University of Glasgow | School of Computing Science

Rust DNS Server

Gordon Adam

Level 3 Project — 9 February 2015

Abstract

A description of rust, why it used for the dns server. why I chose this project.

Contents

Chapter 1

Introduction

Chapter 2

Research

2.1 BIND

BIND [1] was used during the research stage in combination with Wireshark to analyse how queries should be resolved.

2.2 Teepee and rust-http

Chapter 3

Planning

After the research stage had been carried out

3.1 MoSCoW requirements

Listed below are the project functional requirements that were identified with their priority rating based on the MoSCoW system.

3.1.1 Must Have

- *Perform iterative queries*
- *Resolve and handle resources of the type A, CNAME and SOA*
- *Implement a cache*
- *Output of Data Passing through Resolver*

3.1.2 Should Have

- *Commands to control the data output by the server*
- *Resolve PTR, TXT and IPv6 records*

3.1.3 Could Have

- *Storage of cache, that is loaded at start of server*

3.1.4 Would Have

- *Update of Root Servers*

3.2 Project Timeline

The timeline of the project was split into four main sections these sections are:

- *Project Planning:*
- *Prototype Implementation:*
- *Final Implementation and Refinements:*
- *Evaluation and Testing:*

Chapter 4

Design

4.1 User Interface

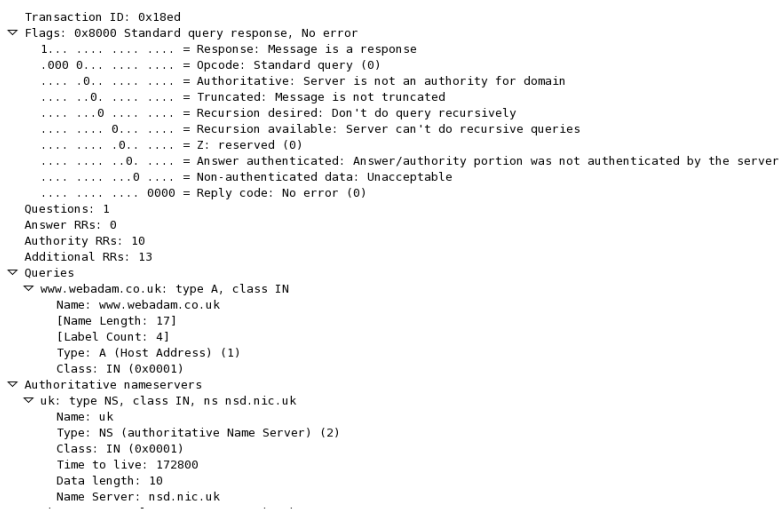
The planned user interface for this project would be very simple, consisting of a very basic command line interface. The reason for this design choice was keep to the program as lightweight, portable and easy to use as possible. The program would also be made as intuitive as it could be, so as to cause as few problems to the end user as possible. This would involve, but not be limited to, detecting the local ip address of the users machine automatically binding to that address and informing the user of it.

There would also be a few rudimentary arguments for the user to invoke at initialisation of the server that would allow the user various options in viewing the packets crossing the server. This would involve such things as options for a full detailed output of the packets, a simplified and redacted version of the packets and an option for just the hostnames of the query in the packet and the id of the packet. Of course if no arguments were given there would be no output from the server.

The inspiration for how to display the packets on the server to the end user would be taken from the packet analyzer Wireshark, this program was heavily used during the research stage of the project. The way in which it displays the information contained in a packet is clear and informative. The way in which Wireshark displays it's output is given in 4.1.

