1. Overview

This project aims to develop a prototype DNS system using a Linux command-line terminal, consisting of a client and server.

The main objective is to implement English domain name resolution by deeply understanding the DNS (Domain Name System) protocol. The system should support the resolution of at least four top-level domains and three-level domain names. It should handle Resource Record types such as A, MX, and CNAME, with iterative resolution. The communication between the client and local DNS server should use UDP, while DNS servers communicate using TCP.Additional features include support for PTR records, caching, and printing trace records for query paths and server response times.

The implemented functionalities include:

- Parsing domain names from standard RR (Resource Record) records stored in TXT format.
- Supporting the resolution of four top-level domains and four-level domain names, including one client and eight servers.
- Supporting the resolution of A, MX, CNAME, and PTR records.
- Correctly utilizing UDP and TCP communication protocols, both of which can be correctly parsed by Wireshark.
- Implementing caching in the Local DNS Server.

- Printing trace records in the log file.

These functionalities are achieved through the development of a DNS system prototype based on a Linux command-line terminal. The project adheres to coding best practices, and the implementation is stable and supports error handling.

2. Requirements Analysis

Basic Functionality

1. The system should support the resolution of English domain names, such as `www.bupt.edu.cn`.
2. The system should include a database of Resource Records (RR) stored in TXT files, providing information like domain name, TTL, record type, and IP addresses.
3. At least four top-level domains (`cn`, `org`, `com`, `us`) should be supported, with the ability to resolve three-level domain names.
4. The system should consist of a client and at least six DNS servers, including a local DNS server.
5. Supported RR types include A, MX, and CNAME. For MX queries, the Additional Section should include the corresponding IP address.
6. Iterative resolution should be implemented to resolve domain names.
7. UDP should be used for communication between the client and local DNS server, while TCP should be used for communication between DNS servers.
8. The application layer protocol for communication should be DNS.
9. Server data can be maintained using file storage.
10. The code should be well-documented with detailed comments, following good programming practices.
11. The system should handle errors gracefully, such as invalid commands, missing or incorrect parameters, and query failures.
12. Each group should collaborate on the project, dividing the tasks between them and submitting the design document and source code.

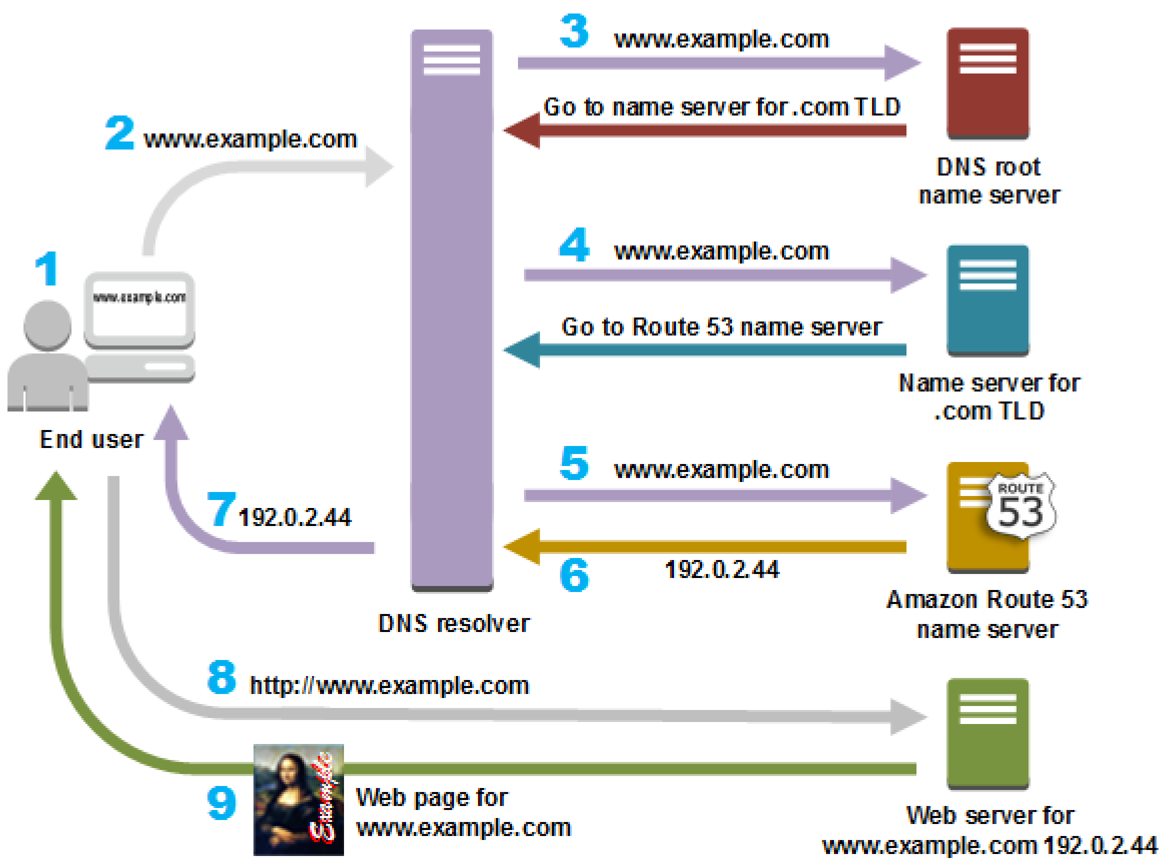
Extended Functionality

1. Support for PTR (Pointer) type of Resource Records should be added.
2. Caching mechanism should be implemented to improve performance.
3. The system should print trace records, including the query path and server response times, for better analysis and debugging.

