Final Project Report

Introduction

This project developed a DNS resolver capable of handling both recursive and iterative queries with an integrated cache mechanism. By using Google's DNS server (8.8.4.4) provided by https://public-dns.info/, the project aimed to achieve accurate and efficient DNS resolution. The final implementation successfully meets these original goals, providing a reliable DNS resolver that efficiently handles different types of DNS queries and supports caching mechanism.
Demo Video Link

Implementation

The DNS resolver consists of the following key components:

Iterative Query Handler: This function is responsible for handling iterative queries. It starts from the given DNS server (Google's DNS server 8.8.4.4) and iteratively queries down the DNS hierarchy until it finds the final answer for a given domain name.

Recursive Query Handler: This function resolves the recursive DNS query path internally and returns the final result to the user by calling dns.resolver in dnspython.

Cache Mechanism: To enhance efficiency, a caching system is implemented. This cache stores recently queried domain names along with their resolved IP addresses and other relevant DNS records, significantly reducing resolution time for frequently accessed domain names. Cache mechanism is only integrated within the iterative query handler.

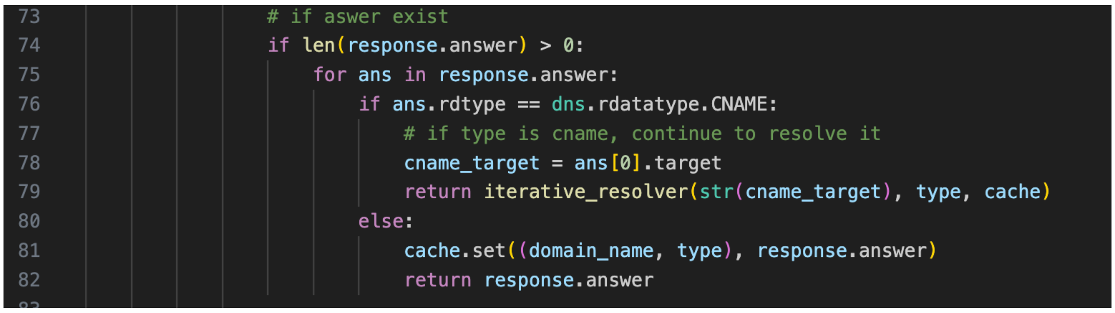
The implementation is mainly developed by using the 'dnspython' library, allowing for detailed handling of DNS protocols and records, as well as 'time' library for cache mechanism realization. The resolver supports various record types including A, AAAA, CNAME, MX, and NS. The cache mechanism uses a simple time-to-live (TTL) approach, where cached entries expire after a predefined duration.

Result and Discussion

Throughout the project, several tests were conducted to ensure the accuracy and efficiency of the resolver. These tests included resolving domain names with different record types, handling nonexistent domains, and testing the cache mechanism's performance.

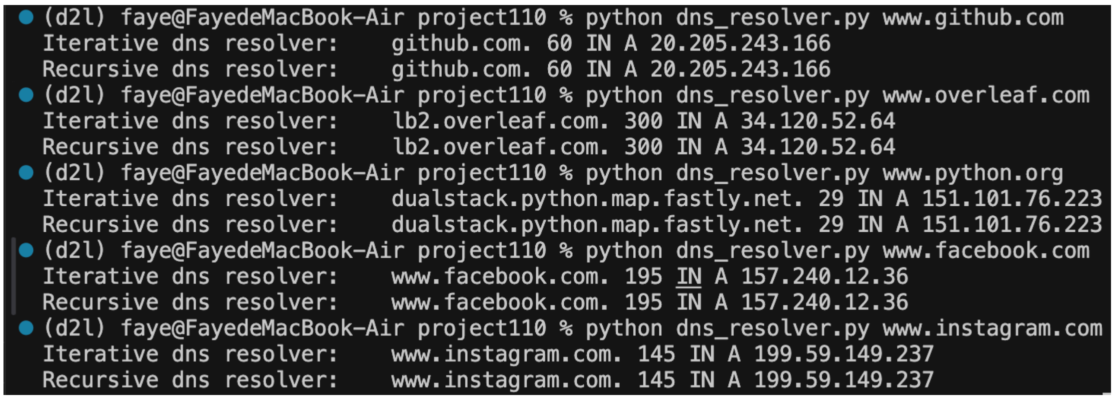
Challenges:

Handling of CNAME Records: One challenge was ensuring proper handling of CNAME records, which required additional logic to follow the chain of records to the final IP address. In my code, it's realized in the iterative query resolver as below:

