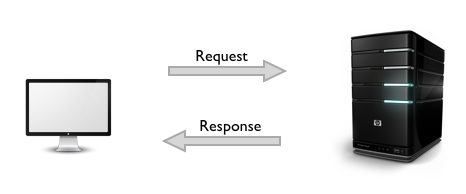
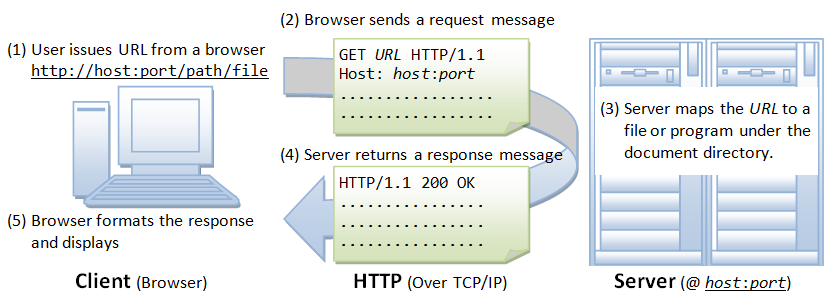
**Hyper Text Transfer Protocol (HTTP)**

1. **Introduction**
   1. **What is HTTP?**

* HTTP stands for Hypertext Transfer Protocol.
* This is the foundation for data communication for the World Wide Web (i.e. internet).
* HTTP allows for communication between a variety of hosts and clients, and supports a mixture of network configurations.
* The HTTP is an application layer protocol that allows web based applications to communicate and exchange data.
* It is the messanger of the web.
* It is used to deliver content, fo example images, videos, audios, documents etc.
* The computers that communicate through HTTP, must speak the http protocol.
* It defines how the messages are modified and transfered, what actions web servers and browsers should take in response to different commands.
* Example. You enter a URL in your browser, an HTTP command is sent to the web server directing it to fetch and transmit the requested web page.
* The HTTP protocol is a request/response protocol based on the client/server based architecture where web browsers, robots and search engines, etc. act like HTTP clients, and the Web server acts as a server.



* 1. **Basic features of HTTP:** There are three basic features that make HTTP a simple but powerful protocol:
* **HTTP is connectionless:** After making a request, the client disconnect from the server, then when the response is ready the server re-establish the connection again and deliver the response.
* **HTTP is media independent:** It means, any type of data can be sent by HTTP as long as both the client and the server know how to handle the data content.
* **HTTP is stateless:** The server and client are aware of each other only during a current request. Afterwards, both of them forget about each other. Due to this nature of the protocol, neither the client nor the browser can retain information between different requests across the web pages.
  1. **How HTTP works**
* The HTTP protocol is a **request/response protocol** based on the client/server based architecture where web browsers, robots and search engines, etc. act like HTTP clients, and the Web server acts as a server.
* **Client Request:** The HTTP client sends a request message, formatted according to the rule of HTTP standard - The **HTTP request**. This message specifies the resources that the client wishes to retrive or includes information to be provided by the server.
* **Server Response:** The HTTP server reads and interprets the message. It takes action relevant to the request and creates an **HTTP Response Message,** which it sends back to the client. The response message indicates whether the request was successful, and may also contain the content of the resource that the client requested, if appropriate.
* In otherwords, communication between a server and a client occurs, via a request/response pair. The client initiates an HTTP request message, which is serviced through a HTTP response message in return.
* Whenever you issue a URL from your browser to get a web resource using HTTP, e.g. http://www.nowhere123.com/index.html, the browser turns the URL into a request message and sends it to the HTTP server. The HTTP server interprets the request message, and returns you an appropriate response message, which is either the resource you requested or an error message. This process is illustrated below:



* **Uniform Resource Locator (URL):** A URL is used to uniquely identify a resource over the web. URL has the following syntax:

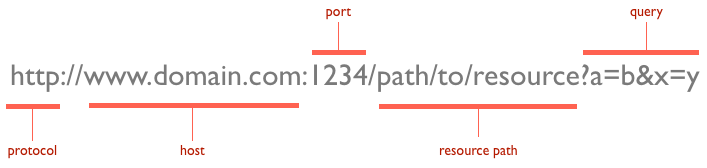
**protocol://hostname:port/path-and-file-name**

There are 4 parts in a URL:

1. **Protocol:** The application-level protocol used by the client and server, e.g., HTTP, FTP, and telnet.
2. **Hostname:** The DNS domain name (e.g., www.nowhere123.com) or IP address (e.g., 192.128.1.2) of the server.
3. **Port:** The TCP port number that the server is listening for incoming requests from the clients.
4. **Path-and-file-name:** The name and location of the requested resource, under the server document base directory.

For example, in the URL **http://www.nowhere123.com/docs/index.html**, the communication protocol is HTTP; the hostname is www.nowhere123.com. The port number was not specified in the URL, and takes on the default number, which is TCP port 80 for HTTP. The path and file name for the resource to be located is "/docs/index.html".

Other examples of URL are: <ftp://www.ftp.org/docs/test.txt,> mailto:user@test101.com, news:soc.culture.Singapore, telnet://www.nowhere123.com/



The default port is 80, but one can be set explicitly, as illustrated in the above image. The resource path is the local path to the resource on the server.