The project aims to develop a DNS system prototype that fulfills these requirements. It will involve the implementation of the client and server components, adhering to the specified communication protocols and file storage methods. The project will be well-documented, with detailed code comments and error handling mechanisms. Additionally, the extended functionality of PTR records, caching, and trace record printing will be implemented to enhance the system's capabilities.

3. Preliminary Design

The preliminary design of the DNS system includes the following components: DNS Client, Local DNS Server, Top-Level Domain (TLD) DNS Servers, Root DNS Servers, and Intermediate DNS Servers. The system follows an iterative query approach to resolve domain names.



DNS Client:

- Sends domain name queries to the Local DNS Server.
- Uses UDP for communication with the Local DNS Server.
- Sends DNS queries using the DNS application layer protocol.

Local DNS Server:

- Receives queries from the DNS Client.
- Checks the local cache for a cached response.
- If the domain name is not found in the cache, forwards the query to the appropriate TLD DNS Server.
- Uses UDP for communication with the TLD DNS Server.

TLD DNS Servers:

- Receives queries from the Local DNS Server.
- Checks the TLD DNS server's cache for a cached response.
- If the domain name is not found in the cache, forwards the query to the appropriate Intermediate DNS Server.
- Uses UDP for communication with the Intermediate DNS Server.

Intermediate DNS Servers:

- Receives queries from the TLD DNS Servers or other Intermediate DNS Servers.
- Checks the Intermediate DNS Server's cache for a cached response.
- If the domain name is not found in the cache, forwards the query to the next-level Intermediate DNS Server.
- Uses UDP for communication with the next-level Intermediate DNS Server.

Root DNS Servers:

- The highest level of DNS servers in the system.
- Receives queries from the Intermediate DNS Servers.
- Provides authoritative responses for the TLD DNS Servers to resolve the domain name.
- Uses TCP for communication with the Intermediate DNS Servers.

The iterative query process starts with the DNS Client sending a query to the Local DNS Server. If the Local DNS Server does not have a cached response, it forwards the query to the appropriate TLD DNS Server. The TLD DNS Server checks its cache and forwards the query to the appropriate Intermediate DNS Server if needed. This process continues until the Root DNS Server provides the authoritative response to resolve the domain name.

This design allows for the resolution of domain names across multiple levels of DNS servers, ensuring the efficient and accurate retrieval of IP addresses associated with the requested domain names.

4. Detailed Design

```
.
├── CMakeLists.txt
├── build
├── data
|   ├── rr1.txt
|   ├── rr2.txt
|   ├── rr3.txt
|   ├── rr4.txt
|   ├── rr5.txt
|   ├── rr6.txt
|   ├── rr7.txt
|   └── rr8.txt
├── logs
├── readme.md
└── src
    ├── client
    |   ├── client.c
    |   └── client.h
    ├── server
    |   ├── local_server.c
    |   ├── local_server.h
    |   ├── server.c
    |   ├── server.h
    |   ├── server_args.h
    |   ├── tier_server.c
    |   └── tier_server.h
    └── utils
        ├── dns_message.c
        ├── dns_message.h
        ├── rr_reader.c
        ├── rr_reader.h
        ├── socket.c
        ├── socket.h
        ├── utils.c
        └── utils.h
```

The detailed design of the DNS system includes the following components and their respective functionalities:

Client Component

- Files: `client.c`, `client.h`
- Responsible for sending DNS queries to the Local DNS Server.
- Implements the UDP communication protocol.
- Parses user input to extract domain names for resolution.
- Utilizes the DNS message format for constructing and sending DNS queries.

Server Component

- Files: `local_server.c`, `local_server.h`, `server.c`, `server.h`, `server_args.h`, `tier_server.c`, `tier_server.h`
- Contains the implementation of different DNS servers involved in the resolution process.
- Local DNS Server:
  - Receives DNS queries from the Client.
  - Acts as a caching server, storing previously resolved queries.
  - Forwards queries to the appropriate TLD DNS Server based on the domain name.
- TLD DNS Server:
  - Receives DNS queries from the Local DNS Server.
  - Resolves queries for top-level domains (.cn, .org, .com, .us) and returns the corresponding IP addresses.
  - Uses the Intermediate DNS Server for further resolution, if required.
- Intermediate DNS Server:
  - Receives DNS queries from the TLD DNS Server or other Intermediate DNS Servers.
  - Resolves queries for specific domain levels (e.g., edu.cn, bupt.edu.cn) and returns the corresponding IP addresses.
  - Utilizes the Root DNS Server for authoritative resolution, if necessary.
- Root DNS Server:
  - Acts as the highest-level DNS server in the system.
  - Receives DNS queries from Intermediate DNS Servers.
  - Provides authoritative responses with IP addresses for the requested domain names.

Utils Component

- Files: `dns_message.c`, `dns_message.h`, `rr_reader.c`, `rr_reader.h`, `socket.c`, `socket.h`, `utils.c`, `utils.h`
- Contains utility functions used by the Client and Server components.
- `dns_message`: Implements functions for constructing and parsing DNS messages according to the DNS protocol.
- `rr_reader`: Reads and processes Resource Records (RR) from the TXT files in the `data` directory.
- `socket`: Provides functions for UDP and TCP socket communication.
- `utils`: Contains general utility functions used throughout the system, such as logging and error handling.

Additional Directories and Files

- `CMakeLists.txt`: CMake configuration file for building the project.
- `build`: Directory for the build output (compiled binaries).
- `data`: Directory containing TXT files storing the RR records for domain name resolution.
- `logs`: Directory where log files can be stored.
- `readme.md`: Markdown file containing project documentation and instructions.

