

(excerpted from 1st edition pre-release)

Computer Networking

A Top-Down Approach Featuring the Internet

[James F. Kurose](#) and [Keith W. Ross](#)

1. Application Layer

1. [Principles of Application-Layer Protocols](#)
2. [The World Wide Web: HTTP](#)
3. [File Transfer: FTP](#)
4. [Electronic Mail in the Internet](#)
5. [The Internet's Directory Service: DNS](#)

Copyright 1996-2000 James F. Kurose and Keith W. Ross

2.2 The World Wide Web: HTTP

In the 1980s the Internet was used by researchers, academics and university students to login to remote hosts, to transfer files from local hosts to remote hosts and vice versa, to receive and send news, and to receive and send electronic mail. Although these applications were (and continue to be) extremely useful, the Internet was essentially unknown outside the academic and research communities. Then in early 1990s the Internet's killer application arrived on the scene -- the World Wide Web. The Web is the Internet application that caught the general public's eye. It is dramatically changing how people interact inside and outside their work environments. It has spawned thousands of start up companies. It has elevated the Internet from just one of many data networks (including online networks such as Prodigy, America On Line and Compuserve, national data networks such as Minitel/Transpac in France, and private X.25 and frame relay networks) to essentially the one and only data network.

History is sprinkled with the arrival of electronic communication technologies that have had major societal impacts. The first such technology was the telephone, invented in the 1870s. The telephone allowed two persons to orally communicate in real-time without being in the same physical location. It had a major impact on society -- both good and bad. The next electronic communication technology was broadcast radio/television, which arrived in the 1920s and 1930s. Broadcast radio/television allowed people to receive vast quantities of audio and video information. It also had a major impact on society -- both good and bad. The third major communication technology that has changed the way people live and work is the Web. Perhaps what appeals the most to users about the Web is that it is *on demand*. Users receive what they want, when they want it. This is unlike broadcast radio and television, which force users to "tune in" when the content provider makes the content available. In addition to being on demand, the Web has many other wonderful features that people love and cherish. It is enormously easy for any individual to make any available available over the Web; everyone can become a publisher at extremely low cost. Hyperlinks and search engines help us navigate through an ocean of Web sites. Graphics and animated graphics stimulate our senses. Forms, Java applets, Active X components, as well as many other devices enable us to interact with pages and sites. And more and more, the Web provides a menu interface to vast quantities of audio and video material stored in the Internet, audio and video that can be accessed on demand.

2.2.1 Overview of HTTP

The **Hypertext Transfer Protocol (HTTP)**, the Web's application-layer protocol, is at the heart of the Web. HTTP is implemented in two programs: a client program and server program. The client program and server programs, executing on different end systems, talk to each other by exchanging HTTP messages. HTTP defines the structure of these messages and how the client and server exchange the messages. Before explaining HTTP in detail, it is useful to review some Web terminology.

A **Web page** (also called a document) consists of objects. An **object** is a simply file -- such as a HTML file, a JPEG image, a GIF image, a Java applet, an audio clip, etc. -- that is addressable by a single URL. Most Web pages consist of a **base HTML file** and several referenced objects. For example, if a Web page contains HTML text and five JPEG images, then the Web page has six objects: the base HTML file plus the five images. The base HTML file references the other objects in the page with the objects' URLs. Each URL has two components: the host name of the server that houses the object and the object's path name. For example, the URL

`www.someSchool.edu/someDepartment/picture.gif`

has `www.someSchool.edu` for a host name and `/someDepartment/picture.gif` for a path name. A **browser** is a user agent for the Web; it displays to the user the requested Web page and provides numerous navigational and configuration features. Web browsers also implement the client side of HTTP. Thus, in the context of the Web, we will interchangeably use the words "browser" and "client". Popular Web browsers include [Netscape Communicator](#) and [Microsoft Explorer](#). A **Web server** houses Web objects, each addressable by a URL. Web servers also implement the server side of HTTP. Popular Web servers include [Apache](#) and [Microsoft Internet Information Server](#). (Netcraft provides a nice survey of Web server penetration [[Netcraft](#)].)

