BRNO UNIVERSITY OF TECHNOLOGY
FACULTY OF INFORMATION TECHNOLOGY

ISA project documentation

DNS resolver

October 22, 2023 Aleksandr Shevchenko (xshevc01)

Contents

1 Problematic

The main purpose of DNS, or the Domain Name System, is translating human readable domain names (for example, www.fit.vut.cz) to machine readable IP addresses. There are 2 basic concepts referring to servers (groups of servers) that are integral to the DNS infrastructure, but each performs a different role and lives in different locations inside the pipeline of a DNS query:

- **Authoritative DNS** is a server that actually holds and is responsible for DNS resource records. Authoritative DNS has the final authority over a domain and is responsible for providing answers to recursive DNS servers with the IP address information.
- **Recursive DNS** responds to a recursive request from a client and takes the time to track down the DNS record. If a recursive DNS has the DNS reference cached, or stored for a period of time, then it answers the DNS query by providing the source or IP information. If not, it makes a series of requests until it reaches the authoritative DNS nameserver for the requested record (or times out or returns an error if no record is found)[1, 2, 4].

2 Application design

DNS program is able to send queries to DNS servers and to print out received responses to the standard output in a readable form. The main stages of the program are:

- Connection establishment with specified server.
- Creating a DNS query.
- Sending the DNS query.
- Receiving a response from the server.
- Parsing the response.

This program is all about correct processing DNS packets and network support. In order to understand the details of each of these parts (which is described in 3), it is necessary to understand the structure of the DNS packet [3].

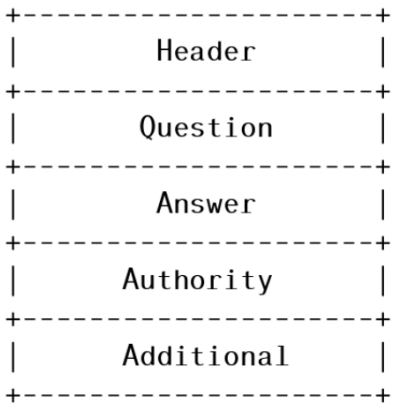

Figure 1: DNS packet structure

2.1 DNS Header

DNS Header is used to carry essential information about DNS messages. The DNS header is 12 bytes in length and consists of the following fields:

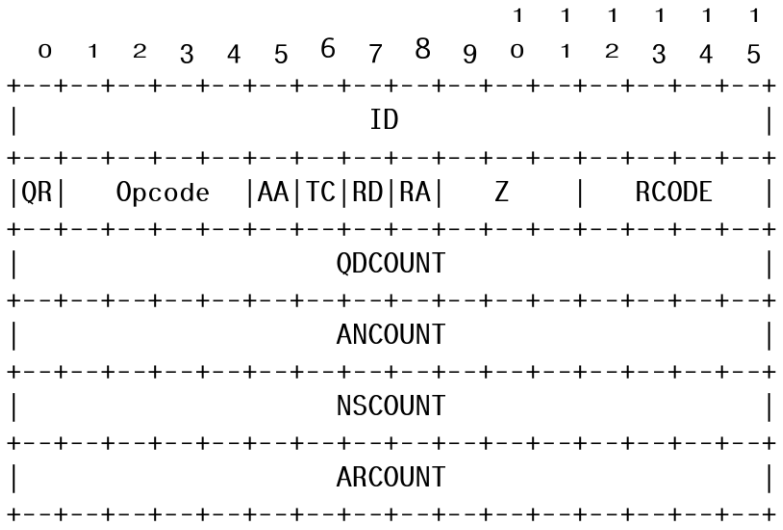

Figure 2: DNS Header structure

- `ID` (16 bits): a unique identifier for the DNS message.
- Flags (16 bits):
  - `QR` (1 bit): is set to 0 for queries and 1 for responses.
  - `Opcode` (4 bits): specifies the type of query (0 for standard query).
  - `AA` (1 bit): set to 1 in responses from authoritative name servers.
  - `TC` (1 bit): set to 1 if the response is truncated due to its size.
  - `RD` (1 bit): set to 1 if the client wants the server to perform recursive resolution.
  - `RA` (1 bit): set to 1 if the server supports recursion.
  - `Z` (3 bits): reserved for future use. On new machines contains of `Z`, `AD` and `CD` (used in DNSSEC)[6].
  - `RCODE` (4 bits): indicates the outcome of the query (0 for no error condition).
- `QDCOUNT` (16 bits): the number of entries in the question section of the message.
- `ANCOUNT` (16 bits): the number of resource records in the answer section of the message.
- `NSCOUNT` (16 bits): the number of name server resource records in the authority records section.
- `ARCOUNT` (16 bits): the number of resource records in the additional records section.

