Aleks Muarskas
Jacob McConnell 260706620
Group 58
ECSE 316 Assignment 1 Report
February 13th 2020

Section 1: Introduction

The main objectives of this assignment were to implement a domain name system client. The main challenges in the problem are parsing user requests, generating a query from the user request according to the DNS protocol, transmitting the request to the specified socket (or default), listening for a response to the request according to the user data, and finally parsing the DNS response to the request and displaying it to the user. Parsing the user requests was straightforward as we simply filled in relevant variables with data from the command line using the args[] array. If there was no tag for say the time argument we would use the default value for that variable. Generating a user request according to the DNS protocol was more challenging because it was more abstract and unlike parsing user input something neither of us had done before. Once we had the request we transmitted it through a socket whose number is either the default one or one specified by the user depending on the results of the parsing. Then we had to listen at that port until a response was received or the timeout period was exceeded. Once it is exceeded we must retransmit the request and repeat the process up to the maximum number of retries which is either parsed from the user request or the default. This process of requesting, listening, timing out and retrying is important because UDP can lose requests so without a timeout it would be possible that the request is lost and your program waits forever.

When a response was received the next challenge was to parse the result and make sure it matched the request. This was also more challenging cause it required understanding the details of the DNS protocol (which was the point of the lab). Once we parsed the response we extracted the answer to our query and displayed it along with the amount of time it took to get it and the number of times we had to try. A challenge here was there are lots of different possible ways each part of the response could look. For example, due DNS packet compression domain names in the QNAME, NAME, and RDATA fields might be represented with a pointer. This adds another layer of complexity that you need to consider at many steps of handling the data in the response. Through this assignment we got a more detailed understanding of each part of the DNS protocol and a more detailed view of how data is transmitted through the internet.

Section 2: DNS Client Program Design

Describe design of program

Our program has several major parts:

a) Default Values

Before the program parses the input, first the default values are assigned for timeout, maxRetries, and the port. In addition the expected location of the server and site name within the input are stored.

b) Parsing the input

A for loop iterates through the input command from the first token of the input to the last. If the token matches the symbol used for a flag ex. “-t”, then the client records the next value in the input stream. When the last two arguments are reached, they are recorded as the server and the website name. After the command line input parsing is completed, the IP address is created. First a substring is created to remove the @ symbol. An error check is done to ensure the first char of the server is the @ symbol. Using the String’s split method to break the IP address down into an array of substrings. An error check is run to ensure the address is made of 4 substrings.

Then a for loop occurs to parseInt each of the substrings into an integer, and then places them in an array of integers. An error check occurs within to ensure the parseInt can catch an exception, as well as check the parsed integer is within the range of 0 and 255.

Next the domain name length is calculated. First the string is split into tokens, then a for loop iterates over each token and adds the lengths of each token to the variable domainNameLength.

c) Creating the DNS message

Now the DNS creation begins. Our DNS message follows the layout shown in the handout. In order to easily manage large buffers of data, we used the ByteBuffer class to simplify the process. The packet’s header is created in the method createHeader(). An empty buffer is passed to the method, which will be returned once modified. A randomized qID is generated and placed as the first two bytes of the header. Lines two and three are initialized with 0x01 in the first bit of each line. ANCOUNT, NSCOUNT, ARCOUNT in the query are initialized to 0, they don't need to be explicitly set.

Afterward the query’s question is created within the method createQuestion(). The method takes in a buffer to have data entered into, the domain name in an array of tokens, and the type of query. QName is stored in the buffer first. The domain name is stored in memory by, for each token, storing the length of the token, then iterating over each character of the token and placing them in the buffer. Afterward a termination bit and buffer is added before QType is input. The first piece of Qtype is determined by the type passed and then the equivalent data byte is stored, for example the type of MX stores 0x000f. Afterwards a buffer is is created until the last byte which contains 1.

Finally the Query is built in the method createQuery(), which combines the buffers containing the header and question made previously.

d) Sending request and awaiting response

First the real IP address is found by using the InetAddress method getByAddress, and our client side socket is created. The data to be sent is stored in the byte array sendData by converting the query to an array. At this point the first output occurs, printing the domain name, server address and type. The program then loops from 0 to the maxRetries set by the user. Within each loop the outgoing and incoming packets are created. The socket sends the outgoing packet, and sets the socket to timeout at the user inputted timeout value. Within two statements that record the system time, the socket attempts to receive the incoming packet. If the packet reaches the socket, the loop is broken. If the loop iterates to the MaxRetries, the program notifies the user of the failure to receive the packet in the given number of attempts. Afterwards the socket is closed.

