Homework #2: Simple DNS Client*

1 Objectives

This homework will give you experience with best-effort network communication using UDP datagrams, and the Domain Name System (DNS), a system that allows us to look up a machine's Internet Protocol (IP) address (i.e., 130.127.69.75) based on it's domain name (i.e., www.clemson.edu). In this homework, you will implement a simple client program that sends DNS requests to other machines to resolve Internet addresses.

Your DNS client will take a domain name (the query) and a DNS server as input, and returns the results from the DNS server to the user. Specifically, each time your code is run it should 1) create a well-formed DNS query packet, 2) send the packet in a UDP datagram to the server, 3) wait for the server's response, and 4) print the server's response to the console (stdout).

Your client should report errors for malformed DNS messages, and ignore any other packets that are unrelated to your client's query. Your client should not crash when corrupted packets are received.

2 Client details

Your homework should be written in either C or C++. You should not use any DNS libraries or system calls that send, construct, or interpret DNS messages (i.e., **getaddrinfo()**, **gethostbyname()**, etc.). All DNS messages should be built from scratch, and all interpretation code should be your own.

Your client should be run from the command-line, with the syntax shown in Listing 1. It's output should take one of the four (4) forms shown in Listing 2. If an response to a query contains multiple answers (i.e., multiple IP addresses or aliases), your client should print an **IP**

*This homework is based on one developed by Alan Mislove for CS4700 at Northeastern University

Listing 1: Syntax for your DNS client.

dnsq [**-t** <**time**>] [**-r** <**retries**>] [**-p** <**port**>] **@**<**svr**> <**name**>

timeout (Optional): Indicates in seconds, how long to wait before regenerating an unanswered query. Default value: 5.

retries (Optional): The number of times the client will re-generate the query, before it quits with an error, if no response is received from the server. Default value: 3.

port (Optional): The DNS server's UDP port. Default value: 53.

server (Required): The IP address of the DNS server (e.g., 192.168.0.12).

name (Required): The name we're looking for.

Listing 2: DNS client output format.

```
IP <tab> <IP address> <tab> <secs can be cached> <tab>
    <auth|nonauth>
CNAME <tab> <alias> <tab> <secs can be cached> <tab>
    <auth|nonauth>
NOTFOUND
ERROR <tab> <description of the error>
```

or **CNAME** line for each one of these. If the requested name does not exist, your client must print a **NOTFOUND** line. If an error occurs, your client should print an **ERROR** line along with a description of the error.

The DNS overview in the following sections will be indispensable throughout the homework. Remember to convert integers, shorts, and longs to network byte order (using hton() and associated functions). You might want to compare the output of your program to other DNS utilities. The **dig** program, on Linux/OS X/UNIX, for example outputs a fair amount of debug information

Your program should compile and run on the lab machines.

bit 0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
ID															
QR	OpCode				AA	TC	RD	RA	Z			RCODE			
QDCOUNT															
ANCOUNT															
NSCOUNT															
ARCOUNT															

Table 1: DNS Header format.

3 The Domain Naming System

There is a lot of DNS-related documentation on the web. We have added some of it to this document, to save you time, and improve clarity. The official reference for the DNS protocol is found in the requests for comment (RFCs) that include DNS (most importantly RFC 1035). Most of this information comes from the RFC.

DNS is a hierarchical protocol built on UDP. A domain (e.g., clemson.edu, whitehouse.gov, etc) is served by a set of DNS servers (1 or more). If you want the IP address for a subdomain of *clemson.edu*, like *www.clemson.edu* or *www.cs.clemson.edu*, you should send a request to the servers that manage the *clemson.edu* domain. In the interest of performance, intermediate servers can cache replies from other servers. Any replies that come from a responsible DNS server are considered *authoritative*. Replies coming from other servers are *non-authoritative*.

3.1 Packet Structure

DNS packets have the following sections: <HEADER> <QUESTION> <ANSWER> <AUTHORITY> <ADDITIONAL>. In this homework we will not use the last two (Authority and Additional). Your client should accept packets with these fields, but you should ignore them.

Note: a single question may receive a response containing multiple answers (if, for example, an address has multiple IP addresses, or if an address has a CNAME and an A record). Your client must process the entire answer section and report on each one of these records.

HEADER

DNS queries and responses use the same header format, which is shown in Table 1.

ID is a 16-bit identifier is chosen by the client. The server copies this ID into any response that results from a query, so that the client can match responses to queries. *it would be wise to select a new, random ID for each request.*

QR is a one-bit flag that specifies whether the message is a query (0) or a response (1).