2.2 DNS Question

DNS Question provides details about what the client wants to know from the DNS server. A DNS question consists of the following components:

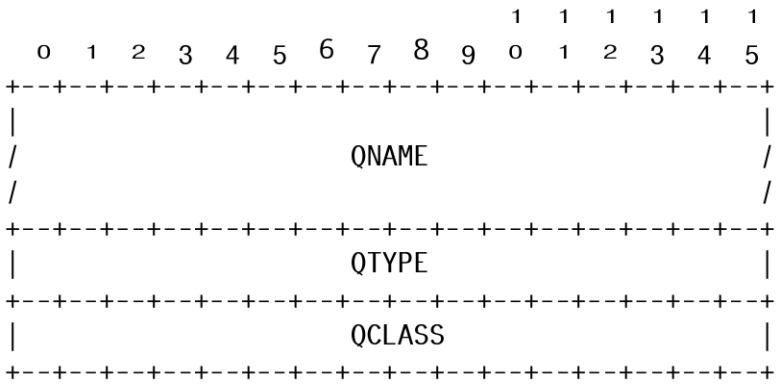

Figure 3: DNS Question structure

- QNAME: a domain name represented as a sequence of labels, where each label consists of a length octet followed by that number of octets.
- QTYPE (16 bits): specifies the type of the query. Common QTYPE values include:
  - A (1): looking for the IPv4 address associated with the domain name.
  - AAAA (28): looking for the IPv6 address associated with the domain name.
  - PTR (12): looking for a reverse DNS record associated with an IP address.
  - CNAME (5): looking for a canonical name, an alias for a domain.
- QCLASS (16 bits): specifies the class of the query, indicating the protocol family. The most common QCLASS value is IN (1): indicates the Internet class.

2.3 DNS Answer

DNS Answer provides information in response to the query made by the DNS client.

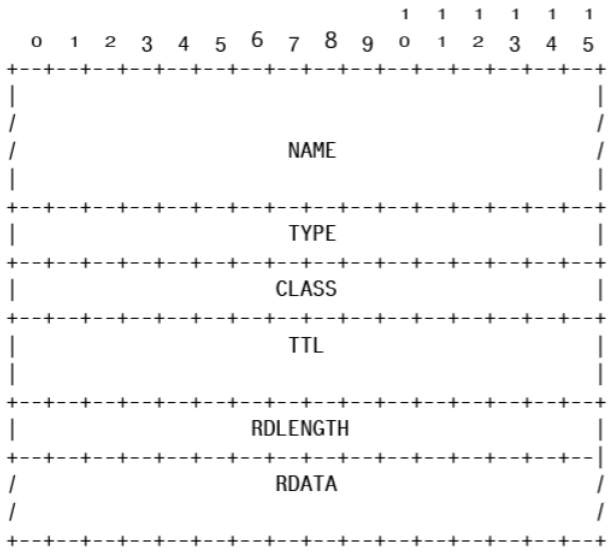

Figure 4: DNS Answer structure

- NAME: identifies the domain name to which the answer corresponds.
- TYPE (16 bits): specifies the type of the resource record contained in the answer. Common values are the same as (3)
- CLASS (16 bits): specifies the class of the resource record. It is typically set to IN (Internet class) for most DNS answers.
- TTL (32 bits): indicates the amount of time, in seconds, that the information provided in the DNS answer is considered valid.
- RDLENGTH (16 bits): specifies the length, in bytes, of the RDATA field that follows.
- RDATA: contains the data associated with the resource record. Its content depends on the TYPE of the resource record. For example, if the TYPE is A, RDATA contains the IPv4 address; if the TYPE is CNAME, RDATA contains the canonical name.

2.4 DNS Authority

DNS Authority contains information about authoritative DNS servers for the domain in question. The structure is similar to DNS Answer (4), but RDATA in the Authority Section contains domain names of authoritative name servers, which are responsible for providing authoritative DNS information for the domain.

Furthermore, the Authority Section typically contains resource records of two specific types: NS and SOA.

2.5 DNS Additional

DNS Additional is used to provide additional information and data that may be relevant to the DNS query. The structure is similar to DNS Answer (4), but the primary purpose of Additional Section is to include extra resource records.