e) Decomposing the DNS into fields

First the message must be decomposed into the fields that make it up by bringing extracting the bits and bytes at certain points within the response. To do this we initialize a ResponseResults object using the constructor with the response DatagramPacket as an input. The DNS packet is then stored in a byte array called dataReceived, which is an attribute of ResponseResults. The rID is found in the first 2 bytes of the response. When the byte data is transferred from the array to a field, it is anded with 0xff, so that the value is copied, not a pointer. QR,AA,TC,RD,RA,and RCODE are found using this method. For values that are more than one byte or one bit, the values are combined using the equation ((dataRecieved[byte_1] << 8) | (dataRecieved[byte_2] & 0xFF)), which is then cast as a 16 bit short. We can access all of these header values as attributes of the ResponseResults object in the main.

f) Interpreting the DNS message

First the QR bit is checked to see if the message is a response. If it is not 1, then a notification is sent to the User to notify them the received packet is not a response and the program terminates. RA is checked to see if the response is recursive, which our program does not support. If AA is 1, the response is recorded as authoritative, otherwise it is nonauth. RCODE is then checked, if it 0 nothing needs to be done, otherwise the user is notified with the equivalent error message. The response time is calculated and then printed along with a number of attempts before the response was received.

g) Printing of Records

First the Answer records are printed. If ANCOUNT is 0 or less, no records were found. Otherwise the system iterates through the answer records, calling the method readRecord(). The method determines the response type and checks the class number. Depending on the response type, the records are printed accordingly. The standard response is the IP address, ttl value, and authoritative String. For MX, CNAME, and NS types the method readAlias() is used to decipher compressed records. Then the name service records are printed then printed, using a similar for loop. If ARCOUNT is greater than zero, the additional records are printed using a similar for loop to the Answer records, otherwise “No Additional Records Found”. Record printing has concluded and the program terminates.

Section 3: Testing

While completing this assignment, the first part that we tested was making sure that parsing the users input was done correctly. We did this by simply printing out each of the variables that could be populated by user inputs and running the partially done code several times with different requests. This was a routine check that we did just to be sure since it was pretty simple code. Fortunately we did do it because we had made a small mistake and had forgotten the logic to parse the -p flag with a custom port number. This problem would have been much harder to pin down later on since there would be more places it could have been in the code. We fixed it and went on to write the rest of the code.

Part a) Testing the -ns and -mx flags

In order to test the differences we will run the same command for the same site with a different flag.

Testing google with the flag:

java DnsClient @8.8.8.8 google.com

```
(base) Jacobs-MacBook-Pro:ournew jacobmcconnell$ java DnsClient @8.8.8.8 google.com
DNSClient sending request for google.com
Server: 8.8.8.8
Request Type: A
Response packet received after 0.194 seconds (0 retries)
*** Answer Section (1 records) ***
IP      216.58.194.174          89      nonauth
*** Additional Section (0 records) ***
NO Additional Records found
```

java DnsClient -ns @8.8.8.8 google.com

```
(base) Jacobs-MacBook-Pro:ournew jacobmcconnell$ java DnsClient -ns @8.8.8.8 google.com
DNSClient sending request for google.com
Server: 8.8.8.8
Request Type: NS
Response packet received after 0.266 seconds (0 retries)
*** Answer Section (4 records) ***
NS      ns4.google.com  16473   nonauth
NS      ns1.google.com  16473   nonauth
NS      ns3.google.com  16473   nonauth
NS      ns2.google.com  16473   nonauth
*** Additional Section (0 records) ***
NO Additional Records found
```

java DnsClient -mx @8.8.8.8 google.com

```
(base) Jacobs-MacBook-Pro:ournew jacobmcconnell$ java DnsClient -mx @8.8.8.8 google.com
DNSClient sending request for google.com
Server: 8.8.8.8
Request Type: MX
Response packet received after 0.898 seconds (0 retries)
*** Answer Section (5 records) ***
MX      alt2.aspmx.l.google.com 461     nonauth
MX      alt1.aspmx.l.google.com 461     nonauth
MX      alt3.aspmx.l.google.com 461     nonauth
MX      aspmx.l.google.com      461     nonauth
MX      alt4.aspmx.l.google.com 461     nonauth
*** Additional Section (0 records) ***
NO Additional Records found
```

