Caching For Performance

Using Client-Side Cache Solutions and Server-Side Caching Configurations to Improve Internet Performance

by
Stephen Pierzchala (stephen@pierzchala.com)

In today's highly competitive e-commerce marketplace, the performance of a Web site plays a key role in attracting new and retaining current customers. New technologies are being developed to help speed up the delivery of content to customers while still allowing companies to get their message across using rich, graphical content. However, in the rush to find new technologies to improve Internet performance, one low-cost alternative is often overlooked: client-side content caching.

Web administrators and content providers are often dismissive of this technique when attempting to improve performance. They are more concerned with ensuring that clients always get the freshest content possible. In their eyes, by allowing their content to be cached on servers they do not directly control, they have lost some level of control over the way that a Web page appears.

This bias against caching is, in most cases, unjustified. By understanding how server software can be used to distinguish unique caching policies for each type of content being delivered, client-side performance gains can be achieved with no new hardware or software being added to an existing Web-site system.

Caching

When a client requests Web content, the information is either retrieved directly from the origin server, from a browser cache on a local hard drive or from a nearby cache server[1]. Where and for how long the data is stored depends on how the data is tagged when it leaves the Web server. Web content can be in one of three cache states: **non-cacheable**, **fresh** or **stale**.

The **non-cacheable** state indicates a file that should never be cached by any device that receives it and that every request for that file must be retrieved from the origin server. This places an additional load on both client and server bandwidth, as well as on the server which responds to these additional requests. In many cases, such as database queries, news content, and personalized content marked by unique cookies, the content provider may explicitly prevent caching to avoid having **stale** served to the client.

A **fresh** file is one that has a clearly defined future expiration date and/or does not indicate that it is non-cacheable. A file with a defined lifespan is valid for a set number of seconds after it is downloaded, or until the explicitly stated expiry time is reached. At that point, the file is considered **stale** and must be re-validated (preferred as it requires less bandwidth) or re-loaded from the origin server.[2]

If a file does not explicitly indicate it is non-cacheable, but does not indicate an explicit expiry period or time, the cache server can be configured to assign the file a default expiry time. When that deadline is reached and the cache server receives a request for that file, the server checks with the origin server to see whether the content has changed. If the file is unchanged, the counter is reset and the existing content is served to the client; if the file is changed, the new content is downloaded, cached according to its settings and then served to the client.

A **stale** file is one that is no longer considered valid by the cache. When a client requests this object, the cache must re-validate or re-load the file from the origin server before the data can served to the client.

Caching information is included in the information sent to and from the server. The state of an item being considered for caching is determined using one or more of the HTTP header messages, which we now describe.

HTTP Caching Header Messages

[1] The proximity that is referred to here is network proximity, not physical proximity. For example, AOL's network has some of the world's largest cache servers and they are concentrated in Virginia; however, because of the structure of AOL's network, these cache servers are not far from the clients.

[2] A re-verify is preferred as it consumes less bandwidth than a full re-load of the object from the origin server. With a re-verification, the origin server just confirms that the file is still valid and the cache server can simply reset the timer on the object.

Each of the HTTP headers messages[3] described below[4] identifies a particular condition that the proxy server must adhere to when deciding whether the content is fresh enough to be served to the requesting client.

Pragma: no-cache is an HTTP/1.0 client request header that requests that the content not be served from cache anywhere en route to the server. This response has been deprecated in favor of the new HTTP/1.1 **Cache-Control** header, but is still used in many browsers. The continued use of this header is necessary to ensure backwards-compatibility, as it cannot be guaranteed that all devices and servers will understand the HTTP/1.1 server headers.

This header should never be seen in a server response header. If a server wants to indicate that content is uncacheable, the server should use one of the HTTP/1.1 **Cache-Control** messages described below.

The **Cache-Control** family of HTTP/1.1 client and server messages can be used to clearly define not only if an item can be cached, but also for how long and how it should be validated upon expiry. There are a large number of options for this header field, but five that are especially relevant to this discussion.[5]

Cache-Control: private/public
This setting indicates what type of devices can cache the data. The **private** setting allows the marked items to be cached by the requesting client, but not by any cache servers encountered en-route. The **public** setting indicates that any device can cache this content. By default, **public** is assumed unless **private** is explicitly stated.

