World-Wide Web Proxies

Ari Luotonen, CERN
Kevin Altis, Intel

April 1994

Abstract

A WWW proxy server, proxy for short, provides access to the Web for people on closed subnets who can only access the Internet through a firewall machine. The hypertext server developed at CERN, cern_httpd, is capable of running as a proxy, providing seamless external access to HTTP, Gopher, WAIS and FTP.

cern_httpd has had gateway features for a long time, but only this spring they were extended to support all the methods in the HTTP protocol used by WWW clients. Clients don't lose any functionality by going through a proxy, except special processing they may have done for non-native Web protocols such as Gopher and FTP.

A brand new feature is caching performed by the proxy, resulting in shorter response times after the first document fetch. This makes proxies useful even to the people who do have full Internet access and don't really need the proxy just to get out of their local subnet.

This paper gives an overview of proxies and reports their current status.

1.0 Introduction

The primary use of proxies is to allow access to the Web from within a firewall *(Fig. 1)*. A proxy is a special HTTP [HTTP] server that typically runs on a firewall machine. The proxy waits for a request from inside the firewall, forwards the request to the remote server outside the firewall, reads the response and then sends it back to the client.

In the usual case, the same proxy is used by all the clients within a given subnet. This makes it possible for the proxy to do efficient caching of documents that are requested by a number of clients.

The ability to cache documents also makes proxies attractive to those not inside a firewall. Setting up a proxy server is easy, and the most popular Web client programs already have proxy support built in. So, it is simple to configure an entire work group to use a caching proxy server. This cuts down on network traffic costs since many of the documents are retrieved from a local cache once the initial request has been made.

Current proxy methodology is based on the earlier gateway code written by Tim Berners-Lee as part of `libwww`, the WWW Common Library [LIBWWW]. Kevin Altis,

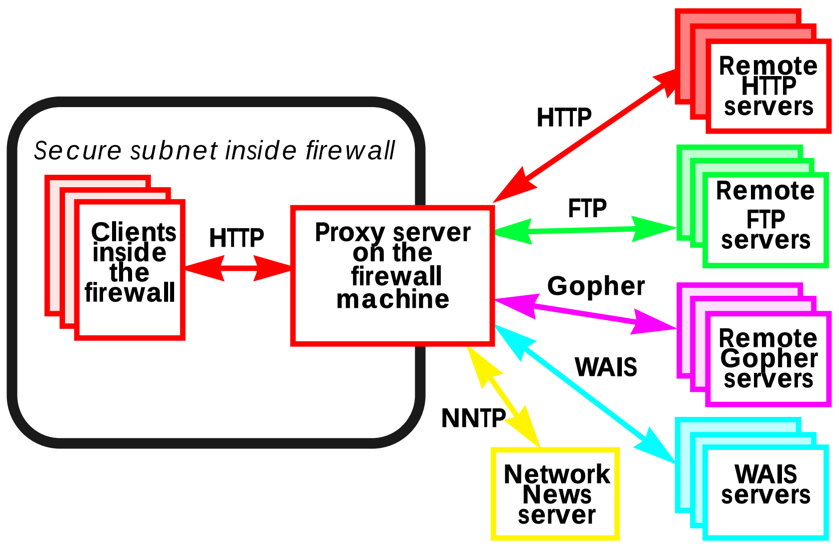

Figure 1.
Overall setup of a proxy. Proxy server is running either on a firewall host or other internal host which has full internet access, or on a machine inside the firewall making connections to the outside world through SOCKS or other firewall software.

Ari Luotonen and Lou Montulli have been the principle designers behind the proxy standard.

Lou Montulli, author of Lynx [LYNX], made the first `libwww` changes to support proxying in collaboration with Kevin Altis. Ari Luotonen maintains the CERN `httpd` [CERN-HTTPD]. Ari has made the server side of the proxy standard a reality and integrated caching into the proxy server

1.1 Why an application level proxy?

An application-level proxy makes a firewall safely permeable for users in an organization, without creating a potential security hole through which “bad guys” can get into the organizations’ net.

