% may not seem like much to the bandwidth-blessed, but when you have to wait often over ten minutes to view a single Web page, every little bit helps.

Can we expect to build a global caching hierarchy without ICP? For the foreseeable future, we do not see another viable alternative. In the decentralized, untamed, global Internet, ICP provides flexible, powerful, necessary glue among non-autonomous Web caches.

[13] `http://ircache.nlanr.net/Cache/Statistics/Hierarchy/`

References

[1] M. Abrams, C. R. Standridge, G. Abdulla, S. Williams, and E. A. Fox, "Caching proxies: Limitations and potentials," in *Proceedings of the Fourth Interntional Conference on the WWW*, (Boston, MA), December 1995.

[2] N. Smith, "What can archives offer the world wide web," September 1994. `http://www.hensa.ac.uk/www94/`.

[3] D. Neal, "The harvest object cache in new zealand," in *Proccedings of the World Wide Web Conference*, May 1995.

[4] D. Povey and J. Harrison, "A distributed internet cache," in *Proceedings of the 20th Australasian Computer Science Conference (to appear)*, February 1997. `http://www.psy.uq.edu.au:8080/~dean/project/`.

[5] A. Cormack, "Caching on janet: Acn report," September 1996. `http://www.psy.uq.edu.au:8080/~dean/project/`.

[6] A. J. Flavell, "Briefing at glasgow university ppe group," July 1996. `http://d1.ph.gla.ac.uk/~flavell/cache.html`.

[7] D. Marwood, "Squid proxy analysis, presented at NLANR cache workshop 1997," April 1997. `http://www.cs.ubc.ca/spider/marwood/Projects/SPA/Report/Report.html`.

[8] V. Almeida, A. Bestavros, M. Crovella, and A. de Oliveira, "Characterizing Reference Locality in the WWW," in *Proceedings of PDIS'96: The IEEE Conference on Parallel and Distributed Information Systems*, (Miami Beach, Florida), December 1996.

[9] T. Berners-Lee, R. Fielding, and H. Frystyk, "Hypertext transport protocol – HTTP/1.0," *Network Working Group RFC 1945*, May 1996. `http://ds.internic.net/rfc/rfc1945.txt`.

[10] R. Fielding *et al.*, "Hypertext transport protocol – HTTP/1.1," *Network Working Group RFC 2068*, January 1997. `http://ds.internic.net/rfc/rfc2068.txt`.

[11] T. Berners-Lee *et al.*, "Uniform resource locators (URL)," *Network Working Group RFC 1738*, December 1994. `http://ds.internic.net/rfc/rfc1738.txt`.

[12] P. Mockapetris, "Domain names - concepts and facilities," *Network Working Group RFC 1034*, November 1987. `http://ds.internic.net/rfc/rfc1034.txt`.

[13] P. Mockapetris, "Domain names - implementation and specification," *Network Working Group RFC 1035*, November 1987. `http://ds.internic.net/rfc/rfc1035.txt`.

[14] V. Fuller, T. Li, J. Yu, and K. Varadhan, "Classless inter-domain routing (CIDR): an address assignment and aggregation strategy," *Network Working Group RFC 1519*, September 1993. `http://ds.internic.net/rfc/rfc1519.txt`.

[15] M. A. Blaze, *Caching in Large Scale Distributed File Systems*. PhD thesis, Princeton University, 1993. `http://ncstrl.cs.princeton.edu/Dienst/UI/2.0/Describe/ncstrl.princeton%2fTR-397-92`.

[16] M. D. Dahlin, C. J. Mather, R. Y. Wang, T. E. Anderson, and D. A. Patterson, "A quantitative analysis of cache policies for scalable network file systems," *Performance Evaluation Review*, vol. 32, p. 150, May 1994. `ftp://ftp.cs.berkeley.edu/ucb/people/tea/xfs.ps`.

[17] D. Muntz and P. Honeyman, "Multilevel caching in distributed file systems - or - your cache ain't nuthin' but trash," in *Proceedings of the USENIX Winter Conference*, pp. 305–313, January 1992.

