GRDNS : A DNS caching/resolving server written in golang with redis DB

Team Members :

- P K Navin Shrinivas [PES2UG20CS237]
- Mohamed Ayaan [PES2UG20CS200]
- Mukund Deepak [PES2UG20CS206]

Abstract of project

The aim was to build a very fast and hyper stable multithreaded DNS server. We achieved the following during the project :

- Resolving from other known DNS server
- Caching in redis database
- Maintaining in-memory indexes of redis records for hyper fast resolve times
- A moderately resiliant implementation of multi threading.

We surely wanted to do more on this project such as :

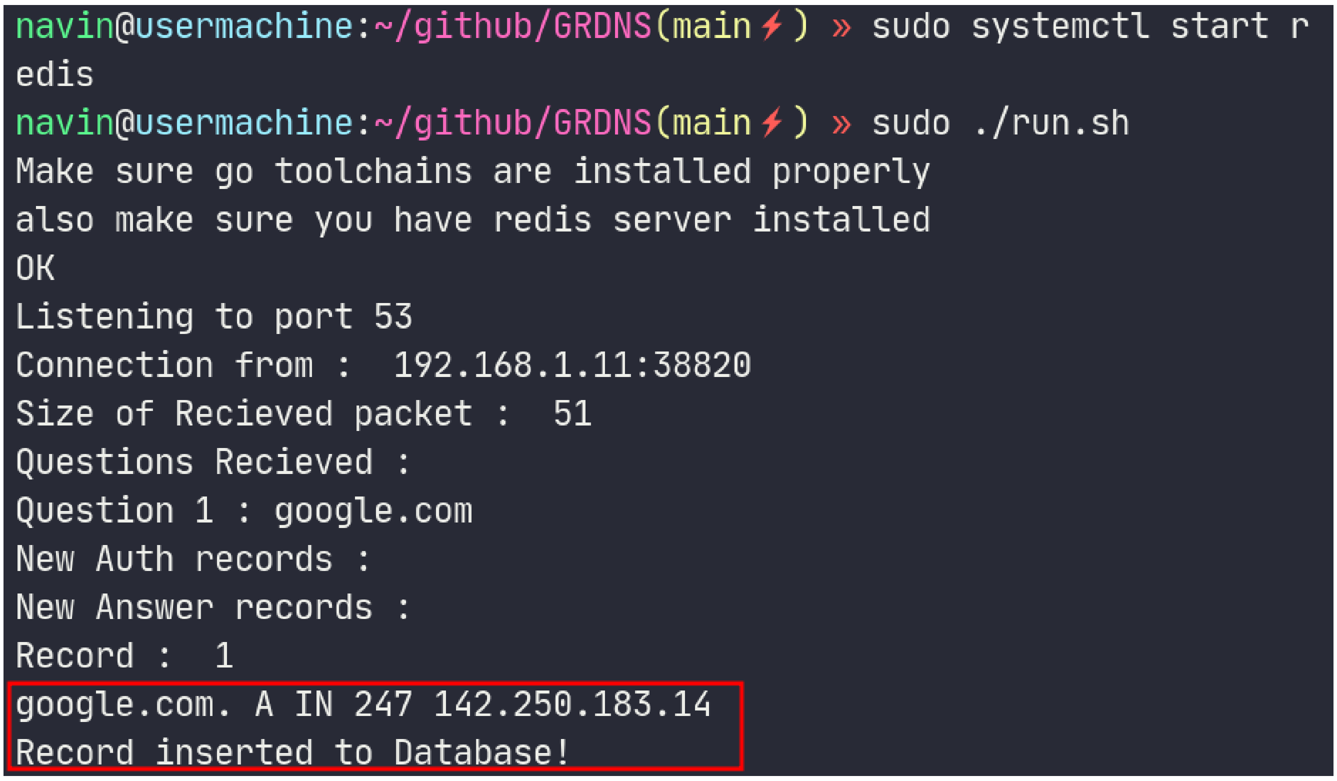
- recursive resolving from root and TLD server.
- Handle CNAME,NS records
- More better multithreading database operations

But given the time frame for this project, all of the following could not be done.

