Pouria SALEHI KATOZI s192865

DNS Thunelling - Client side

INFO0010-4 : Introduction to computer networking

Université de Liège
Faculté des Sciences Appliquées
Année académique 2022-2023

1 Introduction

This project consists of DNS client implementation following RFC 1035 documentation and is capable of sending DNS queries over TCP for specific types of DNS records and retrieving them. This project is developed using java programming and it contains four classes which are Headr.java, question.java, Client.java and Query.java.

2 Software architecture

The initial phase of this project involves gaining a comprehensive understanding of RFC 1035 protocol. Furthermore, it is essential to break down and divide the problem into the manageable subproblems in order to find a solution for each subproblem that leads us to solve the main problem efficiently.

2.1 Client

This class serves as the DNS client class and is responsible for initializing the client class with IP address, port, domain name and question type and the DNS header is initialized with certain flags and information. Depending on the question type specified, either an "A" or "TXT" record, the appropriate question is created. The Query class is used to interact with the DNS server, sending the query and receiving the response. The response is then processed by the output method to display the answer in a human-readable format. This class also contains methods for parsing and displaying DNS query answers. All operations are executed while adhering to proper error handling, ensuring a robust and reliable execution.

```
1  public class Client {
2      public String ipAddress;
3      public String questionType;
4      public String domain;
5      public int port = 53;
6      // Constructor to initialize client information
7      public Client(String ipAddress, int port, String domain, String
           questionType) {
8          this.port = port;
9          this.ipAddress = ipAddress;
10         this.domain = domain;
11         this.questionType = questionType;
12     }
13  public static void main(String[] args) throws
       UnsupportedEncodingException {
14         // Checking command line arguments
15         if (args.length == 2 || args.length == 3) {
16             if (args.length == 3) {
17                 argDefault = args[2];
18             } else { argDefault = "A"; }
19             // Creating client with provided arguments
20             client = new Client(args[0], 53, args[1], argDefault);
21             header = new Header((short) randomId, (short)
                   0b0000000100000000, (short) 1, (short) 0, (short)
                   0,(short) 0);
22             if (client.questionType.equals("A")) {
23                 question = new Question(tmp, (short) 1, (short) 1);
24             } else {
25                 question = new Question(tmp, (short) 16, (short) 1);
26             }
27             Query q = new Query();
28             //Dislaying answer if there is no exception
29         } else { /*Some Error Message*/ }}}
```

2.2 Header

This class represents the DNS header section of the query and it provides methods for creating and calculating header's length.

```
public class Header {

    public  ByteBuffer bytebuffer;
    public Header(short ID, short Flage,
        short QDCOUNT, short ANCOUNT, short NSCOUNT, short ARCOUNT){
        this.bytebuffer = ByteBuffer.allocate(12);
        this.bytebuffer.putShort(ID);
        this.bytebuffer.putShort(Flage);
        this.bytebuffer.putShort(QDCOUNT);
        this.bytebuffer.putShort(NSCOUNT);
        this.bytebuffer.putShort(ARCOUNT);
    }
    public int length(){
        return bytebuffer.array().length;
    }
```

2.3 Question

Question class represents the question section of the DNS query. Its task is to prepare and include the question data in the DNS query message.

```
public class Question {
    public byte[] QNAME;//Byte array for storing the domain name in the
       question
    public Short QTYPE; //Question type (e.g., A or TXT record)
    public Short QCLASS;//Question class (e.g., IN for Internet)
    public ByteBuffer byteBuffer; //ByteBuffer for assembling the
       question data
    public Question(byte[] QNAME, Short QTYPE, Short QCLASS) {
        byte b = 0;
        this.QNAME = QNAME;//Store the provided domain name
        this.QTYPE = QTYPE;//Store the provided question type
        this.QCLASS = QCLASS;//Store the provided question class
        this.byteBuffer = ByteBuffer.allocate(QNAME.length + 2 + 2);
           //Allocate memory for ByteBuffer
        this.byteBuffer.put(this.QNAME);      // Put domain name into
           ByteBuffer
        this.byteBuffer.putShort(this.QTYPE); // Put question type into
           ByteBuffer
        this.byteBuffer.putShort(this.QCLASS);// Put question class
           into ByteBuffer
        this.byteBuffer.rewind();  // Rewind the ByteBuffer to the
           beginning
    }
    public int length() {
        return QNAME.length; // Return the length of the domain name
    }
}
```

