the World Wide Web

9.1 Introduction

The World Wide Web – normally abbreviated to "the Web" or sometimes "the Net" – is a vast collection of electronic documents each composed of a linked set of pages written in HTML. The documents are stored in files on many thousands of computers that are distributed around the global Internet. The concepts behind the Web were conceived in 1989 by Tim Berners-Lee when he was working at the European Particle Physics Laboratory, CERN. An agreement was signed in 1994 between CERN and the Massachusetts Institute of Technology, MIT, to set up a consortium whose aim was to further develop the Web and to standardize the protocols associated with it. The National Center for Supercomputing Applications (NCSA) also made a major contribution to the current widespread use of the Web with the development of MOSAIC, which was the first interactive Web browser based on a graphical user interface. Since that time, many related developments have taken place and, in terms of volume, the Web is now the largest source of data transferred over the Internet.

In this chapter we first present an overview of the operation of the Web and the essential protocols and standards associated with it. We then expand upon these descriptions as we discuss the following:

- **URLs and HTTP**: a URL comprises the name of the file and the location of the server on the Internet where the file is stored while HTTP is the protocol used by a browser program to communicate with a server program over the Internet;
- **HTML**: this is used to define how the contents of each Web page are displayed on the screen of the user's machine – a PC, workstation or set-top box – and to set up the hyperlinks with other pages;
- **forms and CGI script**: these are used in e-commerce applications. Fill-in forms are integrated into a Web page and displayed on the screen of the browser machine to get input from the user and a CGI script is then used at the server to process this information;
- **helper applications and plug-ins**: these are used to process and output multimedia information such as audio and/or video that is incorporated into an HTML page;
- **Java applets**: these are separate programs that are called from an HTML page and downloaded from a Web server. They are then run on the browser machine. Typically, they are used for code that may change or to introduce interactivity to a Web page such as for games playing;
- **JavaScript**: this is also used to add interactivity to a Web page but in this case the code is not a separate program but is included in the page"s HTML code;
- **security** in e-commerce applications;
- **the operation of the Web** including the role of search engines and portals.

In relation to HTML and Java/JavaScript, since there are now many books on each of these topics, the aim here is to give sufficient detail for you to build up a working knowledge of them. Further details can then be found in the bibliography for this chapter.

9.2 Overview

As we have just indicated, in the context of multimedia communications, most interactive applications over the Internet are concerned with interactions with a World Wide Web server. Hence in this section we shall identify a selection of the standards that have been defined for use with this type of interactive application. We shall identify and explain the role of these standards by considering various application scenarios.

9.2.1 Information browsing

The most basic type of interaction using the Web is for information browsing since with this, the Web user wishes only to browse through information that

has been made available on a particular Web server at a site. Typically, the information comprises an integrated set of one or more Web pages. Each page in the set contains linkages to other pages, which can be located either on the same server or on any other server that is connected to the Internet. Typically, the Web pages are written in the HyperText Markup Language (HTML) and contain all the information necessary both to display the contents of a page – text, images, and so on – on the screen of the user/client machine and also to locate the other pages that have linkages with the page. The general arrangement used for information browsing is as shown in Figure 9.1.

A page is accessed and its contents displayed by means of a program known as a **browser** that runs on the user/client machine. The browser locates and fetches each requested page and, by interpreting the formatting commands that the page contains, the page contents are displayed. In addition, by the user clicking the mouse on a linkage point within the displayed page, the page that is linked to that point is accessed by the browser and displayed in the same way. There are a number of browser programs available, some popular examples being Netscape Navigator, NSCA Mosaic and Microsoft Internet Explorer.

A page can contain two types of text – plaintext and underlined text (hypertext) – tables, images, and sometimes other media such as a sound track or a video clip. In the case of underlined text, in addition to the text, this contains all the information that is necessary for the browser to access the contents of the linked page. This is known as a **hyperlink** and the linkage comprises the name of the application protocol (also known as the **scheme**)

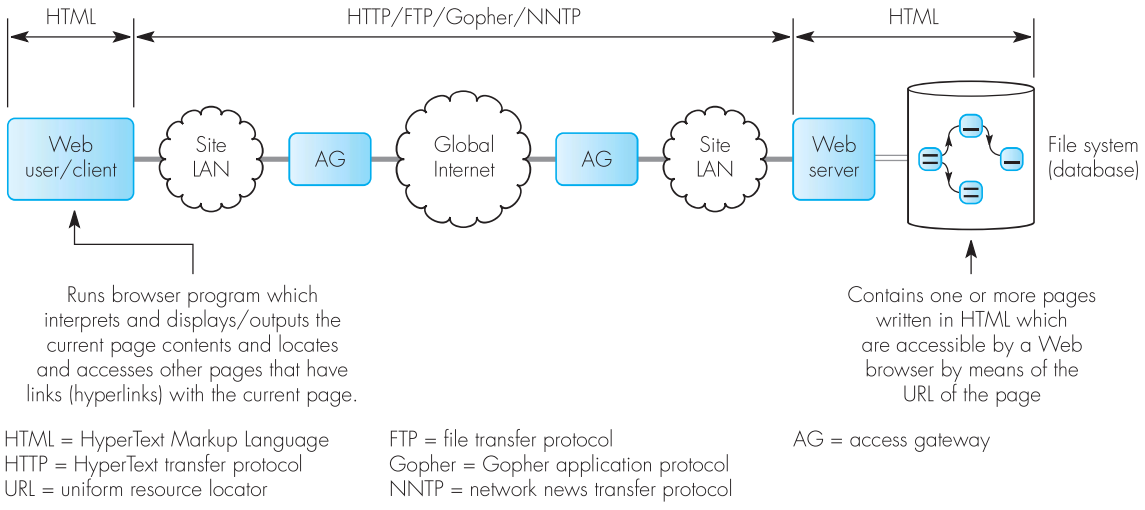


Figure 9.1 Information browsing.

that is to be used – normally the **hypertext transfer protocol** (**HTTP**) – and the symbolic Internet name of the server machine (also known as the **domain**) in which the page is stored. Normally, this contains the top-level page for the site, which in turn contains hyperlinks to all the next-level pages. Alternatively, if a specific page is required, it is also possible to specify the (local) directory and name of the file that contains the required page. Collectively, these fields form what is called the **uniform resource locator** (**URL**) for the page. Two examples are:

http://www.microsoft.com
http://www.mpeg.org/index.html

Note that all the characters can be either upper or lower case.

