Proxy Server

Richy Delaney (08479950)

March 9, 2011

Introduction

The project specification asked us to design a web proxy which could route traffic from a browser to a server and get the response and return it to the browser. Along with the basic proxy functionality, seperate requests must be handled in threads, we were also required to provide a GUI for monitoring requests and managing the proxy. Blacklisting of certain IP addresses and the ability to cache responses was also a requirement.

I decided to approach the project with that in mind, I broke it into stages and approached every section on its own. Through this design description I'll discuss my design decisions for each section and discuss possible faults or point out obstacles I met on the way.

Firstly, a note on my choice of language, I chose python for a number of reasons:

1. Easy to write understandable and manageable code
2. Good networking librarys for low level socket work.
3. Previous experience in the language.
4. It is cross platform, My proxy should work across different operating systems.

Basic Proxy

The first major milestone for me in the project was to get basic proxy functionality working. After a lot of researching into methods of doing socket programming in python, and writing some example connection code, I started to build up my proxy class.

My proxy class was initialised with a port number. From there, I assigned a host which was localhost, I made a new socket which I bound to localhost on the defined port. My browser would connect through this. One of the first obstacles I came up against was the issue of local addresses not being able to

be resued which meant I had to bind my proxy to a different port on each run. I got around this by using the following:

socket.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)

After the socket was bound to the host and port, It was then set to listen, I had an infinite loop which waited for a request from the socket. On the receving of a new request, the proxy would accept it and start a new thread with that request. This was another obstacle, at first I was passing my Request class as the parameter for a new thread and this was causing a huge memory management problem as I had a high number of threads the more requests I made as the threads weren't exiting. To counter this, I decided to make a function which made a new Request and exited after a certain method was called.

My Request class deals with all the response and requests from browser and server.

First thing it does is get the header from the request of the browser. For example in a request for http://www.google.com/ , I want the Request class to get 3 variables, a method, path and protocol from the header and I want the rest of the header (i.e Content-Type) to be stored in a buffer. So the socket which is bound to the browser receives in the request and breaks the request out into the relevant parts.

My Proxy also deals with requests differently depending on the method.

If the method is CONNECT, the algorithm is as follows, I make a connection with the required server. For this I use getaddrinfo to determine the ip address of the server. When a connection to the server is made, I send back a connection successful response to the browser and check if there is any response from either socket(browser or socket) and terminate the thread.

If the method isn't CONNECT, but is GET, POST or any of the other less common method types, I deal with it differently, I do the same connect to target as in connect, I then send a request to the server with the header. The program then goes into a read write stage where it gets responses and forwards them between the two sockets, mainly server to client.

With this, the basic proxy functionality is complete.

Blacklisting

For the blacklisting I decided the best route to go was a text file which they can be configured in, within the GUI, you can add to the blacklist, and the blacklist is stored in a dict object(hashmap) while the program is running for quick lookups.

I do a basic lookup on the server's ip address, if the ip address is found in the black list it is automatically swapped out for the blacklist value. I have designed mine that instead of just terminating the request, to blacklist a website you can just redirect to localhost, this allows for site redirection as a nice bonus to the method.

Through this easy method, I implemented blacklisting effectively in my project.

Managerial Graphical User Interface

When I designed the basic functionality of the project, it was all command line driven and when I went to build the gui I had a choice to make. Do I build a gui on top of what I have or do I somehow seperate the two into communcating programs.

I chose the latter, effectively I wanted my gui to monitor requests coming in and display them in a clear fashion. I decided to build a gui seperate and connect the two using DBUS. I chose to do it this way for a number of reasons:

1. DBUS is easy to configure and has readily accessible libraries for python
2. Seperates backend logic and frontend display
3. Allows simple method passing and string passing which is all I needed.
4. Would allow for expansion onto different applications possibly even written in different languages once they could use DBUS.
5. The Infinite loop of some graphical toolkits is difficult to work with when you have infinite loops in the program

The Gui I went for was a minimalistic but ultimately effective design. I wanted it to be mostly occupied with the requests coming in. The toolkit I chose to use was GTK, as it was easy to implement a simple list display and allowed for things like scrolled windows, menu's and made it easy to make what I wanted easily. It also has good python integration.

I use a simple liststore with three columns, one for method, one for path and one for protocol. These are updated in real time by the proxy application with requests coming from the browser. They are in a scrollable window and allow for the user to see clearly all the requests coming from the browser.

The gui also allows you to add things to the blacklist for blocking. Although my attempt at the GUI took a different approach, I decided it fitted the program where the real work gets done in the background while the frontend has a nice graphical, sortable list of the activity.

Caching

This definetely proved to be the most challenging and difficult part of the project. I implemented caching by caching all responses out to disk in the format they came in. For example, given the path http://www.google.com/projects/blah.html this was cached in the cached folder of the working directory in the following way:

WORKING_DIR + "cached/google.com/projects/blah.html"

This was the way I dealed with all requests so the requested data could be anything from a png file to a html file. The only area I came into some difficulty here was in the area of paths where the browser looks for the default page.

In the case of http://www.google.com/, to get around the fact that this would only have a folder and not an extension, I just added a random extension which I called index. It then appends any response data from the server for that path to that file.

This includes response headers and repsonse body.

On next lookup, the browser requests the data, the proxy then checks its cache, i.e does a file exists query on the location in the cached folder, and sees that it exists and returns it to the browser.

There are lots of area's which proved to be difficult in this, but my final draft works for caching. At times, Dynamic content doesn't like to be cached, for instance something that is updated often will fail to be noticed, there are headers such as Not-Modified which can be used to detect this. Another problem was long url names, some of googles dynamically created content had filenames which could not be stored.

Conclusion

By completing each section at a time, I've completed a low level proxy implementation which features caching, blacklisting as well as a gui for monitoring requests. It is basic, and the fact that a thread for each request is open can cause a system bottleneck if there is a lot of browser activity.