3 Implementation

3.1 Project structure

Files	Description
dns.c	Main body of DNS program
error.c/h	Printing errors and warnings
network.c/h	Creating a socket and sending / receiving packets
parser.c/h	Argument parser, including constants from the task
query.c/h	Creating and parsing packets
dns_test.sh	Tests
Makefile	Makefile

Table 1: Project structure

3.2 Parsing arguments

The entry point to the program is `main` function. From there, `parse_args` is called to parse the arguments specified by the user. The order of the arguments is not important. Since the task does not specify the behaviour of the program when setting parameters `-6` and `-x` at the same time, the parser outputs an error with this combination. Also, if `-h` or `--help` is present among the parameters, then only the help message is printed to standard output and the program ends with success.

3.3 Creating a DNS query

Using the `create_dns_query` function and specified parameters we obtain a query and its length. In the DNS Header, a "random" `ID` is set using `getpid`. Then `set_dns_flags` function sets header flags. If the user has specified Recursion Desired, the corresponding bit is set to 1, the rest to 0. `QDCOUNT` is 1, since we want to send 1 question, the remaining header fields are also 0.

Function `set_query_question` sets the DNS Question. `QTYPE` is set to `A`, `AAAA` or `PTR` depending on whether `-x` or `-6` flags are specified. Auxiliary function `convert_and_set_domain` translates the domain from the usual form to the DNS one (for example, for www.fit.vut.cz it will be 3www3fit3vut2cz0)[5]. Also, if `-x` is set, this function reverses the octets of given IPv4 address and adds `in-addr.arpa` string to it.

3.4 Sending the DNS query

Depending on whether we send `A` or `AAAA` query, a UDP socket is created accordingly and with the help of functions `setup_dns_server4` and `setup_dns_server6` we receive and set information about the server. Auxiliary function `get_server_ip` using DNS resolution helps to get the server address, IPv4 or IPv6. Then the query created in the previous step is sent to the server.

3.5 Receiving and parsing a DNS response

Function `receive_dns_response` receives the response from the server by calling `recvfrom` and if an error occurs, frees up the allocated memory and closes the socket.

The process of response parsing is essentially the reverse of the already described process of creating a query. Parsing consists of several parts, according to the structure of DNS packet:

- **Parsing of DNS Header.** We get information about the `RCODE` and check if an error has occurred. The task does not specify what to do in this case, so I decided to print a warning about the error code and continue parsing the packet further. Also, based on the corresponding flags, we find out whether the response is authoritative, recursive or truncated.
- **Parsing of DNS Question.** From the `QDCOUNT` value, we determine how many questions were received in total and go through each of them. We get all data according to the structure of DNS Question (3).
- **Parsing of DNS Answer.** From the `ANCOUNT` value, we determine how many answers were received in total and go through each of them. We get all data according to the structure of DNS Answer (4). Depending on the `TYPE`, `RDATA` is processed differently. If it's `A` or `AAAA`, then we just read the address, otherwise we read the domain name (or other given information). The same goes to printing out `RDATA` to the output, in the case of an IP address, the program firstly translates it from the binary format to the human readable one.

- **Parsing of DNS Authority.** From the `NSCOUNT` value, we determine how many authorities were received in total and go through each of them. We get all data according to the structure of DNS Answer (4). Unlike the DNS Answer, here we do not need to separately consider the case of IP address in the `RDATA`.
- **Parsing of DNS Additional.** The same principle as for the DNS Answer, except that the counter is now contained in `ARCOUNT`.

Each of these parts use an auxiliary function `read_domain_name` to translate data from a packet into a readable format. Unlike `convert_and_set_domain`, which on the contrary converted the information to DNS format, in response, the data can be shortened in order to reduce the size of packets. The above-mentioned function is able to work with compressed information, finding it in other parts of the packet by pointer:

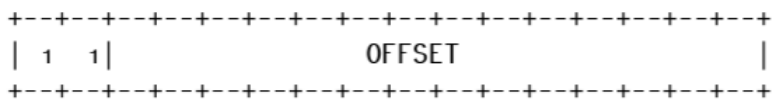

Figure 5: Compressed data

If the first two bits are set to 1, it means that it is a pointer. The `OFFSET` field specifies an offset from the start of the message.

3.6 Extensions and interesting passages

An interesting part of the implementation is the use of `pragma` to specify the alignment of structure members in memory. This prevents the occurrence of a situation when the reference server **merlin** tries to fit data up to blocks of 12 bytes.

