

PRE-FETCH DOCUMENT CACHING TO IMPROVE  
WORLD-WIDE WEB USER RESPONSE TIME

by

David Chunglin Lee

Thesis submitted to the Faculty of the

Virginia Polytechnic Institute and State University

in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE


in

Electrical Engineering

APPROVED:

  
S.F. Midkiff, Chairman

  
W.J. Davis, IV

  
C.E. Nunnally

March, 1996  
Blacksburg, Virginia

Key Words: World-Wide Web, Pre-Fetch Caching

LD  
5655  
V855  
1996  
44  
C.2

# PRE-FETCH DOCUMENT CACHING TO IMPROVE WORLD-WIDE WEB USER RESPONSE TIME

by

David Chunglin Lee

Scott F. Midkiff

Electrical Engineering

(ABSTRACT)

The World-Wide Web, or the Web, is currently one of the most highly used network services. Because of this, improvements and new technologies are rapidly being developed and deployed. One important area of study is improving user response time through the use of caching mechanisms. Most prior work considered multiple user caches running on cache relay systems. These systems are mostly post-caching systems; they perform no “look ahead,” or pre-fetch, functions. This research studies a pre-fetch caching scheme based on Web server access statistics. The scheme employs a least-recently used replacement policy and allows for multiple simultaneous document retrievals to occur. The scheme is based on a combined statistical and locality of reference model associated with the links in hypertext systems. Results show that cache hit rates are doubled over schemes that use only post-caching and are mixed for user response time improvements. The conclusion is that pre-fetch caching Web documents offers an improvement over post-caching methods and should be studied in detail for both single user and multiple user systems.

## Dedication

To my parents, Kun-chieh and Shiow Yeh Lee, who made this work possible.

## Acknowledgments

To Dr. Scott F. Midkiff for keeping me on track, providing excellent advice, and being the best possible advisor.

To Dr. Nat J. Davis, IV for statistical help and questions, and Dr. Charles E. Nunnally for serving on my committee.

To my parents, Kun-chieh and Shiow Yeh Lee, and my brother, Edward "Ted" C. Lee, for all the encouragement and support through the years.

To Rhett D. Hudson for his suggestions and help. To Salah Almajdoub for all the Microsoft Word tricks. To Al "Snowman" Walters and Mark "Grover" Musgrove for providing incentive to get this done. To Yen Chuang and David Putzolu for all the good games.

To my volunteer testers: Salah Almajdoub, Ray Bittner, Henry "G" Green, John Lund, Barry Mullins, Mark "Grover" Musgrove, Rhett Hudson, Tyler Ormsby, Kevin "Bumble" Paar, Jim Peterson, and Elvin Taylor for providing the user trace data that this research work is based on.

To Jeff Vest and Ping Fang for providing statistical consulting.

To Ed Schwab for helping collect some statistics on the BEV World-Wide Web server.

To Bruce Harper for supplying the Virginia Tech server access logs.

To the National Science Foundation for graduate assistantship funding by SUCCEED (Cooperative Agreement No. EID-9109853). SUCCEED is a coalition of eight schools and colleges working to enhance engineering education for the twenty-first century.

# Table of Contents

<b>CHAPTER 1. INTRODUCTION.....</b>	<b>1</b>
1.1 MOTIVATION.....	1
1.2 CACHING PROBLEM.....	2
1.3 THESIS ORGANIZATION.....	5
<b>CHAPTER 2. BACKGROUND .....</b>	<b>8</b>
2.1 THE WORLD-WIDE WEB.....	8
2.2 GLOBAL NAMING SCHEMES.....	9
2.3 HTML.....	11
2.4 HTTP.....	13
2.5 CURRENT ISSUES.....	16
2.6 SUMMARY.....	17
<b>CHAPTER 3. USER RESPONSE TIME IMPROVEMENT TECHNIQUES .....</b>	<b>18</b>
3.1 DEFINITIONS.....	18
3.2 BASIC PERFORMANCE ISSUES .....	18
3.3 CONSTRAINTS ON INCREASING PERFORMANCE.....	20
3.4 PROTOCOL (HTTP) REPLACEMENT .....	21
3.5 DATA COMPRESSION.....	24
3.6 MULTIPLE PROTOCOL INSTANCES.....	26
3.7 CACHING.....	26
3.8 SUMMARY.....	27
<b>CHAPTER 4. CACHING .....</b>	<b>28</b>
4.1 BASIC CONCEPTS .....	28

4.1.1 *User Models*.....29

4.1.2 *Cache Relaying Versus Caching* .....30

4.1.3 *Pre-fetch Caching*.....31

4.2 HYPERTEXT ACCESS GRAPHS .....33

4.3 PRE-FETCH DOCUMENT PREDICTION .....34

4.3.1 *Categorical Heuristics*.....34

4.3.2 *Statistical Heuristics*.....35

4.3.3 *Prediction Involving Multiple Servers*.....37

4.4 POST-FETCH DOCUMENT PREDICTION .....38

4.5 CACHE HIERARCHY.....38

4.6 SUMMARY .....40

**CHAPTER 5. DESIGN AND IMPLEMENTATION .....41**

5.1 DESIGN GOALS.....42

5.2 CLIENT-SERVER INTERACTION.....42

5.3 SERVER DESIGN .....45

5.3.1 *Statistical Access Database*.....45

5.3.2 *Server Process Layout and General NCSA httpd Operation*.....46

5.4 CLIENT DESIGN.....47

5.5 PRE-FETCH PREDICTION ALGORITHM .....50

5.6 SUMMARY .....56

**CHAPTER 6. PERFORMANCE .....57**

6.1 EVALUATION METHOD.....57

6.1.1 *User Traces* .....59

6.1.2 *Parameter Set*.....61



6.2 STUDY LIMITATIONS .....63

6.3 PRELIMINARY ANALYSIS.....64

6.4 PERFORMANCE ANALYSIS.....65

6.5 STATISTICAL ANALYSIS .....65

6.6 GENERAL TRENDS AND OBSERVATIONS .....66

    6.6.1 *Effects of Network Bandwidth* .....67

    6.6.2 *Effects of Document Depth*.....68

    6.6.3 *Effects of Number of Channels* .....69

    6.6.4 *Effects of Cache Size and Document Replacement Policy* .....69

    6.6.5 *Combination Effects* .....70

6.7 SUMMARY .....70

**CHAPTER 7. CONCLUSIONS .....72**

**CHAPTER 8. REFERENCES.....75**

**APPENDIX A. USER DOCUMENT CACHING ON THE WORLD-WIDE WEB .....87**

**APPENDIX B. HTTP REDUNDANCY ANALYSIS.....118**

**APPENDIX C. WEB CHARACTERIZATION.....120**

**APPENDIX D. EXPERIMENTAL RESULTS .....139**

**APPENDIX E. STATISTICAL ANALYSIS OF RESULTS.....145**

**VITA.....151**

## List of Figures

FIGURE 1. STATISTICAL ACCESS DIGRAPH EXAMPLE.....	36
FIGURE 2. SERVER PROCESS DIAGRAM.....	46
FIGURE 3. CLIENT PROCESS DIAGRAM.....	49
FIGURE 4. INITIAL WEB GRAPH.....	52
FIGURE 5. WEB GRAPH AFTER RETRIEVING NODE B.....	53
FIGURE 6. WEB GRAPH AFTER RETRIEVING NODE C.....	54
FIGURE 7. WEB GRAPH WHEN USER MOVES TO NODE B. ....	55
FIGURE 8. USER INSTRUCTIONS.....	60

## List of Tables

TABLE 1. SERVERS USED IN FEASIBILITY ANALYSIS .....	32
TABLE 2. USER READING TIME BETWEEN DOCUMENT FETCHES.....	32
TABLE 3. SAMPLE LIST OF HEURISTIC CATEGORIES .....	34
TABLE 4. FACTORIAL BLOCK.....	63
TABLE 5. EXPERIMENTAL RESULTS FACTORS.....	139
TABLE 6. HIT RATIO RESULTS .....	140
TABLE 7. RESPONSE TIME RESULTS.....	141
TABLE 8. TRANSFER RESULTS BASED ON FILES.....	142
TABLE 9. TRANSFER RESULTS BASED ON BYTES .....	144

# Chapter 1. Introduction

## 1.1 Motivation

The most widely used distributed information system today is the World-Wide Web, or the Web. The Web has now become the largest single component of traffic on the NSFNET backbone in both packets (21.4 percent) and bytes (26.5 percent) as of April 1995 [1]<sup>1</sup>. The Web [2, 3] provides a simple point and click hypertext interface that allows a user to request information only when the user desires that information. Given the popularity of the Web, it provides an important platform for study and improvement.

Improved performance and features of distributed information systems will be important research objectives for several years to come. Other systems, such as Hyper-G [4], Gopher [5], and WAIS [6], exist and provide different models of information access and user features. These models can be used to determine features and performance improvements that should and could be added to the Web. These systems are accessible through the Web through the use of hosts that provide gateway interfaces.

There are a number of features that many people desire to see in the Web. Feature improvement is generally an addition to the existing system and based on user preferences; it is usually not an interesting engineering research topic. However, this

---

<sup>1</sup> NFSNET statistics beyond April 1995 are not available due to a change in the NSFNET structure.

research does consider some aspects of feature improvements as they relate to performance improvements.

Performance improvements involve changing the basic system to achieve improved throughput with the overall goal of reducing user response time. In this case, user response time is defined as the time differential between the time the user requests information and the time the information is provided, in its entirety, to the user. User response time can be reduced through various means. Obviously, the best method is to increase network capacity. This is not feasible in many cases, so alternatives have been and are being sought. These methods include protocol modifications, data compression, and document caching. Each of these will be briefly discussed with a focus on user document caching, which is the primary concern of this research.

## ***1.2 Caching Problem***

Web document caching can occur in many forms. A specific user may have his or her own local document cache. A group of users may have a document cache that all users in the group access. This multiple user caching service is typically called cache relaying or proxy caching. Several studies on cache relay schemes have been performed with some beneficial results [7-9]. The focus of this research is to improve user response time through user document caching. In this research, user document caching incorporates both pre-fetch, or “look ahead,” and post-access caching.

The study implements pre-fetch caching with least-recently used (LRU) document replacement. Pre-fetching is performed using statistical information transmitted from the Web server. The algorithm results in approximately 1.6 times the cache hit rate when compared to post-document caching alone. With the LRU document replacement algorithm and reasonable pre-fetch cache sizes, the results are a 40 to 60 percent cache hit rate. A different sample test of pre-fetch caching, by retrieving links a single level away on a first-come, first-serve basis, resulted in an approximately two percent increase in the cache hit rate. This is a fairly limited result, but the implication is that complex pre-fetch algorithms may not be needed. Locality of reference models in pre-fetching algorithms may prove to be a more cost-effective approach.

The increase in cache hit rates does not necessarily translate into reduced user response time. For low network data rates, the user response time decreased significantly. However, for high network data rates, simulation shows that there is a slight increase in user response time. This is attributed to program design.

Luotonen [10] states that user-based caching is not as useful as cache relaying. The primary reason is that multiple copies of the same document are maintained in different user caches. This results in a high disk space cost for limited benefits. While this may be true, typical users, at least those using a low data rate connection, will most likely see a

performance increase through the use of pre-fetch caching. This, for many users, outweighs the cost of extra disk space. Another reason to prefer cache relaying is that a larger base of users provides for better access statistics. These access statistics could be used for document pre-fetching. While this is also true, individual servers have their own set of statistics and it may be more helpful to use those statistics instead of those based on a cache relay user population. This research investigates the feasibility of using server statistics for document pre-fetching.

For this research, one can consider user caching to be equivalent to a single user using a cache relay. This means that the results and techniques for single user pre-fetch caching based on server access statistics can also be applied to multiple user cache relaying.

Another research objective was to obtain analytical data to characterize the Web. These characteristics, presented in Appendix C, include statistical characteristics of Web documents sizes and composition, HTTP data transfer rates, server types, and character frequencies. This work was used to identify and study problems with HTTP. From this, potential methods to improve HTML document transport are suggested.

### **1.3 Thesis Organization**

Chapter 2 provides an overview of the technical features of the Web. Specifically, the mechanisms behind the hypertext links and protocols are addressed. The three core specifications of the Web, HTML, HTTP, and URLs, are reviewed.

Chapter 3 gives background information on methods to improve user response time. Bottlenecks, as well as some solutions, are noted. These bottlenecks and solutions include HTTP limitations, potential new HTTP features such as data compression and multiple protocol instances, and caching strategies. Rationale to support incremental modifications to HTTP, as well as more study into data compression, is presented.

Chapter 4 describes caching concepts and models as well as current caching research. The chapter outlines the theories and ideas used in this research. A limited overview of prediction heuristics and metrics is provided. A new way to look at prediction, which is based on server access statistics, is presented. The concepts discussed are then used to generate a proposed cache hierarchy.

Chapter 5 provides the general design and implementation of the experimental system used in the research. Design goals and limitations are presented. The layout of the server and client process structure is given and an example of the pre-fetch algorithm is presented. The client also has the capability to perform multiple simultaneous requests.

Chapter 6 presents experimental results for the client and server system described in Chapter 5. The experiments examined the effects of four variables, pre-fetch depth, available network bandwidth, cache size, and number of simultaneous requests, against each other and against various caching situations. Results indicate that the proposed system does improve user response time. Observations are made regarding the effects of the variables used in the experiment.

Chapter 7 provides concluding remarks. A summary of important results and future research directions is given.

Appendix A provides the experimental HTTP header specification used in this research. This is presented in draft Request-For-Comment (RFC), or Internet-Draft, format.

Appendix B provides a brief redundancy overhead calculation.

Appendix C shows the results of an informal analysis of Web characteristics. The [www.yahoo.com](http://www.yahoo.com) server indices were recursively traversed to obtain characteristics such as transfer time and document size.

Appendix D contains the data obtained from the experiments described in Chapter 6.



Appendix E provides the statistical results of a  $2^k$ -factorial analysis on the data contained in Appendix D.

## Chapter 2. Background

This chapter provides an overview of relevant concepts related to the Web. These include the mechanism to encode documents in a display-independent manner, the means to transport these documents from machine to machine, and the method to locate these documents on a heterogeneous network. A brief discussion on current Web issues is provided.

### ***2.1 The World-Wide Web***

The World-Wide Web was proposed by Tim Berners-Lee in 1989 as a means to deliver information at CERN [11]. It has been described as “a wide-area hypermedia information retrieval initiative aiming to give universal access to a large universe of documents” [12]. The Web is composed of three primary specifications:

1. a global naming convention through the use of Uniform Resource Locators (URL) (see Section 2.2),
2. a document display language, specifically the Hypertext Markup Language (HTML) (see Section 2.3), and
3. a mechanism to transport and access these documents, the Hypertext Transfer Protocol (HTTP) (see Section 2.4).

In addition, an HTML browser is required to interpret and display the document display language and various types of servers are required to access and transmit, or “serve,” the documents.

A significant amount of Internet, and hence, Web, technical specifications use a meta-language presentation format known as Backus-Naur Form (BNF) [13] and various modified forms such as the augmented BNF used in RFC 822 [14]. This thesis provides simple examples, where applicable, to present general technical details. BNF notation is not used in this text; however, knowledge of BNF may be necessary to review some of the references.

Several of the references are Internet Drafts, which are working documents that are constantly evolving. They may eventually become Request For Comments (RFCs) or may be withdrawn at any time. All such documents referred to were in circulation at the time of writing.

## ***2.2 Global Naming Schemes***

A stable, consistent, and uniform global naming convention of resources on a network is an important aspect of distributed systems that allows for resource sharing [15]. Without a global naming system, the same user at two different sites may need to use two different

names to access the same document. This is, obviously, undesirable on a world-wide network. One name should access the same document regardless of the user location.

There is work within the Internet Engineering Task Force (IETF) to standardize a Uniform Resource Name (URN) that will define name space and resolution protocols for persistent object names [16, 17]. The URN specification is still in the discussion stage. Because of this, the *de facto* Internet resource naming convention is the Web-specified Universal Resource Identifier (URI), as specified in RFC 1630 [18]. It provides for a consistent usage of registered name spaces and protocols to identify resources on the network. The type of URI specification that the Web itself uses is known as a Uniform Resource Locator (URL), as specified in RFC 1738 [19] and RFC 1808 [20]. Additional information on uniform resource naming can be found by searching the Internet Draft and RFC indices.<sup>1</sup>

Some illustrative examples of URL's are given in the following paragraphs.

The URL:

`ftp://www.vt.edu/README`

---

<sup>1</sup> Available through anonymous FTP at `ds.internic.net` in the directories `/internet-drafts` and `/rfc`, respectively.

specifies to use anonymous FTP (File Transfer Protocol) to the host **www.vt.edu** to get the file **README**.

An example URL that specifies an HTTP request to the host **fiddle.ee.vt.edu**, TCP port **80**, to get the file **home.html** in the path **/succeed** is:

**`http://fiddle.ee.vt.edu:80/succeed/home.html`**

Note that the “/” (forward slash) in the URL has special meaning as a token separator and does not necessarily imply a pathname. Special character encoding mechanisms are also available for use.

## **2.3 HTML**

Displaying an electronic document on various types of hardware and display systems is a complex task. The International Organization for Standards (ISO) Standard Generalized Markup Language (SGML) [21] provides a flexible and portable text-based specification that allows the platform-independent creation and display of documents. The current Web implementation uses a subset of SGML called the Hypertext Markup Language (HTML) [22, 23]. HTML Version 3 provides for advanced document layout and formatting, tables, and in-line images. A specific piece of software, commonly called a

browser, is needed to decode the HTML elements and display the document in a suitable format on the local display system. Two of the more common browsers include Netscape Navigator [24] and NCSA Mosaic [25]. Additional browsers may be found at [26, 27].

The decoding and displaying of HTML documents is done through parsing special markup elements. The elements are enclosed in brackets, ‘<’ and ‘>’ (the greater than and less than characters). Anything outside the bracket pairs is considered displayable text. HTML elements either change the display of the text or graphics objects or provide for layout and organization information about the document. For example, to specify text in bold, one would use the ‘B’ and ‘/B’ HTML elements:

**<B>This text is in BOLD.</B> This text is not in bold.**

which would be displayed by a browser as follows:

**This text is in BOLD. This text is not in bold.**

One central feature of HTML is that it allows links to other documents to be embedded within an image and/or text through the use of URL's and the HREF anchor element.

This feature allows a user to jump directly to information that they need instead of reading information in some pre-defined order. This feature creates a set of documents

that is referred to as hypertext. In this research, hypertext links are used to implement pre-fetch based user document caching.

Another important aspect of HTML is that browsers ignore elements that they do not know how to process. This attempts to provide for seamless extensibility in the Web. In most cases, an older version of an HTML browser can still be used to view a newer version of HTML without significant loss of information. Another example is a terminal-based browser that cannot display graphics or different fonts. This terminal would display the above example text (“<B>This text is in BOLD.</B> This text is not in bold.”) as:

This text is in BOLD. This text is not in bold.

There is currently an effort to create a Virtual Reality Modeling Language (VRML) [28] that allows for three-dimensional display and manipulation through the Web.

## **2.4 HTTP**

Transport of HTML could take place through a variety of means: FTP, electronic mail, and news articles. In fact, gateways on the Internet allow transmission of HTML information through such means as noted above. Gateways also allow for the conversion of non-HTML documents into HTML. This still does not provide a good transport

mechanism for HTML. To solve this, a simple stateless protocol was developed to transport HTML from system to system. This protocol is known as the Hypertext Transfer Protocol (HTTP) and is currently an Internet Draft [29,30]. The current version of HTTP is Version 1.1.

HTTP is a simple request and response protocol that uses concepts from the Multipurpose Internet Mail Extensions (MIME) specification [31]. MIME and HTTP provide a named type, called a “content type,” for every document transferred, such as a GIF image, HTML text, or PostScript file. In HTTP, documents are typically not combinations of different document types. Instead, HTML documents typically contain in-line or external links to separate documents.

HTTP provides a mechanism for negotiation of these content types as well as languages and character sets. This, in theory, allows a browser to obtain proper documents for it to display. For example, assume a server has two different graphic representations, using JPEG and GIF, for a particular image and that a user has a browser that can only display GIF images. The protocol allows the browser to tell the server that it can display a GIF image. Based on this, the server can send the GIF image rather than the JPEG image.

This negotiation typically does not occur on the Web today. Most browsers inform the server that it can accept all types of documents and the server sends all types. Rather than



this negotiation mechanism, the user usually manually selects the document types, languages, and character sets to receive.

HTTP allows various actions, or “methods,” to be performed on a document. The most commonly used methods are GET, HEAD, and POST. The GET method simply retrieves the document and header information specified by a URL. The HEAD method is similar to the GET method but only retrieves the header information. The POST method allows a browser to send a document to the server. Other protocol features include basic authentication, document retrieval based on modification dates, proxy serving of documents, redirection, and expiration of documents.