For Web clients, the modifications needed to support application-level proxying are minor (as an example, it took only five minutes to add proxy support for the Emacs Web browser [EMACS-WWW]).

There is no need to compile special versions of Web clients with firewall libraries, the standard out-of-the-box client can be configured to be a proxy client. In other words, proxying is a standard method for getting through firewalls, rather than having each client get customized to support a special firewall product or method. This is especially important for commercial Web clients, where the source code is not available for modification.

Users don’t have to have separate, specially modified FTP, Gopher and WAIS clients to get through a firewall — a single Web client with a proxy server handles all of these cases. The proxy also standardizes the appearance of FTP and Gopher listings across clients rather than each client having its own special handling.

A proxy allows client writers to forget about the tens of thousands of lines of networking code necessary to support every protocol and concentrate on more important client issues — it’s possible to have “lightweight” clients that only understand HTTP (no native FTP [FTP], Gopher [GOPHER], etc. protocol support) — other protocols are transparently handled by the proxy. By using HTTP between the client and proxy, no protocol functionality is lost, since FTP, Gopher, and other Web protocols map well into HTTP methods.

Clients without DNS (Domain Name Service) can still use the Web. The proxy IP address is the only information they need. Organizations using private network address spaces such as the class A net 10.*.*.* can still use the Internet as long as the proxy is visible to both the private internal net and the Internet, most likely via two separate network interfaces.

Proxying allows for high level logging of client transactions, including client IP address, date and time, URL [URL], byte count, and success code. Any regular fields and meta-information fields in an HTTP transaction are candidates for logging. This is not possible with logging at the IP or TCP level.

It is also possible to do filtering of client transactions at the application protocol level. The proxy can control access to services for individual methods, host and domain, etc.

Application-level proxy facilitates caching at the proxy. Caching is more effective on the proxy server than on each client. This saves disk space since only a single copy is cached, and also allows for more efficient caching of documents that are often referenced by multiple clients as the cache manager can predict which documents are worth caching for a long time and which are not. A caching server would be able to use “look ahead” and other predictive algorithms more effectively because it has many clients and therefore a larger sample size to base its statistics on.

Caching also makes it possible to browse the Web when some WWW server somewhere, or even the external network, is down, as long as one can connect to the cache server. This adds a degree of quality of service to remote network resources such as busy FTP sites and transient Gopher servers which are often unavailable remotely, but may be cached locally. Also, one might construct a cache that can be used to give a demo somewhere with a slow or non-existent Internet connection. Or one can just load a mass of documents to the cache, unplug the machine, take it to the cafeteria and read the documents there.

In general, Web clients’ authors have no reason to use firewall versions of their code. In the case of the application level proxy, they have an incentive, since the proxy provides caching. Developers should always use their own products, which they often weren’t with firewall solutions such as SOCKS. In addition, the proxy is simpler to configure than SOCKS, and it works across all platforms, not just Unix.

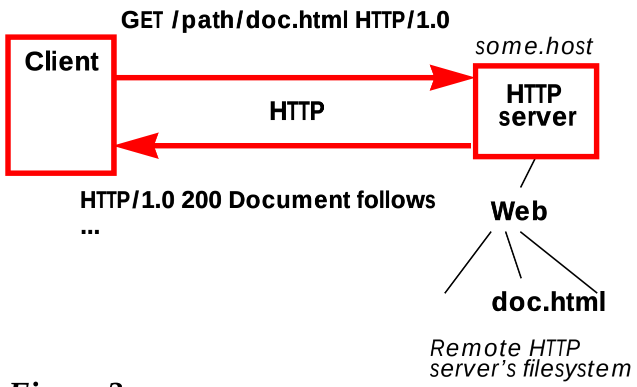
GET /path/doc.html HTTP/1.0
Client
some.host
HTTP
HTTP server
HTTP/1.0 200 Document follows
...
Web
doc.html
Remote HTTP server’s filesystem

Figure 2.
A normal HTTP transaction.
Client makes a request to the HTTP server and specifies the requested resource relative to that server (no protocol or hostname specifier in the URL).

