VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ FAKULTA INFORMAČNÍCH TECHNOLOGIÍ

DNS resolver Documentation

Version 0.0.1

Turytsia Oleksandr (xturyt00)

November 17, 2023

Contents

List of Code Listings

1 Introduction

This comprehensive documentation is designed to help you get started with DNS resolver. In addition it describes certain parts of implementation. Whether you're a first-time user or an experienced developer, this documentation provides the guidance you need to navigate and utilize a program effectively.

2 Getting Started

2.1 Prerequisites

Before using the DNS resolver, make sure you have the following prerequisites installed:

- GCC (Tested version 10.5.0)
- Unix-based system (Tested on FreeBSD)

2.2 Installation

All the source code is located at **src** folder. Run following command in order to compile the project:

```
$ make
```

After the project is compiled, you can run following command to run a quick test:

```
$ ./dns -r -s dns.google www.github.com
```

2.3 Usage

```
$ ./dns [-r] [-x] [-h] [-t] [-6] -s server [-p port] address
```

- **-r** = Recursion Desired (Recursion Desired = 1), otherwise no recursion.
- **-6** = Query type AAAA instead of the default A.
- **-x** = Reverse query instead of direct query. Note, user can specify here ipv6 or ipv4 **address** without specifying **-6** option to make a reverse query.

- **-s server** = IP address or domain name of the server to which the query should be sent. Note, user can specify **server** by its domain or ipv6 address.
- **-p port** = The port number to send the query to, default is set to 53.
- **-h help** = Display help info.
- **-t** = Enables testing mode (TTL is hidden).
- **address** = The address to be queried.

2.4 Output format

The program will construct dns query based on user's input and send packet over udp to a specified dns server. Dns response may have following format:

```
Authoritative: Yes/No, Recursive: Yes/No, Truncated: Yes/No
Question section (1)
 [ADDRESS], [QTYPE], [QCLASS]
Answer section (N)
 [ADDRESS], [TYPE], [CLASS], [TTL], [RDATA]
  ...
Authority section (N)
 [ADDRESS], [TYPE], [CLASS], [TTL], [RDATA]
  ...
Additional section (N)
 [ADDRESS], [TYPE], [CLASS], [TTL], [RDATA]
  ...
```

First line displays response header flags (RD, TC, AA) followed by request response (RR)[1] sections.

3 Implementation

This section provides an overview of the key components and functions in the DNS resolver implementation. It provides a high-level overview of the key implementation details in your DNS resolver code. It explains the structures, functions, and utilities used to create, send, and process DNS queries and responses. Developers and users of this code can refer to this section to understand how the DNS resolver works and how to interact with it.

[1] `https://datatracker.ietf.org/doc/html/rfc1035`

3.1 Structures

- **`dns_header_t`** = This structure represents the DNS header with fields for various flags, counts, and identifiers.
- **`dns_rr_t`** = Defines the format of a DNS resource record (RR) in the answer section, authority section, or additional section of DNS packets. It contains fields for the type, class, time-to-live (TTL), and RDATA length.
- **`dns_question_t`** = Represents the question section of DNS packets, including the query type and class.
- **`dns_soa_t`** = structure specifies the format of the Start of Authority (SOA) resource data (RDATA) in DNS packets. It includes fields for version number, refresh time, retry time, expiration time, and minimum TTL.

3.2 DNS Query Creation

The `create_dns_query` function constructs a DNS query packet based on the provided arguments. It sets various fields in the DNS header and question section, including query type and class (Code 1).

3.3 DNS Query Sending

The `send_dns_query` function creates a socket, sets a receive timeout to 5 seconds, sends a DNS query to a specified DNS server, and receives the response. The timeout feature helps ensure that the code does not wait indefinitely for a response (Code 2).

3.4 Response Processing

In the main function, the received DNS response is processed. The code extracts and interprets the DNS header, question section, answer section, authority section, and additional section, printing the relevant information (Code 3).

4 Limitations

While this DNS resolver program is capable of performing various DNS-related tasks, it comes with certain limitations that users should be aware of.

4.1 Supported Queries

The program is limited to querying only specific types such as A (IPv4 address), AAAA (IPv6 address), and PTR (pointer to a domain name). Queries for other types are not supported.

4.2 Supported Response Types

The program is designed to handle and interpret responses of types A, AAAA, PTR, CNAME, and SOA. Responses of other types are not supported. In case of getting response with other types, a user should see appropriate message.

5 Testing

5.1 Enviroment

The program was tested on FreeBSD in a VUT FIT University network using inputs and outputs of a **dig** command. Be aware that testing in another network will lead to a different outputs and therefore will fail.

5.2 Usage

DNS resolver can be tested using following command:

```
$ make test
```

All the tests are located in a folder **tests**. There is a **test.sh** script that takes inputs from the folder with tests (.in files) and compares them to corresponding output (.out files).

Each test enables testing mode for the DNS resolver, which basically hides attributes with changing values (for example TTL) in order to make testing consistent.

6 Error codes

DNS resolver is also suitable to be used as a script, because it provides distinctive exit error codes, which can help programmers validate its results.

