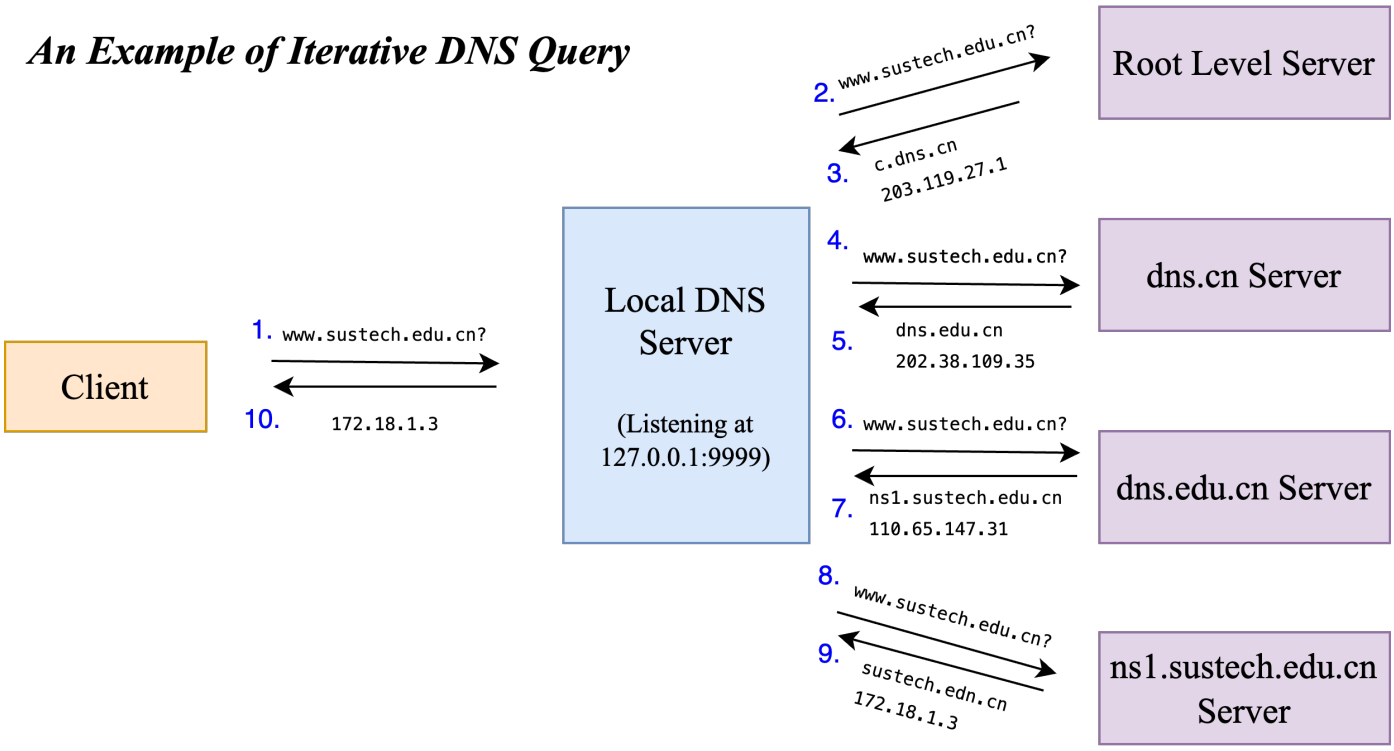
CS305 Programming Assignment 1 Local DNS Server

In this programming assignment, you need to implement a `Local DNS Server` which supports iterative DNS query with Python.

We provide some starter code in this GitHub repository. You should check this link for more details. If you have any questions, please raise an issue or contact Yifei Li (李逸飞) at 12232396@mail.sustech via direct message on QQ or email.

Note: You are not allowed to use third-party libraries such as `dnspython`, `dnslib`, or any other library specifically designed for parsing DNS packets. However, other basic standard libraries like IO and socket are allowed.

