Hypertext Transfer Protocol -- HTTP/1.1
Bob Singh

Source - www.w3c.org
Introduction
The Hypertext Transfer Protocol (HTTP) is an application-level protocol for distributed, collaborative, hypermedia information systems. HTTP has been in use by the World-Wide Web global information initiative since 1990.
The first version of HTTP, referred to as HTTP/0.9, was a simple protocol for raw data transfer across the Internet.
HTTP/1.0, as defined by RFC 1945 [6], improved the protocol by allowing messages to be in the format of MIME-like messages, containing metainformation about the data transferred and modifiers on the request/ response semantics.
However, HTTP/1.0 does not sufficiently take into consideration the effects of hierarchical proxies, caching, the need for persistent connections, and virtual hosts. In addition, the proliferation of incompletely-implemented applications calling themselves "HTTP/1.0" has necessitated a protocol version change in order for two communicating applications to determine each other's true capabilities.
This specification defines the protocol referred to as "HTTP/1.1". This protocol includes more stringent requirements than HTTP/1.0 in order to ensure reliable implementation of its features.
Practical information systems require more functionality than simple retrieval, including search, front-end update, and annotation. HTTP allows an open-ended set of methods that indicate the purpose of a request. It builds on the discipline of reference provided by the Uniform Resource Identifier (URI) , as a location (URL) or name (URN) , for indicating the resource to which a method is to be applied. Messages are passed in a format similar to that used by Internet mail as defined by the Multipurpose Internet Mail Extensions (MIME).
HTTP is also used as a generic protocol for communication between user agents and proxies/gateways to other Internet systems, including those supported by the SMTP , NNTP , FTP , Gopher , and WAIS protocols. In this way, HTTP allows basic hypermedia access to resources available from diverse applications.

HTTP Transaction
All HTTP transaction follow the same format for client and server transactions.
Each client transaction has 3 parts:
1. The request or response line. - The client contact the server on designated port number by specifying an HTTP Command called a method. For e.g. GET /index.html HTTP/1.1
2. A Header Section. - The client sends the optional header information to inform the server of its configuration and the document formats it will accept.
User-Agent: Mozilla/4/05(WinNT: I) Accept: image/gif, image/x-xbitmap, image/jpeg, image/pjpeg */*
A blank line is sent at the end of the header.
3. After the header information additional information - data for CGI programs etc might be sent using the POST method.
Each server transaction has three parts:
1. The server replies with a status containing the three fields: HTTP Version, status code and description. Status code is a 3 digit number indicating the server result to the client request. For eg
    HTTP/1.1 200 OK

2. After the status line the server sends header information to the client about itself and the requested document. A blank line ends with the header.
3. If the client request is successful, the requested data is sent. The data can be an output from a CGI program. If the client request cannot be fulfilled, the additional data may be human-readable explanation of why the sever could not fulfill the request.

HTTP Methods
Get, Head, Post, Link, UnLink, Put, Delete, Options, Trace, Connect
Get - is a request for information located at a specified URI on the server. Get asks for a file usually. Head - is a request like get but doesn't send back the entire file but only returns the header information. Post - is a request that allows data to be sent to the server on a client request. The data is typically passed to the data handling program from the webserver - eg CGI, etc. Link - Requests the header information be associated with a document on the server UnLink - Requests the header information be disassociated with a document on the server Put - Requests the entire body of the request to be stored at the specified URI Delete - Request the removal of data at the URI on the Server. Options - Requests information about communication options available on the Server Trace - Requests the request entire bod be returned intact. Used for Debugging. Connect - A reserved method for SSL Tunneling.