One of the most important features of the protocol is that it uses a series of extensible text request and response MIME header lines. The use of text and a common format allows for new headers to be introduced without causing old implementations to fail. Old implementations simply ignore headers they do not understand. Guidelines for adding new headers have been introduced [32].

The formal syntax of a header is as follows:

[Header] : [Value] <CR><LF>

In the syntax above, the brackets are not part of the elements. CR is US-ASCII 13 and LF is US-ASCII 10. The CR is optional in all cases, but it is needed to be MIME compliant. All HTTP headers are limited to the 7-bit US-ASCII character set and are typically case insensitive. To end the header block, two null headers (blank headers), i.e., two successive CR-LF combinations, are sent.

Some typical negotiation headers that accept all document types, GIF images and HTML text in particular are:

Accept: \*/\*<CR><LF>

Accept: image/gif<CR><LF>

Accept: text/html<CR><LF>

This research takes advantage of the text extensibility feature to introduce new headers related to document access prediction.

## **2.5 Current Issues**

As for any widely used system, there are a number of complaints leveled against the Web. These fall into the two broad categories of feature issues and performance issues. Some of the more common feature issues are lack of session support, poor data security, and limited commercial transaction support. These are reviewed in Section 3.2. Also,

there is no inherent mechanism to provide for “self-organization” on the Web. Currently, links must be manually maintained and some semi-automated databases exist to categorize Web sites.

Performance issues relate to methods to increase performance, such as data compression, multiple protocol instances, protocol redesign, and caching. Performance issues are discussed in more detail in Section 3.1.

## **2.6 Summary**

This chapter discussed the three core components of the Web. These are the means to describe a document in a device-independent manner (HTML), the means to transport these documents over a network (HTTP), and the means to locate these documents on a distributed network (URL/URI). These three specifications combine to allow for a machine and network independent mechanism to obtain information. Methods to improve performance, which concentrate on HTTP, are described in the following chapter.

## **Chapter 3. User Response Time Improvement Techniques**

Response time is the metric that is most important to the user. It measures how fast the user receives a document. Decreasing this time is the objective of most Web performance research. This chapter discusses some implementation limitations on response time and various ways around them. These solutions include data compression, multiple instances of the protocol, a more efficient protocol, and document caching.

### **3.1 Definitions**

The definition of user response time and transfer time, as they are used in this research, are provided below.

- User response time is defined as the time differential between when a user requests a document and when the user receives the document in its entirety.
- Transfer time is defined as the amount of time it takes to transfer a document over the communications medium, including connection and request times.

### **3.2 Basic Performance Issues**

The two most common performance metrics for network-based applications are network transmission time and user response time. In many cases, these two metrics result in the

same value, such as the time needed to retrieve a document over a network. If the same document is later retrieved from a local disk cache, the user response time is much faster than it was before, since the network transmission time becomes zero. The metric that this research is concerned with is user response time, as it includes network transmission time and other factors.

Implementation bottlenecks for HTTP response time are described below.

- **Server implementation.** Obviously, using a slow computer as a server will lead to worse response times than using a fast computer in the same network environment. However, a poor server implementation will also severely degrade response time as transmission times increase due to the server not being fast enough to handle the request load. For example, a scheme as simple as having multiple processes already spawned as compared to spawning a process for each service request can lead to significant server response improvements [33].
- **Client implementation.** Client implementation issues are similar in nature to server implementation issues. A fast parsing and decoding algorithm can decrease response time. Displaying information as it is received also reduces apparent response time. Loading a document using a local cached copy, as opposed to a network copy, also

reduces response time. This thesis investigates the use of caching to improve Web user response time.

- **Transport (network) limitations.** A faster network generally improves user response time. Obviously, this is not always feasible. Other mechanisms such as protocol improvements and data compression can also help reduce response time. These are discussed in Section 3.3.

The user model used by a specific scheme to increase performance must be suitable to the task. Designing for worst-case user behaviors may create a sub-optimal design for 99 percent of user access patterns. For example, assume that a server is expecting an average of 100 connections per minute. One hundred processes are started to handle the expected connections. However, if only a dozen requests are made, then substantial server resources are wasted. This implies that how users are characterized and the performance criteria that result from the characterization are issues that need to be considered. This is discussed in greater detail in Section 4.1.

### ***3.3 Constraints on Increasing Performance***

The existing infrastructure is an important factor to remember when using any intrusive method used to increase performance. Intrusive methods are those that require some aspect of the infrastructure to change. Almost all methods are intrusive. The

World-Wide Web consists of a large number of servers and many of these have scripts and other custom software to perform specialized tasks. These servers often rely on “loopholes” or “features” in the Web. Any proposed design that is not significantly better than the current technology or that does not provide an easy migration path and backwards compatibility to the current technology will not be used.

### **3.4 Protocol (HTTP) Replacement**

Revolutionary replacement of HTTP for purely performance gains is not feasible in the foreseeable future. Huge performance improvements would be required to persuade a large number of people to change to a new protocol. Such significant performance improvements are unlikely. These performance improvements might also come at the loss of the extensibility and ease of implementation that HTTP currently enjoys. Protocol replacement might be beneficial and viable for very slow links or other specialized needs. Optimization of the way HTTP is currently implemented might also reduce the benefits of a new protocol.

The IETF HTTP-Next Generation (HTTP-NG) Working Group is developing a replacement for the current HTTP [34]. There are two areas that must be considered when designing a replacement protocol: features for performance improvements and features for increased functionality.

Spero [35] and Padmanabhan and Mogul [36] provide an overview of performance problems with HTTP. The basic problems include the following.

- Use of a stateless protocol means that negotiation information must be transferred on every request, even if it is for items that are part of the same document. Spero speculates that this redundant information factor is around 95 percent of the header information transferred [37]. The author's analysis shows that a typical worst case is 86 percent. Factoring in the required items that a stateful protocol must transfer would reduce the potential savings. The difference in the redundancy percentages is probably due to the more accurate analysis used to obtain the later number. For a detailed analysis on redundant information, see Appendix B. Regardless of the exact differences, there is a large amount of redundant information that is transferred.
- A significant amount of the negotiation that occurs is useless since implementations do not use the negotiation information. Almost all negotiation is performed by displaying the item to be sent and having the user manually select that item. This implies that this information does not need to be sent. Not using these fields will reduce both the transfer time and the amount of redundant information.
- The single request nature of HTTP means that some unnecessary transfer delay is added by TCP's round-trip time negotiation and slow start algorithm [35].



Padmanabhan and Mogul [36] describe and test some potential protocol modifications that improve response time. These and other latency issues are more important on an extremely slow network connection than a fast connection.

Several desirable features that are missing from HTTP are described below.

- A common feature of many protocols that HTTP is missing is the concept of a session. A session allows various actions to be associated with each other. The design of HTTP implies no direct association between various user actions. However, this causes problems. Consider, for example, an information provider that wants to serve information only to registered users through the use of a password. HTTP allows for basic password authentication, but the user must enter the password for every document access. This can be partially solved through clever, but complex, use of randomly generated documents based on user requests or other schemes. This added complexity is not desirable. A proposed solution exists in the form of HTTP header modifications [38].
- Partial document retrieval allows for selective retrieval of portions of a document. This has been recently specified in an Internet-Draft, which provides an illustrative example: “an Adobe PDF viewer needs to be able to access individual documents by

byte range; the table that defines those ranges is located at the end of the PDF file” [39].

- Commercial security mechanisms are a hot topic on the Web. Bossert, *et al.* provide an overview of security requirements for the Web [40]. A large number of proposals to provide channel security, billing and auditing, and authentication exist. A brief listing is provided in [41-43].

Many of these features and improvements can be added to the existing protocol through header extensions and modifications.

### **3.5 Data Compression**

A conceptually simple performance improvement is data compression. The Web primarily uses seven-bit ASCII text transmitted over an eight-bit data channel. Using the one extra bit for dictionary encoding could result in a sizable performance improvement with little or no extra overhead. From results presented in Appendix C, approximately 76 percent of the documents transferred are text based. For typical text documents, compression can range from 70 percent to 40 percent of the original size [44].

Any compression method used for user response time improvement should allow for fast decompression. It would not be effective to send a compressed document only to have

the decompression take longer than transferring the non-compressed version of the document. Another reason for fast decompression is that servers are usually fast machines, while most clients are not. Note that servers can also pre-compress the documents. However, compression on the fly is more desirable than pre-compressing. Given these considerations, the author believes that some type of dictionary compression method would be preferable.

Character frequency statistics from several experiments are reported in Appendix C. However, there is significant variance between runs. Based on the high variance and the fact that Bell, *et al.* advocate the use of adaptive compression techniques [44], an adaptive dictionary compression technique may be the best method. A text-based dictionary lookup is fast, especially given that 128 slots, using the eighth bit of the ASCII character set, are available at no extra cost.

In theory, documents may be compressed on the server using UNIX *compress* [45] or GNU *gzip* [46]. Many browsers support automatic decompression. The author believes that “*gipping*” is not widely used because it requires significant effort by the server administrator to pre-compress each document. Every update of the document requires decompression, modifications, and then another compression. Most Web document authors seem to avoid such tedious work.

In summary, compression for Web performance improvement is not frequently examined or cited, even though it seems likely that compression could provide a significant performance benefit.

### **3.6 Multiple Protocol Instances**

Multiple protocol instances are the equivalent of wanting to retrieve  $n$  separate files and using  $n$  simultaneous FTP sessions to retrieve them. This, in theory and in most practical cases, is faster than  $n$  sequential FTP sessions. Netscape Navigator, the most commonly used browser [26, 27], uses multiple protocol instances to improve performance. This feature is also used in the experimental system that is described in Chapter 5. The penalty for using this method is that additional network load is created.

### **3.7 Caching**

Given the normal case that loading a document from local memory or disk is much faster than loading over the network, caching is another way that response time might be decreased. Several browsers, including NCSA Mosaic and Netscape Navigator, perform some type of caching. The caching algorithms implemented by browsers vary significantly. NCSA Mosaic stores images and documents in memory on a per session basis, whereas Netscape Navigator does disk storage across sessions.

The two examples cited earlier are single user-based caching schemes. There are other schemes that use statistics for multiple users to implement a caching policy. This is discussed in more detail in Chapter 4

### **3.8 Summary**

HTTP has a high level of overhead due to its simple nature and its extensibility.

However, this simplicity is one of the key factors in its widespread use. Common problems associated with the protocol include statelessness, which leads to a high level of redundancy, unnecessary features, and the single-request nature of the protocol.

Additional features, such as security mechanisms and partial document retrieval, are also desirable. An IETF working group, the HTTP-NG working group, is studying the possibility of a complete overhaul of the protocol. Other groups are making extensions to the protocol to solve these problems.

Data compression, multiple simultaneous requests, and document caching are three ways to transparently improve HTTP performance. Data compression seems to be a simple method with the most benefits. Issuing multiple simultaneous requests has performance benefits, but adds network load. Document caching shows promise, as well, and is discussed in the next chapter.

## Chapter 4. Caching

Data caching is a common method to improve performance in computer systems. CPU caches in static memory improve performance by avoiding the need to retrieve data from slower dynamic memory. Memory disk caches perform the same function for the slower disk drive. Unsurprisingly, Web document caching is an active research area. Most research has been conducted in the area of document caching for large user pools as opposed to document caching for a single user. The research described in this thesis is based on single user document caching.

### **4.1 Basic Concepts**

Document caching attempts to load a document from a faster memory than where it is permanently stored. The document is stored in the faster memory based on the prediction that the user will view the document at a later time. Loading from a remote network-connected disk, e.g., a WWW server, is typically much slower than loading from a local disk. The difficulty in caching is to accurately predict which documents will be viewed next or in the near future.

A key idea used in most caching systems, such as memory and disk cache systems, is the concept of locality of reference. Locality implies that the location of the next document access will be near to the location of the last, or a recent, document access. The hypertext

Web structure provides locality that can be applied to Web document pre-fetching caching.

#### 4.1.1 User Models

The user access model is an important consideration when designing caching schemes.

For a particular model, performance may be excellent. But, if the model does not accurately follow the user's actual access pattern, then performance can degrade substantially. User models for hypermedia research are as follows [47, 48].

- **Directed search browsing** — The directed search model implies that users spend time searching for information within a specific locality of the Web. The user has specific information that he or she is deliberately seeking.
- **General-purpose browsing** — Users behaving according to the general-purpose model spend some time within one Web locality and then go to another. This is repeated throughout the session. A user has some unformulated idea as to what he or she wants to find and the user is looking around for it.
- **Random browsing** — The random model is the “window shopping” model. The user has no real objective in mind and is looking briefly at many documents.

#### **4.1.2 Cache Relaying Versus Caching**

One factor that affects distributed information caching is the level at which caching is done. This can be divided into two general cases, single user caching and multiple user caching. In the Web, multiple user caching is generally referred to as a cache relay. The end goal of each caching method is the same, that is to reduce response time and network load. However, details vary as specific algorithms take advantage of access properties attributed to the users being served by the cache.

A Web proxy server allows a client to retrieve Web documents by going through a special server instead of connecting directly to the server containing the document in question. It “relays” requests for one or more clients. A cache relay is a proxy server that performs caching. The caching is based on statistics generated from individual requests of clients that use the server. It does not account for other users who do not use the cache relay. Glassman reported that his cache relay had a cache hit rate of roughly 50 percent or more [8]. This shows that caching is a viable scheme.

Pitkow and Recker present a caching model based on human memory models [49]. The concept is that a given document is accessed within a specific time frame, say from 1 PM to 2 PM, within a given time period, say within any given week. This document has a certain probability of being accessed within the same time frame in the next week. The algorithm involves caching documents based on the time of day and other parameters.



Their conclusion is that this type of caching can be beneficial for very large caches at a cache relay.

As noted in Chapter 1, current research is not focused on single user caching mechanisms. This study does not attempt to address the question of the performance of single user caching versus multiple user caching. It does, however, look at a different pre-fetch caching concept than generally examined in multiple user caching schemes. Typically, pre-fetching is done based on statistics obtained from a proxy server's local user population. These statistics are typically site specific. This study looks at the statistics obtained from servers. These statistics are based on global user access patterns for the documents being served.

The concept of using global access statistics can be applied to multiple user caching, but it is simpler to study the single user case. From a logical standpoint, the algorithm presented, prediction based on remote server statistics, is of the most benefit for directed search browsers. Thus, this study focuses on the directed search browsing user model.

#### **4.1.3 Pre-fetch Caching**

The key to being able to perform pre-fetch caching is that there is idle time between requests or multiple communications channels that can be used to fetch documents before they have been requested. Idle time is simply the period when the user is viewing the

document. Based on the theory that idle time will be prevalent, a sample measurement was made. The idle time between requests was measured on three different servers that were serving four distinct types of documents, as described in Table 1.

**Table 1.** Servers Used in Feasibility Analysis

Server	Description
<i>www.vt.edu</i>	Virginia Tech WWW server
<i>fiddle.ee.vt.edu</i>	SUCCEED Electronic Connectivity Deliverable Team and IEEE SAC/IEEE Region 3 WWW server
<i>www.bev.net</i>	Blacksburg Electronic Village WWW server

The data for each server is presented in Table 2. The data was obtained by parsing access logs and comparing consecutive HTML requests from the same host within 100 accesses. Consecutive requests beyond 100 accesses were not considered. Single accesses and non-HTML accesses were also ignored. Note that the number of items, or document requests, used to calculated the byte average is given in the third column and the number of items used to calculate the wait time average is given in the fifth column. These numbers differ since the average number of bytes is calculated using every entry whereas the average wait time is limited to successive requests.

**Table 2.** User Reading Time Between Document Fetches

<i>SERVER</i>	<i>Average Bytes or File Size</i>	<i>Number of Items for Bytes</i>	<i>Average Time Between Requests (Seconds)</i>	<i>Number of Items for Time</i>
<i>www.vt.edu</i>	3270.0	999.0	36.9	626.0
<i>fiddle.ee.vt.edu</i>	11954.7	4677.0	98.5	1459.0
<i>www.bev.net</i>	3572.7	1234.0	47.7	622.0
<b>TOTAL Average</b>	<b>6265.8</b>	<b>2303.3</b>	<b>61.0</b>	<b>902.3</b>

From the results in Table 2, the average time between consecutive requests was 61.0 seconds. Each request was 6,265.8 bytes. Considering that the average HTML document size is around 5K to 6K bytes (see Appendix C), this means that a user typically spends a minute or so reading each document, or processes about 100 bytes a second.

## **4.2 Hypertext Access Graphs**

The focus of this research is user document caching as opposed to site document caching. The premise is that hypertext links can be used to predict access. These links allow a graph of the allowable hypertext access patterns to be constructed. For most cases, the only possible paths that can be followed are dictated by the arcs on the graph. Adding a heuristic to these arcs allows a prediction model to be built. This forms the basis of the pre-fetch caching used in this research.

Hypertext links are modeled as an arc in a directed graph (digraph), allowing a Web to be examined from a graph theoretic point of view. The digraph's nodes are the Web documents. The number of hops from a source to a destination node is called the depth, of the destination, away from the source node. Local self-references have a depth of zero (no arc out) and are discarded. Multiple paths from one document to a directly connected document are considered to be one link. Bidirectional paths are not allowed; these are considered separate arcs to and from each document.

### 4.3 Pre-Fetch Document Prediction

All pre-fetch, or “look ahead,” prediction is based on some heuristic measure. The most common pre-fetch caching concept is that of locality of reference, as described in Section 4.1. This concept can be applied to the Web since the number of local references to a given document is limited. There are known paths to the document’s location. The possible heuristics to determine which paths the user will access vary, but typically fall into prediction based by category and prediction based by statistics.

#### 4.3.1 Categorical Heuristics

A categorical prediction heuristic is based on a common observation, such as index documents are fetched more often than terminal information documents. So when the user is idle, index documents would be fetched before information documents.

**Table 3.** Sample List of Heuristic Categories

<i>Category</i>	<i>Description</i>
INDEX	A table of contents or a listing of links. Usually visited often. Typically has many outgoing links.
BOOK	A set of documents organized in a book format. Has links going in forward and backward directions.
ICON	Inline image used in a fashion similar to a desktop icon. Icons are used frequently. Typically has many incoming links.
INFORMATION	Terminal document, typically no further outgoing links and only a few incoming links.
ROOT	Root home document. Typically accessed very often and has links to and from the document.
DATA	Item such as a PostScript file; typically not used often. Typically has only a few links to it.
IMAGE	A graphic that is not used as an icon. Typically has a few links to it.
FORMS	A data entry document. Frequency of use is hard to estimate.

A simple table of categories is presented in Table 3. Note that these categories are arbitrary. They are defined based on some description of a “common” situation and have associated metrics assigned based on the frequency of access. Typically, each document must be assessed and partitioned by a human since automatic categorization is a non-trivial task. Considering that poor categorization may occur, high cache hit rates are unlikely. For these reasons, categorical heuristics are not studied in detail in this research.

#### **4.3.2 Statistical Heuristics**

Statistical information, such as the percentage of users that have accessed a particular document, are likely to improve cache hit rates. If 99 percent of the users do not find a particular document useful enough to access, then it is unlikely that a particular user in question will access it. Also, since server access statistics are fairly accurate, statistics are an improvement over static metrics based on categorical heuristics.

To further improve prediction based on access statistics, one can use the most powerful feature of hypertext, embedded links. A hypertext system can be mapped into a hierarchical graph [50], as discussed in Section 4.2. It is assumed that users usually follow the links in the hypertext graph. This assumption is not always true, but is most of the time. An exception is that most browsers include “forward” and “back” buttons. These buttons allow a user to move, in a linear fashion, to a document that they have

already accessed. While this traversal is still on the graph, the associated access information is not recorded at the server.

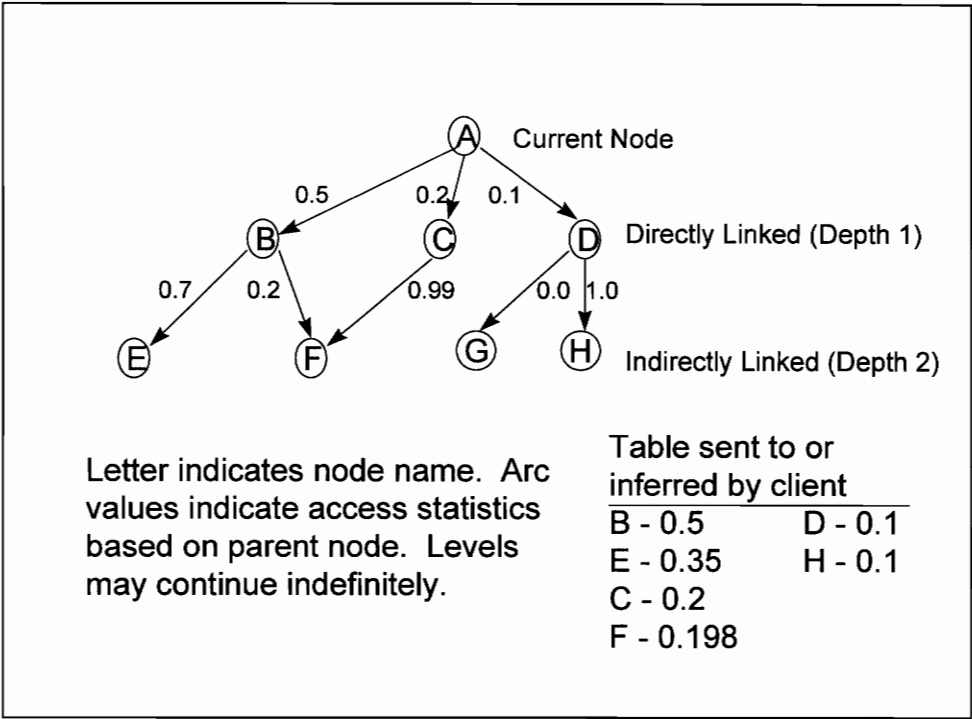


Figure 1. Statistical access digraph example.

