PROGRAMMING ASSIGNMENT 2: DNS Resolver

Course Number: CSE 3101
Course Title: Computer Networking
Submission Deadline: 10 March 2019, 11:59 p.m
Maximum points: 40
START EARLY – THIS IS NOT VERY EASY

Goal: In this assignment your task is to design and implement a simple DNS Resolver, which takes a hostname as input and provides the corresponding IP address (i.e. type A record) as the output. Your DNS resolver should only issue iterative queries and should only support queries for type A records. Note that, the functionality desired of your resolver program is similar to that of the dig (domain information groper) utility available on most Linux machines. You must not make use of DNS library functions (e.g., InetAddress.getbyname() in Java and gethostbyname() in C).

Specification:
You have to write a program called DNS_resolver.c (or DNS_resolver.java), which implements the DNS resolver.

- The program should be executed from a standard command line and should accept one argument, the hostname for which the IP address is desired. Here is the specification for the command line usage:
  DNS_resolver hostname (e.g.: DNS_resolver google.com)
  (or java DNS_resolver hostname for JAVA)
- Your resolver should only support queries for type A records i.e. simple hostname to IP address translations.
- Your program MUST only issue iterative queries to obtain the IP address for the hostname. You CANNOT use recursive queries. This entails that your program MUST also bypass the local DNS server (or the default DNS server) for the domain in which your program is executed (i.e. the CSE local DNS server). Recall that all DNS queries are usually directed to the local DNS server within the domain (see the picture overleap for elaboration on this point).
- Your program should therefore first contact one of the root name servers. Recall that there are 13 root DNS servers. You can find their IP addresses by typing the command "dig . NS" at the command prompt of a CSE Linux machine. Alternately, check Reference 4 at the end of this document - this link provides the IP addresses of the root name servers. You can configure these addresses in your program since they are static.
- The figure on the next page illustrates an example of how your program must operate. Notice that the DNS resolver bypasses the CSE local DNS server and directly sends the query to one of the root DNS servers. In this particular example, the DNS resolver requires 3 iterations (first query to the root server, second query to the TLD server and the final query to the authoritative name server). Note that depending on the hostname the number of iterations could be less or more.
- Once your program has obtained the type A record for the hostname, it should print the IP address to the standard output. For example,
  www.cse.du.ac.bd = 103.221.253.170
- If a hostname does not exist or does not have a type A record, then your resolver should indicate this at the standard output. For example,
  www.nobody.bd = does not exist

- Normally there are several redundant name servers for each DNS zone. If one server is down or has other problems, you should try a different server. Only if none of the servers are responsive can you report a DNS error.
- There are a large number of DNS resource record types. Your program should only support queries for type A records. However, to do this, your resolver will also need to understand the following DNS resource record types:
  - CNAME records: Note that the hostname input to your program could actually be a CNAME (i.e. canonical name). Your program should be able to find the corresponding IP address. Sometimes it may be possible that one CNAME points to another CNAME and so on, thus creating a chain. If such a chain exists, your code must continue to follow the CNAME chain until it either finds an IP address at the end of the chain or process that the CNAME chain is misconfigured (i.e. the chain of CNAMEs leads to a hostname that does not exist).
  - SOA records: SOA records are sent to indicate that a requested name does not exist.
  - NS records: In response to certain queries, your program will receive back NS records, which identify servers that should be queried next (note that the IP address for these servers are usually in the additional information field of the DNS message). Your program should essentially follow this chain of NS records starting at the DNS root server until you find the name server, which contains the Type A address for the hostname. For example, in the figure above, the root server and the TLD server both respond with NS records of the subsequent server along the chain. See RFC's 1034 and 1035 [References 1 & 2] for further details about the operation of DNS, DNS resource record types, and the byte format of DNS messages. The textbook also contains a detailed description of DNS in Section 2.5.
- You must use UDP sockets since DNS typically uses UDP for sending queries. Your program should choose a random socket number (greater than 1023). Recall that DNS servers use port 53 for receiving DNS messages.
- Your program will need to create the DNS query message, encapsulate each message within a UDP segment and then transmit the segment through the datagram socket. Your program must also receive DNS response messages and interpret their content.
- Note that, you may have to set a timer so that if a response is not received from the queried server, then your program can choose an alternate server, if available.
- The goal of this assignment is for you to develop the DNS resolver from scratch. You must create your OWN DNS messages, open your own sockets, send your own query messages and listen for response messages. It is NOT acceptable to call DNS library functions. For example, Java codesmust not use functions like InetAddress.getbyname() and C code must not use functions like gethostbyname(). There will be a significant penalty if you do not adhere to these rules.

