Experiment 1: Network Programming and DNS

1 Introduction

In this lab you will implement a *domain name system* (DNS) client using sockets in Java. If you prefer, the system can also be implemented in Python.

At the end of this lab you should know how to:

1. Understand an existing network protocol specification;
2. Design a network application that adheres to the given protocol; and
3. Implement and test the network application using sockets.

2 Background

2.1 The Domain Name System (DNS)

The internet's domain name system provides the essential service of mapping between domain names (e.g., `mcgill.ca` or `facebook.com`) and IP addresses (e.g., `132.216.177.160` or `31.13.65.1`). This makes the internet more human-friendly, since we only have to remember domain names and not IP addresses.

When you direct an internet-enabled application to a new site (e.g., you open a new website in your browser, or you send an email to someone at a different domain name), the application needs to resolve the domain name to an IP address before it can contact the server. This is accomplished using the DNS request/response mechanism. For more about the DNS system and the protocol it uses, see Section 2.4 of Kurose and Ross (7th edition) or Section 2.5 in Kurose and Ross (6th edition) and the primer document posted along with these instructions on myCourses. You may also look at RFC 1035 (Sections 3 and 4) which give the official description of the DNS resource record format and message format.

2.2 Socket programming

Sockets are the programming mechanism used to implement network applications. If you have previously written an application that reads or writes from a file on disk, then you know you first open a file handle, then read/write using the file handle, and finally close the file handle when you are done. Sockets work in a similar way, but instead of reading/writing to a file on disk, they allow you to read/write to another process (usually running on a different machine).

For more on socket programming read Section 2.7 in Kurose and Ross (6th Edition - Java or 7th edition - Python) and the Java Networking Tutorial from Oracle. If you choose the Python approach, you can read the material available at https://docs.python.org/2/howto/sockets.html

2.3 Summary of background and pre-requisites

Before proceeding with the rest of this lab document make sure you have accomplished these first steps:

1. You are familiar with basic Java or Python programming, including how to compile and execute a program from the command line. We also recommend that you are familiar with an *integrated development environment (IDE)*, such as Eclipse, for debugging.
2. You have read the background on the DNS and on java/Python socket programming mentioned above.

3 Lab requirements

Write a program using Java/Python sockets that:

- Is invoked from the command line (STDIN);
- Sends a query to the server for the given domain name using a UDP socket;
- Waits for the response to be returned from the server;
- Interprets the response and outputs the result to terminal display (STDOUT).

Your DNS client should be capable of performing the following actions:

- Send queries for A (IP addresses), MX (mail server), and NS (name server) records;
- Interpret responses that contain A records (IP addresses) and CNAME records (DNS aliases);
- Retransmit queries that are lost;

Your client must also handle errors gracefully. In particular, if the response message does not conform to the DNS specification, if it contains fields or entries that cannot be interpreted, or if the client receives a response that does not match the query it sent, then an appropriate error message should be printed to the screen indicating what was unexpected or what went wrong.

For this lab you must use Java or Python. You must write the code that constructs a DNS request packet and interprets the DNS response packet. You *may not* use external DNS libraries or classes in your project (such as `net.URL` or `net.InetAddress`). Specifically, in Java you may only use the `net.InetAddress.getByAddress(byte[] addr)` method; you may not use any of the `net.InetAddress` methods that take an argument of the form `String host`. (The same requirements apply if you use `net.Inet4Address` instead of `net.InetAddress`.) Similar instructions apply for a Python implementation; please ask me or the TAs if you are not sure about whether you can make use of a library or class.

3.1 Calling syntax

Your application should be named `DnsClient`, and it should be invoked at the command line using the following syntax (for Java; the options should be the same for Python):

```
java DnsClient [-t timeout] [-r max-retries] [-p port] [-mx|-ns] @server name
```

where the arguments are defined as follows.

- `timeout` (optional) gives how long to wait, in seconds, before retransmitting an unanswered query. Default value: 5.
- `max-retries` (optional) is the maximum number of times to retransmit an unanswered query before giving up. Default value: 3.
- `port` (optional) is the UDP port number of the DNS server. Default value: 53.
- `-mx` or `-ns` flags (optional) indicate whether to send a MX (mail server) or NS (name server) query. At most one of these can be given, and if neither is given then the client should send a type A (IP address) query.
- `server` (required) is the IPv4 address of the DNS server, in a.b.c.d. format
- `name` (required) is the domain name to query for.