HTTP defines how Web clients (i.e., browsers) request Web pages from servers (i.e., Web servers) and how servers transfer Web pages to clients. We discuss the interaction between client and server in detail below, but the general idea is illustrated in Figure 2.2-1. When a user requests a Web page (e.g., clicks on a hyperlink), the browser sends HTTP

request messages for the objects in the page to the server. The server receives the requests and responds with HTTP response messages that contain the objects. Through 1997 essentially all browsers and Web servers implement version HTTP/1.0, which is defined in [RFC 1945](#). Beginning in 1998 Web servers and browsers began to implement version HTTP/1.1, which is defined in [RFC 2068](#). HTTP/1.1 is backward compatible with HTTP/1.0; a Web server running 1.1 can "talk" with a browser running 1.0, and a browser running 1.1 can "talk" with a server running 1.0. Because HTTP/1.1 is now dominant, henceforth when we refer to HTTP we are referring to HTTP/1.1

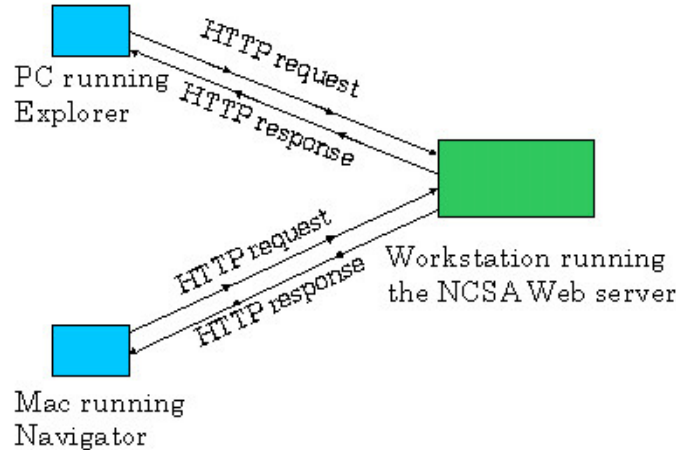


Figure 2.2-1: HTTP request-response behavior

HTTP uses TCP as their underlying transport protocol (rather than running on top of UDP). The HTTP client first initiates a TCP connection with the server. Once the connection is established, the browser and the server processes access TCP through their socket interfaces. As described in Section 2.1, on the client side the socket interface is the "door" between the client process and the TCP connection; on the server side it is the "door" between the server process and the TCP connection. The client sends HTTP request messages into its socket interface and receives HTTP response messages from its socket interface. Similarly, the HTTP server receives request messages from its socket interface and sends response messages into the socket interface. Once the client sends a message into its socket interface, the message is "out of the client's hands" and is "in the hands of TCP". Recall from Section 2.1 that TCP provides a reliable data transfer service to HTTP. This implies that each HTTP request message emitted by a client process eventually arrives in tact at the server; similarly, each HTTP response message emitted by the server process eventually arrives in tact at the client. Here we see one of the great advantages of a layered architecture - HTTP need not worry about lost data, or the details of how TCP recovers from loss or reordering of data within the network. That is the job of TCP and the protocols in the lower layers of the protocol stack.

It is important to note that the server sends requested files to clients without storing any state information about the client. If a particular client asks for the same object twice in a period of a few seconds, the server does not respond by saying that it just served the object to the client; instead, the server resends the object, as it has completely forgotten what it did earlier. Because an HTTP server maintains no information about the clients, HTTP is said to be a **stateless protocol**. We also remark that the Web uses the client-server application architecture. A Web server is always-on, with a fixed IP address and it services requests from potentially millions of different browsers.

2.2.2 Non-Persistent and Persistent Connections

HTTP can use both non-persistent connections and persistent connections. Non-persistent connections is the default mode for HTTP/1.0. Conversely, persistent connections is the default mode for HTTP/1.1.

Non-Persistent Connections

Let us walk through the steps of transferring a Web page from server to client for the case of non-persistent connections. Suppose the page consists of a base [HTML](#) file and 10 JPEG images, and that all 11 of these objects reside on the same server. Suppose the URL for the base HTML file is

www.someSchool.edu/someDepartment/home.index .

