# Alex - a Global Filesystem

*Vincent Cate*

*vac@cs.cmu.edu*

*School of Computer Science*
*Carnegie Mellon University*
*Pittsburgh, Pennsylvania 15213-3890*

## Abstract

The Alex filesystem provides users and applications transparent read access to files in Internet anonymous FTP sites. Today there are thousands of anonymous FTP sites with a total of a few million files and roughly a terabyte of data. The standard approach to accessing these files involves logging in to the remote machine. This means that an application can not access remote files and that users do not have any of their aliases or local tools available when connected to a remote site. Users who want to use an application on a remote file must first manually make a local copy of the file. Not only is this inconvenient, it creates two more problems. First, there is no mechanism for automatically updating this local copy when the remote file changes. The users must keep track of where they get their files from and check to see if there are updates, and then fetch these. Second, many different users at the same site may have made copies of the same remote file, thus wasting disk space.

Alex addresses the problems with the above approach while maintaining compatibility with the existing FTP protocol so that the large collection of currently available files can be accessed. To get reasonable performance, long term file caching must be used. Thus consistency must be addressed. Traditional solutions to the cache consistency problem do not work in the Internet FTP domain: callbacks are not an option as the FTP protocol has no provisions for this, and polling over the Internet is slow. Fortunately, the usage of these files is also not traditional and lends itself to a new approach. Alex relaxes file cache consistency semantics, on a per file basis, and uses special caching algorithms that take into account the properties of the files and of the network. This approach allows a simple stateless filesystem to scale to the size of the Internet.

This research was sponsored in part by The Defense Advanced Research Projects Agency, Information Science and Technology Office, under the title “Research on Parallel Computing”, ARPA Order No. 7330, issued by DARPA/CMO under Contract MDA972-90-C-0035 and in part by the National Science Foundation under grant number ECD-8907068. The views and conclusions contained in this document are those of the author and should not be interpreted as representing the official policies, either expressed or implied, of the U.S. Government.

# 1. Introduction

Today it is possible to access a tremendous amount of data in FTP archives on the Internet. However, the current method of accessing this data is primitive. Users must run the FTP program and then login to the remote machine. Once they have done this, they can not use any of their local tools or aliases. Applications on a user's machine can not access files on remote machines, the user must manually copy them first. If the remote file changes the user must find out somehow and make another copy. Since each user is making copies, there may be many copies of a file at a given site. Since a user can not easily tell if their own copy of a file is out of date they may use an old version of a file. Thus the existing mechanism is cumbersome, slow, inefficient, and provides only limited access to remote data.

To address this problem, and to explore some ideas about wide area filesystems, I am designing and implementing a filesystem called **Alex**. The goal of this system is to let users access files in FTP sites all over the world just as they access files on their local system. Ideally, both the way it is used and its performance should match that of a local filesystem.

Providing transparent access to remote sites requires solutions to the problems of naming, heterogeneity and performance. While caching is the key to performance, this leads to the additional problem of cache consistency. I will describe how these problems are addressed in Alex.

The name **Alex** comes from the ancient Library of Alexandria. Alexandria gathered information from around the world into one easy to access location. Alex does an analogous thing in a very modern way.

In this paper, I describe the Alex architecture, how users interact with it, and what they need to understand to use it. I then describe the current NFS server implementation. After this I discuss related systems. Next I describe some of the useful and novel applications that can be built on top of this filesystem. I conclude with the current status of Alex, some conclusions, and my research agenda.

# 2. User View of Alex

One of my goals was to make Alex easy to use and think about. The hope is to provide a tool with a clean and simple abstraction.

## 2.1. Standard Filesystem Interface

To a user or application, Alex is just a normal filesystem. Any command that works on local files will work on Alex files. Since Alex is a filesystem, nothing needs to be recompiled and no libraries are changed. Thus, users can apply all of their existing skills and tools for using files.

## 2.2. Naming

The user sees a filesystem with a hierarchical global name space. At the top level (**/alex**) there are top-level Internet domains like "**edu**", "**com**", "**uk**", and "**jp**". Each component of the hostname becomes a directory name as shown in Figure 1. Then the remote path is added at the end. If the user does a '**ls /alex/edu/berkeley**" he sees some machine names such as "**ucbvax**" and "**sprite**" and some directories on **berkeley.edu**. From the "**ls**" it is not clear what is where. Alex paths do not use any special characters to delineating the host name.

