The World Wide Web

HAL BERGHEL AND DOUGLAS BLANK

Department of Computer Science
University of Arkansas
Fayetteville, AR 72701
USA

Abstract

This article provides a high-level overview of the World Wide Web in the context of a wide range of other Internet information access and delivery services. This overview will include client-side, server-side and "user-side" perspectives. Underlying Web technologies as well as current technology extensions to the Web will also be covered. Social implications of Web technology will also be addressed.

ADVANCES IN COMPUTERS, VOL. 48
ISBN 0-12-012148-4

1. Introduction

The World Wide Web, or "the Web," is a "finite but unbounded" collection of media-rich digital resources which are connected through high-speed digital networks. It relies upon an Internet protocol suite (see [9] [19]) which supports the cross-platform transmission and rendering of a wide variety of media types (i.e. multimedia). This cross-platform delivery environment represents an important departure from more traditional network communications protocols like email, Telnet and FTP because it is content-centric. It is also to be distinguished from earlier document acquisition systems such as Gopher and WAIS (Wide Area Information Systems) which accommodated a narrower range of media formats and failed to include hyperlinks within their network navigation protocols. Following Gopher, the Web quickly extended and enriched the metaphor of integrated browsing and navigation. This made it possible to navigate and peruse a wide variety of media types on the Web effortlessly, which in turn led to the Web's hegemony as an Internet protocol.

Thus, while earlier network protocols were special-purpose in terms of both function and media formats, the Web is highly versatile. It became the first convenient form of digital communication which had sufficient rendering and browsing utilities to allow any person or group with network access to share media-rich information with their peers. It also became the standard for hyper-linking cybermedia (cyberspace multimedia), connecting concept to source in manifold directions identified primarily by Uniform Resource Locators (URLs).

In a formal sense, the Web is a client-server model for packet-switched, networked computer systems defined by the protocol pair Hypertext Transfer Protocol (HTTP) and Hypertext Markup Language (HTML). HTTP is the primary transport protocol of the Web, while HTML defines the organization and structure of the Web documents to be exchanged. At this writing, the current HTTP standard is at version 1.0, and the current HTML version is 4.0.

HTTP and HTML are higher-order Internet protocols specifically created for the Web. In addition, the Web must also utilize the lower-level Internet protocols, Internet Protocol (IP) and Transmission Control Protocol (TCP). The basic Internet protocol suite is thus designated TCP/IP. IP determines how datagrams will be exchanged via packet-switched networks while TCP builds upon IP by adding control and reliability checking [9, 20].

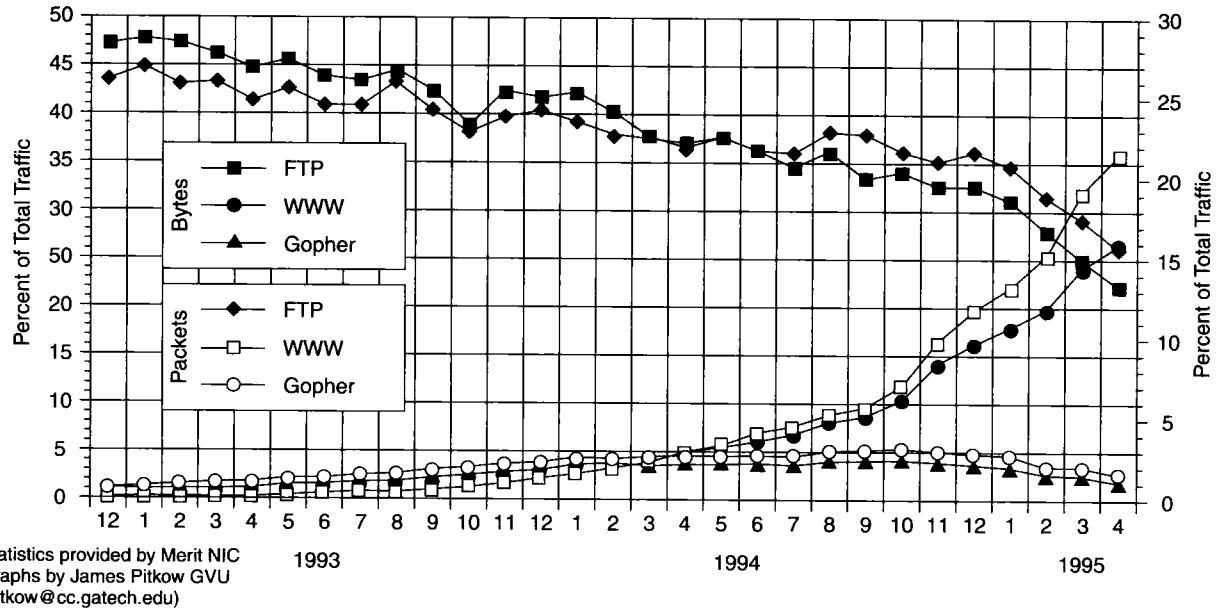

FIG. 1. Merit NIC Backbone statistics for the Web, Gopher and FTP from 1993–1995 in terms of both packet and byte counts (source: Merit NIC and Jim Pitkow [18], used with permission).

According to NSFNET Backbone statistics, the Web moved into first place both in terms of the percentage of total packets moved (21%) and percentage of total bytes moved (26%) along the NSF backbone in the first few months of 1995. This placed the Web well ahead of the traditional Internet activity leaders, FTP (14%/21%) and Telnet (7.5%/2.5%), as the most popular Internet service. A comparison of the evolutionary patterns of the Web, Gopher and FTP are graphically depicted in Fig. 1.

2. The Internet: Precursor to the Web

The Web evolution should be thought of as an extension of the digital computer network technology which began in the 1960s. Localized, platform-dependent, low-performance networks became prevalent in the 1970s. These LANS (local area networks) were largely independent of, and incompatible with, each other. In a quest for technology which could integrate these individual LANs, the US Department of Defense, through its Advanced Research Projects Agency (ARPA, nee DARPA), funded research in inter-networking—or inter-connecting LANS via a wide area network (WAN). The first national network which resulted from this project was called, not surprisingly, ARPANET. For most of the 1970s and 1980s ARPANET served as the primary network backbone in use for interconnecting LANs for both the research community and the US Government.

At least two factors considerably advanced the interest in the ARPANET project. First, and foremost, it was an "open" architecture: its underlying technological requirements, and software specifications were available for anyone to see. As a result, it became an attractive alternative to developers who bristled at the notion of developing software which would run on only a certain subset of available platforms.

Second, ARPANET was built upon a robust, highly versatile and enormously popular protocol suite: TCP/IP (Transmission Control Protocol/Internet Protocol). The success and stability of TCP/IP elevated it to the status of de facto standard for inter-networking. The US military began using ARPANET in earnest in the early 1980s, with the research community following suit. Since TCP/IP software was in essence in the public domain, a frenzy of activity in deploying TCP/IP soon resulted in both government and academe. One outgrowth was the NSF-sponsored CSNET which linked computer science departments together. By the end of the 1980s, virtually every one who wanted to be inter-networked could gain access through government or academic institutions. ARPANET gradually evolved into the Internet, and the rest, as they say, is history.

Not unexpectedly, the rapid (in fact, exponential) growth produced some problems. First and foremost was the problem of scalability. The original

ARPANET backbone was unable to carry the network traffic by the mid-1980s. It was replaced by a newer backbone supported by the NSF, a backbone operated by a consortium of IBM, MCI and Merit shortly thereafter, and finally by the privatized, not-for-profit corporation, Advanced Networks and Services (ANS), which consisted of the earlier NSFNET consortium members. In the mid-1990s, the MCI corporation deployed the high-speed Backbone Network System (vBNS) which completed the trend toward privatization of the digital backbone networks. The next stage of evolution for inter-network backbones is likely to be an outgrowth of the much-discussed Internet II project proposed to the US Congress in 1997. This "next generation" Internet is expected to increase the available bandwidth over the backbone by two orders of magnitude.

Of course, there were other network environments besides the Internet which have met with varying degrees of success. Bitnet was a popular alternative for IBM mainframe customers during the 1970s and early 1980s, as was UUCP for the Unix environment and the email-oriented FIDONET. Europeans, meanwhile, used an alternative network protocol, D.25, for several of their networks (Joint Academic Network—JANET, European Academic and Research Network—EARN). By 1991, however, the enormous popularity of the Internet drove even recalcitrant foreign network providers into the Internet camp. High-speed, reliable Internet connectivity was assured with the European Backbone (EBONE) project. At the time of writing all but a handful of developing countries have some form of Internet connectivity. For the definitive overview of the Internet, see [9].

3. The Success of the Web

It has been suggested [2] that the rapid deployment of the Web is a result of a unique combination of characteristics:

(1) *The Web is an enabling technology.* It was the first widespread network technology to extend the notion of "virtual network machine" to multimedia. While the ability to execute programs on, and retrieve content from, distributed computers was not new (e.g. Telnet and FTP were already in wide use by the time that the Web was conceived), the ability to produce and distribute media-rich documents via a common, platform-independent document structure, was new to the Web.