4.2 Resolver

The configuration for the structure of the resolver and the way in which it participates from within the domain name system has been taken largely from RFC 1035. The configuration as outlined in diagram **??** shows how the resolver achieves this. The process of querying multiple name servers to resolve a single query is called an iterative query, the specifics of this process will be covered within the implementation section. As is shown in the diagram there is also cache included. The cache is checked with each new message that comes in, to see if there is a response that matches the query. After a satisfactory response has been recieved from a name server or from within the cache it is forwarded to the client.



```
  Transaction ID: 0x18ed
▽ Flags: 0x8000 Standard query response, No error
    1... .... .... .... = Response: Message is a response
    .000 0... .... .... = Opcode: Standard query (0)
    .... .0.. .... .... = Authoritative: Server is not an authority for domain
    .... ..0. .... .... = Truncated: Message is not truncated
    .... ...0 .... .... = Recursion desired: Don't do query recursively
    .... .... 0... .... = Recursion available: Server can't do recursive queries
    .... .... .0.. .... = Z: reserved (0)
    .... .... ..0. .... = Answer authenticated: Answer/authority portion was not authenticated by the server
    .... .... ...0 .... = Non-authenticated data: Unacceptable
    .... .... .... 0000 = Reply code: No error (0)
  Questions: 1
  Answer RRs: 0
  Authority RRs: 10
  Additional RRs: 13
▽ Queries
  ▽ www.webadam.co.uk: type A, class IN
      Name: www.webadam.co.uk
      [Name Length: 17]
      [Label Count: 4]
      Type: A (Host Address) (1)
      Class: IN (0x0001)
▽ Authoritative nameservers
  ▽ uk: type NS, class IN, ns nsd.nic.uk
      Name: uk
      Type: NS (authoritative Name Server) (2)
      Class: IN (0x0001)
      Time to live: 172800
      Data length: 10
      Name Server: nsd.nic.uk
```

Figure 4.1: Wireshark Packet Output

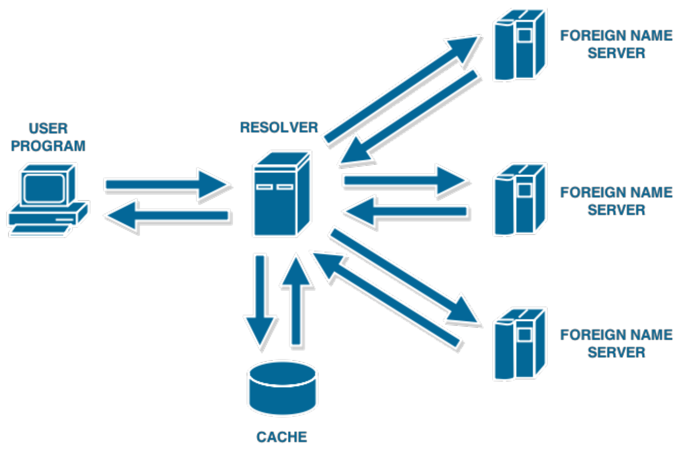


Figure 4.2: Iterative Query

A major decision that had to be made at the time of design was which types of resources the server could handle. Ideally it should be able to support them all however with there being thirty seven seperate resource record types and a short time limit set for the development process, there would need to be priorities set on which types to support. After the research stage it was concluded that the types to be supported would be the ones most commonly witnessed through the use of Wireshark. This was done so as to reduce the chances of a resource type coming through the server that was not recognised. This led to the types A, CNAME, SOA, PTR, TXT being chosen; which stands for address record, canonical name, start of authority, pointer and text record respectively. The details of these types will be explored in section 5.1.

Chapter 5

Implementation

5.1 Message Format

A DNS message is split into five distinct sections, all of which will be covered within the scope of this chapter. The order of the sections contained within a message is layed out in Figure 5.1, it is worth noting that it would not be typical for a message to contain each of the sections outlined. In fact the only mandatory field is the Header which contains details of what is contained in the rest of the message.

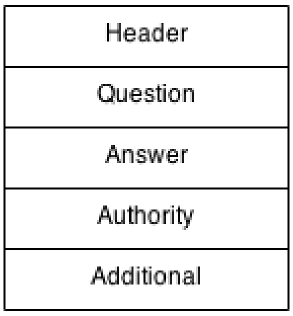


Figure 5.1: Message Format

5.1.1 Header Format

The header is the first section that is read in by the resolver and describes the content of the rest of the message. It is outlined in the following diagram.

As can be seen in the diagram the first field is an identifier which gives a simple way to identify the message during the various stages while it is resolved. As queries come into the resolver with an identifier there is not many occasions in which one is to be generated, it is implemented however by the generation of a random number.

