A DNS Resolver

Johan Montelius
KTH
VT19

Domain Name System

RFC 1035

Local Host | Foreign

+---------+ +----------+ | +--------+
| | user queries | | queries | | |
| User | ---------------| |----------|->|Foreign |
| Program | | Resolver | | | Name |
| | | |<----------|--| Server |
| | | user responses| | responses| | |
+---------+ +----------+ | +--------+

A Cache additions | references

+---------+ +--------+
V
+---------+ +--------+

client: sends request to resolver
resolver: receives requests, queries servers/resolvers and caches responses
server: responsible for sub-domain

The first resolver is most probably running on your laptop.
defmodule DNS do
  @server {8,8,8,8}
  @port 53
  @local 5300

def start() do
  start(@local, @server, @port)
end

def start(local, server, port) do
  spawn(fn() -> init(local, server, port) end)
end

The server is the DNS server to which queries are routed.

def init(local, server, port) do
  case :gen_udp.open(local, [{:active, true}, :binary]) do
    {:ok, local} ->
      case :gen_udp.open(0, [{:active, true}, :binary]) do
        {:ok, remote} ->
          DNS.dns(local, remote, server, port)
        error ->
          :io.format("DNS error opening remote socket: ~w", [error])
      end
    error ->
      :io.format("DNS error opening local socket: ~w", [error])
  end
  end
end

The server loop

def dns(local, remote, server, port) do
  receive do
    {:udp, ^local, _client, _client_port, _msg} ->
      DNS.dns(local, remote, server, port)
    :stop ->
      :ok
    :update ->
      DNS.dns(local, remote, server, port)
    strange ->
      :io.format("strange message ~w", [strange])
      DNS.dns(local, remote, server, port)
  end
end

Let's try.

The server loop

def dns(local, remote, server, port) do
  receive do
    {:udp, ^local, _client, _client_port, _msg} ->
      DNS.dns(local, remote, server, port)
    :stop ->
      :ok
    :update ->
      DNS.dns(local, remote, server, server, port)
    strange ->
      :io.format("strange message ~w", [strange])
      DNS.dns(local, remote, server, port)
  end
end

let's decode the message

definiti
0 7 8 15 16 23 24 31
identifier flags
# query blocks # answer blocks
# authority blocks # additional blocks
transport header data fields

query, answer, authority and additional blocks

Query and response messages of the same format.
message flags

- QR: query or reply
- Op-code: the operation
- AA: authoritative answer (if the server is responsible for the domain)
- TC: message truncated, more to follow
- RD: recursion desired by client
- RA: recursion available by server
- Resp-code: ok or error message in response

```
0 1 4 5 6 7 8 9 11 12 15
QR Op-code AA TC RD RA - Resp-code
```

This is getting complicated.

def decode(<<id::16, flags::binary-size(2), qdc::16, anc::16, ncs::16, arc::16, body::binary>>=raw) do
  <<qr::1, op::4, aa::1, tc::1, rd::1, ra::1, _::3, resp::4>> = flags
  decoded = decode_body(qdc, anc, ncs, arc, body, raw)
  {id, qr, op, aa, tc, rd, ra, rcode, decoded}
end

Why passing the raw message to the decoding of the body?

decode the body

The body consists of a number of: query, response, authoritative (server node) and additional sections.

The answer, authoritative and additional sections follow the same pattern, the query is slightly different.

def decode_body(qdc, anc, ncs, arc, body, raw) do
  {query, rest} = decode_query(qdc, body, raw)
  {answer, rest} = decode_answer(anc, rest, raw)
  {authority, rest} = decode_answer(ncs, rest, raw)
  {additional, _} = decode_answer(arc, rest, raw)
  {query, answer, authority, additional}
end

Note the nestling of the reminder of the body.

decode a query

A query consists of a sequence of queries (we know from the header how many).

- ⟨query⟩ ::= ⟨name⟩ ⟨query type⟩ ⟨query class⟩
- ⟨name⟩ ::= ⟨empty⟩ | ⟨label⟩ ⟨name⟩
- ⟨empty⟩ ::= 8 bits 0
- ⟨label⟩ ::= ⟨length⟩ ⟨byte sequence of length⟩
- ⟨query type⟩ ::= 16 bits (1 = A, ... 15 = MX, 16 = TXT, ...)
- ⟨query class⟩ ::= 16 bits (1 = Internet)
- ⟨length⟩ ::= 8 bits (0..63 i.e. the two highest bits are set to zero)
decode a query

```erlang
def decode_query(0, body, _) do
    {[], body}
end
def decode_query(n, body, raw) do
    {name, <<qtype::16, qclass::16, rest::binary>>} = decode_name(body, raw),
    {decoded, rest} = decode_query(n-1, rest, raw),
    {[{name, qtype, qclass} | decoded], rest}
end
```

decode a name

```erlang
def decode_name(label, raw) do
    decode_name(label, [], raw)
end
def decode_name(<<0::1, 0::1, 0::6, rest::binary>>, names, _raw) do
    {Enum.reverse(names), rest}
end
def decode_name(<<0::1, 0::1, n::6, _::binary>> = label, names, raw) do
    <<name::binary-size(n), rest::binary>> = label
    decode_name(rest, [name|names], raw)
end
```

query example

Erlang binary:

```erlang
<<4,12, 1, 0,
  0, 1, 0, 0,
  0, 0, 0, 0,
  3,119,119,119,3,107,116,104,2,115,101,0,
  0,1,0,1>>
```

Decoded query:

```erlang
{1036,0,0,0,0,1,0,0,{{['www','kth','se'],1,1]],[[][],[]]}}
```

encoding names by offset

The names in answers may use a more compact form of encoding.

Assume we have encoded `www.kth.se` and need to encode `mail.kth.se` - then we can reuse the coding of `kth.se`.

```
⟨label⟩ ::= <length> <byte sequence of length n> | ⟨offset⟩
⟨offset⟩ ::= 16 bits (two highest bits set to ones)
```

The length version will always have the top two bits set to 00 and the offset version will have them set to 11.
def decode_names(<<1::1, 1::1, n::14, rest::binary>>, names, raw) do
  ## offset encoding
  <<_::binary-size(n), section::binary>> = raw
  {name, _} = decode_names(section, names, raw)
  {name, rest}
end
end

All answer sections have the same basic structure:

⟨answer⟩ ::= ⟨name⟩ ⟨type⟩ ⟨class⟩ ⟨ttl⟩ ⟨length⟩ ⟨resource record⟩

- type 16-bits: A-type, NS-, CNAME-, MX- etc
- class 16-bits: Internet, ...
- TTL 32-bits: time in seconds (typical some hours)
- length 16-bits: the length of the record in bytes

The resource record is coded depending on the type of resource.

let's try

forward the reply