Overview



Here is an example of an iterative DNS query about resloving the ip address of `www.sustech.edu.cn`. In this project, we need to implement a local DNS server that listens on 127.0.0.1:9999. When a DNS query is received, the local DNS server will send an iterative query to the root server. Based on the reply from the root server, it will send a query to the next server until it receives the final result. The result will be returned to the client finally.

client: `dig` will be used to test your local DNS server. For example, you can send a DNS query to the local DNS server listening on port 9999 by entering `dig @127.0.0.1 www.sustech.edu.cn a -p 9999` in the command line. When `dig` receives a valid reply, it will print out the response. Here is an example of the output of `dig`:

```
(base) ➜  CS305-Assignment1 git:(main) ✗ dig @127.0.0.1 www.sustech.edu.cn a -p 9999


; <<>> DiG 9.10.6 <<>> @127.0.0.1 www.sustech.edu.cn a -p 9999
; (1 server found)
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 30163
;; flags: qr aa ra; QUERY: 1, ANSWER: 2, AUTHORITY: 2, ADDITIONAL: 5

;; OPT PSEUDOSECTION:
; EDNS: version: 0, flags:; udp: 4096
;; QUESTION SECTION:
;www.sustech.edu.cn.    IN  A

;; ANSWER SECTION:
www.sustech.edu.cn. 3600  IN  CNAME sustech.edu.cn.
sustech.edu.cn.   3600  IN  A 172.18.1.3

;; AUTHORITY SECTION:
sustech.edu.cn.   3600  IN  NS  ns1.sustech.edu.cn.
sustech.edu.cn.   3600  IN  NS  ns2.sustech.edu.cn.

;; ADDITIONAL SECTION:
ns1.sustech.edu.cn. 3600  IN  A 172.18.1.92
ns1.sustech.edu.cn. 3600  IN  AAAA  2001:da8:201d::42:92
ns2.sustech.edu.cn. 3600  IN  A 172.18.1.93
ns2.sustech.edu.cn. 3600  IN  AAAA  2001:da8:201d::42:93

;; Query time: 3467 msec
;; SERVER: 127.0.0.1#9999(127.0.0.1)
;; WHEN: Thu Mar 07 18:33:30 CST 2024
;; MSG SIZE  rcvd: 380
```

local DNS server: The local DNS server will listen on `127.0.0.1:9999`. Since the use of libraries like `dnspython` and `dnslib` is not allowed, you need to implement DNS packet parsing and construction on your own. We provide a backbone file named `local_dns_server.py`, which includes a simple UDP server `MyLocalDNSServerHandler`. You are required to implement a complete iterative DNS query and return the result in the `handle` function of this class. **Please pay attention to every comment in the file.**

Detailed Guidlines

Note: RFC1035 is the main reference for this assignment.

Environment

Python Environment

You should use python 3.8 as the execution environment. Higher versions of Python might lead to errors.

You can use conda to create an environment by `conda create -n assign1 python=3.8`.
Libraries specifically designed for parsing DNS packets are not allowed in this assignment. If you want to introduce any third-party libraries necessarily (not installed by anaconda), please contact SA or discuss in the issues.

Dig

`dig` is a command-line tool for querying DNS servers for information about a domain name. For linux and macOS, `dig` is pre-installed. For Windows, you can check How to install dig or Windows安装dig命令.

Parse the DNS Packet

Since we are not allowed to use third-party libraries, we need to parse the DNS packet manually. The format of a DNS packet is shown below:

```
+---------------------+
|        Header       |
+---------------------+
|       Question      | the question for the name server
+---------------------+
|        Answer       | RRs answering the question
+---------------------+
|      Authority      | RRs pointing toward an authority
+---------------------+
|      Additional     | RRs holding additional information
+---------------------+
```

The DNS query and DNS response are both in the same format. It is clear that a DNS packet consists of header, question, answer, authority, and additional sections. The last three sections (answer, authority, and additional) are constructed by the resource record (RR).
Therefore, we design 4 classes in `local_dns_server.py` to represent the DNS packet (`DNSMessage`), header (`DNSHeader`), question (`DNSQuestion`), and RR (`DNSRR`).

Parse the DNS Header (10 points)

Let's begin with parsing a DNS header.

```
                                1  1  1  1  1  1
  0  1  2  3  4  5  6  7  8  9  0  1  2  3  4  5
+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
|                      ID                       |
+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
```

```
|QR|   Opcode  |AA|TC|RD|RA|   Z    |   RCODE   |
+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
|                    QDCOUNT                    |
+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
|                    ANCOUNT                    |
+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
|                    NSCOUNT                    |
+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
|                    ARCOUNT                    |
+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
```

A typical DNS header consists of 12 bytes. Bytes 0-1 are the ID, bytes 2-3 are the flags, and bytes 4-11 are the counts. You need to parse the header and store the information in the `DNSHeader` class.
`DHSHeader` class has the following attributes:

```python
self.id : int = 0
self.flag : bytes = b'' # mantain the 2 bytes of flags
self.qdcount : int = 0  # number of entries in the question section
self.ancount : int = 0  # number of resource records in the answer section
self.nscount : int = 0  # number of resource records in the authority records section
self.arcount : int = 0  # number of resource records in the additional records section
```