(2) *The Web is a unifying technology.* The unification came through the Web's accommodation of a wide range of multimedia formats. Since such audio (e.g. .WAV, .AU), graphics (e.g. .GIF, .JPG) and animation (e.g. MPEG) formats are all digital, they were already unified in desktop applications prior to the Web. The Web, however, unified

them for distributed, network applications. One Web "browser," as it later became called, would correctly render dozens of media formats regardless of network source. In addition, the Web unifies not only the access to many differing multimedia formats, but provides a platform-independent protocol which allows anyone, regardless of hardware or operating system, access to that media.

(3) *The Web is a social phenomenon.* The Web social experience evolved in three stages. Stage one was the phenomenon of Web "surfing". The richness and variety of Web documents and the novelty of the experience made Web surfing the de facto standard for curiosity-driven networking behavior in the 1990s. The second stage involved such Web interactive communication forums as Internet Relay Chat (IRC), which provided a new outlet for interpersonal but not-in-person communication. The third stage, which is in infancy at the time of writing, involves the notion of virtual community. The widespread popularity and social implications of such network-based, interactive communication is becoming an active area in computing research.

4. Perspectives

4.1 End User's Perspective

Extensive reporting on Web use and Web users may be found in a number of Web survey sites. Perhaps the most thorough of which is the biannual, self-selection World Wide Web Survey which began in January, 1994 (see reference, below). As this article is being written, the most current Web Survey is the ninth (April, 1998). Selected summary data appear in Table I.

Of course a major problem with self-selection surveys, where subjects determine whether, or to what degree, they wish to participate in the survey, is that the samples are likely to be biased. In the case of the Web survey, for example, the authors recommend that the readers assume biases towards the experienced users. As a consequence, they recommend that readers confirm the results through random sample surveys. Despite these limitations, however, the Web surveys are widely used and referenced and are among our best sources of information on Web use.

An interesting byproduct of these surveys will be an increased understanding of the difference between traditional and electronic surveying methodologies and a concern over possible population distortions under a new, digital lens. One may only conjecture at this point whether telephone respondents behave similarly to network respondents in survey settings. In

TABLE I

SUMMARY INFORMATION ON WEB USE

Average age of Web user	35.7 years
Male : female ratio of users	62 : 38
% users with college degrees	46.9
% in computing field	20.6
% in education	23.4
% in management	11.7
% of users from US	80.5 (and slowly decreasing)
% of users who connect via modems with transmission speeds of 33.5 kbps or less	55
% of respondents reported who use the Web for purchases exceeding $100	39
% of users for whom English is the primary language	93.1
% of users who have Internet bank accounts	5.5
% of Microsoft Windows platforms	64.5
% using Apple	25.6
% of users who plan to use Netscape	60
% who plan to use Internet Explorer	15

Source: GVU's WWW User Surveys, `http://www.cc.gatech.edu/gvu/user_surveys/`. Used with permission.

addition, Web surveyors will develop new techniques for non-biased sampling which avoids the biases inherent in self-selection. The science and technology behind such electronic sampling may well be indispensable for future generations of Internet marketers, communicators, and organizers.

4.2 Historical Perspective

The Web was conceived by Tim Berners-Lee and his colleagues at CERN (now called the European Laboratory for Particle Physics) in 1989 as a shared information space which would support collaborative work. Berners-Lee defined HTTP and HTML at that time. As a proof-of-concept prototype, he developed the first Web client navigator-browser in 1990 for the NeXTStep platform. Nicola Pellow developed the first cross-platform Web browser in 1991 while Berners-Lee and Bernd Pollerman developed the first server application—a phone book database. By 1992, the interest in the Web was sufficient to produce four additional browsers—Erwise, Midas, and Viola for X Windows, and Cello for Windows. The following year, Marc Andreessen of the National Center for Supercomputer Application (NCSA) wrote Mosaic for the X Windows System which soon became the browser standard against which all others would be compared. Andreessen went on to co-found Netscape Communications in 1994 whose current browser, Netscape Navigator, remains the current de facto standard Web browser,

despite continuous loss of market share to Microsoft's Internet Explorer in recent years (see, Fig. 2). Netscape has also announced plans to license without cost the source code for version 5.0 to be released in Spring, 1998. At this point it is unclear what effect the move to "open sources" may have (see Figs 2 and 3).

Despite the original design goal of supporting collaborative work, Web use has become highly variegated. The Web has been extended into a wide range of products and services offered by individuals and organizations, for commerce, education, entertainment, "edutainment", and even propaganda. A partial list of popular Web applications includes:

- individual and organizational homepages;
- sales prospecting via interactive forms-based surveys;
- advertising and the distribution of product promotional material;
- new product information, product updates, product recall notices;
- product support—manuals, technical support, frequently asked questions (FAQs);
- corporate record-keeping—usually via local area networks (LANs) and intranets;
- electronic commerce made possible with the advent of several secure HTTP transmission protocols and electronic banking which can handle small charges (perhaps at the level of millicents);

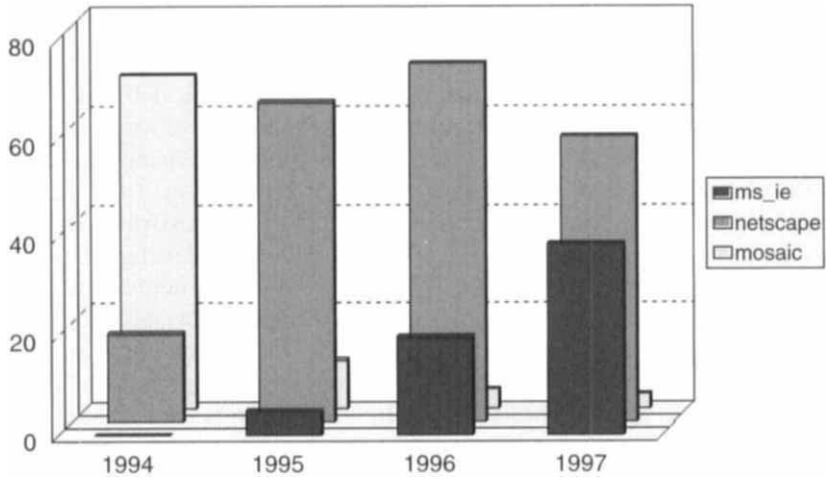

FIG. 2. Market share of the three dominant Web browsers from 1994 through 1997.

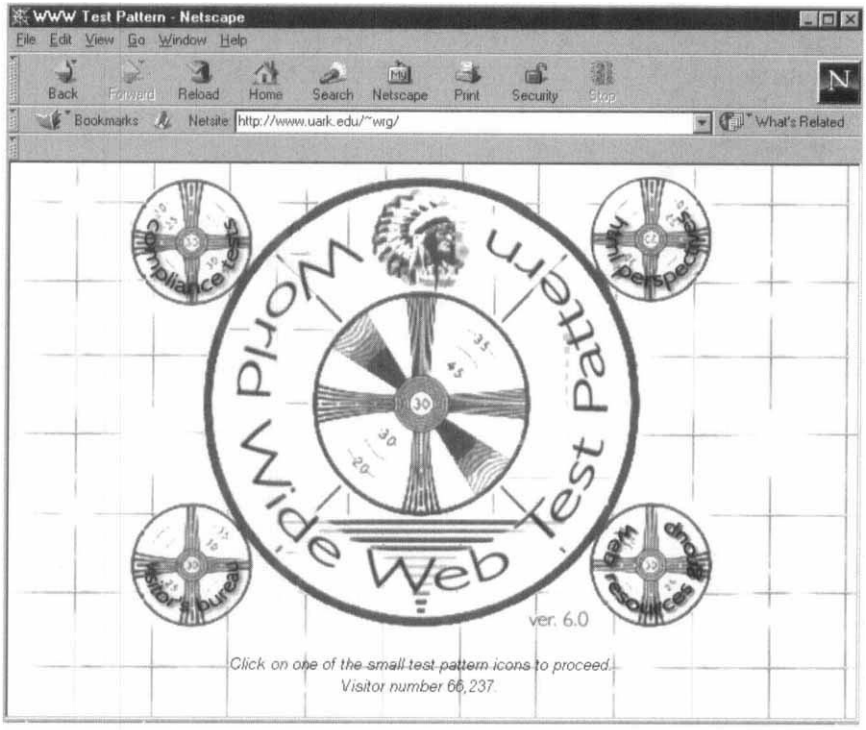

FIG. 3. Navigator 4.x is a recent generic "navigator/browser" from Netscape Corporation. Displayed is a vanilla "splash page" of the World Wide Web Test Pattern—a test bench for determining the level of HTML compliance of a browser.

- religious proselytizing;
- propagandizing;
- digital politics.

Most Web resources at this time are still set up for non-interactive, multimedia downloads (e.g. non-interactive Java [21] animation applets, movie clips, real-time audio transmissions, text with graphics). This will change in the next decade as software developers and Web content-providers shift their attention to the interactive and participatory capabilities of the Internet, the Web, and their successor technologies. Already, the Web is eating into television's audience and will probably continue to do so. Since it seems inevitable that some aspects of both television and the Web will merge in the 21st century, they are said to be convergent technologies. But as of this writing, the dominant Web theme seems to remain static HTML documents and non-interactive animations.

As mentioned above, the uniqueness of the Web as a network technology is a product of two protocols: HTML and HTTP. We elaborate on these protocols below.

