CSCI 379

Computer Networking

Programming assignment #2

Due by 11:59pm, Wednesday, Dec. 27th, 2017

In this project, you will implement a web proxy that passes requests and data between multiple web clients and web servers. This assignment will give you a chance to get to know one of the most popular application protocols on the Internet -- the Hypertext Transfer Protocol (HTTP).

You may work with a partner on this assignment.

HTTP Proxy

Getting Started

- On your host machine (laptop), download the starter package.
- You will find the following starter code files: `http_proxy.py test_scripts README.pdf`

Task Specification

Your task is to build a web proxy capable of accepting HTTP requests, forwarding requests to remote (origin) servers, and returning response data to a client.

The proxy will be implemented in your prefered language, and if you're not using python, an Makefile to generate excutable and a brief instruction has to be provided.

Your proxy program should run without errors. It should take as its first argument a port to listen from. Don't use a hard-coded port number. If you use any other programming language, you need to supply a `Makefile` for compilation, and it should produce a executable file called `http_proxy`.

You shouldn't assume that your proxy will be running on a particular IP address, or that clients will be coming from a pre-determined IP address.

Dummy Proxy (100pts)

Your proxy should listen on the port specified from the command line and wait for incoming client connections. Once a client has connected, the proxy should read data from the client and then check for a properly-formatted HTTP request.

The client issues a request by sending a line of text to the proxy server. This **request line** consists of an HTTP *method* (most often "GET", but "POST", "PUT", and others are possible), a *request URI* (like a URL), and the protocol version that the client wants to use ("HTTP/1.1"). The request line is followed by one or more header lines. The message body of the initial request is typically empty. URI in the HTTP request to a proxy server is required to be absolute URL, so you will obtain both host name and path to the file in **request line**.

Your dummy proxy is only responsible for accepting the HTTP request. All requests should elicit a well-formed HTTP response with status code 501 "Not Implemented". In the body of this HTTP response, you should supply a dummpy HTML page which prints the URL requested by client and a message "You request will be forwarded."(However, it won't...unless you finish the next part).

Note: You need to parse the HTTP message, don't dump the entire message directly on the returned page.

Complete Proxy Servers(Extra point: 80pts)

Before you start this part, make a copy of the code for the dummpy proxy and named it as `http_proxy_dummy.py`. Then continue to work on the file `http_proxy.py`.

Sending Requests to Servers

Once the proxy has parsed the URL in the client request, it can make a connection to the requested host (using the appropriate remote port, or the default of 80 if none is specified) and send the HTTP request for the appropriate resource. The proxy should always send the request in the relative URL + Host header format regardless of how the request was received from the client.

For example, if the proxy accepts the following request from a client:

```
GET http://www.example.com/ HTTP/1.1
```

It should send the following request to the remote server:

```
GET / HTTP/1.1
Host: www.example.com
Connection: close
(Additional client specified headers, if any...)
```

Note that we always send HTTP/1.1 flags and a `Connection: close` header to the server, so that it will close the connection after its response is fully transmitted, as opposed to keeping open a persistent connection. So while you should pass the client headers you receive on to the server, you should make sure you replace any `Connection` header received from the client with one specifying `close`, as shown.

Returning Response to Clients

After the response from the remote server is received, the proxy should send the response message as-is to the client via the appropriate socket.

For any error caught by the proxy, the proxy should return the status 500 'Internal Error'. As stated above, any request method other than GET should cause your proxy to return status 500 'Internal Error' rather than 501 'Not Implemented'. Likewise, for any invalid, incorrectly formed headers or requests, your proxy should return status 500 'Internal Error' rather than 400 'Bad Request' to the client.

Otherwise, your proxy should simply forward status replies from the remote server to the client. This means most 1xx, 2xx, 3xx, 4xx, and 5xx status replies should go directly from the remote server to the client through your proxy. (While you are debugging, make sure that 404 status replies from the

remote server are not the result of poorly forwarded requests from your proxy.)

Concurrent Requests(Optional)

A practical web proxy should be able to support multiple clients at the same time. You may choose appropriate library to add multi-thread support into your program.

Testing Your Proxy

Run your proxy with the following command:

`python http_proxy.py <port> &`, where `port` is the port number that the proxy should listen on. As a basic test of functionality, try requesting a page using telnet:

```
telnet localhost <port>
Trying 127.0.0.1...
Connected to localhost.localdomain (127.0.0.1).
Escape character is '^]'.
GET http://www.example.com/ HTTP/1.1
```

If your proxy is working correctly, the headers and HTML of example.com should be displayed on your terminal screen. Notice here that we request the absolute URL (`http://www.example.com/`) instead of just the relative URL (`/`). A good sanity check of proxy behavior would be to compare the HTTP response (headers and body) obtained via your proxy with the response from a direct telnet connection to the remote server. Additionally, try requesting a page using telnet concurrently from two different shells.

Then try testing your proxy with the supplied `test_proxy.py` script. This will compare the result of fetching 4 pre-determined websites directly versus through your proxy:
`python testing_scripts/test_proxy.py http_proxy.py [port (optional, will be random if omitted)]`

(This script requires `http_proxy.py` to be a executable file. `chmod +x http_proxy.py` can be used on Linux/Mac OS X. On windows, it should be setup by python installer already.)

Things to submit, make a **zip** file named as `csci379_prj2_yourlastname.zip` (it will be `csci379_prj2_chu.zip` for me) including all the files in the starter package.

Submit the zip file through blackboard.

Acknowledgement This assignment is adopted from Prof. Nick Feamster's course COS461 Spring 2017 at Princeton.