2.0 Technical Details

When a normal HTTP request is made by a client, the HTTP server gets only the *path and keyword portion* of the requested URL *(Fig. 2)*; other parts, namely the protocol specifier “`http:`” and the hostname are obviously clear to the remote HTTP server — it knows that it is an HTTP server, and it knows the host machine that it is running on. The requested path specifies the document or a CGI [CGI] script on the local filesystem of the server, or some other resource available from that server.

When a client sends a request to a proxy server the situation is slightly different. *The client always uses HTTP* for transactions with the proxy server, even when accessing a resource served by a remote server using another protocol, like Gopher or FTP.

However, instead of specifying only the pathname and possibly search keywords to the proxy server, *the full URL is specified (Fig. 3 and 4)*. This way the proxy server has all the information necessary to make the actual request to the remote server specified in the request URL, using the protocol specified in the URL.

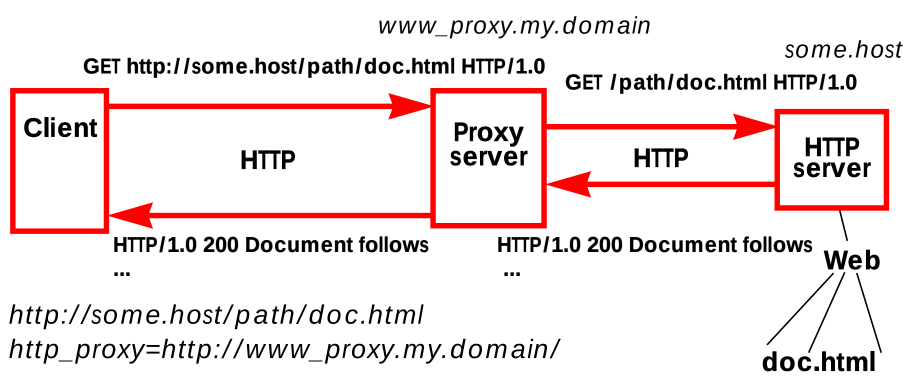
www_proxy.my.domain
some.host
GET http://some.host/path/doc.html HTTP/1.0
GET /path/doc.html HTTP/1.0
Client
HTTP
Proxy server
HTTP
HTTP server
HTTP/1.0 200 Document follows
...
HTTP/1.0 200 Document follows
...
Web
http://some.host/path/doc.html
http_proxy=http://www_proxy.my.domain/
doc.html

Figure 3.
A proxied HTTP transaction.
Client makes a request to the proxy server using HTTP, but specifying the full URL; the proxy server connects to the remote server and requests the resource relative to that server (no protocol or hostname specifier in the URL).

From this point on the proxy server acts like a client to retrieve the document; it calls the same protocol module of `libwww` that the client would call to perform the retrieval. However, the "presentation" on the proxy actually means the creation of an HTTP reply containing the requested document to the client. For example, a Gopher or FTP directory listing is returned as an HTML document.

2.1 Client Side Issues

Most WWW clients are built on top of `libwww`, the WWW Common Library, which handles the different communication protocols used in the Web, namely HTTP, FTP, Gopher, News [NTTP] and WAIS [WAIS].

The entire proxy support is handled automatically for clients using the `libwww`. Environment variables are used to control the library. There is an individual environment variable for each access protocol; e.g. `http_proxy`, `ftp_proxy`, `gopher_proxy` and `wais_proxy`. These variables are set to the URL pointing to the proxy server that is supposed to serve requests of that protocol, e.g.

```
ftp_proxy=http://www_proxy.domain:911/
export ftp_proxy
```

Usually the proxy server is the same for all the protocols, but does not have to be.