Cache-Control: no-cache
This is the HTTP/1.1 equivalent of **Pragma: no-cache** when used by Web clients, as it forces an end-to-end retrieval of the requested files, circumventing a proxy server on the local network. This is also a valid server response header requesting that proxy servers not cache the indicated items when they pass through on the way to the client.

Cache-Control: no-store
This message is the most strongly enforceable of the server-side response headers. Servers use this message to state categorically that items must not be cached. The **no-cache** header can be ignored under certain proxy configurations, while the **no-store** must be obeyed.

Cache-Control: max-age=x
This setting allows indicated files to be cached either by the client or the cache server for "x" seconds.

Cache-Control: must-revalidate
This setting informs the cache server that if the item in cache is stale, it must be re-validated at the origin server before it can be sent to the client.

A number of the **Cache-Control** settings can be combined to form a larger header message with multiple options. For example, an administrator may want to define how long the content is valid for, and then indicate that, at the end of that period, all new requests must be revalidated with the origin server. This can be accomplished by creating a multi-field **Cache-Control** header message like the one below.

```
Cache-Control: max-age=3600, must-revalidate
```

The **Expires** header sets an absolute or relative expiry time for the requested file. This is usually in the future, but some server administrators attempt to negative cache items by setting an expiry date that is in the past – an example of this will be shown below. A better practice is to set explicit **Cache-Control** headers to explicitly prevent caching, while setting an **Expires** header equal to the server date.

Last-Modified is a server response header that most commonly indicates the last time the state of the requested object was updated in the server's filesystem. The cache server can use this to confirm an object has not changed since it was inserted into the cache, allowing for the re-validation, versus the complete re-loading, of objects in cache.

If-Modified-Since is a client-side header message that is sent either by a browser or a cache server and is set by the **Last-Modified** value of the object in cache. When the origin server has not set an explicit cache expiry value and the cache server has had to set an expiry time on the object using its own internal configuration, the **Last-Modified** value is used to confirm whether content has changed on the origin server. If the **Last-Modified** value on an object held by the origin server is newer than that held by the client, the entire file is re-loaded. If these values are the same, the origin server returns a **304 Not Modified** HTTP message. In this case, the cached object is then served to the client and the cache-defined counter for the object is reset.

---

[3] An HTTP header message is a data-control message sent by a Web client or a Web server to indicate a variety of data transmission parameters concerning the requests being made.

[4] There are actually a substantially larger number of header messages that can be applied to a client or a server data transmission to communicate caching information. The most up-to-date list of the messages can be found in section 13 of RFC 2616, "Hypertext Transfer Protocol -- HTTP/1.1".

[5] A complete listing of the Cache-Control settings can be found in RFC 2616, "Hypertext Transfer Protocol -- HTTP/1.1", section 14.9.

Cache Trace Examples

The following two examples show how a server can use header messages to mark content as non-cacheable, or set very specific caching values.

Server Messages for a Non-Cacheable Object

```
HTTP/1.0 200 OK
Content-Type: text/html
Content-Length: 19662
Pragma: no-cache
Cache-Control: no-cache
Server: Roxen/2.1.185
Accept-Ranges: bytes
Expires: Wed, 03 Jan 2001 00:18:55 GMT
```

In this example, the server returns three indications that the content is non-cacheable. The first two are the **Pragma: no-cache** and **Cache-Control: no-cache** statements. Only the **Cache-Control** header is valid, and should be enough to prevent most caches from storing this file. The Web administrator in this example has chosen to erroneously include the **Pragma** header, most likely in the belief that any device, regardless of the version of HTTP used, will clearly understand that this object is non-cacheable.

However, in order to attempt to guarantee that this item is never stored in or served from cache, the **Expires** statement is set to a date and time that is in the past.[6] Again, this is unnecessary, and setting the **Expires** header to equal the date and time on the server at the time of the response[7] is a more than adequate guarantee that the object will not be cached.