The project follows a modular approach, separating different components into their respective directories. This design allows for easier code maintenance, reusability, and collaboration between team members working on different aspects of the project.

Constants and Definitions:

- DNS_PORT: Specifies the default DNS port (port 53).
- ID: Represents the DNS message ID.
- DNS message types: Constants representing different DNS record types (A, CNAME, MX, NS, SOA, TXT, AAAA, PTR).
- DNS_MAX_MESSAGE_SIZE: Maximum size of a DNS message.
- DNS_RESPONSE: Flag indicating a DNS response message.
- DNS_REQUEST: Flag indicating a DNS request message.
- DNS_RECURSION_DESIRED: Flag indicating if recursion is desired.
- DNS_AUTHORITATIVE: Flag indicating if the DNS server is authoritative.
- DNS_QUERY: Flag indicating a DNS query message.
- DNS_CLASS_IN: DNS class IN (Internet).
- Structures: dns_message, dns_header, dns_question, dns_rr, dns_query, dns_cache.

Function Declarations:

- pack_dns_message: Packs DNS message fields into a buffer.
- pack_dns_response: Packs DNS response fields into a buffer.
- parse_dns_message: Parses a DNS message buffer and extracts the answers and questions.
- unpack_dns_answer: Unpacks DNS answer fields from a buffer.
- unpack_dns_question: Unpacks DNS question fields from a buffer.
- solve_answers: Processes DNS answers and retrieves IP addresses.

Structures:

- dns_message: Represents a DNS message, containing fields such as name, type, class, TTL, RDLength, RData, and a pointer to the next message.
- dns_header: Represents a DNS message header, containing fields such as ID, flags, QDCOUNT, ANCOUNT, NSCOUNT, and ARCOUNT.
- dns_question: Represents a DNS question, containing fields such as QNAME, QTYPE, QCLASS, and QNAME length.
- dns_rr: Represents a DNS resource record, containing fields such as name, type, class, TTL, RDLength, preference, and RData.
- dns_query: Represents a DNS query, containing fields for header, question, answer, authority, and additional records.
- dns_cache: Represents a DNS cache entry, containing a question and associated answer.

Thread local_server:

- This thread represents a local DNS server that listens for DNS queries.
- It creates a socket, binds it to the specified port, and starts listening for incoming queries.
- It receives a DNS query, parses it, and checks if the answer is present in the cache.
- If the answer is found in the cache, it retrieves the answer from the cache and sends it back to the client.
- If the answer is not found in the cache, it sends the query to the next DNS server and receives the response.
- It processes the response, retrieves the answer, adds it to the cache, and sends it back to the client.

Thread tier_server:

- This thread represents a DNS server for a specific DNS tier.
- It reads the resource records (RRs) from a file specific to the tier.
- It creates a socket, binds it to the specified port, and starts listening for incoming TCP DNS queries.
- It receives a TCP DNS query, processes it, and sends the response back to the client.

5. Results

Environments

```
OS: Ubuntu 22.04.2 LTS on Windows 10 x86_64
Kernel: 5.15.90.1-microsoft-standard-WSL2
GCC: 11.3.0 (Ubuntu 11.3.0-1ubuntu1~22.04.1)
CMake, Makefile
```

Build

In the project directory, run:

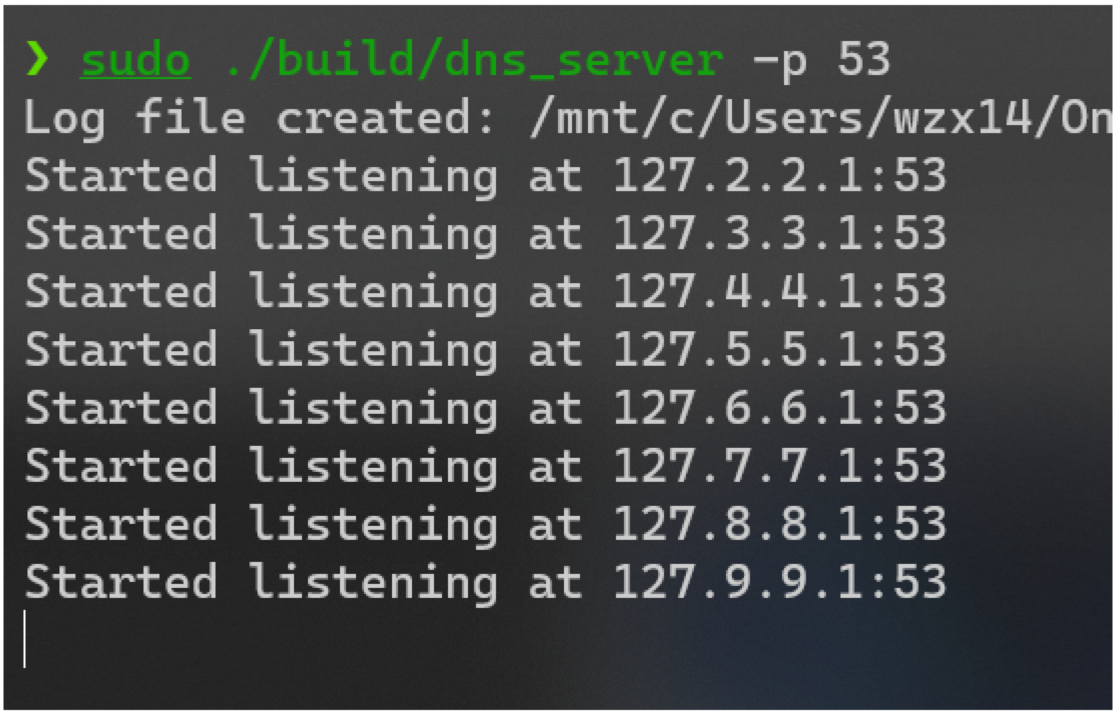
```
cmake . &&make
```

After that you can find the binaries in the `build` folder.

