Extra Credit Task - Modify the dropGooglePackets.c to drop google.com traffic using the domain name, not a speicfic local IP

We ran into a few issues working on this task which we will detail below:

The goal of this task was to make it so that any request to visit google.com would be filtered by the firewall and blocked without needed to add static IPs. Google web sites have many, many servers throughout the world and these are load balanced depending on their usage. As such, blocking just one specific IP would only work if the browser attempted to visit that specific server. This is further complicated with how many different services fall under the Google umbrella. Google drive, Google photos, etc.

Our first attempt at getting this to work was to do a reverse lookup of whatever address the user visited in their browser; however, the reverse lookup address of Google servers shows a DNS name that is not a google.com suffix, but rather

```
C:\Users\Logan>nslookup google.com
Server:  cdns01.comcast.net
Address:  2001:558:feed::1

DNS request timed out.
    timeout was 2 seconds.
Non-authoritative answer:
Name:    google.com
Addresses:  2607:f8b0:4004:808::100e
          173.194.121.38
          173.194.121.40
          173.194.121.37
          173.194.121.34
          173.194.121.36
          173.194.121.46
          173.194.121.33
          173.194.121.41
          173.194.121.39
          173.194.121.32
          173.194.121.35
```

```
C:\Users\Logan>nslookup 173.194.121.39
Server:  cdns01.comcast.net
Address:  2001:558:feed::1

Name:    iad23s26-in-f7.1e100.net
Address:  173.194.121.39
```

This 1e100.net domain could be that of a hosting company, or could just be Google's cloud servers. Blocking anything that comes up as *.1e100.net could end up blocking far more than just google.com. We implemented this in our dropGooglePackets.c file, but after looking at this roadblock, we decided that it would not completely solve the task.

The other thought was to do a dns query for Google servers and store them in an array, that way we could do a comparison between the array and the address that was visited and see if it is known to be Google. The problem with this method, is it will still not return all of Google's servers. Each query only returns 11 results. Another option is to block everything on the 173.x.x.x class A and the 74.x.x.x class A; however, this too may not cover all of the Google addressess, and may block services we are not trying to block.

The final idea we had to block the traffic to google.com was to inspect the data part of the ip packet and look for the string google.com. Doing this we could decide if we wanted to include other services or not depending on the way we crafted the search. The goal of this method is to block any packets that have google.com in the data part of the packet. Unfortunately this would not work for any https packets as their data would be encrypted until it got to the browser; however, it would block any DNS queries to google.com and the client would be unable to get any addresses for its requests. This would be normally done on port 53 TCP and UDP, but we would just apply it to every protocol and have it block anything it sees as Google. This seems to be the best method to block a user's attempts to use Google and without pre-populated information (except the domain search string), but it would take more time than what we have to implement.