

ISA - DNS RESOLVER

 **Skuratovich Aliaksandr**
4BIT student.
Brno University of Technologies
Brno, Božetěchova 1/2
xskura01@vutbr.cz

November 19, 2023

Contents

1	Introduction	2
2	Application Design	2
2.1	Problem Statement	2
2.2	How DNS Works	2
2.3	DNS Query Structure	2
2.4	DNS Response Structure	2
2.5	Design of the DNS Resolver Tool	2
2.6	dns.h	3
2.7	main.cpp	3
3	Implementation Description	3
3.1	Functional Programming	3
3.2	Memory Management	3
4	Testing Methodology	3
4.1	Automated Testing Framework	4
4.2	Test Script Structure	4
4.3	Conditional Server Selection	4
4.4	Modularity and Extensibility	4
4.5	Test Case Generation	4
4.6	Output and Logging	4

1 Introduction

The Domain Name System (DNS) is an essential component of the Internet's infrastructure, providing a way to resolve human-friendly domain names into IP addresses that are required for network communication. The DNS resolver tool developed in this project is a command-line application that sends DNS queries to a specified server and processes the response. This document serves as the technical documentation for the DNS resolver tool, detailing its design, implementation, and testing.

2 Application Design

The DNS resolver is a command-line application that interacts with the Domain Name System (DNS), converting domain names into IP addresses for network routing (and reversed). It operates within DNS's hierarchical structure, facilitating internet connectivity by translating names like `www.google.com` into addresses like `142.251.36.132`.

2.1 Problem Statement

DNS enables users to access websites using easy-to-remember domain names instead of complex IP addresses. This tool allows direct querying of DNS servers to retrieve domain information, bypassing operating system services or third-party apps. Also, it's needed to receive points for the project to get a bachelor's degree.

2.2 How DNS Works

DNS operates on a client-server model, where a browser (client) sends a domain name query to a DNS server. This server can resolve the query recursively by querying other servers until it finds the IP address, or iteratively by directing the client to other servers closer to the answer. Once resolved, the server returns the IP address to the client for website access.

DNS queries are typically sent using the User Datagram Protocol (UDP) due to its low latency. However, Transmission Control Protocol (TCP) can also be used in situations where the reliability of data transmission is crucial, such as when the response data size exceeds the UDP message size limit. In the application, only the former is supported.

2.3 DNS Query Structure

A DNS query consists of a header and a question section. The header contains fields such as the transaction ID, flags (indicating the type of query), and counters for the number of questions and answers. The question section contains the domain name being queried, the type of record being requested (such as A, AAAA, MX, etc.), and the class (usually IN for the internet) (1).

2.4 DNS Response Structure

A DNS response includes the same header as the query, with flags indicating the response status. The response also contains four sections: (1)

- **Question Section:** Echoes the query from the client.
- **Answer Section:** Contains resource records (RRs) for the queried domain name.
- **Authority Section:** Contains RRs pointing to authoritative name servers.
- **Additional Section:** Contains RRs with additional information, such as alternative addresses or services.

Each resource record in the response sections includes the domain name, record type, class, time-to-live (TTL), data length, and the resource data (such as the IP address).

2.5 Design of the DNS Resolver Tool

The DNS resolver tool is structured into several components, each handling a specific aspect of the DNS query process:

- **Argument Parser (`argparser.h`):** Parses and validates command-line arguments provided by the user, such as the DNS server address, port number, and the domain name to query.

- **DNS Packet Constructor** (`dns.h`): Constructs the DNS query packet based on the user's input, including setting the appropriate flags and question section.
- **UDP Communication** (`udp.h`): Handles the network communication, sending the constructed DNS query packet to the specified DNS server and receiving the response packet.
- **DNS Packet Parser** (`dns.h`): Parses the received DNS response packet, extracting information from the answer, authority, and additional sections.
- **Main Application Flow** (`main.cpp`): Orchestrates the overall process, from parsing arguments to constructing and sending the query, receiving the response, and displaying the results to the user.

2.6 dns.h

The `dns.h` header file is responsible for constructing and parsing DNS packets. It defines structures and constants for DNS record types, flags, and classes. It also includes functions for parsing different sections of the DNS response packet, such as the question, answer, authority, and additional sections.

2.7 main.cpp

The `main.cpp` file contains the `main()` function, which orchestrates the flow of the program. It performs the following steps:

1. Parses command-line arguments to configure the DNS query.
2. Constructs the DNS query packet.
3. Sends the query to the specified DNS server and waits for a response.
4. Parses the received DNS response packet.
5. Outputs the results to the user.

3 Implementation Description

The DNS resolver tool is implemented in C++ and utilizes the standard library for various functionalities, including string manipulation, network communication, and memory management. The program adheres to functional programming principles¹, aiming to minimize side effects and use immutable data structures where possible.

3.1 Functional Programming

The application avoids side effects by not modifying the global state or input arguments within functions. Instead, it returns new values to represent state changes. For example, the `dns::parsing::parseSection` function returns a `std::tuple` containing the parsed output and the new offset within the packet. Similarly, all the functions in `dns.h` that perform the parsing, return a `parserResult` tuple instead of modifying the variables. It could be considered less efficient, yet it's safer and more explicit.

3.2 Memory Management

The use of `std::unique_ptr` in the `udp::sendQuery` function ensures that resources are properly managed and released when no longer needed, preventing memory leaks.

4 Testing Methodology

Testing is a crucial part of software development, especially for network applications like the DNS resolver tool, where the program must interact correctly with external systems. The testing process for the DNS resolver tool is designed to be comprehensive, automated, and modular, ensuring that each component of the application functions as expected under various scenarios. I didn't do this. Instead, I've implemented tests in python. Another solution, if we consider a production environment, would be to implement unit tests for each function, but I decided not to overcomplicate things.

¹https://en.wikipedia.org/wiki/Functional_programming

4.1 Automated Testing Framework

The DNS resolver tool uses Python's `unittest` framework for automated testing, offering tools for assertions and test suite management. These tests ensure application correctness without manual intervention, enabling consistent and repeatable runs.

4.2 Test Script Structure

The test script, `test_dns.py`, is structured to cover a wide range of test cases, including:

- Valid DNS queries for both IPv4 and IPv6 addresses.
- Reverse DNS lookups for both IPv4 and IPv6 addresses.
- Handling of various DNS record types
- Invalid command-line arguments and error handling.
- DNS server response timeouts and error scenarios.

Each test case is defined as a function that constructs a command to run the DNS resolver tool with specific arguments. The script then executes the command as a subprocess, captures the output, and compares it against expected results.

4.3 Conditional Server Selection

The test script includes a conditional check for the hostname ending with `'fit.vutbr.cz'`, which is the domain for Brno University of Technology. If the script is run on a BUT server, it includes `'kazi.fit.vutbr.cz'` as one of the DNS servers to query.

4.4 Modularity and Extensibility

The tests are designed to be modular, allowing for easy addition, removal, or modification of individual test cases without affecting others.

4.5 Test Case Generation

The script generates test cases dynamically based on combinations of servers, addresses, and query types. This approach reduces code duplication and allows for an exhaustive testing of different scenarios. For example, the script generates tests for both iterative and recursive queries, as well as for standard and reverse lookups.

4.6 Output and Logging

Test results are logged to a file named `test_log_<timestamp>.log`, where `<timestamp>` is the current date and time.

References

- [1] MOCKAPETRIS, P. *Domain names - concepts and facilities* [<https://www.ietf.org/rfc/rfc1034.txt>]. November 1987. RFC 1034.