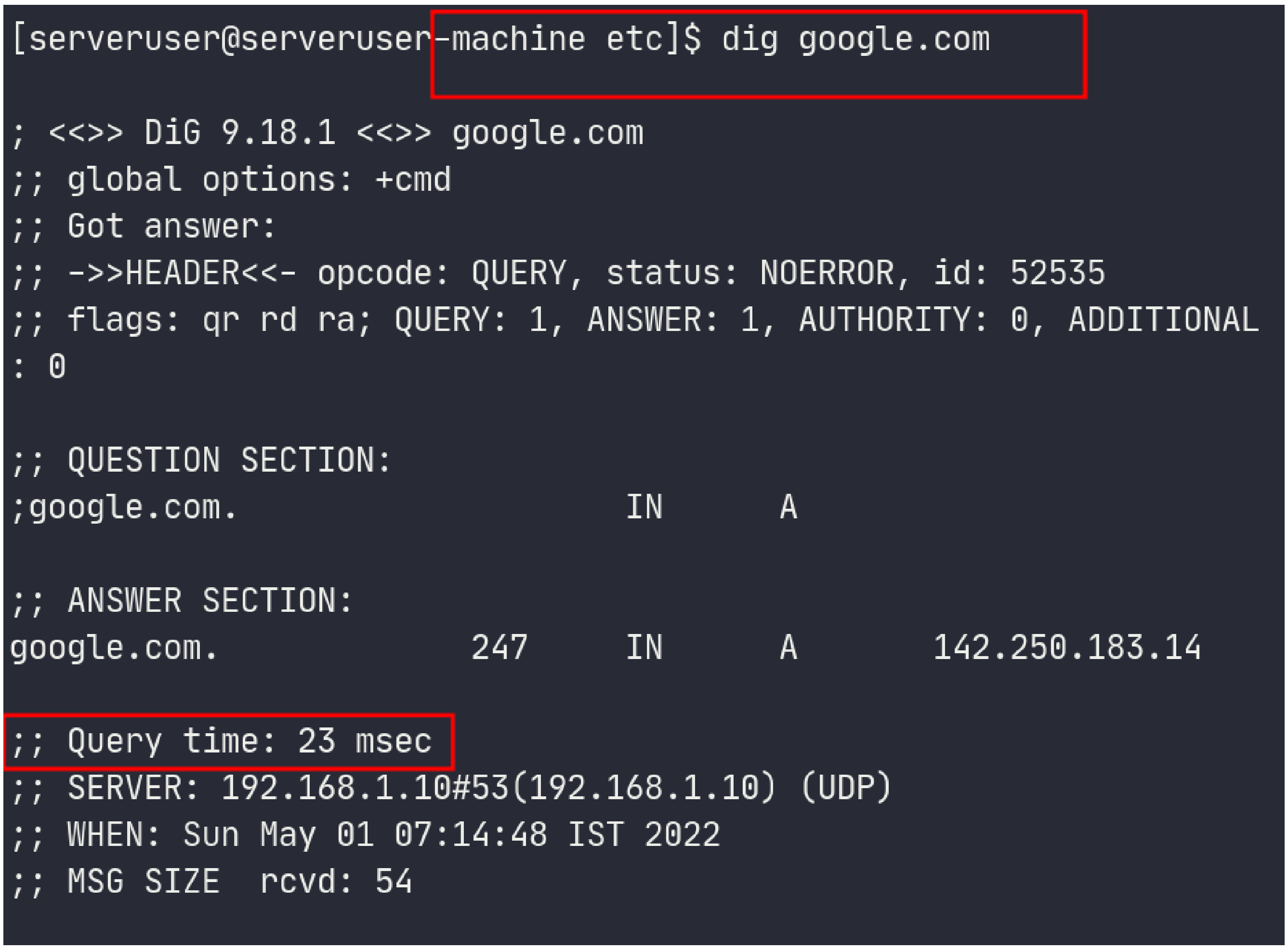
Output screenshots :

Server and DIG :

1.When resolving for the first time :