[18] P. B. Austin, K. A. Murray, and A. J. Wellings, "File system caching in large point-to-point networks," *Software Engineering Journal*, vol. 7, pp. 65–80, January 1992.

[19] M. N. Nelson, B. B. Welch, and J. K. Ousterhout, "Caching in the sprite network file system," *AMC Transactions on computer Systems*, vol. 6, pp. 135–154, February 1988.

[20] A. Lutonen, H. F. Nielsen, and T. Berners-Lee, "Cern httpd," July 1996. `http://www.w3.org/pub/WWW/Daemon/Status.html`.

[21] Bowman, Danzig, Hardy, Manber, Schwartz, and Wessels, "Harvest: A scalable, customizable discovery and access system.," Tech. Rep. CU-CS-732-94, Department of Computer Science, University of Colorado, August 1994. `ftp://ftp.cs.colorado.edu/pub/cs/techreports/schwartz/Harvest.Jour.ps.Z`.

[22] A. Chankhunthod, P. B. Danzig, C. Neerdaels, M. F. Schwartz, and K. J. Worrell, "A hierarchical internet object cache," Tech. Rep. 95-611, University of Southern California, March 1995. `http://catarina.usc.edu/danzig/cache.ps`.

[23] N. Appliance, "Netcache proxy." `http://www.netapp.com/products/level3/netcache/datasheet.html`.

[24] D. Wessels, K. Claffy, and H.-W. Braun, "NLANR prototype web caching system." Research project funded by the National Science Foundation. `http://ircache.nlanr.net/`.

[25] D. Wessels, "Squid internet object cache." `http://squid.nlanr.net/`.

[26] N. Communications, "Netscape proxy." `http://home.netscape.com/comprod/server_central/product/proxy/index.html`.

[27] Microsoft, "Microsoft proxy." `http://www.microsoft.com/proxy/`.

[28] S. Glassman, "A caching relay for the world wide web," in *Proceedings of the First International WWW Conference*, pp. 69–76, May 1994. `http://www.research.digital.com/SRC/personal/Steve_Glassman/CachingTheWeb.html`.

[29] R. Jones, "Digital's world-wide web server: A case study," in *Proceedings of the First International WWW Conference*, May 1994. `http://info.cern.ch/PapersWWW94/rjones.ps`.

[30] B. Nowicki, "NFS: Network file system protocol specification," *Network Working Group RFC 1094*, March 1989. `ftp://ftp.internic.net/rfc/rfc1094.txt`.

[31] J. Gwertzman, "Autonomous replication in wide-area distributed systems." Senior Thesis, April 1995. `http://www.eecs.harvard.edu/~vino/web/push.cache/`.

[32] A. López-Ortiz and D. M. Germán, "A multicollaborative push-caching HTTP protocol for the WWW." `http://daisy.uwaterloo.ca/~alopez-o/cspap/cache/Overview.html`.

[33] J. Gwertzman and M. Seltzer, "The case for geographical push-caching," in *Fifth Annual Workshop on Hot Operating Systems*, pp. 51–55, May 1995. http://www.eecs.harvard.edu/~vino/web/hotos.ps.

[34] D. L. Cohen, "AFS: NFS on steroids," *LAN Technology*, vol. 9, pp. 51–62, March 1993.

[35] M. Baentsch, G. Miller, and P. Sturm, "Introducing application-level replication and naming into today's web," in *Proceedings of the Fifth International World-Wide Web Conference, Paris, France*, May 1996. http://www5conf.inria.fr/fich_html/papers/P3/Overview.html.

[36] V. N. Padmanabhan and J. C. Mogul, "Using predictive prefetching to improve world wide web latency," *ACM SIGCOMM Computer Communication Review*, July 1996. http://daedalus.cs.berkeley.edu/publications/ccr-july96.ps.gz.

[37] H. Inoue and K. Chinen, "Catalyst mode of Wcol." http://shika.aist-nara.ac.jp/products/wcol/cuckoo.html.