The next field is the QR bit which specifies whether the message is a response or a query if it is set to a '1' this identifies the message as a response and a '0' identifies it as a query. After that is the

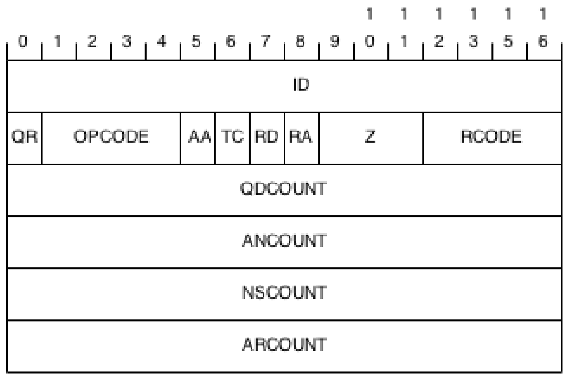


Figure 5.2: Header Format

four bit opcode this specifies the type of query. As of RFC 2929 [2] the various assignments for the opcodes are as follows:

0 Query
1 IQuery (Inverse Query)
2 Status
3 Unassigned
4 Notify
5 Update
6-15 Unassigned

Of these different assignments only the first on the list which is a standard query has been implemented for, otherwise the message is returned to sender with an error code.

What follows next are four single bit flags AA, TC, RD, and RA which stand for Authoritative Answer, Truncated, Recursion Desired and Recursion Available respectively. The authoritative answer flag specifies whether the answer provided in the answer section in the message is from an authority for that domain. Then next flag is truncated bit which tells you whether this message has had to be truncated during transmission. This flag never caused any trouble during implementation as truncation was handled at a lower level.

The third bit 'RD' lets the resolver know that the client desires the query to be performed recursively, support for recursive queries is optional within a name server, this is a resolver however so it is integral that recursion is supported. This means that when a message is received with the RD bit set to '1', the resolver responds to the client with the next bit 'RA' set to '1' to indicate to the client that it can handle recursive queries.

The next field is 3 bits long and is reserved for future use, it is represented by a 'Z' in figure 5.2 and should always be set to '000'. After that there is the RCODE field, this is used in a reply to indicate any errors there may be. It is represented by four bits, although it can be extended using a pseudo resource record [3], this however is outwith the scope of this project due to time restraints. The different error codes as specified in RFC 2136 [4] that may appear, excluding extensions, are:

0 No error condition [NOERROR]

1 Format error [FORMERR]
2 An internal server failure [SERVFAIL]
3 Name does not exist [NXDOMAIN]
4 Opcode not implemented [NOTIMP]
5 Specified operation refused [REFUSED]
6 Name that should not exist does [YXDOMAIN]
7 RRset that should not exist does [YXRRSET]
8 RRset that should exist does not [NXRRSET]
9 Not authority for zone specified [NOTAUTH]
10 Name in message not within zone specified [NOTZONE]
11-15 Unassigned

There was cause for a number of these error codes to be generated by the resolver within this project, the codes that have been implemented are '0', '1', '2' and '4'. The first of these '0' is obviously the simplest, this is to signify that everything is ok and there has been no problem handling the message. The next code '1' was generated and sent when there was some unspecified error while reading in a message that the resolver had received. After that was '2' this was used if the question could not be resolved due to none of the name servers responding and the request timing out. Finally '4' was simple to implement and would be called upon if an Opcode other than a query was present in the Opcode field.

The last four fields in the header are all counters and contain the number of questions, answers, authoritative name servers and additional records within the message respectively. The format of these records will be explained throughout the rest of this section.