Here is what happens:

1. The HTTP client initiates a TCP connection to the server www.someSchool.edu. Port number 80 is used as the default port number at which the HTTP server will be listening for HTTP clients that want to retrieve documents using HTTP.
2. The HTTP client sends a HTTP request message into the socket associated with the TCP connection that was established in step 1. The request message either includes the entire URL or simply the path name </someDepartment/home.index>. (We will discuss the HTTP messages in some detail below.)
3. The HTTP server receives the request message via the socket associated with the connection that was established in step 1, retrieves the object </someDepartment/home.index> from its storage (RAM or disk), encapsulates the object in a HTTP response message, and sends the response message into the TCP connection.
4. The HTTP server tells TCP to close the TCP connection. (But TCP doesn't actually terminate the connection until the client has received the response message in tact.)
5. The HTTP client receives the response message. The TCP connection terminates. The message indicates that the encapsulated object is an HTML file. The client extracts the file from the response message, parses the HTML file and finds references to the ten JPEG objects.
6. The first four steps are then repeated for each of the referenced JPEG objects.

As the browser receives the Web page, it displays the page to the user. Two different browsers may interpret (i.e., display to the user) a Web page in somewhat different ways. HTTP has nothing to do with how a Web page is interpreted by a client. The HTTP specifications ([RFC 1945](#) and [RFC 2068](#)) only define the communication protocol between the client HTTP program and the server HTTP program.

The steps above use non-persistent connections because each TCP connection is closed after the server sends the object -- the connection does not persist for other objects. Note that each TCP connection transports exactly one request message and one response message. Thus, in this example, when a user requests the Web page, 11 TCP connections are generated.

In the steps described above, we were intentionally vague about whether the client obtains the 10 JPEGs over ten serial TCP connections, or whether some of the JPEGs are obtained over parallel TCP connections. Indeed, users can configure modern browsers to control the degree of parallelism. In their default modes, most browsers open five to ten parallel TCP connections, and each of these connections handles one request-response transaction. If the user prefers, the maximum number of parallel connections can be set to one, in which case the ten connections are established serially. As we shall see in the next chapter, the use of parallel connections shortens the response time since it cuts out some of the RTT and slow-start delays. Parallel TCP connections can also allow the requesting browser to steal a larger share of its fair share of the end-to-end transmission bandwidth.

Before continuing, let's do a back of the envelope calculation to estimate the amount of time from when a client requests the base HTML file until the file is received by the client. To this end we define the **round-trip time RTT**, which is the time it takes for a small packet to travel from client to server and then back to the client. The RTT includes packet propagation delays, packet queuing delays in intermediate routers and switches, and packet processing delays. (These delays were discussed in Section 1.6.) Now consider what happens when a user clicks on a hyperlink. This causes the browser to initiate a TCP connection between the browser and the Web server; this involves a "three-way handshake" -- the client sends a small TCP message to the server, the server acknowledges and responds with a small message, and finally the client acknowledges back to the server. One RTT elapses after the first two parts of the three-way handshake. After completing the first two parts of the handshake, the client sends the HTTP request message into the TCP connection, and TCP "piggybacks" the last acknowledgment (the third part of the three-way handshake) onto the request message. Once the request message arrives at the server, the server sends the HTML file into the TCP connection. This HTTP request/response eats up another RTT. Thus, roughly, the total response time is 2RTT plus the transmission time at the server of the HTML file.

Persistent Connections

Non-persistent connections have some shortcomings. First, a brand new connection must be established and maintained for *each requested object*. For each of these connections, TCP buffers must be allocated and TCP variables must be kept in both the client and server. This can place a serious burden on the Web server, which may be serving requests from

hundreds of different clients simultaneously. Second, as we just described, each object suffers two RTTs -- one RTT to establish the TCP connection and one RTT to request and receive an object. Finally, each object suffers from TCP slow start because every TCP connection begins with a TCP slow-start phase. However, the accumulation of RTT and slow start delays is partially alleviated by the use of parallel TCP connections.

With persistent connections, the server leaves the TCP connection open after sending responses. Subsequent requests and responses between the same client and server can be sent over the same connection. In particular, an entire Web page (in the example above, the base HTML file and the ten images) can be sent over a single persistent TCP connection; moreover, multiple Web pages residing on the same server can be sent over one persistent TCP connection. Typically, the HTTP server closes the connection when it isn't used for a certain time (the timeout interval), which is often configurable. There are two versions of persistent connections: **without pipelining** and **with pipelining**. For the version without pipelining, the client issues a new request only when the previous response has been received. In this case, each of the referenced objects (the ten images in the example above) experiences one RTT in order to request and receive the object. Although this is an improvement over non-persistent's two RTTs, the RTT delay can be further reduced with pipelining. Another disadvantage of no pipelining is that after the server sends an object over the persistent TCP connection, the connection hangs -- does nothing -- while it waits for another request to arrive. This hanging wastes server resources.