- Arguments parser error codes:
  - **1** = Unknown option

  - **2** = Invalid port
  - **3** = Port is missing
  - **4** = Source address is missing
  - **5** = Target address is missing
  - **6** = Option already specified

- Sending query error codes:
  - **10** = Socket creation error
  - **11** = Sending query error
  - **12** = Timeout
  - **13** = Receiving response error

- DNS header error codes (By subtracting 30 it is possible to get error codes that correspond RFC 1035 documentation):
  - **31** = Format error
  - **32** = Server fail
  - **33** = Name error
  - **34** = Not implemented
  - **35** = Refused

- Other errors:
  - **20** = perror code
  - **21** = other errors
  - **22** = Family is not supported

```c
void create_dns_query(args_t* args, unsigned char* query) {
    dns_header_t dns_header = {
        .id = htons(getpid()),       // Set the ID to X
        .qr = 0,                     // Query
        .opcode = 0,                 // Standard query
        .aa = 0,                     // Authoritative
        .tc = 0,                     // Truncated
        .rd = args->recursive,       // Recursion Desired
        .ra = 0,                     // Recursion Available
        .z = 0,                      // Reserved
        .cd = 0,
        .ad = 0,
        .rcode = 0,                  // Response code
        .qdcount = htons(1),         // Number of questions
        .ancount = 0,                // Number of answers
        .nscount = 0,                // Number of authority records
        .arcount = 0                 // Number of additional records
    };
    memcpy(query, &dns_header, sizeof(dns_header_t));

    unsigned char* qname = (unsigned char*)(query + sizeof(dns_header_t));
    unsigned char qbuffer[MAX_BUFF] = {0};

    if (args->reverse) {
        if (is_ipv4(args->target_addr)) {
            reverse_dns_ipv4((char*)qbuffer, args->target_addr);
        }
        else {
            reverse_dns_ipv6((char*)qbuffer, args->target_addr);
        }
    }

    compress_domain_name(qname, (char*)qbuffer);

    int len = strlen((char*)qname);

    dns_question_t* qinfo = (dns_question_t*)(qname + len + 1);

    qinfo->qtype = htons(args->ipv6 ? AAAA : args->reverse ? PTR : A);
    qinfo->qclass = htons(1);
}
```

Listing 1: `create_dns_query` function implementation

```c
send_query_err_t send_dns_query(args_t* args,  unsigned char* buffer,
  unsigned char* query, char* addr, int qlen) {
    int sockt;
    int err;
    socklen_t addr_len;
    ssize_t bytes_received;

    struct sockaddr_storage server;

    memset(&server, 0, sizeof(struct sockaddr_storage));

    if (ai_family == AF_INET) {
        // set up for ipv4
    }
    else {
        // set up for ipv6
    }

    sockt = socket(ai_family, SOCK_DGRAM, IPPROTO_UDP);
    if (sockt == -1) {
        // error
    }

    // Set the timeout in seconds
    struct timeval timeout;
    timeout.tv_sec = 5;
    timeout.tv_usec = 0;
    setsockopt(sockt, SOL_SOCKET, SO_RCVTIMEO, &timeout, sizeof(timeout));

    err = sendto(sockt, query, qlen, 0, (struct sockaddr*)&server,
      sizeof(struct sockaddr_storage));
    if (err == -1) {
        // error
    }

    addr_len = sizeof server;
    bytes_received = recvfrom(sockt, buffer, MAX_BUFF, 0,
      (struct sockaddr*)&server, &addr_len);
    if (bytes_received == -1) {
        // error
    }

    close(sockt);
    return 0;
}
```

Listing 2: send_dns_query function implementation

```c
void print_rr(unsigned char* pointer, unsigned char* buffer,
  int n, int is_test) {
    for (int i = 0; i < n; i++) {
        char name[MAX_NAME] = { 0 };

        parse_domain_name(pointer, buffer, name);

        pointer += get_name_length(pointer, name);

        dns_rr_t* dns_rr = (dns_rr_t*)(pointer);

        unsigned short rr_type = ntohs(dns_rr->type);
        unsigned short rr_class = ntohs(dns_rr->class);
        unsigned int rr_ttl = ntohl(dns_rr->ttl);
        unsigned short rr_rdlength = ntohs(dns_rr->rdlength);

        printf(" %s, %s, %s, %d, ", name, get_dns_type(rr_type),
                    get_dns_class(rr_class), is_test ? 0 : rr_ttl);

        switch (rr_type) {
            case A:
                print_ipv4_data(pointer);
                break;
            case CNAME:
            case PTR:
                print_domain_name_data(pointer, buffer);
                break;
            case AAAA:
                print_ipv6_data(pointer);
                break;
            case SOA:
                print_soa_data(((unsigned char*)dns_rr +
                 sizeof(dns_rr_t)), buffer);
                break;
            default:
                printf("%s is not supported yet.\n", get_dns_type(rr_type));
        }

        pointer +=  sizeof(dns_rr_t) + rr_rdlength;
    }
}
```

Listing 3: Parsing dns response