5.1.2 Question Format

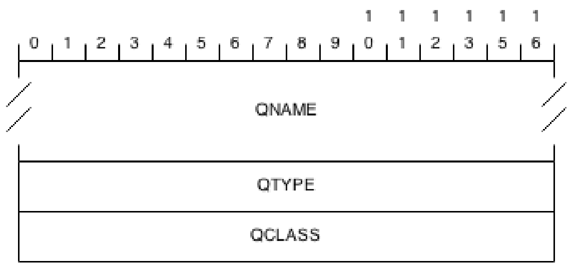


Figure 5.3: Question Format

5.1.3 Resource Record Format

The structure of the resource record was defined in

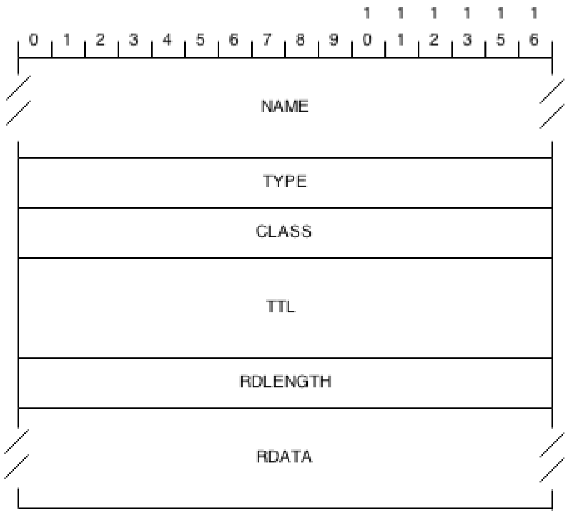


Figure 5.4: Resource Record Format

5.1.4 Message Compression

There is a simple compression scheme that is used to reduce the repetition of domain names within messages. This is done by replacing the repetion of an entire, or part of a name with a pointer as demonstrated in figure 5.5. Therefore this allows a domain name to be represented by one of the following three options:

- sequence of labels ending with a 0
- a pointer
- sequence of labels ending with a pointer

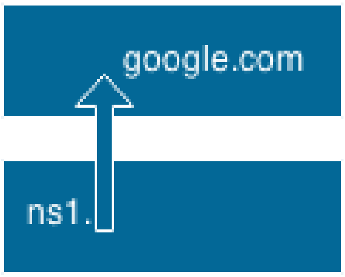


Figure 5.5: Compression

A pointer is made up of two octets, the first two bits of the first octet are set to '11' this is to indicate to the DNS system that it is indeed a pointer. The next fourteen bits indicate the octet in the message at which the rest of the domain name is located. In order to implement this in the resolver an array of octets was generated from the message before reading it in. This allows the resolver to simply lookup the array of the message at the index specified whenever it encounters a pointer in order to decompress the domain name.

5.1.5 Start of Authority Data Format

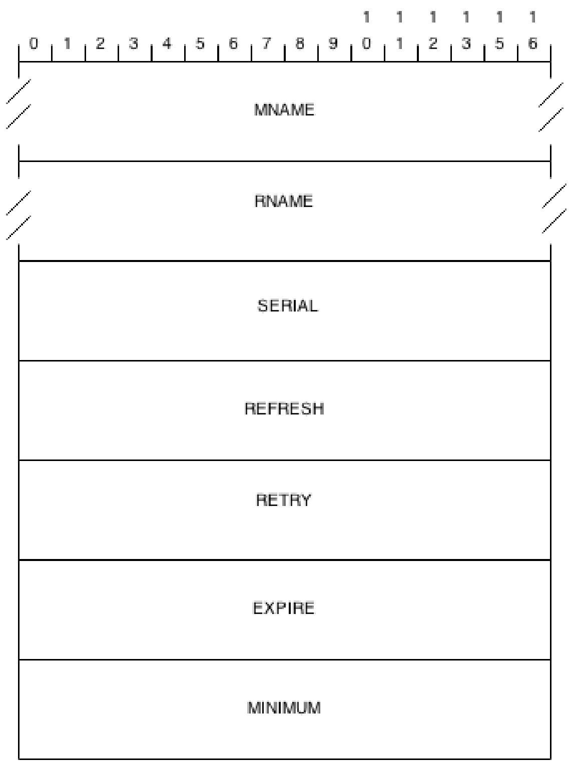


Figure 5.6: Start of Authority Data Format

Chapter 6

Evaluation

Chapter 7

Challenges

Chapter 8

Future Work

Chapter 9

Conclusion

Bibliography

[1] Internet Systems Consortium. Bind. https://www.isc.org/downloads/bind/. Accessed: 25/02/2015.

[2] Eastlake. et al. RFC 2929 DNS IANA Considerations, 2000.

[3] P. Vixie. RFC 2671 Extension Mechanisms for DNS, 1999.

[4] P. et al. Vixie. RFC 2136 Dynamic Updates in the Domain Name System, 1997.

9.1 Appendices

9.1.1 Code Repositories

Django Application - MDRS: https://github.com/Kaffse/multi-device-audio-project

Dissertation: https://github.com/allyjweir/dissertation-teamt-tp3