

# Cachability of Web Objects

Xiaohui Zhang

xiaohui@cs.bu.edu

Computer Science Department

Boston University

Boston, MA 02115 USA

*Abstract*—Much work on the performance of Web proxy caching has focused on high-level metrics such as hit rate and byte hit rate, but has ignored all the information related to the cachability of Web objects. Uncachable objects include those fetched by dynamic requests, objects with uncacheable HTTP [1,2] status code, objects with the uncacheable HTTP header, objects with an HTTP 1.0 cookie [3], and objects without a last-modified header. Although some researchers filter the Web traces before they use them for analysis or simulation, many do not have a comprehensive understanding of the cachability of Web objects.

In this paper we evaluate all the reasons that a Web object might be uncacheable. We use traces from NLANR [4]. Since these traces do not contain HTTP header information, we replay them using request generator to get the response header information. We find that between 15% and 40% of Web objects in our traces can not be cached by a Web proxy server. We use a LRU simulator to show the performance gap when the cachability is either considered or not. We show the characteristics of the cachable data set and find that all its characteristics are fairly similar to that of total data set. Finally, we present some additional results for the cachable and total data set: (1) The main reasons for uncacheability are: dynamic requests, responses without last-modified header, responses with HTTP "302 Moved Temporarily" status code, and responses with a HTTP/1.0 cookie. (2) The cachability of Web objects can not be ignored in simulation because uncacheable objects comprise a huge percentage of the total trace. Simulations without cachability consideration will be misleading.

*Keywords*—Web Caching, HTTP, WWW

## I. INTRODUCTION

With the rapid growth of World Wide Web (WWW), people have devised many techniques to improve its performance. One of them is Web caching. A Web cache sits somewhere between Web servers and clients, watching requests for HTML, image, audio, video and all the other kinds of objects that pass by, and saving a copy of response for itself. Later, if there is another request for the same object, it will use the copy that it has saved, instead of requesting it from origin server. There are two advantages to the use of a Web cache. First, it can reduce latency. Because the request is satisfied from the cache, which is closer to the client than the origin server, it takes much less time for the client to get the object (especially in heavy network traffic or low bandwidth). Second, using a Web cache can lower network traffic. Because objects are served mostly from the Web cache instead of the origin server and Web cache is much closer to clients, the traffic can be reduced dramatically.

There are two kinds of Web caches: browser caches and proxy caches. A browser cache works with the browser in local hard disk. Whenever the browser gets an object from Web server, it keeps a copy on its local disk. For every request, it first checks the browser cache to see whether there is a hit; if not, it will send the request to Web server. A proxy cache works on the same

principle, but on a much larger scale. A proxy is a server between the Internet and its clients. The client should pass through the proxy before it goes out of the local network, and everything from outside the network should pass through the proxy. Because proxy caches usually serve a large number of users and one object requested by one client would be possibly requested by other clients, it could be effective at reducing latency and lower traffic.

In a recent paper [5], researchers in AT&T found it is much better to cache the persistent connection rather than cache data. They recommend that a proxy should keep open a certain number of persistent connections with the most popular Web servers so that the delay for setting up a connection and slow start would be eliminated.

In this paper we evaluate all the information that affects the cachability of Web objects. We discuss the use of the name of the URL, the HTTP method, the HTTP status code, and the HTTP response header to find whether a Web object is cachable. We find that once all these factors are considered, about 15%-40% Web objects are uncacheable. Thus if researchers do not consider the uncacheability in their simulations, their research results may be misleading.

The rest of this paper is organized as follows. Section 2 presents the related work. Section 3 describes the data collection, and Section 4 covers the reasons that objects may be uncacheable. Section 5 presents the simulation results. Section 6 focuses on the characteristics of cachable Web objects. Section 7 presents some additional statistics. Section 8 concludes.

## II. RELATED WORK

Much work has been done on Web proxy caching. First many replacement algorithms have been studied: LRU, LFF, FIFO, LFU, Size [12], Hyper-G [12], LRU-Threshold [13], Log (size)+LRU [13], Lowest-Latency-First [14], Hybrid [14], GreedyDual-Size [15]. Using trace driven simulation, these studies measure performance metrics such as hit rate and byte hit rate. Second, many people working on the scalable proxy architectures. For example Squid [11] can be arranged hierarchically for an improvement in response times and a reduction in bandwidth usage. Finally there are a number of commercial products: Squid (NLANR), Dyna Cache (Inforlibria) [16], Cache Engine (Cisco) [17], Traffic Server (Inktomi) [18].

A number of papers have done statistical measurements similar to ours. One of the earliest was performed by University of California at Berkeley (UCB) in 1996 (Gribble and Brewer [6]) in which they collected traces from Home IP service at UCB for 45 consecutive days (including 24 million HTTP requests).

Since Xiaohui Zhang finished this Master thesis under the instruction of Prof. Mark Crovella, he has joined the innovations lab in Nortel Networks. (xiaohui@nortelnetworks.com)

They analyzed the cache pragmas of HTTP, including "Pragma: no-cache", "Cache-Control", "If-Modified-Since", "Expires" and "Last-Modified". They also analyzed the distribution of file type and size. However they did not look at all HTTP response status codes, and HTTP methods. They also did not discuss cookies, which make an object uncachable in HTTP 1.0. Finally, they did not discuss the increasing proportion of dynamic objects in the Internet, such as Active Server Page (ASP) files.