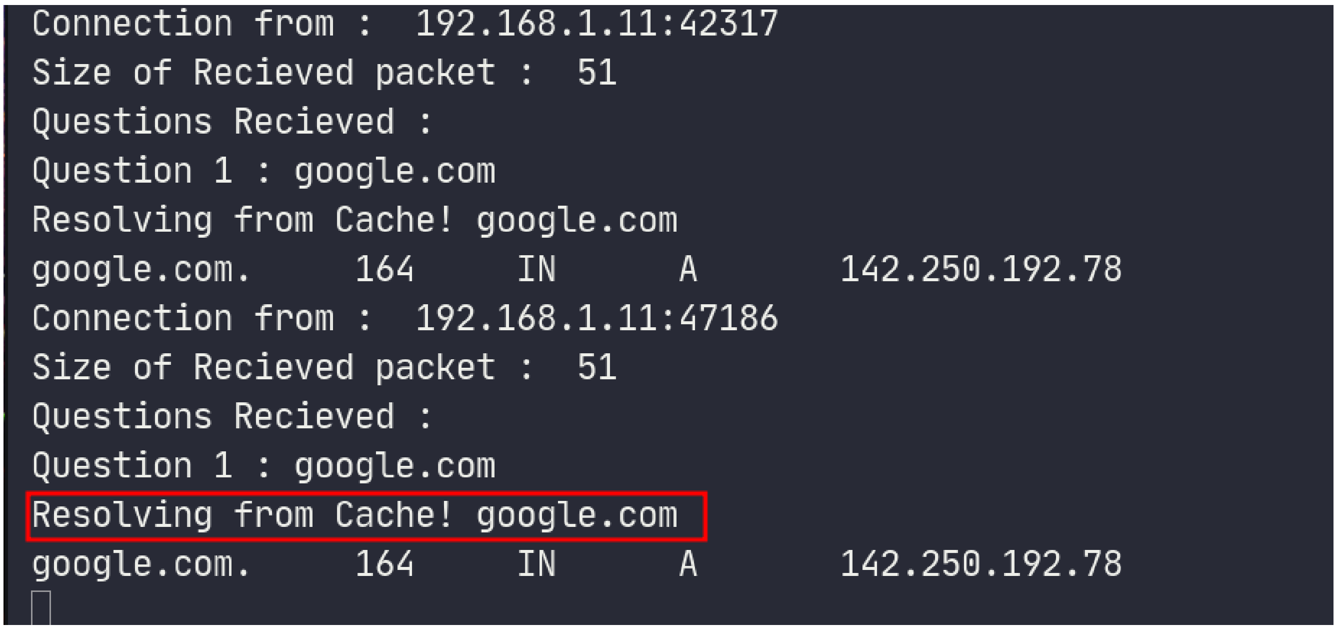


```
navin@usermachine:~/github/GRDNS(main⚡) » sudo systemctl start r
edis
navin@usermachine:~/github/GRDNS(main⚡) » sudo ./run.sh
Make sure go toolchains are installed properly
also make sure you have redis server installed
OK
Listening to port 53
Connection from :  192.168.1.11:38820
Size of Recieved packet :  51
Questions Recieved :
Question 1 : google.com
New Auth records :
New Answer records :
Record :  1
google.com. A IN 247 142.250.183.14
Record inserted to Database!
```

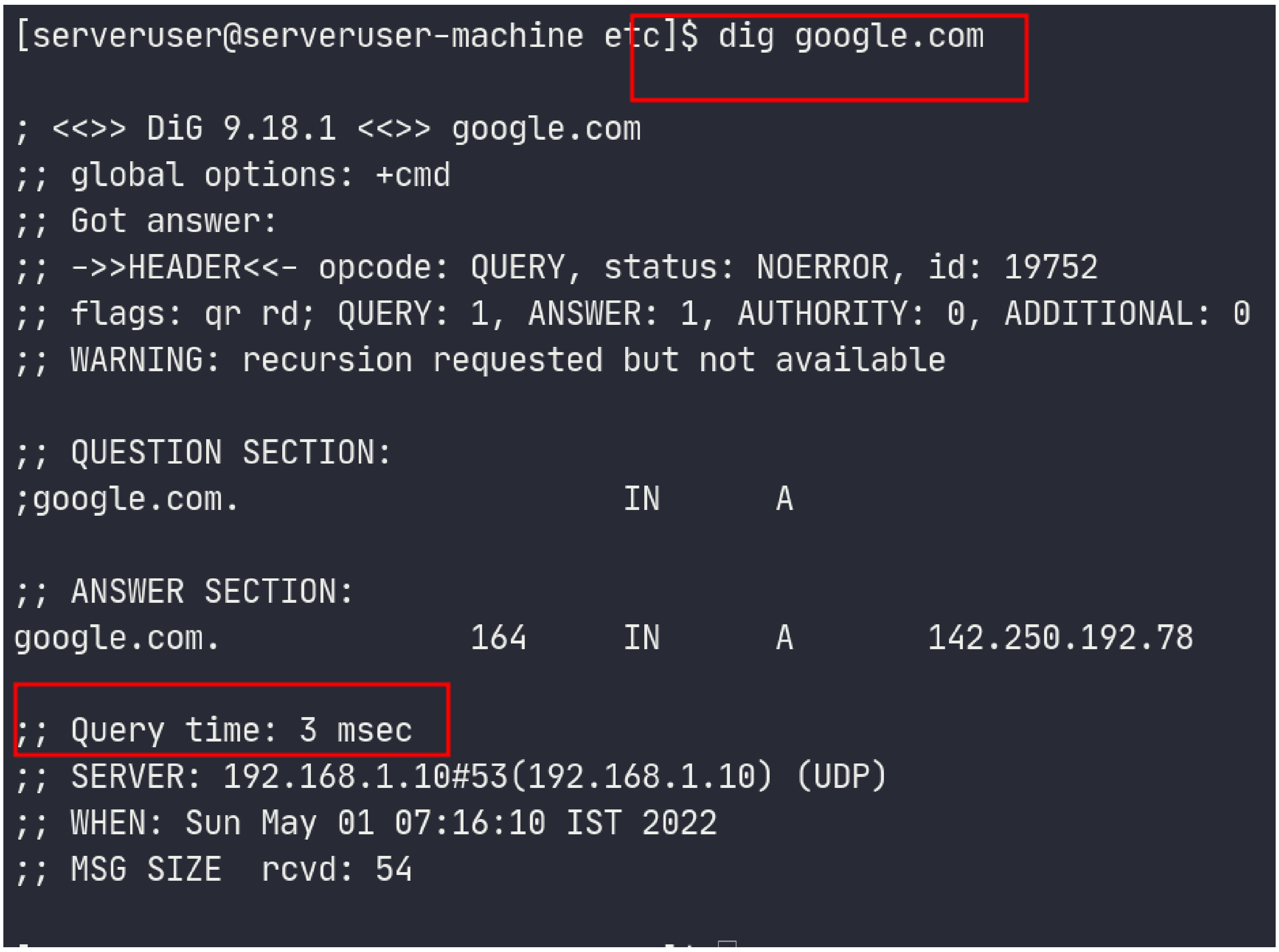


```
[serveruser@serveruser-machine etc]$ dig google.com

; <<>> DiG 9.18.1 <<>> google.com
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 52535
;; flags: qr rd ra; QUERY: 1, ANSWER: 1, AUTHORITY: 0, ADDITIONAL
: 0

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

;; ANSWER SECTION:
google.com.             247     IN      A       142.250.183.14

;; Query time: 23 msec
;; SERVER: 192.168.1.10#53(192.168.1.10) (UDP)
;; WHEN: Sun May 01 07:14:48 IST 2022
;; MSG SIZE  rcvd: 54
```