The default mode of HTTP/1.1 uses persistent connections with pipelining. In this case, the HTTP client issues a request as soon as it encounters a reference. Thus the HTTP client can make back-to-back requests for the referenced objects. When the server receives the requests, it can send the objects back-to-back. If all the requests are sent back-to-back and all the responses are sent back-to-back, then only one RTT is expended for all the referenced objects (rather than one RTT per referenced object when pipelining isn't used). Furthermore, the pipelined TCP connection hangs for a smaller fraction of time. In addition to reducing RTT delays, persistent connections (with or without pipelining) have a smaller slow-start delay than non-persistent connections. This is because that after sending the first object, the persistent server does not have to send the next object at the initial slow rate since it continues to use the same TCP connection. Instead, the server can pick up at the rate where the first object left off. We shall quantitatively compare the performance of non-persistent and persistent connections in the homework problems of Chapter 3. The interested reader is also encouraged to see [\[Heidemann 1997\]](#) and [\[Nielsen 1997\]](#).

2.2.3 HTTP Message Format

The HTTP specifications [RFC 2616] define the HTTP message formats. There are two types of HTTP messages, request messages and response messages, both of which are discussed below.

HTTP Request Message

Below we provide a typical HTTP request message:

```
GET /somedir/page.html HTTP/1.1
Connection: close
User-agent: Mozilla/4.0
Accept: text/html, image/gif, image/jpeg
Accept-language: fr
```

We can learn a lot by taking a close look at this simple request message. First of all, we see that the message is written in ordinary ASCII text, so that your ordinary computer-literate human being can read it. Second, we see that the message consists of five lines, each followed by a carriage return and a line feed. The last line is followed by an additional carriage return and line feed. Although this particular request message has five lines, a request message can have many more lines or as little as one line. The first line of a HTTP request message is called the **request line**; the subsequent lines are called the **header lines**. The request line has three fields: the method field, the URL field, and the HTTP version field. The method field can take on several different values, including GET, POST, and HEAD. The great majority of HTTP request messages use the GET method. The GET method is used when the browser requests an object, with the requested object identified in the URL field. In this example, the browser is requesting the object /somedir/page.html. (The browser doesn't have to specify the host name in the URL field since the TCP connection is already connected to the host (server) that serves the requested file.) The version is self-explanatory; in this example, the browser implements version HTTP/1.1.

Now let's look at the header lines in the example. By including the `Connection:close` header line, the browser is telling the server that it doesn't want to use persistent connections; it wants the server to close the connection after sending the requested object. Thus the browser that generated this request message implements HTTP/1.1 but it doesn't want to bother with persistent connections. The `User-agent:` header line specifies the user agent, i.e., the browser type that is making the request to the server. Here the user agent is `Mozilla/4.0`, a Netscape browser. This header line is useful because the server can actually send different versions of the same object to different types of user agents. (Each of the versions is addressed by the same URL.) The `Accept:` header line tells the server the type of objects the browser is prepared to accept. In this case, the client is prepared to accept HTML text, a GIF image or a JPEG image. If the file `/somedir/page.html` contains a Java applet (and who says it can't!), then the server shouldn't send the file, since the browser can not handle that object type. Finally, the `Accept-language:` header indicates that the user prefers to receive a French version of the object, if such an object exists on the server; otherwise, the server should send its default version.