* Whenever you enter a URL in the address box of the browser, the browser translates the URL into a request message according to the specified protocol; and sends the request message to the server. Let the URLis <http://www.nowhere123.com/doc/index.html.>

|  |  |
| --- | --- |
| **Client (Browser)** | **Server** |
| the browser translated the URL http://www.nowhere123.com/doc/index.html into the following request message: | HTTP response message is as shown: |
| GET /docs/index.html HTTP/1.1  Host: www.nowhere123.com  Accept: image/gif, image/jpeg, \*/\*  Accept-Language: en-us  Accept-Encoding: gzip, deflate  User-Agent: Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1)  (blank line) | HTTP/1.1 200 OK  Date: Sun, 18 Oct 2009 08:56:53 GMT  Server: Apache/2.2.14 (Win32)  Last-Modified: Sat, 20 Nov 2004 07:16:26 GMT  ETag: "10000000565a5-2c-3e94b66c2e680"  Accept-Ranges: bytes  Content-Length: 44  Connection: close  Content-Type: text/html  X-Pad: avoid browser bug    <html><body><h1>It works!</h1></body></html> |
|  |  |

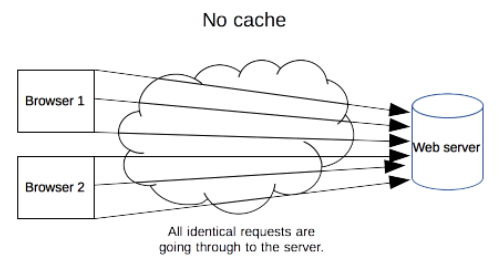
* The browser receives the response message, interprets the message and displays the contents of the message on the browser's window according to the media type of the response (as in the Content-Type response header). Common media type include "text/plain", "text/html", "image/gif", "image/jpeg", "audio/mpeg", "video/mpeg", "application/msword", and "application/pdf".

**Web Caching**

Web Caches and Proxy Server. Advantages of thesame, Average response time with and without proxy server through example, Problem with proxy cache, Why Cookies?, How cookies are maintained at both client and server.

1. **Web Caches**
   1. **What is web caching?**

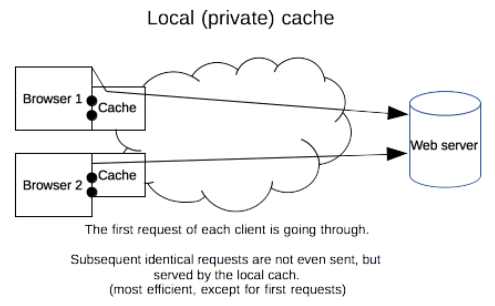
* **Caching** is **storing something temporarily** for fast retrieval later on.
* **Web Caching** is **storing of HTTP responses temporarily** for fast retrieval later on.
* A Web cache sits between one or more Web servers (also known as origin servers) and a client or many clients, and watches requests come by, saving copies of the responses — like HTML pages, images and files (collectively known as representations) — for itself. Then, if there is another request for the same URL, it can use the response that it has, instead of asking the origin server for it again.
  1. **Advantages of Web Caching**
* **To reduce latency** – Because the request is satisfied from the cache (which is closer to the client) instead of the origin server, it takes less time for the client to get the object and display it. This makes Web sites seem more responsive.
* **To reduce traffic** –Web caching reduces the number of requests made to the server. Due to which less bandwidth is consumed by clients and web server load is reduced. This saves money if the client is paying for traffic, and keeps their bandwidth requirements lower and more manageable.
  1. **Systems using web caching :** Systems have their different purpose of web caching.
* Search Engines
* Web Browsers,
* Content Delivery Networks
* Web Proxies are some systems which widely cache web files.
  1. **What Can be Cached?**
* Certain content lends itself more readily to caching than others. These tend to change infrequently, so they can benefit from being cached for longer periods of time. Some very cache-friendly content for most sites are:
* Logos and brand images
* Non-rotating images in general (navigation icons, for example)
* Style sheets
* General Javascript files
* Downloadable Content
* Media Files
* Some items that you have to be careful in caching are:
* HTML pages
* Rotating images
* Frequently modified Javascript and CSS
* Content requested with authentication cookies
* Some items that should almost never be cached are:
* Assets related to sensitive data (banking info, etc.)
* Content that is user-specific and frequently changed
  1. **Kinds of Web Caches**



(Ref-https://developer.mozilla.org/en-US/docs/Web/HTTP/Caching)

1. **Private Caches:**

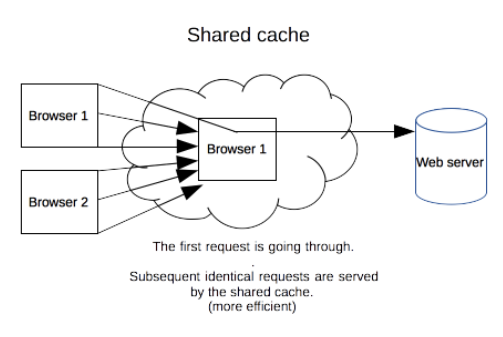
* A private cache is dedicated to a single user.
* For example, a browser cache holds all documents downloaded via HTTP by the user. A section of your computer’s hard disk is used to store such downloads that you’ve seen, just for you later use.
* This cache is used to make visited documents available for back/forward navigation, saving, viewing-as-source, etc. without requiring an additional trip to the server.
* It likewise improves offline browsing of cached content.The lets you set aside



