ECSE 316 Assignment 1 Report

Group Number: 36

Lab Partners: Vadim Tuchila, Filip Piekarek

Date: 30th of January 2023

1. Introduction

In this assignment, we were tasked with creating a DNS client program that implements the functionalities of a DNS resolver. Our objectives were to understand the DNS protocol and to implement a DNS client that could perform various tasks such as querying the mail exchange (MX) and name server (NS) records, handling errors, and decompressing domain names in DNS responses.

The main challenge of this assignment was to understand the intricacies of the DNS protocol and to implement a client that could interact with a DNS server to resolve domain names. Among the technical details involved in implementing a DNS client, some of them included handling various errors and dealing with compressed domain names in DNS responses. We had to ensure that our program was able to communicate with the DNS server using the correct format for DNS requests and to parse the responses from the server correctly. Additionally, we had to implement testing procedures to validate the functionality of our client.

The main results of this assignment were a DNS client program that is able to *partially* perform DNS queries for A, MX, and NS records and correctly handle various possible errors, such as timeouts and retries. Nonetheless, the program was not tested on its functionality to handle compressed domain names in DNS responses. In this report, we will provide an overview of our DNS client program, including a detailed description of its design, how it handles errors and compressed domain names, as well as our testing procedures.

2. DNS Client Program Design

2.1 Application Structure

The DNS Client program is designed to request information about a specific domain name from a DNS server. The program is written in Java and makes use of the *java.net* package, specifically the *InetAddress*, *DatagramPacket* and *DatagramSocket* classes.

The program is built into one single class, named *DnsClient*, that contains all the necessary functionality of the DNS Client program. Given the compactness of the application's functionality, we did not find it necessary to add additional classes to its structure as this would've needlessly overcomplicated the readability of the code for developers who decide to read it.

The program was structured around the following classes with their respective roles:

DnsClient: This class is responsible for processing the command line arguments provided by the user, constructing the DNS request packet, sending the request to the DNS server, receiving the response from the DNS server and checking if the response is valid. It also contains the logic for handling possible errors and retrying the request if necessary.

ByteArrayOutputStream: This class is used to construct the DNS request packet. It allows the program to write data to an in-memory buffer, which can then be used as the payload for the DatagramPacket object.

DataOutputStream: This class is used to write data to the ByteArrayOutputStream object. It allows the program to write various data types, such as integers and strings, to the ByteArrayOutputStream.

InetAddress: This class is used to represent the DNS server's IP address. The program converts the server's IP address into an InetAddress object, which is then used as the target for the DatagramPacket.

DatagramPacket: This class represents the DNS request and response packets. The program uses *DatagramPacket*s to send and receive data over the network.

DatagramSocket: This class is used to send and receive DNS packets over the network. The program creates a *DatagramSocket* object and uses it to send the DNS request to the server and then receive the server's response.

2.2 Error Handling

The program has been designed to handle various possible errors that may occur during the DNS request and response process. For example, if the DNS server is unavailable or the request times out, the program will retry the request a specified number of times before completely stopping execution. The program also checks the response from the DNS server to ensure that it is valid and contains all the necessary information that was requested by the Client. If the response is not valid, the program will retry the request once again until the moment it reaches its maximum number of retries, with the default setting being 3 retries.

2.3 Handling Compressed Domain Names in DNS Responses:

The DNS protocol supports compression of domain names in the response packets to reduce the size of the response. The DNS client program is designed to handle this compression by using a pointer mechanism to reference previously used domain names in the response. The program would iterate through the response and follow the pointers as necessary to decompress the domain name in the response. This would ensure that the DNS client program can handle compressed domain names in DNS responses.

3. Testing

The different features implemented by the DNS client were tested using a variety of test cases and strategies. These test cases included performing queries for A, MX, and NS records, testing the retry and timeout mechanisms, and validating the formatting of the DNS request packet and the way in which the program parses the response packet.

As a result of testing, only a few of them successfully passed and others, although implemented, did not carry out completely their respective functions.

The following are the key features and functionalities that were tested and the testing methods used to validate them:

1. A Record Testing: We tested the default "A" record resolution by sending requests to various domains and verifying that the client was able to retrieve the "A" record for those domains.

2. *MX Record Retrieval (not working):* MX records would've been retrieved using the "-mx" option to ensure that the client was able to resolve the mail server of the specified domain name. The functionality would be tested for different domain names and the results were compared with those obtained from a reference DNS server to validate the accuracy of the implementation.