Two related studies were done at AT&T Research Labs. Douglass et al. [7] collected traces at the Internet connection points for two large corporations, AT&T and DEC, representing 17 days and 950,000 records. They examined several attributes: request times, last modification times, ages, and modification intervals. They also obtained statistics for last modification time, cache pragmas and cookies. Since their focus is not on cachability, their measurements can be compared in certain way with ours. Feldmann et al. [5] collected traces from both dialup modems to a commercial ISP and clients on a fast research LAN. They obtained more statistics on the reasons for uncachability: whether a Cookie was present, whether URL had a '?', Client Cache-Control, Neither GET nor HEAD, Authorization present, Server Cache-Control. However they did not look at all HTTP response status codes. They also did not mention the last-modified header in the response, which is essential for browser and proxy server to verify the object's freshness. For the cases when a cookie was present, they did mention about the difference between HTTP/1.0 and HTTP/1.1, but there was no discussion about HTTP/1.0 cookies and HTTP/1.1 cookies. For the Cache-Control cases, they did not distinguish the different Cache-Control pragmas since some responses with Cache-Control are still cachable. For dynamic files, they only looked at CGI, and "?" but omitted ASP, PL and "=".

Our work is similar to the previous but we consider more reasons and treat uncachability more comprehensively according to HTTP protocol [1,2] and HTTP State Management Mechanism [3]. We consider the reasons for uncachability to be: dynamic URLs, uncachable HTTP methods, uncachable HTTP response status codes, and uncachable HTTP response headers. We also try to discover the causes behind some of these reasons, such as why the server does not put the Last-Modified header with the file.

### III. DATA COLLECTION

We obtained the raw trace data from National Laboratory for Applied Network Research (NLNAR) [4]. The trace log contains: timestamp, elapsed time, client IP (sanitized), cache result code, HTTP status code, bytes transferred, HTTP method, URL, MIME type, and peer status code.

NLNAR runs the Squid [11] proxy caching software. Squid uses hierarchical proxy architecture. In this architecture, the relation between two proxies can be defined as parent or sibling. If you configure your cache to have only siblings, your cache will send UDP queries to the list of siblings. If they don't have the page, Squid will connect directly to the origin Web server. If you configure your cache to have a parent it means that if you don't have the object, and none of your siblings do, it will open a TCP connection to the parent server and ask it to get the page on your behalf. Since it's a TCP connection, this par-

ent will possibly go through the motions of checking its parents and siblings if it doesn't have the page on disk. Currently there are ten major proxy servers all over the United States: sd, sv, uc, pb, lj, rtp, bo2, pa, bo1 and sj. The total daily requests are about 10 million. They publish last seven days' traces on the Web site: ftp://ircache.nlanr.net/. All these servers run on 266MHz Digital Alpha processors, each supported by 256MB of memory, 10GB of hard disk space, and FDDI and Ethernet interfaces. We picked one day's trace (Apr 28, 1999) from six proxy servers: pa, pb, bo1, bo2, sd, sv. In these servers, only pa is a single node; the rest are configured as mutual parents. Later we will find this will greatly affect the uncachability percentage. In this section and next section, we will focus on pa trace which contains 699,563 total requests and 5,909.81 MB of data.

Since there is no HTTP header information for the traces we implemented a trace generator to request all the header information for each trace entry. The software reads the Universal Resource Location (URL) information from the trace and opens a TCP connection with the Web server at port 80. After writing "HEAD [URL] HTTP/1.0", it will read the response header into local buffer. There are 100 parallel threads (the generator can get at least 1 million response headers in two hours in our lab). We drive this trace generator on a SGI workstation running IRIX 6.5. We replayed the trace on Apr 29, 1999, which is just the day after the day the trace was logged, so the trace is fresh enough for us to get all valid response headers.

### IV. UNCACHABLE REASONS

There are many reasons a Web object can be uncachable. We divide them into four categories: 1) dynamic URLs, 2) the HTTP Methods, 3) the HTTP status codes and 4) the HTTP headers.

Here we briefly review the process that a proxy uses to determine when to serve an object from the local cache, if it is available. According to the protocols of HTTP 1.0 [2] and HTTP 1.1 [1], the most common rules that are followed for a particular request are:

- 1) If the object is fetched with an uncachable HTTP header, it is no use to cache it. Even the proxy caches it, all subsequent requests still have to go to the original Web server to fetch the specific document. These headers are: "pragma: no-cache", "Authorization", "Cache-Control: no-cache", "Cache-Control: private", "Cache-Control: no-store" and "Set-Cookie" (for HTTP/1.0).
- 2) If the object is fetched with an uncachable HTTP status code, such as "302 Moved Temporarily", "206 Partial Content", it should not be cached. For the complete list of uncachable HTTP status code, please see Table V. For these objects, subsequent requests have to go to the original Web server.
- 3) If the object is fetched with an uncachable HTTP method: "POST", "PUT", "GET", "DELETE", "OPTION" and "DELETE", it should not be cached. For these objects, subsequent requests have to go to the original Web server.
- 4) A cached object is considered fresh, which means it could be sent to a client without checking with the original server, if:
  - It has an expiry date or other age-controlling directive set, and is still within the fresh period.
  - A browser cache has already seen the object and has been

set to check once per session. This is a heuristic algorithm for browser. It is assumed that the object will not change during one browser session, from user start the browser until user quit it.

– Although not part of the HTTP specification, some proxies follow this rule: a proxy cache has seen the object recently, and it was modified relatively long ago. This is a heuristic algorithm for the proxy server so that it need not check the validation for every request. If an object was modified long time ago and is not changed recently, it is assumed that it is the same at this time. For example, in Squid [11], this heuristic algorithm is part of its refresh algorithm.

Fresh objects are served directly from the cache, without checking with the origin server.

5) If an object is thought to be stale, it should be validated by the original server. The server will send "304 Not-Modified" header to tell the cache that the copy is till good. If an object has no validation information, such as Last-Modified Date or Etag, the object has to be fetched from original server. Usually a dynamic URL has no such kind validation information, since the URL is changed very quickly and it is no use for the cache to save a copy.

6) If there is a miss in the cache, the object has to be fetched from the original server.

Here we review how the server and browser communicate with each other if they all set up with different HTTP versions. This is very important since in HTTP/1.1 there is more information about cache control. But when we looked at how the server and browser set up the HTTP protocol, we will find that there is a difficulty involved in the transition to HTTP/1.1.