OpCode is a 4-bit field that specifies the type of query. *A value of zero (0), represents a standard query (which is all your client will send).*

AA (Authorative Answer) this bit is only meaningful in responses. If it's a one (1) it means that the name server is an authority for the domain name being requested. *This will tell your client whether or not a response is authoritative.*

TC (TrunCation) If this bit is one (1), then the message was truncated. *Your client should exit with an appropriate error message if it receives a truncated message.*

RD (Recursion Desired) This bit tells the name server that it wants a recursive query. *Your client should always request recursion (RD=1).*

RA (Recursion Available) In a response, this bit is set (1) or cleared (0) by the server in order to tell you whether or not the server supports recursion. *You should exit and report an appropriate error message, if you receive a response from a server that doesn't support recursion.*

Z is reserved for future use. *set this field to zero (0).*

RCODE (Response Code) In a response packet, a 4-bit value is interpreted as follows:

- 0 – No error
- 1 – Format error — the server couldn't understand the query
- 2 – Server failure — your packet was fine, but the server had a problem
- 3 – Name error — only meaningful from an authoritative name server. This code means that the requested domain name does not exist.

bit 0 1 2 3 4 5 6 7 8 9 A B C D E F
... QNAME ...
QTYPE
QCLASS

Table 2: DNS Question format.

- 4 – Not Implemented — the server doesn't support that kind of query
- 5 – Refused — The server doesn't like you (actually, it has refused to perform the operation for policy reasons).

QDCOUNT an unsigned 16 bit integer, which specifies the number of entries in the question section. *Your client will only ever send one. So set it to 1.*

ANCOUNT another unsigned 16-bit integer. How many resource records are in the answer section. *For queries, this should be zero, since you aren't providing answers.*

NSCOUNT yet another unsigned 16-bit integer. The number of server resource records in the authority records section. *Set this field to zero (0) and ignore any entries in this section.*

ARCOUNT Unsigned 16-bit int. Number of resource records in the additional records section. *Set to zero, and ignore response entries in this section.*

QUESTIONS

DNS questions have the following format, shown in Table 2.

QNAME The domain name, which is represented as a sequence of labels. Each label consists of a length octet (byte) followed by that number of octets (bytes). The domain name should terminate with a zero length octet (which is the null label of the "root"). See the example, listed below, for more details.

QTYPE A 2-byte code describing the query type. *Your homework only needs to make host address, so you should use the value 0x0001 for this field.*

bit 0 1 2 3 4 5 6 7 8 9 A B C D E F
... NAME ...
TYPE
CLASS
TTL
RDLENGTH
... RDATA ...

Table 3: DNS Answer format.

QCLASS A 2-byte code, describing the query class. *Use 0x0001 for Internet Addresses.*

ANSWERS

DNS answers have the following format, shown in Table 3.

NAME The domain name that was requested. The format is the same as QNAME in the questions.

TYPE A 2-byte code describing the type of the answer that is stored in RDATA, . *Your client needs to be able to interpret A Records (type 0x001) and CNAME records (type 0x0005).*

CLASS A 2-byte code, describing the query class. *Use 0x0001 for Internet Addresses*

TTL A 32-bit unsigned int, representing the number of seconds that the results can (should) be cached.

RDLENGTH 16-bits unsigned. Length of the RDATA field.

RDATA Response data. The format depends on the TYPE field. If the TYPE is 0x0001 for A records, then the format is a 4-byte IP address. If the type is 0x0005 (CNAME), then this is the name of the alias.

0000	19	ca	01	00	00	01	00	00	....
0008	00	00	00	00	03	77	77	77	.....www
0010	04	67	72	61	64	07	63	6c	.grad.cl
0018	65	6d	73	6f	6e	03	65	64	emson.ed
0020	75	00	00	01	00	01			u.....

Table 4: Example DNS query.

3.2 DNS Compression

DNS uses compression to keep packet sizes down. The goal is to eliminate the repetition of domain names in the NAME, QNAME, and RDATA fields. This is accomplished by replacing a domain name or a list of labels at the end of a domain name with a pointer to a prior occurrence of the same name.

A pointer is a 16-bit sequence. The first two bits are both ones (to allow a pointer to be distinguished from a label — labels must be 63 bytes or less). The last 14-bits are the OFFSET, which specifies an offset from the beginning of the message (i.e., the first byte of the ID field in the header). This technique allows a domain name to be represented as either, (1) a sequence of labels ending in a zero byte, (2) a pointer, or (3) a sequence of labels ending in a pointer.

If a domain name is contained in a part of the message subject to a length field (such as the RDATA section of a response), and compression is used, the length of the compressed name is used in the length calculation, rather than the length of the expanded name.

Your client doesn't need to use compression in the messages it generates, however, it must understand arriving messages that use compression.

