Introduction:

Our overall project goals are to implement a DNS resolver in Golang. We made a resolver that allowed for recursive and iterative queries starting with the root name server. Some stretch goals we laid out were adding support for caching and allowing for filtering DNS based on categories of content.

Design/Implementation:

- Our Resolver API has functions to conduct iterative and recursive resolution
- We have decided to use this DNS library for serializing the DNS messages
- Currently, our iterative resolver only returns answers when the record type is A. If the answer needs us to search through a different record type, it will return "no answer found for query".

Recursive Resolver

- Our recursive resolver starts a UDP connection to a public server at "8.8.8.8:53" (since Port 53 is the default port for DNS)
- Then it formats a DNS packet with the query and with the recursion bit on
- Once a response is received, it is printed into the terminal. If there is no response, we print "No answer found for this query" to the terminal.

Iterative Resolver

- Our iterative resolver starts an initial UDP connection with a.root-servers.net at "198.41.0.4"
- Using that response, if there is an answer, we return the whole message. Otherwise we iterate through each record in the additional section.
- For each of the records, we check to see if it is type A. If it is, then we send a query to the IP of that record and recursively query the results if there is no answer. Otherwise, we skip and move on to the next record.
- If all of the possible records were queried and there were no answers returned, then we don't return a DNS message and instead print "No answer found for this query" to the terminal.

Running the program

- Use the command `go run ./cmd/host/main.go` or call `make all` to compile the code and run the code
- In the terminal, make your queries by typing in "`-[r] [domain to query]`" where r is either t, resolve recursively, or f, resolver iteratively.

- Ex. “`-t cs.brown.edu`” or “`-f cs.brown.edu`”

Discussion/Results:

Query: google.com

Our recursive resolver:

```
> -t google.com
 performing recursive resolver
 ;; opcode: QUERY, status: NOERROR, id: 57080
 ;; flags: qr rd ra; QUERY: 1, ANSWER: 1, AUTHORITY: 0, ADDITIONAL: 0

 ;; QUESTION SECTION:
 ;google.com.    IN       A

 ;; ANSWER SECTION:
 google.com.     221     IN      A       142.250.81.238
```

Our iterative resolver:

```
cs1680-user@0a32f3eb1b47:~/DNS-Resolver$ go run ./cmd/host/main.go
 > awaiting query
> -f google.com
 performing iterative resolver
 ;; opcode: QUERY, status: NOERROR, id: 58056
 ;; flags: qr aa; QUERY: 1, ANSWER: 1, AUTHORITY: 0, ADDITIONAL: 0

 ;; QUESTION SECTION:
 ;google.com.    IN       A

 ;; ANSWER SECTION:
 google.com.     300     IN      A       142.251.40.238
```

Dig:

```
cs1680-user@0a32f3eb1b47:~/DNS-Resolver$ dig google.com

; <<>> DiG 9.18.28-0ubuntu0.22.04.1-Ubuntu <<>> google.com
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 16972
;; flags: qr rd ra; QUERY: 1, ANSWER: 1, AUTHORITY: 0, ADDITIONAL: 0

;; QUESTION SECTION:
;google.com.                    IN      A

;; ANSWER SECTION:
google.com.             320     IN      A       142.251.40.110

;; Query time: 36 msec
;; SERVER: 127.0.0.11#53(127.0.0.11) (UDP)
;; WHEN: Tue Dec 17 22:58:31 EST 2024
;; MSG SIZE  rcvd: 44
```

Query: clubpenguin.com

Our recursive resolver:

```
> -t clubpenguin.com
performing recursive resolver
;; opcode: QUERY, status: NOERROR, id: 14049
;; flags: qr rd ra; QUERY: 1, ANSWER: 2, AUTHORITY: 0, ADDITIONAL: 0

;; QUESTION SECTION:
;clubpenguin.com.       IN       A

;; ANSWER SECTION:
clubpenguin.com.        14400   IN      A       165.160.15.20
clubpenguin.com.        14400   IN      A       165.160.13.20
```

Our iterative resolver:

```
> -f clubpenguin.com
performing iterative resolver
No answer found for this query> awaiting query
```

Dig:

```
cs1680-user@0a32f3eb1b47:~/DNS-Resolver$ dig clubpenguin.com

; <<>> DiG 9.18.28-0ubuntu0.22.04.1-Ubuntu <<>> clubpenguin.com
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 28794
;; flags: qr rd ra; QUERY: 1, ANSWER: 2, AUTHORITY: 0, ADDITIONAL: 0

;; QUESTION SECTION:
;clubpenguin.com.               IN      A

;; ANSWER SECTION:
clubpenguin.com.        4502    IN      A       165.160.13.20
clubpenguin.com.        4502    IN      A       165.160.15.20

;; Query time: 19 msec
;; SERVER: 127.0.0.11#53(127.0.0.11) (UDP)
;; WHEN: Tue Dec 17 23:23:56 EST 2024
;; MSG SIZE  rcvd: 65
```

Discussion

Our recursive and iterative resolver returns the expected answers when the servers that it needs to query are type A. Otherwise it returns a message that says "No answer found for this query". Since our iterative resolver only allows queries for A record types, it was unable to get an answer for the clubpenguin.com query. We know this because within our implementation, we explicitly send queries only to A record types. When using the +norec flag through supposed iterative queries with the Dig tool, we get the following series of servers:

@a.root.servers
l.gtld-servers.net. 172800 IN A 192.41.162.30

@192.41.162.30
clubpenguin.com. 172800 IN NS dns1.cscdns.net.