Run

- Start the server:

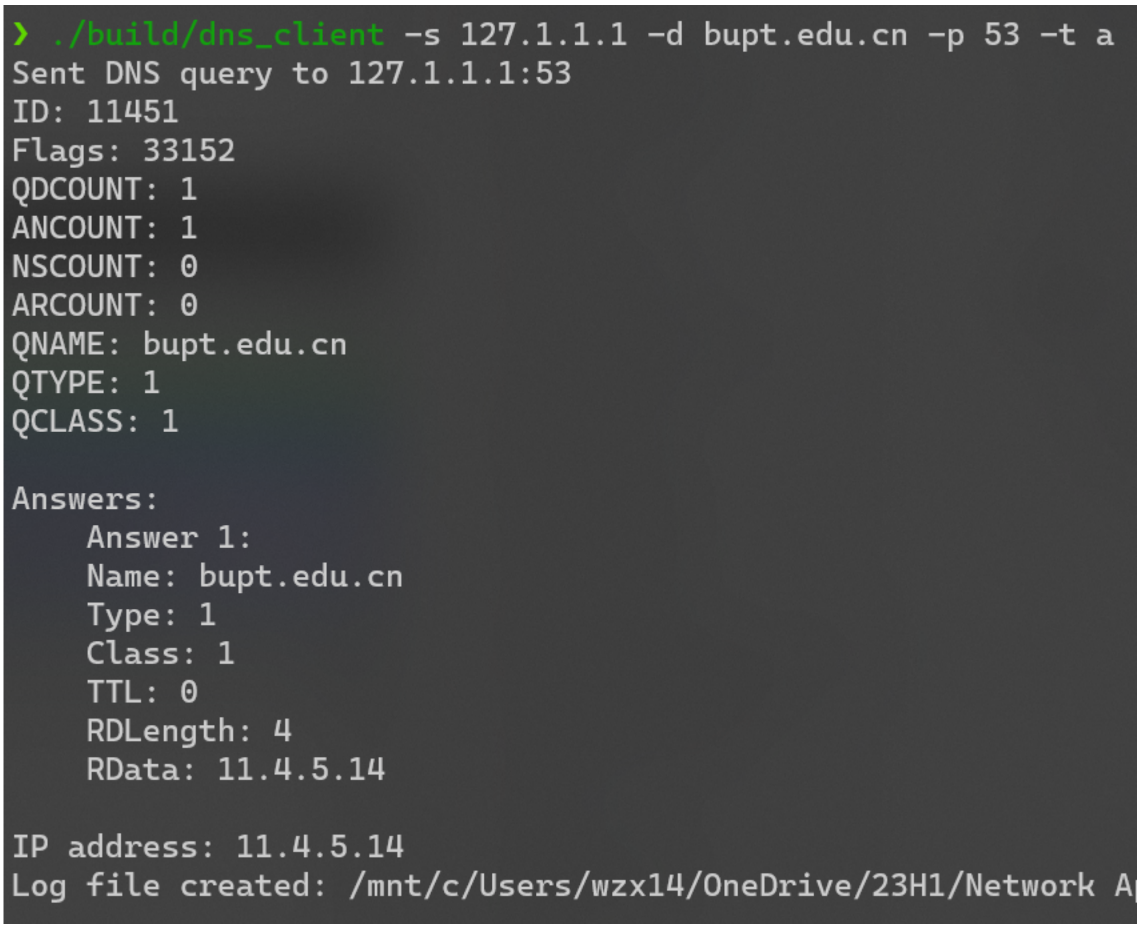
  Run `sudo ./build/dns_server -p 53`



```
❯ sudo ./build/dns_server -p 53
Log file created: /mnt/c/Users/wzx14/On
Started listening at 127.2.2.1:53
Started listening at 127.3.3.1:53
Started listening at 127.4.4.1:53
Started listening at 127.5.5.1:53
Started listening at 127.6.6.1:53
Started listening at 127.7.7.1:53
Started listening at 127.8.8.1:53
Started listening at 127.9.9.1:53
```

- A record:

  Run `./build/dns_client -s 127.1.1.1 -d bupt.edu.cn -p 53 -t a`

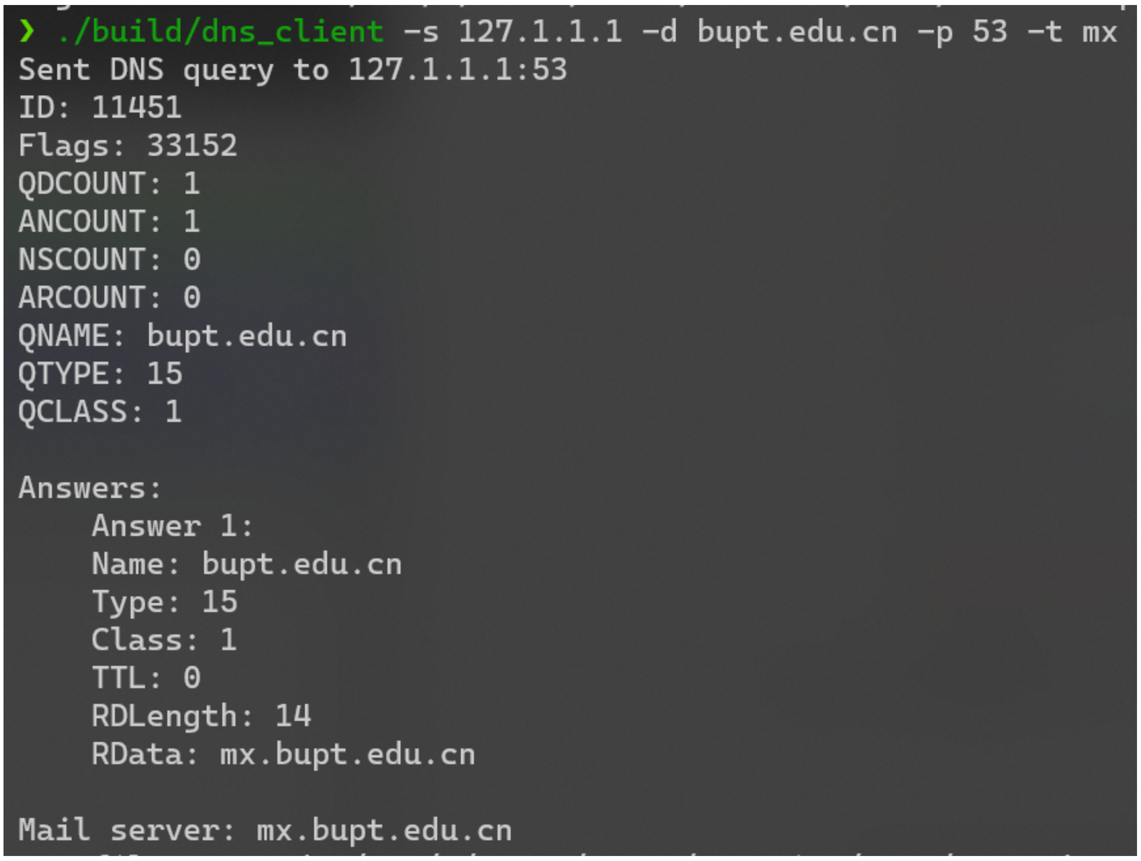

```
❯ ./build/dns_client -s 127.1.1.1 -d bupt.edu.cn -p 53 -t a
Sent DNS query to 127.1.1.1:53
ID: 11451
Flags: 33152
QDCOUNT: 1
ANCOUNT: 1
NSCOUNT: 0
ARCOUNT: 0
QNAME: bupt.edu.cn
QTYPE: 1
QCLASS: 1

Answers:
    Answer 1:
    Name: bupt.edu.cn
    Type: 1
    Class: 1
    TTL: 0
    RDLength: 4
    RData: 11.4.5.14

IP address: 11.4.5.14
Log file created: /mnt/c/Users/wzx14/OneDrive/23H1/Network A
```

- MX record:

  Run `./build/dns_client -s 127.1.1.1 -d bupt.edu.cn -p 53 -t mx`

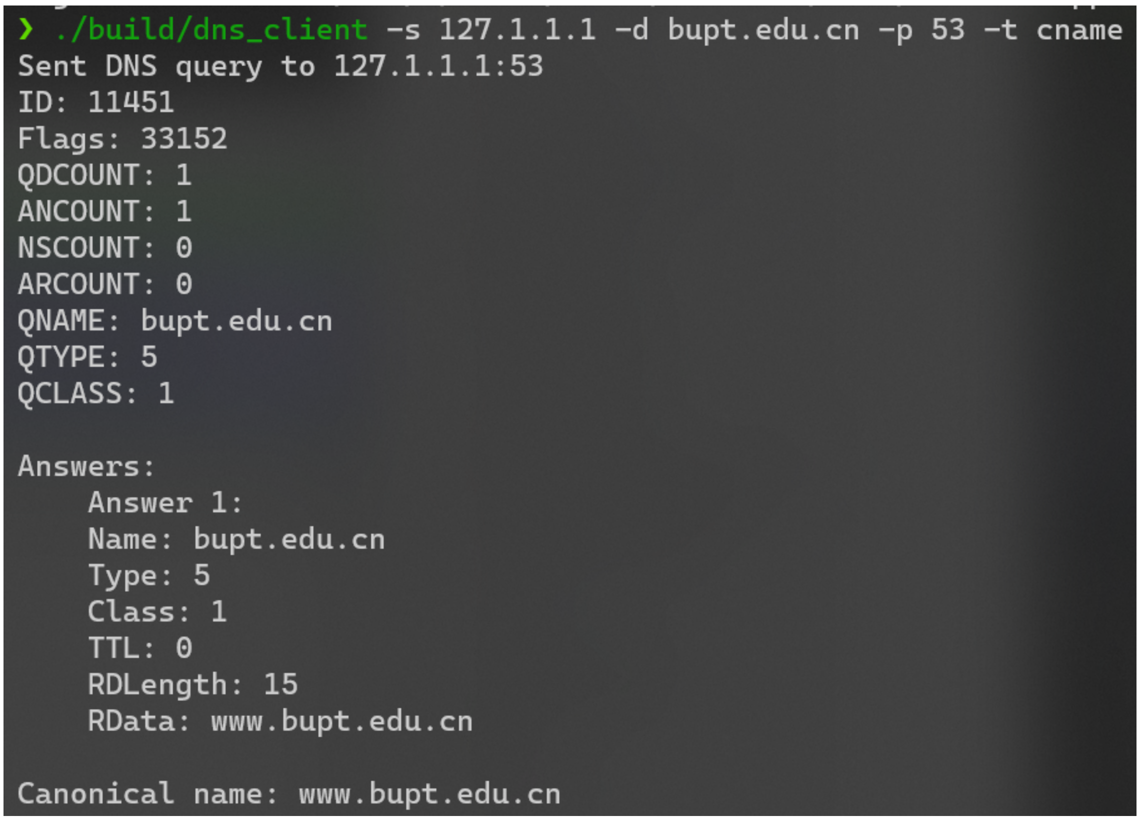

```
❯ ./build/dns_client -s 127.1.1.1 -d bupt.edu.cn -p 53 -t mx
Sent DNS query to 127.1.1.1:53
ID: 11451
Flags: 33152
QDCOUNT: 1
ANCOUNT: 1
NSCOUNT: 0
ARCOUNT: 0
QNAME: bupt.edu.cn
QTYPE: 15
QCLASS: 1

Answers:
    Answer 1:
    Name: bupt.edu.cn
    Type: 15
    Class: 1
    TTL: 0
    RDLength: 14
    RData: mx.bupt.edu.cn

Mail server: mx.bupt.edu.cn
```