5. The Underlying Technologies

5.1 Hypertext Markup Language (HTML)

HTML is the business part of document preparation for the Web. Two not-for-profit organizations play a major role in standardizing HTML: the World Wide Web Consortium (`www.w3.org`) and the Internet Engineering Task Force (`www.ietf.org`). Any document which conforms to the W3C/IETF HTML standards is called a Web message entity. HTML is about the business of defining Web message entities.

The hypertext orientation of HTML derives from the pioneering and independent visions of Vannevar Bush [8] in the mid-1940s, and Doug Englebart [10] and Ted Nelson [14] in the 1960s. Bush proposed mechanical and computational aids in support of associative memory, i.e. the linking together of concepts which shared certain properties. Englebart sought to integrate variegated documents and their references through a common core document in a project called Augment. Nelson, who coined the terms "hypertext" and "hypermedia," added to the work of Bush and Englebart the concept of non-linear document traversal, as his proposed project Xanadu (`www.xanadu.net/the.project`) attempted to "create, access and manipulate this literature of richly formatted and connected information cheaply, reliably and securely from anywhere in the world." Subsequently, Nelson has also defined the notions of "transclusion," or virtual copies of collections of documents, and "transcopyright" which enables the aggregation of information regardless of ownership by automating the procedure by means of which creators are paid for their intellectual property. We won't comment beyond saying that the Web is an ideal test bed for Nelson's ideas.

From technical perspective, HTML is a sequence of "extensions" to the original concept of Berners-Lee—which was text-oriented. By early 1993, when the NCSA Mosaic navigator-browser client was released for the X Windows System, HTML had been extended to include still-frame graphics. Soon audio and other forms of multimedia followed.

After 1993, however, HTML standards were a moving target. Marc Andreessen, the NCSA Mosaic project leader, left the NCSA to form what would become Netscape Corporation. Under his technical supervision, Netscape went its own way in offering new features which were not endorsed by W3C/IETF, and at times were inconsistent with the Standard

Generalized Markup Language (SGML) orientation intended by the designers of HTML. SGML is a document definition language which is independent of any particular structure—i.e. layout is defined by the presentation software based upon Under pressure to gain market share, navigator/browser developers attempted to add as many useful "extensions" to the HTML standards as could be practicably supported. This competition has been called the "Mosaic War," [3] which persists in altered form even to this day.

Although not complete, Table II provides a technical perspective of the evolution of HTML.

TABLE II

HTML EVOLUTION

GML (Generalized Markup Language)	Developed by IBM in 1969 to separate form from content in displaying documents
SGML (ISO 8879 Standard Generalized Markup Language)	Adopted 1986
HTML Version 1 (circa 1992–3)	basic HTML structure rudimentary graphics hypertext
HTML Version 2 (circa 1994)	forms lists
HTML Version 3.2 (circa 1996–7)	tables applets scripts advanced CGI programming security text flow around graphics
HTML Version 4.x (early 1998)	inline frames format via cascading style sheets (vs HTML tags) compound documents with hierarchy of alternate rendering strategies internationalization tty and braille support client-side image maps advanced forms/tables
XML (1998)	Extensible Markup Language. Subset of SGML

Note:

(1) Version 3.2 is actually a subset of Version 3.0, the latter of which failed to get endorsed by W3C/IETF.

(2) Dates are only approximate because of the time lag between the introduction of the technology and the subsequent endorsement as a standard. In some cases this delay is measured in years.

Among the many Netscape innovations are:

- typographical enhancements and fonts;
- alignment and colorization controls for text and graphics;
- dynamic updating (continuous refresh without reload);
- server push/client pull;
- frames;
- cookies;
- plug-ins;
- scripts;
- frames;
- Java applets;
- layers.

Many of these have become part of subsequent HTML standards. In addition to these formal standards, discussion is already underway for a radical extension of HTML called XML (`www.w3.org/XML/Activity`). In many ways, HTML evolved away from its nicely thought-out roots. GML, or Generalized Markup Language, was developed in the 1960s at IBM to describe many different kinds of documents. Standard Generalized Markup Language, or SGML, was based on GML and became an ISO standard years later in the 1980s. SGML still stands today as the mother of all markup languages. Its designers were very careful not to confuse form and content, and created a wonderfully rich language. HTML became a patchwork of ideas as it quickly evolved over the last few years, and muddied the difference between form and content. XML is an effort to reunite HTML with its SGML roots. The development of XML, which began in late 1996, deals with the non-extensibility of HTML to handle advanced page design and a full range of new multimedia. XML will accomplish this by using (1) a more SGML-like markup language (vs HTML), allowing "personal" or "group" -oriented tags, and (2) a low-level syntax for data definition.

To see how XML differs from HTML, we examine a page of HTML code:

```
<html>
<head>
     <title>Bibliography</title>
</head>
<body>
     <p>Smith, Aaron S. (1999). <i>Understanding the Web</i>.
        Web Books, Inc. </p>
</body>
</html>
```

This code, when rendered by an appropriate browser, would appear similar to the following:

Smith, Aaron S. (1999). *Understanding the Web.* Web Books, Inc.

Tags are special symbols in HTML and XML, and are indicated by the surrounding less-than and greater-than symbols. The majority of tags are paired—i.e. they surround the text that they affect. For example, <I> and </I> indicate that the italics should be turned on and off, respectively.

Now, contrast the HTML example with sample XML code:

```
<?XML version="1.0" ?>
<xmldoc>
<bibliography>
      <ref-name> Smith-1999b </ref-name>
      <name>
            <last> Smith </last>
            <first> Aaron </first>
            <mi> S </mi>
      </name>
      <title> Understanding the Web </title>
      <year> 1999 </year>
      <publisher> Web Books, Inc. </publisher>
      <type> Book </type>
</bibliography>
</xmldoc>
```

Like the HTML code, XML is made up of tags. However, XML does not describe how to render the data, it merely indicates the structure and content of the data. HTML does have some of these kinds of tags (for example, `<title>` in the above HTML example) but, for the most part, HTML has evolved completely away from its SGML roots.

XML was designed to be compatible with current HTML (and SGML, for that matter). Today's most common Web browsers (Microsoft's Internet Explorer and Netscape's Navigator) do not support XML directly. Instead, most XML processors have been implemented as Java applications or applets (see the Web Consortium's website for a list of XML processors at `www.w3.org`). Such a Java processor could be instructed to render the XML inside the browser exactly like the rendered HTML.

One of the nice properties of XML is the separation of content and format. This distinction will surely help tame the Wild Web as it will allow easier searching, better structuring, and greater assistance to software agents in general. However, this isn't XML's greatest virtue: what makes XML a great leap forward for the Web is its ability to create new tags. Much like a modern database management system can define new fields, XML can

create a new tag. In addition, XML tags can also have structure like the name field above was composed of first, last, and middle initial. As long as client and server agree on the structure of the data, they can freely create and share new data fields, types, and content via XML.

Some have said that XML "does for data what Java does for programs." Examples of XML applications are the math-formula markup language, MathML (`www.w3.org/TR/WD-math/`), which combines the ability to define content with a less-powerful suite of features to define presentation. Another example is RDF, a resource description format for metadata (`http://www.w3.org/RDF/Overview.html`), which is used in both PICS, the Platform for Internet Content Selection (`http://www.w3.org/PICS/`) and SMIL, the Synchronized Multimedia Integration Language, which is a declarative language for synchronizing multimedia on the Web. The XML prototype client is Jumbo (`www.venus.co.uk/omf/cml`). Although XML will help make marking up Web pages easier, there is still a battle raging over which system should be responsible for the details of rendering pages. Current HTML coders must take responsibilty for exact placement over page layout, and getting a standard look across browsers is non-trivial. However, SGML leaves the page layout details up to the browser. Exactly how this important issue will play out remains to be seen.

5.2 Hypertext Transfer Protocol (HTTP)

HTTP is a platform-independent protocol based upon the client-server model of computing which runs on any TCP/IP, packet switched digital network—e.g. the Internet. HTTP stands for Hyper Text Transfer Protocol and is the communication protocol with which browsers request data, and servers provide it. This data can be of many types including video, sound, graphics, and text. In addition, HTTP is extensible in that it can be augmented to transfer types of data that do not yet exist.

HTTP is an application layer protocol, and sits directly on top of TCP (Transmission Control Protocol). It is similar in many ways to the File Transmission Protocol (FTP) and TELNET. HTTP follows the following logical flow:

(1) A connection from the client's browser is made to a server, typically by the user having clicked on a link.
(2) A request is made of the server. This request could be for data (i.e. a "GET") or could be a request to process data (i.e. "POST" or "PUT").
(3) The server attempts to fulfill the request. If successful, the client's

browser will receive additional data to render. Otherwise, an error occurs.

(4) The connection is then closed.