#### 1) HTTP/1.0 Server

The server can only understand the requests in HTTP/1.0 format. All the requests in HTTP/1.1 format will be return back for "505 Version Not Supported".

#### 2) HTTP/1.1 Server

The server can understand both HTTP/1.0 and HTTP/1.1 requests. It will send back the response only in HTTP/1.1 format. This seems it doesn't make sense, but in fact, it is easy for the user upgrade their browser from HTTP/1.0 to HTTP/1.1 and currently relatively few people are still using early versions of Netscape or IE. It is not necessary for server to support both versions of the HTTP protocols.

#### 3) HTTP/1.0 Browser

There are very few people using HTTP/1.0 browser. If this version browser is used to access the HTTP/1.1 server, the browser probably will not understand some HTTP/1.1 responses.

#### 4) HTTP/1.1 Browser

Currently IE 4.0 and Netscape 4.5, which are most popular in Internet, use HTTP/1.1. But when they make first request to a server, they will only send request in HTTP/1.0 in case the Web server is running in HTTP/1.0. So many cache-control semantics in HTTP/1.1 can not be used. But the browser can understand both HTTP/1.0 and HTTP/1.1 response. If most servers were upgraded to HTTP/1.1, the browser could send the request in HTTP/1.1 at first, and fall back to HTTP/1.0 if it fails. Unfortunately, there are still an important percentage (20% according to our measurement, see Section 7.4) of servers running in HTTP/1.0.

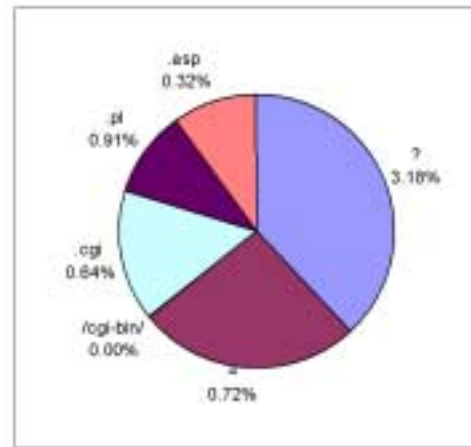


Fig. 1. Requests by Dynamic URL Type

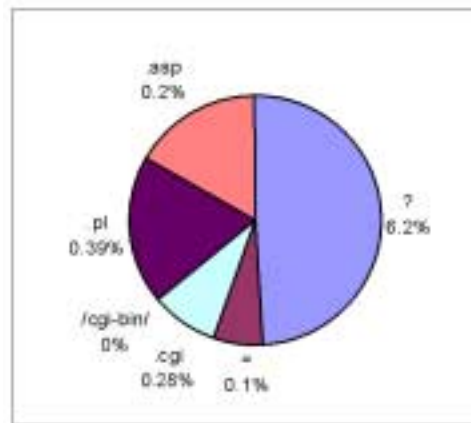


Fig. 2. Bytes by Dynamic URL Type

#### A. Dynamic URLs

A dynamic URL can be identified by the presence of "?", "=", "/cgi-bin/", ".cgi", ".pl", or ".asp".

- "?", "=": Usually these occur together in a query. For example to search "Web Caching" in Yahoo: <http://search.yahoo.com/bin/search?p=Web+Caching>
- "/cgi-bin/", ".cgi": Common Gateway Interface (CGI) file executed by the Web server, usually in the cgi-bin directory
- ".pl": Perl script file
- ".asp": Active Server Page (ASP) is a server based scripting language that is used to build database driven Web sites where the browser may have no scripting at all.

The following is the percentage these types take in the whole trace and the distribution of different kinds of dynamic URLs. The total dynamic URLs comprise 5.3% of the whole trace. CGI files are rare within a /cgi-bin/ directory. Above all the query type takes the largest percentage of the total set of dynamic requests.

Other studies have shown different value for some of these measurements. One study in 1997, from AT&T Research Lab [5] measured the percentage for "?" as 9.85% and the percentage for "/cgi-bin/" as 5.42%. Another paper from University of California at Berkeley [6] measured that there were about 1.0%

CGI requests in 1996. We think the reasons for this difference are probably:

- 1) With the growth of the Internet, the percentage of CGI request also increased;
- 2) ASP seizes a great percentage from CGI and PL request;
- 3) Different populations were inspected.

### B. Uncachable Methods

In HTTP 1.0, there are three kinds of methods: GET, HEAD and POST. In HTTP 1.1, there are seven kinds of methods: GET, HEAD, POST, PUT, GET, DELETE, OPTION and TRACE. There are only two methods can be cached: GET and HEAD. Through the trace we found there are only three kinds of methods are ever used: GET, HEAD and POST. The reason for this is that all the Web browsers, both Microsoft Internet Explorer and Netscape Communicator, are configured to only send HTTP 1.0 request, since there are still lot of server running HTTP 1.0 protocol which can not understand the information in HTTP 1.1 request header.

Method	Percentage
GET	99.79%
HEAD	0.0075%
POST	0.1984%

TABLE I  
REQUESTS BY HTTP METHODS

### C. Uncachable HTTP Status Codes

According to the HTTP/1.1 protocol, we can divide the HTTP status codes into three categories (Table II): cachable, negatively cachable and uncachable. Negatively cachable means that for a short amount of time, we can send the cached result to the client without fetching it from original Web server. For example if the object has an HTTP response of "204 No Content", the cache knows there is no content for this object. So for a short time, without validation from original server, the cache can send "204 No Content" to client.

HTTP Status Code	Percentage
Cachable	74%
Negative cachable	16.1%
Uncachable	9.1%

TABLE II  
REQUESTS BY HTTP METHODS

(1) As illustrated in Table III, the responses containing following status codes are cachable:

\*: If a response has a code as 304, we should consider whether the object is in browser cache or in proxy cache. If it is in proxy cache, it can be thought as cachable for the proxy server.