Specific Caching Information in Server Messages

```
HTTP/1.1 200 OK
Date: Tue, 13 Feb 2001 14:50:31 GMT
Server: Apache/1.3.12
Cache-Control: max-age=43200
Expires: Wed, 14 Feb 2001 02:50:31 GMT
Last-Modified: Sun, 03 Dec 2000 23:52:56 GMT
ETag: "1cbf3-dfd-3a2adcd8"
Accept-Ranges: bytes
Content-Length: 3581
Connection: close
Content-Type: text/html
```

In the example above, the server returns a header message **Cache-Control: max-age=43200**. This immediately informs the cache that the object can be stored in cache for up to 12 hours. This 12-hour time limit is further guaranteed by the **Expires** header, which is set to a date value that is exactly 12 hours ahead of the value set in the **Date** header message.[8]

These two examples present two variations of Web server responses containing information that makes the requested content either completely non-cacheable or cacheable only for a very specific period of time.

How does caching work?

Devices on the Internet cache content, then serve this stored content when the original client or another client that uses that same cache requests the same file. This rather simplistic description covers a number of different cache scenarios, but two will be the focus of this paper – browser caching and caching servers.[9]

For the remainder of this paper, the caching environment that will be discussed is one involving a network with a number of clients using a single cache server, the general Internet, and a server network with a series of Web servers on it.

Browser Caching

Browser caching is what most people are familiar with, as all Web browsers perform this behavior by default. With this type of caching, the Web browser stores a copy of the requested files in a cache on the client machine in order to help speed up page downloads. A performance increase is achieved by serving stored files from this directory on the local hard drive instead of retrieving these same files from the Web server, which resides across a much slower Internet connection.

To ensure that old content is not being served to the client, the browser checks its cache first to see if an item is in cache. If the item is in cache, the browser then confirms the state of the object with the origin server to see if the item has been modified at the source since the browser last downloaded it. If the object has not been modified, the origin server sends a **304 Not Modified** message, and the item is served from the local browser cache.

---

[6] The initial file request that generated this header was sent on February 12, 2001.

[7] The next example demonstrates the **Date** header

[8] The **Date** header message indicates the date and time on the origin server when it responded to the request.

[9] A third type of caching, Reverse Caching or HTTPD Accelerators, are used at the server side to place highly cacheable content into high-speed machines that use solid-state storage to make retrieval of these objects very fast. This reduces the load on the Web servers and allows them to concentrate on the generation of dynamic and personalized content.

First Request for a file

```
GET /file.html HTTP/1.1
Accept: image/gif, image/x-xbitmap, image/jpeg,
image/pjpeg, application/vnd.ms-powerpoint,
application/vnd.ms-excel, application/msword,
application/x-comet, */*
Accept-Language: en-us
Accept-Encoding: gzip, deflate
User-Agent: Mozilla/4.0 (compatible; MSIE 5.5;
Windows NT 5.0)
Host: 24.5.203.101
Connection: Keep-Alive

HTTP/1.1 200 OK
Date: Tue, 13 Feb 2001 20:00:22 GMT
Server: Apache
Cache-Control: max-age=604800
Last-Modified: Wed, 29 Nov 2000 15:28:38 GMT
ETag: "1df-28f1-3a2520a6"
Accept-Ranges: bytes
Content-Length: 10481
Keep-Alive: timeout=5, max=100
Connection: Keep-Alive
Content-Type: text/html
```

In the above example, the file is retrieved from the server for the first time, and the server sends a **200 OK** response and then returns the requested file. The data shown here is just application trace data. For a more complete example of the application and network properties of a Web object retrieval, see the Appendices.