A simple example is provided in Figure 1. Nodes A through H represent different documents. Node A is the document that the current user has most recently requested; it is the root of all possible digraphs for this state. Nodes B, C, and D are the next possible documents that the user can access using the hypertext links from node A. These are one depth unit away from the root node A. Nodes E, F, G, and H are two depth units away

from node A. For example, node E can be accessed using a hypertext link in B, but there is no hypertext link directly from A to E.

Each arc is labeled with a probability of access based on previous accesses of the destination node. Note that all probabilities sum to at most unity. Sums can be less than unity since there is the possibility that a user reaches a document and goes no further.

Each depth level can be assigned a factor to reduce the probability of access from node A, since these access probabilities are given from the parent node of the link in question. An alternative method is to simply multiply the probabilities along the path. The probability of accesses for a depth two search from node A, using multiplicative probabilities, is given in the table in Figure 1. For example, node E has a probability of 0.7 of being accessed from node B and node B has a probability of 0.5 of being accessed from node A, implying that the probability of access of node E from node A is 0.35.

There is the possibility that simple pre-fetching along the possible traversal routes can also obtain a good hit ratio and performance improvement. This is briefly examined in this research.

#### **4.3.3 Prediction Involving Multiple Servers**

Statistical pre-fetch prediction has one serious implementation problem. When multiple servers are being accessed by a particular user, there is no server data to predict when a

user will move from one server to another server. If the user goes to the next sever, a metric must be applied to determine when to stop pre-fetching documents from the old server. This stop pre-fetch metric will need to be carefully researched. If a poor metric is used, then unused files will be pre-fetched and the cache hit ratio will decrease slightly. This issue is not addressed in this research.

#### ***4.4 Post-Fetch Document Prediction***

Post-fetch prediction is typically based on the assumption that once a document has been seen by a user, there is a high probability that the user will look at that document again. Most current browsers implement some form of post-fetch document retrieval. Some of the implementations are only session based. A new session results in a document being retrieved across the network as opposed to being loaded from local disk. Others cache across sessions.

#### ***4.5 Cache Hierarchy***

Current Web caching mechanisms can be divided into user caches and cache relays. Cache relay systems can be classified based on the number of machines that the relay serves. These classifications range from lab cache servers to a University-wide cache server. However, the current basic function of any cache relay system is to make access predictions for some set of users. In the future, cache relay functions may include making pre-fetch prediction. So, the simplest cache relay is the single user cache. User caching can be divided into three separate caching functions.



- **Pre-fetch prediction cache** — A pre-fetch prediction cache stores documents that the user has not yet viewed. The system has retrieved these documents based on a prediction that the user is likely to view the documents.
- **Temporary document cache** — A temporary document cache stores documents that the user has previously viewed.
- **Permanent document cache** — A permanent document cache also stores documents that the user has already viewed. However, these documents have been viewed many times, meaning that the user is almost always going to view these documents during any given session. For typical usage, the temporary and permanent document cache can be combined into a single temporary cache.

In theory, this hierarchy of caches allows better prediction based on past accesses than one large document post-cache. However, permanent document caches are not studied as they require a large amount of statistical information from users and servers that is not readily available or easily obtainable. Pre-fetch and temporary caches are considered in this research.

## **4.6 Summary**

Pre-fetching documents may improve Web performance. One minute of idle time on the network, a typical document viewing time, allows a large number of documents to be pre-loaded. A potential pre-fetch method is one that converts a series of Web documents into a graph and assigns probabilities to the possible user access patterns. This provides a simple method to implement pre-fetching. Additionally, using access statistics from individual server logs provides a unique solution for obtaining the probabilities for a pre-fetch algorithm.

The pre-fetching scheme, combined with a traditional post-cache scheme, leads to the simple cache hierarchy presented in Section 4.5. This scheme is the basis for the experimental system described in Chapter 5.

## Chapter 5. Design and Implementation

Designing a HTTP pre-fetch caching system that uses multiple communications channels on a UNIX workstation is a complex task that requires attention to how the operating system allows multiple processes, or threads of control, to interact. A number of other issues also exist. The interaction between client and server must be specified to be backward compatible and still perform the desired tasks efficiently. Pre-fetch statistics must also be obtained and stored.

This research on user-based pre-fetch caching resulted in two programs. The NCSA *httpd* server [51] was modified so that it could accept pre-fetch requests and transmit document access probabilities to the client. A client that could interpret these probabilities and make multiple requests while still servicing the user's requests was also implemented. The client was designed to be integrated into a library that was not specifically designed to handle multiple requests. A communications mechanism to transport the access probabilities was specified as an HTTP header extension and is described in Appendix A. The protocol and other implementation changes are transparent to the server administrator and, more importantly, to the user.

## **5.1 Design Goals**

The design goals were as follows.

1. The design must be easily integrated into NCSA Mosaic for X-Windows [25] and run on multiple platforms. This can be done by ensuring that the design works with the World-Wide Web common code library (*libwww*) [52], used by NCSA Mosaic. The experimental system was developed without using the library; however, the implementation was tested to compile with the library. Note that compatibility with *libwww* results in a multiple process design that introduces some severe performance penalties. These penalties are addressed in Chapter 6.
2. The design must be simple and involve little to no intrusion into the infrastructure, i.e., it must be backward compatible.
3. No change in user access models or methods must be made. Some amount of server changes may be necessary.

## **5.2 Client-Server Interaction**

Pre-fetch document retrieval and prediction can occur at the server or at the client. Each approach is briefly discussed.

Server-side prediction requires extensive HTTP protocol modifications, which violates one of the design goals (see Section 5.1). The primary advantages and disadvantages are as follows.

1. Server lookup tables can be pre-computed and automated. No link information needs to be transferred if the server can automatically send the document. However, this is especially difficult if pre-fetching is to be performed across multiple servers. Essentially, access information must be traded across servers and the statistics may need to be corrected for different user sets.
2. The server does not know the state of the client. This information must be transferred and continuously updated.

A client-based prediction scheme has the following advantages and disadvantages.

1. The client already knows the documents that it has and does not have.
2. The client does not know the prediction information and has to receive this information from the server. It also does not necessarily know all of the available server documents.
3. The client may have to compute values and occasionally make guesses about which documents to retrieve.

A combination of client- and server-based pre-fetch document retrieval and prediction is also possible, but requires extensive modifications to HTTP. Based on the relative advantages and disadvantages, client side pre-fetch prediction is used in this work. This allows for a simple, non-intrusive modification to the HTTP protocol through the introduction of two new headers.

1. **Predict** — The **Predict** header is used by the client to request prediction information for a particular document. If a server does not perform prediction or understand this header, then the header is simply ignored.
2. **Predict-Link** — This header is used by the server to send link prediction information for the current document. For example, prediction probabilities values of other documents will be sent. The client should understand this since it is only sent in response to the **Predict** header.

A key feature of this header design is that server processing of the parameters requested by the client does not have to occur exactly as expected by the client. This allows different servers to implement different statistical computations based on their own needs, within constraints. For example, categorical heuristics can be used instead of statistical heuristics by sending fixed probabilities to the clients. Clients can choose to interpret the values as they desire and pre-fetch document in a slightly different order as desired, for

example pre-fetching graphic images after source documents. These headers and their use are described in detail in Appendix A.

## **5.3 Server Design**

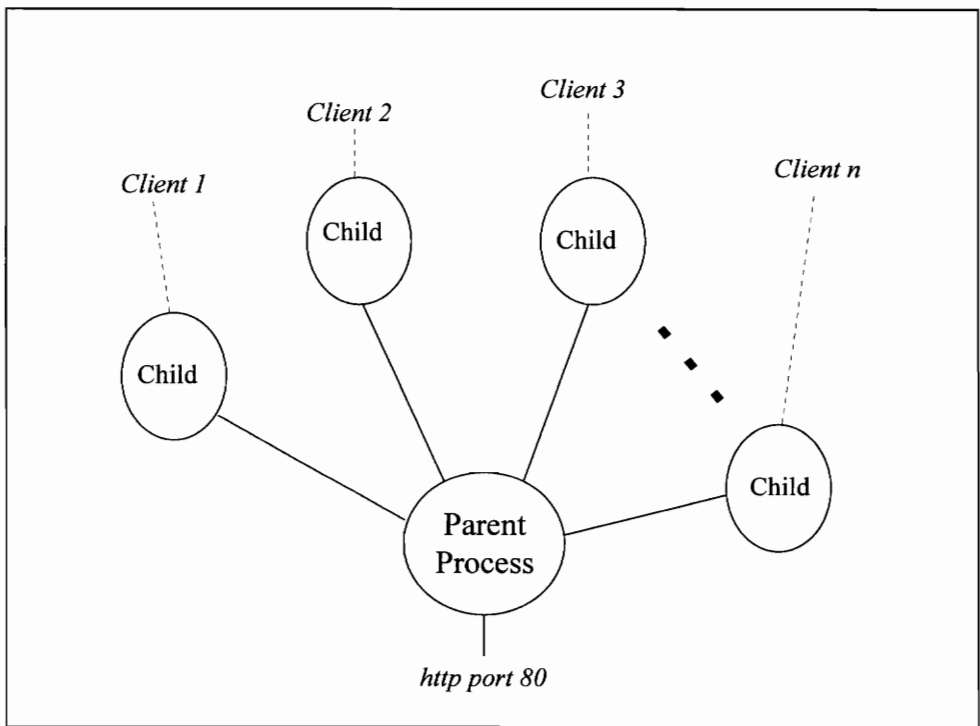
### **5.3.1 Statistical Access Database**

Since the caching scheme is based on statistical access patterns, a server with a high number of accesses was modified to save user access patterns using a special database. To provide this data to the client, the server was modified to access this special database and interface with the client using pre-fetch caching. The database treats each document as a graph node, as discussed in Section 4.2, and stores statistical access information for that document and its outgoing hypertext links. The information stored is the percentage of users who traversed the outgoing link among all users who accessed the document.

These access statistics do not faithfully model actual user access patterns. For example, the actions of the “forward” and “back” buttons on browsers are not recorded. Other effects, such as user caching, also reduce the accuracy of the model. User caching results in user accesses that are made locally but not recorded by the server. However, the access statistics should be accurate enough to enable good prediction.

### 5.3.2 Server Process Layout and General NCSA *httpd* Operation

The NCSA *httpd* server (version 1.4) [51] was modified to access the statistical prediction database and recognize and respond to the new headers. The server was designed by NCSA to operate as a traditional one process per client daemon, as illustrated in Figure 2. These concurrent processes allow the server to simultaneously service multiple clients. The solid lines in Figure 2 indicate UNIX inter-process communications (IPC) channels, and the dashed lines indicate network socket channels.



**Figure 2.** Server process diagram.



Each process is a separate entity that, once spawned from the parent process, requires no further interaction. Since each process is self-contained, the required modifications to NCSA *httpd* were minor. They entailed adding parsing logic to identify the **Predict** header and code to read the database and send out the **Predict-Link** headers.

The basic algorithm is as follows.

1. The parent process receives a connection request. The parent spawns, or has pre-spawned, a client process that processes the request. The parent process goes back to step 1.
2. The child determines the type of request.
3. The child processes the request headers. A flag is set if the **Predict** header is noted.
4. The child attempts to fulfill the request. If the **Predict** header flag is set, the process sends any available **Predict-Link** information.
5. The child process sends the requested document then logs the request and exits.

## **5.4 Client Design**

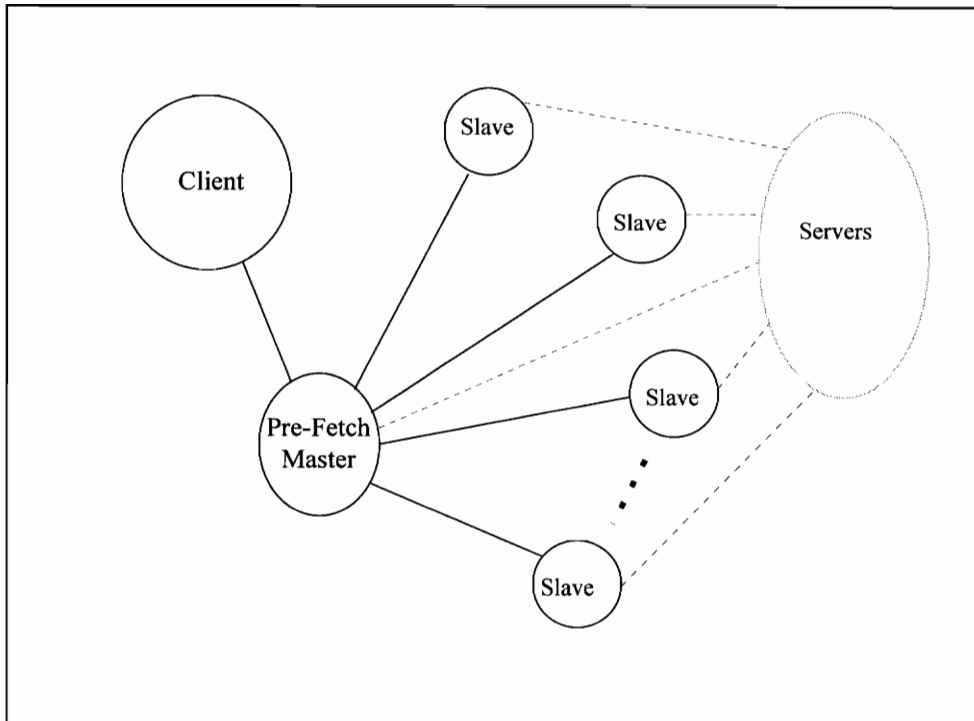
A key goal in designing the client was to integrate the client changes transparently into the *libwww* library. This means that developers using *libwww* would not have to change anything for this system to work. This was achieved by building the client as a multiple process program and remapping the socket calls. Since it uses multiple processes, a

variety of UNIX IPC mechanisms were used to communicate between each process. This design is inefficient compared to a multi-threaded approach, but threading is not universal in UNIX environments so that mechanism was not used.

In the design, there are three basic classes of process.

1. The client application itself, which processes user inputs.
2. A master pre-fetch process that interfaces between the client application and the network. The master process also manages several slave processes that pre-fetch documents. Specifically, the master process implements the cache deletion and insertion algorithm, receiving complete files from the slaves, performs the pre-fetch prediction algorithm and directs slave processes to obtain documents as needed, and serves as an intermediary between the client requests and the actual network or cache loads.
3. Multiple concurrent slave processes, each performing a simple HTTP GET request across the network.

Figure 3 shows the process organization. As in Figure 2, the solid lines in Figure 3 indicate UNIX IPC communication channels and the dashed lines indicate network socket channels.



**Figure 3.** Client process diagram.

All internal communications involve the pre-fetch master. This creates a bottleneck as the pre-fetch master performs all the essential tasks in the system. However, this particular design required modifications of only four procedures in *libwww*. All applications written to use that library should be able to use this code without modifications to the way they access the network.

Alternative designs were explored, especially dealing with the master-slave relationships. The task of maintaining distributed cache coherency and a distributed pre-fetch prediction algorithm were complex. Tests using file locking resulted in high overhead generated by

testing file locks. An iterative approach, i.e., one process cycling through many small tasks, required complex interactions that had to be re-coded if algorithms were changed.

The inefficiencies of the design would not affect cache hit rate experiments. The design was expected to be efficient enough for user response time tests. This, however, proved to be false for high network transfer speeds.

### ***5.5 Pre-Fetch Prediction Algorithm***

The implemented pre-fetch server reads the statistical access database and provides outgoing link information only for the current requested document, i.e., of zero depth. It does not provide probabilities for the other documents pointed to by the current document. Note that the specification in Appendix A allows for servers to provide more than one level of information. This feature was not used as this would have increased database complexity.

To obtain probabilities for depths greater than zero, the client requests the documents that are referred to by the outgoing links. This is performed recursively to some maximum specified depth. A weight is then applied to the returned probability values by multiplying the values by a constant factor. This constant factor is increased as the document depth increases.

Other methods of obtaining weighted probabilities for depths greater than zero could have been implemented. Examples would include having the server maintain extensive pre-computed probability tables for each document or recursively obtain and compute, upon client request, the weighted probabilities. These other algorithms are more complex and require significant redesign of the server. Since the access statistics are not completely accurate and the protocol allows for many different implementations, the implemented method was a reasonable compromise between multiplicative probabilities, as described in Section 4.3.2, and other methods.

The basic algorithm is as follows.

1. At the “current document,” obtain heuristics (probabilities) for all direct links. Enter these into a table. Note that the “current document” is the document that the user is currently viewing.
2. Retrieve the document with the highest heuristic value and enter its link heuristic information into the table based on the following rules.
  - Weight the heuristic by some amount depending on its depth from the “current document.” The implementation uses a fixed weight of ten percent per unit of depth.
  - If the entry is a duplicate, remove the entry with the lowest heuristic value.

3. If the user views another document, reduce all heuristic values in the table by some weight. The implementation uses a fixed weight of ten percent.

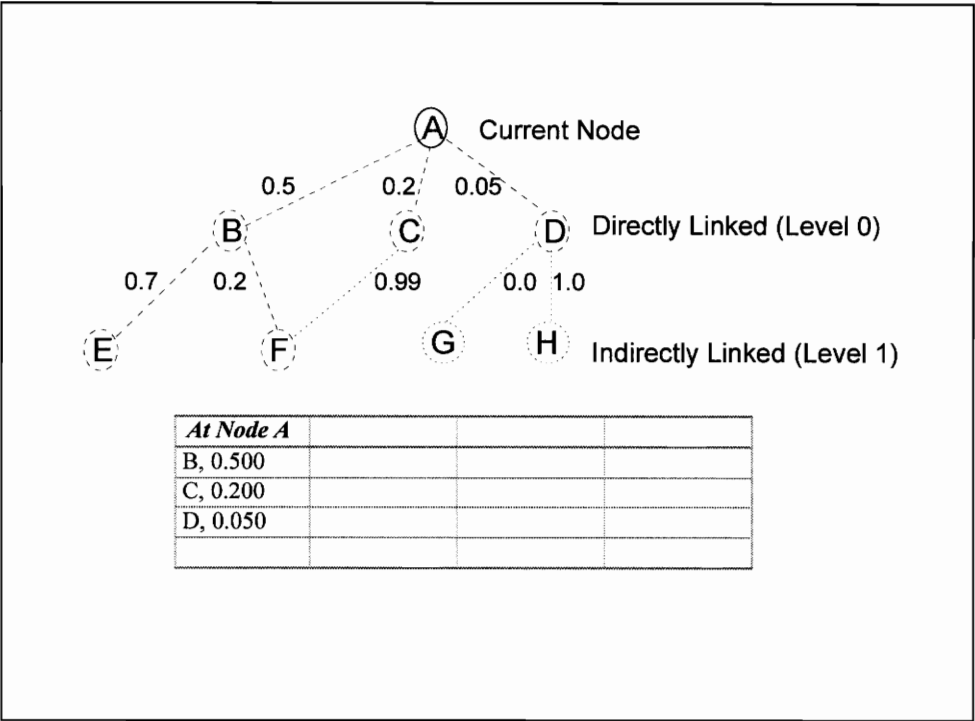
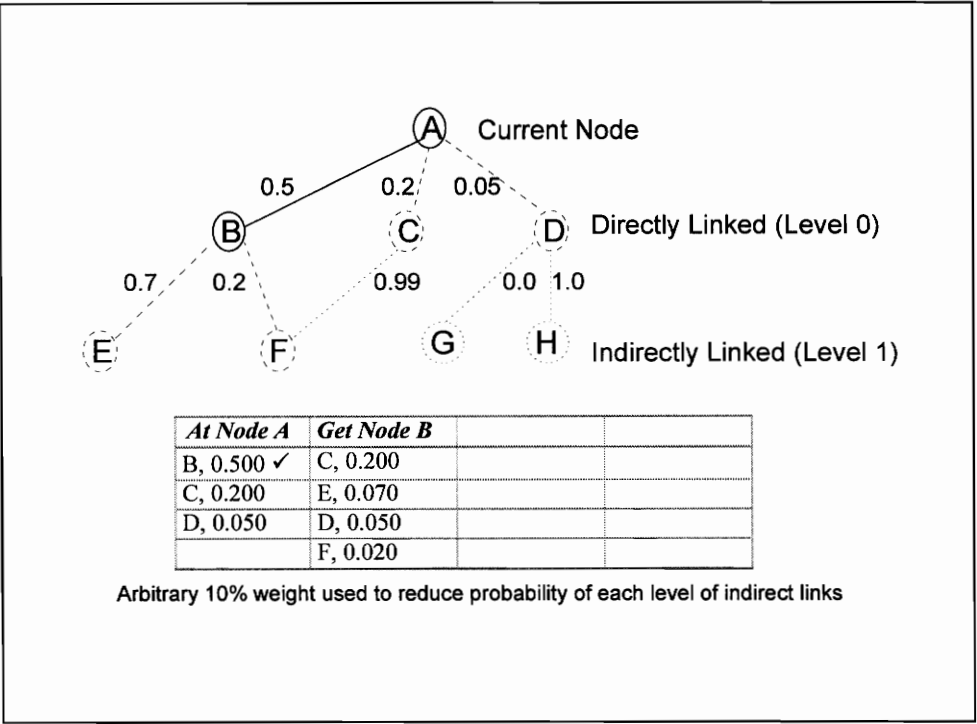


Figure 4. Initial web graph.