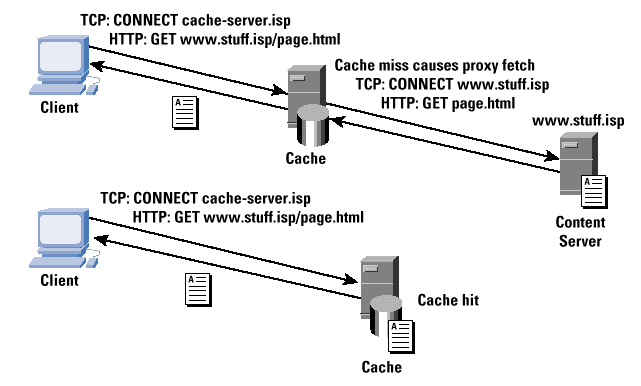
(Ref-https://developer.mozilla.org/en-US/docs/Web/HTTP/Caching)

1. **Shared Caches**

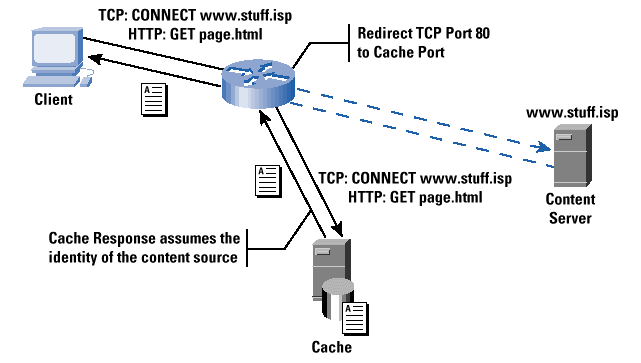
* A shared cache is a cache that stores responses to be reused by more than one user.
* For example, web proxy caches work on the same principle like browser caches, but a much larger scale.
* For example, an ISP or your company might have set up a web proxy as part of its local network infrastructure to serve many users so that popular resources are reused a number of times, reducing network traffic and latency.
* Because proxy caches aren’t part of the client or the origin server, but instead are out on the network, requests have to be routed to them somehow. One way to do this is to use your browser’s proxy setting to manually tell it what proxy to use; another is using interception. Interception proxies have Web requests redirected to them by the underlying network itself, so that clients don’t need to be configured for them, or even know about them.
* Proxy caches are a type of shared cache; rather than just having one person using them, they usually have a large number of users, and because of this they are very good at reducing latency and network traffic. That’s because popular representations are reused a number of times.



(Ref-https://developer.mozilla.org/en-US/docs/Web/HTTP/Caching)



(Ref.https://www.cisco.com/c/en/us/about/press/internet-protocol-journal/back-issues/table-contents-2/ipj-archive/article09186a00800c8903.html)



(Ref.https://www.cisco.com/c/en/us/about/press/internet-protocol-journal/back-issues/table-contents-2/ipj-archive/article09186a00800c8903.html)