Terminology
This specification uses a number of terms to refer to the roles played by participants in, and objects of, the HTTP communication.
connection A transport layer virtual circuit established between two programs for the purpose of communication.
message The basic unit of HTTP communication, consisting of a structured sequence of octets matching the syntax defined in section 4 and transmitted via the connection.
request An HTTP request message, as defined in section 5.
response An HTTP response message, as defined in section 6.
resource A network data object or service that can be identified by a URI. Resources may be available in multiple representations (e.g. multiple languages, data formats, size, resolutions) or vary in other ways.
entity The information transferred as the payload of a request or response. An entity consists of meta information in the form of entity-header fields and content in the form of an entity-body, as described in section 7.
representation An entity included with a response that is subject to content negotiation. There may exist multiple representations associated with a particular response status.
content negotiation The mechanism for selecting the appropriate representation when servicing a request. The representation of entities in any response can be negotiated (including error responses).
variant A resource may have one, or more than one, representation(s) associated with it at any given instant. Each of these representations is termed a `variant.' Use of the term `variant' does not necessarily imply that the resource is subject to content negotiation.
client A program that establishes connections for the purpose of sending requests.
user agent The client which initiates a request. These are often browsers, editors, spiders (web-traversing robots), or other end user tools.
server An application program that accepts connections in order to service requests by sending back responses. Any given program may be capable of being both a client and a server; our use of these terms refers only to the role being performed by the program for a particular connection, rather than to

the program's capabilities in general. Likewise, any server may act as an origin server, proxy, gateway, or tunnel, switching behavior based on the nature of each request.

origin server The server on which a given resource resides or is to be created.

proxy An intermediary program which acts as both a server and a client for the purpose of making requests on behalf of other clients. Requests are serviced internally or by passing them on, with possible translation, to other servers. A proxy must implement both the client and server requirements of this specification.

gateway A server which acts as an intermediary for some other server. Unlike a proxy, a gateway receives requests as if it were the origin server for the requested resource; the requesting client may not be aware that it is communicating with a gateway.

tunnel An intermediary program which is acting as a blind relay between two connections. Once active, a tunnel is not considered a party to the HTTP communication, though the tunnel may have been initiated by an HTTP request. The tunnel ceases to exist when both ends of the relayed connections are closed.

cache A program's local store of response messages and the subsystem that controls its message storage, retrieval, and deletion. A cache stores cachable responses in order to reduce the response time and network bandwidth consumption on future, equivalent requests. Any client or server may include a cache, though a cache cannot be used by a server that is acting as a tunnel.

cachable A response is cachable if a cache is allowed to store a copy of the response message for use in answering subsequent requests. The rules for determining the cachability of HTTP responses are defined in section 13. Even if a resource is cachable, there may be additional constraints on whether a cache can use the cached copy for a particular request.

first-hand A response is first-hand if it comes directly and without unnecessary delay from the origin server, perhaps via one or more proxies. A response is also first-hand if its validity has just been checked directly with the origin server.

explicit expiration time The time at which the origin server intends that an entity should no longer be returned by a cache without further validation.

heuristic expiration time An expiration time assigned by a cache when no explicit expiration time is available.

age The age of a response is the time since it was sent by, or successfully validated with, the origin server.

freshness lifetime The length of time between the generation of a response and its expiration time.

fresh A response is fresh if its age has not yet exceeded its freshness lifetime.

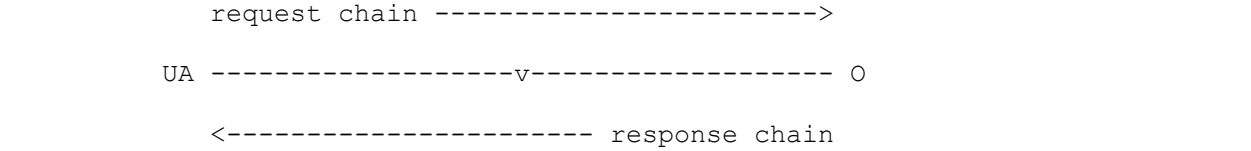
stale A response is stale if its age has passed its freshness lifetime.

semantically transparent A cache behaves in a "semantically transparent" manner, with respect to a particular response, when its use affects neither the requesting client nor the origin server, except to improve performance. When a cache is semantically transparent, the client receives exactly the same response (except for hop-by-hop headers) that it would have received had its request been handled directly by the origin server.

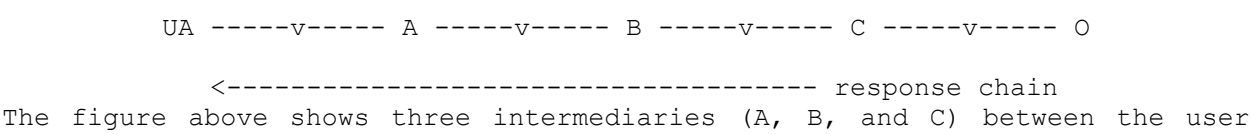
validator A protocol element (e.g., an entity tag or a Last-Modified time) that is used to find out whether a cache entry is an equivalent copy of an entity.