An example of the algorithm is given in Figures 4 to 7. Figure 4 provides the initial state of the system from the client’s perspective. Lines indicate links and circles indicate documents. Solid lines and circles are links and documents that have been traversed, or viewed. Dashed lines and circles are links and documents that the client knows about but has not yet traversed. Dotted lines and circles are links and documents that the server

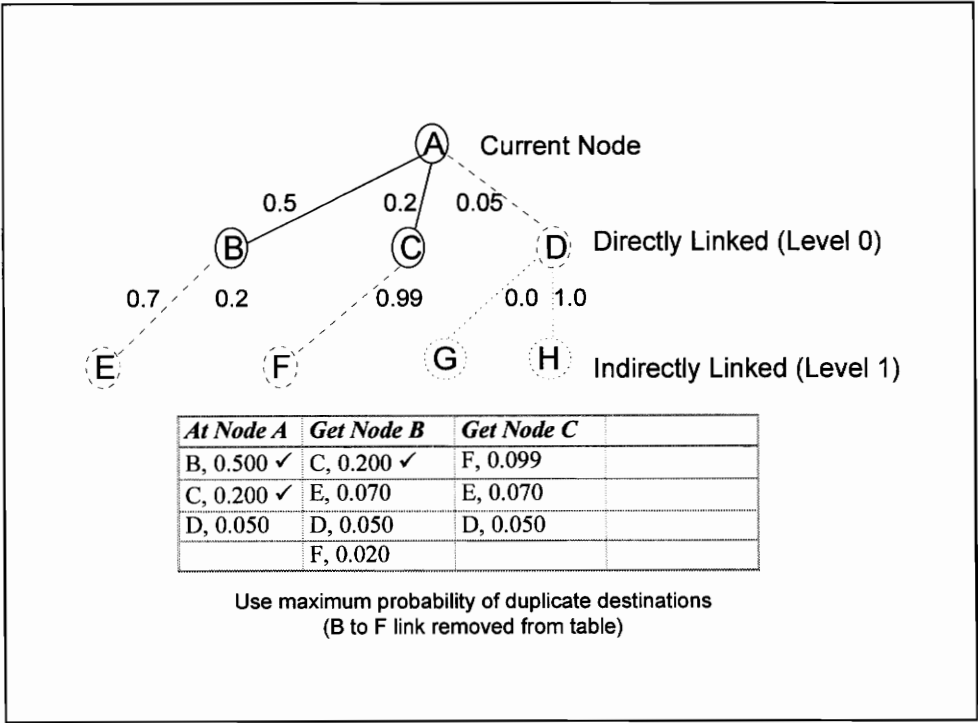
knows about but that the client has no information. The table used to store probability information is used to predict document access patterns.

In Figure 4, the initial state is that the client knows about nodes A, B, C, and D. It currently is displaying node A. The assumptions for this example are that only one channel is available and the user is viewing node A. This means that there is currently no network transmission. Since the network is idle, the client selects node B for retrieval because it has the highest probability of being viewed next. This is shown in Figure 5. The check mark in the table indicates that the client has obtained this node.



**Figure 5.** Web graph after retrieving node B.

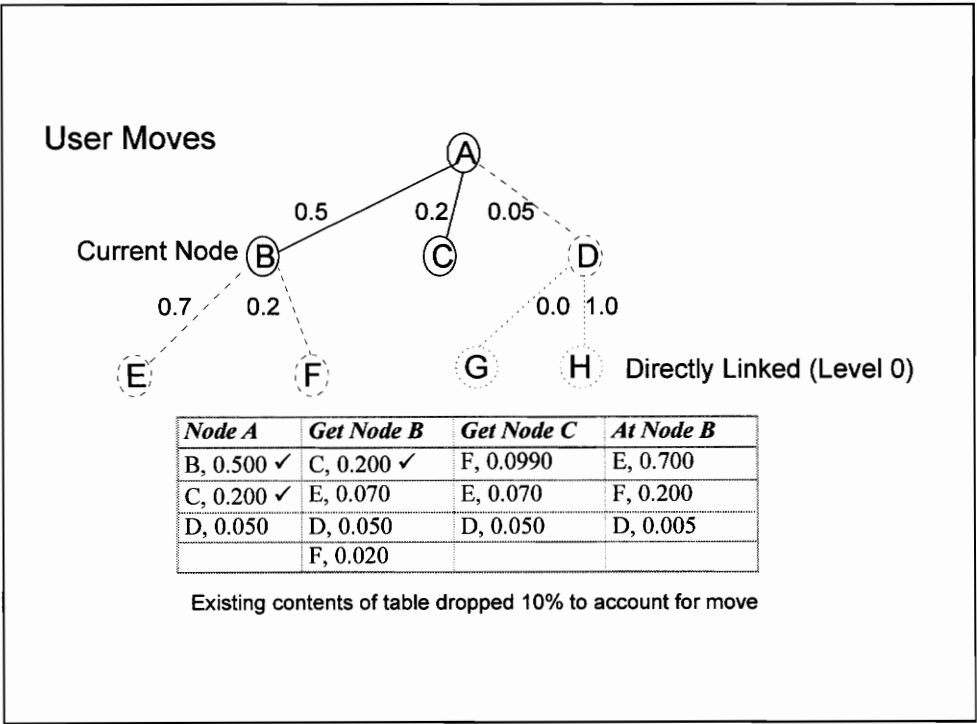
In Figure 5, the table has been updated to include all the nodes that the client can see after obtaining nodes A and B. Probabilities from node B to node E and F are set to ten percent of their original values since E and F are at depth two from A. The next step, assuming the user is still idle, is to obtain node C. The results are shown in Figure 6. Note that the link from B to F is removed since the link from C to F has a higher access probability. This process continues until the user moves to a new document or there are no more documents to retrieve.



**Figure 6.** Web graph after retrieving node C.



An example of the user deciding to view node B is given in Figure 7. The probabilities are again updated with B as the current node. The link from B to F reappears and the link from C to F is removed. Link B to F has a higher probability since the user is currently at node B.



**Figure 7.** Web graph when user moves to node B.

Pre-fetch tables are maintained using a simple ordered insertion and lowest probability deletion algorithm. The table size is fixed at 100 items. Probabilities obtained beyond this limit are discarded.

## **5.6 Summary**

This chapter presented the general design of a system to implement pre-fetch caching in the Web, include the basic structure of the server and client. The server design is fairly standard, whereas the client has some design “quirks” so that it can be integrated into *libwww*. The method to communicate prediction information was described and an detailed specification is available in Appendix A. Finally, the specific pre-fetch algorithm was illustrated.

## Chapter 6. Performance

Testing was done to evaluate the effectiveness of the concepts developed in this research. Traces generated by multiple users were used to drive the pre-fetch caching system. These traces recorded the document and its time of request for a particular user. The performance of the system was tested for different cache sizes, maximum document retrieval depths, available network bandwidth, and number of simultaneous requests. Statistical results and experimental observations are presented.

### 6.1 Evaluation Method

The NCSA *httpd* server daemon (version 1.4) [51] running on [fiddle.ee.vt.edu](http://fiddle.ee.vt.edu) (**fiddle**) was used to obtain user access patterns. The server modifications associate the **Referrer** HTTP headers, which report the document that the user utilized to access the current document, with the current document (URL) retrieved. This increases the accuracy of the server prediction access database. However, this does not solve all of the accuracy problems with the access logs as discussed in Section 5.3.1.

The server, **fiddle**, provides access to documents for the SUCCEED Electronic Connectivity Deliverable Team and the IEEE Student Activities Committee/IEEE Region 3. The server had an average 2,943 hits (accesses) per day from June 9, 1995 to

September 5, 1995. The server access data was collected over 16 days, from June 9, 1995, at 16:51:37, to June 25, 1995, at 20:41:26. The user trace data was taken from July 14, 1995, to July 21, 1995. Note that this is after the server access data was recorded.

The research could have used several methods for obtaining performance results for user-based caching, including obtaining document access traces for a massive number of users on a real server. This trace could then be used to drive the simulation of the caching algorithm. Alternatively, real users can be used to obtain performance results in real-time. These methods are typically used in cache relay experiments [8-9, 49]. Other methods include randomly generating document requests and selecting a few “typical” users and obtaining a trace of their activities. This research uses the last approach of tracing a few “typical” users.

The first option was not used for the same reasons that the access logs do not provide entirely accurate prediction values. Traces recorded by the server have numerous idiosyncrasies indicating that those traces are not accurate. Foremost was the high number of “impossible” jumps — there was no path between the two documents — from document to document in the server trace logs. Caching is the best explanation for these jumps. Another reason is that the number of accesses by a majority of users was below six HTML documents. The focus of this study is directed search browsing and the assumption is that such users will visit a large number of documents. Additionally,

deployment of the experimental browser for real-time performance testing was not feasible. It was also felt that random accesses would not accurately model real users, even random browsing users. This led to obtaining actual user traces from users given a directed browsing task.

### **6.1.1 User Traces**

Eleven users were asked to perform a directed browsing task on **fiddle** using a modified version of the NCSA Mosaic (version 2.6) browser. The modifications mapped **fiddle** to another server, which was mounted using the Network File System (NFS) to the same documents, to simplify tracking user access patterns. Using the original server, **fiddle**, would result in other users appearing in the data and require searching and re-parsing of the data. Originally, the test plan called for using both the IEEE and SUCCEED documents on **fiddle**. However, due to the nature of the IEEE documents which require some familiarity with the IEEE, those documents were not used. Using only the SUCCEED trace patterns resulted in ten user traces being selected instead of eleven. The eleventh user did not perform SUCCEED document retrievals.

All users were beyond novice level in Web browsing expertise. The users were graduate or senior undergraduate students in the Bradley Department of Electrical Engineering. Each user was given essentially the same set of instructions and a question related to a directed browsing task. These are shown in Figure 5.

**INSTRUCTIONS**

1. Press reload every time you go back or go to a document that you've been to before.
2. Try to stay on the server `fiddle.ee.vt.edu`.
3. You are looking for information content. There are no absolute answers to the questions below, other than you are done when you reach the point you think you know the answer enough to tell somebody else. It is primarily a directed browsing task.

**SUCCEED**

QUESTION: Assume that you are interested in starting a virtual corporation. Or, essentially, you have a group of geographically isolated individuals that are in your workgroup. You wish to make this a cohesive workgroup. Find out general information as to what areas, technologies, or other aspects of electronic means of communications would help you do this.

If you know the technologies already then you may be interested in seeking specific details, like products. Otherwise you may be seeking to determine what technologies are available.

**Figure 8.** User instructions.

The resulting user traces contained an average of 27.4 requests and had an average duration of 862 seconds (14.37 minutes). An average of seven requests per user were duplicates. Nothing prevented the user from leaving the server `fiddle`. Requests for documents that were not on `fiddle` were not included in the study.

The user traces were converted into a format suitable for simulation. Otherwise, the traces were unchanged. The default action of NCSA Mosaic to cache inline images, even when the user reloads the document, was kept in the simulation. This was done considering that images are typically larger than HTML documents and that almost every

browser caches images. The theoretical hit rate of the experimental system should be higher since, for a true no caching situation, images would not be cached.

### 6.1.2 Parameter Set

Four parameters, or factors, were selected as experimental parameters. Each factor had two value levels. From this, a multivariate analysis  $2^k$  factorial block was designed. The four factors are described below.

- **Bandwidth** — The capacity of the network connection between the server and the client is the bandwidth factor. The two value levels were fixed at a software-induced 1000 bytes per second bandwidth and unimpeded (no software imposed limit) Ethernet bandwidth. The maximum capacity available on the 10-BaseT Ethernet segment was estimated at 600 kilobytes per second, which was measured through 100 megabyte FTP transfers.
- **Depth of returned prediction links** — The depth factor, as discussed in Section 4.2, is the limit on how deep the simulation will try to retrieve documents. The depth limit value was selected to be zero and ten levels deep.
- **Number of channels** — The number of channels indicates the number of simultaneous transfers that can be active. There is always one channel available for

the user, so that immediate response can be given to the user. The other additional channels support the pre-fetch function. Levels are one additional channel and eight additional channels.

- **Pre-fetch cache size** — The caching algorithm was based on a least-recently-used (LRU) cache replacement scheme. Each different cache type (see Section 5.1) uses the same LRU algorithm. The cache sizes used for pre-fetch prediction were 50,000 bytes and 1,000,000 bytes. Temporary prediction cache size and permanent prediction cache size were fixed at large values of 10,000,000 bytes, which is significantly larger than the total size of all documents viewed.

The high and low values selected were based on what seemed to be reasonable minimum and maximum values rather than theoretical high and low values.

All possible combinations of parameters resulted in a factorial block of 16 different observations per experiment, as presented in Table 4. Each experiment, consisting of one user trace, was performed three times to reduce the effects of background network traffic. A total of ten user traces were used, resulting in a total of 480 different observations for the entire test suite.



**Table 4.** Factorial Block

<i>Designation</i>	<i>Depth</i>	<i>Speed</i>	<i>Channel</i>	<i>Cache Size</i>
0000	0	1 kbps	2	50 kb
0001	0	1 kbps	2	1 MB
0010	0	1 kbps	9	50 kb
0011	0	1 kbps	9	1 MB
0100	0	Normal	2	50 kb
0101	0	Normal	2	1 MB
0110	0	Normal	9	50 kb
0111	0	Normal	9	1 MB
1000	10	1 kbps	2	50 kb
1001	10	1 kbps	2	1 MB
1010	10	1 kbps	9	50 kb
1011	10	1 kbps	9	1 MB
1100	10	Normal	2	50 kb
1101	10	Normal	2	1 MB
1110	10	Normal	9	50 kb
1111	10	Normal	9	1 MB

## 6.2 Study Limitations

The general assumptions and limitations for the performance experiments are as follows.

1. In this research, the temporary document cache is the same as the permanent document cache. These caches were given a fairly large default size of 10,000,000 bytes.

2. There was a limit of 100 document references that can be in the pre-fetch tables.

There was no limit on the data types that can be retrieved for the cache. This may be

desirable since some users may only have a text browser and thus may not retrieve graphic images. However, it would be reasonable to expect that most users would like to see graphic images.

3. Simulations were allowed to run all day. Thus some effects of varying network loads may be included. Additionally, the server accessed the documents through NFS, which may have introduced some slight network delay. Each simulation was run three times in an attempt to average out these effects.

### **6.3 Preliminary Analysis**

A sample run of link-only pre-fetch prediction was made to assess the performance of that algorithm relative to the statistical pre-fetch algorithm. The link-only prediction scheme is to pre-fetch documents based on the links in the current document in the order that they appear. Order of appearance is the only metric used to determine which document to pre-fetch over another document. Effectively, a statistical weight of 100 percent was assigned to all links. Both experimental runs used the “best” possible parameter settings for both algorithms, which are noted below.

1. Bandwidth was the highest possible.
2. Depth was ten levels.
3. The number of channels was set at eight.
4. The cache size was set to 1,000,000 bytes.

The results for these experiments are shown in Appendix D. The relevant identifiers in the table are “linkonly” and “1111.” The outcome for this limited experiment between the two different pre-fetching algorithms was that using only document links is slightly worse than using statistical information. This indicates that if better prediction data is available it is quite likely that better performance can be achieved.

#### **6.4 Performance Analysis**

Complete experimental results are in Appendix D. The experiments tested performance against a control of a browser not using pre-fetch caching. The conclusion is that pre-fetch caching has a performance advantage over no pre-fetch caching. Details are presented in Section 6.5.

#### **6.5 Statistical Analysis<sup>1</sup>**

The original intent was to statistically compare the performance of the various parameter settings. Cache hit ratios and effective user response times were determined from the data in Appendix D. These were then transformed into bivariate variables through the use of a sinusoidal and logarithmic function, respectively. This was necessary since the statistical analysis model requires a standard normal distribution [53]. The results and factors were then processed using SAS [54], a statistical analysis package, and run as a

---

<sup>1</sup> Two consultants from the Virginia Tech Department of Statistics Consulting Center, Jeff Vest and Ping Fang, helped in the preparation of the analysis results.

multivariate analysis of variance under the general linear model procedure. The results for the statistical analysis are included in Appendix E.

Analysis was performed on all possible combinations of factors. The four-way analysis was tested against the hypothesis of no overall four-way combination effects and the results rejected this hypothesis. Further analysis revealed that all combinations also rejected the null hypothesis of no overall effects. This means that each factor has a significant effect on the variables. The overall result is that the statistical multivariate analysis was inconclusive. All four factors, bandwidth, depth, number of channels, and pre-fetch cache size, are equally important variables in affecting the cache hit ratio and user response time. Since no statistical conclusions can be drawn, the results presented in this study are inferences based on mean comparisons of various experimental variables and parameters.

## ***6.6 General Trends and Observations***

The original prediction was that environments with slow network speeds would not benefit as much as those with high network speeds. However, this proved to be false. The high network speeds, in this design, actually suffer somewhat. This is most likely due to the design of the program, since processes must wait on the master process for service. Ideally, given that network and CPU resources are not saturated, the response time for pre-fetch caching should, at worst, equal that for non-pre-fetch caching. This did

not occur. Hit rates for all situations with pre-fetching are at least double what they were without pre-fetching.

The primary variables of interest are the pre-fetch hit rate and the effective user response time. The pre-fetch hit rate, as opposed to the total hit rate, gives an accurate indicator on how well the prediction algorithm works. The effective user response time indicates the performance effect on the user.

#### **6.6.1 Effects of Network Bandwidth**

The most noticeable effect of network bandwidth is that the higher bandwidth (available Ethernet capacity) observations uniformly have worse effective user response times than the lower bandwidth observations (1,000 bytes per second). In theory, given sufficient network capacity and computer speed, the worst possible pre-fetch times would be close to the non-pre-fetch times. In the implementation studied, the average worst time per request is about eight times higher than the nominal time of 0.35 seconds. The nominal time is the user response time without any pre-fetch caching. A single extreme time that is 45.99 times higher than the nominal time causes this high average. The typical range was 1.10 to 2.82 times higher. This effect is attributed to the design of the experimental client software. Each process, including the user request process, must wait on several processes to be able to access the cache and the pre-fetch algorithms. The contention and internal data transfer times were significantly greater than the network transfer times.

This can also be observed and verified by the increase in response time relative to the number of actual files retrieved. The higher response times occur when an extremely large number of files are retrieved. An average of 937 files are retrieved in these cases compared to average actual user requests of 27.4 files.

The average actual user response time was 11.1 seconds. The average response time reduction for low network bandwidth was 61 percent. This represents a significant improvement, up to a 7.2 second decrease, in the user response time. This improvement could possibly be better, considering the above problems with delays introduced by client process contention.

The network capacity does not appear to have any direct linear effect on the cache hit ratio. The hit ratio varies significantly across different network bandwidths and the relationship to network capacity is not obvious.

#### **6.6.2 Effects of Document Depth**

A pre-fetch depth of zero appears to provide better overall performance than a higher depth. The average zero-depth cache hit ratio was approximately equal to the average of the higher depth pre-fetch cache hit ratio, but the average number of unnecessary files for zero-depth cases was considerably less than for the higher depth pre-fetch. Several samples showed that there were actually no unnecessary files downloaded.

The hit rate for the higher depth pre-fetch varied greatly, from 0.292 to 0.617. The hit rate for zero-depth varied from 0.350 to 0.471. This supports the observation, as made in Section 6.5.4, that the caching algorithm seems to be significantly influenced by the pre-fetch depth. These values may also have been affected by the LRU replacement policy, as will be discussed in Section 6.6.4.