1. **Gateway Caches**

* Also known as “reverse proxy caches” or “surrogate caches,” gateway caches are also intermediaries, but instead of being deployed by network administrators to save bandwidth, they’re typically deployed by Webmasters themselves, to make their sites more scalable, reliable and better performing.
* Requests can be routed to gateway caches by a number of methods, but typically some form of load balancer is used to make one or more of them look like the origin server to clients.
* **Content delivery networks (CDNs)** distribute gateway caches throughout the Internet (or a part of it) and sell caching to interested Web sites. Speedera and Akamai are examples of CDNs.
  1. **Terminology:** When dealing with caching, there are a few terms that you are likely to come across that might be unfamiliar. Some of the more common ones are below:
* ****Origin server****: The origin server is the original location of the content. If you are acting as the web server administrator, this is the machine that you control. It is responsible for serving any content that could not be retrieved from a cache along the request route and for setting the caching policy for all content.
* ****Cache hit ratio****: A cache's effectiveness is measured in terms of its cache hit ratio or hit rate. This is a ratio of the requests able to be retrieved from a cache to the total requests made. A high cache hit ratio means that a high percentage of the content was able to be retrieved from the cache. This is usually the desired outcome for most administrators.
* ****Freshness****: Freshness is a term used to describe whether an item within a cache is still considered a candidate to serve to a client. Content in a cache will only be used to respond if it is within the freshness time frame specified by the caching policy.
* ****Stale content****: Items in the cache expire according to the cache freshness settings in the caching policy. Expired content is "stale". In general, expired content cannot be used to respond to client requests. The origin server must be re-contacted to retrieve the new content or at least verify that the cached content is still accurate.
* ****Validation****: Stale items in the cache can be validated in order to refresh their expiration time. Validation involves checking in with the origin server to see if the cached content still represents the most recent version of item.
* ****Invalidation****: Invalidation is the process of removing content from the cache before its specified expiration date. This is necessary if the item has been changed on the origin server and having an outdated item in cache would cause significant issues for the client.
  1. **How (and how not) to Control Caches (HTTP Cache Control Mechanism)**
* There are several tools that Web designers and Webmasters can use to fine-tune how caches will treat their sites.
* The caching mechanism of these systems can be controlled using caching **meta tags or HTTP caching headers**. These systems do caching to decrease the bandwidth usage and also decrease web server overload.
* **HTML Meta Tags :** HTML authors can put tags in a document’s <HEAD> section that describe its attributes. These meta tags are often used in the belief that they can mark a document as uncacheable, or expire it at a certain time. Meta tags are easy to use, but aren’t very effective. That’s because they’re only honored by a few browser caches, not proxy caches (which almost never read the HTML in the document). While it may be tempting to put a Pragma: no-cache meta tag into a Web page, it won’t necessarily cause it to be kept fresh.
* **HTTP Headers:** On the other hand, true HTTP headers give you a lot of control over how both browser caches and proxies handle your representations. They can’t be seen in the HTML, and are usually automatically generated by the Web server. However, you can control them to some degree, depending on the server you use. HTTP headers are sent by the server before the HTML, and only seen by the browser and any intermediate caches.
* Caching policy is dependent upon two different factors. The caching entity itself gets to decide whether or not to cache acceptable content. It can decide to cache less than it is allowed to cache, but never more.
* The majority of caching behavior is determined by the caching policy, which is set by the content owner. These policies are mainly articulated through the use of specific HTTP headers.
* Through various iterations of the HTTP protocol, a few different cache-focused headers have arisen with varying levels of sophistication. The ones you probably still need to pay attention to are below:

|  |  |  |
| --- | --- | --- |
| **Sl. No.** | **Header Name** | **Description** |
| 1 | **Expires** | The Expires header is very straight-forward, although fairly limited in scope. Basically, it **sets a time in the future when the content will expire**. At this point, any requests for the same content will have to go back to the origin server. This header is probably best used only as a fall back.  If Expires header is assigned an future date and time then the response is cache till that time and requests are not made to the server. But if it is assigned to a past time or -1 then these systems do not cache the response. Expires header has no way to instruct client to revalidate cache. Even if we provide Expires header with Last-Modified the client will not revalidate the cache. |
| 2 | **Cache-Control** | This is the more **modern replacement for the Expires** header. It is well supported and implements a much more flexible design. In almost all cases, this is preferable to Expires, but it may not hurt to set both values. |
| 3 | **Etag** | The Etag header is **used with cache validation.** The origin can provide a unique Etag for an item when it initially serves the content. When a cache needs to validate the content it has on-hand upon expiration, it can send back the Etag it has for the content. The origin will either tell the cache that the content is the same, or send the updated content (with the new Etag). |
| 4 | **Last-Modified** | This header **specifies the last time** that the item was modified. This may be used as part of the validation strategy to ensure fresh content. |
| 5 | **Content-Length** | While not specifically involved in caching, the Content-Length header is important to set when defining caching policies. Certain software will refuse to cache content if it does not know in advanced the size of the content it will need to reserve space for. |
| 6 | **Vary** | A cache typically uses the requested host and the path to the resource as the key with which to store the cache item. The Vary header can be **used to tell caches to pay attention to an additional header when deciding whether a request is for the same item.** This is most commonly used to tell caches to key by the Accept-Encoding header as well, so that the cache will know to differentiate between compressed and uncompressed content. |

* 1. **The HTTP Cache Control Header used for Cache Mechanism**
* The **Cache-Control HTTP/1.1 header** is used to **specify directives for caching mechanisms** in both requests and responses. A number of different policy instructions can be set using this header, with multiple instructions being separated by commas. Use this header to define your caching policies with the variety of directives it provides.
* Caches used in clients, servers, and network (proxy servers, content delivery networks)
* **Cache-Control (HTTP/1.1)**