HTTP uses the same underlying communication protocols as do all the applications that sit on top of TCP. For this reason, one can use the TELNET application to make an HTTP request. Other TCP-based applications include FTP, TFTP (Trivial File Transfer Protocol), and SMTP (Simple Mail Transfer Protocol), to name just a few. Consider the following example:

```
% telnet www.uark.edu 80
GET / HTTP/1.0
Accept: text/html
Accept: text/plain
User-Agent: TELNET/1.0
```

This command made from any operating system with access to the TELNET program requests to talk to port 80, the standard HTTP port, of a machine running a web server (TELNET normally uses port 23). A request is made to get the root document (GET /), in a particular protocol (HTTP/1.0), and accepting either text or HTML. The data (i.e. HTML codes) are returned, and the connection is closed. Note: the ending empty line is required. Conversely, consider:

```
HTTP/1.0 200 OK
Server: Netscape-Enterprise/3.0K
Date: Sun, 03 May 1998 22:25:37 GMT
      Content-type: text/html
      Connection: close

<HTML>
<HEAD>
...
```

These are the data returned from the previous request. First, the server responds with the protocol (HTTP/1.0 in this example), gives the corresponding code (200 OK), provides details of the server (Netscape-Enterprise), date and time, and the format of the following data (text/html). Finally, an empty line separates the header from the actually HTML code.

This type of processing is called "stateless". This makes HTTP only slightly, yet importantly, different from FTP. FTP has "state"; an FTP session has a series of settings that may be altered during the course of a dialog between client and server. For example, the "current directory" and "download data type" settings maybe be changed during an FTP dialog. HTTP, on the other hand, has no such interaction—the conversation is

limited to a simple request and response. This has been the most limiting aspect of HTTP. Much current Web development has centered around dealing with this particular limitation of the protocol (i.e. cookies).

Although HTTP is very limited, it has shown its flexibility through what must be one of the most explosive and rapidly changing technological landscapes ever. This flexibility is made possible via the protocol's format negotiations. The negotiation begins with the client identifying the types of formats it can understand. The server responds with data in any of those formats that it can supply (text/html in the above example). In this manner, the client and server can agree on file types yet to be invented, or which depend on proprietary formats. If the client and server cannot agree on a format, the data is simply ignored.

6. Dynamic Web Technologies

Web technologies evolved beyond the original concept in several important respects. We examine HTML forms, the Common Gateway Interface, plug-ins, executable content, and push technologies.

6.1 Common Gateway Interface

The support of the Common Gateway Interface (CGI) within HTTP in 1993 added interactive computing capability to the Web. Here is a one-line C program that formats the standard greeting in basic HTML. (Note: make sure the binary is marked executable. Also, often the binary will need to have a. cgi extension to tell the HTTPD server that it should be executed rather than simply displayed). Any program capable of reading from "standard input" and writing to "standard output" can be used as a CGI program, although the interpreted language Perl has, by far, been the most used. The following code fragment illustrates this point:

```
main() {
   printf("Content-type: text/html\n\n<html><body>
           <h1>Hello World\!</body></html>\n");
}
```

Many are surprised to find the amount of data a CGI program has access to from the apparent anonymous browser (more on this later). For example, a CGI program can identify what Web page referred the user to this site, the browser the user is using, the user's IP address, and a host of other information (including the host) (see below):

```
DOCUMENT_ROOT            /home/csci/public_html
```

```
GATEWAY_INTERFACE       CGI/1.1
HTTP_ACCEPT             image/gif, image/x-xbitmap,
                        image/jpeg, image/pjpeg,
                        image/png, /
HTTP_ACCEPT_CHARSET     iso-8859-1, *,utf-8
HTTP_ACCEPT_ LANGUAGE   en
HTTP_CONNECTION         Keep-Alive
HTTP_HOST               entropy.uark.edu
HTTP_REFERER            http://dangermouse.uark.
                        edu/~dblank/samples/
HTTP_USER_AGENT         Mozilla/4.04 [en] (X11; I;
                        Linux 2.0.33 i686)
PATH                    /sbin:/usr/sbin:/bin:/usr/bin
QUERY_STRING
REMOTE_ADDR             130.184.201.233
REMOTE_HOST             dangermouse.uark.edu
REQUEST_METHOD          GET
SCRIPT_FILENAME         /home/dblank/public_html/scm.cgi
SCRIPT_NAME             /~dblank/scm.cgi
SERVER_ADMIN            root@localhost
SERVER_NAME             entropy.uark.edu
SERVER_PORT             80
SERVER_PROTOCOL         HTTP/1.0
SERVER_SOFTWARE         Apache/1.2.5
```

In general, a CGI program has access to environment information regarding the network transaction. This browser data is relayed to the CGI program via environment variables.

A CGI program may create a specialized Web page that is based on any of the above variable values. For example, if a user did not come from a specific Web page, then the user could be rejected. In the example below, one could test `HTTP_REFERER` to see if it equals "`http://dangermouse.uark.edu/~dblank/samples/`". If it does, more data would be shown; otherwise a message indicating inappropriate access could be displayed.

```
  include <stdlib.h>
main() {
  if (strcmp(getenv("HTTP_REFERER"),
           "http://dangermouse.uark.edu/~dblank/samples/")
            == 0 ) printf("Content-type:
text/html\n\n<html><body><h1>Welcome\!");
  else
     printf("Content-type: text/html\n\n<html><body><h1>Access
  Denied<p>"); printf("</body></html>\n");
}
```

In this case, "more data" is a dynamic Web page. This CGI C program tests to make sure that the user is coming from a particular HTTP referrer.

Perhaps the most important use of CGI to this point has been the dynamic processing of CGI forms which enable input from the Web user-client to be passed to the server for processing (discussed below). While, in theory, CGI programs can provide server-side programming for virtually any Web need, network bandwidth constraints and transmission delays may make some heavily interactive and volumetric applications infeasible.

6.2 Forms

Forms were added to HTML version 2 around 1994. Forms allow users to give feedback to servers through standard GUI objects: text boxes, check boxes, buttons, etc. (see Fig. 4).

The HTML code below produced the screen in Fig. 4. Sections of HTML code are marked between the `<FORM>` and `</FORM>` tags. In this example, the CGI program (discussed below) is executed when the "View" button is clicked. Three additional pieces of data are sent to the executing program via environment variables: `passwd`, `file`, and `html`. `Passwd` and `file` are both textual, and `html` is a boolean check box. Notice that `passwd` is of type "password" which makes text typed into the area appear as asterisks.

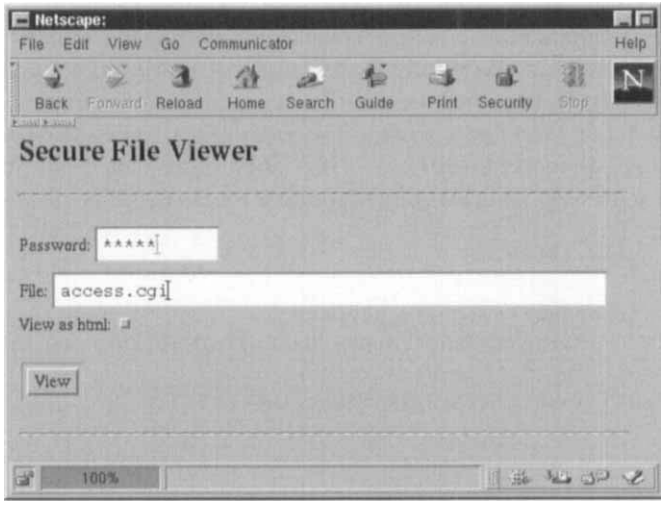

FIG. 4. Screen dump of Secure File Viewer form.

The word "checked" appearing at the end of the checkbox tag indicates that is initially true (i.e. checked).

```
<HTML>
<BODY>
<h1>Secure File Viewer</h1>
<HR>
<FORM METHOD="POST" ACTION="access.cgi">
Password:
<input type=password name=passwd value="" size=10></input>
File: <input type=text name=file value=""
size=50></input><br>
View as html: <input type=checkbox name=html checked>
<br>
<br>
<input type="submit" name="submit" value=" View ">
</FORM>
<HR>
</BODY>
</HTML>
```

The final piece of this form is the actual program that is run when "View" is pressed. This program, `access.cgi`, is shown below:

```
#!/usr/bin/perl
require "cgi-lib.pl";
&ReadParse(*input);
if ($input{'passwd'} eq 'himom') {
  if ($input{'html'}) {
    exec "echo Content-TYPE: text/html; echo; cat
    ".$input{'file'} || die;
  } else {
     exec "echo Content-TYPE: text/plain; echo; cat
    ".$input{'file'} || die;
  }
} else {
  print "Content-type: text/html\n\n"; print
"<HTML><BODY><HR>"; print "Error not the correct
password!\n";
}
print "<HR></BODY></HTML>\n";
1;
```

We observe that this is a simple Perl script that checks a password before viewing a source file.

This program should be marked as executable, and requires the standard Perl library `cgi-lib.pl`.

`Access.cgi` is a CGI program (described below) written in Perl that checks for a password ("`himom`" in this case), and displays a file either as HTML or as plain text depending on the checkbox. This method of password protecting files is not secure: anyone with access to the file system can read `access.cgi` and therefore view the password (read access is required in order to execute a CGI script on most Web servers). Currently, there is not a foolproof method for allowing password-bearing Web surfers to have access to files while preventing general file system users. Many sites have a dedicated server with no general-purpose users for just this purpose, thereby making this form secure.

Forms were a welcome addition to early HTML standards. However, the amount of dynamic interaction allowed is quite limited.

6.3 Helper apps

So-called "Helper apps" are extensions of the network browser metaphor which diminished the browser-centricity by supporting multimedia through separate, special-purpose "players." In this way, a wider range of multimedia could be rendered than could be economically and practicably built into the browser itself.