(2) As illustrated in Table IV, the response containing following status codes are negatively cachable:

HTTP Status Code	Percentage
200 OK	61.06%
203 Non-Authoritative Information	0.00%
300 Multiple Choices	0.00%
301 Moved Permanently	0.50%
410 Gone	0.00%
304*Not Modified	12.44%

TABLE III  
CACHABLE HTTP STATUS CODES

HTTP Status Code	Percentage
204 No Content	0.03%
305 Use Proxy	0.00%
400 Bad Request	0.00%
403 Forbidden	0.13%
404 Not Found	1.10%
405 Method Not Allowed	0.00%
414 Request-URI Too Long	0.00%
500 Internal Server Error	0.06%
502 Bad Gateway	0.00%
503 Service Unavailable	13.76%
504 Gateway Timeout	1.07%

TABLE IV  
NEGATIVE CACHABLE HTTP STATUS CODES

(3) As illustrated in Table V, The response containing following status codes are uncachable:

If a response has a code as "304 Not Modified" but the object is in browser cache, it is uncachable for the proxy server. If a response has a code as "302 Moved Temporarily" and has no expire date, it is uncachable. Also here we can see most of uncachable status code have seldom been used. Only four of them are used: Partial Content, Moved Temporarily, See Other, Unauthorized.

### D. Uncachable Response Headers

#### D.1 Last-Modified

Last-Modified Date is a very important header in the HTTP protocol. Usually the browser can use it to validate the document. The browser will send IMS (If Modified Since) to the original server to check whether the file has been modified since the Last-Modified Date. The server will send "304 Not Modified" to browser if it finds that the object is still valid, otherwise it will just send back the whole object. In HTTP/1.1, the browser can also use ETag header to validate the object. But since all browsers currently only send HTTP/1.0 requests, ETag is never used for this purpose.

The reasons for no Last-Modified Date are:

- The object is dynamically generated
- The original server asks the browser to fetch the object directly from it and to calculate the actual accesses or log user behavior

HTTP Status Code	Percentage
100 Continue	0.00%
101 Switching Protocols	0.00%
201 Created	0.00%
202 Accepted	0.00%
205 Reset Content	0.00%
206 Partial Content	0.14%
302 Moved Temporarily	3.28%
303 See Other	0.00%
304 Not Modified	5.81%
401 Unauthorized	0.24%
402 Forbidden	0.00%
406 Not Acceptable	0.00%
407 Proxy Authentication Required	0.00%
408 Request Timeout	0.00%
409 Conflict	0.00%
411 Length Required	0.00%
412 Precondition Failed	0.00%
413 Request Entity Too Large	0.00%
414 Unsupported Media Type	0.00%
501 Not Implemented	0.00%
505 HTTP Version Not Supported	0.00%

TABLE V  
UNCACHABLE HTTP STATUS CODES

- There is something wrong with the Web server or the Web server is not configured well.

If the browser had known that response is in HTTP/1.1 version, it could use the ETag (the entity tag may be used for comparison with other entities from the same resource) to send the If-Not-Match to validate the object. We measured there are about 7.14% objects without Last-Modified header and ETag header. Table VI lists all the other measurements. Other studies have only measured the percentage of response without Last-Modified header but not with ETag header together so they obtain a larger number.

Trace	Percentage
NLANR	7.14%
AT&T [7]	20.6%
Digital [7]	38%
UC-Berkeley [6]	45.5%

TABLE VI  
NO LAST-MODIFIED HEADER BY TRACES

## D.2 Set-Cookie

- Using HTTP/1.0, a header containing "Set Cookie" means the object is uncachable.
- Using HTTP/1.1, a header containing "Set Cookie" and also containing either "Cache-control: no cache="set-cookie"" or "Cache-control: private" means the object is uncachable.

From the HTTP State Management Mechanism [3], we know that prior to HTTP/1.1, there was no mechanism to suppress caching of "Set-Cookie" headers. The Set-Cookie header has to be stored together with the response so that the response can not be cached. In HTTP/1.1, the origin server can only suppress caching of header by either "Cache-control: no cache="set-cookie"" or "Cache-control: private". So we replay the trace and check their response header we found that 5.75% objects contain a cookie but only 1.78% objects can not be cached. This result is quite different from what other people measured before. For example, in the AT&T Lab paper [5], they measured that roughly 30% of requests had cookies but they did not notify whether it is a HTTP/1.0 cookie or HTTP/1.1 cookie. But we believe that there would not typically be 30% responses with a cookie. Most images file will never contain a cookie. That means you can get it without any cookie. The Web master will only set the cookie with HTML file.

Traces	Total Cookie	Uncachable Cookie
NLANR	5.75%	1.78%
AT&T [5]	18.83%-30.20%	-
Digital [7]	4.3%	-

TABLE VII  
COOKIE HEADER BY TRACES

## D.3 Cache-Control

From the HTTP/1.1 [1], we know the Cache-Control header allows a client or server to transmit a variety of directives in either requests or responses. From [1],

These directives typically override the default caching algorithms. As a general rule, if there is any apparent conflict between header values, the most restrictive interpretation should be applied (that is, the one that is most likely to preserve semantic transparency). However, in some cases, Cache-Control directives are explicitly specified as weakening the approximation of semantic transparency (for example, "max-stale" or "public").

- Cache-Control: private  
Indicates that all or part of the response message is intended for a single user and MUST NOT be cached by a shared cache. This allows an origin server to state that the specified parts of the response are intended for only one user and are not a valid response for requests by other users. A private (non-shared) cache may cache the response.
- Cache-Control: no-cache  
Indicates that all or part of the response message MUST NOT be cached anywhere. This allows an origin server to prevent caching even by caches that have been configured to return stale responses to client requests.

We found that only 0.89% of responses were the uncachable Cache-Control header. In [6], they found that 0.1% of responses had the Cache-Control header in 11/08/96 and 0.5% in 4/28/97. In [5], they found that 1.65% of responses in their ISP trace and 1.10% in the research trace had a Cache-Control header. We do not do the measurement for the request from client. But in [5],

they found that only 0.004% of requests had a Cache-Control header. In [6], they found that 7.47% requests in ISP traces and 14.38% in ISP.

