Massachusetts Institute of Technology
Department of Electrical Engineering and Computer Science

6.033 Lab, Spring 1998 | Handout 4 — A Web Proxy | February 25

Due date of preliminary design: Tuesday, March 3
Due date for implementation: Tuesday, March 17
Due date for design paper: Thursday, March 19

1 Introduction

Now that you have programmed an asynchronous TCP proxy, you will fully enjoy creating an asynchronous web proxy ☺ . In this lab you will have to make design decisions regarding what features to include and what tradeoffs to make between conflicting design criteria. In addition, you will write a preliminary design document and final design document.

In this handout, we call the *client* an application program that establishes connections for the purpose of sending requests[3]. Typically the client is a web browser (e.g., lynx or Netscape). We call the *server* an application program that accepts connections in order to service requests by sending back responses (e.g., the Apache web server)[1]. Note that a proxy can act as both a client and server. Moreover, a proxy could communicate with other proxies (e.g., a cache hierarchy).

2 Design Criteria

RFC 1945 defines a web proxy as a transparent, trusted intermediary between web clients and web servers for the purpose of making requests on behalf of clients. Requests are serviced internally or by passing them, with possible translation, on to other servers. A proxy must interpret and, if necessary, rewrite a request message before forwarding it. In particular, your proxy must address:

- *Caching.* The proxy must cache web files. Web caches decrease latency at the cost of serving stale data[5].
- *Non-blocking.* Your web proxy must operate asynchronously.
- *Transparency.* The proxy should be transparent to the client and server. Your design cannot rely on any special modification to the client or server.

Your proxy should function correctly for any HTTP/1.0 GET, POST, or HEAD request. However, you may ignore any references to cookies and authentication[1].

[1] Although, you are welcome to handle cookies and authentication.

2.1 Desirable Properties of Your Web Proxy

Your web proxy should tolerate many simultaneous requests. The web proxy will accept connections from multiple clients and forward them using multiple connections to the appropriate servers. No client or server should be able to hang the web proxy by refusing to read or write data on its connection.

How will you keep web cache coherent? Clients dislike stale pages. But unlike file system caches, stale home pages do have some value[5]. For this lab, you should use some form of a *validation check* to keep the cache up-to-date. You could test for freshness by using the `If-Modified-Since` or `Expires` header. This allows the server to determine freshness. On the other hand, the proxy could cache pages for a fixed period of time. In this case, the proxy determines freshness. Moreover, your proxy cache should work correctly. If asked for `http://www.lcs.mit.edu/`, the proxy should not return data on `http://www.mit.edu/`.

Search RFC 1945 for any warnings about "proxy" behavior. The lab TA's will test that your proxy handles requests as stated in RFC 1945.

3 The HTTP Protocol

The Hypertext Transfer Protocol (HTTP) is the most commonly used protocol on the web today. For this lab, you will use a less bleeding-edge specification: HTTP version 1.0.

The HTTP protocol assumes a reliable connection and, in current practice, uses the TCP protocol to provide this reliable connection. The TCP protocol provides the reliable transport of bytes between programs on two separate machines, even over an unreliable network. Luckily for us, the TCP protocol is built into the UNIX operating system.

The HTTP protocol is a request/response protocol. When a client opens a connection, it immediately sends its request for a file. A web server then responds with the file or an error message. You can try out the protocol yourself. For example, try:

```
(~/)% telnet web.mit.edu 80
```

Then type

```
GET /6.033/www/ HTTP/1.0
```

followed by a couple carriage returns. See what you get.

To form the path to the file to be retrieved on a server, the client takes everything after the machine name and port number. For example, `http://www.mit.edu/original/` means we should ask for the file `/original/`. If you see a URL with nothing after the machine name and port, then `/` is assumed[2].

On most servers, the HTTP protocol lives on port 80. However, it turns out that port 80 is protected on most UNIX systems, so we will have to run our web proxy on a higher port (≥ 1024). To use other ports, we need to modify our URLs a bit, adding the port number after the machine

[2] The server determines what page to return when just given `/`. Typically this default page is `index.html` or `home.html`

name. For example, entering `http://www.mit.edu:8008/` into your favorite web browser connects to the machine www.mit.edu on port 8008 using the HTTP protocol.