Firstly, you should finish the `from_wire(data: bytes) -> DNSHeader` function to parse the header. This function is invoked by `test.py`, which takes the raw bytes of a DNS header as input and returns a `DNSHeader` object. For debugging, a raw DNS header is provided in `./raw_packet/header.raw` (only contains the first 12 bytes of a DNS packet).
You can modify the input parameter of `__init__` function if necessary. But do not change the input parameter of `from_wire` function.
To test your parsing result, you can run `python test.py`.
When you successfully parse the header, you can see the following output:

```
python test.py
**header test passed**
.FF
======================================================================
FAIL: test_question (__main__.TestDNSResolver)
----------------------------------------------------------------------
Traceback (most recent call last):
  File "test.py", line 36, in test_question
    self.assertEqual(type(question), DNSQuestion)
AssertionError: <class 'NoneType'> != <class 'local_dns_server.DNSQuestion'>

======================================================================
FAIL: test_whole_msg (__main__.TestDNSResolver)
----------------------------------------------------------------------
Traceback (most recent call last):
  File "test.py", line 44, in test_whole_msg
```

```
    self.assertEqual(type(msg), DNSMessage)
AssertionError: <class 'NoneType'> != <class 'local_dns_server.DNSMessage'>

----------------------------------------------------------------------
Ran 3 tests in 0.001s

FAILED (failures=2)
```

It shows that you pass the header test.

Parse the DNS Question (10 points)

You can find the format of a DNS question in the section 4.1.2 of RFC1035.
Similarly, we provide a bytes-format question section in `./raw_packet/question.raw` for you to test your parsing result.
We make sure that all DNS packets only contain one question.

- `self.qname`: a string representing the domain name
- `self.qtype`: an int representing the query type
- `self.qclass`: an int representing the query class

When you finish the `from_wire` and `__init__` function of `DNSQuestion`, you can run `python test.py` to test your parsing result.

Parse the DNS Message (30 points)

The `DNSMessgae` consists of the following attributes:

- `self.header`: a `DNSHeader` object
- `self.question`: a `DNSQuestion` object, all testcases will contains only one question
- `self.answer`: a list of `DNSRR` objects
- `self.authority`: a list of `DNSRR` objects
- `self.additional`: a list of `DNSRR` objects

You need to implement the `from_wire` function to parse the whole DNS packet.
The most important thing is to parse the RRs in the answer, authority, and additional sections.
All the RRs are in the same format. You can find the format of a RR in the section 4.1.3 of RFC1035.
The `DNSRR` class has the following attributes:

- `self.name`: a string representing the domain name
- `self.type`: an int representing the type of the RR
- `self.class_`: an int representing the class of the RR
- `self.ttl`: an int representing the time to live of the RR
- `self.rdlength`: an int representing the length of the RDATA field (the number of bytes)
- `self.rdata`: a string or byte object representing the RDATA field

Hint1: domain name can be represented in three formats: normal format, compressed format, and mixed format (normal and compressed). You need to handle both of them. See 4.1.4. Message compression of RFC1035.

Hint2: You only need to handle the following types of RRs: A, NS, CNAME, and AAAA. For type A, `self.rdata` should be a string of IPv4 address (e.g. 183.2.172.42). For type AAAA, `self.rdata` should be a string of IPv6 address (e.g. 2001:0DB8:AC10:FE01::). For type NS and CNAME, `self.rdata` should be a string of domain name. The `self.rdata` of other types of RRs can be byte object.

When you finsh the `from_wire` function of `DNSHeader`, `DNSQuestion` and `DNSMessage`, you can run `python test.py` to test your parsing result.
When you successfully parse the whole DNS packet, you can see the following output:

```
python test.py
**header test passed**
.**question test passed**
.**whole msg test passed**
.
----------------------------------------------------------------------
Ran 3 tests in 0.001s

OK
```

This means you pass all the tests. We will run your code with another test case (also contains 3 tests) to check your parsing result. When you pass all the tests, you can get 50 points.

Build the DNS server (40 points)

You need to finish the `handle` function in `MyLocalDNSServerHandler` class to implement a iterative DNS server. The iteration process can be terminated when there is A type RR in the answer section. If there is only CNAME type RR in the answer section, you need to send a new query to the server of the CNAME and append the answer section of the new reply to the original reply.