Traces	Server Cache-Control	Client Cache-Control
NLANR	1.06%	0.532%
AT&T [5]	1.10% 1.65%	7.47% 14.38%
Digital [6]	0.1% 0.5%	0.004%

TABLE VIII  
CACHE-CONTROL HEADER BY TRACES

#### D.4 Pragma Header

The Pragma general-header field is used to include directives that may apply to any recipient along the request/response chain. "Pragma: no-cache" means the browser should send the request to the origin server even if it has a cached copy. This has the same meaning as "Cache-Control: no-cache", and is defined for backwards compatibility with HTTP/1.0.

Traces	Percentage
NLANR	1.46%
AT&T [5]	5.9%

TABLE IX  
PRAGMA: NO-CACHE HEADER BY TRACE

#### D.5 Authorization

HTTP provides a simple challenge-response authentication mechanism which may be used by a server to challenge a client request and by a client to provide authentication information. A client that wishes to authenticate itself with a server—usually after receiving a "401 Unauthorized" response—does so by including an Authorization request-header field with the request. When the proxy receives a request containing an Authorization field, it must not return the corresponding response as a reply to any other request from other client. So the proxy can only cache this copy for this client. Usually the client will have a local copy in its local cache so that it is no use for the proxy to cache it.

Traces	Percentage
NLANR	0.40%
AT&T [5]	1.10%-1.69%

TABLE X  
AUTHORIZATION BY TRACES

#### D.6 Expires

The Expires entity-header field gives the date/time after which the response should be considered stale. A stale cached object may not be returned by a proxy unless it is first validated

with the original server. We found that about 3.8% headers containing Expires, but that most of them have at least two to three days of age, so are still cachable.

Traces	Percentage
NLANR	3.8%
AT&T [5]	4.7%-5.0%

TABLE XI  
EXPIRE HEADER BY TRACES

#### E. Put It All Together

In this section, we consider all the reasons for an object to be uncachable. We found that about 19.19% of objects should not be cached. Obviously, this number is big enough to affect analysis and simulation.

NLANR uncachable	Percentage
Dynamic	5.31%
HTTP Method	0.20%
HTTP Status Code	9.47%
No Last-Modified Header	7.14%
Set-Cookie Header	1.78%
Cache-Control Header	1.06%
Pragma Header	1.46%
Authorization Header	0.40%
Total	19.19%

TABLE XII  
UNCACHABLE REASONS DISTRIBUTION

#### F. Comparison among different traces in NLANR

In Table XIII, we analyze different traces from NLANR Squid hierarchy. Among these, sd appears to have an unusual population, so that it has low uncachable percentage and high hit rate. pa is a lone node, so that it will not get retransferred requests from other proxy servers. Because all the other proxy servers will retransfer missed requests to the nearest parents and all uncachable requests will induce misses, the nearest parent will get a high percent uncachable rate. This is the result we find from Table XIII where bo1, bo2, pb and sv have parents and children.

## V. SIMULATION RESULT

### 5 Simulation Result

In this section, we run a cache simulation of the trace using LRU eviction. The simulation shows that if we omitted consideration of uncachable objects, hit rates would be obviously higher than if we considered it. There are three kinds of simulation:

1) Consideration of whether an object is cachable is omitted. This is the case for most previous simulations.

Reason	pa	pb	bo1	bo2	sd	sv
Dynamic	5.3%	12.7%	13.1%	12.1%	5.4%	18.5%
Post Uncachable	0.2%	0.6%	1.0%	0.7%	0.2%	1.8%
Uncachable Result	9.5%	19.0%	18.7%	19.7%	6.7%	18.8%
No Last-Modified Date	7.1%	13.6%	11.2%	11.3%	4.2%	13.7%
Set-Cookie (HTTP/1.0)	1.8%	3.9%	3.6%	2.5%	2.7%	3.2%
Cache-Control	1.1%	2.0%	2.6%	2.1%	0.8%	1.6%
Pragma:no-cache	1.5%	4.5%	2.6%	2.3%	1.4%	4.0%
Authorization	0.4%	0.2%	0.2%	0.4%	0.1%	0.6%
Total	19.2%	36.3%	35.7%	35.3%	15.3%	39.6%
Hit	41%	22%	25%	24%	71%	19%

TABLE XIII  
COMPARISON AMONG DIFFERENT TRACES

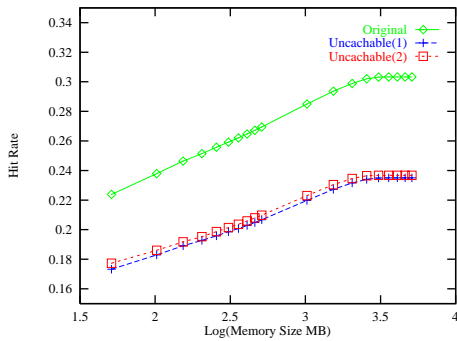


Fig. 3. LRU Simulation Hit Rate

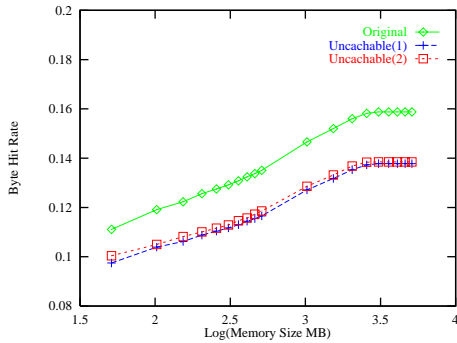


Fig. 4. LRU Simulation Byte Hit Rate

- 2) The cachability of objects is considered but the uncachable objects are still cached. This is the case for a real proxy cache. We call it uncachable(1) in Fig.3 and Fig.4.
- 3) The cachability of objects is considered and the uncachable objects are thrown away immediately. We call it uncachable(2) in Fig.3 and Fig.4.