For example to query for www.mcgill.ca IP address using the McGill DNS server, enter

```
java DnsClient @132.206.85.18 www.mcgill.ca
```

or to query for the mcgill.ca mail server using Google's public DNS server with a timeout of 10 seconds and at most 2 retries, enter

```
java DnsClient –t 10 –r 2 –mx @8.8.8.8 mcgill.ca
```

3.2 Output behavior

Your program should print output to STDOUT using the following standard format. The first lines should summarize the query that has been sent:

```
DnsClient sending request for [name]
Server: [server IP address]
Request type: [A | MX | NS]
```

Where the fields in square brackets are replaced by appropriate values. Then, subsequent lines should summarize the performance and content of the response. When a valid response is received, the first line should read:

```
Response received after [time] seconds ([num-retries] retries)
```

If the response contains records in the Answer section then print:

```
***Answer Section ([num-answers] records)***
```

Then, if the response contains A (IP address) records, each should be printed on a line of the form:

```
IP <tab> [ip address] <tab> [seconds can cache] <tab> [auth | nonauth]
```

Where <tab> is replaced by a tab character. Similarly, if it receives CNAME, MX, or NS records, they should be printed on lines of the form:

```
CNAME <tab> [alias] <tab> [seconds can cache] <tab> [auth | nonauth]
MX <tab> [alias] <tab> [pref] <tab> [seconds can cache] <tab> [auth | nonauth]
NS <tab> [alias] <tab> [seconds can cache] <tab> [auth | nonauth]
```

If the response contains records in the Additional section then also print:

```
***Additional Section ([num-additional] records)***
```

along with appropriate lines for each additional record that matches one of the types A, CNAME, MX, or NS. You can ignore any records in the Authority section for this lab.

If no record is found then a line should simply be printed saying

```
NOTFOUND
```

If any errors occur during execution then lines should be printed saying

```
ERROR <tab> [description of error]
```

Be specific with your error messages. Some example of errors your DNS client may output are:

```
ERROR <tab> Incorrect input syntax: [description of specific problem]
ERROR <tab> Maximum number of retries [max-retries] exceeded
```

```
ERROR <tab> Unexpected response [description of unexpected response content]
```

4 Important dates, deliverables, and evaluation

4.1 Demo

The demo will take place in class on Sept 26th or 27th and will count for 3% of your final grade in the course. For the demo, you should be able to execute your DnsClient for different queries
You will also be asked questions about the DNS protocol specification and how you implemented it (e.g., referring to specific portions of the code), as well as what tests you performed to ensure your application behaves as required.

4.2 Report

The report is due at 23:59 on Sept. 27th, and it will count for 3% of your final grade. Your report should include the following items:

- The names and McGill ID numbers of both group members
- Design: Describe the design of your application. Include a requirements analysis and a diagram depicting software structure.
- Testing: Describe how you tested your application to make sure it behaves as required
- Experiment:
    - 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?
    - Use your client to run DNS queries for 5 different website addresses, of your 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). (*Note:* You probably need to be on campus to get responses from McGill DNS servers.)
    - Do the responses from both servers match? If not, how do they differ? Speculate on why they might differ, and propose additional experiments you could conduct to further explore your hypothesis. (You may receive bonus marks if you conduct these additional experiments and report on the outcome, indicating whether the results validated or refuted your hypothesis.)

To submit your report, upload a PDF file on myCourses. Only one report needs to be uploaded per group. The report should be brief – approximately 4 pages. More detailed instructions will be provided next week.

4.3 Code

When submitting your report on myCourses, you should also upload a zip file containing your code. Make sure to include all source files required to compile and execute your program. Also include a readme file giving any special instructions required to compile your code and mentioning what version of Java or Python you used when writing/testing your program.

5 Additional advice

Java's support for unsigned integers is somewhat limited compared to other languages. For example, a `byte` is an 8-bit signed integer (taking values in the range -128 to 127), whereas a `char` is a 16-bit (unsigned) Unicode character. See the Java Primitive Data Types tutorial for more.

Rather than using character input/output streams, consider using the class java.nio.ByteBuffer when preparing a packet to send. This will give you some control, byte-by-byte, of the data in the packet.