

# Retro Browser

-Shantanu Shah

scs49@cornell.edu

Dept. of Computer Science, Cornell University

15<sup>th</sup> Dec 2005

## Abstract

Given more than a petabyte of internet cache what is the simplest way of exposing it to end users? What is the preferred technique which requires minimum configuration changes on the users' part and is as natural as browsing the internet itself? What mechanism will prove to be effective in enabling users to access this data with ease with little or no trade-offs involved?

## Introduction

The Web Lab project[5] involves storing a large amount of internet cache – projected to be more than a petabyte – in a SQL database. The database contains HTML pages and other documents found on the internet in various formats like JPEG, PDF, PS, BMP, etc. Dynamically generated pages based on user preferences and sections of websites that require authentication are not part of this database. All the documents in the database are associated with a crawlid, which in turn is based on the archival date and time. In combination with the URL of the page, this uniquely determines the page within the database. The Retro Browser is a system configured to expose this data to the end users.

The major component of the retro browser is an Apache server configured to be used as a proxy server. The user component is any commodity browser configured to use this proxy server. The setup of the server and the browser are explained in further sections.

To obtain the data from the database, the retro browser proxy server utilizes the Basic Access Web Service [5]

## Scope

The retro browser design takes into account the following factors:

1. The system is designed with the general user in mind. This assumption implies that the user may not be able to install new software, run scripts and need not be technically proficient.
2. The user must be able to use the vast amount of data in the most comprehensible way. This implies that we use the standard web-browsing method of delivering pages that the user is most familiar with.
3. The user will expect internet based tools to work with no or very little modification. Apart from the commodity web browser, the user must be able to run download managers, web crawlers, etc. as required with almost no modification.

## Related Work

The internet archive [6], which is the source of data for our system, hosts a very large number of web pages. It has chosen to expose these pages as standard HTML pages through the Wayback Machine [6]. This satisfies the requirements 1 and 2 as described in the previous section. The internet archive exposes these pages by modifying the URL of the page to identify the particular crawl. For example, Cornell University's homepage <http://www.cornell.edu> as of March 01, 2001 is served by the URL: <http://web.archive.org/web/20010301082453/http://www.cornell.edu/> All the URLs referenced by this page are also modified by the wayback machine before serving the page. Note that, in this approach, web crawlers and other tools may require modification for correct results.

## Procedure Followed

We have followed the proxy method of serving pages keeping in view all the three design goals mentioned earlier.

Before proceeding further, it is necessary to revise the basic functioning of the HTTP

protocol, especially the handling of HTTP cookies.

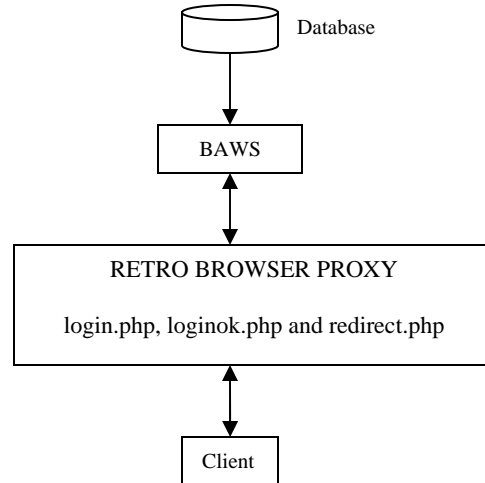
**HTTP Protocol[1]:** HTTP is a simple stateless request/response protocol used to transmit documents over the World Wide Web. A client, typically a web browser, sends requests to a web server and the web server responds with appropriate status code. The web server may further send a message of it's own which typically contains the contents of the file requested. Please refer to [1] for further details.

**HTTP Cookies [2]:**

HTTP cookies provide a way of maintaining state in the otherwise stateless HTTP protocol. An HTTP Cookie, popularly known just as a 'cookie' is a small text file that contains state information. The server can set multiple cookies, containing appropriate data as required. The cookie is passed to the originating server with every request. The server can also delete cookies that it has previously set. Every cookie is associated with a single domain or sub-domain. All cookies are domain-specific. The client will perform actions related to the cookies only when it is communicating with the domain, i.e. cookie set by a domain cannot be accessed by another domain. Please refer to [2] for further details.