Testing facebook with the flag:

java DnsClient @8.8.8.8 facebook.com

```
(base) Jacobs-MacBook-Pro:ournew jacobmcconnell$ java DnsClient @8.8.8.8 facebook.com
DNSClient sending request for facebook.com
Server: 8.8.8.8
Request Type: A
Response packet received after 0.432 seconds (0 retries)
*** Answer Section (1 records) ***
IP      157.240.206.35          299     nonauth
*** Additional Section (0 records) ***
NO Additional Records found
```

java DnsClient -ns @8.8.8.8 facebook.com

```
(base) Jacobs-MacBook-Pro:ournew jacobmcconnell$ java DnsClient -ns @8.8.8.8 facebook.com
DNSClient sending request for facebook.com
Server: 8.8.8.8
Request Type: NS
Response packet received after 0.226 seconds (0 retries)
*** Answer Section (4 records) ***
NS      d.ns.facebook.com       21257   nonauth
NS      c.ns.facebook.com       21257   nonauth
NS      b.ns.facebook.com       21257   nonauth
NS      a.ns.facebook.com       21257   nonauth
*** Additional Section (0 records) ***
NO Additional Records found
```

java DnsClient -mx @8.8.8.8 facebook.com

```
(base) Jacobs-MacBook-Pro:ournew jacobmcconnell$ java DnsClient -mx @8.8.8.8 facebook.com
DNSClient sending request for facebook.com
Server: 8.8.8.8
Request Type: MX
Response packet received after 0.255 seconds (0 retries)
*** Answer Section (1 records) ***
MX      smtpin.vvv.facebook.com 2775    nonauth
*** Additional Section (0 records) ***
NO Additional Records found
```

As shown above changing the flag successfully resulted in the different requested responses.

Part b) Testing the timeout value

Since we can't set fractional timeout value we tested this using the google DNS on the McGill wifi so that it would time out.

This is the default settings:

```
(base) Jacobs-MacBook-Pro:ournew jacobmcconnell$ java DnsClient @8.8.8.8 facebook.com
DNSClient sending request for facebook.com
Server: 8.8.8.8
Request Type: A
Communication ERROR: Maximum number of retries 3 reached
```

This is with the timeout increased to 10 seconds:

```
(base) Jacobs-MacBook-Pro:ournew jacobmcconnell$ java DnsClient -t 10 @8.8.8.8 facebook.com
DNSClient sending request for facebook.com
Server: 8.8.8.8
Request Type: A
Communication ERROR: Maximum number of retries 3 reached
```

Note that there is no indication in the program but it took twice as long to finish as I watched it.

Part c) Testing the maximum

The default setting:

```
(base) Jacobs-MacBook-Pro:ournew jacobmcconnell$ java DnsClient @8.8.8.8 facebook.com
DNSClient sending request for facebook.com
Server: 8.8.8.8
Request Type: A
Communication ERROR: Maximum number of retries 3 reached
```

Increasing the max tries to 4:

```
(base) Jacobs-MacBook-Pro:ournew jacobmcconnell$ java DnsClient -r 4 @8.8.8.8 facebook.com
DNSClient sending request for facebook.com
Server: 8.8.8.8
Request Type: A
Communication ERROR: Maximum number of retries 4 reached
```

By running the above commands we validated that the program works.

We did not specifically test the validity of the DNS packet we transmitted. We would have liked to but we did not have time. We did try to check the DNS packets that we received using nslookup. This actually helped us cause it showed that we were having issues with parts of the results we were displaying because we hadn't properly incremented our counters in some of the loops. Other than that though we would have liked to have done more testing.

Section 4: Experiment

Part a) What are the IP addresses of McGill's DNS servers? Use the Google public DNS server (8.8.8.8) to perform a NS query for mcgill.ca and any subsequent follow-up queries that may be required. What response do you get? Does this match what you expected?

Using the following command: java DnsClient –ns @8.8.8.8 mcgill.ca
We received the following response:

```
(base) Jacobs-MacBook-Pro:ournew jacobmcconnell$ java DnsClient -ns @8.8.8.8 mcgill.ca
DNSClient sending request for mcgill.ca
Server: 8.8.8.8
Request Type: NS
Response packet received after 0.133 seconds (0 retries)
*** Answer Section (2 records) ***
NS      pens2.mcgill.ca 652     nonauth
NS      pens1.mcgill.ca 652     nonauth
*** Additional Section (0 records) ***
NO Additional Records found
```

Q) Does this match what we expected?

Yes we got a response and we didn't have any particular expectations about it. The names to our request include mcgill in them so it seems like the response was not nonsense is related to the McGill server.

Part b) Use our client to run DNS queries for 5 different website addresses, of our choice, in addition to www.google.com and www.facebook.com, for a total of seven addresses. Query the seven addresses using both a McGill DNS server (132.206.85.18) and the Google public DNS server (8.8.8.8).