3.3 Example DNS Query

Table 4 shows an example query, which should be interpreted as follows:

- ID = 0x19ca
- Message is a query (QR = 0)
- OpCode = 0 (standard query)
- AA = 0

- Not truncated (TC = 0)
- Recursion desired (RD = 1)
- Z = 0 (as it should be)
- 1 Question
- 0 Answers
- Questions (QNAME = (3) www (4) grad (7) clemson (3) edu; QTYPE = 0x001; QCLASS = 0x0001)

3.4 Example DNS Response

Table 5 shows an example response, which should be interpreted as follows:

- ID = 0x19ca (looks familiar!)
- Message is a response (QR = 1)
- OpCode = 0 (standard query)
- AA = 0 (server is not an authority)
- Not truncated (TC = 0)
- Recursion desired (RD = 1)
- Recursion available
- RCODE = 0x0000 (no error)
- 1 Question
- 2 Answers
- Questions (QNAME = (3) www (4) grad (7) clemson (3) edu; QTYPE = 0x001; QCLASS = 0x0001)
- Answer #1 — Name = www.grad.clemson.edu (pointer!); Type= CNAME (alias); Class = 0x0001; TTL= 44 minutes, 39 seconds; RDLENGTH = 13; RDATA = (10) allvirtual (pointer to clemson.edu)

0000	19	ca	81	80	00	01	00	02	....
0008	00	00	00	00	03	77	77	77	.....www
0010	04	67	72	61	64	07	63	6c	.grad.cl
0018	65	6d	73	6f	6e	03	65	64	emson.ed
0020	75	00	00	01	00	01	c0	0c	u.......
0028	00	05	00	01	00	00	0a	77	.......w
0030	00	0d	0a	61	6c	6c	76	69	...allvi
0038	72	74	75	61	6c	c0	15	c0	rtual...
0040	32	00	01	00	01	00	00	00	2.......
0048	73	00	04	82	7f	eb	31		s.....l

Table 5: Example DNS response.

- Answer #2 — Name = all virtual.clemson.edu (pointer!); Type: A (host address); Class = 0x0001; TTL = 1 minute, 55 seconds; RDLENGTH = 4; RDATA = 130.127.235.49

Note: you can use Wireshark to capture as many example queries and responses as you want. It will also parse your client's queries for you, which can be incredibly helpful, when it comes to debugging.

4 More Hints

You may be wondering where to find a real DNS server to test against. If you're on OS X, Linux, or any other UNIX-based OS, then "/etc/resolve.conf" should have your current DNS servers IP addresses. Just run "cat /etc/resolve.conf" and you should see a list of name servers at the bottom.

Also, DNS typically uses port 53.

5 Submission Instructions

The suggested completion time for this homework is 2 weeks.

When your homework is complete, archive your source materials, using the following command:

```
> tar cvzf youruserid-hw2.tar.gz README Makefile <list of source files>
```

The **Makefile** should build your program (by running **make**).

The **README** file should include your name, a short description of your homework, and any other comments you think are relevant to the grading. It should have five clearly labeled sections titled with KNOWN PROBLEMS, and DESIGN. The KNOWN PROBLEMS section should include a specific description of all known problems or program limitations. This will not hurt your grade, and may improve it.

The DESIGN section should include a short description of how you designed your layers (especially anything you thought was interesting or clever about your solution). You should also include references to any materials that you referred to while working on the homework. Please do not include special instructions for compilation. The compilation and running of the tests will be automated (see details below) and your instructions will not be followed.

Please make sure you include only those source files, not the object files. Before you submit this single file, please use the following command to check and make sure you have everything in the `youruserid-hw2.tar.gz` file, using the following script:

```
> tar tvzf youruserid-hw2.tar.gz
```

This command should put all of your source files in the current directory. It should not create any subdirectories. Running `make` after extracting the files should build your `dnsq` program, which should end up in the current directory.

Submit your `youruserid-hw2.tar.gz` file via **handin.cs.clemson.edu**. You must name your archive **youruserid-hw2.tar.gz**, and it must compile and run. We will compile your code using your **Makefile**, and run it using the syntax described in Listing 1.

6 Grading

Your homework will be graded based on the results of functional testing and the design of your code. We will run several tests to make sure it works properly and correctly handles various error conditions. We will test your client against real DNS servers. We may also test your client against our own custom DNS server that sometimes misbehaves (spitting out corrupted packets, for example). Your code should output a reasonable error message when it encounters a corrupted packet. It should not crash. Specifically, you will receive 10% credit if your code successfully compiles, and 10% for code style and readability. The rest of your score will be determined by the number of tests that your client

passes.

Your source materials should be readable, properly documented and use good coding practices (avoid magic numbers, use appropriately named variables, etc). You code should be -Wall clean (you should not have any warnings during compilation). **Our testing of your code will be thorough. Be sure you test your application thoroughly.**

7 Collaboration

You will work independently on this homework. You must *not* discuss the problem or the solution with classmates, and all of your code must be your own code.

You may, of course, discuss the homework with our grader, and you may discuss conceptual issues related to the homework that we have already discussed in lecture (do not post any code or implementation details of the algorithms). **Collaborating with peers (or strangers on the Internet) on this homework, in any other manner will be treated as academic misconduct.**