CS39006: Assignment 6
Implementing a Custom Protocol using Raw Sockets
Date: March 28, 2024
Due Date: April 08, 2024

In this assignment, you'll implement a custom protocol that works like a DNS (but much simpler than DNS) to find out the IP address corresponding to a domain name; let's call this protocol as simDNS. You need to implement a simDNS server and a simDNS client. The simDNS server runs a server that accepts a simDNS query packet and replies with a simDNS response packet. The protocol is not reliable, so there can be packet loss. simDNS runs on top of IP packets, and uses the IP protocol field as 254 (Find out what a protocol field 254 means).

A simDNS Query packet has the following fields:
i) ID: 16 bits (used to map the request with the response, if there are multiple requests from the same client)
ii) Message Type: 1 bit (0: Query, 1: Response)
iii) Number of queries: 3 bits (You can query for a maximum of 8 domain names through a single packet)
iv) Query strings: variable bytes (the size of each query string can be at most of 32 bytes; the first 4 bytes contain an integer that indicates the size of the domain name in character, and then the remaining bytes contain the actual domain name for the query. As you already know the size of the domain name strings, these strings need not have the null character at the end.)

A simDNS Response packet has the following fields:
i) ID: 16 bits
ii) Message Type: 1 bit (0: Query, 1: Response)
iii) Number of responses: 3 bits
iv) Responses: multiple of 33 bits (the first bit is a flag bit that indicates whether it is a valid response; if it is a valid response, then the remaining 32 bits contain the corresponding IP address. For one domain name, you should return only one IP address.)

The responses should be in the same order of the queries. For some query, if the server cannot find a response (the domain name does not have a valid IP address), the flag bit is set to 0, and the remaining 32 bits contain garbage values (may be all zeros). Otherwise, the flag bit is set to 1 and the rest of the 32 bits contain the corresponding IP addresses.

Implement the server as follows.

1. Open a raw socket to capture all the packets till Ethernet (use `ETH_P_ALL`).
2. Bind the raw socket to the local IP address.
3. Read all the packets that are received to this socket, check the IP header to see if the protocol field is 254, if not discard.
4. If the protocol field is 254, then read the query header. If it is a simDNS query, then parse the query strings.
5. Generate a simDNS response. For all the query strings, use `gethostbyname` (check the man page for details; if `gethostbyname` returns more than one IP for a name, return

the first one only.) to find out the corresponding IP address. Populate the response fields depending on whether the function returns a valid IP address or not.

8. Send the simDNS response to the client which had sent the query (you can find out the client IP address from the IP header).
7. Wait for the next query.

Implement the simDNS client as follows.

1. Open a raw socket to capture all the packets till Ethernet (use `ETH_P_ALL`).
2. The client waits for the query string from the user. The format for the query string is as follows:
   `getIP N <domain-1> <domain-2> <domain-3> … <domain-N>`
   where N is the number of request strings.
3. Check if N <= 8; if no, throw an error message and wait for the next query.
4. If N <=8, check if all the domain names follow the following formatting guidelines.
   a. Only alphanumeric characters are used,
   b. Hyphens can also be used but it cannot be used at the beginning and at the end of a domain name. Two consecutive hyphens are not allowed.
   c. Spaces and special characters (such as !, $, &, _ and so on) are not allowed, except the dot,
   d. The minimum length is 3 characters, and the maximum length is 31 characters.

   If it does not follow the formatting guidelines, then throw an error message and wait for the next query. You need to drop the entire query even if only one of them does not follow the correct format.
5. If all the query strings are valid, construct a simDNS query message and send it through the raw socket. Use a pending query table to insert the query ID (the query ID needs to be unique for each query from a client application). After sending the message, return back to the prompt.
6. Use the `select()` call to check if you have received a response. Note that you need to filter out the received packets based on the protocol number and the simDNS server IP. When a response is received, check the pending query table to see if a query with the same ID was sent earlier. If so, then parse the response and print the output as follows:

   ```
   Query ID: XXXXXXX
   Total query strings: N
   <domain-1> <IP-1>
   <domain-2> <IP-2>
   <domain-3> NO IP ADDRESS FOUND
   …
   <domain-N> <IP-N>
   ```

   Delete the ID from the pending query table, and return back to the prompt. On timeout over the `select` (you need to set the timeout accordingly), retransmit the query (with the same ID) if you have not received the response for that query in the pending query table. After three such retries, if you still do not receive the response, throw an error message to the prompt and delete the query ID from the pending query table. Finally, if the user types "`EXIT`" in the prompt, then close the raw socket and exit.

To test query failures in your code, write a `dropmessage(float prob)` function at the simDNS server (similar to the previous assignment) that takes a probability value as the input and drops the incoming simDNS queries accordingly. Test your code with different values of the drop probabilities.

What to Submit:

You need to submit two .c files – simDNSServer.c and simDNSClient.c, along with a makefile to compile the programs and generate the executables. Also include a README file to mention how to run your program.