Overall Operation

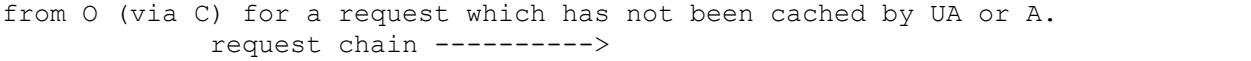
The HTTP protocol is a request/response protocol. A client sends a request to the server in the form of a request method, URI, and protocol version, followed by a MIME-like message containing request modifiers, client information, and possible body content over a connection with a server. The server responds with a status line, including the message's protocol version and a success or error code, followed by a MIME-like message containing server information, entity metainformation, and possible entity-body content.
Most HTTP communication is initiated by a user agent and consists of a request to be applied to a resource on some origin server. In the simplest case, this may be accomplished via a single connection (v) between the user agent (UA) and the origin server (O).

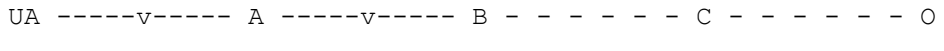


A more complicated situation occurs when one or more intermediaries are present in the request/response chain.
There are three common forms of intermediary: **proxy**, **gateway**, **and tunnel.**
A **proxy** is a forwarding agent, receiving requests for a URI in its absolute form, rewriting all or part of the message, and forwarding the reformatted request toward the server identified by the URI.
A **gateway** is a receiving agent, acting as a layer above some other server(s) and, if necessary, translating the requests to the underlying server's protocol.
A **tunnel** acts as a relay point between two connections without changing the messages; tunnels are used when the communication needs to pass through an intermediary (such as a firewall) even when the intermediary cannot understand the contents of the messages.



The figure above shows three intermediaries (A, B, and C) between the user agent and origin server.
A request or response message that travels the whole chain will pass through four separate connections.
This distinction is important because some HTTP communication options may apply only to the connection
with the nearest, non-tunnel neighbor, only to the end-points of the chain, or to all
connections along the chain. Although the diagram is linear, each participant may be engaged in multiple,
simultaneous communications. For example, B may be receiving requests from many clients other than A,
and/or forwarding requests to servers other than C, at the same time that it is handling A's request.
Any party to the communication which is not acting as a tunnel may employ an internal cache for handling requests. The effect of a cache is that the request/response chain is shortened if one of the participants along the chain has a cached response applicable to that request. The following illustrates the resulting chain if B has a cached copy of an earlier response from O (via C) for a request which has not been cached by UA or A.





Not all responses are usefully cachable, and some requests may contain modifiers which place special requirements on cache behavior. HTTP requirements for cache behavior and cachable responses are defined in section 13.

In fact, there are a wide variety of architectures and configurations of caches and proxies currently being experimented with or deployed across the World Wide Web; these systems include national hierarchies of proxy caches to save transoceanic bandwidth, systems that broadcast or multicast cache entries, organizations that distribute subsets of cached data via CD-ROM, and so on. HTTP systems are used in corporate intranets over high-bandwidth links, and for access via PDAs with low-power radio links and intermittent connectivity. The goal of HTTP/1.1 is to support the wide diversity of configurations already deployed while introducing protocol constructs that meet the needs of those who build web applications that require high reliability and, failing that, at least reliable indications of failure.

HTTP communication usually takes place over TCP/IP connections. The default port is TCP 80, but other ports can be used. This does not preclude HTTP from being implemented on top of any other protocol on the Internet, or on other networks. HTTP only presumes a reliable transport; any protocol that provides such guarantees can be used; the mapping of the HTTP/1.1 request and response structures onto the transport data units of the protocol in question is outside the scope of this specification.

In HTTP/1.0, most implementations used a new connection for each request/ response exchange. In HTTP/1.1, a connection may be used for one or more request/response exchanges, although connections may be closed for a variety of reasons (see section 8.1).