As mentioned earlier, due to the incompleteness of the task, I decided to implement the following properties:

- Error for `-6` and `-x` combination.
- Parsing DNS packet even with a non-zero error code.

In order for the DNS Authority section to write out at least basic information, as an **extension**, I implemented handling `SOA` responses. The only difference between the usual `RDATA` and the `RDATA` for `SOA` is that it is necessary to read it twice (primary server name and responsible person name). The rest of the `SOA` data are not parsed.

For some other types, such as `NS`, `DS`, `DNSKEY` and `MX` my program just prints out the type, not parsing the rest. It lets the user at least to see which response type was received.

4 Instructions on use

To unpack the project archive, use `tar -xvf xshevc01.tar`. Then, use `make` to compile the project and `make test` to run tests.

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

Description: Perform DNS queries to resolve hostnames to IP addresses or retrieve other DNS information.

Options:

- `-r`: Enable recursion (Recursion Desired = 1), otherwise recursion is disabled.
- `-x`: Perform a reverse DNS query instead of a direct query.
- `-6`: Perform an AAAA record query instead of the default A record query.
- `-s server`: Specify the IP address or domain name of the DNS server to send the query to.
- `-p port`: Specify the port number for the DNS query (default is 53).
- `address`: The address or hostname to query.

5 Testing

To test the program, I wrote a script `dns_test.sh`. It compares the output at different parameters with the result of the `dig` program. Tests can be ran by command `make test`.

The biggest problem during testing was that WSL Ubuntu does not support IPv6 queries, as well as Ubuntu on a separate OS (at least it does not work with kazi.fit.vutbr.cz). Therefore, tests on these platforms produce errors. However, on the merlin.fit.vutbr.cz reference server, all tests pass without errors (they won't run on eva.fit.vutbr.cz because of different default shell).

```
xshevc01@merlin: ~/3rocnik/isa/isa$ make test
chmod +x dns_test.sh
./dns_test.sh

TESTS FROM THE ASSIGNMENT:
Test 1: passed!       (www.fit.vut.cz)
Test 2: passed!       (www.github.com)

TESTS FOR A:
Test 1: passed!       (roundcube.fit.vutbr.cz)
Test 2: passed!       (vut.cz)
Test 3: passed!       (slovnik.seznam.cz)
Test 4: passed!       (fluentu.com)

TESTS FOR AAAA (-6):
Test 1: passed!       (www.fit.vut.cz)
Test 2: passed!       (roundcube.fit.vutbr.cz)
Test 3: passed!       (slovnik.seznam.cz)
Test 4: passed!       (dns.google)

TESTS FOR PTR (-x):
Test 1: passed!       (147.229.9.26)
Test 2: passed!       (140.82.121.3)
Test 3: passed!       (8.8.8.8)
Test 4: passed!       (147.229.9.20)

TESTS FOR DIFFERENT SERVERS (-s):
Test 1: passed!       (dns.google)
Test 2: passed!       (resolver1.opendns.com)
Test 3: passed!       (1dot1dot1dot1.cloudflare-dns.com)

TESTS FOR SOA (bonus):
Test 1: passed!       (fluentu.com)
Test 2: passed!       (vut.cz)

OTHER TESTS:
Test 1: passed!       (-6 and -x)
Test 2: passed!       (-p a)
Test 3: passed!       (-s -x kazi...)
Test 4: passed!       (no -s)
Test 5: passed!       (-h)
xshevc01@merlin: ~/3rocnik/isa/isa$
```

Figure 6: Testing on merlin.fit.vutbr.cz

6 References

[1] AWS.Amazon. What is dns? [Online] https://aws.amazon.com/route53/what-is-dns/.

[2] Cloudflare. What is dns? [Online] https://www.cloudflare.com/learning/dns/what-is-dns/.

[3] Alan Mislove. Fundamentals of computer networking. project 1: Simple dns client. [Online] https://mislove.org/teaching/cs4700/spring11/handouts/project1-primer.pdf, 2011.

[4] P. Mockapetris. Domain names - implementation and specification. [Online] https://datatracker.ietf.org/doc/html/rfc1035, 1987.

[5] Silver Moon. Dns query code in c with linux sockets. [Online] https://www.binarytides.com/dns-query-code-in-c-with-linux-sockets/, 2020.

[6] Yu Ng. How dns works? [Online] https://www.catchpoint.com/blog/how-dns-works, 2014.