**Figure 1. Example Alex Name**

## 2.3. Control of Consistency

By default, the system guarantees to the user that the only updates that might not yet be reflected locally are ones that have happened in the last 5% of the reported age of the file. For example, if the user sees a file as being 20 hours old the system promises that the data given to the user will be at most 1 hour stale. If a file is shown as being last modified 20 days ago, the file may be up to 1 day stale. There is a maximum of 1 month and a minimum of 5 minutes. The user can override these rules and ask for an update of any directory at any time.

# 3. Alex Implementation

As shown in Figure 2, Alex is currently implemented as an NFS server. Alex uses the FTP protocol [Postel85] to talk to Internet FTP sites. Machines on the local area net use the NFS protocol [Nowicki89] to talk to the Alex server. NFS was chosen because it makes it trivial to add Alex to a wide range of machines. On most machines, adding Alex just involves using a simple command called mount. It is so easy, that a user who has none of my software, and is just talking with me on the phone, can start using Alex in about a minute. AFS [Howard87] scales to more clients per server but for the expected number of simultaneous users it was felt that NFS would work fine. Another motivation for choosing NFS was that it is simple and stateless. Because NFS is stateless it is fine if Alex dies and restarts (for example, during installation of a new version of the server). Since this takes only a couple of seconds, none of the clients notice.

## 3.1. Background

A few important characteristics of the Alex domain motivated my design choices. Key to the Alex approach is using existing FTP sites without any change to the FTP protocol. This determines a number of aspects of the domain.

FTP, like AFS [Howard87], gives us whole file transfer. The FTP protocol does not do callbacks as does AFS. Thus, Alex is fully responsible for checking the consistency of its cache with the FTP site. The FTP protocol returns directory information as a unit (not one file at a time). This, like whole file transfer, reduces the number of times you pay the network latency cost. This is a big performance win in a wide area network as big as the Internet.

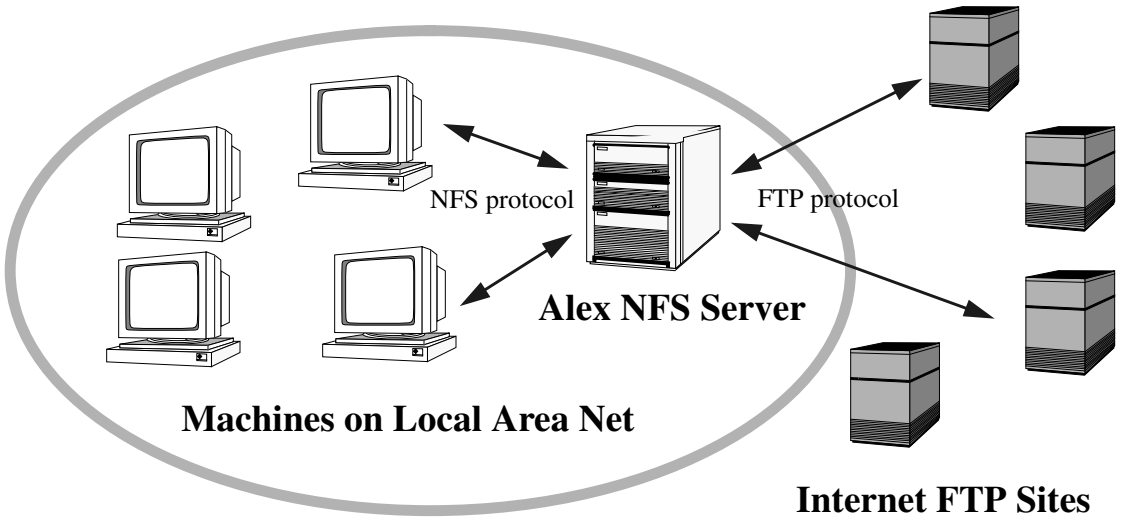


**Figure 2. Alex is an NFS Server**

Also important are characteristics of the domain FTP operates in. This domain is truly heterogeneous. While FTP hides some of this it also exposes plenty. Both the syntax for specifying names and the directory information returned varies among the different types of FTP sites. Another problem is that a noticeable percentage of the FTP sites are either down or inaccessible at any one time.