When the environment variable for a given protocol is set, the `libwww` code causes a connection to always be made to the proxy rather than directly to the remote server. Some clients also provide additional means of configuring the client to use a proxy server (e.g. Mosaic for X [MOSAIC-X] can use X resources and Mosaic for Windows [MOSAIC-WIN] uses settings in its initialization file).

The latest (as of April 1994) `libwww` (version 2.15) also supports an exception list so clients don't have to always go through the proxy. This is useful for avoiding the proxy for local servers where the clients can make a direct connection.

Another difference in the protocol between the client and the proxy is that the requested URL has to be *a full URL* when it is requested from the proxy server. These are the only differences between a normal and proxied HTTP transaction. The simplicity of proxy support in a Web client means that even clients not based on `libwww` can easily support the proxy.

Proxy support is implemented only for HTTP/1.0 on the server side so clients must use that protocol. This is not a problem because `libwww` does this automatically, and most clients (if not all) have already been upgraded to use HTTP/1.0.

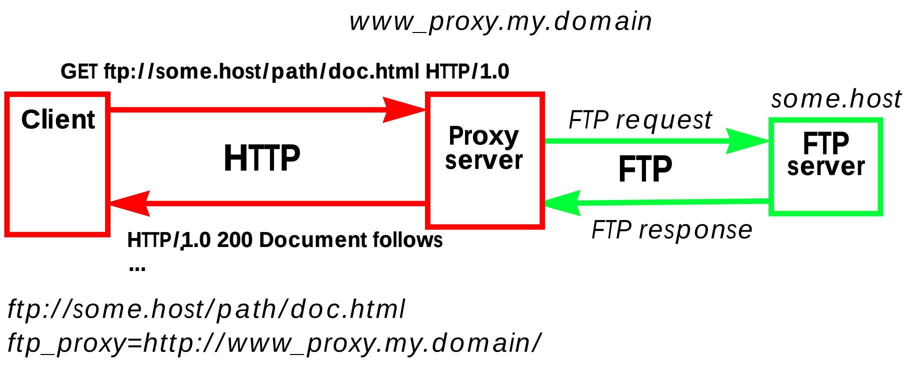
www_proxy.my.domain
GET ftp://some.host/path/doc.html HTTP/1.0
Client
HTTP
Proxy server
FTP request
FTP
some.host
FTP server
FTP response
HTTP/1.0 200 Document follows
...
ftp://some.host/path/doc.html
ftp_proxy=http://www_proxy.my.domain/

Figure 4.
A proxied FTP transaction.
Client makes a request to the proxy server, using HTTP, even though the actual resource is served by an FTP server. The proxy server sees from the full URL that an FTP connection should be made, and retrieves the file from the remote FTP server. Result is sent back to the client using HTTP.

2.2 Server Side Issues

The proxy server has to be able to act as both a server and a client. It is a server when accepting HTTP requests from clients connecting to it, but it acts like a client to the remote servers that it connects to in order to actually retrieve the documents for its own clients. The header fields passed to the proxy from the client are used without modification when the proxy connects to the remote server so that the client does not lose any functionality when going through a proxy.

A complete proxy server should speak all the Web protocols, the most important ones being HTTP, FTP, Gopher, WAIS and NNTP. Proxies that only handle a single Internet protocol such as HTTP are also a possibility, but a Web client would then require access to other proxy servers to handle the remaining protocols.

CERN `httpd,` which is one of the HTTP server programs, has a unique architecture in that it is currently the only HTTP server that is built on top of the WWW Common Library, which is otherwise just used by Web clients. Unlike other HTTP servers which only understand the HTTP protocol, CERN `httpd` is able to speak all of the Web protocols just like Web clients can as all the protocols are implemented by `libwww`.