- CNAME record:

  Run `./build/dns_client -s 127.1.1.1 -d bupt.edu.cn -p 53 -t cname`



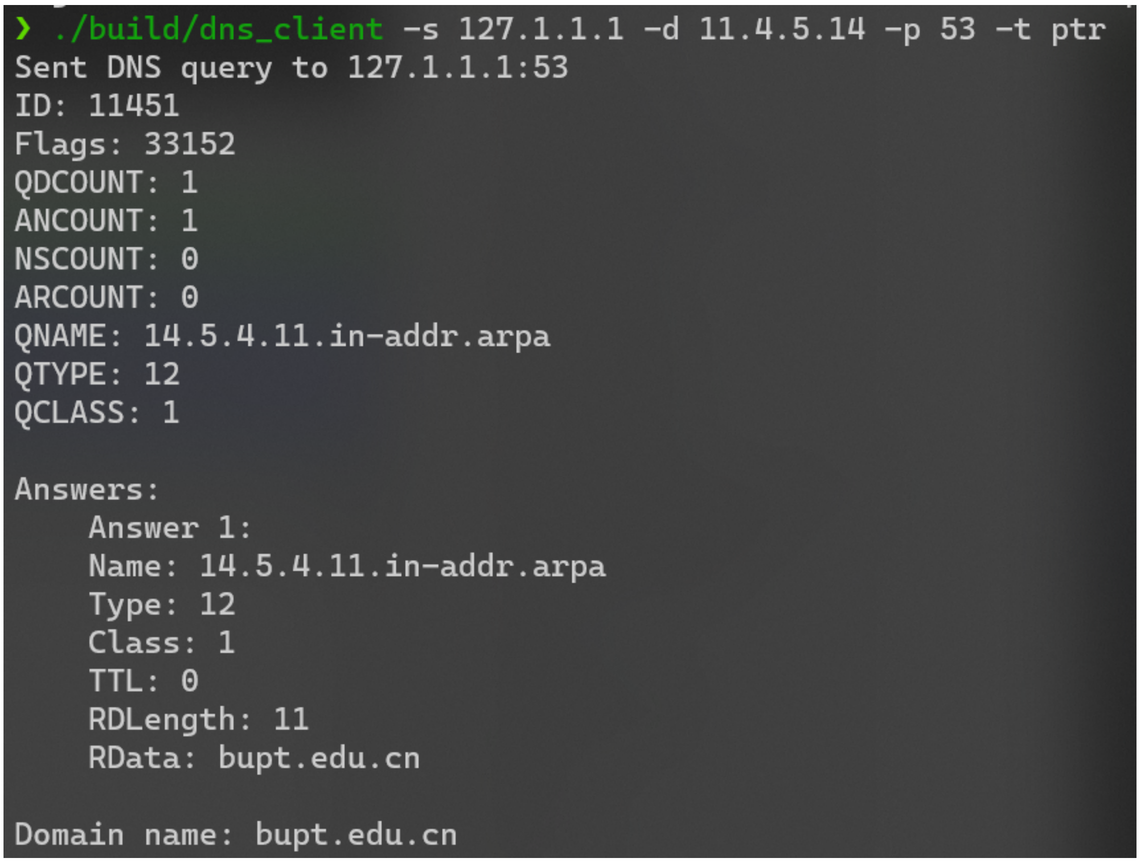
```
❯ ./build/dns_client -s 127.1.1.1 -d bupt.edu.cn -p 53 -t cname
Sent DNS query to 127.1.1.1:53
ID: 11451
Flags: 33152
QDCOUNT: 1
ANCOUNT: 1
NSCOUNT: 0
ARCOUNT: 0
QNAME: bupt.edu.cn
QTYPE: 5
QCLASS: 1

Answers:
    Answer 1:
    Name: bupt.edu.cn
    Type: 5
    Class: 1
    TTL: 0
    RDLength: 15
    RData: www.bupt.edu.cn

Canonical name: www.bupt.edu.cn
```

- PTR record:

  Run `./build/dns_client -s 127.1.1.1 -d 11.4.5.14 -p 53 -t ptr`



```
❯ ./build/dns_client -s 127.1.1.1 -d 11.4.5.14 -p 53 -t ptr
Sent DNS query to 127.1.1.1:53
ID: 11451
Flags: 33152
QDCOUNT: 1
ANCOUNT: 1
NSCOUNT: 0
ARCOUNT: 0
QNAME: 14.5.4.11.in-addr.arpa
QTYPE: 12
QCLASS: 1

Answers:
    Answer 1:
    Name: 14.5.4.11.in-addr.arpa
    Type: 12
    Class: 1
    TTL: 0
    RDLength: 11
    RData: bupt.edu.cn

Domain name: bupt.edu.cn
```

- Cache:

  Resend the query, will find `2023-06-10 17:16:34 Found in cache: bupt.edu.cn` in server logs. Local DNS server will load the response from its cache.