### **6.6.3 Effects of Number of Channels**

As expected, more simultaneous sessions, or channels, led to an apparent increase in the hit ratio. However, more channels also resulted in a significant increase in the number of unnecessary files. There appears to be no correlation between the number of channels and the user response time.

### **6.6.4 Effects of Cache Size and Document Replacement Policy**

There appears to be some degradation introduced by the LRU document replacement policy. A sample experiment, which is not reported in detail, with a 10,000,000 byte pre-fetch cache resulted in a significant increase in the hit ratio. This means that documents requested by the user are deleted from the smaller cache before they are used. Specific experiments for this effect were not conducted. However, the early deletion of cached documents can be observed from the aggregate average over the 480 observations of 3.8 megabytes of deleted files for 400 kilobytes of files that the user actually requests.

The LRU policy is overwhelmed by retrieving an order of magnitude more bytes and files than are used.

Another observation is that there are a large number of unnecessary files. An average of 144.1 files were never used and 133.5 of these were deleted out of the cache.

All of the observations indicate that the cache was fairly active and the cache size limited the hit ratio. It may have also adversely affected user response time. The implication of this is that a better caching policies should be used for inserting items into and deleting items from the cache. Better insertion control should reduce the number of unnecessary files and better deletion control should increase the hit ratio.

#### **6.6.5 Combination Effects**

The effects of factor combinations were hard to observe due to the large number of combinations and the variety of data values. Since the statistical analysis showed that four-way combinations had a significant effect, there are no definite conclusions that can be drawn.

#### **6.7 Summary**

Experimental results indicate that pre-fetching is a viable scheme to improve Web performance. A better client design should allow for performance increases without the



observed response time decreases. The LRU strategy for cache control is probably not the most desirable option.

All of the measured variables had statistically significant effects on performance. The general result is that the individual “better” cache parameter settings of high depth levels, high network speeds, high number of multiple sessions, and large cache sizes, tended to provide better results. Combinations of settings had more ambiguous results, especially in the case of high network speeds and a large number of sessions.

## Chapter 7. Conclusions

This research began by exploring possible methods to improve user response time in World-Wide Web transactions. Statistical characteristics of average transfer time, user idle time, and document size and composition indicated that user-based pre-fetch caching would be a beneficial area to explore. Pre-fetch methods were evaluated, designed, and implemented. The resulting system uses a modified Web server to send statistical information about document links to an experimental browser. The browser can then pre-fetch documents while the user views other documents. Performance experiments were conducted to see if, and by how much, the system improves user response time. Additionally, the experiments studied the effects of various cache parameters to see how they affect performance.

The study showed that the user response time for low-speed or highly congested network connections is substantially improved (decreased) by a simple pre-fetch prediction implementation. This improvement, on average, varies from an 18 percent to 63 percent reduction in response time for an average response time of 11.1 seconds. However, the study also showed that there is a slight increase, of 109 to 135 percent, in times for fast network connections. This increase in response time is attributed to implementation problems that result in significant client process contention.

From the Georgia Tech's Graphic, Visualization, and Usability Center's Second World-Wide Web survey, at least 20 percent of respondents indicated that they were running servers with network connection capacity of 128 kbps or less [55]. Considering that 62 percent of the client population appears to be using slow lines, i.e., less than T1 [56], the pre-fetch prediction mechanisms outlined here should provide a performance improvement for many users. Limitations imposed by increased network traffic caused by pre-fetching were not explored in this study, so achievable performance increases may not be as good as this study indicates.

The results also show that LRU may not be the best policy for pre-fetch cache management. Many unused files are retrieved and many needed files are retrieved and then discarded prior to use. Other recent studies have also shown that the LRU policy is a poor choice for proxy servers [9]. However, the LRU policy still improves hit ratios and, in certain situations, user response time. Another consideration with the caching policy involves cache coherency. Cache coherency problems can reduce the actual hit ratio and performance benefits. An increasing number of documents, such as current news briefs, seem to be based on document reloading and/or frequent updates, meaning that a cached document will be more likely to be out of date.

Given that all the factors under study had a significant effect on the hit ratio and user response time, more research is needed. The factors under study were a selection of the

possible factors that would be used in any realistic pre-fetch caching system. Future studies should consider that these factors and the number the levels were insufficient and should use factors with higher degrees of freedom for each parameter.

The single user caching paradigm has numerous problems. Many of these have been noted by Luotonen and others [9, 10]. The experimental system developed through this research indicates that document pre-fetching is a viable user response time improvement method. Since the concepts used in this research can be applied to both single-user and multiple-user systems, justification for deploying a pre-fetch scheme exists. Future research should be directed at determining better algorithms for user-based pre-fetch caching as well as systems for cache-relay. Additional work needs to be done in the area of pre-fetching across multiple systems. Such questions as when to stop pre-fetching following a server change can substantially affect performance of the caching mechanism.

## Chapter 8. References

- [1] J.E. Pitkow, "GVU Center NSFNET Statistics," Web Document, Graphics, Visualization, and Usability Center, Georgia Institute of Technology, May 22, 1995, <http://www.cc.gatech.edu/gvu/stats/NSF/merit.html> (February 1996).
  
- [2] T. Berners-Lee, "An Executive Summary of the World-Wide Web Initiative," Web Document, World-Wide Web Consortium, February 23, 1995, <http://www.w3.org/hypertext/WWW/Summary.html> (February 1996).
  
- [3] R.J. Vetter, C. Spell, and C. Ward, "Mosaic and the World-Wide Web," *Computer*, Vol. 27, No. 10, pp. 49-56, October 1994.
  
- [4] F. Kappe, "Hyper-G: A Distributed Hypermedia System," *Proc. INET '93*, pp. DCC-1 - DCC-9, August 1993. Also available at <ftp://ftp.ncsa.uiuc.edu/Hyper-G/papers/inet93.ps>.
  
- [5] F. Anklesaria, M. McCahill, P. Lindner, D. Johnson, D. Torrey, and B. Alberti, "The Internet Gopher Protocol: A Distributed Document and Search and Retrieval

Protocol,” RFC-1436, University of Minnesota, March 1993. Available at <ftp://ds.internic.net/rfc/rfc1436.txt>.

- [6] M.J. Fullton, K.J. Goldman, B.J. Kunze, H. Morris, and F. Schiettecatte, “WAIS over Z39.50-1988,” RFC-1625, June 1994. Available at <ftp://ds.internic.net/rfc/rfc1625.txt>.
  
- [7] M. Abrams, S. Williams, G. Abdulla, S. Patel, R. Ribler, and E.A. Fox, “Multimedia Traffic Analysis Using Chitra95,” *TR-95-05*, Dept. of Computer Science, Virginia Polytechnic Institute and State University, April 7, 1995. Also available at <http://ei.cs.vt.edu/~succeed/95multimediaAWAFPR/95multimediaAWAFPR.html>.
  
- [8] S. Glassman, “A Caching Relay for the World-Wide Web,” *Proc. of the First International World-Wide Web Conf.*, Amsterdam: Elsevier, pp. 69-76, May 25-27, 1994. Also available at <http://www1.cern.ch/PapersWWW94/steveg.ps>.

- [9] M. Abrams, C.R. Standridge, G. Abdulla, S. Williams, and E.A. Fox, "Caching Proxies: Limitations and Potentials," *TR 95-12*, Department of Computer Science, Virginia Polytechnic Institute and State University, July 1995.
- [10] A. Luotonen and K. Altis, "World-Wide Web Proxies," Electronic Proceedings, *Proc. of the First International World-Wide Web Conf.*, May 25-27, 1994, <http://www1.cern.ch/PapersWWW94/luotonen.ps> (February 1996).
- [11] T. Berners-Lee, "History to Date," Web Document, World-Wide Web Consortium, October 3, 1995, <http://www.w3.org/hypertext/WWW/History.html> (February 1996).
- [12] K. Hughes, "Entering the World-Wide Web: A Guide to Cyberspace," Web Document, Honolulu Community College, October 1993, <http://www.hcc.hawaii.edu/guide/www.guide.html> (February 1996).
- [13] P. Naur, ed., "Revised Report on the Algorithm Language ALGOR 60," *Computer Journal*, Vol. 5, No. 1, pp. 349-367, April 1963.

- [14] D. Crocker, "Standard for the Format of ARPA Internet Text Messages," RFC-822, University of Delaware, August 1982. Available at <ftp://ds.internic.net/rfc/rfc822.txt>.
- [15] F.B. Schneider and S. Mullender, ed. *Distributed Systems, Second Edition*, New York: ACM Press, 1993.
- [16] K. Sollins, "Thoughts on Standardizing URN Resolution Protocols," Internet-Draft, Massachusetts Institute of Technology, June 1995. (This is a working document.) Available at <ftp://ds.internic.net/internet-draft/draft-ietf-uri-urn-thoughts-0.0.txt>.
- [17] P. Hoffman and R. Daniel, "URN Resolution Overview," Internet-Draft, Proper Pub., May 1995. (This is a working document.) Available at <ftp://ds.internic.net/internet-drafts/draft-ietf-uri-urn-res-descript-00.txt>.
- [18] T. Berners-Lee, "Universal Resource Identifiers in WWW: A Unifying Syntax for the Expression of Names and Address of Objects on the Network as used in the World Wide Web," RFC-1630, CERN, June 1994. Available at <ftp://ds.internic.net/rfc/rfc1630.txt>.



- [19] T. Berners-Lee, L. Masinter, and M. McCahill, "Uniform Resource Locators (URL)," RFC-1738, CERN, Xerox Corp., University of Minnesota, December 1994. Available at <ftp://ds.internic.net/rfc/rfc1738.txt>.
  
- [20] R. Fielding, "Relative Uniform Resource Locators," RFC-1808, University of California at Irvine, June 1995. Available at <ftp://ds.internic.net/rfc/rfc1808.txt>.
  
- [21] B. Marchal, "An Introduction to SGML," Web Document, January 3 1996, <http://www.brainlink.com/~ben/sgml/> (February 1996).
  
- [22] T. Berners-Lee and D. Connolly, "Hypertext Markup Language: A Representation of Textual Information and Metainformation for Retrieval and Interchange, Version 2," RFC-1866, CERN, November 1995. Available at <ftp://ds.internic.net/rfc/rfc1866.txt>.
  
- [23] D. Raggett, "HyperText Markup Language Specification Version 3.0," Internet-Draft, World-Wide Web Consortium, March 1995. (This is a working draft.) Available at <ftp://ds.internic.net/internet-drafts/draft-ietf-html-specv3-00.txt>.

- [24] Netscape Communications Corp., *Netscape Navigator*, version 2.0 Gold Beta 1, 1996 Available at <http://home.netscape.com/>.
- [25] NCSA Software Development Group, *NCSA Mosaic for X-Windows*, version 2.6, 1995. Available at <ftp://ftp.ncsa.uiuc.edu/Mosaic/Unix/binaries/2.6>.
- [26] J. Cates, "The Creative Internet: Browser Statistics," Web Document, California Institute of Technology, July 1995,  
<http://www.galcit.caltech.edu/~ta/browsers/browser.shtml> (August 1995).
- [27] D.J. Garaffa, "BrowserWatch," Web Document,  
<http://www.browserwatch.com/> (February 1996).
- [28] G. Bell, A. Parisi, and M. Pesce, "The Virtual Reality Modeling Language, Version 1.0 Specification," Web Document, May 26, 1995,  
<http://vrml.wired.com/vrml.tech/vrml10-3.html> (February 1996).
- [29] T. Berners-Lee, R. Fielding, and H. Frystyk, "Hypertext Transfer Protocol - HTTP/1.0," Massachusetts Institute of Technology, Internet-Draft, October 1995.  
(This is a working draft.) Available at  
<ftp://ds.internic.net/internet-drafts/draft-ietf-http-v10-spec-04.txt>.

- [30] R. Fielding, H. Frystyk, and T. Berners-Lee, "Hypertext Transport Protocol - HTTP/1.1," Internet-Draft, University of California, Irvine, November 1995. (This is a working draft.) Available at <http://ds.internic.net/internet-drafts/draft-ietf-http-v11-spec-00.txt>.
- [31] N. Borenstein and N. Freed, "MIME (Multipurpose Internet Mail Extensions) Part One: Mechanisms for Specifying and Describing the Format of Internet Message Bodies," RFC-1521, September 1993. Available at <ftp://ds.internic.net/rfc/rfc1521.txt>.
- [32] D.M. Kristol, "A Proposed Extension Mechanism for HTTP," Internet-Draft, AT&T Bell Labs, January 1995. (This is a working draft.) Available at <ftp://ds.internic.net/internet-drafts/draft-kristol-http-extensions-00.txt>.
- [33] R.E. McGrath, "Performance of Several HTTP Demons on an HP 735 Workstation," NCSA Computer and Comm. Group, April 25, 1995. Unpublished Technical Report. Available at <http://www.ncsa.uiuc.edu/InformationServers/Performance/V1.4/report.html>.

- [34] S.E. Spero, "Progress on HTTP-NG," Web Document, University of North Carolina,  
<http://www.w3.org/hypertext/WWW/Protocols/HTTP-NG/http-ng-status.html> (February 1996).
- [35] S.E. Spero, "Analysis of HTTP Performance Problems," Web Document, July 1994, <http://sunsite.unc.edu/mdma-release/http-prob.html> (February 1996).
- [36] V.N. Padmanabhan and J.C. Mogul, "Improving HTTP Latency," Electronic Proceedings, *Proc. Second World-Wide Web Conf.*, October 1994,  
<http://www.ncsa.uiuc.edu/SDG/IT94/Proceedings/DDay/mogul/HTTPLatency.html> (February 1996).
- [37] S.E. Spero, "Next Generation Hypertext Transport Protocol," University of North Carolina, Internet-Draft, March 1995. (This is a working draft.) Available at <http://sunsite.unc.edu/ses/ng-notes.txt>.
- [38] A. Hopmann, "HTTP Session Extension," Internet-Draft, ResNova Software, Inc., July 1995. (This is a working draft.) Available at <ftp://ds.internic.net/internet-drafts/draft-ietf-http-ses-ext-00.txt>.

- [39] A. Luotonen and J. Franks, "Byte Ranges with HTTP URLs," Internet-Draft, June 1995. (This is a working draft.) Available at <ftp://ds.internic.net/internet-drafts/draft-luotonen-http-url-byterange-00.txt>.
- [40] G. Bossert, S. Cooper, and W. Drummond, "Requirements for HyperText Transfer Protocol Security," Internet-Draft, April 1995. (This is a working draft.) Available at <ftp://ds.internic.net/internet-drafts/draft-bossert-httpsec-req-00.txt>.
- [41] K.E.B. Hickman, "The SSL Protocol," Internet-Draft, Netscape Communications Corp., February 1995. (This is a working draft.) Available at <ftp://ds.internic.net/internet-drafts/draft-hickman-netscape-ssl-01.txt>.
- [42] J. Hostetler, J. Franks, and P. Hallam-Baker, "A Proposed Extension to HTTP: Digest Access Authentication," Internet-Draft, March 1995. (This is a working draft.) Available at <ftp://ds.internic.net/internet-drafts/draft-ietf-http-digest-aa-01.txt>.
- [43] D.R. Stinson, *Cryptography: Theory and Practice*, Boca Raton: CRC Press, 1995.

- [44] T.C. Bell, J.G. Cleary, and I.H. Witten, *Text Compression*, Englewood Cliffs: Prentice Hall, 1990.
- [45] Sun Microsystems Corp. SunOS 5.4, *compress*, 1994.
- [46] Jean-loup Gailly, et al., *gzip*, Version 1.4, August 1993. Available at <ftp://ftp.uu.net/pub/OS/gnu>.
- [47] L.D. Catledge, "Characterizing Browsing Strategies in the World-Wide Web," Georgia Institute of Technology, 1994. Unpublished Technical Report. Available at <http://www.gatech.edu/lcc/idt/Students/Catledge/browsing/UserPatterns.Paper4.formatted.html>.
- [48] F. Valdez, M. Chignell, and B. Glenn, "Browsing Models for Hypermedia Databases," *Proc. of the Human Factors Society - 32nd Annual Meeting*, Anaheim, Cal., Human Factors Society, pp. 318-322, October 24-28, 1988.
- [49] J.E. Pitkow and M.M. Recker, "A Simple Yet Robust Caching Algorithm Based on Dynamic Access Patterns," *Electronic Proceedings, Proc. 2nd World Wide Web Conf.*, 1994,

<http://www.ncsa.uiuc.edu/SDG/IT94/Proceedings/DDay/pitkow/caching.html> (February 1996).

- [50] R.A. Botafogo, E. Rivlin, and B. Shneiderman, "Structural Analysis of Hypertexts: Identifying Hierarchies and Useful Metrics," *ACM Transactions on Information Systems*, Vol. 10, No. 2, pp. 142-180, April 1992.
- [51] NCSA Software Development Group, *httpd*, Version 1.4, 1995. Available at [ftp://ftp.ncsa.uiuc.edu/Web/httpd/Unix/ncsa\\_httpd/httpd\\_1.4](ftp://ftp.ncsa.uiuc.edu/Web/httpd/Unix/ncsa_httpd/httpd_1.4).
- [52] W3C, W3C Reference Library (*libwww*), Version 3, 1995. Available at <http://www.w3.org/hypertext/WWW/library>.
- [53] J. Vest and P. Fang, Personal Communications, September 7, 1995. Department of Statistics Consulting Center, Virginia Polytechnic Institute and State University.
- [54] "SAS/STAT User's Guide, Release 6.03 Edition," Cary: SAS Institute, Inc., December 1992.

- [55] J.E. Pitkow and M.M. Recker, "Using the Web as a Survey Tool: Results from the Second WWW User Survey, " *TR-94-40*, Graphics, Visualization, and Usability Center, Georgia Institute of Technology, 1994. Also available at <http://ftp.gvu.gatech.edu/pub/gvu/tech-reports/94-40.ps.Z>.
- [56] J.E. Pitkow and C. Kehoe, "GVU's Third WWW User Survey," Graphics, Visualization, and Usability Center, Georgia Institute of Technology, Web Document, April 1995, [http://www.cc.gatech.edu/gvu/user\\_surveys](http://www.cc.gatech.edu/gvu/user_surveys) (February 1995).
- [57] R. Fielding, H. Frystyk, and T. Berners-Lee, "Hypertext Transport Protocol - HTTP/1.1," Internet-Draft, University of California, Irvine, November 1995. (This is a working draft.) Available at <http://ds.internic.net/internet-drafts/draft-ietf-http-v11-spec-00.txt>.
- [58] NCSA *httpd* Software Development Group, "Features of NCSA HTTPD 1.5," Web Document, August 1, 1995, <http://hoohoo.ncsa.uiuc.edu/docs/features-1.5.html#KeepAlive> (February 1996).



## **Appendix A. User Document Caching on the World-Wide Web**

The following is a draft document that is included to provide additional details about the research presented in this thesis. There may be a more current document available than the draft presented here.

The draft document has been reformatted to fit thesis constraints.

INTERNET DRAFT

David C. Lee

<draft-lee-www-cache-04.txt>

Scott F. Midkiff

February, 1996

Virginia Tech

Experimental

Bradley Department of Electrical Engineering

Expires July 1, 1996

## **User Pre-Fetch Document Caching on the World-Wide Web**

### **Status of This Memo**

This document is an Internet-Draft. Internet-Drafts are working documents of the Internet Engineering Task Force (IETF), its areas, and its working groups. Note that other groups may also distribute working documents as Internet-Drafts.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as “work in progress.”

To learn the current status of any Internet-Draft, please check the “1id-abstracts.txt” listing contained in the Internet-Drafts Shadow Directories on [ftp.is.co.za](ftp://ftp.is.co.za) (Africa), [ftp.nordu.net](ftp://ftp.nordu.net) (Europe), [munnnari.oz.au](ftp://ftp.munnari.oz.au) (Pacific Rim), [ds.internic.net](ftp://ftp.ds.internic.net) (US East Coast), or [ftp.isi.edu](ftp://ftp.isi.edu) (US West Coast).

Distribution of this document is unlimited. It is available at the following URI.

<http://fiddle.ee.vt.edu/~dlee/thesis/rfc-predict.html>.

## **Abstract**

This RFC specifies an experimental flexible client-based predictive pre-fetch (“look ahead”) request mechanism for the World-Wide Web. The specification provides for additional header modifications to the Hypertext Transport Protocol [1, 2]. It is currently specified for use with HTTP 1.1.

The pre-fetch mechanism can be used for both multiple user caching (proxy or cache relaying) and single user caching. The caching concept relies on accessing statistics from the server.