no-store - never cache this message

no-cache - may cache but need revalidation

public - public may cache

Private - intended for single user

max-age - set expiration

must-revalidate - require revalidation

* **HTTP/1.0:**

Expires: Thu, 01 Jan 1970 00:00:00 GMT

Pragma: no-cache

|  |  |  |  |
| --- | --- | --- | --- |
| **Sl. No.** | **Directives & Description** | **Sl. No.** | **Directives & Description** |
| 1 | **no-cache**   * It forces a cache to send the request to the origin server for validation before releasing a cached copy to the client, every time. * This, in effect, marks the content as stale immediately, but allows it to use revalidation techniques to avoid re-downloading the entire item again |  | **no-cache**   * Same   **Example**  **Cache-Control: no-cache** |
| 2 | **no-store**   * The cache should not store anything about the client request or server response. * A request is sent to the server and a full response is downloaded each and every time. * This is appropriate to set if the response represents sensitive data. * The no-store option supersedes the no-cache if both are present. |  | **no-store**   * Same   **Example**  **Cache-Control: no-store**  **Cache-Control: no-store, no-cache** |
| 3 | **max-age = seconds**   * It specifies in seconds, the amount of time the response will be cached. After the response has expired its deleted from the cache. * In otherwords, it specifies the maximum amount of time in seconds that a cached response will be considered as fresh. * This option takes its value in seconds with a maximum valid freshness time of one year (31536000 seconds). |  | **max-age = seconds**   * **Same**   **Example**  **Cache-Control: max-age=100** |
| 4 | **no-transform**   * This option tells caches that they are not allowed to modify the received content for performance reasons under any circumstances. This means, for instance, that the cache is not able to send compressed versions of content it did not receive from the origin server compressed and is not allowed. |  | **no-transform**   * **Same**   **Example**   * **Cache-Control: no-transform** |
| 5 | **max-stale [ = seconds ]**   * It indicates that the client is willing to accept a response that has exceeded its expiration time. If seconds are given, it must not be expired by more than that time.   **Example**  **Cache-Control: max-stale=500** |  | **public**   * It indicates that the response may be cached by any cache (private or shared)   **Example**   * **Cache-Control: public** |
| 6 | **min-fresh = seconds**   * It indicates that the client is willing to accept a response whose freshness lifetime is not less than its current age plus the specified time in seconds.   **Example**  **Cache-Control: min-fresh=300** |  | **private**   * It indicates that all or part of the response message is intended for a single user and must not be cached by a shared cache. * Responses are by default private.   **Example**  **Cache-Control: private** |
| 7 | **only-if-cached**   * It does not retrieve new data. * The cache can send a document only if it is in the cache, and should not contact the origin-server to see if a newer copy exists.   **Example**   * **Cache-Control: only-if-cached** |  | **must-revalidate**   * This indicates that the freshness information indicated by max-age, s-maxage or the Expires header must be obeyed strictly. Stale content cannot be served under any circumstance. This prevents cached content from being used in case of network interruptions and similar scenarios. * When both Cache-Control and Expires are present, Cache-Control takes precedence.   **Example**  **Cache-Control:must-revalidate** |
| 8 |  |  | **proxy-revalidate**   * It is same as must-revalidate but for proxy servers.   **Example**   * **Cache-Control: procxy-revalidate** |
| 9 |  |  | **s-maxage = seconds**   * Similar to max-age, except that it only applies to shared (e.g., proxy) caches. * The maximum age specified by this directive overrides the maximum age specified by either the max-age directive or the Expires header. The s-maxage directive is always ignored by a private cache.   **Example**   * **Cache-Control: s-maxage=1000** |
| 10 |  |  |  |

**Example:**

|  |  |
| --- | --- |
| **Cache-Control : max-age=86400**  **Cache-Control : max-age=60** | Response can be cached by browser and any intermediary caches (that is, it's "public") for up to 1 day (60 seconds x 60 minutes x 24 hours).  This indicates that the response can be cached and reused for the next 60 seconds. |
| **Cache-Control : private, max-age=600** | Response can be cached by the client’s browser only for up to 10 minutes (60 seconds x 10 minutes). |
| **Cache-Control : no-store** | Response is not allowed to be cached and must be fetched in full on every request. |
| **Expires: -1**  **Cache-Control: no-store** | Here these systems will not cache the response. |
| **Expires: Thu, 15 Aug 2060 09:00:00 GMT**  **Cache-Control: no-cache, must-revalidate, expires=360000000** | Here these systems will cache the response but before serving the response the client will try to revalidate but as we didn’t provide Last-Modified header, client will send revalidation request without If-Modified-Since and therefore server will response with 200 status code which is refetching the page again. |
| **Expires: Thu, 15 Aug 2015 09:00:00 GMT**  **Last-Modified: Thu, 15 Aug 2011 09:00:00 GMT** | Here browser will cache the document till 15 Aug 2015 09:00:00. Client will not revalidate the cache before serving. |
| **If-Modified-Since: Mon, 15 August 2003 00:00:00 GMT** | Now the server sees the If-Modified-Since header and checks if it was modified since then or not. If its modified than it returns a normal 200 success response. And can incude Last-Modified header if needed again. But if its not modified than server returns 304 Page Not Modified response. On return of 304 response, search engines consider the previously indexed information to be still fresh and valid. |

<http://qnimate.com/all-about-web-caching/>

1. **Defining optimal Cache-Control policy**. : Follow the decision tree below to determine the optimal caching policy for a particular resource, or a set of resources, that your application uses. Ideally, you should aim to cache as many responses as possible on the client for the longest possible period, and provide validation tokens for each response to enable efficient revalidation.



**How is expiration time calculated**

expirationTime = responseTime + freshnessLifetime – currentAge

The freshness lifetime is calculated based on several headers. If a “Cache-control: max-age=N” header is specified, then the freshness lifetime is equal to N. If this header is not present, which is very often the case, then we look for an”Expires” header. If an “Expires” header exists, then its value minus the value of the “Date” header determines the freshness lifetime. Finally, if neither header is present, then we look for a “Last-Modified” header. If this header is present, then the cache’s freshness lifetime is equal to the value of the “Date” header minus the value of the “Last-modified” header divided by 10. If none of this headers are there then the response is not cached.

responseTime is the time at which the response was received according to the client.