CERN `httpd` has been able to run as a protocol gateway since version 2.00, released in March 1993, but additional features were required so the CERN `httpd` could act as a full proxy. With version 2.15, the server was enhanced to accept full URLs. The same server can now act as a proxy server for multiple protocols since the client always passes a full URL, thus allowing the proxy to understand which protocol to use to interact with the destination server. The CERN `httpd` can even act simultaneously as a normal HTTP server, serving local files in addition to proxying.

The server has been greatly improved during the spring of 1994. The original implementation didn't pass the access authorization information to the remote server which is essential in accessing protected documents. The body part of the message which is present with POST and PUT methods was not forwarded prior to version 2.15, which prevented HTML forms from working with the POST method.

Caching of documents has been introduced, giving noticeable speed-ups in retrieve times. Caching is a wide subject on its own and will not be studied in great detail in this paper.

It is also possible to compile a special SOCKS version of CERN `httpd` — this means that the proxy server does not have to run on the firewall machine, but rather it speaks to the outside world through SOCKS. Note, that this means "SOCKSifying" only the `httpd`, not the client programs.

In FTP the passive mode (PASV) is supported, in case a firewall administrator wants to deny incoming connections above port 1023. However, not all the FTP servers support

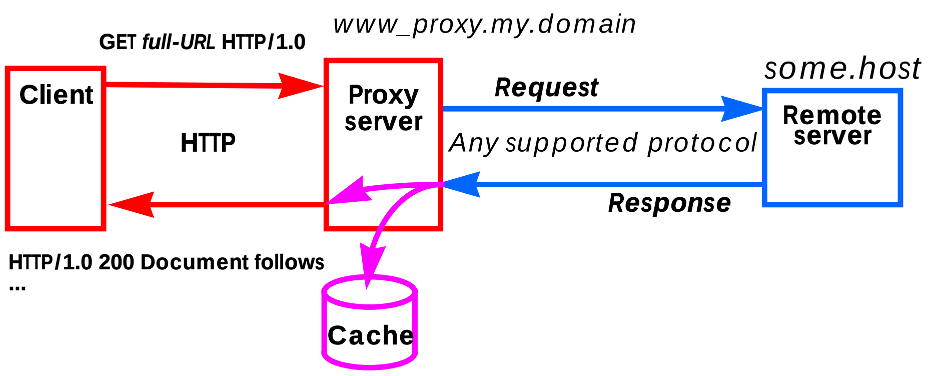

Figure 5.
A caching proxy.
The requested document is retrieved from the remote server and stored locally on the proxy server for later use.

PASV which causes a fall-back to normal (PORT) mode. This fails if incoming connections are refused, but this is what would happen in any case, even if a separate FTP tool was used.

2.3 Caching

The basic idea in caching is simple: store the retrieved document into a local file for further use so it won't be necessary to connect to the remote server the next time that document is requested *(Fig. 5 and 6)*.

However, there are many problems that need to be coped with once caching is introduced. How long is it possible to keep a document in the cache and still be sure that it is up-to-date? How to decide which documents are worth caching and for how long?

Document expiry has been foreseen in the HTTP protocol which contains an object header specifying the expiry date of an object. However, currently there are very few servers that actually give the expiry information, and until servers start sending it more commonly we will have to rely on other, more heuristic approaches, like only making a rough estimate of the time to live for an object.

More importantly, since many of the documents in the Web are "living" documents, specifying an expiry date for them is generally a difficult task. A given document may remain unchanged for a relatively long time, then suddenly change. This change may have been unforeseen by the document author and so wouldn't be accurately reflected in the expiry information.

