ISA – DNS resolver

Marek Kozumplík
xkozum08

November 20th 2023

Contents

1 Introduction

The purpose of this application is to send DNS queries to specified DNS servers. After receiving answer from the server, the application will parse the answer and output the information in this format:

```
$./dns -r -s 8.8.8.8 example.com
Authoritative: No, Recursive: Yes, Truncated: No
Question section (1)
  example.com., Type: A, Class: IN
Answer section (1)
  example.com., Type: A, Class: IN, TTL: 6044, Data length: 4, 93.184.216.34
Authority section (0)
Additional section (0)
```

```
%./dns www.github.com  -s kazi.fit.vutbr.cz
Authoritative: No, Recursive: No, Truncated: No
Question section (1)
  www.github.com., Type: A, Class: IN
Answer section (1)
  www.github.com., Type: CNAME, Class: IN, TTL: 1792, Data length: 2, github.com.
Authority section (8)
  github.com., Type:NS, Class: IN, TTL: 134002, Data length: 20, dns3.p08.nsone.net.
  github.com., Type:NS, Class: IN, TTL: 134002, Data length: 25, ns-1707.awsdns-21.co.uk.
  github.com., Type:NS, Class: IN, TTL: 134002, Data length: 22, ns-421.awsdns-52.com.
  github.com., Type:NS, Class: IN, TTL: 134002, Data length: 23, ns-1283.awsdns-32.org.
  github.com., Type:NS, Class: IN, TTL: 134002, Data length: 7, dns4sonenet.
  github.com., Type:NS, Class: IN, TTL: 134002, Data length: 7, dns2sonenet.
  github.com., Type:NS, Class: IN, TTL: 134002, Data length: 7, dns1sonenet.
  github.com., Type:NS, Class: IN, TTL: 134002, Data length: 22, ns-520.awsdns-01.net.
Additional section (1)
  ns-421.awsdns-52.com., Type: A, Class: IN, TTL: 134002, Data length: 4, 205.251.193.165
```

1.1 How to use

`./dns [-r] [-x] [-6] -s server [-p port] address` Where:

- -r : Recursion desired
- -x : Reverse query
- -6 : AAAA query instead of A
- -s : IP address or domain name of the DNS server
- -p : port (default is 53)
- address : requested address (or domain name if -x)
- -h: prints help

2 Implementation

Queries can be sent to DNS servers using their IPv4/IPv6 address or domain name. AAAA query is possible using -6 argument.

Reverse queries are possible using -x and IP address. Reverse queries for IPv6 address are also possible. In case of reverse query, the program ignores -6 parameter (because -6 expects AAAA query but we need PTR) and checks if the address is IPv4 or IPv6 using regex, then it sends the correct query. The IPv6 address must be full.

The program can handle A, AAAA, CNAME, NS, PTR type of answers/questions. It will also print all (compressed) names, IPv4 and IPv6 addresses.

The argument order does not matter if `POSIXLY_CORRECT` is set to 1. Otherwise the address parameter must be the last one.

2.1 Basic information

The application is written in C++ programming language using standard libraries. I used functions `socket()`, `sendto()` and `recvfrom()` to send UDP packets that contain DNS query. The `dns_header`, `dns_question` and `dns_answer` structs are based on RFC 1035. The DNS resolver is divided into `dns.cpp, arg_parser.cpp, encoder.cpp, printer.cpp` and `.hpp`.

The main function in `dns.cpp` first calls `arg_parser` functions to parse input arguments. Then a buffer is created and the `dns_header structure` is stored into the buffer with values based on the parsed arguments.

2.2 Converting to DNS format

Then `encoder.cpp` functions are called to convert the domain name or IP address to the dns format:

```
void convert_domain_to_dns(char *hostname, unsigned char *result)
{
    int last_dot_pos = 0;
    int char_cnt = 0;
    int i = 0;
    while(hostname[i] != '\0')
    {
        if (hostname[i] == '.')
        {
            result[last_dot_pos] = char_cnt;
            char_cnt = 0;
            last_dot_pos = i + 1;
        }
        else
        {
            char_cnt += 1;
            result[i + 1] = hostname[i];
        }
        i++;
    }
    result[last_dot_pos] = char_cnt;
    result[i + 1] = '\0';
}
```

These functions will store the formatted address into the buffer. Then the message will be sent to the server and after receiving the answer, the `printer.cpp` functions will be called to decode and print the desired output. If `rcode` of the answer is not 0, the application will exit with error.

2.3 Compression

The `printer.cpp` functions can also handle compression of names. I found out that even only part of names can be compressed.
For example:
`name 2` can have address of `name 1` after `cccc` instead of the whole name
`name 1: bbbb.aaaa`
`name 2: cccc.bbbb.aaaa`
This means I have to check for compression every byte. If the function finds compression bytes, it gets the offset from them and calls itself recursively with the offset.

```
bool is_name_compressed(unsigned char *name)
{
  return (name[0] == 0b11000000);
}

int get_compressed_offset(unsigned char *name)
{
  return ((name[0] & 0x3F) << 8) + name[1];
}
void print_domain(unsigned char buf[65536], unsigned char *buf_pointer)
{
  if (is_name_compressed(buf_pointer))
  {
    buf_pointer = &buf[get_compressed_offset(buf_pointer)];
    print_domain(buf, buf_pointer);
    return;
  }
  int i = 1;
  int letters = buf_pointer[0];
  while (buf_pointer[i] != '\0')
  {
    if (is_name_compressed(&buf_pointer[i]))
    {
      buf_pointer = &buf[get_compressed_offset(&buf_pointer[i])];
      print_domain(buf, &buf_pointer[i]);
      return;
    }
    if (letters == 0)
    {
      std::cout << '.';
      letters = buf_pointer[i];
    }
    else
    {
      std::cout << buf_pointer[i];
      letters--;
    }
    i++;
  }
  std::cout << '.';
 }
```

3 Testing

I made Python script that takes arguments from `.in` files in folder `tests`, runs the `./dns` with the arguments and compares the output with `.out` files. I ignore the TTL number because it changes every second. I used `dig` and `Wireshark` to create the output for different test cases. `POSIXLY_CORRECT` should be set to 1 when running the tests.

The tests can be run using `make test`.

Not everything can be tested using these automated tests because the output from the server changes frequently for some test cases. I tested these cases manually.

I also used valgrind with these test cases to check for memory leaks and errors.

```
valgrind --leak-check=full ./dns ...
```

4 Sources

- RFC 1035
  Information on DNS servers, resolvers, queries, DNS header format, format of DNS question and answer
- RFC 3596 - DNS Extensions to Support IP Version 6
- Whatsmydns.net - Reverse DNS generator
  Tutorial on how to create reverse DNS query for IPv4 and IPv6
- Binarytides.com, Silver Moon, May 18, 2020 - DNS Query Code in C with Linux sockets
  Example of how to send DNS query using socket, sendto, recvfrom. Example of struct of DNS header based on RFC1035
- Dataquest.io, September 6, 2022 - Python Subprocess: The Simple Beginner's Tutorial (2023)
  Subprocess in python for testing
- GeeksforGeeks, Aakash_Pancha, 10 Nov, 2021 - std::setbase, std::setw , std::setfill in C++
  For printing IPv6 address in a nice format
- GeeksforGeeks, abhishek_padghan, 15 Feb, 2023 - How to validate an IP address using ReGex