Programming Notes

- Don't debug by embedding print statements in your code unless it is absolutely necessary. Instead, learn to use a symbolic debugger like gdb (or jdb for java). You can also find some nice GUI front-ends for gdb that make it easier to use. By using a debugger, you will save yourself tons of time finding the bugs in your code. Symbolic debuggers are used by professional programmers debug code so why not take advantage of this assignment to hone your professional skills. Note: Compile your code with the -g option in gcc so that the compiler will include debugging information in the executable. The javac compiler also has a -g option that does the same thing, but check the documentation.

- Note that the functionality desired in your program is similar to that of the domain information groper (dig) utility, available on Linux machines. Therefore, a good idea might be to compare the results obtained from your program with that obtained by dig.
- In writing your code, make sure to check for an error return from your system calls or method invocations, and display an appropriate message. In C/C++ this means checking and handling error return codes from your system calls. In Java, it means catching and handling IOExceptions.
- Make sure you close every socket that you use in your program. If you abort your program, the socket may still hang around and the next time you try and bind a new socket to the port ID you previously used (but never closed), you may get an error. Also, please be aware that port ID's, when bound to sockets, are system-wide values and thus other students may be using the port number you are trying to use. On Linux systems, you can run the command netstat to see which port numbers are currently assigned.
- Do not worry about the reliability of UDP in your assignment. It is very unlikely that small UDP packets containing DNS messages are lost. If your program appears to be losing or corrupting packets on a regular basis, then it is very likely due to a fault in your program.

Additional Notes

- This is a group assignment. You are expected to work on this with your partner.
- Where to Start: The textbook contains code for a simple UDP client server application, which is a good place to start. (Section 2.8). RFC’s 1034 and 1035 will be useful resources.
- Language and Platform: You are free to use either C, C++ or JAVA to implement this assignment. Please choose a language that you are comfortable with. Your assignment will be tested on the Linux Platform. Make sure you develop your code under Linux.

Assignment Submission

Your program should be called DNS_resolver.c/ DNS_resolver.cpp/ DNS_resolver.java. You may ofcourse have additional header files and/or helper files. If you are using C/C++/Java, then you MUST submit a makefile/script along with your code. In addition you should submit a small report, report.pdf (no more than 3 pages) describing the program design and a brief description of how your system works. Also discuss any design trade offs considered and made. Describe possible improvements and extensions to your program and indicate how you could realize them. If your program does not work under any particular circumstances please report this here. Also indicate any segments of code that you have borrowed from the Web or other books.

- **Late Submission Penalty:** Late penalty will be applied as follows:
  - 1 day after deadline: 10% reduction
  - 2 days after deadline: 20% reduction
  - 3 days after deadline: 30% reduction
  - 4 days after deadline: 40% reduction
  - 5 or more days late: NOT accepted
- **Plagiarism**

You are to write all of the code for this assignment yourself. All source codes are subject to strict checks for plagiarism, via highly sophisticated plagiarism detection software. These checks may include comparison with available code from Internet sites and assignments from previous semesters. In addition, each submission will be checked against all other submissions of the current semester. Please note that we take this matter quite seriously. The LIC will decide on appropriate penalty for detected cases of plagiarism. The most likely penalty would be to reduce the assignment mark to ZERO. We are aware that a lot of learning takes place in student conversations, and don't wish to discourage those. However, it is important, for both those helping others and those being helped, not to provide/accept any programming language code in writing, as this is apt to be used exactly as is, and lead to plagiarism penalties for both the supplier and the copier of the codes. Write something on a piece of paper, by all means, but tear it up/take it away when the discussion is over. It is OK to borrow bits and pieces of code from sample socket code out on the Web and in books. You MUST however acknowledge the source of any borrowed code. This means providing a reference to a book or a URL when the code appears (as comments). Also indicate in your report the portions of your code that were borrowed. Explain any modifications you have made (if any) to the borrowed code.