The type of data stored in FTP sites and the access patterns for this data are different from that of normal user files. The files change much less often than do normal files. Another important aspect of this domain is that users frequently tolerate using stale copies of files.

## 3.2. Naming

A path name in Alex has three parts. The first is "/alex". The second part is an Internet host name, with the most significant piece first (such as "edu") and the "."s replaced by "/"s. The third part is the path on the remote machine relative to the ftp starting directory on that machine. An example path is **/alex/edu/berkeley/pub/virus.patch**.

To resolve a name Alex first matches with the longest matching hostname it knows about, in a fashion similar to Sprite's prefix tables [Welch86]. Next, it sees if the rest of the path works. If it does not, Alex will create a possible hostname and call a nameserver. This approach seems to work almost everywhere. However, there is an ambiguity in the above name in that **pub** in is a directory on **berkeley.edu** and there is a **pub.berkeley.edu** (which both map onto the same name in Alex). Fortunately this does not happen often, in fact, pub.berkeley.edu is the only site found so far for which this does not work (and it does allow FTP anyway).

One problem with this type of naming is that a user who has set up his account to search the current directory for commands before other places such as "/bin" gets very poor performance at times. For example, if such a user wants to run "ls" while in /alex/edu/berkeley the shell will check to see if there is a /alex/edu/berkeley/ls. This causes the Alex server to check for a host called "ls.berkeley.edu" (assuming there was no "ls" file). Doing this takes a call to a remote nameserver, which is expensive. As long as they know this, users can set up their search path to avoid this problem (they should also change their path for security reasons).

## 3.3. Caching

To get reasonable performance a number of types of information are cached in various ways. The most important of these are listed here:

1 **Machine names** - Alex caches machine names in RAM so it can quickly resolve file names (saving calls to the nameserver).

2 **Open FTP connections** - Setting up an FTP connection takes several seconds. Once Alex has a connection it keeps it open as long as the remote site allows. Sometimes fetching a second small file from the same site is so fast that it looks as if it had been in the file cache.

3 **Type of FTP site** - Alex needs to know the type of FTP site it is talking to in order to specify names of files and directories in the appropriate syntax.

4 **Time zone information** - FTP sites report file modification times in their local timezone. NFS reports times in Greenwich time. Alex needs to know the timezone of the FTP site to make the conversion. To get the time zone of a machine may take two calls to standard Internet servers on that machine ("daytime" and "time").

5 **Directory information and files** - Copies of remote files are stored as Unix files. Directory information is parsed and the parsed information is stored as a Unix file.

6 **Failures of certain types** - Certain types of failures are cached for certain periods. If the nameserver says that it does not know of some host Alex caches that fact (since the nameserver does not cache failures). Also, if a host rejects anonymous access, Alex caches that fact. If the FTP cannot connect it caches that for a shorter period. Caching failure information is very important for good performance. Some failures are detected with timeouts, which can take a long time. Also, NFS retries can occur while waiting for a timeout from some FTP site. If the failures were not cached, each of the NFS retries would cause another long attempt to contact the FTP site.

## 3.4. Directories

Alex organizes directories to use disks more efficiently than a normal Unix filesystem. When a user does an "ls -F" or any operation which inspects the meta-data for each file in a directory, the Unix filesystem requires a disk seek for each file whose meta-data is not already cached in RAM. Since Alex stores all the directory information together sequentially on the disk, accesses to it are much faster. This is simple to do since there are no "hard links" to deal with. We don't actually store any details of disk allocation, so much of normal meta-data information is not needed. Inlining all file information only about doubles the directory size.

## 3.5. Handling Heterogeneity

There are a three ways in which heterogeneity affects Alex. These are specifying files and directory names, parsing the directory information returned, and deciding to transfer a file in binary or text mode.