Fig.3 and Fig.4 are the hit rate and byte hit rate for different simulation policy. From these two graphs, we can see:

- 1) If we omit consideration of cachability, the simulation result is not very accurate. The gap between when cachability is considered and when it is omitted is up to 7 (This can be seen from the Original and Uncachable(1) line).
- 2) It is not very useful to distinguish whether one object is cachable or not when the proxy server get it, since it will not im-

prove the caching performance greatly. (This can be seen from the Uncachable(2) and Uncachable (1)).

Trace	pa	pb	bo1
Real Hit Rate	41%	22%	25%
Uncachable(1) Hit Rate (warm up for 8 days)	41%	19%	23%
Original Hit Rate (warm up for 8 days)	51%	34%	35%

TABLE XIV  
WARMUP SIMULATION

If we consider the cachability of Web object, the simulation result would be close to that of reality. As the experiment illustrated in Table XIV, I randomly picked three proxy servers: pa, pb and bo1. For all these three servers, the hit-rate difference between simulation and reality is less than 3%. But if we do not consider the cachability, the difference would be more than 10%. And more, if we run more warm-up traces, the simulation result would be even closer to reality for the cachable(1) case.

## VI. CHARACTERISTICS OF CACHABLE RESPONSES

Quite a bit is known about statistics of Web traces, but since cachable objects are more valuable than the total objects, we focus on a comparison of the cachable objects and total objects.

### A. Simple Statistical Comparison

We compared some of the statistics of cachable objects versus total objects. We find that there is some evidence that cachable objects are typically larger than uncachable objects.

### B. Log-Log Complementary Distribution (LLCD) and CDF Distribution

We plot the LLCD to visually show that the cachable objects have a heavy tail similar to the total objects. These two are pretty close to each other. But when we plot the cachable and uncachable data sets together, we can found the uncachable data set is slightly larger than cachable data set. The most important

Statistics	Cachable	Total
Sample Size	557175	689500
Minimum	17	17
Maximum	33726757	33726757
Mean Reference	1.41	1.44
Mean Size	8990	8382
Median Size	1232	1113
Standard Deviation	136888	127823

TABLE XV  
SIMPLE STATISTICS OF CACHABLE AND TOTAL

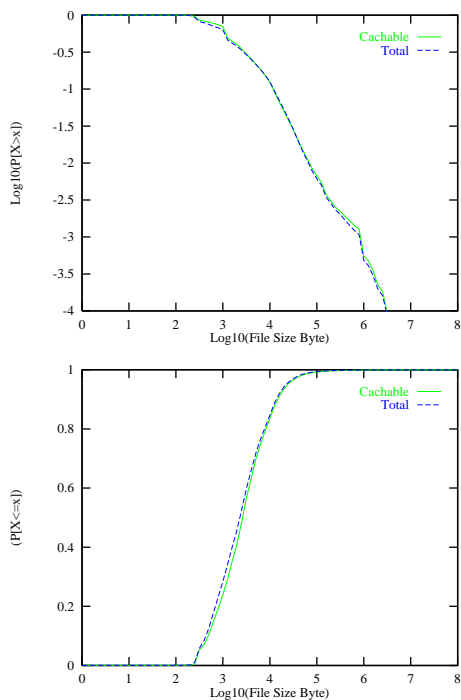


Fig. 5. LLCD and CDF of Log Transformed Cachable and Total Object Sizes

thing is that all the three data sets, cachable, and uncachable and total, have a heavy tailed distribution.

### C. Histograms of Log Transformed Object Sizes

In Fig.7, we can see that cachable objects have a pretty similar size distribution as total objects, except for some popular files. But the cachable data set contains slightly more small files and slightly less large files than the total data set. This plot shows that the lognormal distribution is also a good model for the cachable data set.

### D. Zipf's Law

The last statistical property we examine concerns Zipf's law that can characterize the relative popularity of documents in the Web. In Fig.8, the difference shows that the most popular documents are less popular in the cachable data set than in the total data set.