Having looked at an example, let us now look at the general format for a request message, as shown in Figure 2.2-2:

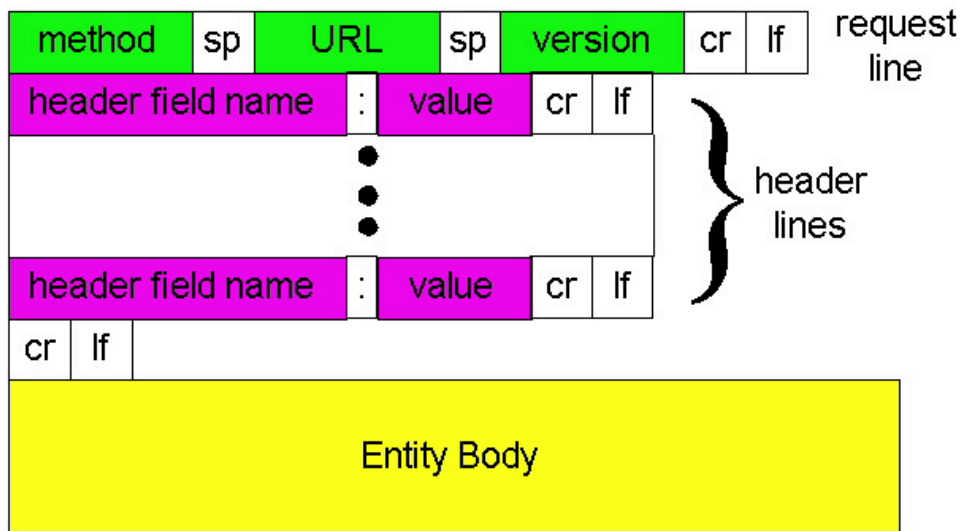


Figure 2.2-2: general format of a request message

We see that the general format follows closely the example request message above. You may have noticed, however, that after the header lines (and the additional carriage return and line feed) there is an "Entity Body". The Entity Body is not used with the GET method, but is used with the POST method. The HTTP client uses the POST method when the user fills out a form -- for example, when a user gives search words to a search engine such as [Yahoo](#). With a POST message, the user is still requesting a Web page from the server, but the specific contents of the Web page depend on what the user wrote in the form fields. If the value of the method field is POST, then the entity body contains what the user typed into the form fields. The HEAD method is similar to the POST method. When a server receives a request with the HEAD method, it responds with an HTTP message but it leaves out the requested object. The HEAD method is often used by HTTP server developers for debugging.

HTTP Response Message

Below we provide a typical HTTP response message. This response message could be the response to the example request message just discussed.

```
HTTP/1.1 200 OK
Connection: close
Date: Thu, 06 Aug 1998 12:00:15 GMT
Server: Apache/1.3.0 (Unix)
Last-Modified: Mon, 22 Jun 1998 09:23:24 GMT
Content-Length: 6821
Content-Type: text/html
```

data data data data data ...

Let's take a careful look at this response message. It has three sections: an initial **status line**, six **header lines**, and then the **entity body**. The entity body is the meat of the message -- it contains the requested object itself (represented by *data data data data data ...*). The status line has three fields: the protocol version field, a status code, and a corresponding status message. In this example, the status line indicates that the server is using HTTP/1.1 and that that everything is OK (i.e., the server has found, and is sending, the requested object).

Now let's look at the header lines. The server uses the `Connection: close` header line to tell the client that it is going to close the TCP connection after sending the message. The `Date:` header line indicates the time and date when the HTTP response was created and sent by the server. Note that this is not the time when the object was created or last modified; it is the time when the server retrieves the object from its file system, inserts the object into the response message and sends the response message. The `Server:` header line indicates that the message was generated by an Apache Web server; it is analogous to the `User-agent:` header line in the HTTP request message. The `Last-Modified:` header line indicates the time and date when the object was created or last modified. The `Last-Modified:` header, which we cover in more detail below, is critical for object caching, both in the local client and in network cache (a.k.a. proxy) servers. The `Content-Length:` header line indicates the number of bytes in the object being sent. The `Content-Type:` header line indicates that the object in the entity body is HTML text. (The object type is officially indicated by the `Content-Type:` header and not by the file extension.)

Note that if the server receives an HTTP/1.0 request, it will not use persistent connections, even if it is an HTTP/1.1 server. Instead the HTTP/1.1 server will close the TCP connection after sending the object. This is necessary because an HTTP/1.0 client expects the server to close the connection.