This trend toward helper apps began in 1994. By the year's end most Web browsers included generic launchpads which could spawn pre-specified multimedia players based on the filetype/file extent (`.WAV` designated MS Window's audio file, `.QT` designated Quicktime, etc.). In practice, one simply downloaded the autonomous helper apps which could cover the range of multimedia desired.

The generic launchpad was a significant technological advance for two reasons. First, it de-coupled the evolutionary paths, and hence the development paces, of browsers and multimedia. The first multimedia Web browsers relied entirely on internal media perusers, thereby creating a bottleneck as the pace of development of new multimedia formats exceeded that of the internal perusers. By de-coupling, both browser and multimedia developers could advance independently without fear of incompatibility.

Second, generic launchpads spawn external processes which execute independently of the Web browser and hence render the multimedia in an external window. This process-independence discourages the development of helper apps that are proprietary to a particular browser, which led to the rapid growth of freeware, shareware and commercial helper apps that are now available for popular client platforms. That the helper apps could be used in isolation of the browser became a collateral advantage for easy perusal of local multimedia files as well.

This generic, browser-independent approach toward rendering multimedia would be challenged twice in 1996, first by "plug-ins" and then by "executable content."

6.4 Plug-ins

"Plug-in" (alt. "Add-on") technology increased the media-rendering capability of browsers while avoiding the time-consuming spawning of so-called "helper apps" through the browser's launchpad. The speed advantage of the plug-ins, together with the tight coupling that exists between the plug-ins and the media formats which they render, made them a highly useful extension.

Plug-ins, as the name implies, are external applications which extend the browser's built-in capability for rendering multimedia files. However, unlike helper apps plug-ins render the media "inline"—that is, within the browser's window in the case of video, or with simultaneous presentation in the case of audio. In this way the functionality of the plug-in is seamlessly integrated with the operation of the browser. Plug-ins are often proprietary and browser-specific because of this tight integration. Some of the more popular current plug-in technologies are Web telephony, virtual reality and 3-D players, and real-time (or, "streaming") audio and video. (The word "streaming" has come to mean the ability of a multimedia file to begin to be experienced by the user before the entire file has completely downloaded. For example, an hour-long audio file may begin playing after a few seconds of downloading and continue playing while the rest of the file is downloaded. This works by having the plug-in download a buffer of data prior to execution, thereby keeping just ahead of the player.)

Plug-ins proved to be a useful notion for creating extendable browsers. However, there was a rub: plug-in developers must write and compile code for each target platform. Although most modern platforms support C and C++ compilers, porting plug-in applications is still non-trivial. This requirement was eliminated through the notion of executable content.

6.5 Executable Content

Executable content continues the theme of tight integration between multimedia peruser and browser, but with a slight twist. In the case of executable content, the multimedia and the peruser are one. That is, an enabled browser will download the executable files which render the multimedia and execute them as well, all within the browser's own workspace on the client.

The advent of executable content added a high level of animated media rendering and interactive content on the client side. There are many methods

with which to implement data in an executable content paradigm. One example is to simply use the plug-in technology described above. However, this method requires that the user previously download the necessary plug-in. In addition, developers would be required to write and maintain their code on all platforms that they want to support, as mentioned.

A better solution was to provide a standard environment for programming so that programs could be treated as ubiquitously as HTML. While there are several competing paradigms for Web-oriented executable content, including JavaScript, Telescript, and Active-X, the cross-platform language, Java [21, 26], was the clear environment of choice by early 1997. Sun Microsystems created Java so that developers could produce platform-independent modules that are executable on enabled Web browsers. A Java program run on the Web is called an applet.

```
public class HelloWorld extends java.applet.Applet {
     public void paint(java.awt.Graphics g) {
          g.drawString("Hello World!");
     }
}
```

This applet is a sample program written in Java and stored with the filename `HelloWorld.java`. It displays the standard greeting.

The development and use of a Java Applet runs as follows:

(1) A programmer writes an object-oriented program (ending with `java`; see above).
(2) A compiler (Sun's version is written in Java) translates the source code into Java byte-code (also called a class file as they end with `.class`).
(3) A Java virtual machine (JVM), usually located in a browser, interprets the byte-code into machine-specific commands.

Java has probably sparked more activity in computer science than the Web itself. In one important sense, Java is a computer. The Java language has commands for performing normal operations, but also includes methods for rendering graphical objects. The concept of "Java" actually has three, very different parts. These are: the JVM, the Java syntax, and the Java byte code compiler. Currently, all of these are tightly integrated, but are slowly becoming three distinct parts. For example, any language could (theoretically) be compiled for the JVM. Programmers could continue to write in (a subset) of C++ or COBOL, and their code would live on running in modern browsers. Also, browsers are utilizing the concept of "just-in-time" (JIT) compilation. This allows the server computer to compile the Java byte-code to native code right after downloading. The Java syntax is also being

compiled directly to native code so that it may act as a "regular" general-purpose language as well.

These issues are further complicated by a huge amount of research in related areas. For example, Sun has recently developed a chip that actually implements the JVM in hardware.

Not surprisingly, this latest extension of the Web, which involves executing foreign programs downloaded across the networks, is not without security risk. However, Java has been designed from the ground up with security issues in mind.

Currently, Java runs in one of two modes: application or applet. In application mode, Java has full access to one's system (printers, drives, monitor, etc.)—there are no security limitations. This allows programmers to write general-purpose applications. Corel has written versions of their popular office suite is such a manner, which allows it to run on any machine with a JVM.

However, most of the interest in Java has been in its applet mode. The designers of Java were sensitive to security issues, and therefore general access to the client's system could not be permitted. Although this greatly restricts what Java applets can do, it is a relatively secure system [13]. For the first time, such applets offer the ability for safe executable content. (Note that Java 1.1 permits a user to allow Java applets to have full or partial control of the client's machine, disks, etc. This greatly enhances the power of applets, but creates a more complex security control system for the user.)

Java has been the most successful executable content language, but there are many others. Tcl [17, 24] (pronounced "tickle") has also become a popular scripting language for executable content, but does require a plug-in in order to operate as it is not built into the current browsers. Tcl was "retro-fitted" for Web use with security features (the so-called "Safe Tcl" version). Tcl, like Java, supports basic programming functions as well as a graphical user interface (GUI).

Probably the second most successful technology for executable content is Microsoft's Active-X. Active-X is not a language, but rather a method for dynamically downloading and using shared libraries. Active-X is generally a Windows-only solution that requires programmers to build the executable content under that OS. However, a third party developer, Ncompass (`www.ncompass.co.uk`) has developed a method for Unix systems running Netscape's Navigator to execute Active-X executables. This technique shows Active-X's similarity to plug-ins, as NCompass has been implemented as such.

However, unlike Java, Active-X has no security model. An Active-X program in execution has the full range of OS functions available to it as would any typical application [13]. This illustrates the tradeoff between

providing applets with greater power and flexibility on the one hand and providing the host with greater security on the other. Java and Active-X take different approaches to this trade-off.

To minimize the potentially disastrous security problem with Active-X, Microsoft attempts to deal with the security issue with "certification." Certification is the formal procedure of listing your company and programs with an official registrar (Verisign is one such company). Internet Explorer (the only current browser supporting Active-X) will only allow Active-X applets that have been certified and accepted by the user to be downloaded and executed (Java 1.1 applets may also be certified). Of course, there is nothing to prevent a programmer from doing devious things after their Active-X control has been installed on a machine once it has been certified. Certification does not prevent evil-doers, but at least one knows where the harm came from afterwards. Because of these properties, Active-X is really only a choice for Windows-only intranets where clients can faithfully trust their servers.

As can be seen, browsers are a complex environment supporting many kinds of interacting scripts, plug-ins, etc. As of early 1998, many security problems were still being reported in the media.

6.6 Programming

Java can also be used as a server scripting language (such a server side program is sometimes called a "servlet"). Usually, one wants server code to be as small and fast as possible, and Java doesn't currently meet these criteria. However, it makes a good prototyping language as it has many built-in functions that make it extremely useful for Web-based server applications (see source code, below). When there are native compilers (i.e. compilers that produce machine code rather than byte code) for server machines, Java will not only be the best prototyping language for these applications, but will be the best Rapid Application Development (RAD) tool as well.

