OVERVIEW

This problem set has two questions, each with several parts. Answer them as clearly and concisely as possible. You may discuss ideas with others in the class, but your solutions must be your own. Do not look at anyone else’s solutions or copy them from anywhere. (Please refer to the Duke University honor code). Turn in your solutions in on February 1, 2013 in class.

P1: DNS

In the first part of this question, you will perform some hands-on DNS queries using dig and play with DNS lookups from various applications to understand more about the DNS. In the second part of this question, you will implement a variation on a stub DNS resolver. RFC 1035 may be helpful for answering some of these questions.

1. In this question, we warm up by learning a few things about Duke University DNS setup.

(a) What are the authoritative nameservers for duke.edu? How long will your resolver cache the records pointing to these nameservers? What are the Computer Science Department authoritative nameservers (i.e., for the domain cs.duke.edu)? Give two benefits of topologically diverse authoritative nameservers. Why do NS records return names, rather than IP addresses?
(b) What is another “canonical name” for the Duke University Web server?

(c) What is the primary mail exchanger for cs.duke.edu?

2. Now that you have had some experience playing with `dig`, in this part of the problem, we will implement a stub resolver that performs iterative DNS queries. Most of the time, stub resolvers send queries with the RD (Recursion Desired) bit turned on. In this problem, you are not allowed to use the recursion bit. Of course, you are welcome to solve this problem any way you like (C, C++, JAVA, Phyton, Perl, shell script, etc). If you prefer, you may use the Ruby skeleton code provided at [http://www.cs.duke.edu/courses/spring08/cps214/hw/ps1/dns-resolv-rb.tgz](http://www.cs.duke.edu/courses/spring08/cps214/hw/ps1/dns-resolv-rb.tgz). This may save you the trouble of figuring out which modules to use, instrumenting your own performance measurements, etc.

(a) Why do stub resolvers typically set the RD bit?

(b) Implement a stub resolver that performs only iterative queries to resolve A records. To answer the next question, you will want to make it possible to provide an option to your program to control the root nameserver. Your resolver need not do anything special as far as caching, etc., but you should handle timeouts (e.g., querying the next preferred authoritative nameserver if the first does not respond). Just make sure you can (1) point it at different root nameservers and (2) measure the time taken to resolve a query (the skeleton code is instrumented for this).

(c) Use your query to resolve (1) www.cs.duke.edu and (2) www.nytimes.com at the following nameservers.

- a.root-servers.net (198.41.0.4)
- f.root-servers.net (192.5.5.241)
- m.root-servers.net (202.12.27.33)
- a.gtld-servers.net (192.5.6.30)

  i. Through what sequence of nameservers was each query referred? How long did each referral step take? Based on this, what fraction of DNS query time is saved by caching at local resolvers?

  ii. What is the first referral when you send a query `www.cs.duke.edu` to `a.gtld-servers.net`? Is the answer the same everytime? Why or why not?

  iii. How do stub resolvers typically choose root nameservers?

Please hand in your code to this problem as well. The code should be commented. Also, include a README file with instructions for compiling/executing your code.

**P2: CDN UNDER THE HOOD**

In this problem, you need to observe the existence of CDN through `traceroute` and reverse DNS lookup. Two types of tools are needed:
(1) A tool that can monitor where the packet went through. For example, you can use one of the following: (a) Wireshark or (b) A browser addon that can do traceroute (E.g., On Chrome, one called CoNetServ works) or (c) An online service does similar measurement from multiple locations (E.g., http://tools.websitepulse.com/tools.php)

(2) Reverse DNS lookup – meaning given the IP address, resolve the domain name and related information. (E.g., http://www.itistimed.com/).

Choose a website that uses CDN to deliver its content (E.g., Hulu). Do traceroute or use one of the tools in (1) to find out the path leading to that content and answer the following questions:

(a) How many hops are involved in the path?
(b) What domain does each of these hops belong to? Which domain does the last hop belong to? So who is serving this content?
(c) Where are the nodes on this path physically located (list the states)? How long (in miles) is this path in total?
(d) What is the total latency? Which hop contributes the most to the latency?
(e) Repeat the traceroute a few times. Does the path change? Does the final hop change? Why?