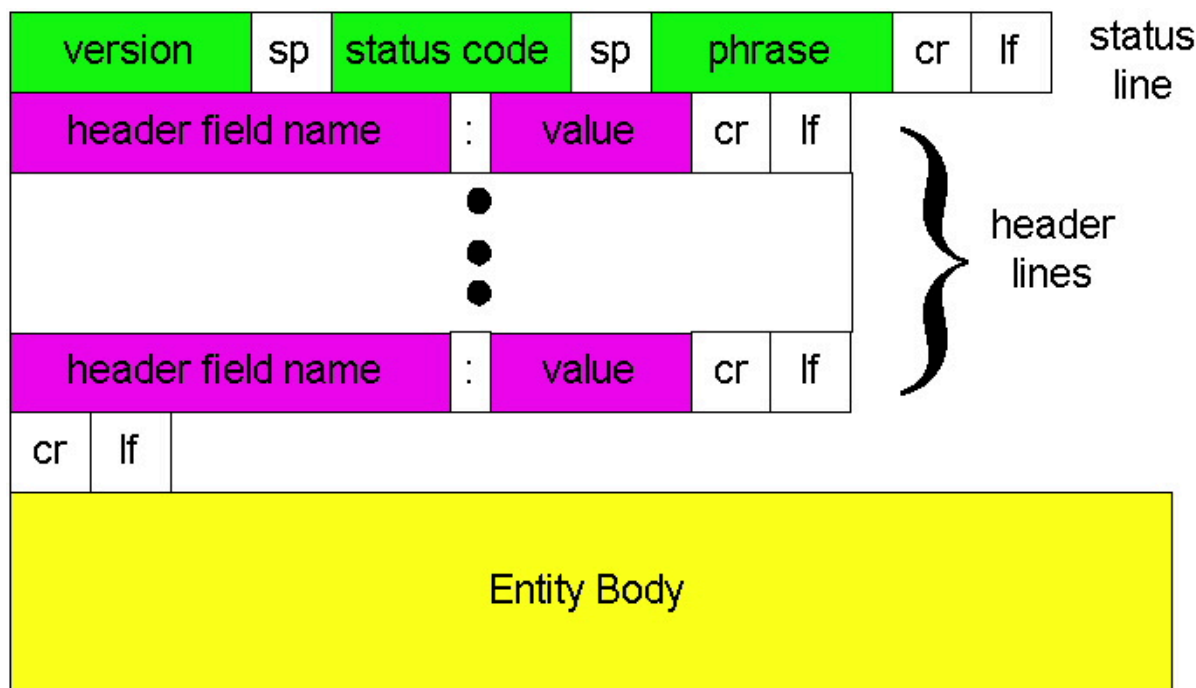


Figure 2.2-3: General format of a response message

Having looked at an example, let us now examine the general format of a response message, which is shown in Figure 2.2-3. This general format of the response message matches the previous example of a response message. Let's say a few additional words about status codes and their phrases. The status code and associated phrase indicate the result of the request. Some common status codes and associated phrases include:

- 200 OK: Request succeeded and the information is returned in the response.

- 301 Moved Permanently: Requested object has been permanently moved; new URL is specified in Location: header of the response message. The client software will automatically retrieve the new URL.
- 400 Bad Request: A generic error code indicating that the request could not be understood by the server.
- 404 Not Found: The requested document does not exist on this server
- 505 HTTP Version Not Supported: The request HTTP protocol version is not supported by the server.

How would you like to see a real HTTP response message? This is very easy to do! First Telnet into your favorite WWW server. Then type in a one-line request message for some object that is housed on the server. For example, if you can logon to a Unix machine, type:

```
telnet www.eurecom.fr 80
GET /~ross/index.html HTTP/1.0
```

(Hit the carriage return twice after typing the second line.) This opens a TCP connection to port 80 of the host `www.eurecom.fr` and then sends the HTTP GET command. You should see a response message that includes the base HTML file of Professor Ross's homepage. If you'd rather just see the HTTP message lines and not receive the object itself, replace GET with HEAD. Finally, replace `/~ross/index.html` with `/~ross/banana.html` and see what kind of response message you get.

In this section we discussed a number of header lines that can be used within HTTP request and response messages. The HTTP specification (especially HTTP/1.1) defines many, many more header lines that can be inserted by browsers, Web servers and network cache servers. We have only covered a small fraction of the totality of header lines. We will cover a few more below and another small fraction when we discuss network Web caching at the end of this chapter. A readable and comprehensive discussion of HTTP headers and status codes is given in [Luotonen 1998]. An excellent introduction to the technical issues surrounding the Web is [Yeager 1996].

