University of Liège

Introduction to computer networking (INFO0010-4)

Second part of the assignment

Author :
Louis Hogge s192814
Juliette Waltregny s194379

Professor : G. Leduc
Assistant : M. Aerts
Year : 2022-2023

1 Software architecture

We have broken down the problem in 7 major classes, each one responsible for a specific part of the work :

1. **Server.java :** *which contains 2 classes*
   (a) ***class Server :***
   Main class responsible for accepting client queries and creating threads to handle them.
   (b) ***class RunnableClient implements Runnable :***
   Class which contains the overridden *run* method. Each thread in charge of managing a client query use this method to decode the query and write an appropriate server response.
2. **Message.java :**
   A class responsible for handling message objects.
   — Reads them by separating the query in 2 parts : header and question. Dispatches these parts to the classes and methods in charge to handle them.
   — Writes them by creating header, question and, for those containing an answer, resource records thanks to a call to the classes and methods in charge, calculates the length of the message and concatenates it with the header, the question and, when needed, resource records.
3. **Header.java :**
   A class responsible for handling the header part of a message.
   — Reads it and verifies if each component has the right value. Sends the appropriate error code.
   — Write them by concatenating the right bits/bytes in order to have all the required components : ID, flags, QDCOUNT, ANCOUNT, NSCOUNT and ARCOUNT.
4. **Question.java :**
   A class responsible for handling the question part of a message.
   — Reads it and verifies if each component has the right value. Sends the appropriate error code.
   — Write them by concatenating the right bits/bytes in order to have all the required components : QNAME, QTYPE and QCLASS. It basically reuse the one sent by the client in its query.
5. **ResourceRecord.java :**
   A class responsible for handling the resource records part of a message. Here there is only a method to write some resource records that goes through each component : NAME, TYPE, CLASS, TTL, RDLENGTH and RDATA. NAME is the decoded domain name of the encapsulated base32 url and RDATA contains our base64 encoded HTTP response formatted in <character-string>s as described in TXT RDATA definition in RFC 1035.
6. **RequestHTTP.java :**
   Class responsible for making HTTP requests and returning HTTP responses. Decodes the base32 url that comes in argument, performs the request and encodes the response in base64 before returning it.

2 Length limit

The hard limit on the HTTP response size is due to the size limit on TCP packets. Indeed, the packet size is limited by MTU of network resources. If the message size exceeds this limit, the Transport layer will split it into multiple packets of maximum size 65535 (60000 in our case).

The solution to overpass this limit is to send multiple packets with the data of the message. These packets will be reassembled after being received and the full message will be reconstructed. This method enables to send larger messages.

3 DNS Tunneling in practice

To send the queries to an uncontrolled DNS server, this said server must know the domain name used to be able to perform DNS recursion and hunt down the correct ip address of the targeted server. This is done by enrolling into a registrar and buying a domain name that will identify the server.
The server therefore lacks a "Name Server".

Furthermore, to perform DNS recursion, the server must support DNS decapsulation and do it recursively until the wanted data is accessible.

4 Limits & Possible Improvements

The limits of our code, in terms of robustness are the following :

— We do not check whether the input argument of the *makeHTTPRequest* method is valid, which could trigger errors.

— We could have also set a timeout when establishing HTTP URL connections, to avoid blocking the other processes if the server were to be unreachable.

— We could also set a read timeout, which could occur if the server is taking to long to read and send information. This could be due to slow internet connection or if the host is offline and could trigger unwanted behaviours, which is why a timeout might stop it.

— We do not check if the encapsulated url is really encoded in base32, we just assume it. When it's not the case, we still perform the decoding step which results in some random string that creates a name error with rcode 3 although the encapsulated url might be valid.

— In the question part *QNAME* has the following form : <encapsulated base32 encoded url>.<owned domain name>. In the resource record part *NAME* has the following form : <encapsulated base32 decoded url>. The choice of having in *QNAME* the encapsulated url encoded in base32 and in *NAME* the same encapsulated but this time decoded has been done to avoid handling compression in resource records writing. No repeated bytes so no compression needed.

In terms of improvements :

— We implemented GET HTTP requests. We could think about implementing the POST, PUT, PATCH and DELETE requests.

— Error management is not uniform, it is due to the need to print *NAME* and *TYPE* to stdout which means that, in our architecture, certain methods cannot be terminated by throwing a new exception.

— We could implement the compression support for the resource records part.

— Overall, we could think about implementing multihops DNS tunneling as suggested in Point 3.