**Architecture:**

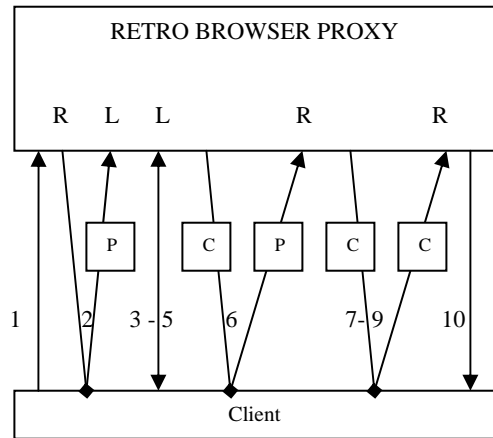
The retro browser client interacts with the retro browser proxy in a standard HTTP client-server fashion. As noted earlier, to fetch the appropriate page from the database, the crawlid is required in addition to the URL. The retro browser server expects a cookie specifying a crawlid with every request. If such a cookie is not found with the request, the user is asked to specify the crawlid. The architecture is aimed to reduce the number of times the user is asked to select the crawl, and successfully brings down this number to one, i.e. the user is asked to specify the crawl only for the first request. Further requests may or may not contain a cookie since cookies are tied to a domain. However, the retro browser proxy ensures that the cookie is replicated for all domains using a series of redirects. The architecture of Retro Browser can be summarized by the following diagram:



**Fig 1: The Retro Browser Architecture**

All the URLs requested by the browser are rewritten by redirect.php. This page will redirect to login.php if no cookie specifying the crawl id is found for this domain. login.php determines and sets a crawlid based on user choices. This cookie is set for the domain of the proxy server, and this value is propagated by passing a parameter to redirect.php during redirection. The steps are detailed below:

**Scenario 1:**



*P specifies that a parameter is sent with the request, and C specifies that a cookie is sent with the request. R denotes that the request is handled by redirect.php and L denotes that the request is handled by login.php. Numbers indicate the steps as detailed below*

**Fig 2: Client Server interaction on first access (Scenario 1)**

The user accesses the first URL (i.e. no cookies set): (Please refer to fig 2.)

1. The input URL is rewritten and the page redirect.php is sent to the client.
2. redirect.php determines whether a cookie or a parameter is present. Since no cookie or parameter is found at this step, it redirects to login.php with the URL as a parameter.
3. login.php determines whether a cookie is set for this proxy server. Since no cookie is found at this step, login.php displays all the crawls available to the user.
4. The user selects a crawl for the specified URL. Note that a crawl once selected will remain selected until the user manually changes it by entering the URL for login.php manually.
5. At this step, a cookie is sent to the client. This cookie is set for the domain of the proxy.
6. The client is redirected to the original URL, this time with a parameter that specifies the crawl id requested.
7. Once again, this URL is rewritten with the page redirect.php. This time however, redirect.php obtains a crawl id is present as a parameter.
8. redirect.php sets a cookie for this domain based on the parameter crawlid passed. It then redirects to itself, i.e. the original URL the user entered.
9. When the client requests this page, the cookie associated in previous step is sent with this request. This request is rewritten with redirect.php.
10. redirect.php finds a cookie associated with this request that contains the requested crawl id. It fetches the associated page using BAWS [5] and serves it to the client.
11. Further requests to this domain have a cookie set along with them. All pages are rewritten by redirect.php and finding the cookie set, redirect.php server the correct page.

At this point, the user has two cookies set: 1) The cookie associated with the proxy server domain and 2) The cookie associated with the domain of the requested URL. Note that both the cookies have the crawlid as their content. The cookie associated with the domain of the URL requested saves additional redirects to login.php

which would otherwise be required similar to scenario 2.

### **Scenario 2:**

The user accesses a page in another domain after successfully accessing a page using scenario 1, i.e. the user has a cookie associated with the proxy server domain and the user accesses a page in a domain for which no cookie has been set.

This scenario is similar to scenario 1, except that the user is not required to chose a crawl when he is displayed the login page. Instead, the crawl chosen when the user accessed the first page is selected for this page also. Thus, no action is required on the user's part, automatically the previously selected crawl is used to display this page, giving the user an experience of surfing the internet as of the previously selected crawl.