Alex determines the name syntax an FTP site uses by its response to the "**pwd**" command and other commands. Once this information is known, it is straightforward to specify files in the format that the FTP site wants. Alex caches this information so that it only needs to be figured out the first time or when a problem is recognized because the site changed types (e.g. switched from VMS to Unix).

The FTP "**dir**" command returns directory information. On Unix machines this is the result of an "**ls -l**". Similar information is returned for other operating systems. Alex uses this to create all of the information for the NFS meta-data request. Alex can parse directory information from a variety of machine types (some examples are shown in Figure 3). File sizes and dates are used to determine the consistency of locally cached data. Thus, Alex, as well as the user, requires this information. Some systems, such as VMS FTP sites, give no indication of the size of a file. For these sites, Alex can only give the correct size of a file after it has been fetched. In the case of IBM VM/CMS systems, Alex can only estimate the size of a file before it is fetched (CMS provides only a maximum record size and number of records). Despite these difficulties, Alex hides the heterogeneity so well that a user can "**cd**" to a VMS machine and never know it.

**DEC VMS:**

**00README.TXT;7 6 9-APR-1991 18:14 [ANONYMOUS] (RWED,RE,RE,RE)**

**PUB.DIR;1 8 18-OCT-1990 07:20 [ANONYMOUS] (RWED,RE,RE,RE)**

**Unix:**

**dr-xr-xr-x 3 system 320 Jul 16 14:25 0.7alpha**

**-rw-rw-rw- 1 system 853 Jul 16 13:41 directory**

**IBM VM/CMS**

**FUSION BIBLIO4 V 80 2219 45 11/25/91 7:00:07 FUSION**

**FUSION BIBLIO5 V 78 132 3 11/27/91 10:01:11 FUSION**

**Figure 3. Heterogeneous Directory Information**

The decision to transfer data in "binary" or "text" format is made as follows. On Unix machines we always use binary (they all seem to use ASCII). On others, we use text unless the filename extension indicates that the file is one of a few types, such as a ".gif" file, that need to be transferred as binaries.

## 4. Related Work

There are many related systems. **Prospero** [Neuman89] and **Jade** [Rao91] replace libraries to create a "virtual filesystem". The problem with this approach is that applications must use these new libraries. In Prospero this means re-compiling with the new libraries. Jade replaces SunOS dynamic libraries, so applications on Suns that used dynamic libraries can access remote files via Jade. In this approach there will always be many applications that do not work with remote files. Prospero and Jade both have user specific name spaces, unlike Alex which has a single global namespace. Alex's canonical naming allows one user to say, "look around in **/alex/org/eff/ftp**" without fear of confusion.

Several applications give users easy access to remote files from within the application. The best example of this is **ange-ftp** [Norman92] which works from within GNU Emacs. **Symbolics Lisp machines** [Pearlmutter92] provided a similar function for a Lisp environment.

There are several hypertext applications that let users access files in FTP sites. Three of these are **World Wide Web** [Berner-Lee92], **Hyperbole** [Weiner91] and **HyperFTP** [Hornig90].

There are a number of systems that propagate changes to files with relaxed consistency. The **SUP** [Shafer89] and **Siphon** [Prusker91] systems are like this. Along this line, but even closer to Alex, is **Mirror** [McLoughlin91]. Mirror lets a user specify which files from a remote FTP site he wants to shadow locally.

The only other system I know of that gives access to remote FTP files via a real filesystem is **Touch** [Next92]. I have only limited information about this system. In this system there is only one site "mounted" at a time. There is no long term caching. The name space is not fixed (a user can mount the remote site anyplace he wants to). Each user runs a copy of Touch, while in Alex users mount a shared fileserver. Thus, Alex gets between user cache hits.

The **AFS** [Howard87] filesystem addresses issues of naming and performance for wide area filesystems. However, it is different in that it requires that both the server and the client be running AFS, while Alex uses existing FTP sites without modification.

# 5. Building on Top of Alex

While Alex nicely handles the naming and accessing issues, it does not address the indexing and organizing of the files. Since there is so much information available, this is very important. Fortunately, since it looks like a normal filesystem, Alex can easily be combined with other applications to get indexing and organization along with easy access. Systems such as WAIS [Kahle91], archie [Emtage92], Prospero [Neuman89], Gopher [Alberti91], Hyperbole [Weiner91], and World Wide Web [Berners-Lee92] can be combined with Alex to provide this functionality.