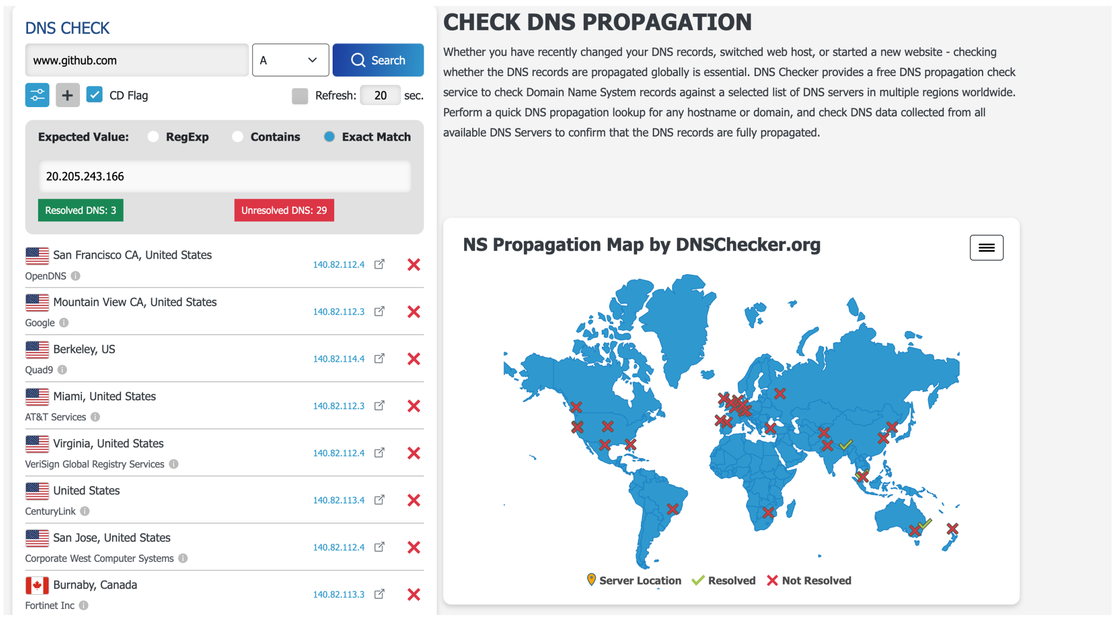


```python
                # if aswer exist
                if len(response.answer) > 0:
                    for ans in response.answer:
                        if ans.rdtype == dns.rdatatype.CNAME:
                            # if type is cname, continue to resolve it
                            cname_target = ans[0].target
                            return iterative_resolver(str(cname_target), type, cache)
                        else:
                            cache.set((domain_name, type), response.answer)
                            return response.answer
```

Variability in DNS Responses: Testing revealed that DNS responses can vary based on geographic location, network settings, and the specific DNS server used. To test the output, I use the website "dnscheker.org". Screenshots from the program's output are listed below. Some of the domains, such as "www.github.com" and "www.overleaf.com", were verified on the website "dnschecker.org" after running my own DNS resolver. It can be observed that the IP addresses returned by the code match those obtained on the website (since a single domain can correspond to multiple IP addresses, the presence of a green tick on the right side of the map on the site indicates that there are matching IP addresses, confirming the correctness of the code's output). However, the IP addresses of some large websites, which use CDN and load balancing technologies, vary at different times (an example will be provided below, where running the same code several times to request the IP address of "www.facebook.com" yields different outputs each time). Such websites cannot be validated through "dnschecker.org".



```
● (d2l) faye@FayedeMacBook-Air project110 % python dns_resolver.py www.github.com
  Iterative dns resolver:    github.com. 60 IN A 20.205.243.166
  Recursive dns resolver:    github.com. 60 IN A 20.205.243.166
● (d2l) faye@FayedeMacBook-Air project110 % python dns_resolver.py www.overleaf.com
  Iterative dns resolver:    lb2.overleaf.com. 300 IN A 34.120.52.64
  Recursive dns resolver:    lb2.overleaf.com. 300 IN A 34.120.52.64
● (d2l) faye@FayedeMacBook-Air project110 % python dns_resolver.py www.python.org
  Iterative dns resolver:    dualstack.python.map.fastly.net. 29 IN A 151.101.76.223
  Recursive dns resolver:    dualstack.python.map.fastly.net. 29 IN A 151.101.76.223
● (d2l) faye@FayedeMacBook-Air project110 % python dns_resolver.py www.facebook.com
  Iterative dns resolver:    www.facebook.com. 195 IN A 157.240.12.36
  Recursive dns resolver:    www.facebook.com. 195 IN A 157.240.12.36
● (d2l) faye@FayedeMacBook-Air project110 % python dns_resolver.py www.instagram.com
  Iterative dns resolver:    www.instagram.com. 145 IN A 199.59.149.237
  Recursive dns resolver:    www.instagram.com. 145 IN A 199.59.149.237
```



DNS CHECK

www.github.com | A | Search

[✓] CD Flag   Refresh: 20 sec.