2.4 Query

The Query class is responsible for communicating by using socket programming inorder, to send a DNS query over TCP to a DNS server and receiving the response.

```
public class Query{

    public byte[] query(byte[] bytesToSend, Client c, short lenghtMsg)
        throws IOException {
    //Initiate a new TCP connection with a Socket
    Socket socket = new Socket(c.get_ipAddress(),  53);
    OutputStream out = socket.getOutputStream();
    InputStream in = socket.getInputStream();
    socket.setSoTimeout(5000);
    byte[] b = new byte[2];
    ByteBuffer.wrap(b).order(ByteOrder.BIG_ENDIAN).asShortBuffer()
    .put(lenghtMsg);
    out.write(b);
    out.flush();
    //send a query in the form of a byte array
    out.write(bytesToSend);
    out.flush();
    //isRetrive the reponse lenght, as described in RFC 1035 (4.2.2 TCP
        usage)
    byte[] lengthBuffer = new byte[2];
    in.read(lengthBuffer); //Verify it returns 2
    //convert bytes to length (data sent over the network is always
        big-endian)
    int length = ((lengthBuffer[0] & 0xff) << 8) | (lengthBuffer[1] &
        0xff);
    //Retrieve the full reponse
    byte[] reponseBuffer = new byte[length];
    in.read(reponseBuffer); //Verify it returns the value of "lenght"
    in.close();
    socket.close();
    return reponseBuffer;
    }
}
```

3 Message-oriented communication using a stream

The program and DNS server communicate with each other by establishing a socket communication channel over TCP. The DNS query message is constructed using a byte buffer that includes the DNS header, question, and other essential information. This message is subsequently transmitted through the established TCP socket using output streams (OutputStream) to write the data. The method expects a response, which is received as an array of bytes containing the answers to the query. The recovery of the response involves several steps :

1. Creating a TCP socket communication channel

```
    //Initiate a new TCP connection with a Socket
    Socket socket = new Socket(c.get_ipAddress(),  53);
    OutputStream out = socket.getOutputStream();
    InputStream in = socket.getInputStream();
```

2.The 6th and 7th elements of the byte array are examined to determine the number of answers and they are stored in a variable called ANCOUNT. This count is crucial for processing the response accurately.

```
int ANCOUNT = queryAnswer[6] << 8; // Shifting 6th byte to the left
    ANCOUNT += queryAnswer[7];
```

3.Significant details such as the record type (TYPE), Time-to-Live (TTL), and Resource Data Length (RDATA) are positioned in the byte array after the 32nd element. The program calculates the appropriate positions for TYPE and TTL to extract their values. Additionally, the process involves further steps to determine the length of the data (RDATA), which specifies the size of the RDATA section. The program iterates through the array of answers, executing the previously mentioned steps for each answer. This iterative approach ensures the recovery and processing of all pertinent information.

```
        //Extract answer type
        byte[] type = new byte[2];
        type[0] = queryAnswer[s + 0];
        type[1] = queryAnswer[s + 1];
        int typeTmp = convert(type);
        String QuestionTypes[] = {"A", "NS", "MD", "MF", "CNAME",
            "SOA", "MB", "MG", "NULL", "WKS", "WKS", "PTR", "HINFO",
            "MINFO", "MX", "TXT"};
        String c = QuestionTypes[typeTmp - 1];
        s += 4;
        //Extract TTL
        byte[] ttl = new byte[4];
        ttl[0] = queryAnswer[s + 0];
        ttl[1] = queryAnswer[s + 1];
        ttl[2] = queryAnswer[s + 2];
        ttl[3] = queryAnswer[s + 3];
        int ttlTmp = (((ttl[0] & 0xff) << 24) | ((ttl[1] & 0xff) << 16)
            | ((ttl[2] & 0xff) << 8) | (ttl[3] & 0xff));
        s += 4;
        //Extract length of data
        byte[] lendata = new byte[2];
        lendata[0] = queryAnswer[s];
        lendata[1] = queryAnswer[s + 1];
        int lendataTmp = convert(lendata);
        //Extract RDATA
        byte[] rdata = new byte[lendataTmp];
        s += 2;
        for (int j = 0; j < lendataTmp; j++) {
            rdata[j] = queryAnswer[s + j];
        }
        s += lendataTmp;
```

4 Limits and Possible Improvements :

4.1 limits

The program is only as good as the DNS server that it is querying. Any unreliability or inefficiency of the server significantly affects this program. In addition, there is a lack of support for recursive queries, multiple questions in a single query, and other question types.

4.2 Possible Improvements

Some possible improvements involve adding more precise error handling and exceptions while creating, reading from, or writing to the socket. Making the code much more modular by splitting some methods into smaller ones and adding additional features to enhance the program's capability to work with other types of DNS records and multithreading to improve performance of program.