2.When resolving for the second time :



```
Connection from :  192.168.1.11:42317
Size of Recieved packet :  51
Questions Recieved :
Question 1 : google.com
Resolving from Cache! google.com
google.com.     164     IN      A       142.250.192.78
Connection from :  192.168.1.11:47186
Size of Recieved packet :  51
Questions Recieved :
Question 1 : google.com
Resolving from Cache! google.com
google.com.     164     IN      A       142.250.192.78
```



```
[serveruser@serveruser-machine etc]$ dig google.com

; <<>> DiG 9.18.1 <<>> google.com
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 19752
;; flags: qr rd; QUERY: 1, ANSWER: 1, AUTHORITY: 0, ADDITIONAL: 0
;; WARNING: recursion requested but not available

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

;; ANSWER SECTION:
google.com.             164     IN      A       142.250.192.78

;; Query time: 3 msec
;; SERVER: 192.168.1.10#53(192.168.1.10) (UDP)
;; WHEN: Sun May 01 07:16:10 IST 2022
;; MSG SIZE  rcvd: 54
```

The Project/Code :

We have this project up on github and open for contributions at all times : here

As for this submission, I have included only significant parts of the code as the code base is large and can not be fit in a pdf file (ZIP file attached instead).

Few significant parts of the code :

handle_request :

```
func handle_request(buffer []byte,Caddr *net.UDPAddr,Conn *net.UDPCon
    packetlayers := gopacket.NewPacket(buffer,layers.LayerTypeDNS,gop
    //Above gives a set of layer of the packet revieved
    //Where the DNS layer is filled with our Recieved bits
    DNSlayer := packetlayers.Layer(layers.LayerTypeDNS)
    //Above only extracts the DNS layer from set of layers
    //with above layer we can create an object :)
    DNSpacketObj := DNSlayer.(*layers.DNS)
    fmt.Println("Questions Recieved : ")
    for i,it:=range DNSpacketObj.Questions{
        fmt.Println("Question",i+1,":",string(it.Name))

        req_id := DNSpacketObj.ID; //Used by All DNS systems to ensur
        var response = new(dns.Msg);
        if EntryExists(string(it.Name)){
            response.MsgHdr.Response = true;
            response.MsgHdr.Rcode = 0; //No error handling :(
            response.MsgHdr.RecursionDesired = true;
            l := new(dns.Msg)
            l.Unpack(buffer)
            response.Question = l.Question;
            ReturnWithAnswers(string(it.Name),response)

        }else{
           response = resolve(string(it.Name))
        }


        if response!=nil{
            response.MsgHdr.Id = req_id;
            resbuf,_ := response.Pack()

            //Writing back to client
            _, err := Conn.WriteToUDP(resbuf, Caddr)
            checkError(err)
        }
    }
}
```

database function :

```
func FlushToDB(Record ResponseStruct) bool {

    var pool = newPool()
    var c = pool.Get()

    _,err := c.Do("HSET", record_number,"name", Record.Name,"ttl",Rec
    Record.Rawclass, "type", Record.Rawrrtype, "reply", Record.Rawstr
    if err != nil {
        checkError(err)
        return false
    }
    domain_map[Record.Name] = append(domain_map[Record.Name],record_n
    record_number++;
    fmt.Println("Record inserted to Database!")
    return true
}
```