In the case of images (and other media types), the media type is in the form of a **tag** (**IMG**) with a parameter that indicates the name of the field where the image is stored. These are written in the page text at the point where the image is to be displayed and, when the browser interprets the tag, it reads the (compressed) image from the file. The file name also includes the image format and, by using a corresponding decompression algorithm, the browser displays the (decompressed) image at the appropriate point on the screen. In the case of audio and video, these are output either in a similar way (if the browser contains the appropriate decompression code) or the contents of the file containing the audio/video are passed by the browser to a separate program for output known as a **helper application** or **external viewer**.

Each page is accessed and transferred over a TCP connection using the HTTP application-level protocol. Each HTTP interaction comprises a request from the browser written in the form of an ASCII string and a response from the server written in the RFC 822 format with MIME extensions, both of which we described earlier in Section 8.3.2. In order to allow for the possibility that the linked set of pages may be distributed over a number of different servers, a separate TCP connection is established between the client and server for each interaction and, once the response has been received, the TCP connection is cleared. HTTP is known, therefore, as a **stateless protocol** and we shall describe it in more detail in Section 9.3.2.

In addition to using HTTP to access pages written in HTML, a browser can also access other information using a number of the older application protocols. These allow access to the contents of:

- a file on the client machine on which the browser is running,
- a remote file using the **file transfer protocol** (**FTP**),
- a Gopher (text-only) file using the **gopher protocol**,
- a news article from a UseNet server using the **network news transfer protocol** (**NNTP**).

A summary of the protocols that we have identified is presented in Figure 9.2. In the figure it is assumed that all pages are written in HTML and that

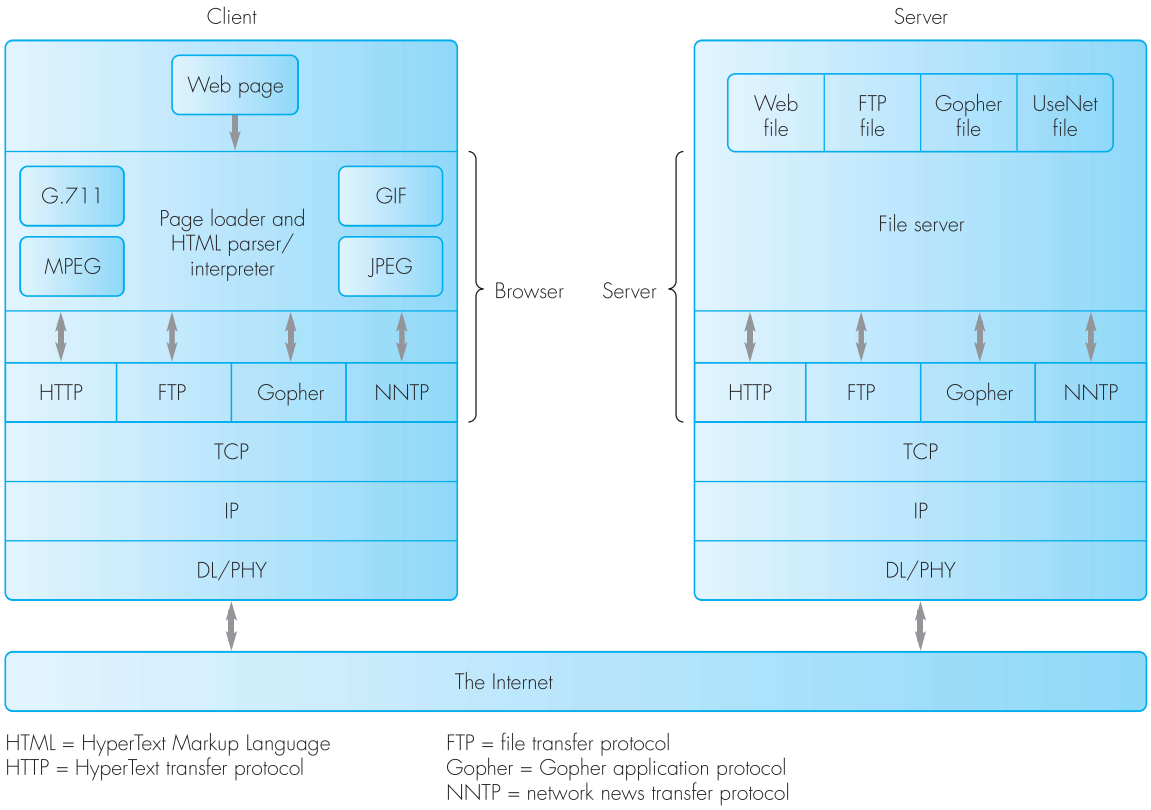


Figure 9.2 Protocol stack to support information browsing.

the browser, in addition to HTTP, supports all the other application protocols we have just listed. Also, that it has various support facilities to decompress the contents of image, audio and video files that may be included in a page.

9.2.2 Electronic commerce

When browsing the Web for information, the flow of information is unidirectional from the server to the client machine. However, some applications also involve the transfer of information in the reverse direction from the client to a server; for example, after browsing the information at a site, to send details of your credit card in order to purchase, say, a book or theater ticket. This is just one example of what is known more generally as **electronic commerce** or **e-commerce** and there is a range of standards associated with this type of application.

In order to meet this requirement, it is possible to include what is known as a **form** into an HTML page. In the same way that a printed order form

contains blank spaces for you to enter your name and other information and to make selections, so a typical HTML form is written to have a similar appearance. The user then uses the mouse and keyboard to enter the requested information and, when all the information has been entered and the appropriate selections made, typically, the user clicks on a symbolic **submit** button to initiate the sending of the entered information back to the server machine.