The following is a very small (insecure) Web server written in Java (after Niemeyer and Peck [15]). This program runs on a server and retrieves HTML files to standard browsers. Warning: this little server can serve up any file on your computer, so don't use it on a machine containing sensitive data.

```
********************** File Httpdaemon.java
import java.net.*;
import java.io.*;
import java.util.*;
public class Httpdaemon {
  public static void main (String argv[] ) throws
IOException {
```

```
  ServerSocket ss = new
                ServerSocket(Integer.parseInt(argv[0]));
  while (true) {
    new HttpdConnection( ss.accept() );
   }
 }
}
class HttpdConnection extends Thread {
  Socket sock;
  HttpdConnection(Socket s) { //constructor
    sock = s; setPriority(NORM_PRIORITY - 1); start();
  }

  public void run() {
    try {
      OutputStream out = sock.getOutputStream();
      String req = new
DataInputStream(sock.getInputStream()).readLine();
      System.out.println("Request: " + req);
      StringTokenizer st = new StringTokenizer( req);
      if ( (st.countTokens() >= 2) &&
st.nextToken().equals("GET")) {
      if ( (req = st.nextToken()).startsWith("/") )
           req = req.substring(1); if ( req.endsWith("/")
           || req.equals("") )
           req = req + "index.html";
      try {
           FileInputStream fis = new FileInputStream
( req); byte [] data = new byte [fis.available() ];
fis.read( data ); out.write( data);
        } catch (FileNotFoundException e) {
            new PrintStream(out).println("404 Not Found");
        }
      } else new PrintStream(out).println("400 Bad
Request");
      sock.close();
    } catch (IOException e) {
    System.out.println("I/O error " + e);
    }
  }
}
```

6.7 DHTML

The idea of Dynamic HTML is to expose the set of events that allows a Webmaster to program a page to respond to many common interactions

between the user and the document. The proposed Dynamic HTML event model is based on two powerful features for controlling the document's behavior: event bubbling and default actions. Whenever the user interacts with the page an event is fired. The user generates events in the standard ways: moving the mouse, clicking a mouse button, or typing on the keyboard within a document. Changes in document state can also fire events include the loading of the document, images, or objects.

DHTML is similar to JavaScript, except DHTML is even less stable, neither Microsoft nor Netscape support it, and neither Microsoft nor Netscape have determined exactly how it should work. Future browser support for DHTML is unknown [12].

6.8 Server-Side Includes

Server-Side Includes (SSI) are another method of dynamically creating webpage content. SSIs are commands which are parsed and processed by the web server. SSIs may be the easiest method for the Web master to create dynamic Web pages. Consider this simple HTML code:

```
<HTML>
<BODY>
<!--#echo var="LAST_MODIFIED" -->
</BODY>
</HTML>
```

To automatically add a "last modified" date to a page requires a simple `#echo` SSI command as shown above. This Web page would display text in the form of "`Monday, 15-Jun-98 03:09:31 CDT`." Notice that the SSI command is sandwiched between `<!-- and-->`. These HTML tags signal a comment, and therefore, are ignored by browsers if the server does not support SSI.

SSIs are useful for several reasons:

- they create easy to maintain code; all commands stay in the HTML files;
- they help create structured, uniform sites;
- they are easy for non-programmers to use.

An SSI is composed of four main commands, `#config`, `#echo`, `#include`, and `#exec`, which allow a Web master to config settings, display information, include source files, and execute programs respectively. For example, one could include a standard footer with the SSI `<!--#include virtual="footer.txt" -->`.

TABLE III
SUMMARY OF SCRIPTING LANGUAGES (AFTER LAIRD AND SORAIZ [12])

Scripting language	Advantages	Disadvantages	Source
JavaScript	Standard in browsers	Only works in browsers	http://developer.netscape.com/one/javascript
MetaCard	Easy for nonprogrammers to learn	Very small customer base	http://www.metacard.com
Perl	Widely used and dominant in CGI; specialized extensions are available	GUI, Windows, Mac OS maintenance given less attention	http://www.perl.org
Python	A clean, portable, maintainable language	Base of Python expertise still small	http://www.python.org
Rexx	Available for well integrated with all IBM OSes, including mainframes	Impoverished library of facilities compared to Perl, Python, and Tcl	http://www.rexxla.org
Tcl	Simple syntax, easily learned, extensible	Clumsy for arithmetic and some other operations	http://tclconsortium.org
VBScript	Resembles Visual Basic	Single source; useful only with Microsoft Web products	http://www.microsoft.com/scripting/vbscript/default.htm

However, the SSI directives have recently been further enhanced. SSI now includes: `#set`, `#if`, `#else`, and `#endif`. This creates a powerful programming environment for creating dynamic HTML pages as the program below illustrates. This is a sample HTML page which uses the environment variable `HTTP_USER_AGENT` to dynamically determine what text to display.

```
<HTML>
<!-- #if expr="$HTTP_USER_AGENT = /^Mozilla/" -->
You are using a Netscape Browser
<!-- #else -->
You are using another browser
<!-- #endif -->
</HTML>
```

6.9 Push Technologies

An interesting technology "about face" occurred in the mid-1990s. The Web witnessed the deployment of information acquisition tools which went beyond the original "information-pull" concept behind the Web. "Push technology" or "push-phase technology" [4] found wide use in 1996–97—to the regret of many MIS managers who watched their corporate bandwidth wither away under the strain! In its most basic form, push technology is an automated delivery environment which produces timely downloads of information without end-user involvement. In this capacity, it is seen as the most recent offshoot of an evolutionary path which begins with telegraphy.

Early wire services (e.g. Associated Press, Reuters) were also "pushy" in the same sense as modern push clients. Both distribute information automatically without requiring end-user requests. Email is pushy in this regard as well—one : one in the interpersonal case, and one : many when done with distribution lists and alias files. Television and radio are also pushy. Like email, Web push is inherently digital and network-based, and like television and radio and the wire services, it supports a wide range of broadcasting and applications. The most recent incarnation of push technology is a close descendent of the "server push" concept developed by Netscape in 1995. The principle behind this Web extension was "dynamic updating". It was thought that there were likely to be many situations in which it would be desirable to continuously update Web browser windows with volatile information. Over the past few years, server-push has been used to produce multi-cell animations, slide shows, "ticker tapes", automatic pass-through of Web splash pages, and so forth. Dynamic updating was a way of overcoming the "stateless" protocol of the Web, which disconnects the client-server connection immediately after each transaction cycle, as previously described.

Actually, server-push was just one-half of Netscape's dynamic updating duo. The other half was client-pull. Server-push refreshed information displayed on the client through pre-determined, timed, server-initiated transmissions of HTML documents. However, this approach is server-invasive, requiring special server-side executables to create and deliver the refresh stream, and accordingly server push has fallen into disuse (a "deprecated feature", in Web terminology).

Client-pull, on the other hand, remains in use within the Netscape community for the display of constantly-updated HTML pages. Unlike server-push, client-pull requires no special programs to operate. The Web browser client initiates an HTTP connection and request for information from a server when it sees a particular token of the `<META>` tag in an HTML document. To illustrate, the tag

```
<META http-equiv="refresh" content="5;url=
                                    http://www.widget.com">
```

would cause a pull-compliant browser to refresh the current browser window with the document at `http://www.widget.com` five seconds after loading the current page. Without a URL specified, the browser will refresh itself with a reload of the current page. The "pull" is shut off as soon as a document is reached which does not have a refresh `<META>` tag.

For both server-push and client-pull, the idea is a simple one: provide data downloads without requiring user intervention. However, early server-push and client-pull technologies were deficient in one major respect: they were both context- and content-insensitive. That is, all accesses to a URL—whether pushed or pulled—produced the same results for all users at any given moment in time. This context/content insensitivity became the bête noir of Netscape's dynamic updating technology because it produced an information access and delivery system that wasn't scalable—the delivery of numerous, complex, timely, and personalized documents require as many URLs as there are documents. In order to minimize information overload, some mechanism needed to be created to build the content and context sensitivity into the push technology, itself.

Current push client-server environments (see Table, IV) have many characteristics in common. For one, they are set up to distribute both internal (to the vendor) and third party information. Second, most push environments require proprietary clients which operate independently of any Web browser. Third, they mainly rely on a "client-polling" (or "smart pulling") model of triggering downloads (counterintuitive as it seems, most "push" environments are not technically push at all, but are called push because the client-polling is user-transparent).

TABLE IV

SELECTED PUSH TECHNOLOGY VENDORS (PROGRAMS)—CLIENT-SIDE ONLY

BackWeb	www.backweb.com
Global Village (NewsCatcher)	www.globalvillag.com
inCommon (Downtown)	www.incommon.com
Intelliserv	www.verity.com
Intermind (Communicator)	www.intermind.com
Lanacom (Headliner)	www.lanacom.com (now part of BackWeb)
NewsEDGE	www.newsedge.com
Pointcast	www.pointcast.com
Marimba (Castanet)	www.marimba.com
Wayfarer (Incisa)	www.wayfarer.com

However, push clients also differ in some respects. For one thing, some produce revenue by subscription while others (e.g. Pointcast) achieve revenue through advertising. Though most of the clients are built around Java and HTML, some rely on Microsoft's Channel Definition Format and XML. Some support SQL database interface, where others support some mix of Common Gateway Interface (CGI), SQL, Open Database Connectivity, and so forth. All environments vary with respect to the type and variety of end-user filtering tools.

Though no silver bullet, modern push technology holds out promise of several advantages:

(1) automatic downloads—in some cases "differential downloading" which only downloads the files that have changed;
(2) automated announcements of updated content;
(3) coherent information streaming via content channels;
(4) delivery and rendering independence from browser;
(5) automated but interactive Web document management;
(6) managed delivery;
(7) server-side information filtering and screening.

Currently, hundreds, if not thousands, of media-rich (if not content-rich) channels are available for push technology use. That these channels provide useful information to some end-user communities is beyond dispute. What remains to be seen is whether future push development will develop end-user controls and filters adequate to the challenge of accommodating each information consumer's personal bandwidth.