@dns1.cscdns.net
clubpenguin.com. 14400 IN A 165.160.13.20
clubpenguin.com. 14400 IN A 165.160.15.20

From the results, we can see that we need to query an NS record type eventually to get to the answer. When making the queries of our iterative resolver verbose, we can see that we get no answer when we encounter that NS record type.

```
cs1680-user@0a32f3eb1b47:~/DNS-Resolver$ go run ./cmd/host/main.go
> awaiting query
> -f clubpenguin.com
performing iterative resolver
;; opcode: QUERY, status: NOERROR, id: 29707
;; flags: qr tc; QUERY: 1, ANSWER: 0, AUTHORITY: 13, ADDITIONAL: 11

;; QUESTION SECTION:
;clubpenguin.com.       IN       A

;; AUTHORITY SECTION:
com.    172800  IN      NS      l.gtld-servers.net.
com.    172800  IN      NS      j.gtld-servers.net.
com.    172800  IN      NS      h.gtld-servers.net.
com.    172800  IN      NS      d.gtld-servers.net.
com.    172800  IN      NS      b.gtld-servers.net.
com.    172800  IN      NS      f.gtld-servers.net.
com.    172800  IN      NS      k.gtld-servers.net.
com.    172800  IN      NS      m.gtld-servers.net.
com.    172800  IN      NS      i.gtld-servers.net.
com.    172800  IN      NS      g.gtld-servers.net.
com.    172800  IN      NS      a.gtld-servers.net.
com.    172800  IN      NS      c.gtld-servers.net.
com.    172800  IN      NS      e.gtld-servers.net.

;; ADDITIONAL SECTION:
l.gtld-servers.net.     172800  IN      A       192.41.162.30
l.gtld-servers.net.     172800  IN      AAAA    2001:500:d937::30
j.gtld-servers.net.     172800  IN      A       192.48.79.30
j.gtld-servers.net.     172800  IN      AAAA    2001:502:7094::30
h.gtld-servers.net.     172800  IN      A       192.54.112.30
h.gtld-servers.net.     172800  IN      AAAA    2001:502:8cc::30
d.gtld-servers.net.     172800  IN      A       192.31.80.30
d.gtld-servers.net.     172800  IN      AAAA    2001:500:856e::30
b.gtld-servers.net.     172800  IN      A       192.33.14.30
b.gtld-servers.net.     172800  IN      AAAA    2001:503:231d::2:30
f.gtld-servers.net.     172800  IN      A       192.35.51.30

first this in additional section:  l.gtld-servers.net.  172800  IN      A       192.41.162.30
response:  ;; opcode: QUERY, status: NOERROR, id: 54189
;; flags: qr; QUERY: 1, ANSWER: 0, AUTHORITY: 2, ADDITIONAL: 0

;; QUESTION SECTION:
;clubpenguin.com.       IN       A

;; AUTHORITY SECTION:
clubpenguin.com.        172800  IN      NS      dns1.cscdns.net.
clubpenguin.com.        172800  IN      NS      dns2.cscdns.net.

answer len:  0
No answer found for this query> awaiting query
```