In addition to having a standardized way for a user of a client machine to enter and initiate the sending of information (forms/submit), there is also a standard for use at the server for processing the received information. This is known as the **common gateway interface** (**CGI**) and, in addition to accepting and processing the input from forms, the CGI may also initiate the output of other (unsolicited) pages that contain related information. The general arrangement that is used to support e-commerce is shown in Figure 9.3 and we shall present further details of forms and CGI in Section 9.4.6.

As we shall see, a second function associated with CGI is that of **network security**. Clearly, when information such as credit card details is sent over a network, it is essential that it is received only by the intended recipient. Hence there are standards for achieving this and, as we shall expand upon in Section 10.8, they are based on either a **private** or **public key encryption** scheme. At the application level, it is also necessary to authenticate that a particular transaction was initiated by the owner of the credit card and not an impostor. Again there are associated standards that we shall also discuss in Section 10.8.

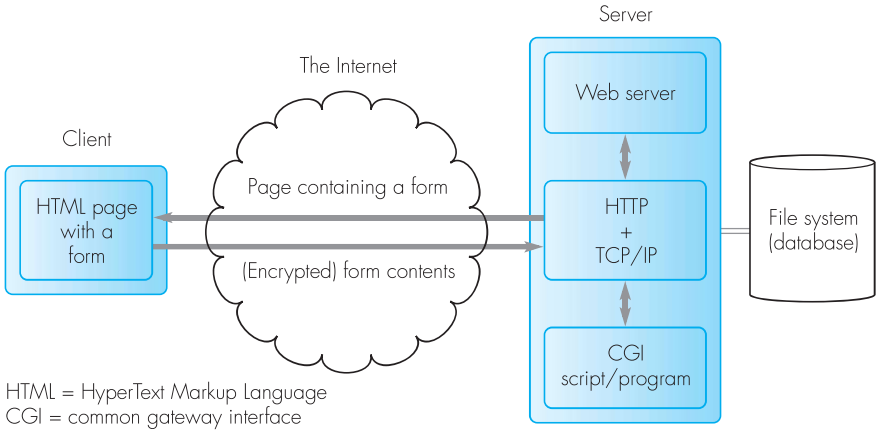


Figure 9.3 Electronic commerce.

9.2.3 Intermediate systems

The discussion in the previous two sections assumed that both the client and server machines were connected directly to the Internet. In some instances, however, this is not the case and communication between the client and server is achieved through a networking device known as an **intermediate system**.

For example, as we explained in Section 3.7, many enterprise networks now use the same set of protocols as are used with the Internet, the enterprise network then being known as an intranet. This is done both to simplify access to the Web from the various sites that make up the enterprise network and also to enable (external) Web users to access information that is stored on a server connected to the enterprise network. Normally, however, for security reasons access to a server that is connected to an intranet is not provided directly but rather through an intermediate system known as a **firewall** or **security gateway**.

As we show in Figure 9.4(a), the gateway controls the flow of information both to and from the intranet. To do this, the gateway intercepts all incoming requests from the Internet for access to the enterprise server and also all responses from the server that need to be forwarded over the Internet. Each Internet packet contains the IP address of both the source of the packet and the intended recipient/destination. Hence in the most basic type of gateway, the gateway simply maintains a separate list of all source and destination IP addresses that are allowed to pass both into and out from the intranet and any packets that have addresses different from these are discarded. This approach is known as **packet filtering** and is often used when the intranet itself comprises a large number of interconnected sites. In practice, however, it is not too difficult for a hacker to break this system and hence an alternative approach that performs the filtering operation at the application layer rather than the IP layer is also used. With this approach, the gateway behaves like the enterprise server to all incoming requests and only if the gateway is satisfied that a request is from a legitimate user is it relayed to the real server. Similarly, all responses from the real server are sent via the gateway. The same controls are applied to internal requests from a client connected to the intranet for an external server.

A second type of intermediate system is required when a browser supports only the HTTP application protocol. To access information that requires a different application protocol from HTTP it is necessary to use what is known as a **proxy server**. As we show in Figure 9.4(b), all requests for information are passed to the proxy server using the HTTP, but a proxy server can also communicate with other servers using application protocols such as FTP, NNTP and Gopher. Hence if the information requested requires a different application protocol from HTTP, the proxy server makes the request on behalf of the client using the appropriate protocol. Similarly, on receipt of the requested information, this is passed to the client using HTTP. As we show in the figure, a proxy server can support a number of clients and,

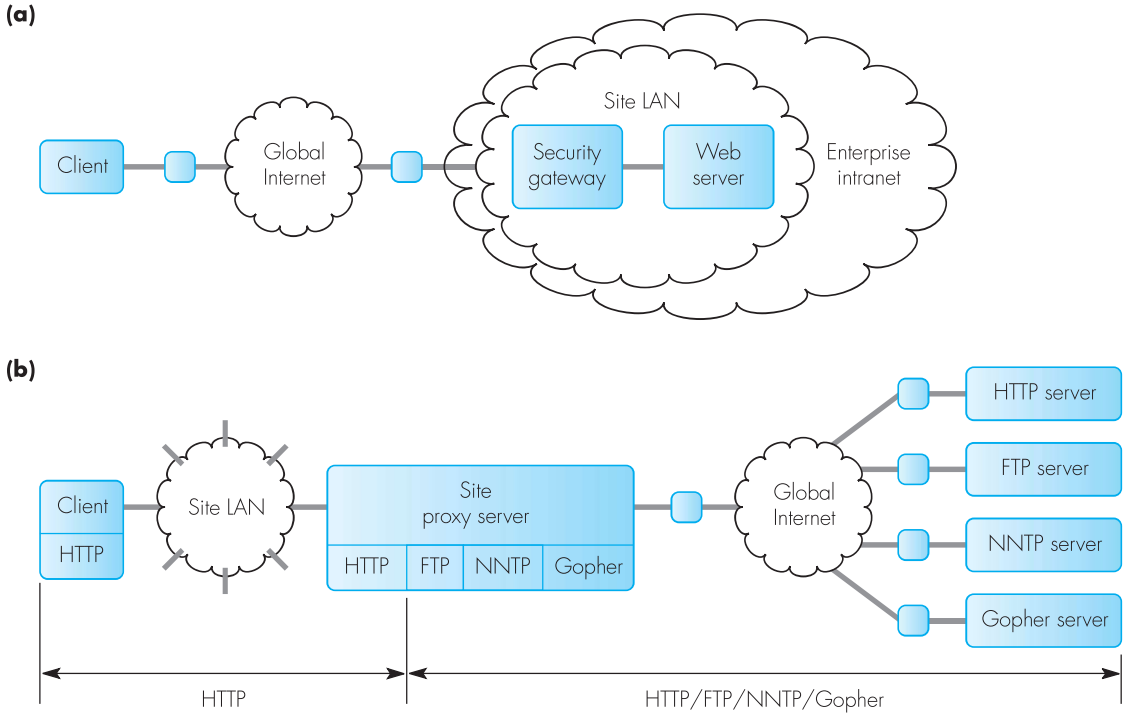


Figure 9.4 Intermediate systems: (a) security gateway; (b) proxy server.