How does a browser decide which header lines it includes in a request message? How does a Web server decide which header lines it includes in a response messages? A browser will generate header lines as a function of the browser type and version (e.g., an HTTP/1.0 browser will not generate any 1.1 header lines), user configuration of browser (e.g., preferred language) and whether the browser currently has a cached, but possibly out-of-date, version of the object. Web servers behave similarly: there are different products, versions, and configurations, all of which influence which header lines are included in response messages.

References

Some of the best information about HTTP can be found in the [W3C pages](#). Their [overview page](#) is an excellent starting point for a wealth of information about the HTTP activities at the W3C. You will also find material on [HTTP-Next Generation](#) and [Web caching](#). If you are interested in HTTP, the [W3C site](#) will keep you busy for a long, long time.

[Claffy 1998] K. Claffy, G. Miller and K. Thompson, "The Nature of the Beast: Recent Traffic Measurements from the Internet Backbone, CAIDA Web site, <http://www.caida.org/Papers/Inet98/index.html>, 1998.

[Heidemann 1997] J. Heidemann, K. Obraczka and J. Touch, Modeling the Performance of HTTP Over Several Transport Protocols," IEEE/ACM Transactions on Networking, Vol. 5, No. 5, October 1997, pp. 616-630.

[Huffaker 1998] B. Huffaker, J. Jung, D. Wessels and K. Claffy, Visualization of the Growth and Topology of the NLANR Caching Hierarchy, <http://squid.nlanr.net/Squid/> <http://www.caida.org/Tools/Plankton/Paper/plankton.html> , 1998.

[Luotonen 1998] A. Luotonen, "Web Proxy Servers," Prentice Hall, New Jersey, 1998.

[Netcraft] Survey of Web Server Penetration, Netcraft Web Site, <http://www.netcraft.com/Survey/>

[NLANR] A Distributed Testbed for National Information Provisioning, <http://ircache.nlanr.net> .

[Nielsen 1997] H. F. Nielsen, J. Gettys, A. Baird-Smith, E. Prud'hommeaux, H.W. Lie, C. Lilley, [Network Performance Effects of HTTP/1.1, CSS1, and PNG, W3C Document](#), 1997 (also appeared in SIGCOMM' 97).

[RFC 1945] T. Berners-Lee, R. Fielding, and H. Frystyk, "Hypertext Transfer Protocol -- HTTP/1.0," [RFC 1945], May 1996.

[RFC 2068] R. Fielding, J. Gettys, J. Mogul, H. Frystyk, and T. Berners-Lee, "Hypertext Transfer Protocol -- HTTP/1.1," [RFC 2068], January 1997

[RFC 2109] D. Kristol and L. Montulli, "HTTP State Management Mechanism," [RFC 2109], February 1997.

[RFC 2186] K. Claffy and D. Wessels, "Internet Caching Protocol (ICP), version 2," [\[RFC 2186\]](#), September 1997.

[Ross 1997] K.W. Ross, "Hash-Routing for Collections of Shared Web Caches," IEEE Network Magazine, Nov-Dec 1997

[Ross 1998] K.W. Ross, Distribution of Stored Information in the Web, A Online Tutorial, <http://www.eurecom.fr/~ross/CacheTutorial/DistTutorial.html>, 1998.

[Squid] Squid Internet Object Cache, <http://squid.nlanr.net/Squid/>

[Valloppillil 1997] V. Valloppillil and K.W. Ross, "Cache Array Routing Protocol," Internet Draft, [<draft-vinod-carp-v1-03.txt>](#), June 1997.

[Yeager 1996] N.J. Yeager and R.E. McGrath, "Web Server Technology," Morgan Kaufmann Publishers, San Francisco, 1996.

Search RFCs and Internet Drafts

If you are interested in an Internet Draft relating to a certain subject or protocol enter the keyword(s) here.

Query:

Press button to submit your query or reset the form:

Query Options:

☒ Case insensitive

Maximum number of hits:

[Return to Table Of Contents](#)

Copyright Keith W. Ross and James F. Kurose 1996-2000 . All rights reserved.