Web Proxy

--- Term Project Specification

1. Purpose

A web proxy is capable of delivering content on behave of remote web servers. In this assignment you are asked to develop a multithreaded web proxy supporting a set of functionalities according to RFC 2616.

2. Background

As shown in Fig. 1, a proxy server receives requests from client browsers and forwards those requests to the destination web servers. Upon receiving the response (could be an html file, other file types, or even only an error message) from web servers, the proxy returns the response to the clients. To do that, the proxy server needs to understand the HTTP messages and their exchanging scenario, as defined in RFC1945 (HTTP 1.0) and RFC 2616 (HTTP 1.1).

A proxy server usually performs *caching* in addition to ordinary forwarding. This functionality stores the newly fetched content in the local disk cache, and replies the requesting client with the content in the cache. The server needs to periodically check with the origin (web server) whether the stored content is still valid. If not valid or is expired, the server then fetches the content again.

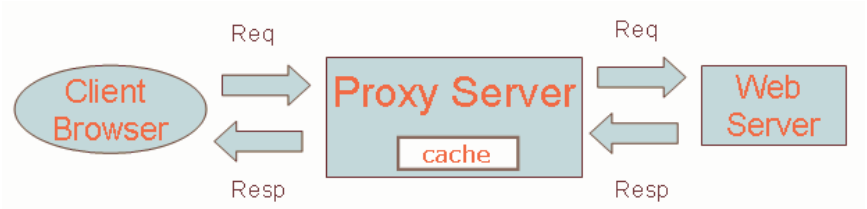

Fig. 1. Architecture and processing flow of a proxy server.

3. Functional Requirements

The functional requirements of this project include

(1) GET, HEAD commands. The GET is the mostly used command in HTTP, carrying out the actual fetching of the content. Your program will need to parse a command line like,

GET http://www.test.com/test.html HTTP/1.1,

from clients and then make a TCP connection to the target web server to get the requested page. The HEAD is similar to the GET except that only metainformation, rather than a complete message-body (ex: the page), is returned. Refer to section 9.4 of the RFC 2616 for detailed description.

(2) Support of the *If-Modified-Since* conditional request. This header field in an HTTP request is typically used to examine whether the content stored in the proxy server is still valid. If not, the server fetches again the content from the origin site, forwards it to the client, and replaces the existing expired one. Refer to section 14.25 in RFC 2616 for details. Note that you do not have to implement the caching operation. Just use an ordinary file in your local cache and compare it with the remote one.

(3) HTTP return codes including 304 (Not Modified), 501 (Not Implemented).

4. Testing

TAs will use the open source "wget" to test your final implementation. Use it for testing before demonstrating your server to TAs. Refer to http://www.gnu.org/software/wget/manual/wget.html for more information.

5. Problem Discussions (answer as many as you can to get extra points)

(1) What more could be expected, except request forwarding and caching, from a proxy server? Describe the possible implementation method.

(2) How do you implement the server if the response from the server is demanded to be sent back directly to the client? This sometimes happens when caching is not needed and scalability is a concern.

(3) Is it possible for you to make your proxy server a one-way transparent one, namely the client browsers do not have to explicitly configure the setting?

(4) What about a two-way transparent proxy? What's the possible maneuver to accomplish that? Express your opinions.

6. Report Format

The report should contain (1) purpose, (2) method, (3) implementation description, (4) program listing (comments are needed), and (5) screen dumping: the process of the compilation and result of the program; just as you include in a regular mini-project.

7. References

[1] RFC1945, "Hypertext Transfer Protocol -- HTTP/1.1". Available at http://www.faqs.org/rfcs/rfc2616.html.

[2] RFC2616, "Hypertext Transfer Protocol -- HTTP/1.1". Available at http://www.faqs.org/rfcs/rfc2616.html.

[3] Squid web proxy cache. http://www.squid-cache.org/.

[4] Wget project. http://www.gnu.org/software/wget/.