Expected Value: ( ) RegExp ( ) Contains (●) Exact Match

20.205.243.166

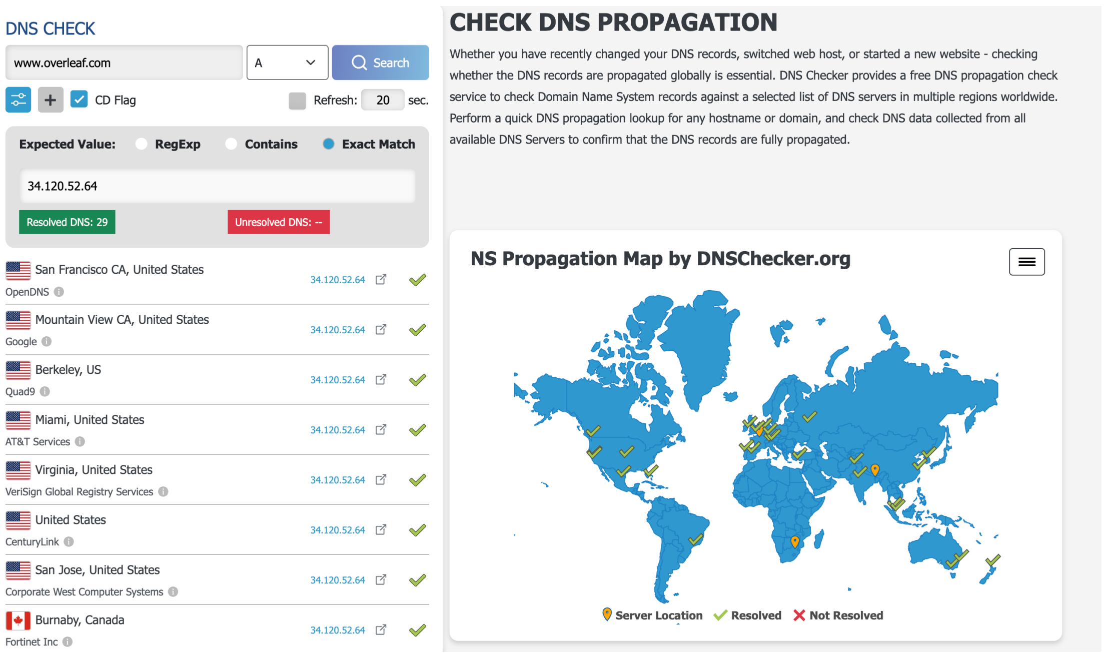
Resolved DNS: 3   Unresolved DNS: 29

San Francisco CA, United States OpenDNS	140.82.112.4	✗
Mountain View CA, United States Google	140.82.112.3	✗
Berkeley, US Quad9	140.82.114.4	✗
Miami, United States AT&T Services	140.82.112.3	✗
Virginia, United States VeriSign Global Registry Services	140.82.112.4	✗
United States CenturyLink	140.82.113.4	✗
San Jose, United States Corporate West Computer Systems	140.82.112.4	✗
Burnaby, Canada Fortinet Inc	140.82.113.3	✗

CHECK DNS PROPAGATION

Whether you have recently changed your DNS records, switched web host, or started a new website - checking whether the DNS records are propagated globally is essential. DNS Checker provides a free DNS propagation check service to check Domain Name System records against a selected list of DNS servers in multiple regions worldwide. Perform a quick DNS propagation lookup for any hostname or domain, and check DNS data collected from all available DNS Servers to confirm that the DNS records are fully propagated.

NS Propagation Map by DNSChecker.org



DNS CHECK

www.overleaf.com | A | Search

[✓] CD Flag   Refresh: 20 sec.

