

## Programming Assignment 2 — Simple DNS Client

*Assigned: Sep. 28**Due: Oct. 7, 11:59PM (Phase 0) and Oct. 17, 11:59PM (Phase 1)*

## 1 Introduction

This assignment has two primary purposes:

- Teach you communication using UDP
- Teach you how to use `select(2)` for IO multiplexing

You will write a simple, partially functioning DNS client. DNS is the Domain Name System, and is the system used to (among other things) map hostnames (`bowman.cs.umd.edu`) to IP addresses (`128.8.128.147`). DNS is used to maintain many other types of mappings as well, but we will only query DNS for A (address) records that map hostnames to IP addresses.

The host to IP addresses are kept at DNS servers. The input to your code will be a set of pairs of the following form: (IP address of DNS server, query), e.g.:

```
(128.8.128.7, poole.cs.umd.edu)
(128.8.128.13, bowman.cs.umd.edu)
```

Here, we're asking `128.8.128.7` to give us the IP address for `poole`, and `128.8.128.13` for `bowman`. The server and the queries need not be restricted to the `cs.umd.edu` domain and your code will not need any change for querying any DNS server on the Internet. Even though DNS servers respond to TCP queries, we will rely EXCLUSIVELY on UDP for this assignment.

In class, I will give you a handout that describes the DNS protocol (and all the fields etc. in detail). Please read this, but this is NOT necessary for the solution. If you have any questions about any field, ask NOW.

You will develop your code in two phases.

### 1.1 Phase 0

Your code accepts only one (DNS server, target) pair, and sends this single QUESTION to the server, waits for the response, and parses and prints the answer.

You will need to be able to create a query packet, and then parse the RRs in the answer. The server may return “compressed” data, and you should be able to handle this.

The phase 0 solution will be due on 7th of October, 2007.

#### 1.1.1 Executable

The command line syntax and output for the phase 0 executable is as follows:

```
./dns_client < query.txt
```

where ./dns\_client is your executable file, query.txt is a plain text file containing a set of pairs of the form: (IP address of DNS server, query)

When a response is received, print it in the following form:

Header:

```
ID=<hex number>, flags=<hex number>, QDCOUNT=<dec number>, ANCOUNT=<dec number>,
NSCOUNT=<dec number>, ARCOUNT=<dec number>
```

Question:

```
QNAME=<double-quoted string>, QTYPE=<dec number>, QCLASS=<dec number>
```

Answer:

```
<resource record>
```

Authority:

```
<resource record>
```

```
<resource record>
```

```
...
```

Additional:

```
<resource record>
```

```
<resource record>
```

```
...
```

<dec number> bytes received

Format of <resource record>:

```
NAME=<double-quoted string>, TYPE=<dec number>, CLASS=<dec number>,
```

```
TTL=<dec number>, RDLENGTH=<dec number>,
```

```
RDATA=<IP address in dot format OR double-quoted string for hostname>
```

### 1.1.2 Sample output

Header:

```
ID=0x0, flags=0x8480, QDCOUNT=1, ANCOUNT=1, NSCOUNT=3, ARCOUNT=3
```

Question:

```
QNAME="poole.cs.umd.edu", QTYPE=1, QCLASS=1
```

Answer:

```
NAME="poole.cs.umd.edu", TYPE=1, CLASS=1, TTL=3600, RDLENGTH=4, RDATA=128.8.126.63
```

Authority:

```
NAME="cs.umd.edu", TYPE=2, CLASS=1, TTL=72000, RDLENGTH=7, RDATA="dns2.cs.umd.edu"
```

```
NAME="cs.umd.edu", TYPE=2, CLASS=1, TTL=72000, RDLENGTH=7, RDATA="bozo.cs.umd.edu"
```

```
NAME="cs.umd.edu", TYPE=2, CLASS=1, TTL=72000, RDLENGTH=7, RDATA="dns1.cs.umd.edu"
```

Additional:

```
NAME="bozo.cs.umd.edu", TYPE=1, CLASS=1, TTL=3600, RDLENGTH=4, RDATA=128.8.128.38
```

```
NAME="dns1.cs.umd.edu", TYPE=1, CLASS=1, TTL=3600, RDLENGTH=4, RDATA=128.8.128.7
```

```
NAME="dns2.cs.umd.edu", TYPE=1, CLASS=1, TTL=3600, RDLENGTH=4, RDATA=128.8.128.13
```

155 bytes received

## 1.2 Phase 1

Once you can talk to a server and get responses, you will send multiple queries out to a set of servers. In this assignment, please DO NOT use threads. Instead, we will use select(2) and timeouts to

multiplex multiple socket connections.

For multiple outstanding queries, you will use the ID field in the header to figure out which response corresponds to which query. This is especially useful for multiple outstanding queries to the same server.

Your code should detect when a query has not been answered (using a fixed timeout) and include retransmissions (with a fixed number of tries) before printing an error.

The phase 1 solution will be due on 17th of October, 2007.