In our experience the McGill DNS server only worked on mcgill wifi, and the public DNS server did not work on mcgill wifi(tethered using an american cell phone plan).

1. Google.com

   Using the command: java DnsClient @8.8.8.8 google.com

   We received the following response:

```
(base) Jacobs-MacBook-Pro:ournew jacobmcconnell$ java DnsClient @8.8.8.8 google.com
DNSClient sending request for google.com
Server: 8.8.8.8
Request Type: A
Response packet received after 0.358 seconds (0 retries)
*** Answer Section (1 records) ***
IP      216.58.194.174          69      nonauth
*** Additional Section (0 records) ***
NO Additional Records found
```

   Using the command: java DnsClient @132.206.85.18 google.com

   We received the following response:

```
(base) Jacobs-MacBook-Pro:ournew jacobmcconnell$ java DnsClient @132.206.85.18 google.com
DNSClient sending request for google.com
Server: 132.206.85.18
Request Type: A
Response packet received after 0.006 seconds (0 retries)
*** Answer Section (1 records) ***
IP      172.217.13.110          33      nonauth
*** Additional Section (0 records) ***
NO Additional Records found
```

2. Facebook.com

   Using the command: java DnsClient @8.8.8.8 facebook.com

   We received the following response:

```
(base) Jacobs-MacBook-Pro:ournew jacobmcconnell$ java DnsClient @8.8.8.8 facebook.com
DNSClient sending request for facebook.com
Server: 8.8.8.8
Request Type: A
Response packet received after 0.274 seconds (0 retries)
*** Answer Section (1 records) ***
IP      157.240.206.35          299     nonauth
*** Additional Section (0 records) ***
NO Additional Records found
```

   Using the command: java DnsClient @132.206.85.18 facebook.com

   We received the following response:

```
(base) Jacobs-MacBook-Pro:ournew jacobmcconnell$ java DnsClient @132.206.85.18 facebook.com
DNSClient sending request for facebook.com
Server: 132.206.85.18
Request Type: A
Response packet received after 0.007 seconds (0 retries)
*** Answer Section (1 records) ***
IP      31.13.80.36             11      nonauth
*** Additional Section (0 records) ***
NO Additional Records found
```

3. Reddit.com

   Using the command: java DnsClient @8.8.8.8 reddit.com

   We received the following response:

```
(base) Jacobs-MacBook-Pro:ournew jacobmcconnell$ java DnsClient @8.8.8.8 reddit.com
DNSClient sending request for reddit.com
Server: 8.8.8.8
Request Type: A
Response packet received after 0.135 seconds (0 retries)
*** Answer Section (4 records) ***
IP      151.101.1.140           153     nonauth
IP      151.101.129.140         153     nonauth
IP      151.101.193.140         153     nonauth
IP      151.101.65.140          153     nonauth
*** Additional Section (0 records) ***
NO Additional Records found
```

Using the command: java DnsClient @132.206.85.18 reddit.com

We received the following response:

```
(base) Jacobs-MacBook-Pro:ournew jacobmcconnell$ java DnsClient @132.206.85.18 reddit.com
DNSClient sending request for reddit.com
Server: 132.206.85.18
Request Type: A
Response packet received after 0.011 seconds (0 retries)
*** Answer Section (4 records) ***
IP      151.101.129.140         300     nonauth
IP      151.101.65.140          300     nonauth
IP      151.101.193.140         300     nonauth
IP      151.101.1.140           300     nonauth
*** Additional Section (0 records) ***
NO Additional Records found
```

4. Twitter.com

   Using the command: java DnsClient @8.8.8.8 twitter.com

   We received the following response:

```
(base) Jacobs-MacBook-Pro:ournew jacobmcconnell$ java DnsClient @8.8.8.8 twitter.com
DNSClient sending request for twitter.com
Server: 8.8.8.8
Request Type: A
Response packet received after 0.137 seconds (0 retries)
*** Answer Section (2 records) ***
IP      104.244.42.65           531     nonauth
IP      104.244.42.129          531     nonauth
*** Additional Section (0 records) ***
NO Additional Records found
```

   Using the command: java DnsClient @132.206.85.18 twitter.com

   We received the following response:

```
(base) Jacobs-MacBook-Pro:ournew jacobmcconnell$ java DnsClient @132.206.85.18 twitter.com
DNSClient sending request for twitter.com
Server: 132.206.85.18
Request Type: A
Response packet received after 0.005 seconds (0 retries)
*** Answer Section (2 records) ***
IP      104.244.42.129          103     nonauth
IP      104.244.42.1            103     nonauth
*** Additional Section (0 records) ***
NO Additional Records found
```