- Trace:

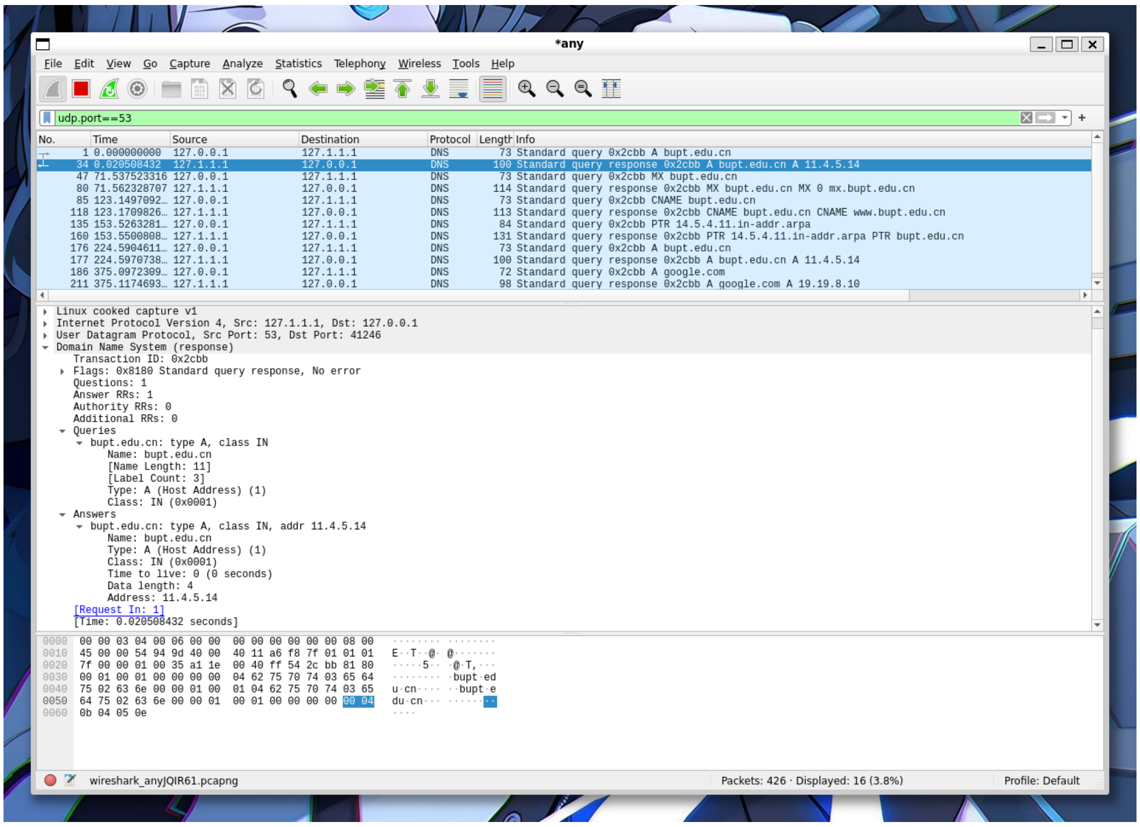
  In server logs:

```
2023-06-10 17:12:50 Received DNS query: bupt.edu.cn
2023-06-10 17:12:50 Received DNS query: bupt.edu.cn, sent DNS response: -
>127.3.3.1
2023-06-10 17:12:50 Received DNS query: bupt.edu.cn, sent DNS response: cn-
>127.4.4.1
2023-06-10 17:12:50 Received DNS query: bupt.edu.cn, sent DNS response:
edu.cn->127.5.5.1
2023-06-10 17:12:50 Received DNS query: bupt.edu.cn, sent DNS response:
bupt.edu.cn->11.4.5.14
2023-06-10 17:12:50 Found the IP address: 11.4.5.14

2023-06-10 17:19:05 Received DNS query: google.com
2023-06-10 17:19:05 Received DNS query: google.com, sent DNS response: -
>127.3.3.1
2023-06-10 17:19:05 Received DNS query: google.com, sent DNS response: com-
>127.7.7.1
2023-06-10 17:19:05 Received DNS query: google.com, sent DNS response:
google.com->19.19.8.10
2023-06-10 17:19:05 Found the IP address: 19.19.8.10
```

- Wireshark capture example:

  Run `sudo wireshark`, choose `any`


*any
File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help

udp.port==53

```
No.  Time          Source     Destination  Protocol Length Info
  1 0.000000000   127.0.0.1  127.1.1.1    DNS       73 Standard query 0x2cbb A bupt.edu.cn
 34 0.020508432   127.1.1.1  127.0.0.1    DNS      100 Standard query response 0x2cbb A bupt.edu.cn A 11.4.5.14
 47 71.537523316  127.0.0.1  127.1.1.1    DNS       73 Standard query 0x2cbb MX bupt.edu.cn
 80 71.562328707  127.1.1.1  127.0.0.1    DNS      114 Standard query response 0x2cbb MX bupt.edu.cn MX 0 mx.bupt.edu.cn
 85 123.1497092…  127.0.0.1  127.1.1.1    DNS       73 Standard query 0x2cbb CNAME bupt.edu.cn
118 123.1709826…  127.1.1.1  127.0.0.1    DNS      113 Standard query response 0x2cbb CNAME bupt.edu.cn CNAME www.bupt.edu.cn
135 153.5263281…  127.0.0.1  127.1.1.1    DNS       84 Standard query 0x2cbb PTR 14.5.4.11.in-addr.arpa
160 153.5500808…  127.1.1.1  127.0.0.1    DNS      131 Standard query response 0x2cbb PTR 14.5.4.11.in-addr.arpa PTR bupt.edu.cn
176 224.5904611…  127.0.0.1  127.1.1.1    DNS       73 Standard query 0x2cbb A bupt.edu.cn
177 224.5970738…  127.1.1.1  127.0.0.1    DNS      100 Standard query response 0x2cbb A bupt.edu.cn A 11.4.5.14
186 375.0972309…  127.0.0.1  127.1.1.1    DNS       72 Standard query 0x2cbb A google.com
211 375.1174693…  127.1.1.1  127.0.0.1    DNS       98 Standard query response 0x2cbb A google.com A 19.19.8.10
```

```
Linux cooked capture v1
Internet Protocol Version 4, Src: 127.1.1.1, Dst: 127.0.0.1
User Datagram Protocol, Src Port: 53, Dst Port: 41246
Domain Name System (response)
    Transaction ID: 0x2cbb
    Flags: 0x8180 Standard query response, No error
    Questions: 1
    Answer RRs: 1
    Authority RRs: 0
    Additional RRs: 0
    Queries
        bupt.edu.cn: type A, class IN
            Name: bupt.edu.cn
            [Name Length: 11]
            [Label Count: 3]
            Type: A (Host Address) (1)
            Class: IN (0x0001)
    Answers
        bupt.edu.cn: type A, class IN, addr 11.4.5.14
            Name: bupt.edu.cn
            Type: A (Host Address) (1)
            Class: IN (0x0001)
            Time to live: 0 (0 seconds)
            Data length: 4
            Address: 11.4.5.14
    [Request In: 1]
    [Time: 0.020508432 seconds]
```

```
0000  00 00 03 04 00 06 00 00  00 00 00 00 00 00 08 00   ········ ········
0010  45 00 00 54 94 9d 40 00  40 11 a6 f8 7f 01 01 01   E··T··@· @·······
0020  7f 00 00 01 00 35 a1 1e  00 40 ff 54 2c bb 81 80   ·····5·· ·@·T,···
0030  00 01 00 01 00 00 00 00  04 62 75 70 74 03 65 64   ········ ·bupt·ed
0040  75 02 63 6e 00 00 01 00  01 04 62 75 70 74 03 65   u·cn···· ··bupt·e
0050  64 75 02 63 6e 00 00 01  00 01 00 00 00 00 00 04   du·cn··· ········
0060  0b 04 05 0e                                        ····
```

wireshark_anyJQIR61.pcapng   Packets: 426 · Displayed: 16 (3.8%)   Profile: Default

6. Summary and Conclusion

In summary, the detailed design of the DNS system includes components such as the Client, Server, and Utils, each with specific functionalities. The Client component handles sending DNS queries to the Local DNS Server, while the Server component consists of the Local DNS Server, TLD DNS Server, Intermediate DNS Server, and Root DNS Server, responsible for receiving and resolving DNS queries. The Utils component provides utility functions for DNS message handling, resource record processing, socket communication, and general utilities.

The design follows a modular approach, separating different components into their respective directories. This allows for easier code maintenance, reusability, and collaboration between team members. The design also includes structures, constants, function declarations, and thread descriptions for the various components.

Overall, the detailed design provides a clear understanding of the system's architecture, functionality, and interactions between different components. It serves as a blueprint for implementing the DNS system and can guide the development process.

In conclusion, the detailed design of the DNS system demonstrates a well-thought-out architecture that encompasses the necessary components and functionalities for DNS resolution. By following this design, developers can build and deploy an effective and reliable DNS system that handles queries, resolves domain names, and caches responses for improved performance.

Challenges

During the development of our project, we also encountered some challenges. One of the main difficulties was the complexity of implementing DNS packets according to the DNS standard. This task required us to carefully follow the specifications and guidelines of the DNS protocol to ensure the correct structure and format of the packets.

The DNS protocol has a specific format for message headers, question sections, answer sections, and additional sections. Each section has its own fields and rules that need to be correctly implemented. Even a small mistake in constructing or parsing the DNS packets could result in Wireshark flagging them as malformed packets. This required us to pay close attention to the details and ensure compliance with the DNS standard throughout the implementation process.

To overcome these challenges, we had to thoroughly study the DNS protocol and its specifications. We carefully implemented the various fields and sections of the DNS packets, ensuring their correct format and alignment. We also performed extensive testing and debugging to identify and fix any issues or errors in the packet construction and parsing.

Despite the challenges, successfully implementing the DNS packet structure according to the DNS standard was a valuable learning experience. It enhanced our understanding of network protocols and the importance of adhering to standards to ensure interoperability and compatibility. We also gained insights into the functioning of DNS systems and the complexities involved in DNS resolution.

In conclusion, while the implementation of DNS packets according to the DNS standard posed some challenges, we were able to overcome them through diligent research, careful implementation, and thorough testing. These challenges provided us with valuable insights and improved our skills in network protocol implementation.

Appendix

Source Code: t0saki/**Project-DNS**