in some instances, performs other functions such as those associated with a security gateway.

9.2.4 Java and JavaScript

A disadvantage of using only HTML to write Web pages is that it is then relatively difficult to incorporate new features into pages – such as a new decoder – since this would necessitate modifications to the browser code. To overcome this constraint, it is possible to implement portions of the code for a page as self-contained subprograms – known as **applets** – which are independent of the HTML interpreter. Then, in the same way that an image is accessed and displayed when its tag is interpreted (by the HTML interpreter), so an applet is identified by a tag and, when this is interpreted, the applet code is loaded and run.

The advantage of using applets is that since each applet is a self-contained program, by implementing those parts of a page that contain code that is likely to change in the form of applets (for example media decoders), then any changes that do occur can be incorporated into the server rather than the browser. For example, if the browser contained only a particular type of image

decoder and pages became available that contained images that were encoded using a different coder, then without applets the browser code would need to be modified. By using applets, however, the new decoder could be written as an applet – located either on the same server as the current page or on a different server – and, by simply specifying the applet with an applet tag within the page, so the applet for the new decoder would be loaded and run without any modifications to the browser itself. It is also possible to include applets for sound and video. The sound/video can then be output either when the applet is loaded or under control of the user at the click of the mouse.

An example of a programming language that is used to produce applets that are downloaded from a server is **Java**. This is based on C++ but, in order to obtain portability, there are no input/output statements associated with Java. A program written in Java is compiled to run on any machine. The compiled program is known as an applet and, in order for the applet to be run on a variety of different types of (client) machine, the applet code produced by the compiler is for what is called a **virtual machine**. The compiled/applet code is known as **bytecode** and, to run the applet, the browser, in addition to an HTML interpreter, must also contain an interpreter of the Java bytecode. The general scheme is shown in Figure 9.5.

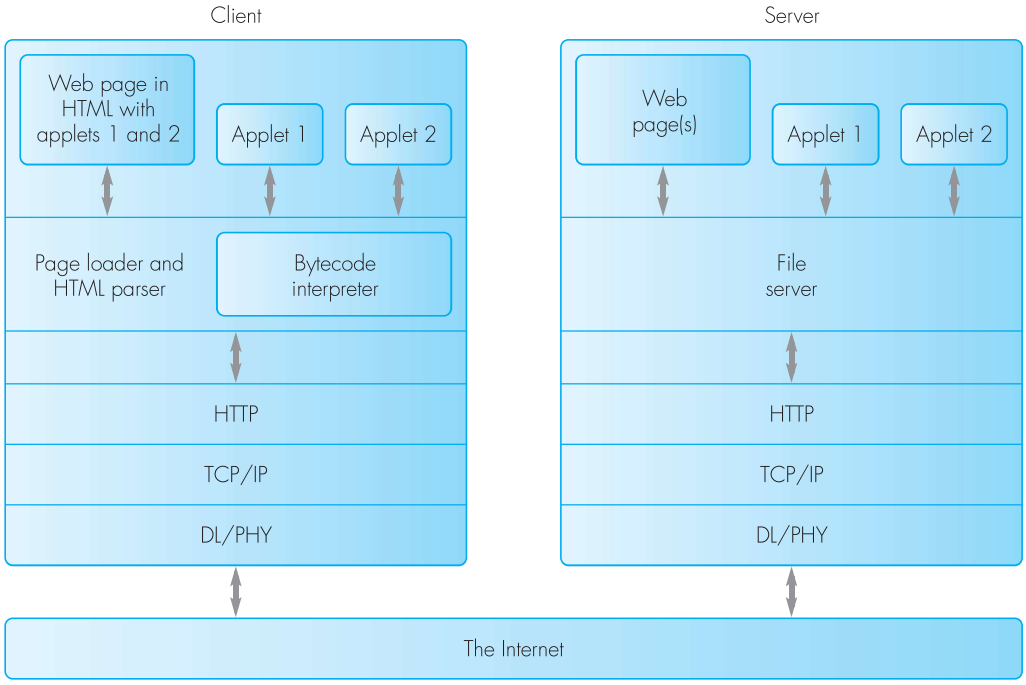


Figure 9.5 Protocol stack to support the browsing of pages containing Java applets.

In addition to downloading applets into pages written in HTML, it is also possible to implement a browser simply as a collection of applets. In this case, at startup, the browser comprises just a Java loader/interpreter and everything else is then implemented in the form of applets that are loaded on demand. Hence to access a page written in HTML, the HTML loader/interpreter would be loaded first and, if the accessed page contains an image, then the corresponding image decoder would be loaded and so on. Again, the advantage of this approach is that new versions of the various software components can be introduced more readily.

It is also possible to embed Java code into an HTML page directly. The language used to do this is called **JavaScript**. We shall return to the subject of Java and JavaScript in Section 9.5.

9.3 URLs and HTTP

The Web is made up of a vast collection of documents/pages that are stored in files located on many thousands of (server) computers distributed around the global Internet. As we saw in Section 9.2.1, using HTML it is possible to create on a server an electronic document in the form of a number of pages with defined linkages between them. A user then gains access to a specific page using a client program called a browser which runs on a multimedia PC/workstation/set-top box that has access to the Internet.