Expected Value: ( ) RegExp ( ) Contains (●) Exact Match

34.120.52.64

Resolved DNS: 29   Unresolved DNS: --

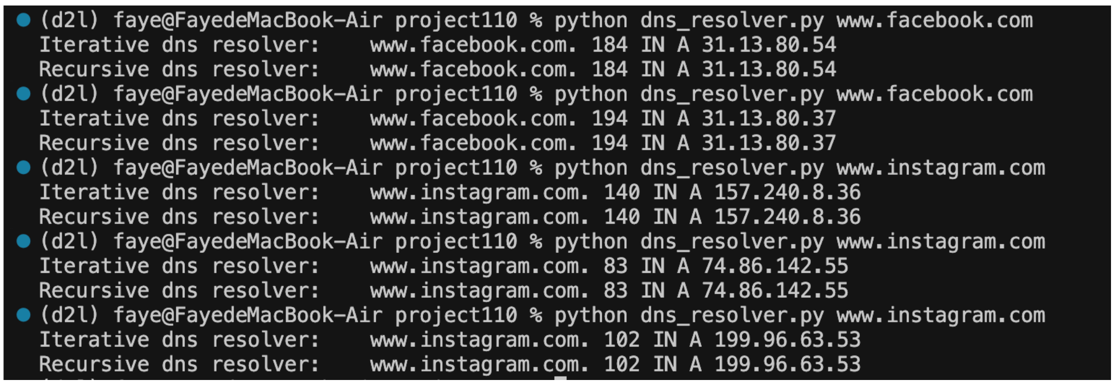
San Francisco CA, United States OpenDNS	34.120.52.64	✓
Mountain View CA, United States Google	34.120.52.64	✓
Berkeley, US Quad9	34.120.52.64	✓
Miami, United States AT&T Services	34.120.52.64	✓
Virginia, United States VeriSign Global Registry Services	34.120.52.64	✓
United States CenturyLink	34.120.52.64	✓
San Jose, United States Corporate West Computer Systems	34.120.52.64	✓
Burnaby, Canada Fortinet Inc	34.120.52.64	✓

CHECK DNS PROPAGATION

Whether you have recently changed your DNS records, switched web host, or started a new website - checking whether the DNS records are propagated globally is essential. DNS Checker provides a free DNS propagation check service to check Domain Name System records against a selected list of DNS servers in multiple regions worldwide. Perform a quick DNS propagation lookup for any hostname or domain, and check DNS data collected from all available DNS Servers to confirm that the DNS records are fully propagated.

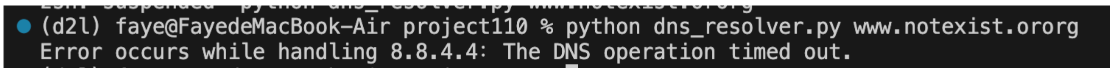
NS Propagation Map by DNSChecker.org

Domains that IPs change frequently:



```
(d2l) faye@FayedeMacBook-Air project110 % python dns_resolver.py www.facebook.com
Iterative dns resolver:    www.facebook.com. 184 IN A 31.13.80.54
Recursive dns resolver:    www.facebook.com. 184 IN A 31.13.80.54
(d2l) faye@FayedeMacBook-Air project110 % python dns_resolver.py www.facebook.com
Iterative dns resolver:    www.facebook.com. 194 IN A 31.13.80.37
Recursive dns resolver:    www.facebook.com. 194 IN A 31.13.80.37
(d2l) faye@FayedeMacBook-Air project110 % python dns_resolver.py www.instagram.com
Iterative dns resolver:    www.instagram.com. 140 IN A 157.240.8.36
Recursive dns resolver:    www.instagram.com. 140 IN A 157.240.8.36
(d2l) faye@FayedeMacBook-Air project110 % python dns_resolver.py www.instagram.com
Iterative dns resolver:    www.instagram.com. 83 IN A 74.86.142.55
Recursive dns resolver:    www.instagram.com. 83 IN A 74.86.142.55
(d2l) faye@FayedeMacBook-Air project110 % python dns_resolver.py www.instagram.com
Iterative dns resolver:    www.instagram.com. 102 IN A 199.96.63.53
Recursive dns resolver:    www.instagram.com. 102 IN A 199.96.63.53
```

Error handling test: My DNS resolver also include error handling. If we input a non-exist domain, it will print information below and exit this program.



```
● (d2l) faye@FayedeMacBook-Air project110 % python dns_resolver.py www.notexist.ororg
  Error occurs while handling 8.8.4.4: The DNS operation timed out.
```

Conclusion and Future Work

This project is a valuable learning experience, providing me a practical opportunity to realize what we have learned in the class. The project successfully implemented the DNS resolver which can handle both recursive and iterative query, along with a functioning cache mechanism.

If this project were to be continued, future work would focus on:

Enhance Error Handling: Improving the system's ability to specifically handle different kinds of errors.

Improve test ability: Improving test ability by finding out how to test domains that use load balancing and CDN technologies, to prove that the IP addresses outputted by the program are correct.