### 1.2.1 Executable

The command line syntax and output for the phase 1 executable is as follows:

```
./dns_client [-t <timeout in second>] [-r <max num of retries>]
```

For example, a call at the prompt

```
$ ./dns_client -t 2.5 -r 3 < query.txt
```

means that the timeout is 2.5 seconds and maximum number of retries is 3.

The default values of the timeout and maximum number of retries are 5.0 and 3 respectively. The maximum number of queries in query.txt is 10. (Hint: use getopt)

When a response is received, print:

```
Query #<query number>: responded <response message>
```

The format of <response message> is the same as the one specified in phase 0. Do NOT print duplicated response. When timeout occurs, print:

```
Query #<query number>: timeout
```

When retransmission is performed, print:

```
Query #<query number>: retransmit
```

### 1.2.2 Sample input (i.e. query.txt)

```
(128.8.128.7, poole.cs.umd.edu)
(128.8.128.13, bowman.cs.umd.edu)
```

### 1.2.3 Sample output (all responded)

```
$ ./dns_client -t 1 -r 2 < query.txt
```

```
Query #0: responded
```

```
Header:
```

```
ID=0x0000, flags=0x8480, QDCOUNT=1, ANCOUNT=1, NSCOUNT=3, ARCOUNT=3
```

```
Question:
```

```
QNAME="poole.cs.umd.edu", QTYPE=1, QCLASS=1
```

```
Answer:
```

```
NAME="poole.cs.umd.edu", TYPE=1, CLASS=1, TTL=3600, RDLENGTH=4, RDATA=128.8.126.63
```

```
Authority:
```

```
NAME="cs.umd.edu", TYPE=2, CLASS=1, TTL=72000, RDLENGTH=7, RDATA="bozo.cs.umd.edu"
```

```
NAME="cs.umd.edu", TYPE=2, CLASS=1, TTL=72000, RDLENGTH=7, RDATA="dns1.cs.umd.edu"
```

```
NAME="cs.umd.edu", TYPE=2, CLASS=1, TTL=72000, RDLENGTH=7, RDATA="dns2.cs.umd.edu"
```

```
Additional:
```

```
NAME="bozo.cs.umd.edu", TYPE=1, CLASS=1, TTL=3600, RDLENGTH=4, RDATA=128.8.128.38
```

```
NAME="dns1.cs.umd.edu", TYPE=1, CLASS=1, TTL=3600, RDLENGTH=4, RDATA=128.8.128.7
NAME="dns2.cs.umd.edu", TYPE=1, CLASS=1, TTL=3600, RDLENGTH=4, RDATA=128.8.128.13
155 bytes received
```

Query #1: responded

Header:

```
ID=0x0001, flags=0x8480, QDCOUNT=1, ANCOUNT=1, NSCOUNT=3, ARCOUNT=3
```

Question:

```
QNAME="bowman.cs.umd.edu", QTYPE=1, QCLASS=1
```

Answer:

```
NAME="bowman.cs.umd.edu", TYPE=1, CLASS=1, TTL=3600, RDLENGTH=4, RDATA=128.8.128.147
```

Authority:

```
NAME="cs.umd.edu", TYPE=2, CLASS=1, TTL=72000, RDLENGTH=7, RDATA="dns2.cs.umd.edu"
```

```
NAME="cs.umd.edu", TYPE=2, CLASS=1, TTL=72000, RDLENGTH=7, RDATA="bozo.cs.umd.edu"
```

```
NAME="cs.umd.edu", TYPE=2, CLASS=1, TTL=72000, RDLENGTH=7, RDATA="dns1.cs.umd.edu"
```

Additional:

```
NAME="bozo.cs.umd.edu", TYPE=1, CLASS=1, TTL=3600, RDLENGTH=4, RDATA=128.8.128.38
```

```
NAME="dns1.cs.umd.edu", TYPE=1, CLASS=1, TTL=3600, RDLENGTH=4, RDATA=128.8.128.7
```

```
NAME="dns2.cs.umd.edu", TYPE=1, CLASS=1, TTL=3600, RDLENGTH=4, RDATA=128.8.128.13
```

156 bytes received

#### 1.2.4 Sample output (all timeout)

```
$ ./dns_client -t 1 -r 2 < query.txt
```

```
Query #0: timeout
```

```
Query #0: retransmit
```

```
Query #1: timeout
```

```
Query #1: retransmit
```

```
Query #0: timeout
```

```
Query #0: retransmit
```

```
Query #1: timeout
```

```
Query #1: retransmit
```

```
Query #0: timeout
```

```
Query #1: timeout
```

## 2 Reference

RFC 1035: <http://www.faqs.org/rfcs/rfc1035.html>

Read Section 4. Obviously, you're encouraged to read the entire RFC, but you only NEED to read Section 4.

## 3 Bonus

Resolve arbitrary names without using recursion. Get the list of root server names, and "walk" the DNS tree as necessary to resolve any name. Of course, you're welcome to include other types of queries (MX, HINFO, etc.) once you've mastered the basics.