Associated with each access request is the uniform resource locator (URL) of the requested file/page. This comprises the domain name of the server computer on which the file/page is stored and the file name. To obtain a page the browser communicates with a peer application process in the named Web server computer using the HyperText transfer protocol (HTTP). The contents of the named file are then transferred to the browser and displayed on the screen according to the HTML markup descriptions the page contains. A schematic diagram showing this overall mode of operation is given in Figure 9.6. In this section we describe first the structure of URLs and then the operation of HTTP. We defer how a URL is embedded into an HTML page until section 9.4 when we describe HTML in more detail.

9.3.1 URLs

The standard format of the URL of an HTML page consists of:

- the application protocol to be used to obtain the page,
- the domain name of the server computer,
- the pathname of the file,
- the file name.

Thus an example URL referring to an HTML page on the Web is:

http://www.mpeg.org/mpeg-4/index.html

where *http* is the protocol used to obtain the Web page, *www.mpeg.org* is the domain name of the server, */mpeg-4* is the path name, and *index.html* is the file name.

Normally, a browsing session starts by a user entering the URL of the home page associated with a particular document in the *location* field provided by the browser. If the URL of the page is not known then it can be obtained either from the user's own local Web/Net directory – which is built up from previously given or previously used URLs – or by using the search facility supported by the browser. We shall expand upon this feature later in Section 9.7. Once a URL has been entered, the browser proceeds to access the (home) page from the server named in the URL using the specified protocol and the given file name.

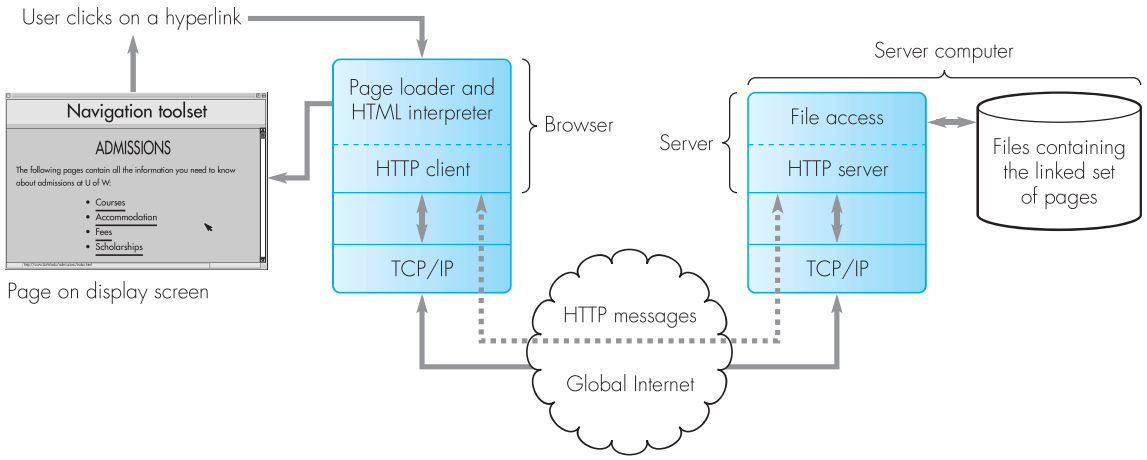
Note that when we access most home pages it is not necessary to specify the full URL since the server will look for the file named *index.html* if one is not specified. For example, the home page associated with the above URL could be specified as:

http://www.mpeg.org

Note also that a final forward slash is used to indicate the URL relates to a directory rather than a file name. For example:

http://www.mpeg.org/mpeg-4/

In addition to obtaining a page/file using HTTP, most browsers allow a user to obtain a file using a range of other application protocols. In general,



Note: Hyperlinks contain a URL which includes the domain name of the server computer and the name of the file containing the HTML code of the selected page

Figure 9.6 Basic principles and terminology associated with the World Wide Web.

these are standard Internet application protocols that predate the Web. For example, as we saw in Section 8.4, FTP is the standard Internet application protocol used to transfer a file. As a result, many servers still use FTP for all file transfers. Hence to obtain the contents of a file from such a server it is possible to specify *ftp:* as the protocol in a URL instead of *http:*. An example of a typical URL is then of the form:

ftp://yourcompany.com/pub/

Typically, this file will contain the list of publications/files – note that *pub/* indicates a directory – that are available from the file server *yourcompany.com*. Normally, as we saw in Section 8.4.6, a user logs on to most public domain FTP servers using *anonymous* for the user name and his or her e-mail address for the password. Hence these are entered when requested by the browser. The browser then obtains the file using the FTP protocol and displays the contents on the browser screen.

A protocol name of *file*: is used to indicate the file is located on the same computer as the browser; that is, your own computer. This is a useful facility when developing a Web page since it allows you to view the contents of the page before you make it available on the Web. An example URL is:

file://hypertext/html/mypage.htm

Note that since some older versions of DOS allow only three characters in a file name extension, the final *l* of *html* is sometimes missing.

The *news:* protocol relates to an Internet application protocol defined in RFC 977 called the **network news transfer protocol** (**NNTP**). It is used to transfer the text-only messages associated with **UseNet** which is also called **NetNews**. This consists of a world-wide collection of **newsgroups** each of which is a discussion forum on a specific topic. Two examples are COMP, which has topics relating to computers, computer science and the computer industry, and SCI, which has topics relating to the physical sciences and engineering. A person interested in a particular topic can subscribe to be a member of the related newsgroup. A subscriber can then post (send) an article to all the other members of the same newsgroup and receive the articles that are written by all the other members of the same group.

To do this, a user agent similar to that used with SMTP is used. This is called a **news reader** and the protocol that is used to transfer the messages associated with UseNet is NNTP. Some examples of the request/command messages associated with NNTP are:

- LIST: this is used to obtain a list of all the current newsgroups and their articles;
- GROUP *grp*: this is used to obtain a list of the articles associated with the newsgroup *grp*;
- ARTICLE *id*: this is used to obtain article *id*;

- POST*id*: this is used to send article *id* to the members of a specified newsgroup.