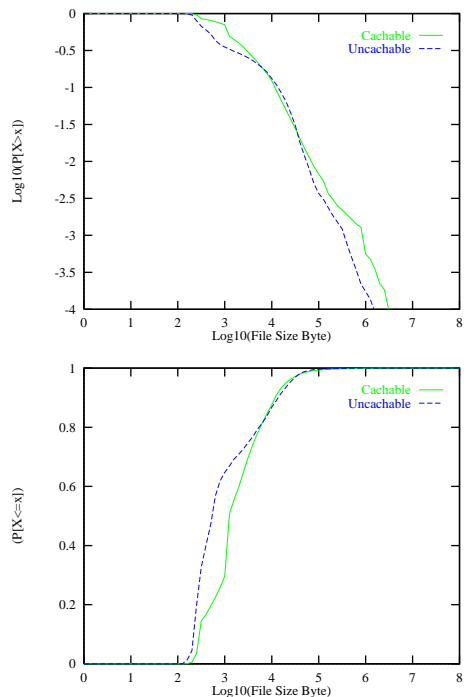


Fig. 6. LLCD and CDF of Log Transformed Cachable and Uncachable Object Sizes

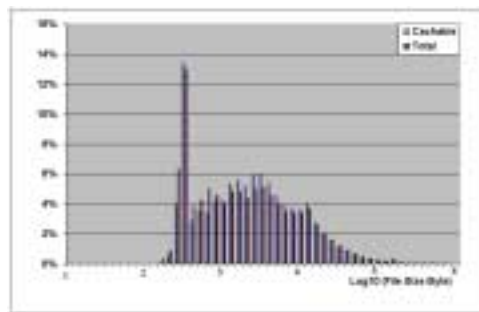


Fig. 7. Histograms of Log-Transformed Object Sizes

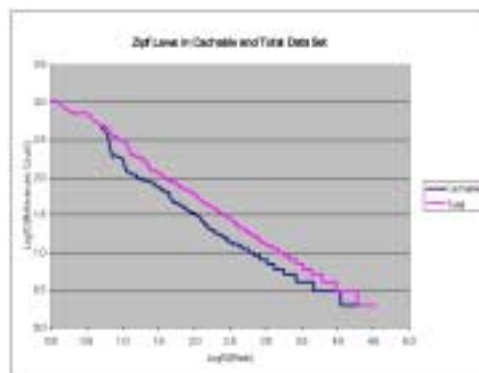


Fig. 8. Zipf's Law in Cachable and Total Data Set

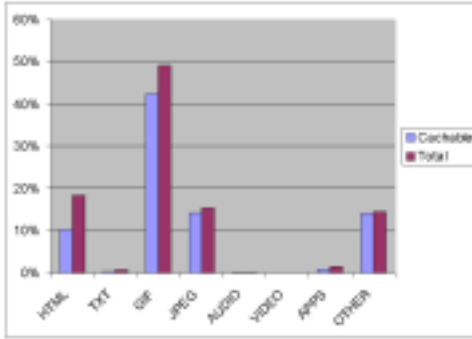


Fig. 9. Requests by File Type

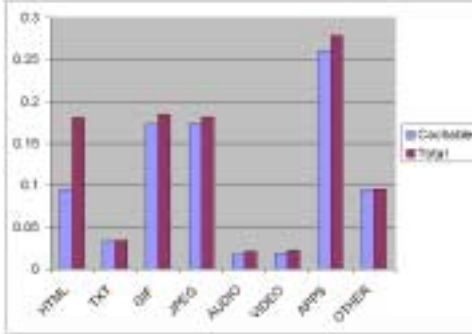


Fig. 10. Request Bytes by File Type

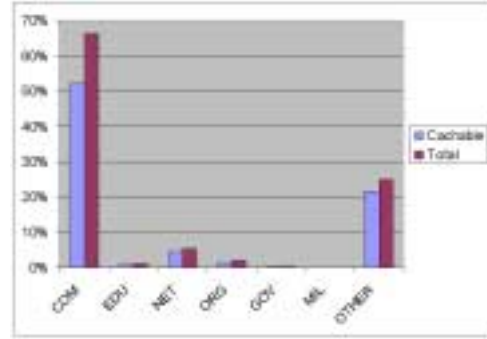


Fig. 11. Requests by Domain

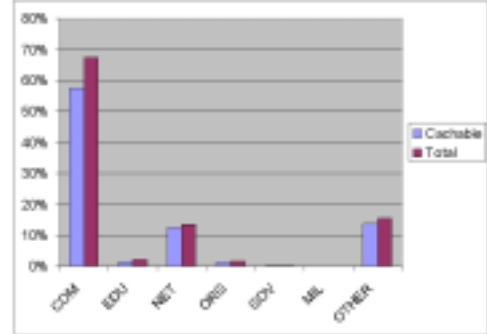


Fig. 12. Request Bytes by Domain

### E. The Relation between Size and References

At the first glance, if we only look at the correlation coefficient of size and references, we will find there is no relation between size and references. But if we look at the Hit Rate Byte Hit Rate for infinite cache (H-B), we find H-B is still large for both the cacheable and total data sets. From [10], we know

$$H - B = -Cov(R_i, S_i) / \mu_r \mu_b \quad (1)$$

$H_B$  indicates a preference for smaller files whereas  $H_B$  indicates a preference for larger files. Here we can see that for both the cacheable and total data sets, smaller files are preferred.

	Cacheable	Total
Correlation Coefficient	-0.002974	-0.003536
Hit Rate-Byte Hit Rate (Infinite Cache)	0.149651	0.133203

TABLE XVI  
RELATION BETWEEN SIZE AND REFERENCES

## VII. ADDITIONAL STATISTICS

### A. File Type Distribution

In Fig.9 and Fig.10, we plot the request distribution and request bytes distribution by file type (such as html, txt, gif, jpeg, audio, video, etc.). We found that cachability has no strong relation to the file type, except for html. Only half of the html files are cacheable. There are some other interesting statistics: (1) Although the number of gif files is as twice as the number of jpeg files, the total bytes of gif files is pretty close to that of jpeg

files. This means that the mean size of jpeg files is twice that of gif files. (2) Applications take 2.0% of the total requests but it takes 31.5% of the total bytes, which is the largest function of the total bytes.

### B. Domain Distribution

In Fig.11 and Fig.12, we plot the request distribution and request byte distribution by domain type, such as com, edu, net, org, gov, mil, etc.. We found the cachability has no strong relation to the other domains, except for com. There about objects in com domain can not be cached.

### C. Web Server Distribution

In Fig.13 and Fig.14, we plot the request distribution and request byte distribution by Web servers, such as Apache, Netscape, Microsoft IIS, etc. We found there is no particular preference for uncacheable objects among different Web servers.

### D. HTTP Protocol Distribution

In Fig.15 and Fig.16, we plot the request distribution and request byte distribution by different two HTTP versions, such as HTTP 1.0 and HTTP 1.1. We found there is no particular preference for uncacheable objects among different HTTP versions.

### E. Putting It All Together

We analyzed cachability according to object type, domain, Web server and HTTP version. We find that cachability has little relation to either Web Server type or HTTP version type. This means the implementation of Web servers and protocols will not