# Table of Contents

**1.0 INTRODUCTION .....92**

1.1 PREDICTION METHODS.....92

*1.1.1 Categorical Heuristics.....93*

*1.1.2 Statistical Heuristics.....94*

1.1.3 OBTAINING STATISTICS FOR THE BROWSER.....95

1.2 DEFINITIONS OF PROTOCOL HEADER PARAMETERS.....96

*1.2.1 Prediction "Depth".....96*

*1.2.2 Statistical Prediction Values .....96*

**2.0 CLIENT REQUEST HEADERS.....97**

2.1 APPLICABLE METHODS .....98

2.2 ADDITIONAL METHODS.....99

2.3 REQUEST HEADERS .....100

*2.3.1 Prediction Parameters.....101*

*2.3.2 Prediction Link Ignore Headers.....104*

*2.3.3 Heuristical Specifier Headers.....106*

**3.0 SERVER RESPONSE HEADERS.....106**

3.1 PREDICTION INFORMATION ACROSS SERVERS .....107

3.2 SERVER RESPONSE HEADERS.....107

*3.2.1 No Prediction Information Available .....107*

*3.2.2 Link Information.....110*

**4.0 CONCLUDING REMARKS .....112**

**5.0 SECURITY .....113**

**6.0 REFERENCES .....114**

**7.0 AUTHOR'S ADDRESSES.....116**

## **1.0 Introduction**

Local disk access is typically several times faster than network access. This leads to the observation that if a user views a document that is already on the local disk, then the user will experience a faster response time. Many methods of performing Web document caching exist. Most research is focused on cache relays [3-5]. These vary from storing all documents visited to predictively “looking ahead” [6] and retrieving documents that an algorithm predicts the user will view. This memo deals with a pre-fetching scheme for the World-Wide Web.

This memo first reviews some prediction metrics and then provides some definitions that will be used in the specification. This is followed by a discussion of client and server HTTP headers.

### ***1.1 Prediction Methods***

Predictive pre-fetching requires information about the likelihood of a particular document being accessed in the future. There are two basic methods that can be used to develop these prediction probabilities. The first is to categorize documents into assigned next access probabilities based on some characteristic of the document. The second is to use historical user access patterns as the basis for next access probabilities. These are discussed in more detail in [6], but an overview is provided here.

### 1.1.1 Categorical Heuristics

Categorical heuristics require “buckets” into which documents are tossed. Documents may be classified by examining the number of links contained within a document or the number of documents that link to it. For example, a document that has a large number of links is typically an “index” document. An “icon” image document will typically have many documents linking to it. Pure “informational” documents will typically have few links from it and few documents linking to it. Another classification scheme may be based on the organization of the documents. A “book” classification may mean that the documents are organized in a page forward and backward manner with an associated table of contents.

The classification is used as the prediction metric. A “index” document will typically be accessed more often than a “informational” document. Thus, each category has some associated access probability.

Categorical heuristics have two main problems.

1. Complex logic is needed to automatically and accurately classify each document.  
One would envision that an administrator would have to manually classify each document.
2. The classifications may not be good predictors of actual user access probabilities. For example, not all “index” documents are equally likely to be accessed.

The protocol specified in this memo does not completely specify a mechanism to use categorical heuristics.

### **1.1.2 Statistical Heuristics**

The other method to predict accesses is to use historical user access patterns to predict future user access patterns. For example, if 95 percent of users have accessed document X and 10 percent have accessed document Y, the pre-fetch algorithm should obtain document X before Y. It may or may not decide to obtain document Y at all. The basic requirements of statistical prediction are as follows.

1. Adaptable. If the statistics are not adaptable, then over time, the statistics may not match the actual frequency of access and, thus, the predictions will be inaccurate.
2. Accurate. If the statistics are not accurate, then the predictions will not be accurate.

Using current logging methods, some which may require modification, there are two easily obtainable statistics.

1. Total document hit statistics. The total document hit statistics are the total number of users who have accessed that document on that server.



2. Total link traversal statistics. The total link traversal statistics are the number of users who have traversed a particular link from a particular source document. The sum of the number of users traversing a link in the source document plus the number of users who did not traverse a link (perhaps because they ended the session) equals the total document hit count for the source document.

These two statistics are not entirely accurate as they do not reflect all possible user movements. For example, by using caching algorithms, there are documents that are accessed that are not logged by the server. Another example is that the forward and back buttons on many browsers allow a user to traverse up and down links without accessing the server.

### ***1.1.3 Obtaining Statistics for the Browser***

Numerous methods are available to provide the browser with the access statistics necessary to perform pre-fetch prediction. These include a new protocol, embedding statistics within HTML documents, and modifications to HTTP. This memo specifies HTTP header modifications to provide the client with the appropriate statistical information.

## **1.2 Definitions of Protocol Header Parameters**

### **1.2.1 Prediction "Depth"**

Depth is defined to be the number of reference links (specified by an HTML HREF element) away from a specific document. A depth of zero represents links from the specific document. A depth of one represents links from documents of depth zero. How an implementation uses the depth feature of the protocol is loosely specified. For clients, the general inference is that the larger the number, the more items the user wants to pre-fetch. For servers, the depth metric should be fairly tightly bound to the number of links away from a specific document. Note also that the deeper the level, the worse the statistical accuracy becomes. A server implementation may decide to ignore a client request for pre-fetch retrievals at large depths.

### **1.2.2 Statistical Prediction Values**

Prediction values, for a particular document, are a measure of the probability that the document will be viewed in the near future. The range that the experimental specification specifies for the prediction values is as follows.

1.000 The user will almost certainly view this document.

0.000 It is highly unlikely that the user will view this document.

The prediction values are not necessarily statistical access probabilities. However, it is strongly encouraged that they do have some statistical relationship. The prediction values can be any metric as long as they fall into the bounds specified above. Clients should interpret any value above 1.000 as 1.000 and any negative value as 0.000. Significant digits beyond three decimal places should be ignored. Clients should accept any number of significant digits, however.

Note that clients can apply their own weights to the server supplied prediction values. For example, if a client tracks the accuracy of server prediction values and notes that they are not accurate, it may choose to only obtain documents with very high prediction values.

## 2.0 Client Request Headers

The normal HTTP client headers can be used in the pre-fetch scheme. Non-pre-fetching servers should ignore pre-fetch headers. All document references returned by the pre-fetch scheme are based on the Universal Resource Locator (URL) specification of RFC 1738 [6] and RFC 1808 [7]. All standard and relative URLs can be used where an URI/URL specification is allowed.

### 2.1 Applicable Methods

The headers specified below are only applicable to the GET and HEAD methods specified in HTTP 1.1. Since prediction has no meaning for the other methods, the prediction headers are to be ignored in the content of other headers.

The actions that the server should associate with the methods are as follows.

GET	The client sends the GET request and the prediction headers (and other headers) are returned by the server with the document in question. By implication, this means that the user will view the document. Servers should be aware that the user may not actually view the document.
-----	--

HEAD	The client sends the HEAD request and the prediction headers (and other headers) are returned with the document header information. Servers should interpret this action as a query for prediction information.
------	---

Servers may ignore client prediction headers if they are not sent with either method above or those listed in Section 2.1.1.

## **2.2 Additional Methods**

This specification also allows for an additional method type to be introduced.

PREDICT-GET	The functionality and parameters of the PREDICT-GET method are the same as for the HTTP GET method. The requested document should be returned or an error returned. This allows the server to log these requests separately without severely skewing the server log statistics.
-------------	---

PREDICT-GOT	The functionality and parameters of the PREDICT-GOT method are the same as for the HTTP HEAD method. Only document header information is returned. This allows the server to log these requests as documents viewed by a user that was requested through the pre-fetching mechanism.
-------------	--

Clients are encouraged to use these methods if the server supports them. This allows the server to keep more accurate statistics when pre-fetching is used. Servers adhering to this specification must support these new methods.

### **2.3 Request Headers**

Kristol specifies extension mechanisms for HTTP [8]. In a sense, these headers could be applied as extension headers and handed off to the appropriate prediction manager.

However, considering the large bulk of headers that may be generated, an additional keyword is not in the best interests of performance.

The headers have been structured to be extendible, as prediction method changes, with a limited basic set of required parameters. The parameters are defined to be short and their context may vary within different prediction schemes. General, common sense interpretations are given that each prediction scheme must observe.

If the server does not understand a parameter, it should ignore it.

Note that if a client sends the HTTP 1.1 **KeepAlive** header, then the server should remember client prediction request information across the session. That is, the prediction document information should be sent for every request after a **Predict** header is seen. A

new **Predict** header should cause the server to reset the parameters to its prediction algorithm. This applies to all client-initiated headers.

### 2.3.1 Prediction Parameters

The basic header format is as follows.

**Predict** = "Predict" ":" depth-range ["," parameter["=" value]]

Parameters are as follows.

**Depth-Range** = "d" "=" (DIGIT\*3) "-" (DIGIT\*3)

**Max-Size** = "ms" "=" (DIGIT\*100) ["TB"|"GB"|"MB"|"KB"]

**Num-Items** = "n" "=" (DIGIT\*5)

**Allow-Types** = "allow-types" "=" (Content-Type)[, (Content-Type)]

**Allow-Methods** = "allow-methods" "=" (HTTP Method)[, (HTTP Method)]

**Remote-Servers** = "rservers" "=" (DIGIT\*3)

Descriptions of each parameter and options follow. The defaults, if the parameter is not supplied, are given in parenthesis. Note that negative values are not allowed. Server's should treat all negative values as an absolute value.

<b>Depth-Range</b>	<p>Maximum depth range to return. Must be supplied.</p> <p>Servers should make every effort to accurately supply data according to this parameter.</p>
<b>Max-Size</b>	<p>Maximum total size of items to return (default 1 MB).</p> <p>Note that this is not the size of the prediction information but the size of the documents that are referenced to by the prediction information. This can be used by clients specify the available disk space. A maximum size of zero (0) indicates that the client does not wish to receive prediction information.</p>
<b>Num-Items</b>	<p>Maximum number of items (default “20”). A value of zero indicates that the client does not wish to receive prediction information.</p>
<b>Allow-Types</b>	<p>Allowable HTTP Content-Types (default “*/”). The server should only provide prediction information for the documents that have the specified Content-Types. Multiple Content-Types are separated by commas.</p>
<b>Allow-Methods</b>	<p>List of allowed HTTP methods (default is all methods).</p> <p>The server should only provide prediction information for</p>



documents that can be accessed by the specified methods.

Multiple HTTP methods are separated by commas.

**Remote-Servers**      Number of server hops away to obtain prediction statistics from. For example, if the value is two, then the current server should try to obtain pre-fetch statistics for documents on other servers. These other servers are one hop and two hops away from the current server. By default, this value is set to zero.

Some examples are shown below.

**Predict: d=0-0; n=1, ms=1KB;**

This example specifies that prediction information for only one document, up to one kilobyte in size, should be sent. This document can only be linked from the currently requested document.

**Predict: d=1-10; ms=1MB; rservers=5; Allow-Methods=GET, POST**

This example specifies that prediction information for documents that total up to one megabyte and at most ten links from the current request, should be sent. The server should only send prediction information for documents that can be

accessed via **GET** and **POST**. If the server has prediction information from other servers of up to five hops away, it should use that as well.

**Predict: d=0-1; ms=20KB; n=10; Allow-Types=text/\*, image/\*;**

This example specifies that prediction information for documents, at a depth of zero to one and up to 20 kilobytes of documents, should be sent. No more than ten document prediction headers should be sent and these documents should either be text or image types.

To be compliant with this specification, this header must be supported. The only required option to recognize is the **depth-range**. Servers may ignore options they do not understand.

If authentication is required to access documents, then the server may choose not to respond to requests for prediction information for documents that the user is not authorized to receive. Authentication header information should be sent by the client in addition to the prediction request.

### **2.3.2 Prediction Link Ignore Headers**

The link ignore headers are used to inform the server of URIs or matches of URIs to ignore in returning prediction information.

```
Predict-Ignore      = "Predict-Ignore" ":" URI-Location |  
URI-Base-Location "*"
```

To ignore a specific URI, use the **URI-Location**. Otherwise use the **URI-Base-Location** with a wildcard to ignore a set of URIs. Regular expressions are not allowed.

The following is an example of ignoring a set of documents.

```
Predict-Ignore: http://fiddle.ee.vt.edu/succeed/*
```

This example informs the server **fiddle.ee.vt.edu** to ignore all HTTP accessible documents that have an URI that conforms to the pattern

**http://fiddle.ee.vt.edu/succeed**. Thus,

**http://fiddle.ee.vt.edu/succeed/home.html** would be not be in any prediction information but **http://fiddle.ee.vt.edu/succeed.html** would be.

This header must be supported by the server. It is optional for the client.

### 2.3.3 Heuristical Specifier Headers

This header field is reserved for future definition and usage. This is to allow the usage of heuristical categorization and keyword based pre-search information. The format is given below.

**Predict-Heuristic** = "Predict-Heuristic" ":" undefined

This header must be recognized by the server. Since it is not defined, it is ignored. It is optional for the client.

Usage of **Predict-Heuristic** and **Predict** is allowed. If a server receives both, it should use the **Predict-Heuristic** characteristics to assist in sending link statistics as defined by **Predict**.

## **3.0 Server Response Headers**

### ***3.1 Prediction Information Across Servers***

This specification allows for servers to transfer or obtain prediction information from other servers and use that to send to clients. A distributed prediction database may help increase user response time.

No method is specified as to how to do this. A suggested method would be to have the server send **HEAD** requests for a specific document for which it has a link and obtain prediction values to store.

### ***3.2 Server Response Headers***

A server that performs prediction will only send the prediction response headers in response to detection of a valid client **Predict** header. Otherwise, it will be indistinguishable from a server that does not support prediction.

#### ***3.2.1 No Prediction Information Available***

No prediction information and server errors can be indicated in two methods. The first method is indistinguishable from a server that does not support prediction; that is, sending no header information at all. The second is to inform the client of why there is

no prediction information. This header must only be used in the case of an error or unavailable prediction information.

Predict-Link-Info = "Predict-Link-Info" ":" Reason-Code Reason-String  
Reason-Code = (DIGIT\*3)

Typical reasons will be as follows.

Reason-Code	Reason-String
1XX	Statistical link failures
100	Failed. No link information: %s.
101	Failed. No prediction information for URI %s.
102	Failed. Unauthorized or unsupported option(s): %s.
103	Failed. Unauthorized or unsupported access request: %s.
104	Failed. Unauthorized or unsupported method: %s.
112	OK. Unauthorized or unsupported option(s): %s.
113	OK. Unauthorized or unsupported access request: %s.
114	OK. Method %s remapped to %s.
2XX	Heuristical link failures

200	Failed. No link information: %s.
201	Failed. No prediction information for URI %s.
202	Failed. Unauthorized or unsupported option(s): %s.
203	Failed. Unauthorized or unsupported access request: %s.
204	Failed. Unauthorized or unsupported method: %s.
212	OK. Unauthorized or unsupported option(s): %s.
213	OK. Unauthorized or unsupported access request: %s.
214	OK. Method %s remapped to %s.
5XX	Server errors
500	Server prediction error: %s.
501	Prediction database failure.
599	General server error.

This header is optional. Multiple **Predict-Link-Info** headers may be sent. Some **reason-codes** are failures while others indicate warnings. The HTTP **status-code** should not be used to convey prediction failure information.

### 3.2.2 Link Information

The sorting order for link information is defined to be in decreasing prediction values. Thus, the first link sent should be of the highest prediction value and the last link sent should be of the lowest prediction value. All link information should be sent in sorted order. Following the design philosophy of the Internet, the client should be prepared to accept link information in any order. The client may even wish to sort the link information according to its own algorithm.

Predict-Link = "Predict-Link" ":" URI-Location", p-value [" ,"  
parameters ["=" value]]

Parameters are as follows.

Depth = "d" "=" (DIGIT\*3)

Size = "sz" "=" (DIGIT\*100)

Links-out = "lo" "=" (DIGIT\*5)

Links-in = "li" "=" (DIGIT\*5)

P-heuristic = "h" "=" <undefined>

Content-type = "content-type" "=" (Content-Type)

**Method** = "method" "=" (HTTP Method)

P-value      "p" =" [0|1] ". "(DIGIT\*3)



Parameter explanations are as follows.

Depth	Document depth away from requested document.
Size	Size of document, which should be equivalent to the Content-Length.
Links-out	Number of links in the document (including image and applet types).
Links-in	Number of links to the document (if known). This value can be an estimate.
P-heuristic	Heuristic categories. Currently undefined.
Content-type	HTTP Content-Type associated with document.
Method	Allowed HTTP access methods for document.
P-value	Prediction value as defined in Section 1.2.2.

Examples (assuming the base document is

<http://fiddle.ee.vt.edu/~dlee/thesis/rfc-predict.html>) are shown below.

Predict-Link: <http://fiddle.ee.vt.edu/succeed/>, p=0.992; d=0; sz=13262

Potentially pre-fetch the document at URI [/fiddle.ee.vt.edu/succeed/](http://fiddle.ee.vt.edu/succeed/). This has a high prediction value, so most clients will pre-fetch this document.

Predict-Link: ftp:index.html, p=0.032

Get the document, via FTP, at URI /fiddle.ee.vt.edu/~dlee/thesis/index.html.

This has a low prediction value so most clients would not pre-fetch this document.

It is recommended that implementations send the depth parameter used in the prediction, since server implementations may not support the requested client depth. Also note that a prediction value of 0.000 may be sent indicating the existence of an URI and the client may still choose to retrieve that document.

## **4.0 Concluding Remarks**

This work is part of M.S. thesis work at Virginia Tech [6]. It is an experimental specification that appears to provide for response time reductions and fairly high cache hit ratios.

## **5.0 Security**

This RFC does not deal with security considerations.

## 6.0 References

- [1] T. Berners-Lee, R. Fielding, and H. Frystyk, "Hypertext Transfer Protocol - HTTP/1.0," Massachusetts Institute of Technology, Internet-Draft, October 1995.  
This is a working draft. Available at  
<ftp://ds.internic.net/internet-drafts/draft-ietf-http-v10-spec-04.txt>.
  
- [2] R. Fielding, H. Frystyk, and T. Berners-Lee, "Hypertext Transport Protocol - HTTP/1.1," Internet-Draft, University of California, Irvine, November 1995. This is a working draft. Available at <http://ds.internic.net/internet-drafts/draft-ietf-http-v11-spec-00.txt>.
  
- [3] A. Luotonen and K. Altis, "World-Wide Web Proxies," Electronic Proceedings, *Proc. of the First International World-Wide Web Conf., 1994*,  
<http://www1.cern.ch/PapersWWW94/luotonen.ps> (February 1996).
  
- [4] M. Abrams, C.R. Standridge, G. Abdulla, S. Williams, and E.A. Fox, "Caching Proxies: Limitations and Potentials," *TR 95-12*, Department of Computer Science, Virginia Polytechnic Institute and State University, July 1995.

- [5] S. Glassman, "A caching relay for the World-Wide Web," *Proc. of the First International World-Wide Web Conf.*, Amsterdam: Elsevier, pp. 69-76, October 1994. Also available at <http://www1.cern.ch/PapersWWW94/steveg.ps>.
- [6] D.C. Lee, "Improving User Response Time for the World-Wide Web Using Pre-Fetch Data Caching," *Master's Thesis*, Virginia Polytechnic Institute and State University, 1996. Available at <http://fiddle.ee.vt.edu/~dlee/thesis/text.ps>.
- [6] T. Berners-Lee, L. Masinter, and M. McCahill, "Uniform Resource Locators (URL)," RFC-1738, CERN, Xerox Corp., University of Minnesota, December 1994. Available at <ftp://ds.internic.net/rfc/rfc1738.txt>.
- [7] R. Fielding, "Relative Uniform Resource Locators," RFC-1808, University of California, Irvine, June 1995. Available at <ftp://ds.internic.net/rfc/rfc1808.txt>.
- [8] D.M. Kristol, "A Proposed Extension Mechanism for HTTP," Internet-Draft, AT&T Bell Labs, January 1995. This is a working draft. Available at <ftp://ds.internic.net/internet-drafts/draft-kristol-http-extensions-00.txt>.

## **7.0 Author's Addresses**

David C. Lee

EE Graduate Student

Bradley Department of Electrical Engineering

347 Whittemore Hall

Blacksburg, Virginia 24061-0111

EMAIL: [dlee@vt.edu](mailto:dlee@vt.edu)

PHONE: (540) 231-8398

Dr. Scott F. Midkiff

Associate Professor

Bradley Department of Electrical Engineering

340 Whittemore Hall

Blacksburg, Virginia 24061-0111

EMAIL: [midkiff@vt.edu](mailto:midkiff@vt.edu)

PHONE: (540) 231-5190

## Appendix B. HTTP Redundancy Analysis

Redundancy in the HTTP protocol is frequently cited as an undesirable feature of HTTP [34-37]. A “back of the envelope” calculation of typical overhead is provided here. The values represent those collected from analysis (see Appendix C) of what seem to be the most common server and browser available, NCSA’s *httpd* server and Netscape Navigator’s browser [26, 27]. Note that the overhead of HTTP, as indicated by the characterization in Appendix C, is approximately two percent.