The *news*: protocol enables a member to both read a news article and to post an article from within a Web page. An example of a URL relating to a newsgroup involved in the preparation of HTML documents is:

news:comp.infosystems.www.authoring.html

Note that in this case the two forward slashes following the colon are not required. When this URL is entered, typically, the news reader part of the browser responds by first obtaining the list of articles on this topic using the GROUP command and the NNTP protocol. It then displays the articles on the screen in the form of a scrollable list. The user can then click on a specific article in the list and the browser/news reader will obtain the article contents using the ARTICLE command and display this on the screen. Normally, each article has the e-mail address of the author, his or her affiliation, and the date the article was posted. A similar procedure is followed to post an article using the POST command. Normally, the user is prompted by the news reader for, say, the name of the (local) file containing the article. The file contents are then sent using NNTP.

The *gopher*: protocol relates to an Internet application protocol called Gopher. The Gopher system is similar in principle to the Web inasmuch as it is a global delivery and retrieval system of documents. In Gopher, however, all items of information are text only. When a user logs on to a Gopher server a hierarchical menu of files and directories is presented each of which may have links to the menus on other servers. A user can then access the contents of a file or directory by clicking on it. The *gopher*: protocol enables the user of the browser to do this within a Web page.

The *mailto*: protocol is provided to enable a user to send an e-mail from within a Web page. Typically, this facility is initiated by the user entering a URL with *mailto*: in the protocol part. This is followed by the e-mail address of the intended recipient. Normally, the (e-mail) user agent part of the browser responds by displaying a **form** containing the other (e-mail) header fields at the top (to be filled in) and space for the actual message. A facility is then provided to initiate the sending of the mail, which, typically, is sent using either SMTP or, if the e-mail server supports it, HTTP. An example URL is:

mailto:yourname@youruniversity.edu

Note that the two forward slashes following the colon are not required with this protocol.

Universal resource identifiers (URIs)

A limitation of a URL is that it specifies a single host name. In many instances, however, a Web site may have so many access requests/hits for a

particular document/page that copies of the document must be placed on multiple hosts. Also, to reduce the level of Internet traffic, each of these hosts may be geographically distributed around the Internet. To enable this to be implemented, an alternative page identifier called a URI can be specified with some browsers. Essentially, this is a generic URL since, typically, it contains only the file name. The remaining parts of the URL are determined by the context in which the URI is given and these are filled in by the browser itself. We shall give an example of this in the next section when we discuss cache servers.

9.3.2 HTTP

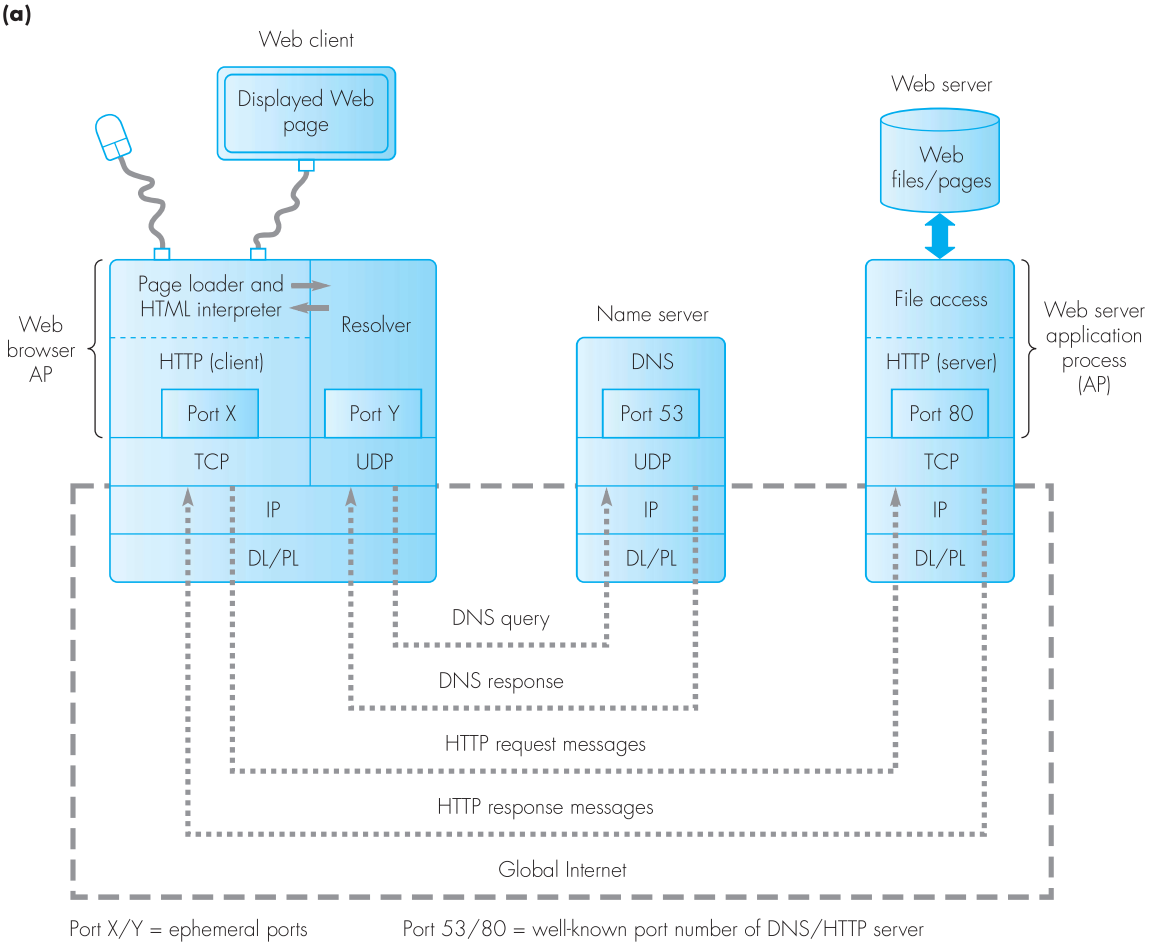
HTTP is the standard application protocol – also known as a method – that is used to obtain a Web page and also other items of information relating to a page such as an image or a segment of audio or video. The earlier version of HTTP is defined in RFC 1945 and a later version (1.1) in RFC 2068. It is a simple request-response protocol: the browser side sends a request message and the server side returns a response message. Figure 9.7(a) shows the protocol stack associated with Web browsing, and a selection of the *commands/methods* associated with request messages, together with their use, are shown in part (b) of the figure.