Second Request for a file

```
GET /file.html HTTP/1.1
Accept: */*
Accept-Language: en-us
Accept-Encoding: gzip, deflate
If-Modified-Since: Wed, 29 Nov 2000 15:28:38 GMT
If-None-Match: "1df-28f1-3a2520a6"
User-Agent: Mozilla/4.0 (compatible; MSIE 5.5;
Windows NT 5.0)
Host: 24.5.203.101
Connection: Keep-Alive

HTTP/1.1 304 Not Modified
Date: Tue, 13 Feb 2001 20:01:07 GMT
Server: Apache
Connection: Keep-Alive
Keep-Alive: timeout=5, max=100
ETag: "1df-28f1-3a2520a6"
Cache-Control: max-age=604800
```

The second request for a file sees the client send a request for the same object 40 seconds later, but with two additions. The server asks if the file has been modified since the last time it was requested by the client (**If-Modified-Since**). If the origin server cannot confirm the state of the requested object using the **If-Modified-Since** field, the client asks if the object's **Etag** tracking code has changed using the **If-None-Match** header message.[10] The origin server responds by verifying that object has not been modified and confirms this by returning the same **Etag** value that was sent by the client. This rapid client-server exchange allows the browser to quickly determine that it can serve the file directly from its local cache directory.

Caching Server

A caching server performs functions similar to those of a browser cache, only on a much larger scale. Where a browser cache is responsible for storing Web objects for a single browser application on a single computer, a cache server stores Web objects for a large number of clients or perhaps even an entire network. With a cache server, all Web requests from a network are passed through caching server, which then will serve the requested files to the client. The cache server can deliver content either directly from its own cache of objects, or by retrieving objects from the Internet and then serving them to clients.[11]

Cache servers are more efficient than browser caches, as this network-level caching process makes the object available to all users of the network once it has been retrieved. With a browser cache, each user – and, in fact, each browser application on a specific computer – must maintain a unique cache of files that is not shared with other clients or applications.

Also, cache servers use additional information provided by the Web server in the headers sent along with each Web request. Browser caches simply re-validate content with each request, confirming that the content has not been modified since it was last requested. Cache servers use the values sent in the **Expires** and **Cache-Control** header messages to set explicit expiry times for objects they store.

---

[10] The **Etag** or entity tag is used to identify specific objects on a Web server. Each item has unique **Etag** value, and this value is changed each time the file is modified. As an example, the **Etag** for a local Web file was captured. This data was re-captured after the file was modified – two carriage returns were inserted.

Test 1 – Original File
ETag: "21ccd-10cb-399a1b33"
Test 2 – Modified File
ETag: "21ccd-10cd-3a8c0597"

[11] This is where the other name for a cache server comes from, as the cache server acts as a proxy for the client making the request. The term proxy server is outdated as the term proxy assumes that the device will do exactly as the client requests; this is not always the case due to the security and content control mechanisms which are a part of all cache servers today. The client isn't always guaranteed to receive the complete content they requested. Infact, many networks do not allow any content into the network that does not first go through the cache devices on that network.

First Request for a file through a cache server

```
GET http://24.5.203.101/file.html HTTP/1.1
Accept: image/gif, image/x-xbitmap, image/jpeg,
image/pjpeg, application/vnd.ms-powerpoint,
application/vnd.ms-excel, application/msword,
application/x-comet, */*
Accept-Language: en-us
Accept-Encoding: gzip, deflate
User-Agent: Mozilla/4.0 (compatible; MSIE 5.5;
Windows NT 5.0)
Host: 24.5.203.101
Proxy-Connection: Keep-Alive

HTTP/1.0 200 OK
Date: Tue, 16 Jan 2001 15:46:42 GMT
Server: Apache
Cache-Control: max-age=604800
Last-Modified: Wed, 29 Nov 2000 15:28:38 GMT
ETag: "1df-28f1-3a2520a6"
Content-Length: 10481
Content-Type: text/html
Connection: Close
```

The first request from the client through a cache server shows two very interesting things.[12] The first is that although the client request was sent out as HTTP/1.1, the server responded using HTTP/1.0. The browser caching example above demonstrated that the responding server uses HTTP/1.1. The change in protocol is the first clue that this data was served by a cache server.