The current age is usually close to zero, but is influenced by the presence of an Age header, which proxy caches may add to indicate the length of time a document has been sitting in its cache. The precise algorithm, which attempts avoid error resulting from clock skew, is described in RFC 2616 section 13.2.3.

(Ref-https://developers.google.com/web/fundamentals/performance/optimizing-content-efficiency/http-caching)

**Typical HTTP 1.1 response headers might look like this:**

**HTTP/1.1 200 OK**

**Date: Fri, 30 Oct 1998 13:19:41 GMT**

**Server: Apache/1.3.3 (Unix)**

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

**Expires: Fri, 30 Oct 1998 14:19:41 GMT**

**Last-Modified: Mon, 29 Jun 1998 02:28:12 GMT**

**ETag: "3e86-410-3596fbbc"**

**Content-Length: 1040**

**Content-Type: text/html**

* 1. **Cache control :** HTTP defines three basic mechanisms for controlling caches: freshness, validation, and invalidation.

1. **Freshness**

* This allows a response to be used without re-checking it on the origin server, and can be controlled by both the server and the client. For example, the **Expires response header** gives a date when the document becomes stale, and the **Cache-Control: max-age** directive tells the cache how many seconds the response is fresh for.

1. **Validation**

* Stale items in the cache can be validated in order to refresh their expiration time. This can be used to check whether a cached response is still good after it becomes stale.
* When using the "must-revalidate" directive, the cache must verify the status of the stale resources before using it and expired ones should not be used.
* For example, if the response has a Last-Modified header, a cache can make a conditional request using the If-Modified-Since header to see if it has changed.
* The ETag (entity tag) mechanism also allows for both strong and weak validation.
* HTTP 1.1 introduced a new kind of validator called the ETag. ETags are unique identifiers that are generated by the server and changed every time the representation does. Because the server controls how the ETag is generated, caches can be sure that if the ETag matches when they make a If-None-Match request, the representation really is the same.
* Almost all caches use Last-Modified times as validators; ETag validation is also becoming prevalent.

1. **Invalidation**

* Invalidation is the process of removing content from the cache before its specified expiration date. This is necessary if the item has been changed on the origin server and having an outdated item in cache would cause significant issues for the client.
* This is usually a side effect of another request that passes through the cache. For example, if a URL associated with a cached response subsequently gets a POST, PUT or DELETE request, the cached response will be invalidated.

Most modern Web servers will generate both ETag and Last-Modified headers to use as validators for static content (i.e., files) automatically; you won’t have to do anything. However, they don’t know enough about dynamic content (like CGI, ASP or database sites) to generate them; see Writing Cache-Aware Scripts.

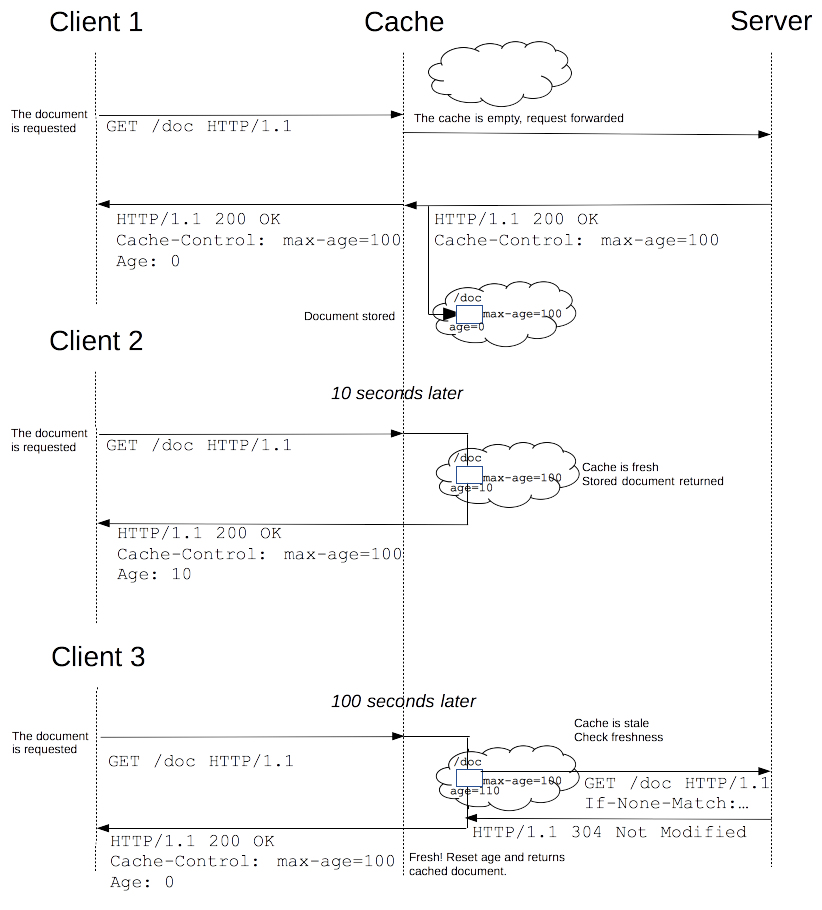
* 1. **Freshness**
* Once a resource is stored in a cache, it could theoretically be served by the cache forever. Caches have finite storage so items are periodically removed from storage. This process is called cache eviction. On the other side, some resources may change on the server so the cache should be updated. As HTTP is a client-server protocol, servers can't contact caches and clients when a resource changes; they have to communicate an expiration time for the resource. Before this expiration time, the resource is fresh; after the expiration time, the resource is stale. Eviction algorithms often privilege fresh resources over stale resources. Note that a stale resource is not evicted or ignored; when the cache receives a request for a stale resource, it forwards this request with a If-None-Match to check if it is in fact still fresh. If so, the server returns a 304 (Not Modified) header without sending the body of the requested resource, saving some bandwidth.
* The **freshness lifetime is calculated based on several headers.** If a "**Cache-control: max-age=N"** header is specified, then the freshness lifetime is equal to N. If this header is not present, which is very often the case, it is checked if an **Expires header** is present. If an Expires header exists, then its value minus the value of the Date header determines the freshness lifetime. Finally, if neither header is present, look for a **Last-Modified header.** If this header is present, then the cache's freshness lifetime is equal to the value of the Date header minus the value of the Last-modified header divided by 10.
* If none of this headers are there then the response is not cached. responseTime is the time at which the response was received according to the client.The current age is usually close to zero, but is influenced by the presence of an Age header, which proxy caches may add to indicate the length of time a document has been sitting in its cache.
* The expiration time is computed as follows:

**expirationTime = responseTime + freshnessLifetime - currentAge**

where responseTime is the time at which the response was received according to the

browser.

* Here is an example of this process with a shared cache proxy:



* 1. **FAQs**

1. **Which is better Last-Modified/If-Modified-Since or ETag/If-None-Match?**

* I prefer to use Last-Modified/If-Modified-Since because there are many clients that don’t support ETag/If-None-Match.

1. **What is must required for clients to cache?**

* Client need to know how long to hold the response in cache. You should provide expiration time or re-validation permission so that clients will cache a document. If none is provided then client will not at all cache the response. In short we can say that if client can calculate the expiration time then the response will be cached.
* Expiration time can be provided using Expires or max-age. Re-validation permission can be provided using no-cache, must-revalidate or Last-Modified.

1. **What are the most important things to make cacheable?**

* A good strategy is to identify the most popular, largest representations (especially images) and work with them first.

1. **How can I make my pages as fast as possible with caches?**

* The most cacheable representation is one with a long freshness time set. Validation does help reduce the time that it takes to see a representation, but the cache still has to contact the origin server to see if it’s fresh. If the cache already knows it’s fresh, it will be served directly.

1. **I understand that caching is good, but I need to keep statistics on how many people visit my page!**

* If you must know every time a page is accessed, select ONE small item on a page (or the page itself), and make it uncacheable, by giving it a suitable headers. For example, you could refer to a 1x1 transparent uncacheable image from each page. The Referer header will contain information about what page called it.
* Be aware that even this will not give truly accurate statistics about your users, and is unfriendly to the Internet and your users; it generates unnecessary traffic, and forces people to wait for that uncached item to be downloaded. For more information about this, see On Interpreting Access Statistics in the references.

1. **How can I see a representation’s HTTP headers?**

* Many Web browsers let you see the Expires and Last-Modified headers are in a “page info” or similar interface. If available, this will give you a menu of the page and any representations (like images) associated with it, along with their details.
* To see the full headers of a representation, you can manually connect to the Web server using a Telnet client.
* To do so, you may need to type the port (be default, 80) into a separate field, or you may need to connect to www.example.com:80 or www.example.com 80 (note the space). Consult your Telnet client’s documentation.
* Once you’ve opened a connection to the site, type a request for the representation. For instance, if you want to see the headers for http://www.example.com/foo.html, connect to www.example.com, port 80, and type:

GET /foo.html HTTP/1.1 [return]

Host: www.example.com [return][return]

Press the Return key every time you see [return]; make sure to press it twice at the

end. This will print the headers, and then the full representation. To see the headers

only, substitute HEAD for GET.

1. **My pages are password-protected; how do proxy caches deal with them?**

* By default, pages protected with HTTP authentication are considered private; they will not be kept by shared caches. However, you can make authenticated pages public with a Cache-Control: public header; HTTP 1.1-compliant caches will then allow them to be cached.
* If you’d like such pages to be cacheable, but still authenticated for every user, combine the Cache-Control: public and no-cache headers. This tells the cache that it must submit the new client’s authentication information to the origin server before releasing the representation from the cache. This would look like:
* Cache-Control: public, no-cache
* Whether or not this is done, it’s best to minimize use of authentication; for example, if your images are not sensitive, put them in a separate directory and configure your server not to force authentication for it. That way, those images will be naturally cacheable.