## 5.1. Customizing the Name Space

Symbolic links provide a user with the ability to "customize his namespace" though not with the power of some of the systems focusing on this issue. Symbolic links can be used to group related information together. For example, **/alex/edu/cmu/cs/sp/alex/links/cs-tr** is a directory with symbolic links to more than 110 directories around the world that each contain computer science technical reports (more than 3,000 papers so far). Also, symbolic links can be used to give very short names as aliases for longer ones. For example, **cs-tr** in my home directory is a symbolic link to **/alex/edu/cmu/cs/sp/alex/links/cs-tr**. A symbolic link to a remote file is as convenient as a copy and has several advantages. First, it does not take up any of the user's disk space. Next, if the remote copy changes a local copy would be out of date but the symbolic link can show the new data. Finally, the symbolic link documents where the file comes from, making it easy to go back later and look for related files.

While most symbolic links on FTP sites are modified to be relative to the machine they come from, links that point to **/alex** or **/afs** are recognized as referring to a global namespace and are not modified. This means that any FTP site can easily maintain a set of symbolic links to interesting places in Alex (as is done in the **cs-tr** directory above).

## 5.2. Integrating Archie, Prospero, and Alex using Archia

**Archie** [Emtage92] is a database that allows users to search for filenames in FTP sites. There is a client program that uses the **Prospero** [Neuman89] protocol to query the Archie database. For each match, **archie** returns a host, directory, and filename. **Archia** uses this client program and combines these 3 pieces into an Alex path. So a user can type **archia jokes** and get Alex pathnames with the word **jokes** in them. The user can then easily access these files with Alex.

### 5.3. Integrating Alex and WAIS

**WAIS** [Kahle91] stands for Wide Area Information Server. This is a software package that makes it easy to set up a database. Essentially, all you need to do is give it a list of filenames and it will index the contents. A WAIS server is run and people from around the Internet can access this database with the client software. Since Alex makes files in FTP sites look like normal files, it becomes easy to use something like WAIS to index files from all around the world. I have created two databases to experiment with this. One indexes README files, the other, computer science technical papers.

I created the **readmes** database by indexing files with names including the string "README". The initial set of pathnames was made using Archia. Since then I have used the Unix **find** command in /alex to find more. Currently I have indexed about 25,000 README files from around the world. Users can search on any words in any of these files. Since README files usually describe files in the directory they are in, this can frequently help a user find what they want.

For the **cs-techreports** database I have indexed about 3,000 computer science technical papers from around the world. I started with the above mentioned list of symbolic links to directories with technical papers. I used this list of directories and the Unix **find** command to get the filenames for the papers. I then used WAIS to index these.

## 6. Status

Alex is up and working. There are roughly 100 users. The server load is usually small since the average user accesses Alex for less than a half hour per week. The NFS protocol is fine for this level of load. Users have visited over 2,000 different FTP sites via Alex. Currently about 800 MB of disk is being used to cache data. With current levels of activity files usually stay in the cache more than a month past the time they are last accessed.

While real performance results are not yet available (see the Research Agenda section), I have some preliminary performance numbers. I did a "**find /alex/edu -print**" and let it run for a day. It found about 10,000 files per hour (most of which were not in the cache to start with) and had only finished a small fraction of the **edu** hosts before I killed it. When Alex was reading the 25,000 README files it averaged about 10 seconds per file (again most were not in the cache to start with). The techreports also averaged about 10 seconds per file.

The **ftpable-readmes** and **cs-techreports** database are available for use by anyone on the internet. Both receive steady use from around the world.

The source code for Alex has not yet been made available but should be this summer. Check **README** on **alex.sp.cs.cmu.edu** for current information.

## 7. Research Agenda

There are many research issues related to Alex. Below are some of the main ones I plan to explore.

## 7.1. Issues for Caching Algorithms

In this domain, there are many factors to consider when deciding what to keep in the cache and what to evict. As in other caching situations, how recently and how frequently a file has been used, as well as its size, are important factors. Additional new factors are past latency and bandwidth experienced when accessing the source for the file. The source's availability is also a factor. Thus, caching algorithms are potentially complex.