In general, these are self explanatory. Note, however, that the GET method is used also to, say, obtain an image or segment of audio – shown as a block of data – from a named file. We shall expand upon this in later sections. Also, as we shall see in Section 9.4.6, the POST method is used to send the information entered by a user in e-commerce applications.

The well-known port number of HTTP is port 80. The Web server application process (AP) at each Web site continuously listens to this port for an incoming TCP connection request (SYN) from a Web browser. Note that in a Unix machine the AP is referred to as a **daemon** and the HTTP in these machines is sometimes called **HTTPD**. When a browser has an HTTP request message to send, it initiates the establishment of a new TCP connection to port 80 on the named server in the URL. It then initiates the sending of the request message over the established connection and, with earlier versions of HTTP, after the related response message has been received correctly by the browser, the server initiates the release of the connection. The TCP connection associated with this mode of operation is called **nonpersistent**.

As we can deduce from our earlier discussion of TCP in Section 7.3.2, the use of a new TCP connection for each request/response message transfer has a number of disadvantages. Firstly, when accessing a Web page that contains multiple entities within it – an image for example – a time delay is incurred for each entity that is transferred while a new TCP connection is established. This is a function of the network round-trip time (RTT), which can be significant. Secondly, each new transfer starts with the slow start procedure and hence for a large entity, this can lead to additional delays.

(a)



(b)

HTTP	Use
GET <file name>	Read a Web page/block of data from the named file
HEAD <file name>	Read the header only of the specified Web page
PUT <file name>	Write a Web page or block of data to the named file
POST <file name>	Append a Web page/block of data to that in the named file
DELETE <file name>	Delete the named file

Figure 9.7 HTTP principles: (a) protocol stack; (b) a selection of the requests/methods supported.

To reduce the effect of these delays, when multiple entities are specified within a page – and hence to be transferred – many browsers set up a number of TCP connections so that each entity can be transferred concurrently. Typically, a browser may establish up to five or even ten concurrent connections. However, although this can reduce the overall time delay associated with a Web access, the use of multiple connections leads to added overheads at both the client and server sides since both must maintain state information for each of the connections. This can be particularly significant for a popular/busy server, which may get many hundreds of concurrent requests.

Hence with later versions of HTTP – version 1.1 onwards – unless informed differently, the server side leaves the initial TCP connection in place for the duration of the Web session. The TCP connection is then called **persistent** and, once in place, the browser may send multiple requests without waiting for a response to be received. Typically, the end of a session is determined by a timer expiring when no further transfers over the connection take place. Note, however, that the different versions of HTTP can interwork with each other.

Message formats

All HTTP request and response messages are NVT ASCII strings similar to those used with SMTP. With the earlier versions of HTTP – up to HTTP version 0.9 – what are called *simple request/response messages* are used. This means there is no type information associated with the request message, which comprises only the method – GET, HEAD, and so on – followed by the related file name in the form of an ASCII string. The response message is in the form of a block of ASCII characters with no headers and no MIME extensions. An example of a simple HTTP request message is:

GET/mpeg-4/index.html

which is sent over the previously established TCP connection to the related server AP.

With the later versions of HTTP – version 1.0 onwards – MIME extensions are supported using what are called *full request/response messages*. To discriminate a full request from a simple request, a field containing the HTTP version number is added to the request line. This is followed by the text associated with a number of other RFC 822 headers, each of which is on a separate line. These were given earlier in Table 8.1 and include:

- **general headers**: these do not relate to the entity to be transferred and an example is the MIME version number;
- **request headers**: these are used to specify such things as the sender"s name/e-mail address and the media types and encodings that the browser supports;

- **entity headers**: these relate to the entity to be transferred and include the content type and, when sending an entity, the content length.

The end of the header is indicated by a blank line. Then, if the request contains a message body, this is followed by the entity being transferred such as a block of HTML text – a script – relating to a page.

The header fields associated with a full response message start with the HTTP version number followed by the response status code. Some examples are:

200 accepted
304 not modified: the requested page has not been modified
400 bad request
404 not found: requested page does not exist on this server.

This is followed by the name and location of the server and, if the response contains an entity in the message body, a *Content-Type*: and a *Content-Length*: field. Also, if the contents relate to a binary file, they are followed by a *Content-Transfer-Encoding: Base64* header field.

An example showing a selection of the header fields associated with a full request/response message interchange is shown in Figure 9.8. The example relates to the transfer of the HTML page with the URL of:

http://www.mpeg.org/mpeg-4/index.html

Hence prior to sending the GET message, a TCP connection to the server *www.mpeg.org* will have been established.

The meanings of the various fields in the request message shown in part (a) of the figure are as follows. The *Connection: close* header line indicates to the server that the browser does not need a persistent connection. The *User-agent:* line contains the name of the browser and its version number. Often the server contains a number of versions of a page and this enables the server to send the version that is best suited to the browser. The *Accept:* line indicates the entity types that the browser is able to accept, which, as we can see, is determined by the compression software/hardware it supports.

The meanings of the various header fields in the response message shown in part (b) are mainly self explanatory. In the example, the body contains an entity comprising a string of NVT ASCII characters representing the HTML text of a Web page. Alternatively, if the *Content-Type:* was, say, *image/jpeg* then a *Content-Type-Encoding: Base64* header field would be present. For a more complete list of the MIME headers you should refer back to Table 8.1 and Figure 8.9 and their accompanying text.