6.10 State Parameters

A method of keeping track of state in the client-server communication is

by passing messages via additional arguments. For example, one can pass additional parameters to a CGI program via the URL by placing a question mark followed by a list of messages (strings) which are separated by plus signs:

```
dangermouse.uark.edu/samples/params.cgi
                                    ?hithere+whatsup?+buddy
```

The Perl code below would give the following results:

```
hithere
whatsup\?
buddy
```

which are stored in the normal command line variables (`@ARGV` in Perl's case). Notice that the second question mark is treated as data. The program below, `params.cgi`, is a Perl [22, 23] script, which illustrates how one could keep track of state via additions to the URL and environment variables.

```
#! /usr/bin/perl
print "Content-type: text/html\n\n";
print "<html>\n";
print "<head>\n";

print "$ARGV[0]<br>\n";
print "$ARGV[1]<br>\n";
print "$ARGV[2]<br>\n";
print "<body></html>\n"
```

Another method allowing client and server to communicate information such as state is the use of hidden fields. This method uses fields in a Web form. As stated, a form is Web page that accepts user input. Input can be given in text boxes, check boxes, or button selects. There can be many forms per Web page. Here is a section of a Web page that defines a form and two input fields: id and password:

```
<form method=post>
Enter ID: <input name="id"> <br>
Enter password: <input type=password name="password"> <br>
<input type=hidden name="date" value="01/06/1999"> <br>
<input type="submit" name="continue">
</form>
```

There are actually four types of fields on this form: a text box (id), a password-type text box such that text typed appears as asterisks (password), a hidden field (date), and a button (continue). All field values are stored in operating system environment variables which can then be accessed by CGI programs. For example, the program displayed above would create three environment variables named id, password, and date.

As with most new technology, we trade a little privacy for ease of use. This will probably always be the case on the Internet.

7. Security and Privacy

To the average user, the Internet and its protocols seem very safe. However, this is far from the case. The Internet was set up in a cooperative environment, and still remains so to this day. For example, IP relies on servers all over the world acting as a giant bucket-brigade to pass message packets around from place to place. There is barely anything except ethics keeping advanced users from "peeking" at messages as they go by one's server on their way to other destinations.

Two technologies that have emerged to fix this major security hole are the Secure Socket Layer (SSL) and S-HTTP.

7.1 Secure Socket Layer

The Secure Socket Layer is a security protocol that sits on top of TCP/IP to prevent eavesdropping, tampering, or message forgery over the Internet. The latest version of the SSL protocol (Version 3.0) has been submitted to the IETF and is available as an Internet Draft.

7.2 Secure HTTP (S-HTTP)

Secure HTTP was developed by Enterprise Integration Technologies (`www.eit.com`) to keep commercial transactions protected. S-HTTP is a secure protocol over HTTP for identification when entering into a server. Both the server and the client identify each other using a public key system. S-HTTP encrypts certain pages that flow between the Web server and the client. This encryption is usually only done to pages that contain sensitive information, such as a credit card number. That means that if anyone attempts packet sniffing or eavesdropping in any way, the intruder will see only the encoded message.

For the Web to become a medium for commonplace commerce, security and privacy must become seemlessly integrated into it. Currently, it takes extra effort for Web users and servers to insure they are communicating over a secure line. When secure communication is the default, we expect on-line electronic transactions to be the most common method of paying bills, buying products, and performing most other types of commercial interactions.

7.3 Cookies

A cookie is another technique for allowing clients and servers to keep track of state. Cookies were created by Netscape corporation. The program below is a Perl script that attempts to set a cookie on any client that reads the Web page (given that the client's browser is cookie-capable, and that the user accepts it). The cookie will automatically be retrieved from the client's machine and stored in the `HTTP_COOKIE` environment variable whenever a page is accessed from the `DOMAIN` specified in the `META` tag.

```
#! /usr/bin/perl
print "Content-type: text/html\n\n";
print "<html>\n";
print "<head>\n";
print "<META HTTP-EQUIV=\"Set-Cookie\" ";
print " CONTENT=\"CODENAME=Smith";
print " EXPIRES=Mon, 01-06-99 12:00:00 GMT;";
print " PATH=/;DOMAIN=.uark.edu\">\n";
print "<body><h1>";

print $ENV{'HTTP_COOKIE'};
print "<br>\n";
print $ENV{'SERVER_NAME'};
print "<br>\n";

print "</body></html>\n"
```

Cookies are composed of two parts: the field name, and the field data. In this example, the field name is `CODENAME` and the data is "`Smith`." Whenever a page is accessed at the `.uark.edu` domain, the `HTTP_COOKIE` environment variable will contain "`CODENAME=Smith`" and possibly other cookie field/data pairs, separated by semicolons. This particular cookie will expire on 1 June 1999 as specified in the `EXPIRES` clause. Until then, the cookie's information will be stored on the user's disk. Netscape's Navigator stores cookie information in a file called "`cookies`"—e.g. `\Programs\Netscape\Navigator\cookies.txt` in the case of Netscape running under Windows. After accessing the above Web page, there would be an entry in the cookie file similar to:

```
.uark.edu TRUE / FALSE 915624000 NAME Smith
```

If the `EXPIRES` clause is not given, the cookie will expire when the browser is exited and will never be stored on disk.

There is an additional keyword that may appear in the `META` clause for use with Netscape browsers: `SECURE`. If the `SECURE` keyword appears, then the Netscape client will only send the field/data pair back if the client and server

are communicating over a secure protocol, such as the Secure Socket Layer (SSL). Of course, the field/data pair could be encoded on the server, thereby protecting sensitive information even without SSL.

Because of privacy issues, the use of cookies continues to be hotly debated. According to the most recent Web survey [18] only 22% of the Web users surveyed accept cookies from any source, while 23% receive a warning before cookies are set, allowing them to make a decision on a case-by-case, and approximately 6% refuse all cookies. Surprisingly, the remaining survey participants either didn't know what a cookie was or didn't care about having a policy.

Cookies can help make the Web experience more useful by recording information on individual network transactions, which can streamline subsequent interactions. Despite this advantage, many remain concerned about the potential loss of privacy by allowing Web servers to "track" them.

Factors which mitigate against privacy concerns include:

(1) Only the domain that sets a cookie can read that cookie.
(2) Sites can only set cookies for their domain.
(3) All of the information that is stored in the cookie on the client's machine could easily be stored on the server.
(4) There is a hard limit to how many cookie field/data pairs a client can store (Netscape's Navigator can hold 300 total) and a limit per domain (20 for Navigator). This guards against "denial of services" attacks so that wayward servers can't gobble up a client's diskspace by giving it too many cookies to eat.
(5) Cookie data is typically encoded or is composed of a random site-based database key.

Given these points, one realizes that cookies might be the lesser of two evils: having the client store tracking data or having the server store tracking data. If the server keeps tabs on the browsers, the end-user might never know. Of course, the volume of data collected by highly-active servers would discourage unrefined cookie collection. Since servers do keep tabs of network transactions (e.g. in the form of values of environment variables) perhaps not much privacy is breached.

Here is a typical entry that was generated when the above Web page was accessed (from the Apache Web server's default settings for logging):

```
dangermouse.uark.edu - - [31/May/1998:22:01:19 -0500] "GET
/samples/hw.cgi HTTP/1.0" 200 194
```

If the client stores the data, one could change it, delete it, and generally be aware of what sites are openly keeping track of state. Anonymous

browsing is impossible, although there are some steps one can take to be less obvious:

(1) Use an "anonymizer". An anonymizer is a "go-between" web site. A user asks the anonymizer to retrieve a Web page, the anonymizer does so, and returns the HTML data back to the user. Unfortunately, one must trust the anonymizer, for it knows who is accessing what pages.
(2) Physically move to a different computer. This prevents Web sites from having accurate data. Of course, this is not always a possibility.
(3) Use dynamically assigned IP numbers.

Although cookies cannot get to data on your hard drive, they can be used to track you. For example, DoubleClick Corporation (`http://www.doubleclick.com/`) uses cookies to keep track of your browsing from many thousands of Web sites. This data can then be sold to marketing companies. Whenever a DoubleClick ad appears, a request is made from their site for the advertising graphic. Doing so not only retrieves the image, but allows them to check your cookies, and assign you an ID if you haven't been to a DoubleClick-sponsored site before. Afterward, whenever you go to a site with a DoubleClick ad, they can keep statistics on your visits.

The latest versions of Netscape Navigator provide a method of refusing cookies that are issued from sites other than the one that you currently are viewing. This will foil most global tracking schemes (such as DoubleClick's) without interfering with the more useful cookies that keep track of local state (i.e. for shopping carts). Of course, one could also simply delete the cookie file thus throwing a monkey wrench into the usefulness of the cookie technology. (On systems supporting symbolic links, one can simply `ln -s /dev/null` cookies and cookies are automatically sent to the void.)

In addition to server logs, security issues also include the client machine. One should consider that typical modern browsers leave a trail of exactly what has been viewed over the Internet. Such evidence can be found in a browser's cache, a directory of temporary files documenting every graphic and HTML page viewed.