The second item of interest is that the file that is initially served by the proxy server has a **Date** field set to January 16, 2001. This server is not serving stale data; this is the default time set by the cache server to indicate a new object that has been inserted in the cache.[13]

Second Request for a file through a cache server – Second Browser

```
GET http://24.5.203.101/file.html HTTP/1.1
Host: 24.5.203.101
User-Agent: Mozilla/5.0 (Windows; U; Windows NT
5.0; en-US; 0.7) Gecko/20010109
Accept: */*
Accept-Language: en
Accept-Encoding: gzip,deflate,compress,identity
Keep-Alive: 300
Connection: keep-alive

HTTP/1.0 200 OK
Date: Tue, 16 Jan 2001 15:46:42 GMT
Server: Apache
Cache-Control: max-age=604800
Last-Modified: Wed, 29 Nov 2000 15:28:38 GMT
ETag: "1df-28f1-3a2520a6"
Content-Length: 10481
Content-Type: text/html
Connection: Close
```

Third Request for a file through a cache server – Second Client Machine

```
GET http://24.5.203.101/file.html HTTP/1.0
Accept: image/gif, image/x-xbitmap, image/jpeg,
image/pjpeg, application/vnd.ms-powerpoint,
application/vnd.ms-excel, application/msword, */*
Accept-Language: en-us
User-Agent: Mozilla/4.0 (compatible; MSIE 5.5;
Windows NT 5.0)
Host: 24.5.203.101
Proxy-Connection: Keep-Alive

HTTP/1.0 200 OK
Date: Tue, 16 Jan 2001 15:46:42 GMT
Server: Apache
Cache-Control: max-age=604800
Last-Modified: Wed, 29 Nov 2000 15:28:38 GMT
ETag: "1df-28f1-3a2520a6"
Content-Length: 10481
Content-Type: text/html
Connection: Close
```

A second request through the cache server, using another browser on the same client configured to use the cache server, indicates that this client retrieved the file from the cache server, not from the origin server. The **Date** field in the server response is the same as the initial request and the protocol has once again been swapped from HTTP/1.1 to HTTP/1.0.

The third example shows that the object is now not only available to different browsers on the same machine, but now that it is available to different machines on the same network, using the same cache server. By requesting the same content from another client machine on the same network, it is clear that the object is served to the client by the cache server, as the **Date** field set to the same value observed in the previous two examples.

---

[12] The data shown here is just application trace data. For a more complete example of what the application and network properties of a Web object retrieval are, please see Appendix A and B.

[13] All the data captures used in this example were taken on February 11-14, 2001.

Why should data be cached?

Many Web pages that are downloaded by Web browsers today are marked as being non-cacheable. The theory behind this is that there is so much dynamic and personalized content on the Internet today that if any of it is cached, people using the Web may not have the freshest possible content or they may end up receiving content that was personalized for another client making use of the same cache server.

The dynamic and personalized nature of the Web today does make this a challenge, but if the design of a Web site is examined closely, it can be seen that these new features of the Web can work hand-in-hand with content caching.

How does caching affect the perceived user experience? In both the browser caching and caching server discussions above, caching helps attack the problem of Internet performance on three fronts. First, caching moves content closer to the client, by placing it on local hard-drives or in local network caches. With data stored on or near the client, the network delay encountered when trying to retrieve the data is reduced or eliminated.

Secondly, caching reduces network traffic by serving content that is **fresh**, as described above. Cache servers will attempt to confirm with the origin server that the objects stored in cache – if not explicitly marked for expiry – are still valid and do not need to be fully re-loaded across the Internet. In order to gain the maximum performance benefit from object caching, it is vital to specify explicit cache expiry dates or periods.

The final performance benefit to properly defining content caching configurations on an origin server is that server load and bandwidth usage is reduced. If the server uses carefully planned explicit caching policies, server load can be greatly reduced, improving the user experience.

When determining how the configuration of a Web server can be modified to improve content cacheability, it is important keep in mind two very important considerations. First, the content and site administrators must have a very granular level of control over how the content will or won't be cached once it leaves their server. Secondly, within this need to control how content is cached, ways should be found to minimize the impact client requests have on bandwidth and server load by allowing some content to be cached.

Take the example of a large, popular site that is noted for its dynamic content and rich graphics. Despite having a great deal of dynamic content, caching can serve a beneficial purpose without compromising the nature of the content being served. The primary focus of the caching evaluation should be on the rich graphical content of the site.

