Homework 1 Report

Dhanashri Patil
SBU ID 111461286

PART A:

This source code consists of an implementation of iterative DNS resolver in Python. The 13 root servers from https://www.iana.org/domains/root/servers are used as a static list. The dig tool accepts the hostname/FQDN and type as arguments and calls iterative_resolver() function from mydnsresolver.py
The algorithm works as below:

1. Tokenize the given hostname for eg. www.google.com
2. Pass “com” to one of the root name server as a DNS query of type UDP along with the type (A/NS/MX)
3. If we get answer section filled, then stop
4. Else pick up the servers from additional section as name servers for next iteration by appending next token to the partial hostname
5. Iterate over all the name server at every level till we get the IP resolved

Special Cases Handled:

1. If we get some CNAME in answer section, then call the iterative_resolver() function again to resolve that CNAME.
2. In some cases, you may need additional resolution. For example, google.co.jp will often not resolve to an IP address in one pass. Here, I have resolved the NS with SOA type that we get in the Authority section after querying “google.co.jp” for the same name server that we used to query “co.jp”. The ip that we get in answer section of this iteration will be used as name server for resolving “google.co.jp”
3. Some dns query take longer time, so I have also used timeout to avoid waiting forever for a response

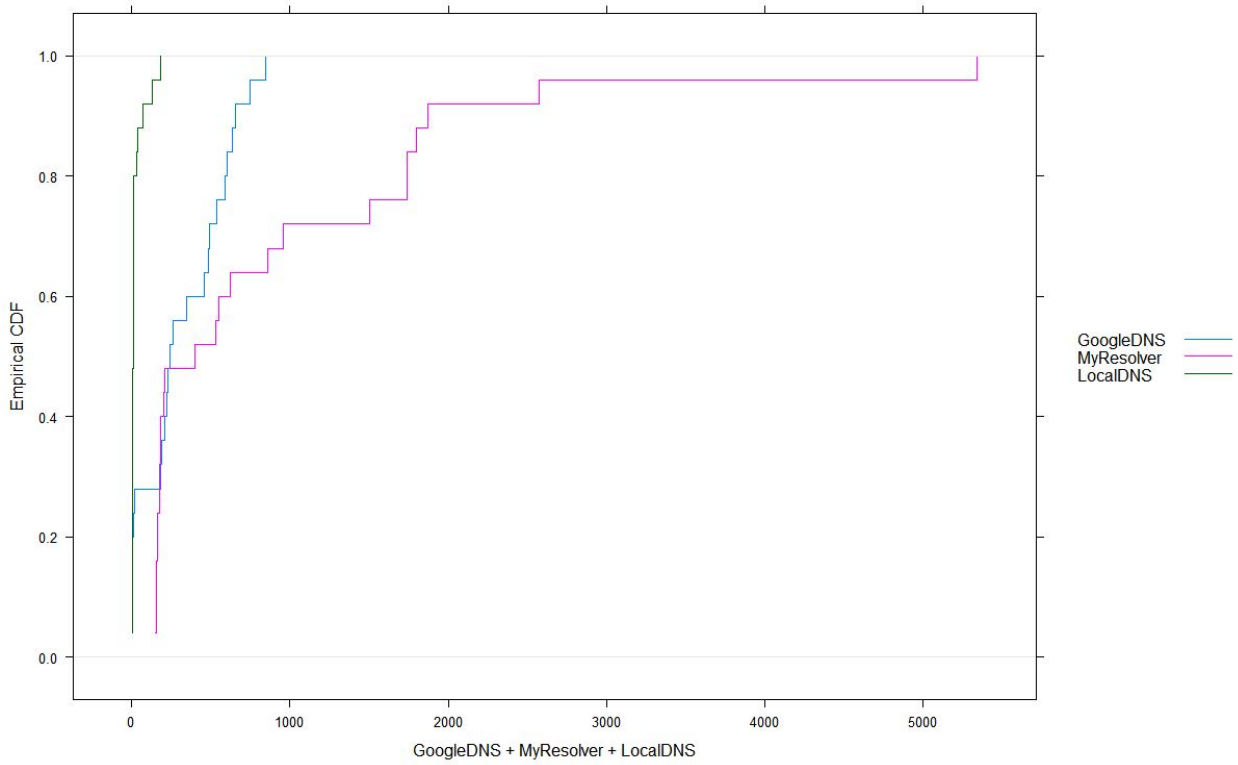
PART B:
DNSSEC Implementation Details:
In DNSSEC, we validate whether the information given by the given name server comes from an authentic source or not. The validate() method incorporates 3 passes of validation for a given pair of name server and host as below:

1. Get the parent zone for given hostname (for example .com is zone for google.com). Make DNS query with type=dns.rdatatype.DNSKEY and flag want_dnssec=True to the parent/zone. Validate DNSKey(public signing key of zone) and RRSIG (cryptographic signature)
2. Make DNS query to the child/zone(given hostname) with type=dns.rdatatype.DNSKEY and flag want_dnssec=True. Based on whether the answer or authority section of response is present extract DS(hash of DNS key record) and RRSIG of DS. Validate these two

3. **Chain of trust validation :** IF DS is not an instance of NSEC and NSEC3, calculate the new DS by using make_ds with SHA256 algorithm. Compare this DS with the previous DS (stored from previous iteration of validate function for its parent zone) Thus at every level a trust between a zone and its parent zone is established

When the given DS is not an instance of NSEC or NSEC3 we stop the validation and further resolution stating “DNSSEC not supported”. In case of failure at any of the above three passes we stop the validation and further resolving task by stating “DNSSEC validation failed” followed by exit() If at every stage all three passes get successfully cleared, we display the output of given query on the console.

PART C:



Note:

The above graph shows output for three resolvers:

1. Local DNS Resolver (Green line)
2. My DNS Resolver (Purple line)
3. Google DNS Resolver (Blue line)

X Axis: CDF function for top 25 websites in terms of their probability of time taken

Y Axis: Time taken to resolve the IP in milli seconds (Average of 10 runs for for getting answer for the query to resolve the website)

Top 25 Websites taken from http://www.alexa.com/topsites :

["Google.com", "Youtube.com", "Facebook.com", "Baidu.com", "Wikipedia.org", "Reddit.com", "Yahoo.com", "Google.co.in", "Qq.com", "Taobao.com", "Amazon.com", "Tmall.com", "Twitter.com", "Google.co.jp", "Live.com", "Instagram.com", "Vk.com", "Sohu.com", "Jd.com", "Weibo.com", "360.cn", "Google.de", "Google.co.uk", "Google.com.br"]

Refernces:

1. https://www.cloudflare.com/dns/dnssec/how-dnssec-works/
2. DNSPython documentation