I need to determine what the important parameters are for caching algorithms and how they relate to each other. In particular, how latency, bandwidth, and availability should affect what is kept in the cache. It will be very interesting to know what sort of hit ratios we can get for the files and directories.

There are many other interesting things to measure, such as the number of times a file is used once in the cache and how long files go between usages.

## 7.2. Sharing of Cached Data Between Users

One of the benefits of this approach is that a cached copy of a file is frequently available because some other user has accessed the file. In [Ewing92] they found that just over half of the FTP traffic for users at one site during a short 2 week period was for files that more than one user accessed. Alex caches data for much longer periods and should get much higher levels of inter-user sharing of cached data. It will be interesting to see how often this happens. Preliminary indications are that this will be common for Alex users.

## 7.3. Characterization of Usage Pattern

FTP files are not typical user files, and the access patterns for FTP files are different from normal files. They are often archive files that don't ever change. Frequently the ones that do change are replaced by a new file of a slightly different name. For example **pbmplus05oct91.tar.Z** was replaced by **pbmplus10dec91.tar.Z**. This is reminiscent of immutable files in the Cedar filesystem [Gifford88].

## 7.4. Naming Ambiguity Problems

Among the systems visited, the only naming ambiguity noticed was **/alex/edu/berkeley/pub**. It turns out that there is nothing to ftp on **pub.berkeley.edu** so even this one instance has not been a problem. None of my users have experienced the problem. However, I plan to do a careful investigation looking for instances of this problem.

## 7.5. Quantifying Consistency

Alex can tell when it has given out a stale file later when it updates that directory. From this I can estimate the percentage of times users get stale data. Alex can also tell how far out of date the stale files were.

There are also cases where Alex can tell that it has stale data and clean things up. For example, if a user tries to fetch a file that no longer exists, Alex quickly updates the directory. Also if Alex fetches a file and the size is not what it was expecting Alex updates the directory information for that directory. Measuring how often these events happen also gives us some measure of consistency.

Any user can request the update of any directory. Users may desire this because they have reason to believe that the directory information is out of date (for example they saw a post saying a new file was in some

directory but it does not show up in Alex). Alex can tell if the directory has changed or not when it does this. It may also be that the user has had problems in the past and does not trust Alex to keep the files consistent. So measuring how often users do this and the percentage of time the directory really was out of date will give us two more interesting numbers to quantify Alex's cache consistency (real and user perceived).

### 7.6. Latency for Large Files

Right now when a user does a "**more bigfile**" where **bigfile** is not yet in the cache he does not see anything until the whole FTP has finished (since FTP is whole file transfer). This means that the user sees long latency at times. However, I will probably modify Alex so that it starts using data as soon as it comes in. Once this is done the user will see reasonable latency even for large files not in the cache.

### 7.7. Latency of Making FTP Connections

Setting up an FTP connection takes several seconds even on a lightly loaded machine. There are several ways that this latency might be reduced. It is possible that the FTP server code has not been tuned for fast connections, and some work in this area could improve things. Another approach is to try to reduce the number of times it is necessary to set up connections by keeping connection alive for longer periods. The problem with this approach is that it will only help a small fraction of the time, and it will increase the load on FTP sites. A third approach is to use other protocols such as NFS or Prospero on sites where they are available. However, this currently only helps with a small fraction of the FTP sites.

## 8. Conclusions

Alex is clearly a leap forward in accessing remote data. It makes it very easy for users and applications to access files from all around the world. Along with ease of use it also improves performance. While using FTP gives one the feel of looking at a remote place through a narrow tube, Alex gives one the feeling of actually visiting the remote place. World travel has never been easier.

This technology can be combined in a simple modular way with many different systems. Thus Alex is also an enabling technology.

This paper and other documents on Alex can be found in **/alex/edu/cmu/cs/sp/alex/doc**.

## 9. Bibliography

Bob Alberti, Farhad Anklesaria, Paul Lindner, Mark McCahill, Daniel Torrey, "Notes on the Internet Gopher protocol", Microcomputer and Workstation Networks Research Center, Spring 1991; Revised December 1991, /alex/edu/umn/micro/boombox/pub/gopher/gopher_protocol.