The format of the request for HTTP is quite simple. A request consists of a method followed by arguments, each separated by a space and terminated by a carriage return/linefeed pair. Your web proxy should support three methods: `GET`, `POST`, and `HEAD`[3]. Methods take two arguments: the file to be retrieved and the HTTP version. Additional headers can follow the request. The web proxy will especially care about the following headers: `Allow, Date, Expires, From, If-Modified-Since, Pragma: no-cache, Server`. However, your proxy must handle the other HTTP/1.0 headers[3]. Fortunately, the web proxy can forward most of the requests verbatim to the appropriate server. Only a handful of headers require proxy intervention.

Once the request line is received, the web proxy should continue reading the input from the client until it encounters a blank line. The proxy should then fetch the appropriate file and send back a response (usually the file contents) and close the connection.

4 Using a Web Proxy

To use a web proxy, you must configure your web browser. For Lynx, wget, or Mosaic, you must set an environment variable. The following sets your proxy to web.mit.edu.

```
(~/)% setenv http_proxy http://web.mit.edu/
```

In Netscape, find the Network Preferences and manually setup a proxy. For instance, you can set the HTTP proxy to web.mit.edu and the port to 80. Remember to revert your changes. Not all requests will work transparently through the web.mit.edu proxy.

5 HTTP in Action!

How does one watch an HTTP request in action? To make a simple HTTP request, most people will use `telnet`. However, `telnet` does not let you watch incoming HTTP requests. For a more sophisticated connection, use netcat. Netcat (`nc`) lets you read and write data across network connections using UDP or TCP[10]. For instance, this listens to the network on port 8000:

```
(~/)% add sipb
(~/)% nc -l -v localhost -p 8000
listening on [any] 8000 ...
```

Now point your favorite web browser to `http://localhost:8000/`. My version of Netscape generates:

```
connect to [127.0.0.1] from localhost [127.0.0.1] 5854
GET / HTTP/1.0
Connection: Keep-Alive
User-Agent: Mozilla/3.01 (X11; U; Linux 2.0.30 i586)
Host: localhost:8000
Accept: image/gif, image/x-xbitmap, image/jpeg, image/pjpeg, */*
```

The first line asks for a file called / using HTTP version 1.0. Look in RFC 1945 for details on the remaining lines. Lynx produces a similar request:

```
connect to [127.0.0.1] from localhost [127.0.0.1] 5917
GET / HTTP/1.0
Host: localhost:8000
Accept: application/postscript, image/gif, application/postscript, */*;q=0.001
Accept-Encoding: gzip, compress
Accept-Language: en
User-Agent: Lynx/2.6  libwww-FM/2.14
```

If you set the proxy to `http://localhost:8000/`, your browser will try to use port 8000 as the proxy. Retrieving `http://c0re.l0pht.com/~weld/netcat/readme.html` produces:

```
connect to [127.0.0.1] from localhost.mit.edu [127.0.0.1] 2328
GET http://c0re.l0pht.com/~weld/netcat/readme.html HTTP/1.0
If-Modified-Since: Thursday, 12-Sep-96 02:25:13 GMT; length=63340
Referer: http://c0re.l0pht.com/~weld/netcat/
Proxy-Connection: Keep-Alive
User-Agent: Mozilla/3.01Gold (X11; U; OpenBSD 2.2 i386)
Pragma: no-cache
Host: c0re.l0pht.com
Accept: image/gif, image/x-xbitmap, image/jpeg, image/pjpeg, */*
```

The above shows what a web browser sends to a web proxy. Obtaining sample data from a real web proxy is a little trickier. Set your proxy to `http://web.mit.edu/` and run locally:

```
nc -v -l web.mit.edu -p 8000
listening on [127.0.0.1] 8000 ...
```

When I ask my web browser for `http://tiramisu.mit.edu:8000/` (my local machine), netcat reports:

```
connect to [18.238.0.32] from ARACHNOPHOBIA.MIT.EDU [18.69.0.27] 1068
GET / HTTP/1.0
Proxy-Connection: Keep-Alive
Host: tiramisu:8000
Accept: image/gif, image/x-xbitmap, image/jpeg, image/pjpeg, */*
User-Agent: Mozilla/3.01 (X11; U; Linux 2.0.30 i586)  via proxy gateway
  CERN-HTTPD/3.0 libwww/2.17
```

Try this on Athena. Look for differences between the web browser's request and the corresponding proxy request.