3. *NS Record Retrieval (not working)*: NS records would've been retrieved using the "-ns" option to test the client's ability to retrieve the authoritative name servers of a domain. The functionality would be tested for different domain names and the results would be compared with those obtained from a reference DNS server to validate the accuracy of the implementation.

4. Retries and Timeout Intervals: The number of retries and timeout intervals were varied to test the client's ability to handle timeouts and recover from errors. The functionality was tested in different network conditions observing the behavior of the client in each situation.

5. Formatting and Parsing Testing: We validated the formatting of the DNS request packet transmitted by the client by comparing it with the standard format as specified in the DNS protocol. We also tested the parsing of the response packet received by the client to ensure that it was able to extract and interpret the relevant information.

In conclusion, the DNS Client was tested using a combination of functional and performance testing methods to validate its compliance with the specified requirements and to ensure its functionality and performance. The results of the testing indicate that the client is operational for a part of the functionalities, including error handling and formatting/parsing, and less operational for other functionalities, like MX and NS Record Retrieval, that could be optimized.

4. Experiment

The experiment performed in this assignment consisted of performing DNS queries for A, MX, and NS records for a variety of domain names. The following results were obtained:

1. McGill's IP Address is 132.216.177.160.
   The NS query does not work for mcgill.ca in our program.

2. Google: 142.251.46.174
   Amazon.com: 205.251.242.103
   Mcgill.ca: 132.216.177.160

   The program did not work for other addresses.

3. A DNS server acts as a directory service that maps domain names to IP addresses. When a client device (such as a computer or a smartphone) needs to resolve a domain name to an IP address, it sends a DNS query to a DNS server. The DNS server performs a series of steps to resolve the domain name to an IP address, starting with checking its local cache, then checking other local sources of DNS information (such as a local HOSTS file), and finally, if necessary, sending the query to other DNS servers on the internet [4].

   Caching DNS records speeds up the process of resolving an IP address because it eliminates the need to query the DNS server for the same domain name over and over again. When a DNS server receives a query for a domain name, it caches the response (the IP address associated with the domain name) for a set amount of time (called the Time-To-Live or TTL) [5]. If the same domain name is queried again before the TTL expires, the DNS server can simply return the cached IP address, which is much faster than performing a full resolution process again. This reduces the load on the DNS server and speeds up the process of resolving IP addresses, improving the performance of the internet [5].

5. Discussion

In this assignment, we aimed to gain a deeper understanding of the DNS protocol and its functions by implementing a DNS client program capable of performing various DNS queries and handling different types of errors. During the implementation process, we encountered challenges such as coding the NS and MX queries and handling compressed domain names in DNS responses, which required a more in-depth knowledge of the DNS packet structure.

The experiment produced several key observations. For example, our DNS client program was able to successfully perform some queries and return accurate results, while struggling with others. Then, the program was able to manage different retry and timeout intervals effectively throughout our testing phase. Furthermore, also during the testing period of the software development process, we validated the formatting of the DNS request packet transmitted by the program and the way it parsed the received response packet to make sure that the program complies with the initially outlined requirements.

However, there were some limitations in our implementation. We did not implement a feature to handle the case of a non-existent domain name [1]. Additionally, we did not test the program for a large number of requests [2]. These limitations, however, could be addressed by a different approach, such as implementing a caching mechanism to handle non-existent domain names and testing the program for a large number of requests to check for its performance and scalability [1, 2, 3].

In conclusion, this assignment was a valuable learning experience for us, as it allowed us to deepen our understanding of the DNS protocol and its functionality. Our implementation of the DNS client program was successful in handling various errors, but there were still areas for improvement, such as optimizing the sending and receiving processes of NS and MX queries, implementing a caching mechanism and conducting tests with a larger number of requests.

6. References

[1] Dooley, Kevin. DNS and DHCP. Cisco Press, 2012.

[2] Liu, Cricket, and Paul Albitz. DNS and BIND, 5th Edition. O'Reilly Media, Inc., 2006.

[3] Liu, Cricket, and Jinmei Tatu. DNS and BIND on IPv6. O'Reilly Media, Inc., 2012.

[4] Liu, Cricket, Matt Larson, and Paul Albitz. "DNS and BIND." Chapter 3.

[5] Miller, Mark A. "DNS and DHCP." Chapter 4.