Task 2.2: Web Proxy

Building on the Web Server described in Task 2.1, this task is concerned with building a Web Proxy. This operates in much the same way as a web server, with one significant difference: once configured to use the Proxy Cache application, a client will make all requests for content **via** this proxy.

Normally, when we make a request (without a Web Proxy), the requests travels from the host machine to the destination. The Web Server then processes the request and sends back a response message to the requesting client. However, when we use a Web Proxy, we place this additional application between the client and the web server. Now, both the request message sent by the client, and the response message delivered by the web server, pass through the Web Proxy. In other words, the client requests the objects via the Web Proxy. The Web Proxy will forward the client's request to the web server. The web server will then generate a response message and deliver it to the proxy server, which in turn sends it to the client. The message flow is as below:

As with the Web Server, your Web Proxy application is only expected to handle HTTP/1.1 GET requests. Similarly, the Web Proxy will also bind to a specific port (this can be the same as the Web Server), and continue to listen on this port until stopped.

Debugging and Testing

As with Task 2.1, there are a number of ways to test your Web Proxy. For example, to generate requests using wget, we can use the following:

```
wget lancaster.ac.uk -e use_proxy=yes -e http_proxy=127.0.0.1:8000
```

This assumes that the Web Proxy is running on the local machine and bound to port 8000. In this case, the URL requested from the proxy is lancaster.ac.uk.

A web browser can also be used to the same effect. Many web browsers support the use of a web proxy through configuration. For example, the Chromium browser can be started in the following way to utilise the Web Proxy:

```
chromium-browser --proxy-server="127.0.0.1:8000"
```

A caveat when testing your Web Proxy: some websites have enabled HTTP Strict Transport Security (HSTS) (RFC6797). This forces clients (including both wget and a web browser) to use HTTPS rather than HTTP. HTTPS is a secure version of HTTP, but we will consider this out of scope for this practical. To check if a website has this feature enabled, use the following tool, and ensure that the website you are using for testing purposes is **not** listed:

https://hstspreload.org/

As with the other tasks, Wireshark can be used to capture and investigate packets sent to and from your proxy. As the proxy will be receiving local requests from the web browser, as well as making external requests to fetch content, it is necessary to

capture packets on both the external (eth0) and loopback (lo) interfaces.

Marking Criteria

For this task, the majority of marks will be awarded for demonstrating a working Web Proxy. You are expected to show the functionality of such through the use of either wget or a properly configured web browser. Note that you are not expected to demonstrate the Web Proxy using a website with HSTS enabled (see above).

Additional marks will be awarded for the following aspects:

- Binding the Web Proxy to a configurable port, defined as an optional argument
- Support for other HTTP request types (PUT, DELETE, etc.)
- Object caching: A typical Web Proxy will cache the web pages each time the client makes a particular request for the first time. The basic functionality of caching works as follows. When the proxy gets a request, it checks if the requested object is cached, and if yes, it returns the object from the cache, without contacting the server. If the object is not cached, the proxy retrieves the object from the server, returns it to the client and caches a copy for future requests. In practice, the proxy server must verify that the cached responses are still valid and that they are the correct responses to the client's requests. You can read more about caching and how it is handled in HTTP in RFC2068. Add the simple caching functionality described above. You do not need to implement any replacement or validation policies. Your implementation, however, will need to be able to write responses to the disk (i.e., the cache) and fetch them from the disk when you get a cache hit. For this you need to implement some internal data structure in the proxy to keep track of which objects are cached and where they are on the disk. You can keep this data structure in main memory; there is no need to make it persist.

As before, please note that the features mentioned above are considered supplementary; you do not have to complete them all, and you can still receive a satisfactory mark without completing *any* of them. They are intentionally challenging and designed to stretch you.