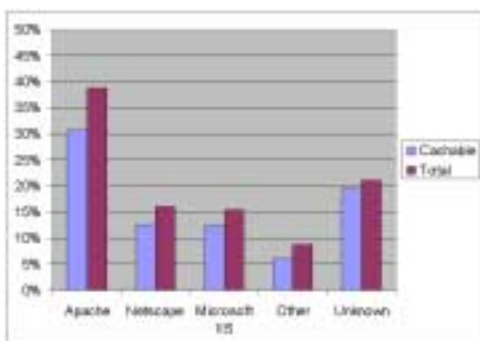


Fig. 13. Requests by Web Server

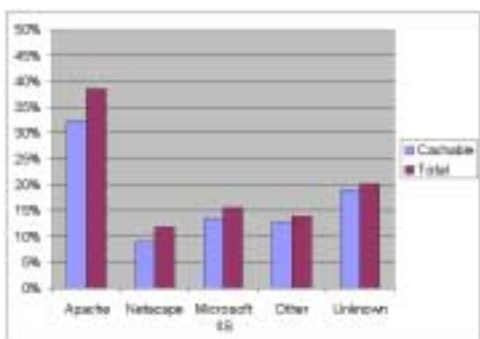


Fig. 14. Request Bytes by Web Server

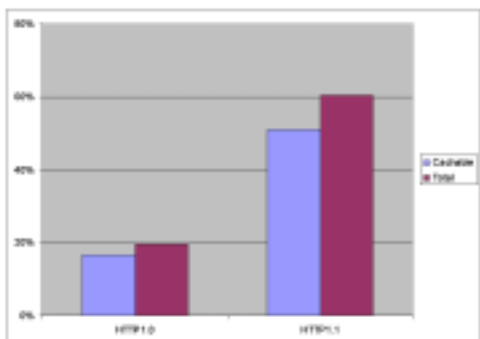


Fig. 15. Requests by HTTP Version

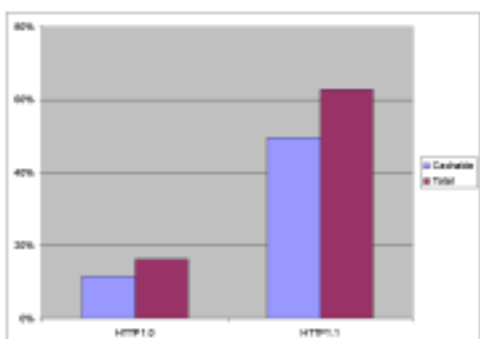


Fig. 16. Requests Bytes by HTTP Version

affect cachability. However cachability is related to a file is domain and type. These files in html format and in com domain tend to have poorer cachability.

## VIII. SUMMARY

In this paper, first we describe the reasons for uncachability of web objects. We find that almost 15% to 40% of web objects in our traces can not be cached. The main reasons for uncachability are: 1) dynamic requests, 2) responses without Last-Modified header, 3) responses with HTTP "302 Moved Temporarily" status code, and 4) responses with a HTTP/1.0 cookie.

Second we use a simple cache simulation to show that other simulation results may be misleading if one does not consider the cachability of Web objects.

Third we show some statistics of cachable Web objects. We find that almost all the cachable Web objects' characteristics are pretty close to that of the total set of Web objects. Both the sets of cachable and total Web requests prefer small files.

Last, we show the distribution of cachable and total Web objects among different file types, domain types, Web server types and HTTP protocol types. We find that the files in HTML format and in the COM domain tend to have a poorer cachability.

## ACKNOWLEDGMENTS

The author would like to thank Prof. Mark Crovella for helping with encouragement, discussions, ideas and experiment methods.

## REFERENCES

- [1] R.Fielding, J.Gettys, J.Mogul, H.Frystyk and T.Berners-Lee *Hypertext Transfer Protocol – HTTP/1.1* RFC 2068, Jan 1997
- [2] T. Berners-Lee, R.Fielding and H.Frystyk *Hypertext Transfer Protocol – HTTP/1.0* RFC 1945, May 1996
- [3] D. Kristol and L.Montulli *HTTP State Management Mechanism* RFC 2109, Feb 1997
- [4] National Lab of Applied Network Research (NLNLR) *Sanitized access log* <http://ircache.nlanr.net/Traces/>, Apr 28, 1999
- [5] R. Cceres, F. Douglis, A. Feldmann, G. Glass, and M. Rabinovich *Web Proxy Caching: The Devil is in the Details* Proc. of ACM SIGMETRICS Workshop on Internet Server Performance, June 1998
- [6] S. Gribble and E. Brewer *System Design Issues for Internet Middleware Services: Deductions from a large Client Trace* Proc. of the 1997 USENIX Symposium on Internet Technologies and Systems (USITS'97), Dec 1997
- [7] F. Douglis, A. Feldmann, B. Krishnamurthy and J. Mogul *Rate of Change and other Metrics: a Live Study of the World Wide Web* Proc. of the 1997 USENIX Symposium on Internet Technologies and Systems (USITS'97), Dec 1997
- [8] M. Crovella and A. Bestavros *Self-similarity in World Wide Web traffic* Proc. of ACM SIGMETRICS, May 1996
- [9] V. Almeida, A. Bestavros, M. Crovella and A. Oliveira *Characterizing reference locality in the WWW* Proc. of 1996 International Conference on Parallel and Distributed Information System (PDIS'96), Dec 1996
- [10] P. Barford, A. Bestavros, A. Bradley and M.Crovella *Changes In Web Client Access Patterns: Characteristics and Caching Implications* BUCS-TR-1998-023, 1998
- [11] Squid Frequently Asked Questions <http://squid.nlanr.net/Squid/FAQ>
- [12] S.Williams, M.Abrams, C.Standbridge, G.Abdulla and E.Fox *Removal Policies in Network Caches for World-Wide Web Documents* Proc. of the ACM Sigcomm, Aug 1996
- [13] M.Abrams, C.Standbridge, G.Abdulla, S. Williams and E.Fox *Caching Proxies: Limitations and Potentials* WWW-4, Boston Conference, Dec, 1995
- [14] R.Wooster and M.Abrams *Proxy Caching the Estimates Page Load Delays* In the 6th International World Wide Web Conference, Apr, 1997
- [15] P.Cao and S.Irani *Cost-Aware WWW Proxy caching Algorithms* Proc. of the 1997 USENIX Symposium on Internet Technologies and Systems (USITS'97), Dec 1997
- [16] InfoLibria Inc. <http://www.infolibra.com/>

[17] Cisco Inc. <http://www.cisco.com/>

[18] Inktomi Corporation. <http://www.inktomi.com/>