Tim Berners-Lee, Robert Cailliau, Jean-Francois Groff, Bernd Pollermann, "World-Wide Web: The Information Universe", CERN, 1211 Geneva 23, Switzerland, /alex/ch/cern/nxoc01/pub/WWW/doc/Article_9202.ps

A. Emtage, P. Deutsch, "archie - An Electronic Directory Service for the Internet", Usenix Conference Proceedings, pp. 93-110, San Francisco, CA, January 1992. /alex/ca/mcgill/cs/quiche/archie/doc/archie-usenix92-paper.ps.

David J. Ewing, Richard S. Hall, Michael F. Schwartz, “A Measurement Study of Internet File Transfer Traffic”, January 1992, Department of Computer Science, University of Colorado at Boulder, CU-CS-571-92, /alex/edu/colorado/cs/bruno/pub/cs/techreports/schwartz/RD.Papers/PostScript/FTP.Meas.ps.Z.

Douglas Hornig, “HyperFTP”, 1990 /alex/edu/indiana/bio/fly/util/mac/hyperftp.readme, see also /alex/edu/sdsu/ucselx/pub/mac/HyperFTP.sit.Hqx

David K. Gifford, Roger M. Needham, Michael D. Schroeder, “The Cedar File System”, Communications of the ACM, Vol 31, No 3, March 1988, pp. 288-298.

Brewster Kahle, “Wide Area Information Servers”, Thinking Machines, April, 1991, /alex/com/think/quake/wais/wais-discussion/overview.text.

J. Howard, M. Kazar, S. Menees,D.Nichols, M. Satyanarayana, R. Sidebothamn, M. West, “Scale and Performance in a Distributed file System”, Operating Systems Review, Vol 21, No 5, November 1987. See also /alex/org/central/grand/pub/doc/afs.

B. Clifford Neuman, “The Virtual System Model for Large Distributed Operating Systems”, Department of Computer Science, University of Washington, TR-89-01-07, /alex/edu/washington/cs/pub/pfs/doc/UW-CS-89-01-07.PS.Z

Next Touch, Personal communication with Beta user of Touch.

Andy Norman, “ange-ftp”, /alex/edu/mit/ai/gnu/alpha/ange-ftp/ange-ftp.el.Z, see also /alex/edu/reed/pub/mailing-lists/ange-ftp.

Bill Nowicki, “NFS: Network File System Protocol Specification”, Request for Comments: 1094, Sun Microsystems, March 1989, /alex/com/sri/nisc/phoebus/rfc/rfc1094.txt.

Barak Pearlmutter, Personal communication.

Francis Prusker, Edward Wobber, “The Siphon: Managing Distant Replicated Repositories”, Proceedings of the workshop on Management of Replicated Data, IEEE, Nov. 1990, pp. 44-47.

J. Postel, J. Reynolds, “File Transfer Protocol (FTP)”, Request for Comments: 959, /alex/com/sri/nisc/phoebus/rfc/rfc959.txt.

Herman Chung-Hwa Rao, “The Jade File System (Ph. D. Dissertation)”, Department of Computer Science, University of Arizona, TR 91-18. See also “Accessing Files in an Internet: The Jade File System”, /alex/edu/arizona/cs/llp/jade.ps

Steven Shafer, Mary Thompson, “The SUP Software Upgrade Protocol”, School of Computer Science, Carnegie Mellon, September 1989, /alex/edu/cmu/cs/mach/sup/dup.doc.

Bob Weiner, “Hyperbole Manual - Hypertext for Everyday Work”, Brown University, Dec. 1991, /alex/edu/brown/cs/wilma/pub/hyperbole/hypb.ps.Z

B. B. Welch and J. K. Ousterhout, “Prefix Tables: A Simple Mechanism for Locating Files in a Distributed Filesystem”, Proc. of the 6th International Conference on Distributed Computing Systems, May 1986, pp. 184-189.

Lee McLoughlin, “mirror.man”, Aug. 1991, /alex/uk/ac/ic/doc/src/archiving/mirror/mirror.shar.