6 Administrivia

6.1 Where to Start?

Read over some of the suggested literature at the end of this document. You can find the Internet Request for Comments (RFC's) in the `rfc` locker: `attach rfc; cd /mit/rfc; ls rfc1945.txt`.

After you have a general understanding of the problem, play with netcat and begin your web proxy design. Once you have convinced yourself of the correctness of your design, you should implement the web proxy. Likely you will discover new, fascinating problems and will need to modify your design appropriately.

We may handout additional source code to help test the performance and correctness of your web proxy.

6.2 Handin Procedure

When you're ready to hand in the lab, run `/mit/6.033/lab/handin/turnin2`. This will create a ~`/6.033/p2` directory. Copy your project to this directory. You should not copy object code or binary executables. We will compile the web proxy.

The project should include a valid Makefile and the source code for the web proxy. The Makefile should create a proxy named `webproxy`. The proxy must support at least two command-line options: `-help` should explain the usage of your web proxy and `-p` <*port*> should run the proxy on port <*port*>. You may add any other options as long as you document them. If you make any improvements to the web proxy specifications, please include comments in a README file.

For example, to run a proxy on port 8000 of your local machine:

```
(~/)% ./webproxy -p 8000 &
```

6.3 Due Dates

A preliminary design document describing your web proxy is due on **Tuesday, March 3**. It should contain a description of the web proxy with arguments for why it meets the design constraints. In addition, the paper should describe any additional twists you may have decided to add to the specification. If you cannot find a lab TA, bring your preliminary document to NE43-521C before 5PM (when the main doors lock).

The final code for the web proxy is due on **Tuesday, March 17**. Follow the directions in section 6.2 to handin your project.

The final design document for the web proxy is due on **Thursday, March 19**. This design paper should be comparable in length (maximum 10 pages) and quality to the first design projects for 6.033. You can use the final design document in place of your 6.033 design project 1 (eligible for the Phase II writing requirement). Read the Mayfield Handbook for a detailed description on how to write a design document[11]. At a minimum, your final design document should contain an abstract, introduction, design criteria, implementation, recommendation, and summary.

6.4 Collaboration

Like the TCP Proxy, this is an individual project. The design documents and code must be yours, but you are otherwise free (and encouraged) to discuss the design and implementation details with other 6.033 lab students. Use the acknowledgments section of your final design document to give credit where it is due.

One last piece of advice: Do not wait until the last week to begin[3].

References

[1] *Apache Web Proxy*, `http://www.apache.org/docs/mod/mod_proxy.html`.

[2] T. Berners-Lee. *Propagation, Replication and Caching on the Web*, `http://www.w3.org/Propagation/`.

[3] T. Berners-Lee, et al. *RFC 1945: Hypertext Transfer Protocol – HTTP/1.0*, `http://ds.internic.net/rfc/rfc1945.txt`, May 1996.

[4] *CERN Web Proxy*, `http://www.w3.org/Daemon/User/Proxies/Proxies.html`.

[5] A. Dingle, T. Partl. *Web Cache Coherence*, `http://sun3.ms.mff.cuni.cz/~dingle/webcoherence.html`, May 1996.

[6] *SquidCache*, `http://squid.nlanr.net/Squid/`.

[7] R. Fielding, et al. *RFC 2068: Hypertext Transfer Protocol – HTTP/1.1*, January 1997.

[8] J. Franks, et al. *RFC 2069: An Extension to HTTP : Digest Access Authentication*, January 1997.

[9] J. C. Mogul, et al. *RFC 2145: Use and Interpretation of HTTP Version Numbers*, May 1997.

[10] Netcat. `http://c0re.l0pht.com/~weld/netcat/`.

[11] L. Perelman, et al. *The Mayfield Handbook of Technical & Scientific Writing*, `http://tute.mit.edu/afs/athena/course/21/21.guide/www/home.htm`.

[12] D. Wessels. *Web Caching Reading List*, `http://ircache.nlanr.net/Cache/reading.html`.

[3] I told you so! ☺