2.4 Protocol Additions

When it is essential that the retrieved document is up-to-date, it is necessary to contact the remote server for each GET request. The HTTP protocol already contains the HEAD method for retrieving a documents' header infor-

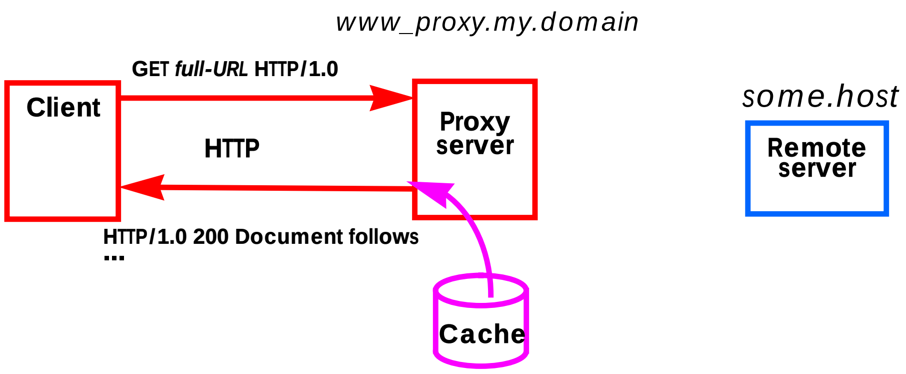

Figure 6.
Cache hit on the proxy.
If an up-to-date version of the requested document is found in the cache of the proxy server no connection to the remote server is necessary.

mation, but not the document itself. This is useful for checking if the document has been modified since the last access.

However, in cases where the document has changed, it would be very inefficient to make a second connection to the remote server to do the actual GET request to retrieve the document. The overhead of making a connection is often considerable.

The HTTP protocol was therefore extended to contain an `If-Modified-Since` request header, making it possible to do *a conditional GET request*. The GET request is otherwise the same except that this header contains the date and time that the object currently in the client (proxy cache) was last modified.

If the document has not been modified since the date and time specified it will not be sent back, instead only the relevant object meta-information headers, such as a new expiry date will be returned, along with a special result code. If the document has been modified it will be sent back as if the request was just a normal GET request.

The conditional GET makes several types of utilities more efficient. It can be used by mirroring software that has to refresh a large number of files on a regular basis. The caching proxy server could refresh its cache regularly during periods of client inactivity, not just at times when a document is explicitly requested.

It's worth noting that the conditional GET header is backward compatible. HTTP is defined so that unknown header fields are ignored. If the remote HTTP server does not support the conditional GET request no harm will be done, the server will just send the document in full. Fortunately all the major HTTP servers already support the conditional GET header.

The caching mechanism is disk based and persistent, which means it survives restarts of the proxy process as well as the server machine itself. Because of this feature, caching opens up new possibilities when the caching proxy server and a Web client are on the same machine. The proxy can be configured to use only the local cache, making it possible to give demos without an internet connection. You can even unplug a portable machine and take it to the cafeteria.

3.0 The Future

As the public enthusiasm for proxies has arisen just recently, there are many features that are still in their early stages, though the basic functionality is already there. Caching is clearly a wide and complicated area, and it is one of the parts of the proxy server that needs to be greatly enhanced. The proxy could be enhanced to do lookahead, retrieving all documents that are likely to be accessed soon. For example, all the documents referenced by a document that was requested recently, including all the inlined images.

The HTTP protocol should be further enhanced to allow multipart requests and responses; this would allow both caching and mirroring software to refresh large amounts of files in a single connection, rather than re-connecting to the remote server once for each file. Multipart messages are also needed by Web clients for retrieving all the inlined images with one connection.

Several aspects of the proxy architecture need to standardized. A proxy server port number should be assigned by the Internet authority. On the client side there is a need for a fallback mechanism for proxies so that a client can connect to a second or third proxy server if the primary proxy failed (like DNS). Also a dynamic lookup method for finding the closest proxy server is necessary; this might be achieved by using a standard DNS name, for example `www_proxy.my.domain`. This kind of dynamic host lookup is not just proxy-centric — Web clients should have the same kind of mechanism for finding a local home page, and the closest functional server in a set of servers mirroring each other.