[38] K. ichi Chinen and S. Yamaguchi, "An interactive prefetching proxy server for improvements of WWW latency," in *INET 97, Kuala Lumpur, Malaysia*, June 1997. http://www.isoc.org/inet97/proceedings/A1/A1_3.HTM.

[39] C. Davis, P. Vixie, T. Goodwin, and I. Dickinson, "A means for expression location information in the domain name system," *Network Working Group RFC 1876*, January 1996. http://ds.internic.net/rfc/rfc1876.txt.

[40] K. Moore, J. Cox, and S. Green, "Sonar - a network proximity service." http://www.netlib.org/utk/projects/sonar/.

[41] P. Francis, "Host proximity service (hops)." http://www.ingrid.org/hops/wp.html.

[42] R. L. Carter and M. E. Crovella, "Dynamic server selection using bandwidth probing in wide-area networks," Tech. Rep. TR-96-007, Boston University Computer Science Department, March 1996. ftp://cs-ftp.bu.edu/techreports/96-007-dss-using-bandwidth.ps.Z.

[43] A. J. D. Guyton and M. F. Schwartz, "Locating nearby copies of replicated internet servers," in *Proceedings of ACM SIGCOMM '95, Cambridge, MA*, pp. 288–298, August 1995. ftp://ftp.cs.colorado.edu/pub/cs/techreports/schwartz/ReplSvrLoc.ps.Z.

[44] S. Floyd, L. Zhang, and V. Jacobson, "Adaptive web caching." DARPA-funded Research Project, May 1997. http://irl.cs.ucla.edu/awc.html.

[45] S. Gadde, M. Rabinovich, and J. Chase, "Reduce, reuse, recycle: An approach to building large internet caches," in *Workshop on Hot Topics in Operating Systems (HotOS)*, April 1997. http://www.cs.duke.edu/ari/cisi/crisp-recycle/.

[46] Various, "Icp working group [sic] mailing list archive," 1997. http://squid.nlanr.net/Mail-Archive/icp-wg/archive/.

[47] D. Wessels and K. Claffy, "Internet cache protocol (ICP), version 2," *draft-wessels-icp-v2-03.txt*, July 1997. http://ds.internic.net/internet-drafts/draft-wessels-icp-v2-03.txt.

[48] D. Wessels and K. Claffy, "Application of internet cache protocol (ICP), version 2," *draft-wessels-icp-v2-appl-02.txt*, July 1997. `http://ds.internic.net/internet-drafts/draft-wessels-icp-v2-appl-03.txt`.

[49] CERT, "UDP port denial-of-service attack." `ftp://info.cert.org/pub/cert_advisories/CA-96.01.UDP_service_denial`.

[50] M. Baentsch, L. Baum, G. Molter, S. Rothkugel, and P. Sturm, "World-wide web caching – the application level view of the internet," *IEEE Communications Magazine*, vol. 35, June 1997. `http://www.uni-kl.de/AG-Nehmer/baentsch/Publications.html`.

[51] R. L. Rivest, "The MD5 message-digest algorithm," *Network Working Group RFC 1321*, April 1992. `ftp://ftp.internic.net/rfc/rfc1321.txt`.

[52] Various, "Squid-users mailing list archives," 1996-7. `http://squid.nlanr.net/Squid/Mail-Archive/squid-users/`.

[53] D. Wessels and K. Claffy, "NLANR cache statistics." Daily statistics for the NLANR caches. `http://ircache.nlanr.net/Cache/Statistics/`.

[54] Various, "Links to publicly available proxy cache statistics." `http://ircache.nlanr.net/Cache/cache-stats-links.html/`.

[55] V. N. Padmanabhan and J. C. Mogul, "Improving HTTP latency," in *Proceedings of the Second International WWW Conference*, 1994. `http://www.ncsa.uiuc.edu/SDG/IT94/Proceedings/DDay/mogul/HTTPLatency.html`.

[56] R. P. Wooster and M. Abrams, "Proxy caching that estimates page load delays," in *Proceedings of the Sixth International WWW Conference*, April 1997. `http://www6.nttlabs.com/index_by_topic.html#server`.