1. **Should I worry about security if people access my site through a cache?**

* https:// pages are not cached (or decrypted) by proxy caches, so you don’t have to worry about that. However, because caches store http:// responses and URLs fetched through them, you should be conscious about unsecured sites; an unscrupulous administrator could conceivably gather information about their users, especially in the URL.
* In fact, any administrator on the network between your server and your clients could gather this type of information. One particular problem is when CGI scripts put usernames and passwords in the URL itself; this makes it trivial for others to find and use their login.
* If you’re aware of the issues surrounding Web security in general, you shouldn’t have any surprises from proxy caches.

1. **I’m looking for an integrated Web publishing solution. Which ones are cache-aware?**

* It varies. Generally speaking, the more complex a solution is, the more difficult it is to cache. The worst are ones which dynamically generate all content and don’t provide validators; they may not be cacheable at all. Speak with your vendor’s technical staff for more information, and see the Implementation notes below.

1. **My images expire a month from now, but I need to change them in the caches now!**

* The Expires header can’t be circumvented; unless the cache (either browser or proxy) runs out of room and has to delete the representations, the cached copy will be used until then.
* The most effective solution is to change any links to them; that way, completely new representations will be loaded fresh from the origin server. Remember that any page that refers to these representations will be cached as well. Because of this, it’s best to make static images and similar representations very cacheable, while keeping the HTML pages that refer to them on a tight leash.
* If you want to reload a representation from a specific cache, you can either force a reload (in Firefox, holding down shift while pressing ‘reload’ will do this by issuing a Pragma: no-cache request header) while using the cache. Or, you can have the cache administrator delete the representation through their interface.

1. **I run a Web Hosting service. How can I let my users publish cache-friendly pages?**

* I prefer If you’re using Apache, consider allowing them to use .htaccess files and providing appropriate documentation.
* Otherwise, you can establish predetermined areas for various caching attributes in each virtual server. For instance, you could specify a directory /cache-1m that will be cached for one month after access, and a /no-cache area that will be served with headers instructing caches not to store representations from it.
* Whatever you are able to do, it is best to work with your largest customers first on caching. Most of the savings (in bandwidth and in load on your servers) will be realized from high-volume sites.

1. **I’ve marked my pages as cacheable, but my browser keeps requesting them on every request. How do I force the cache to keep representations of them?**

* I prefer Caches aren’t required to keep a representation and reuse it; they’re only required to not keep or use them under some conditions. All caches make decisions about which representations to keep based upon their size, type (e.g., image vs. html), or by how much space they have left to keep local copies. Yours may not be considered worth keeping around, compared to more popular or larger representations.
* Some caches do allow their administrators to prioritize what kinds of representations are kept, and some allow representations to be “pinned” in cache, so that they’re always available.

1. **I’m looking for an integr**

* I prefer

1. **I’m looking for an integr**

* I prefer

1. **I’m looking for an integr**

* I prefer

1. **I’m looking for an integr**

* I prefer

1. **I’m looking for an integr**

* I prefer

1. **I’m looking for an integr**

* I prefer

1. **Cookies**
   1. **What is a cookie and what does it do?**

* Cookies are messages that web servers pass to your web browser when you visit Internet sites. Your browser stores each message in a small file, called **cookie.txt** . When you request another page from the server, your browser sends the cookie back to the server.
* These files typically contain information about your visit to the web page, as well as any information you've volunteered, such as your name and interests.
  1. **Why do we use cookies?**
* It is mostly used by Web Server as an ID that was used to identify YOU from others. It is also used by program (eg. JavaScript) to store a temporal data used in the program itself.
* **Cookies at Browser:** Cookies are most commonly used to track website activity. When you visit some sites, the server gives you a cookie that acts as your identification card. Upon each return visit to that site, your browser passes that cookie back to the server. In this way, a web server can gather information about which web pages are used the most, and which pages are gathering the most repeat hits. Cookies are also used for online shopping. Online stores often use cookies that record any personal information you enter, as well as any items in your electronic shopping cart, so that you don't need to re-enter this information each time you visit the site.
* **Cookies at Server**: Servers can use cookies to provide personalized web pages. When you select preferences at a site that uses this option, the server places the information in a cookie. When you return, the server uses the information in the cookie to create a customized page for you.
  1. **What is a cookie policy?**
* The Cookie Law is a piece of privacy legislation that requires websites to get consent from visitors to store or retrieve any information on a computer, smartphone or tablet. Almost all websites use cookies – little data files – to store information in people’s web browsers.
  1. **Are Cookies secure?**
* Cookies that are sent over HTTP (port 80) are not secure as the HTTP protocol is not encrypted. Cookies sent over HTTPS (port 443) are secure as HTTPS is encrypted. So, if Facebook sends/receives cookies via HTTP, they can be stolen and used nefariously. That’s why you should always check for a green padlock right beside your browser’s url bar. A website without a green padlock is insecure and all data that are sent and received can be hijacked with Man-in-the-middle attack.
  1. **Are Cookies bad?**
* No, cookies are stored in txt files which will never cause any trouble to your device.
  1. **How cache & cookies work**
* Cookies are files created by sites you visit. They make your online experience easier by saving browsing data.
* The cache remembers parts of pages, like images, to help them open faster during your next visit.