The average header size, including the client request and server response, as well as the average request length, is approximately 250 bytes. There are a number of (typically) required items, and these are the length of the data being transmitted, the modification date, the server response code, the type of request being made, and the request length. A conversion to binary, for the best compression, leads to the following.

- length — 4 bytes
- modification date — 4 bytes
- response code — 2 bytes
- request type — 1 byte
- request length — approximately 25 bytes on average



This results in 36 bytes of required data rather than the approximately 250 bytes needed by the current protocol. The percentage of unnecessary data transferred is calculated as follows.

$$1 - \frac{36}{250} = 86\% \quad (\text{Equation 1})$$

This comparison assumes that an infinite (very large) number of requests are made to eliminate the effects of the necessary protocol negotiation. This is not an accurate assumption, especially considering that this would be at least ten's of bytes for a responsive stateful protocol. This assumption biases the comparison in favor of the "new" stateful protocol over HTTP 1.0/1.1. With compression of the textual headers and inclusion of transactions per session, overhead is moved to the session rather than document access transaction. Session overhead can be significant, since a casual examination of our servers indicates that most users request under ten documents per session.

## **Appendix C. Web Characterization**

The following is a draft paper that is included to provide additional details about the characteristics of Web documents and traffic. There may be a more current document available than the draft presented here.

The draft paper has been reformatted to fit thesis constraints.

# World-Wide Web Characterization Study

David C. Lee

*dlee@vt.edu*

(540) 231-8398

Scott F. Midkiff

*midkiff@vt.edu*

(540) 231-5190

Virginia Polytechnic Institute and State University

The Bradley Department of Electrical Engineering

Blacksburg, Virginia 24061-0111

## Abstract

*This paper provides a simple characterization of the World-Wide Web in terms of average document sizes, average transmission times, character frequencies, Hypertext Transport Protocol header sizes, basic document type frequencies, and server product frequencies. The method used to obtain the data was to recursively obtain and collect statistical information on the documents referenced by the [www.yahoo.com](http://www.yahoo.com) Web document database. In the study, 15,000 nodes were visited with an average of six or more documents retrieved per node. The results may be of use to researchers studying methods to improve the Web or developers of client/server applications for the Web.*

## Introduction

This paper presents a characteristic study of the World-Wide Web, or the Web. These characteristics include the average size of Web documents, image, and other data types, the number of references in an average Web document, the average connection times, the server product types and frequencies, the transmitted data character frequencies, and so forth. Note that a high variance in the averages exists. Additionally, the Web is still in an explosive growth stage which means that the presented statistics may change significantly in the future.

The basic study was performed by recursively retrieving Web documents linked to by the [www.yahoo.com](http://www.yahoo.com) Web document database. This work was originally conducted as preliminary research for a MS Thesis examining Web pre-fetch catching schemes [1]. Some key results of the study are as follows.

- NCSA *httpd* [2] was the most common server (43.2 percent of total surveyed hosts),
- Hypertext Markup Language (HTML) [3] and inline images compose most of the documents surveyed, with 83.4 percent of total documents; however, these two document types only comprise 40.3 percent of total bytes,

- an average of 19.1 links (including inline images) per HTML document with 4.2 inline images per HTML document,
- and the average URL size (not including the protocol, server port, and server host name) was 21.4 characters.

## Experimental Setup

A recursive automated web browser, *StatFetcher*, was developed to gather data for the study. The Web browser included a simple Hypertext Transport Protocol (HTTP) [4] implementation, a HTML [3] parser module, and recursive control logic. The operation of the browser amounted to retrieving a document, extracting all the link references, retrieving the new documents, and so forth. A hard limit on the number of recursion levels could be specified by the researcher. A recursion level is the number of documents away from the starting, or root, document. In the reported study, the maximum recursion level was set at 15.

The protocol implementation conformed to HTTP version 1.0 [4]. The HTTP module provided the document type identification based on the **Content-Type** headers and document filename extensions when **Content-Type** headers were not available. Server product information was obtained from the **Server** header line. Note that only server product names were recorded and not specific version recursion information.

*StatFetcher* sent headers as shown in Table 1. The C programming language "%s" notation indicates where the Uniform Resource Locator (URL) [5] path reference was placed. From our observations, *StatFetcher* headers were roughly 200 to 1000 bytes shorter than typical browser headers. Character frequencies do not include the **User-Agent** header but otherwise include the client and server headers.

**Table 1.** *StatFetcher* HTTP Headers

GET %s HTTP/1.0<CR><LF>
Accept: /*<CR><LF>
User-Agent: StatFetcher/2.00<CR><LF>
<CR><LF>

A special HTML parser was built to parse the HTTP references encoded in the **href**, **img**, and **applet** elements. These elements represent document links, in-line images, and Web Java-based [6] applet applications. Relative URL references [7] were followed based on the URL of the current document. The parser ignored all other HTML elements and non-HTTP references and makes some attempts to correct HTML errors. Errors not corrected resulted in failed connections or HTTP non-200 response warnings. The HTTP-based links were retrieved in the order extracted, unless the recursion depth limit was reached. If the depth limit was reached, no links for the document would be

retrieved. It is important to note that no special timeout provisions were imposed by the software itself; timeouts were handled by the underlying operating system.

The first link that *StatFetcher* retrieved was the root index document of the Yahoo database, at the URL <http://www.yahoo.com/>. The Yahoo database is organized in a hierarchical fashion which supports the recursive nature of the study. The documents in Yahoo were included in the data obtained.

Some hosts and file types were deliberately excluded from the study. The exclusions were hosts with large Web document databases, references to Common Gateway Interface (CGI) based scripts [8], and references to HTTP search queries. As defined in HTTP 1.0, the search queries are URL's which contained a question mark ("?").

Accurate detection of duplicate HTML documents is important since a large number of duplicate documents can substantially skew the characterization. Originally, the duplicate detection method was straight-forward. The entire path and host information were stored and compared. This resulted in large disk space requirements and substantial search times as the data set grew in size. Currently, four items are recorded along with the host address and server port information. These include two CRC-16 values and a simple checksum that are calculated over the URL path reference. The CRC-16 generator polynomials are as follows.

$$x^{16} + x^{12} + x^5 + 1$$

$$x^{16} + x^{15} + x^2 + 1$$

The last item stored is the last ten characters of the URL path reference. All four values, as well as the host address and port number, must match in order for the URL to be considered a duplicate. This substantially reduces the data storage requirements while providing no false negatives, or no missed detection of duplicates, for hosts that are not multi-homed. That is, hosts that have more than one IP address for the server may have some duplicate documents recorded in the results. The number of such occurrences should be very low. Note that there is a chance that false positives, or inaccurate (false) detection of duplicates, may result. From a casual examination of the logging output, very few, if any, false positives resulted. Even if false positives occurred, they should not skew the statistics any more than a failed connection would.

## **Experimental Results**

*StatFetcher* was run on a Sun Microsystems Sparc 5 running the Solaris 2.4 operating system. The 10-Base-T Ethernet network in the Virginia Tech Information Systems Center 475 Whittemore Hall Laboratory was connected, through the campus FDDI



backbone, to the Internet via a T2 line. The data used to for these results are available on the Web at the following URL.

<http://fiddle.ee.vt.edu/~dlee/thesis/webstudy/v2.00/run2.html>

The study period lasted a total of 23 days, from January 16, 1996 at 9:20 AM to February 8, 1996 at 11:00 PM. It resulted in 167.8 hours, or approximately 7 days, of successful data transfers. The study included weekend and weekday hours. Several scheduled local network and electrical outages occurred during the study period. These outages resulted in temporary suspension of the data gathering process. Additionally, at least two observed partial 20-30 minute (non-local) backbone outages were observed during the study period. The data gathering process was not suspended during any non-local (observed or not observed) backbone outages.

A total of 15,000 Web servers were visited with 98,371 documents retrieved. This resulted in an average of 6.6 documents retrieved per server. A enforced limit of 50 documents per server was used and a specified maximum of 15,000 nodes could be visited. Potentially, 220,757 documents could have been retrieved. A total of 76,779, or 34.7 percent, of the potential documents were found to be duplicates. A total of 5,636, or 2.6 percent, of the attempts resulted in unsuccessful connection errors or non-HTTP

protocols. A total of 76,669, or 34.7 percent, of the potential requests were not made because more than 50 documents per server would have been retrieved.

The study contained a total of 4,165 clearly non-US hosts. Non-US hosts were defined as hosts that did not have a **gov**, **mil**, **net**, **com**, **edu**, or **us** domain and the recorded address was not a numerical IP address.

Of the successful requests, 1,295 responses were incorrect or resulted in an unexpected connection terminated. A total of 14,490 returned documents were the result of unsuccessful (non-200 HTTP response code) attempts. A non-200 response code typically means that the document specified was not available or incorrect. These errors are most probably the result of document maintenance oversights by the document author. 10,939, or 75.5%, of these non-200 response errors resulted in warning HTML documents generated by the server. These warning documents were recorded in the study's data set.

A total of 1.2 gigabytes of data was transferred over the network. The data included two percent of overhead (25 megabytes) and approximately 1.2 gigabytes of documents. Overhead data is defined to be HTTP headers. This is illustrated in Table 1. Note that the overhead data may vary by a percent, depending on what type of browser is used. Percentages given are percentages of total bytes transferred.

**Table 2.** Total Byte Overhead Breakdown of the Study

	NUMBER OF BYTES
Total Bytes	1,226,973,794 (100.0%)
Total Document Bytes	1,201,989,482 (98.0%)
Total Header Bytes	24,984,312 (2.0%)
Total Request Header Bytes	8,097,710 (0.7%)
Total Response Header Bytes	16,886,602 (1.4%)

Note that a significant amount of variance was found in the following data. This was not unexpected. Documents on the Web are of a variety of media and subject material and are subject to high variance in content. All the average statistics presented meet the 95 percent confidence interval test and can be considered good statistics. Detailed statistical information can be found at the URL mentioned above.

The average link capacity was determined to be 1,989.3 bytes per second. This relatively low capacity results from both non-US hosts being in the survey and the increasingly common severe congestion on the Internet. The average transfer time was 6.1 seconds with 1.3 seconds spent performing the connection to the remote host. An average of 0.4 seconds, from the 1.3 seconds, was used for Domain Name Service lookups.

Reported connection time averages loosely supports the NCSA *httpd* development group's testing of their HTTP 1.1 [9] **Keep-Alive** functionality. The NCSA developers experienced a 33 percent improvement in time response by keeping the connection open

for multiple requests [10]. The data from the current study shows that the average connection time is 21 percent. However, the data collected from previous studies correlated well with the NCSA testing. These previous studies showed that the average connection time was approximately 30 percent.

One specific HTML document characteristic that may be of interest is that an average document has 19.1 links referring to other documents. Inline images were 22 percent (4.2 links) of the references and 0.01 links, or 0.1 percent, referred applet documents. The rest of the references were to other documents types. The average URL request size, not including the host, port, and protocol type, was 21.4 characters.

Table 2 provides average breakdown of files by document type and Table 3 provides the average breakdown by file sizes by document type. The first column, in both tables, lists the document type and, in parenthesis and where applicable, the general HTTP Content-Type is given. The second column provides the total files and bytes for Table 3 and 4, respectively. The third column in Table 4 presents average file size information. Percentages given are the percentages of total number of files or bytes transferred.

The row entitled “HTTP non-200 responses code HTML responses” indicate HTML documents that were received from a server in response to a incorrect HTTP request. Inline images are images (GIF, JPEG, or any other image format) retrieved that are

referenced within a document through the HTML **IMG** language element. All other images are images retrieved that are referenced through the HTML **REF** anchor elements. Inline images are rendered by a browser with the document it is referred to by whereas other images will be displayed as a separated document. Note that the numbers for Virtual Reality Modeling Language (VRML) [11] and applet documents may be somewhat deceiving since several VRML and Java “how-to” Web sites were included in the study. These “how-to” sites typically have large collections of example documents. The last table entry, “other document types,” merely indicates that all other document types are included in this designation.

**Table 3.** Total File Characteristics of the Study

<i><b>DOCUMENT TYPE</b></i>	<i><b>NUMBER OF FILES</b></i>
Total number of files transferred	98,371 (100.0%)
HTML (text/html)	61,141 (62.2%)
VRML (text/vrml)	317 (0.3%)
HTTP non-200 response code HTML responses	10,939 (11.1%)
Other text types (text/*)	2,310 (2.3%)
Inline images (image/*)	20,823 (21.2%)
Non-inline images (image/*)	1,354 (1.4%)
Video (video/*)	108 (0.1%)
Audio (audio/*)	155 (0.2%)
Applets	375 (0.4%)
PDF (*.pdf)	37 (0.0%)
Postscript (*.postscript)	260 (0.3%)
Other document types (*.*)	550 (0.6%)

Table 3 shows that the categories of document types is fairly complete. From observations during the study run, most of the documents in the “other” category were miscellaneous binary documents such as programs and word processing documents. HTML documents and associated inline images are the predominate document types in the survey. The HTTP HTML responses can be discarded for the most part since they represent error conditions that occurred. These error responses were probably mostly due to errors in the source documents.

**Table 4.** Total Byte Characteristics of the Study

<i><b>DOCUMENT TYPE</b></i>	<i><b>NUMBER OF BYTES</b></i>	<i><b>AVERAGE SIZE</b></i>
Total number of bytes transferred	1,201,989,482 (100.0%)	12,218.9
HTML (text/html)	364,533,244 (30.3%)	5,962.2
VRML (text/vrml)	61,797,666 (5.1%)	194,945.2
HTTP non-200 response code HTML responses	2,575,757 (0.2%)	235.5
Other text types (text/*)	95,907,511 (8.0%)	41,518.4
Inline images (image/*)	120,660,770 (10.0%)	5,794.6
Non-inline images (image/*)	126,778,299 (10.5%)	93,932.4
Video (video/*)	124,947,374 (10.4%)	1,156,920.1
Audio (audio/*)	27,543,109 (2.3%)	177,697.5
Applets	850,458 (0.1%)	2,267.9
PDF (*.pdf)	17,068,472 (1.4%)	461,310.1
Postscript (*.postscript)	70,975,825 (5.9%)	272,983.9
Other document types (*/*)	188,350,997 (15.7%)	342,456.4

Table 4 provides a contrast for the information presented in Table 3. As can be seen from Table 4, HTML and inline image documents are typically short and only account for a combined 40.3 percent of total bytes transferred. Other document types are, on average, significantly larger, except the applet document type.

A total of 164 different server products were encountered in the study. Note that the original design of the study did not anticipate more than one Web server per host. *StatFetcher* did record hosts with multiple different servers but may not have kept entirely accurate totals for each server type. The margin of error should be insignificant. The numbers presented in Table 5 should be fairly close to what was actually discovered in the study. Table 5 presents the top ten server products that were encountered in the study.

**Table 5.** Top Ten Server Products Encountered in the Study

<i><b>SERVER NAME</b></i>	<i><b>NUMBER OF HOSTS (out of 15,000)</b></i>
1. NCSA <i>httpd</i> Server	6,474 (43.2%)
2. Apache	2,097 (14.0%)
3. CERN	1,995 (11.6%)
4. Netscape Communications Server <sup>1</sup>	1,011 (7.0%)
5. Netscape Commerce Server <sup>2</sup>	880 (5.8%)
6. WebStar	522 (3.5%)
7. WebSite	343 (2.3%)
8. MAC HTTP	219 (1.5%)
9. HTTPS	215 (1.4%)
10. No server specified	111 (0.7%)

<sup>1</sup>This total includes both the Netscape Communications Server and the Netsite Server.  
<sup>2</sup>This total includes both the Netscape Commerce Server and the Netsite Commerce server.

Character frequencies for each of the document types were also obtained. The character frequencies counted the occurrences of each different 8-bit binary combination in the transmitted data stream. These frequencies are available at the URL provided in the beginning of this section.

## **Conclusions**

The indications from this characterization is that overhead induced by connection time is more significant (21 percent) than those induced by HTTP headers (two percent). With the introduction and potentially wide-spread use of the HTTP 1.1 **Keep-Alive** header, HTTP headers themselves will become the most obvious reducible overhead. From a casual observation of the recent World-Wide Web related Internet Drafts, HTTP headers seem to be continuously expanding. Reducing the header overhead is one of the goals of the HTTP-NG effort [12].

Text-based HTML documents are the predominate document type with 62.2 percent of total documents and 30.3 percent of total bytes. All text document types comprise of 75.9 percent of total documents and 43.6 percent of total bytes. Implementation of simple on-the-fly compression schemes may reduce the size of an average text document by 40 to 70 percent [13]. Performing compression on text alone would provide a



significant data reduction for the Internet. Compressing other document types may also provide a tremendous increase in capacity for the Internet.

The Web characteristics presented in this paper will undoubtedly change over time.

Considering that the Web begun to dominate the NSFNET backbone traffic in April 1995 [14], the results discussed may help researchers curtail the rapid growth in Web network traffic.

## References

- [1] D.C. Lee, "Pre-Fetch Document Caching to Improve World-Wide Web User Response Time", *Master's Thesis*, Virginia Polytechnic Institute and State University, March 1996. Available at <http://fiddle.ee.vt.edu/~dlee/thesis/text.ps>.
- [2] NCSA Software Development Group, *httpd*, Version 1.5a, 1996. Available at [ftp://ftp.ncsa.uiuc.edu/Web/httpd/Unix/ncsa\\_httpd/httpd\\_1.5a](ftp://ftp.ncsa.uiuc.edu/Web/httpd/Unix/ncsa_httpd/httpd_1.5a).
- [3] T. Berners-Lee and D. Connolly, "Hypertext Markup Language: A Representation of Textual Information and Metainformation for Retrieval and Interchange,

Version 2," RFC-1866, CERN, November 1995. Available at  
<ftp://ds.internic.net/rfc/rfc1866.txt>.

- [4] T. Berners-Lee, R. Fielding, and H. Frystyk, "Hypertext Transfer Protocol - HTTP/1.0," MIT/LCS, Internet-Draft, October 1995. This is a working draft.  
Available at  
<ftp://ds.internic.net/internet-drafts/draft-ietf-http-v10-spec-04.txt>.
  
- [5] T. Berners-Lee, "Universal Resource Identifiers in WWW: A Unifying Syntax for the Expression of Names and Address of Objects on the Network as used in the World Wide Web," RFC-1630, CERN, June 1994. Available at  
<ftp://ds.internic.net/rfc/rfc1630.txt>.
  
- [6] Sun Microsystems Computer Corp., "The Java Language Specification, Version 1.0 Beta," October 1995.
  
- [7] R. Fielding, "Relative Uniform Resource Locators," RFC-1808, UC Irvine, June 1995. Available at <ftp://ds.internic.net/rfc/rfc1808.txt>.
  
- [8] NCSA Software Development Group, "The Common Gateway Interface,"  
<http://hoohoo.ncsa.uiuc.edu/cgi/overview.html> (January 1996).

- [9] R. Fielding, H. Frystyk, and T. Berners-Lee, "Hypertext Transport Protocol - HTTP/1.1," Internet-Draft, UCI, November 1995. This is a working draft. Available at <http://ds.internic.net/internet-drafts/draft-ietf-http-v11-spec-00.txt>.
- [9] NCSA *httpd* Server Development Team, "Features of NCSA HTTPD 1.5," <http://hoohoo.ncsa.uiuc.edu/docs/features-1.5.html#KeepAlive> (January 1996).
- [11] G. Bell, A. Parisi, and M. Pesce, "The Virtual Reality Modeling Language, Version 1.0 Specification," Web document, May 1995, <http://vrml.wired.com/vrml-tech/vrml10-3.html> (August 1995).
- [12] S.E. Spero, "Next Generation Hypertext Transport Protocol," UNC, Internet Draft, March 1995. This is a working draft. Available at <http://sunsite.unc.edu/ses/ng-notes.txt>.
- [13] T.C. Bell, J.G. Cleary, and I.H. Witten, *Text Compression*, Englewood Cliffs: Prentice Hall, 1990.

- [14] J.E. Pitkow, "GVU Center NSFNET Statistics," Georgia Institute of Technology Graphics, Visualization, and Usability Center,  
<http://www.cc.gatech.edu/gvu/stats/NSF/merit.html> (August 1995).

## Appendix D. Experimental Results

This appendix presents experimental results. Table 5 contains the parameters, or factors, and their settings. The first column is the designated identifier for each setting. The row labeled **AVERAGE** is the average for all 16 settings. The row labeled **linkonly** is the data set associated with document link pre-fetching where each link is effectively given an equal access probability.