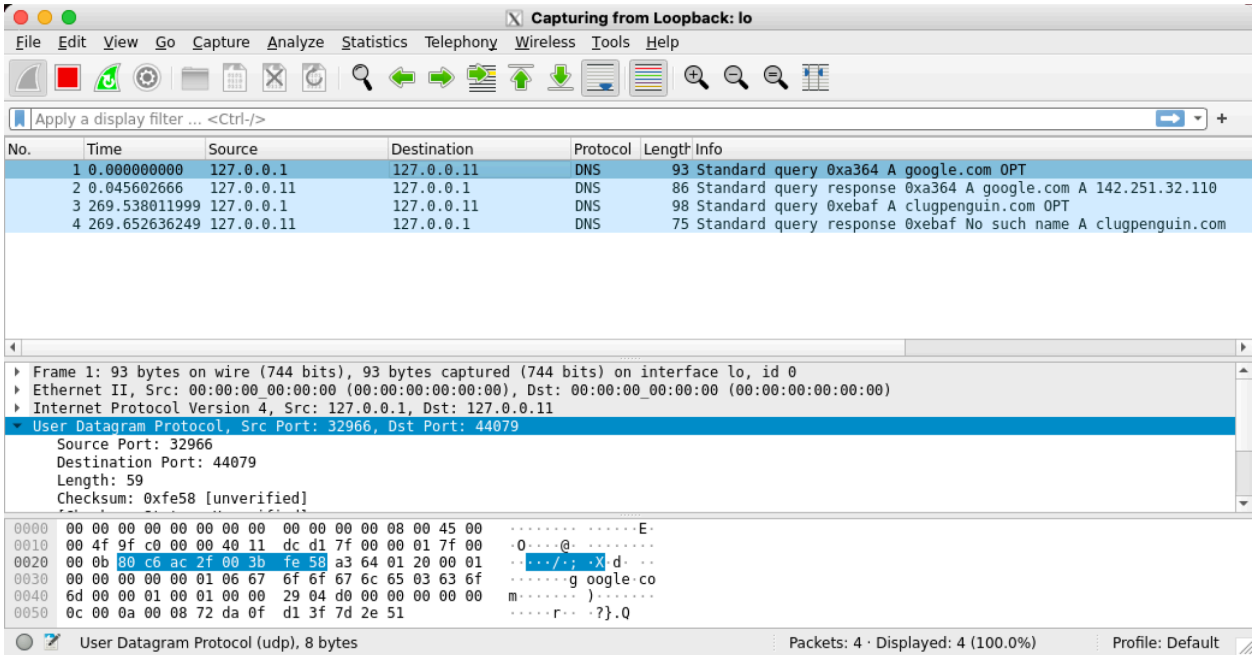
② the server we query next

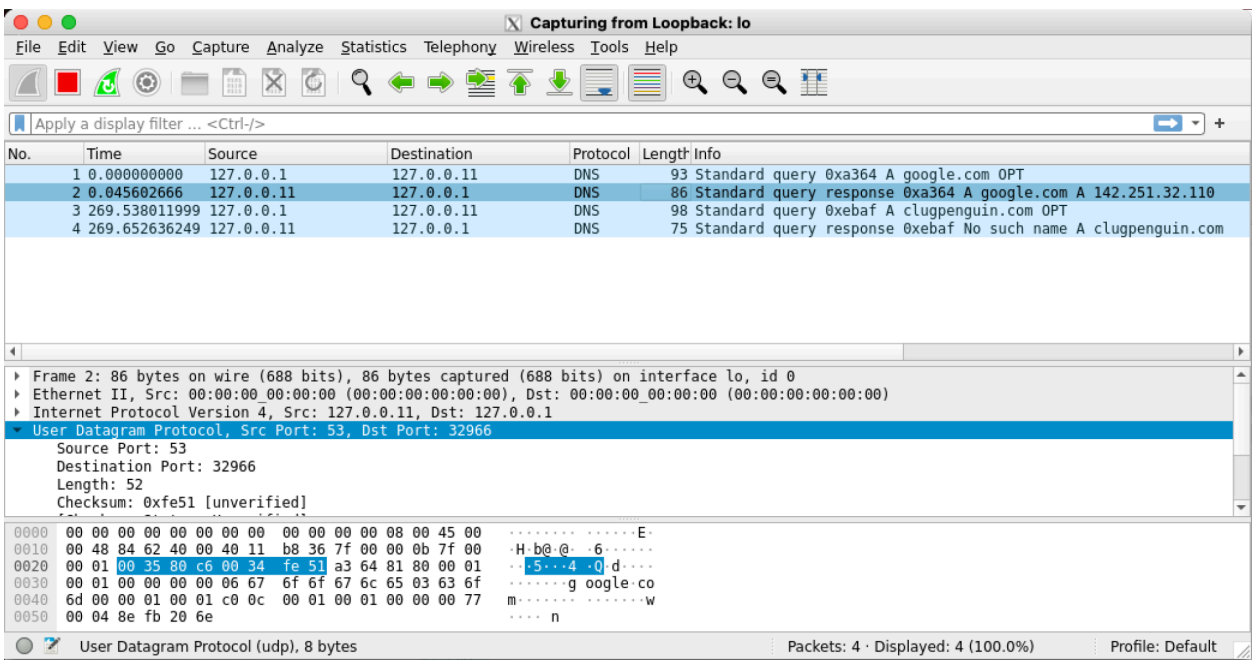
① results from root server

no type A records

③ the results from querying l.gtld

Additionally, wireshark doesn't seem to track our DNS packets being sent out. In our current implementation, we create a udp4 connection on port 53 and varying IP addresses to send and receive messages from. Comparatively, Dig sends out the initial packet from a random port to a specific port for each query (44079). Then it receives the DNS response from port 53.





We are speculating that wireshark is not recognizing our packets as DNS packets because of this difference in port numbers.

Demo

DNS-resolver-demo.mp4

Conclusions/Future work:

Throughout this project, we had fun learning about how DNS packets are transmitted and interpreted. Understanding the dns library was slightly challenging due to us not knowing how a DNS packet looked to be able to compare it to the structs in the library. We were able to do some research online to learn about the parts that make up the header and main body of a DNS packet. From there, we were able to implement our resolver without an issue.

In the future, we'd like to add support for caching and also filtering by content type to try and capture the full functionality of modern DNS resolvers. If we were to filter by content type, we anticipate having to add support for different types of records that we'd be able to query for. As for caching, we would just need to implement a data structure to keep track of when queries are made and when they have to be reaped.