Only CNAME type RR in the answer section

when using `dig @127.0.0.1 www.baidu.com a -p 9999` to reslove the ip address of `www.baidu.com`, there will be only a CNAME type RR in the answer section, which points to `www.a.shifen.com`. Then you need to send a new query to the server of `www.a.shifen.com` and append the answer section of the new reply to the original reply.

```
(base) ➜  ~ dig @127.0.0.1 www.baidu.com a -p 9999
;; Warning: Message parser reports malformed message packet.

; <<>> DiG 9.10.6 <<>> @127.0.0.1 www.baidu.com a -p 9999
; (1 server found)
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 53401
;; flags: qr aa; QUERY: 1, ANSWER: 1, AUTHORITY: 0, ADDITIONAL: 1

;; OPT PSEUDOSECTION:
; EDNS: version: 0, flags:; udp: 1232
;; QUESTION SECTION:
;www.baidu.com.                 IN      A

;; ANSWER SECTION:
www.baidu.com.          1200    IN      CNAME   www.a.shifen.com.

;; Query time: 236 msec
;; SERVER: 127.0.0.1#9999(127.0.0.1)
;; WHEN: Fri Mar 08 10:52:11 CST 2024
;; MSG SIZE  rcvd: 90
```

Only query www.baidu.com

wrong!

```
(base) ➜  ~ dig @127.0.0.1 www.baidu.com a -p 9999

; <<>> DiG 9.10.6 <<>> @127.0.0.1 www.baidu.com a -p 9999
; (1 server found)
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 40214
;; flags: qr aa; QUERY: 1, ANSWER: 2, AUTHORITY: 5, ADDITIONAL: 9
;; WARNING: Message has 44 extra bytes at end

;; QUESTION SECTION:
;www.baidu.com.                 IN      A

;; ANSWER SECTION:
www.baidu.com.          1200    IN      CNAME   www.a.shifen.com.
www.a.shifen.com.       300     IN      A       182.61.200.6

;; AUTHORITY SECTION:
www.a.shifen.com.       300     IN      A       182.61.200.7
a.shifen.com.           1200    IN      NS      ns5.a.shifen.com.
a.shifen.com.           1200    IN      NS      ns1.a.shifen.com.
a.shifen.com.           1200    IN      NS      ns2.a.shifen.com.
a.shifen.com.           1200    IN      NS      ns3.a.shifen.com.

;; ADDITIONAL SECTION:
a.shifen.com.           1200    IN      NS      ns4.a.shifen.com.
ns1.a.shifen.com.       300     IN      A       110.242.68.42
ns2.a.shifen.com.       600     IN      A       220.181.33.32
ns3.a.shifen.com.       300     IN      A       36.155.132.12
ns3.a.shifen.com.       300     IN      A       153.3.238.162
ns4.a.shifen.com.       300     IN      A       14.215.177.229
ns4.a.shifen.com.       300     IN      A       111.20.4.28
ns5.a.shifen.com.       600     IN      A       180.76.76.95
ns5.a.shifen.com.       600     IN      AAAA    240e:bf:b801:1006:0:ff:b04f:346b

;; Query time: 482 msec
;; SERVER: 127.0.0.1#9999(127.0.0.1)
;; WHEN: Fri Mar 08 10:54:02 CST 2024
;; MSG SIZE  rcvd: 660
```

Query www.baidu.com
and www.a.shifen.com

right

No additional section in the reply

In some cases, there is no additional section in the reply, which means there is no IP address of the server should be queried next. But the autority section contains the NS type RR (a domain name). You should send a new query to resolve the IP address of the domain name in the autority section. When you get the IP address, you can do the next query.

You can test `www.baidu.com`, `www.example.com`, `www.bilibili.com`, and any other domain name you like.

Note: You should start the local DNS server by running `python local_dns_server.py` in one terminal. Then use `dig` to test your local DNS server in another terminal.

Report (10 points)

The report should contain the following information:

- Your name and student ID
- The most difficult part of the assignment (code and explanation)
- How to handle the situation when there is only CNAME type RR in the answer section (code and explanation)
- Capture the DNS packets by using **Wireshark** when using `dig` to test your local DNS server. You can take `www.sustech.edu.cn` or `www.baidu.com` as an example

What to submit

- `local_dns_server.py`: don't change the file name
- `SID_Name_report.pdf`: the report file should be named as `SID_Name_report.pdf`. For example, if your student ID is 12345678 and your name is Zhang San, the report file should be named as `12345678_ZhangSan_report.pdf`.

You don't need to submit the `test.py` file. Passing the tests in `test.py` is only for your own debugging, does not mean you gain the full score. We will run your code with another test case to check your parsing result.

Grading

- 10 points for parsing the DNS header
- 10 points for parsing the DNS question
- 30 points for parsing the whole DNS packet
- 40 points for building the DNS server
- 10 points for the report