If the images of this site all have unique names that are not shared by any other object on the site, or the images all reside in the same directory tree, then this content can be marked differently within the server configuration, allowing it to be cached.[14] A policy that allows these objects to be cached for 60, 120 or 180 seconds could have a large affect on reducing the bandwidth and server strain at the modified site. During this seemingly short period of time, many of the requests for the same object could originate from a large corporate network or ISP. If local cache servers for this client-side network can handle these requests, both the server and client sides of the transaction could see immediate performance improvements.

Taking a server header from an example used earlier in the paper, it can be demonstrated how even a slight change to the server header itself can help control the caching properties of dynamic content.

Dynamic Content

```
HTTP/1.1 200 OK
Date: Tue, 13 Feb 2001 14:50:31 GMT
Server: Apache/1.3.12
Cache-Control: no-store, must-revalidate
Expires: Sat, 13 Feb 2001 14:50:31 GMT
Last-Modified: Sun, 03 Dec 2000 23:52:56 GMT
ETag: "1cbf3-dfd-3a2adcd8"
Accept-Ranges: bytes
Content-Length: 3581
Connection: close
Content-Type: text/html
```

Static Content

```
HTTP/1.1 200 OK
Date: Tue, 13 Feb 2001 14:50:31 GMT
Server: Apache/1.3.12
Cache-Control: max-age=43200, must-revalidate
Expires: Wed, 14 Feb 2001 02:50:31 GMT
Last-Modified: Sun, 03 Dec 2000 23:52:56 GMT
ETag: "1cbf3-dfd-3a2adcd8"
Accept-Ranges: bytes
Content-Length: 3581
Connection: close
Content-Type: text/html
```

As can been seen above, the only difference in the headers sent with the Dynamic Content and the Static Content are the **Cache-Control** and **Expires** values. The Dynamic Content example sets **Cache-Control** to

[14] The description used here is based on the configuration options available with the Apache/1.3.x server family, which allows caching options to be set down to the file level. Other server applications may vary in their methods of applying individual caching policies to different sets of content on the same server.

no-store, must-revalidate and **Expires** to the same time as the **Date** header. This should prevent any cache from storing this data or serving it when a request is received to retrieve the same content.

The Static Content modifies these two settings, making the requested object cacheable for up to 12 hours – **Cache-Control** value set to 43,200 seconds and an **Expires** value that is exactly 12 hours in the future. After the period specified, the browser cache or caching server must re-validate the content before it can be served in response to local requests.

The **must-revalidate** item is not necessary, but it does add additional control over content. Some cache servers will attempt to serve content that is stale under certain circumstances, such as if the origin server for the content cannot be reached. The **must-revalidate** setting forces the cache server to re-validate the stale content, and return an error if it cannot be retrieved.

Differentiating caching policies based on the type of content served allows a very granular level of control over what is not cached, what is cached, and for how long the content can be cached for and still be considered fresh. In this way, server and Web administrators can improve site performance a little or no additional development or capital cost.

It is very important to note that defining specific server-side caching policies will only have a beneficial affect on server performance if **explicit** object caching configurations are used. The two main types of explicit caching configurations are those set by the “Expires” header and the “Cache-Control” family of headers – as seen in the example above. If no explicit value is set for object expiry, performance gains that might have been achieved are eliminated by a flood of unnecessary client and cache server requests to re-validate unchanged objects with the origin server.

Conclusion

Despite the growth of dynamic and personalized content on the Web, there is still a great deal of highly cacheable material that is served to clients. However, many sites do not take advantage of the performance gains that can be achieved by isolating the dynamic and personalized content of their site from the relatively static content that is served alongside it.

Using the inherent ability to set explicit caching policies within most modern Web-server applications, objects handled by a Web-sever can be separated into unique content groups. With distinct caching policies for each defined group of Web objects, the Web-site administrator, not the cache administrator, has control over how long content is served without re-validation or re-loading. This granular control of explicit content caching policies can allow Web-sites to achieve noticeable performance gains with no additional outlay for hardware and software

Bibliography

Web Caching (Duane Wessels, Nathan Torkington [editor]), O’Reilly and Associates. 2001

Web Protocols and Practice: HTTP/1.1, Networking Protocols, Caching, and Traffic Measurement (Balachander Krishnamurthy and Jennifer Rexford), Addison Wesley Longman, Inc, 2001

.