1. The input URL is rewritten and the page redirect.php is sent to the user.
2. redirect.php determines whether a cookie or a parameter is present. Since no cookie or parameter is found at this step, it redirects to login.php with the URL as a parameter.
3. login.php determines whether a cookie is set for this proxy server. Since a cookie with a crawl id is found, the client is redirected to original URL, this time with a parameter that specifies the crawl id requested.
4. Once again, this URL is rewritten with the page redirect.php. This time however, redirect.php determines that a crawl id is present as a parameter.
5. redirect.php sets a cookie for this domain based on the parameter crawlid passed. It then redirects to itself, i.e. the original URL the user entered.
6. When the client requests this page, the cookie associated in previous step is sent with this request. This request is rewritten with redirect.php.
7. redirect.php finds a cookie associated with this request that contains the requested crawl id. It fetches the associated page using BAWS [5] and serves it to the client.
8. Further requests to this domain have a cookie set along with them. All pages are rewritten by redirect.php and

finding the cookie set, redirect.php server the correct page.

### Scenario 3:

The user wishes to change or verify the selected crawl and visits login.php (without being redirected to this page).

1. login.php determines that the client has not been redirected from any other page and displays a list of all available crawls.
2. The user selects a crawl for the specified URL. Note that a crawl once selected will remain selected until the user manually changes it.
3. At this step, a cookie with the selected crawlid is sent to the user's browser. This cookie is set for the domain of the proxy.
4. The user can now input any URL and the system will further proceed as described in scenario 2.

## Implementation Details

### URL Rewriting in Apache:

URL Rewriting in apache web server allows URL manipulation on the server side. In our case, all URLs except login.php and loginok.php are rewritten by redirect.php, i.e. the page redirect.php is served to the client in response to any URL except those mentioned above. Note that the URL is rewritten and not redirected, i.e. the client is not aware of this substitution of the pages. The client is not redirected as would happen in a regular HTTP redirect request. Thus, this is a transparent process which allows us to serve the right page to the user without changing the URL as seen by the client.

PHP is used for scripting on the server side. PHP is a powerful scripting language and easy to understand and manipulate.

The source code files are available at [5].

## Applications

Although the retro browser has been developed as a mechanism for exposing pages in the Web Lab project, the usage is not limited to the web lab project itself. The proxy setup can be used to serve pages in any environment where additional parameters other than the URL are required by the server from the user before processing a request.

## Results

The system as described above has been implemented and tested for a limited number of connections on a personal computer. The resulting system works on a standalone basis.

The proxy server has been tested to work with cookie-enabled clients such as Internet Explorer 6, Firefox 1.5 and Opera 8, which are all commodity browsers.

## Future work

Future work in this area involves deploying the server on a server-class machine. A 64-bit architecture may provide scalability but may also involve additional porting issues that need to be investigated.

The system described above has been tested with a limited number of connections. It must be tested for a large number of users, perhaps thousands to find out the areas of optimization. A cluster of proxy servers can be implemented to achieve scalability.

The system as described above does not do perform any authentication. Authentication may be required for some pages or for restricting the access as appropriate. Adding authentication will require further steps that may include SSL implementation or a simple username/password technique. In this case a signup form and an additional database table for authentication may be warranted.

## References

- 1] HTTP version 1.1 Protocol RFC:  
<http://www.w3.org/Protocols/rfc2616/rfc2616.html>
- 2] HTTP State Management Mechanism, RFC 2965: <http://www.ietf.org/rfc/rfc2965.txt>
- 3] Apache URL Rewriting guide:  
<http://httpd.apache.org/docs/1.3/misc/rewriteguide.html>
- 4] Apache URL rewriting engine (mod\_rewrite) reference documentation:  
[http://httpd.apache.org/docs/1.3/mod/mod\\_rewrite.html](http://httpd.apache.org/docs/1.3/mod/mod_rewrite.html)
- 5] Web Lab Project:  
<https://gforge.cis.cornell.edu/projects/wri/>
- 6] Internet Archive: <http://www.archive.org>

## **Acknowledgements**

This work is part of the Web Laboratory, which is a joint project of Cornell University and the Internet Archive. The author acknowledges the support from Prof. William Arms, Cornell University for this work. This work is funded in part by National Science Foundation grants 0403340, 0127308, and 0537606.