5. Linkedin.com

   Using the command: java DnsClient @8.8.8.8 linkedin.com

   We received the following response:

```
(base) Jacobs-MacBook-Pro:ournew jacobmcconnell$ java DnsClient @8.8.8.8 linkedin.com
DNSClient sending request for linkedin.com
Server: 8.8.8.8
Request Type: A
Response packet received after 0.117 seconds (0 retries)
*** Answer Section (1 records) ***
IP      108.174.10.10           1837    nonauth
*** Additional Section (0 records) ***
NO Additional Records found
```

Using the command: java DnsClient @132.206.85.18 linkedin.com

We received the following response:

```
(base) Jacobs-MacBook-Pro:ournew jacobmcconnell$ java DnsClient @132.206.85.18 linkedin.com
DNSClient sending request for linkedin.com
Server: 132.206.85.18
Request Type: A
Response packet received after 0.006 seconds (0 retries)
*** Answer Section (1 records) ***
IP      108.174.10.10           610     nonauth
*** Additional Section (0 records) ***
NO Additional Records found
```

6. Euswiki.ca

   Using the command: java DnsClient @8.8.8.8 euswiki.com

   We received the following response:

```
(base) Jacobs-MacBook-Pro:ournew jacobmcconnell$ java DnsClient @8.8.8.8 euswiki.com
DNSClient sending request for euswiki.com
Server: 8.8.8.8
Request Type: A
Response packet received after 0.329 seconds (0 retries)
*** Answer Section (1 records) ***
IP      156.241.148.190         599     nonauth
*** Additional Section (0 records) ***
NO Additional Records found
```

   Using the command: java DnsClient @132.206.85.18 euswiki.com

   We received the following response:

```
(base) Jacobs-MacBook-Pro:ournew jacobmcconnell$ java DnsClient @132.206.85.18 euswiki.com
DNSClient sending request for euswiki.com
Server: 132.206.85.18
Request Type: A
Response packet received after 0.373 seconds (0 retries)
*** Answer Section (1 records) ***
IP      156.241.148.190         600     nonauth
*** Additional Section (0 records) ***
NO Additional Records found
```

7. Youtube.com

   Using the command: java DnsClient @8.8.8.8 youtube.com

   We received the following response:

```
(base) Jacobs-MacBook-Pro:ournew jacobmcconnell$ java DnsClient @8.8.8.8 youtube.com
DNSClient sending request for youtube.com
Server: 8.8.8.8
Request Type: A
Response packet received after 0.555 seconds (0 retries)
*** Answer Section (1 records) ***
IP      216.58.194.206          299     nonauth
*** Additional Section (0 records) ***
NO Additional Records found
```

Using the command: java DnsClient @132.206.85.18 youtube.com

We received the following response:

```
(base) Jacobs-MacBook-Pro:ournew jacobmcconnell$ java DnsClient @132.206.85.18 youtube.com
DNSClient sending request for youtube.com
Server: 132.206.85.18
Request Type: A
Response packet received after 0.025 seconds (0 retries)
*** Answer Section (1 records) ***
IP      172.217.13.206          300     nonauth
*** Additional Section (0 records) ***
NO Additional Records found
```

Part c) Have the responses we've seen above matched?

Most of the time the above responses did not match. However they did match (ie we got the same ip address) for euswiki.com. We suspect this is because larger websites like youtube and facebook have multiple servers some of which might be closer to the McGill DNS records vs the public google one. However the euswiki probably only has one server because it is a small website. That's why we got the same result for both attempts.

Section 5: Discussion

Close with a brief discussion of the key observations obtained in this experiment. What were your main results? What were the main challenges in implementation? How could these possibly be addressed by a different approach?

The main observation from the experiment is that we got different results for different DNS records for large websites but we got the same result for small websites that are probably only hosted in one place. This indicates that large websites are probably hosted with multiple ip addresses.Main challenges were using the buffers to store the data correctly and ensuring that we were storing the correct bytes. Additionally it was difficult to travel through the records because we were not incrementing properly through the records. This created incorrect results but it was difficult to tell that this was happening. After carefully looking through our code we realized that sometimes we were not incrementing and fixed this. But since all of the data is in bytes and in a format meant for machines to understand it is hard to visualize what the problems can be. I think if we had taken an more modular object oriented approach it might have made each part of the process and of the DNS record more understandable for us as programmers and therefore easier to discover mistakes(we did do a little of this with the ResponseResults object but we definitely could have done a better job throughout).