4.0 Conclusions

Thanks to standard proxy support in the clients, and the wide availability of the `cern_httpd` proxy server, anyone behind a firewall can now have full Web access through the firewall host with minimum effort and without compromising security. Corporate users don't have to be denied access to the Web any longer.

Considering the extremely fast growth of the Web, its ability to replace FTP, and the fact that by the time this paper is published the Web usage has already passed Gopher usage metered by the packet statistics in the network, the use of caching proxy servers becomes essential to allow the growth to continue in case the total Internet capacity doesn't keep up with the Web growth rate. The proxy caching makes it possible to gain "virtual bandwidth" as documents often get returned from a nearby cache rather than from some far away server.

5.0 Authors

Ari Luotonen is writing his Master's Thesis at CERN until the summer 1994 on the architecture of generic hypertext servers. He is studying software engineering and mathematics in Tampere University of Technology, Finland, and will graduate in May 1995. His electronic mail address is luotonen@www.cern.ch.

Kevin Altis is an Internet Program Architect at Intel Corporations' Media Delivery Laboratory in Hillsboro, Oregon. He is interested in PC oriented usage of multi-media information via the Internet. His electronic mail address is altis@ibeam.intel.com.

6.0 References

[HTTP] HyperText Transfer Protocol, <URL:http://info.cern.ch/hypertext/WWW/Protocols/HTTP/HTTP2.html>

[FTP] File Transfer Protocol. J.Postel and J.Reynolds, File Transfer Protocol, Internet RFC 959, October 1985. <URL: ftp://ds.internic.net/rfc/rfc959.txt>

[GOPHER] The Internet Gopher. F.Anklesaria et.al., The Internet Gopher Protocol, Internet RFC 1436, March 1993. <URL: ftp://ds.internic.net/rfc/rfc1436.txt>

[WAIS] Wide-Area Information System. <URL: http://www.wais.com/z3950.html>

[NNTP] Network News Transfer Protocol, B.Kantor and Phil Lapsley, Network News Transfer Protocol, Internet RFC 977, February 1986. <URL: http://info.cern.ch/hypertext/WWW/Protocols/rfc977/rfc977.html>

[CGI] The Common Gateway Interface, Rob McCool, 1993-1994. <URL: http://hoohoo.ncsa.uiuc.edu/cgi/>

[URL] Uniform Resource Locators. <URL: http://info.cern.ch/hypertext/WWW/Addressing/Addressing.html>

[LIBWWW] The WWW Common Library. <URL: http://info.cern.ch/hypertext/WWW/Library/Status.html>

[CERN-HTTPD] CERN hypertext daemon, <URL: http://info.cern.ch/hypertext/WWW/Daemon/Status.html>

[LYNX] Lynx, a full-featured WWW client for character terminals.<URL: http://www.cc.ukans.edu/lynx_help/Lynx_users_guide.html>

[MOSAIC-X] NCSA Mosaic for X Window System. <URL: http://www.ncsa.uiuc.edu/SDG/Software/Mosaic/Docs/mosaic-docs.html>.

[MOSAIC-WIN] NCSA Mosaic for Microsoft Windows. <URL: http://www.ncsa.uiuc.edu/SDG/Software/WinMosaic/HomePage.html>.

[EMACS-WWW] The Emacs WWW Browser by William Perry. <URL: http://moose.cs.indiana.edu/usr/local/www/elisp/w3/docs.html>

CERN httpd as a proxy server: <URL: http://info.cern.ch/hypertext/WWW/Daemon/User/Proxies/Proxies.html>

Proxy support in Mosaic for X: <URL: http://www.ncsa.uiuc.edu/SDG/Software/Mosaic/Docs/proxy-gateways.html>

Proxy support in WinMosaic: <URL: http://www.ncsa.uiuc.edu/SDG/Software/WinMosaic/ProxyInfo.html>