The interactions between all of the types of scripting and dynamic Web technologies is still not well defined. For example, the code below is a JavaScript program that discovers a visitor's IP address. Although this information can also be found in a CGI's environment variables, as well as a server's access log files, if JavaScript could access more personal data (such as the user's email address) and give that to the server, then privacy is further lost.

```
<html>
<head>
<title>Get IP Address</title>
</head>
<HR>Getting IP Address...</HR>
<SCRIPT LANGUAGE="JavaScript">
<!--
function getIP() {
  if (navigator.javaEnabled()) {
      baseAddress = java.net.InetAddress.getLocalHost()
userDomain = baseAddress.getHostName() return
(userDomain.toString())
  } else {
      return null
  }
}
domainName = getIP()
alert('You are ' + domainName + ' and I now know it!')
//-->
</SCRIPT>
</body>
</html>
```

8. The Web as a Social Phenomenon

The social effect of the Web is not well understood. Not surprisingly, the zeal to harness and exploit the richness of Web resources and technology, combined with the desire to capitalize on commercial Web services, have taken precedence over efforts to understand the social dimensions of Web use.

Much of what little we know of Web behavior seems to be derived from two disparate sources. Descriptive statistics produced by the Web surveys are most useful to measure isolated events and independent activities—e.g. how many Windows users use Netscape.

The second source is the study of the use of email. Email's status as a de facto paradigm of "interpersonal though not-in-person communication" makes it a useful testbench for testing hypotheses about network behavior, generally. Since email and the Web share several characteristics, e.g. they both minimize the effects of geographical distance between users, they are both based on user-centric models of communication, both rely on self-imposed interrupts, both are paperless and archivable by default, both create potential security and privacy problems, and neither requires continuous endpoint-to-endpoint network connectivity, email can teach us something about Web behavior.

However, both sources provide incomplete views of Web behavior. Descriptive statistics tell us little about either the causes of emerging trends or the connections and associations between various aspects of Web use (e.g. to what extent, if any, do anonymous Web engagements promote discussion of controversial topics?).

There are differences between email and the Web as well. Email deals with network, peer-to-peer communication partnerships, where the present Web remains primarily an information-delivery system. Email, in its most basic form at least, exemplifies push-phase technology, while the current Web is mostly pull-phase in orientation. Of course, the onset of new technologies such as Web teleconferencing and virtual communities, will change the nature of such comparisons.

While definitive conclusions about the social aspects of Web use remain elusive, some central issues have been identified for future study (see Table V). We are slowly coming to understand the capabilities of the Web for selected applications and venues. To illustrate, early use convincingly demonstrated that the Web was a popular and worthwhile medium for presenting distributed multimedia, even though we cannot as yet quantify the social benefits and institutional costs which result from this use. As CGI was added to the Web, it became clear that the Web would be an important location-independent, multimodal form of interactivity—although we know little about the motivations behind such interactivity, and even less about how one would measure the long-term utility for the participants and their institutions.

8.1 Virtual Communities

As mentioned above, the Web's primary utility at the moment is as an information delivery device—what some authors have called the "document

TABLE V

SOCIAL ISSUES AND WEB BEHAVIOR

- How central is location transparency to Web use? To what extent will Web "communities" be used to replace or enhance veridical counterparts?
- How will future Web technologies deal with information overload?
- To what extent will interactive and participatory Web engagement become enticing and immersive?
- What are the benefits and weaknesses of anonymous engagement and relative identity environments. Will relative identity havens create new problems for law enforcement?
- To what extent will Web engagement enhance or supplement alternative modes of information exchange?
- What technologies will emerge to reduce Web transaction friction?
- etc.

TABLE VI

POTENTIAL ADVANTAGES AND DISADVANTAGES OF ELECTRONIC COMMUNITIES

Advantages	Disadvantages
• potential for dynamic involvement where membership may be transitory and the infrastructure of the community informally defined	• quality of experience may not justify the participation, or may degrade over time
• location transparency for members, as all electronic communities are potentially global	• potential loss of privacy by invasive Web technologies such as global tracking via cookies, CGI environment variable recording, and the like
• capability of self-administration and self-organization by a membership in continuous flux	• some forms of electronic communication lack intensity, and some may lack content (e.g. more information exchange doesn't imply better information exchange)
• creation of "thought swarms" through the continuous, interactive stimulation of participants	• not all experiences translate well into the electronic realm, as documented by the easy
• increased attention on content	• misinterpretation of email and the "flaming" that can ensue

phase" of the Web. However, more powerful and robust Web applications will soon begin to take hold. Perhaps the most significant future application will involve the construction of virtual communities.

Virtual, or electronic, communities, are examples of interactive and participatory forums conducted over digital networks for the mutual benefit of the participants and sponsors. They may take on any number of forms. The first attempts to establish virtual communities dates back to the mid-1980s with the community, "freenet" movement. While early freenets offered few services beyond email and Telnet, many quickly expanded to offer access to documents in local libraries and government offices, Internet relay chats, community bulletin boards, and so forth, thereby giving participants an enhanced sense of community through another form of connectivity.

Virtual communities of the future are likely to have both advantages and disadvantages when compared to their veridical counterparts (Table VI).

9. Conclusion

The World Wide Web represents the closest technology to the ideal of a completely distributed network environment for multiform communication. As such, it may be thought of as a paradigm shift away from earlier network protocols.

Many feel that the most significant impact of the Web will not be felt until the 21st century, when technologies are added and converged to make the Web a fully interactive, participatory, and immersive medium by default.

Security and privacy will undoubtedly continue to be important issues as new methods are integrated into current Web technologies. The complexity of interacting components is nearly out of hand at this point; future Web protocols may help create the appropriate structure necessary for continued robust Web development.

The world will continue to become a smaller place as cultures continue to be only a click away. The Web promises to have one of the largest impacts on general society of any technology thus far created. For a more thoroughgoing analysis of the Web's future by its founder, see references [6] and [7].

ACKNOWLEDGMENTS

We wish to thank Marvin Zelkowitz and anonymous reviewers for their comments on earlier drafts of this chapter.

REFERENCES

[1] ACM Electronic Communities Project, `http://www.acm.org/~ccp/` (information on the use of the Web for Electronic Communities).

[2] Berghel, H. (1998). The client side of the World Wide Web. *Encyclopedia of Computer Science* 4th edn (ed. by Anthony Ralston, Edwin Reilly, and David Hemmendinger). Petrocelli.

[3] Berghel, H. (1998). Who won the mosaic war? *Communications of the ACM*, October.

[4] Berghel, H. (1998). Push technology. *Networker*, **2**(3), 28–36, ACM Press.

[5] Berghel, H. (1997). Email: the good, the bad and the ugly. *Communications of the ACM*, **40**(4), 11–15.

[6] Berners-Lee, T. (1996). WWW: Past, present and future. *Computer*, **29**(10), 69–77.

[7] Berners-Lee, T., Cailliau, R., Luotonen, A., Nielsen, H., and Secret, A. (1994). The World Wide Web. *Communications of the ACM*, **37**(8), 76–82.

[8] Bush, Vannevar (1945). As we may think. *Atlantic Monthly*, July. Online at `www.isg.sfu.ca/~duchier/misc/vbush/`).

[9] Comer, D. (1997). *The Internet Book: Everything You Need to Know About Computer Networking and How the Internet Works*. Prentice-Hall, Inc., Upper Saddle River, NJ.

[10] Englebart, D. (1986). The augmented knowledge workshop. *The History of Personal Workstations: Proceedings of the ACM Conference*, (ed. K. Anderson). ACM Press, New York, pp. 73–83.

[11] Flanagan, D. (1996). *Java in a Nutshell*. O'Reilly & Associates, Inc. Sebastopol, CA.

[12] Laird C., and Soraiz, K. (1998). Get a grip on scripts. *BYTE*, June. The McGraw-Hill Companies, Inc. New York, NY.

[13] McGraw, G., and Felten, E. W. (1997). *Java Security: Hostile Applets, Holes, and Antidotes*. John Wiley and Sons, Inc.

[14] Nelson, T. (1995). The heart of connection: hypermedia unified by transclusion, *Communications of the ACM*, **38**(8), 31–33.

[15] Niemeyer, P., and Peck, J. (1966). Exploring Java. O'Reilly & Associates, Inc. Sebastopol, CA.
[16] NSFNET Backbone Traffic Distribution Statistics, April, 1995. http://www.cc.gatech.edu/gvu/stats/NSF/merit.html.
[17] Ousterhout, J. (1994). Tcl and the Tk Toolkit. Addison-Wesley Publishing Company. Reading, MA.
[18] Pitkow, J., *et al.*, GVU's WWW User Surveys, http://www.cc.gatech.edu/gvu/user_surveys/.
[19] Quercia, V. (1997). Internet in a Nutshell. O'Reilly & Associates, Inc. Sebastopol, CA.
[20] Roberts, D. (1996). Internet Protocols. Coriolis Books, Scottsdale, AZ.
[21] Sun Microsystems (1995). *The Java Language: a White Paper.* Web document. URL http://java.sun.com.
[22] *The Perl Journal.* (http://tpj.com) Readable Publications, Inc. Somerville, MA.
[23] Wall, L., and Schwatz, R. (1991). *Programming Perl.* O'Reilly & Associates, Inc. Sebastopol, CA.
[24] Welch, B. (1997). Practical Programming in Tcl and Tk. Prentice Hall, Inc. Upper Saddle River, NJ.
[25] WWW Security FAQ http://cip.physik.uni-wuerzburg.de/www-security/wwwsf6.html
[26] Java for 1998. *PC Magazine*, 7 April 1998.