**Table 5.** Experimental Results Factors

Designation	Depth	Bandwidth	Channel	Cache Size
0000	0	1000	2	50
0001	0	1000	2	1000
0010	0	1000	9	50
0011	0	1000	9	1000
0100	0	0	2	50
0101	0	0	2	1000
0110	0	0	9	50
0111	0	0	9	1000
1000	10	1000	2	50
1001	10	1000	2	1000
1010	10	1000	9	50
1011	10	1000	9	1000
1100	10	0	2	50
1101	10	0	2	1000
1110	10	0	9	50
1111	10	0	9	1000
AVERAGE	N/A	N/A	N/A	N/A
linkonly	10	0	9	1000

Table 6 shows the hit ratio results. **Hit Ratio** (column two) is composed of **Partial** and **Prefetch** (columns three and four, respectively) and indicates all pre-fetch cache hits. **Partial** (column three) represents partial pre-fetch cache hits, i.e., hits where the requested document was being pre-fetched at the time of the user request. **Prefetch** (column four) represents completely pre-fetched documents. **Total Hit Ratio** (column 6) is the combination **Hit Ratio** and **Partial** (columns 2 and 3) of **Temp** and **Hit Ratio** (columns 4 and 5, respectively). It represents the total cache hit ratio for all caches. **Temp** (column 5) represents the post-cache (temporary cache) hit rates.

**Table 6.** Hit Ratio Results

Designation	Hit Ratio	Partial	Prefetch	Temp	Total Hit Ratio
0000	0.431	0.282	0.453	0.26	0.713
0001	0.502	0.162	0.401	0.263	0.664
0010	0.477	0.355	0.382	0.263	0.832
0011	0.665	0.068	0.471	0.263	0.733
0100	0.45	0.203	0.392	0.262	0.653
0101	0.49	0.123	0.35	0.263	0.613
0110	0.59	0.13	0.457	0.263	0.72
0111	0.625	0.091	0.454	0.263	0.716
1000	0.417	0.247	0.401	0.263	0.664
1001	0.608	0.162	0.507	0.263	0.77
1010	0.494	0.182	0.414	0.263	0.676
1011	0.813	0.067	0.617	0.263	0.88
1100	0.42	0.134	0.292	0.262	0.554
1101	0.461	0.097	0.296	0.263	0.558
1110	0.517	0.079	0.333	0.263	0.596
1111	0.622	0.072	0.432	0.262	0.694
AVERAGE	0.536	0.142	0.416	0.262	0.678
linkonly	0.602	0.058	0.398	0.263	0.66

Table 7 presents user response time results. The effective response time (column two) is user response time with pre-fetch caching. The control response time (column three) is the response time without any pre-fetch caching but with temporary (post) caching. The ERT/CRT column indicates the ratio of effective response time versus control response time. This ratio indicates the performance improvement (for values less than 1) or degradation (for values greater than 1).

**Table 7.** Response Time Results

Designation	Effective RT	Control RT	ERT/CRT
0000	8.078	11.193	0.722
0001	8.58	11.082	0.774
0010	5.225	11.088	0.471
0011	7.94	11.079	0.717
0100	0.487	0.35	1.391
0101	0.418	0.345	1.212
0110	0.579	0.349	1.659
0111	0.357	0.344	1.038
1000	9.275	11.074	0.838
1001	5.117	11.077	0.462
1010	6.184	11.079	0.558
1011	3.884	11.078	0.351
1100	0.579	0.336	1.723
1101	0.961	0.343	2.802
1110	15.911	0.346	45.986
1111	0.997	0.354	2.816
AVERAGE	4.661	5.72	0.815
linkonly	0.462	0.354	1.305

Table 8 presents results that characterize the cache insertion and deletion algorithm. The second column indicates the average total number of files retrieved, the third column

provides the average number of unnecessary files retrieved, and the fourth column reports the average number of files that were deleted. The average number of files that an user viewed is given in column five. This includes all requests, including post-cache hits. Columns six and seven provide the percentage of unnecessary and deleted files, as compared to total number of files retrieved, respectively. Negative values of unnecessary files are not errors. Negative values result because of post-cache hits.

**Table 8.** Transfer Results Based on Files

Designaton	Total Files	Unnec.	Deleted	Total Used	% Unnec.	% Del.
0000	38.8	11.4	14.367	27.4	0.294	0.370
0001	26.4	-1	0	27.4	-0.038	0.000
0010	47.333	19.933	26.333	27.4	0.421	0.556
0011	26.4	-1	0	27.4	-0.038	0.000
0100	39.733	12.333	14.933	27.4	0.310	0.376
0101	27.967	0.567	0	27.4	0.020	0.000
0110	43.267	15.867	16.7	27.4	0.367	0.386
0111	26.5	-0.9	0	27.4	-0.034	0.000
1000	63.367	35.967	37.3	27.4	0.568	0.589
1001	69.067	41.667	0	27.4	0.603	0.000
1010	693.033	665.633	661.3	27.4	0.960	0.954
1011	152.7	125.3	43.3	27.4	0.821	0.284
1100	71.833	44.433	43.567	27.4	0.619	0.607
1101	325.667	298.267	282.967	27.4	0.916	0.869
1110	937.4	910	908.467	27.4	0.971	0.969
1111	154.433	127.033	86.067	27.4	0.823	0.557
AVERAGE	171.494	144.094	133.456	27.4	0.840	0.778
linkonly	40.767	13.367	0.067	27.4	0.328	0.002



Table 9 presents the same type of information as Table 8, except that the average values here are the bytes sent and received. Column two shows the average total bytes in the cache upon experiment termination, which is related to the other columns as follows.

$$\text{Total Bytes} = \text{Cache} - \text{Deleted} + \text{User R/W} \quad (\text{Equation 2})$$

The variable named Cache consists of the values in column six, Deleted represents values in column seven, and User R/W represents the values in column five. Column three shows the average total number of user bytes read from the network to service user trace requests, including HTTP headers. Column four indicates the average bytes written (HTTP requests) to service user trace requests. Column five, **User R/W**, is composed of column three and four. Column six shows the average number of bytes read from the cache to fulfill user requests, and the last column, number seven, provides the average number of bytes deleted from the cache.

**Table 9. Transfer Results Based on Bytes**

Designation	Total Bytes	User Read	User Write	User R/W	Cache	Deleted
0000	494862.16	401629.84	4184.3	405814.1	280845.6	197367.2
0001	351314.44	421294.13	4259.7	425553.8	235017.1	0
0010	559837.25	285640.94	3676.5	289317.4	171708.9	373989.4
0011	350958.91	421355	4259.7	425614.7	246008.8	0
0100	509929.72	403060.19	4193.7	407253.9	280882.3	208224.1
0101	364446.28	421355	4259.7	425614.7	247663	0
0110	591799.75	421486.56	4254	425740.6	249880	271071.5
0111	352095.28	406164	4178.1	410342.1	227073.4	0
1000	774823.31	413738.78	4232.2	417971	254445.9	447729.1
1001	741842.88	421361.59	4259.7	425621.3	328933.8	0
1010	6362063	256249.14	3509.4	259758.5	139551.6	6185431
1011	2581541.5	421361.59	4259.7	425621.3	360430.7	1456403
1100	861862.56	418312.47	4248.7	422561.2	222594.1	522772.3
1101	13514172	421361.59	4259.7	425621.3	225683.7	12383788
1110	36674916	419726.88	4244	423970.9	187567.3	36348184
1111	4757834.5	420155.03	4254.5	424409.5	219655.3	3517613
AVERAGE	4365240.5	398390.81	4158.35	402549.2	242371.3	3869536
linkonly	1148856.9	412412.5	2839.8	415252.3	209178.1	5599.867

# Appendix E. Statistical Analysis of Results

This appendix provides the results from the SAS [54] analysis of the data presented in Appendix D. SAS was set-up for multivariate factorial testing using its general linear model. The classes are the four parameters, or factors, presented Section 6.1.2.

The first set of data is a comparison conducted using the average response time as the dependent variable. The second set uses the cache hit ratios as the dependent variable. The sources are the factors and combinations thereof. Low “Pr > F” values indicate high correlation.

```

                                The SAS System
                                10:17 Thursday, September 7, 1995

                                General Linear Models Procedure
                                Class Level Information

                                Class      Levels      Values
                                SPEED        2        0 1000
                                DEPTH        2        0 10
                                CHANNEL      2        1 8
                                CACHESIZ    2      50000 1000000

                                Number of observations in data set = 480

                                General Linear Models Procedure

                                Dependent Variable: LNAVGTPR

                                Source          DF      Sum of Squares      F Value      Pr > F
                                Model            15      822.82842227      352.99      0.0001
                                Error           464      72.10627688
                                Corrected Total 479      894.93469915

                                R-Square          C.V.          LNAVGTPR Mean
                                0.919428          46.59603          0.84601618

                                Source          DF      Type I SS      F Value      Pr > F
                                SPEED            1      495.40631065      3187.91      0.0001
                                DEPTH            1      30.71471665      197.65      0.0001
                                SPEED*DEPTH      1      78.06416719      502.34      0.0001
                                CHANNEL           1      9.52435374      61.29      0.0001
                                SPEED*CHANNEL     1      45.01046540      289.64      0.0001

```

DEPTH*CHANNEL	1	15.45734350	99.47	0.0001
SPEED*DEPTH*CHANNEL	1	21.56645503	138.78	0.0001
CACHESIZ	1	21.57468862	138.83	0.0001
SPEED*CACHESIZ	1	9.01259006	58.00	0.0001
DEPTH*CACHESIZ	1	18.91349512	121.71	0.0001
SPEED*DEPTH*CACHESIZ	1	0.00118912	0.01	0.9303
CHANNEL*CACHESIZ	1	15.72123876	101.17	0.0001
SPEED*CHANNE*CACHESI	1	31.47647820	202.55	0.0001
DEPTH*CHANNE*CACHESI	1	16.88245261	108.64	0.0001
SPEE*DEPT*CHAN*CACHE	1	13.50247763	86.89	0.0001

Source	DF	Type III SS	F Value	Pr > F
SPEED	1	495.40631065	3187.91	0.0001
DEPTH	1	30.71471665	197.65	0.0001
SPEED*DEPTH	1	78.06416719	502.34	0.0001
CHANNEL	1	9.52435374	61.29	0.0001
SPEED*CHANNEL	1	45.01046540	289.64	0.0001
DEPTH*CHANNEL	1	15.45734350	99.47	0.0001
SPEED*DEPTH*CHANNEL	1	21.56645503	138.78	0.0001
CACHESIZ	1	21.57468862	138.83	0.0001
SPEED*CACHESIZ	1	9.01259006	58.00	0.0001
DEPTH*CACHESIZ	1	18.91349512	121.71	0.0001
SPEED*DEPTH*CACHESIZ	1	0.00118912	0.01	0.9303
CHANNEL*CACHESIZ	1	15.72123876	101.17	0.0001
SPEED*CHANNE*CACHESI	1	31.47647820	202.55	0.0001
DEPTH*CHANNE*CACHESI	1	16.88245261	108.64	0.0001
SPEE*DEPT*CHAN*CACHE	1	13.50247763	86.89	0.0001

Dependent Variable: ISINPHIT

Source	DF	Sum of Squares	F Value	Pr > F
Model	15	4.01190776	20.47	0.0001
Error	464	6.06337754		
Corrected Total	479	10.07528530		

R-Square	C.V.	ISINPHIT Mean
0.398193	26.19929	0.43632369

Source	DF	Type I SS	F Value	Pr > F
SPEED	1	1.02134018	78.16	0.0001
DEPTH	1	0.00166121	0.13	0.7216
SPEED*DEPTH	1	0.71541797	54.75	0.0001
CHANNEL	1	0.50746196	38.83	0.0001
SPEED*CHANNEL	1	0.11365920	8.70	0.0033
DEPTH*CHANNEL	1	0.03549789	2.72	0.1000
SPEED*DEPTH*CHANNEL	1	0.03369737	2.58	0.1090
CACHESIZ	1	0.46813266	35.82	0.0001
SPEED*CACHESIZ	1	0.23676751	18.12	0.0001
DEPTH*CACHESIZ	1	0.43433003	33.24	0.0001
SPEED*DEPTH*CACHESIZ	1	0.05553879	4.25	0.0398
CHANNEL*CACHESIZ	1	0.33524997	25.66	0.0001
SPEED*CHANNE*CACHESI	1	0.03225647	2.47	0.1168
DEPTH*CHANNE*CACHESI	1	0.00324944	0.25	0.6183
SPEE*DEPT*CHAN*CACHE	1	0.01764709	1.35	0.2458

Source	DF	Type III SS	F Value	Pr > F
SPEED	1	1.02134018	78.16	0.0001
DEPTH	1	0.00166121	0.13	0.7216
SPEED*DEPTH	1	0.71541797	54.75	0.0001
CHANNEL	1	0.50746196	38.83	0.0001
SPEED*CHANNEL	1	0.11365920	8.70	0.0033
DEPTH*CHANNEL	1	0.03549789	2.72	0.1000
SPEED*DEPTH*CHANNEL	1	0.03369737	2.58	0.1090
CACHESIZ	1	0.46813266	35.82	0.0001

SPEED*CACHESIZ	1	0.23676751	18.12	0.0001
DEPTH*CACHESIZ	1	0.43433003	33.24	0.0001
SPEED*DEPTH*CACHESIZ	1	0.05553879	4.25	0.0398
CHANNEL*CACHESIZ	1	0.33524997	25.66	0.0001
SPEED*CHANNE*CACHESI	1	0.03225647	2.47	0.1168
DEPTH*CHANNE*CACHESI	1	0.00324944	0.25	0.6183
SPEE*DEPT*CHAN*CACHE	1	0.01764709	1.35	0.2458

General Linear Models Procedure  
Multivariate Analysis of Variance

Dependent Variable: LNAVGTPR

Source	DF	Type III SS	F Value	Pr > F
SPEED	1	495.40631065	3187.91	0.0001
DEPTH	1	30.71471665	197.65	0.0001
SPEED*DEPTH	1	78.06416719	502.34	0.0001
CHANNEL	1	9.52435374	61.29	0.0001
SPEED*CHANNEL	1	45.01046540	289.64	0.0001
DEPTH*CHANNEL	1	15.45734350	99.47	0.0001
SPEED*DEPTH*CHANNEL	1	21.56645503	138.78	0.0001
CACHESIZ	1	21.57468862	138.83	0.0001
SPEED*CACHESIZ	1	9.01259006	58.00	0.0001
DEPTH*CACHESIZ	1	18.91349512	121.71	0.0001
SPEED*DEPTH*CACHESIZ	1	0.00118912	0.01	0.9303
CHANNEL*CACHESIZ	1	15.72123876	101.17	0.0001
SPEED*CHANNE*CACHESI	1	31.47647820	202.55	0.0001
DEPTH*CHANNE*CACHESI	1	16.88245261	108.64	0.0001
SPEE*DEPT*CHAN*CACHE	1	13.50247763	86.89	0.0001

Error	464	72.10627688
-------	-----	-------------

Dependent Variable: ISINPHIT

Source	DF	Type III SS	F Value	Pr > F
SPEED	1	1.02134018	78.16	0.0001
DEPTH	1	0.00166121	0.13	0.7216
SPEED*DEPTH	1	0.71541797	54.75	0.0001
CHANNEL	1	0.50746196	38.83	0.0001
SPEED*CHANNEL	1	0.11365920	8.70	0.0033
DEPTH*CHANNEL	1	0.03549789	2.72	0.1000
SPEED*DEPTH*CHANNEL	1	0.03369737	2.58	0.1090
CACHESIZ	1	0.46813266	35.82	0.0001
SPEED*CACHESIZ	1	0.23676751	18.12	0.0001
DEPTH*CACHESIZ	1	0.43433003	33.24	0.0001
SPEED*DEPTH*CACHESIZ	1	0.05553879	4.25	0.0398
CHANNEL*CACHESIZ	1	0.33524997	25.66	0.0001
SPEED*CHANNE*CACHESI	1	0.03225647	2.47	0.1168
DEPTH*CHANNE*CACHESI	1	0.00324944	0.25	0.6183
SPEE*DEPT*CHAN*CACHE	1	0.01764709	1.35	0.2458

Error	464	6.06337754
-------	-----	------------

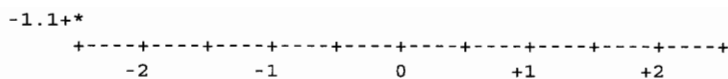
Univariate Procedure

Variable=TPR

Moments

N	480	Sum Wgts	480
Mean	0	Sum	0
Std Dev	0.387988	Variance	0.150535
Skewness	0.466422	Kurtosis	1.490105
USS	72.10628	CSS	72.10628
CV	.	Std Mean	0.017709





Variable=PHIT

#### Moments

N	480	Sum Wgts	480
Mean	0	Sum	0
Std Dev	0.11251	Variance	0.012658
Skewness	-0.37717	Kurtosis	-0.09255
USS	6.063378	CSS	6.063378
CV	.	Std Mean	0.005135
T:Mean=0	0	Pr> T	1.0000
Num ^= 0	480	Num > 0	268
M(Sign)	28	Pr>= M	0.0120
Sgn Rank	3036	Pr>= S	0.3185
W:Normal	0.966162	Pr<W	0.0001

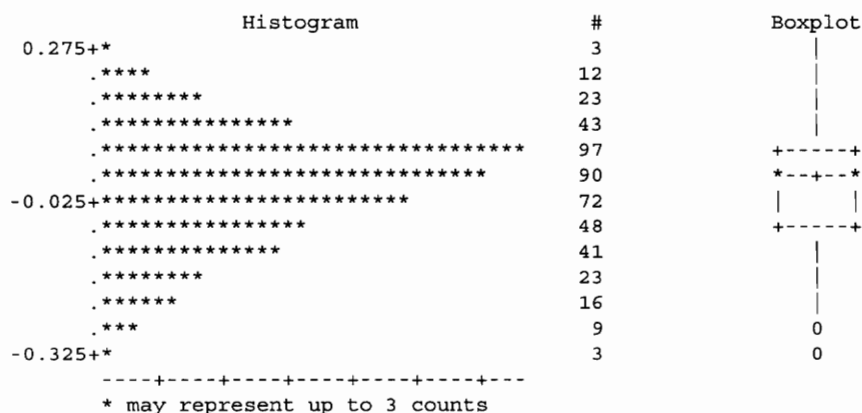
#### Quantiles (Def=5)

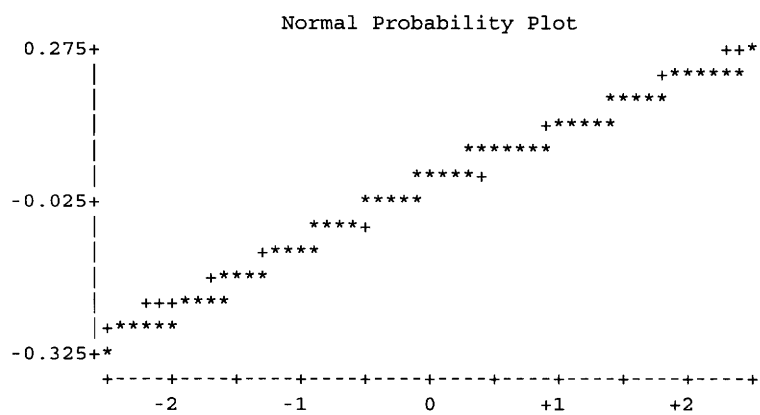
100% Max	0.261379	99%	0.228501
75% Q3	0.0723	95%	0.175108
50% Med	0.013937	90%	0.129272
25% Q1	-0.06974	10%	-0.15514
0% Min	-0.30618	5%	-0.20183
		1%	-0.29606
Range	0.567563		
Q3-Q1	0.142041		
Mode	0.101619		

#### Extremes

Lowest	Obs	Highest	Obs
-0.30618 (	276)	0.228501 (	405)
-0.30618 (	275)	0.229979 (	406)
-0.30618 (	274)	0.261379 (	28)
-0.29606 (	261)	0.261379 (	29)
-0.29606 (	259)	0.261379 (	30)

Variable=PHIT







## VITA

David C. Lee was born on September 19, 1972. He graduated from Nitro High School in 1990 and finished his Bachelor of Science in Computer Engineering at Virginia Tech in 1994. He was a recipient of the Marshall Hahn Engineering Scholarship in 1990 and worked as a Co-operative Education student with Lexmark International in Lexington, Kentucky. He has been a Graduate Research Assistant with the Southeastern University and College Coalition for Engineering Education (SUCCEED) since August 1994.

He has been a member of IEEE since January 1993, serving as Branch Chairman of the IEEE Student Branch at Virginia Tech in May 1993 to August 1994. Additionally, he has been a member of Eta Kappa Nu since January 1994. He was also President of the Bradley Department of Electrical Engineering's Student Advisory Council in August 1993 to May 1994. He has also served as the Student Representative for IEEE Region 3 since May 1994. He was awarded the 1993-1994 Larry K. Wilson IEEE Regional Student Activities Award for Region 3 for his involvement in bringing the IEEE Region 3 Southeast Conference to Virginia Tech in 1997.

A handwritten signature in black ink, reading "David C. Lee". The signature is written in a cursive, flowing style.