Conditional GET

As we saw earlier in Figure 9.4(b), in many instances a browser does not communicate directly with the required server but rather through an

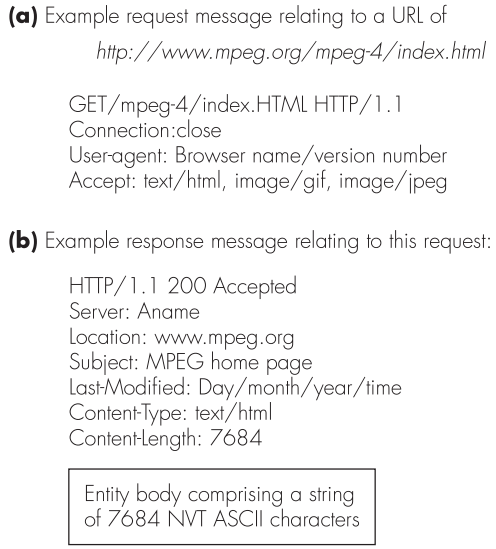


Figure 9.8 An example of a full request/response message relating to HTTP: (a) request message; (b) response message.

intermediate system called a proxy server. In the figure it was assumed that the browsers at a site supported only the HTTP protocol and that the proxy server was used to access the contents of files using different protocols such as FTP and NNTP. As we can deduce from our discussion of URLs, this will avoid each of the browsers having the code of each of these protocols. In addition, however, a proxy server normally caches the Web pages and other entities that it obtains – on behalf of the browsers that it serves – on hard disk. Then, when a browser makes a request for a page/entity that the proxy server has cached, the proxy server can return this directly without going back to the server holding the original source. The latter is called the **origin server** and, when it performs this function, the proxy server is also known as a (Web) **cache server**.

Although caching reduces the response time for subsequent requests for a cached page/entity, there is a possibility that the cached entity may be out of date as a result of the original being modified/updated subsequent to the cache server receiving it. Hence because caching is widely used, an additional request message called a **conditional GET** is used by the cache server to ensure the response messages that are returned to the browsers contain up-to-date information.

A conditional GET request message is one that includes a header line of *If-Modified-Since:* and its use is illustrated in Figure 9.9. To avoid duplication, the header fields that have already been discussed are left out of the messages

shown. The following should be noted when interpreting the message sequence.

- It is assumed that the browsers in all the client machines attached to the access network – site/campus LAN, ISP network, and so on – have been configured to send all request messages to the proxy/cache server.
- The sequence starts with a browser requesting an HTML page from the proxy server using a GET request message (1).
- The proxy server has a cached copy of the page and, associated with it, the day/date/time when the page was cached. This is obtained from the *Last-Modified:* header field in the response message returned by the origin server to an earlier request.
- Before returning the cached page to the browser, the proxy server sends a conditional GET request message to the origin server – defined in the page URL – with the date and time the current copy of the page it holds was last modified in the *If-Modified-Since:* header field (2).
- On receipt of this, the origin server checks to see if the requested page has been modified since the date in the *If-Modified-Since:* field.
- In the figure it is assumed that the contents have not been changed and hence the origin server returns a simple response message with a status code of *304 Not modified* in the header and an empty entity body (3).
- On receipt of this, the proxy server returns a copy of the cached page to the browser (4).

In the event that the requested page/entity had been changed, then a copy of the new page/entity would be returned by the origin server. A copy of

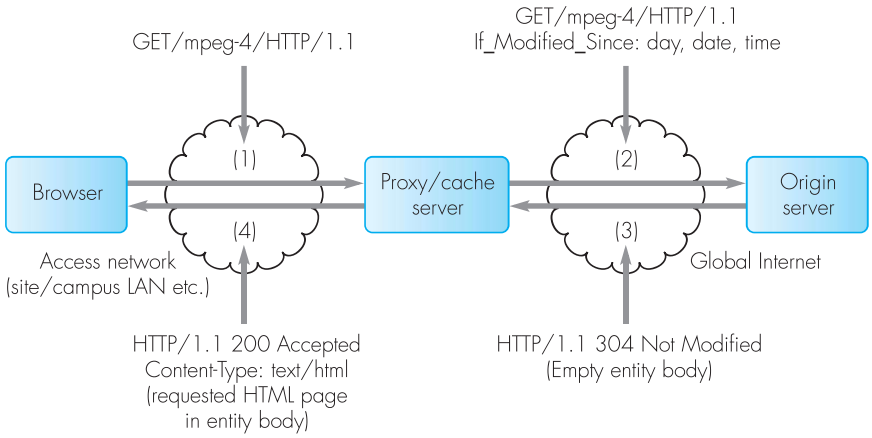


Figure 9.9 Proxy/cache server operation with conditional GET.

the new page would then be cached by the proxy server – together with the date and time from the *Last-Modified:* field – before it is forwarded to the browser. Thus the savings obtained with a cache server come from the absence of an entity body in the response message from the origin server. Clearly for large pages/entities this can be considerable. In addition, further savings can be obtained by also having a higher-level cache server associated with, for example, each regional/national network. The proxy server associated with each access network is then configured to send all request messages to a specified higher-level cache server. In general, the higher the level this is in the Internet hierarchy the more requests it will receive and, as a result, the more cached pages/entities it holds.

9.4 HTML

The HyperText Markup Language, HTML, is the standard language used to write Web pages. As we saw in Section 9.2.1, HTML is the markup language that is used to describe how the contents of a document/page are to be displayed on the screen of the computer by the browser. Moreover, since the markup commands relate to a complete page, the browser automatically displays the page contents within the bounds allocated for the page; that is, irrespective of whether this is a small window on a low resolution screen or a large window on a high-resolution screen.

The markup/format commands are known as directives in HTML and the majority are specified using a pair of **tags**. In addition to the various types of directive associated with a string of text – which specify how the string is to be presented on the display – there are tags to enable a hyperlink to be specified as well as tags to specify an image or a segment of audio or video within a page and how these are to be displayed/output. There are also fill-in forms and other features. In this section we describe how each of these features is specified in HTML. It should be stressed, however, that HTML is continuously being revised and what follows should be considered only as an introduction to the subject.

9.4.1 Text format directives

The HTML text associated with a Web page is written in the ISO 8859-1 Latin-1 character set, which, for the English alphabet, is the same as the ASCII character set. However, for someone creating Web pages in a Latin alphabet, when using an ASCII keyboard, escape sequences must be used. For example, the Latin character *è* is represented by the ASCII string *&egrave* and *é* by *&eacute*.

The HTML text can be entered using either a word processor with facilities for creating and editing an HTML document/page or directly using the facilities provided by a Web browser. Typically, the complete string of characters