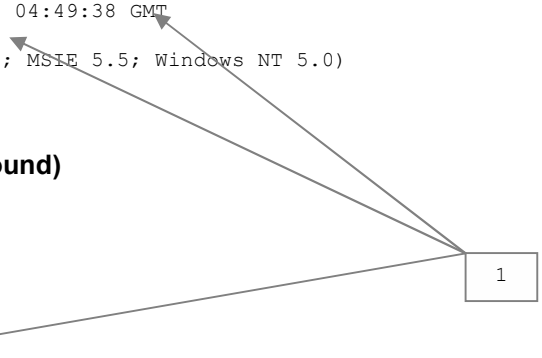
Appendix A: Example of a Web request processed by a browser cache

Application and Network Side (Outbound)

```
GET / HTTP/1.1
Accept: image/gif, image/x-xbitmap, image/jpeg, image/pjpeg, application/vnd.ms-powerpoint,
application/vnd.ms-excel, application/msword, */*
Accept-Language: en-us
Accept-Encoding: gzip, deflate
If-Modified-Since: Sat, 19 Aug 2000 04:49:38 GMT
If-None-Match: "219cb-46d-399e11e2"
User-Agent: Mozilla/4.0 (compatible; MSIE 5.5; Windows NT 5.0)
Host: grabit.keynote.com
Connection: Keep-Alive
```

Application and Network Side (Inbound)

```
HTTP/1.1 304 Not Modified
Date: Thu, 15 Feb 2001 18:16:14 GMT
Server: Apache
Connection: Keep-Alive
Keep-Alive: timeout=15, max=100
ETag: "219cb-46d-399e11e2"
```



1. The only caching information passed back to the server when this file verified by the browser is the “If-Modified-Since” and “If-None-Match” data. The second item is a fall-back condition using the entity tag (“Etag”) value of the requested file, in the case that none of the other values can be verified by the server. As the entity tag sent by the client matches the one held by the server, the 304 Not Modified message was returned by the server.

Appendix B: Example of a Web request processed by a cache server

Application Side (Outbound)

```
GET http://grabit.keynote.com/ HTTP/1.0
Accept: */*
Accept-Language: en-us
Proxy-Connection: Keep-Alive
If-Modified-Since: Sat, 19 Aug 2000 04:49:38 GMT; length=1133
User-Agent: Mozilla/4.0 (compatible; MSIE 5.5; Windows NT 5.0)
Host: grabit.keynote.com
Pragma: no-cache
```

Cache Side (Outbound)