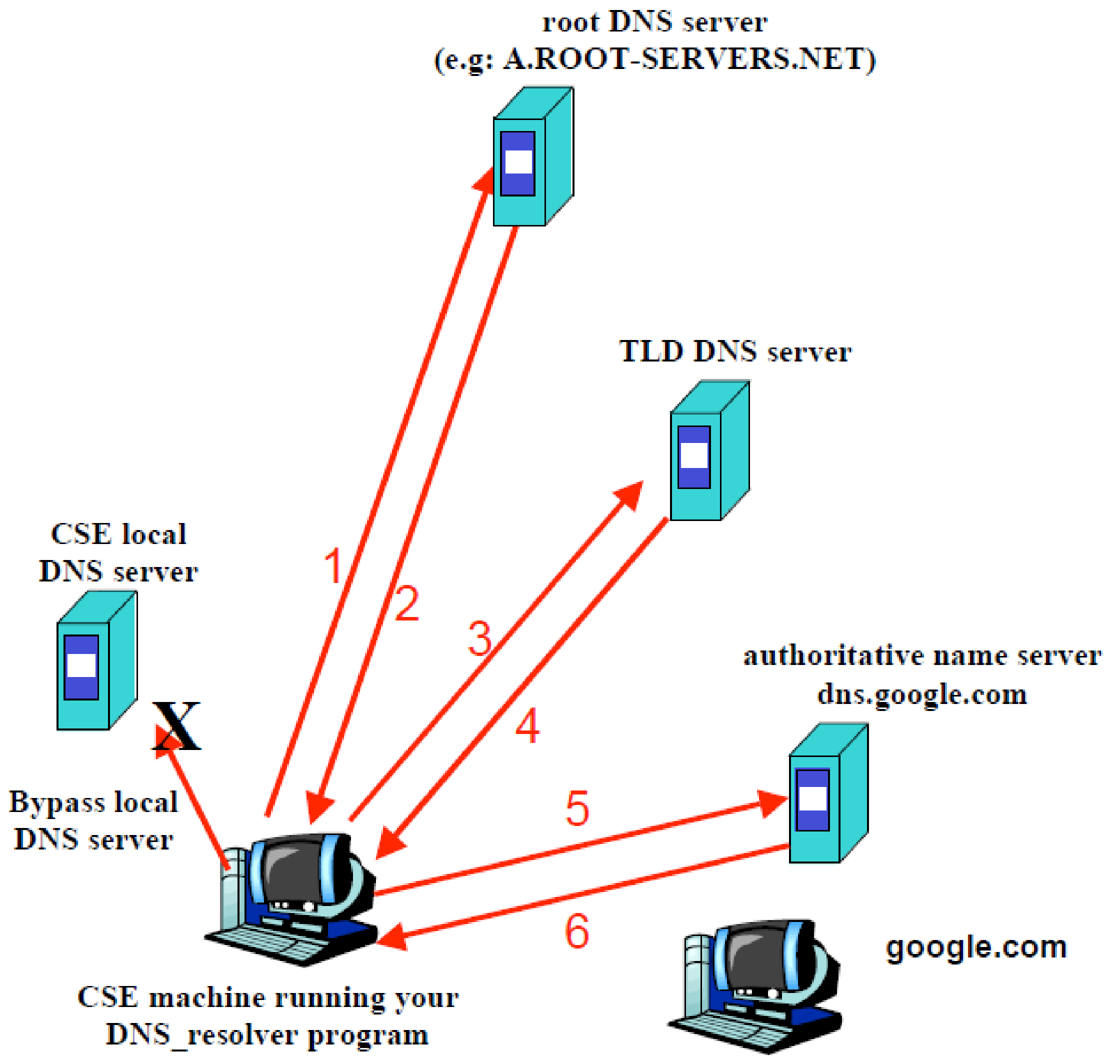
Marking Policy

You should test your program rigorously and verify the results by using dig before submitting your code. Your code will be marked using the following criteria:

- Correct compilation of all files: 1 mark
- Source code design (good structure and well commented): 3 marks
- Successful resolution of Type A addresses: 16 marks
- Successful resolution of CNAME records: 6 marks
- Successful indication of erroneous hostnames: 7 marks
- Report: 4 marks
- Makefile: 3 marks

USEFUL REFERENCES

1) P. Mockapetris, Domain Names – Concepts and Facilities, RFC 1034, IETF, November 1987, http://www.ietf.org/rfc/rfc1034.txt.
2) P. Mockapetris, Domain Names – Implementation and Specifications, RFC 1035, IETF, November 1987, http://www.ietf.org/rfc/rfc1035.txt.
3) J. Kurose and K. Ross, Computer Networking: A Top-Down Approach Featuring the Internet, 4th Edition, 2007. (Refer to Section 2.5).
4) List of Root Name Servers, Wikipedia, http://en.wikipedia.org/wiki/Root_nameserver.
5) L. Parziale, D. Britt, C. Davis, J. Forrester, W. Liu, C. Matthews, N. Rosselot, TCP/IP Tutorial and Technical Overview, IBM Redbooks, December 2006,
http://www.redbooks.ibm.com/redbooks/pdfs/gg243376.pdf (Refer to Chapter 12).
6) The TCP/IP Guide, Free Internet Resource,
http://www.tcpipguide.com/free/t_DNSOverviewFunctionsandCharacteristics.htm.
7) DNS Reverse Name Resolution,
http://www.tcpipguide.com/free/t_DNSReverseNameResolutionUsingtheINADDRARPADo


root DNS server
(e.g: A.ROOT-SERVERS.NET)
TLD DNS server
CSE local
DNS server
X
Bypass local
DNS server
authoritative name server
dns.google.com
1
2
3
4
5
6
google.com
CSE machine running your
DNS_resolver program