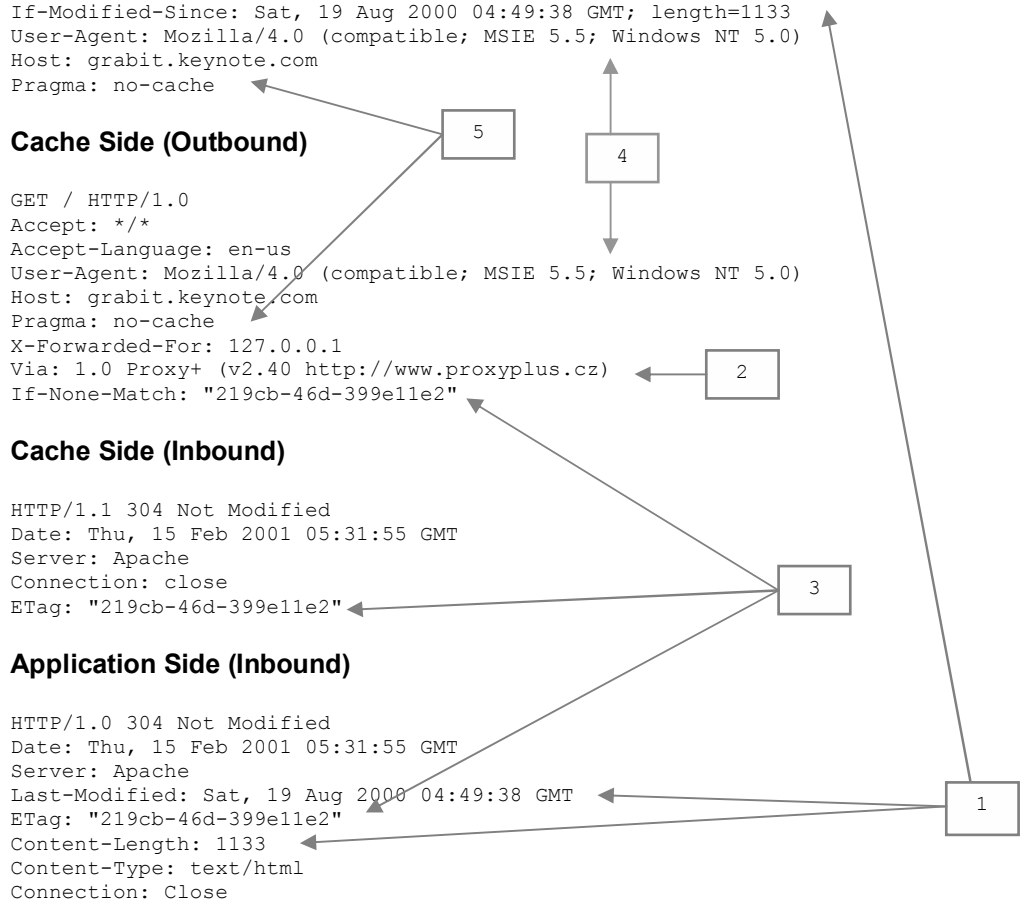
```
GET / HTTP/1.0
Accept: */*
Accept-Language: en-us
User-Agent: Mozilla/4.0 (compatible; MSIE 5.5; Windows NT 5.0)
Host: grabit.keynote.com
Pragma: no-cache
X-Forwarded-For: 127.0.0.1
Via: 1.0 Proxy+ (v2.40 http://www.proxyplus.cz)
If-None-Match: "219cb-46d-399e11e2"
```

Cache Side (Inbound)

```
HTTP/1.1 304 Not Modified
Date: Thu, 15 Feb 2001 05:31:55 GMT
Server: Apache
Connection: close
ETag: "219cb-46d-399e11e2"
```

Application Side (Inbound)

```
HTTP/1.0 304 Not Modified
Date: Thu, 15 Feb 2001 05:31:55 GMT
Server: Apache
Last-Modified: Sat, 19 Aug 2000 04:49:38 GMT
ETag: "219cb-46d-399e11e2"
Content-Length: 1133
Content-Type: text/html
Connection: Close
```



1. “If-Modified-Since” and “Content-Length” headers are stripped by cache server on the outbound transmission and re-inserted by cache server when the server data is passed back to the application.
2. Cache server inserts “X-Forwarded” and “Via” headers, as required by HTTP standard. This can be removed in the server settings to make the use of a proxy server invisible to server.
3. Cache server uses “ETag” header, the unique file marker for all objects on a Web server, to ensure that the content has not changed since last retrieval.
4. “User-Agent” string is passed intact through the cache server. This helps prevent any issues with browser-specific content as the cache server appears to be the browser it is acting as a proxy for.
5. The “Pragma: no-cache” header appears as the process outlined above resulted from a refresh of content initiated by the browser. This header forces the request to go directly to the origin server in order to ensure that the content held by the cache server is still valid and not stale.

Item 5 listed above is the most important item to note here. The only reason an outbound network request to the origin server was initiated was that a page refresh was requested by the browser application. If the browser (or another browser application, or another machine using the same cache) initiated a request for the same object after it had been re-loaded into the cache, no outbound network traffic seen.

This was tested using a cache server used by all browser applications being run on the same machine as the cache server. When a second browser application requested a page that had been previously cached by another browser application, no network traffic was generated, as the local cache server handled the delivery of the cached content internally.