Drink: implementation details

Who?  
Stéphane Bortzmeyer  
stephane+fosdem@bortzmeyer.org

When?  
FOSDEM, 4 February 2023
What is Drink

A tramway station in Antwerp
Demo

% dig @2001:4860:4860::8888 2+2.op.dyn.bortzmeyer.fr TXT

...;

;; Got answer:

;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 41999

;; flags: qr rd ra ad; QUERY: 1, ANSWER: 2, AUTHORITY: 0, ADDITIONAL: 1

...

;; ANSWER SECTION:

2+2.op.dyn.bortzmeyer.fr. 21600 IN TXT "4"
2+2.op.dyn.bortzmeyer.fr. 21600 IN RRSIG TXT 8 5 86400 ( 20230203154000 20230208040000 63937 dyn.bortzmeyer.fr. rnCWshkZl1lQInUPnahx1WjUVP66Bdd+ff/suCItb9e+

...

;; WHEN: Sat Feb 04 10:09:13 CET 2023

;; MSG SIZE rcvd: 276
A dynamic authoritative name server

Services: returns the IP address of the client, and a few others, probably less useful.

Goals: learn, have fun, implement a lot of DNS stuff, test ideas at IETF hackathons.
A dynamic authoritative name server

Services: returns the IP address of the client,
A dynamic authoritative name server

- Services: returns the IP address of the client, and a few others, probably less useful (well, ECS echo could be useful).
A dynamic authoritative name server

- Services: returns the IP address of the client, and a few others, probably less useful.
- Goals: learn, have fun, implement a lot of DNS stuff (TCP, NSID, cookies, DNSSEC), test ideas at IETF hackathons.
Implementation

Implementation

- Written in Elixir (with the Erlang runtime),

Free software at https://framagit.org/bortzmeyer/drink.
Implementation

- Written in Elixir,
- Relies on some libraries: many interesting issues (no perfect DNS library, unlike Go or Python).
Implementation

- Written in Elixir,
- Relies on some libraries: many interesting issues.
- Can itself call remote microservices (so can be slow and unreliable),

Free software at https://framagit.org/bortzmeyer/drink.
Implementation

- Written in Elixir,
- Relies on some libraries: many interesting issues.
- Can itself call remote microservices,
- Free software at https://framagit.org/bortzmeyer/drink.
Parallelism

Elixir/Erlang favor massive parallelism,
Every DNS request is a process,
Every TCP connection is a process,
Ancillary stuff is a process,
Consequences: a crashed or stuck request does not
block the server,
For TCP, pipelining and out-of-order replies worked
without even thinking of it.
Unlike what many people say, parallel programming is
simpler.
Parallelism

- Elixir/Erlang favor massive parallelism,
Parallelism

- Elixir/Erlang favor massive parallelism,
- Every DNS request is a process (an Erlang one, not an OS one),
Parallelism

- Elixir/Erlang favor massive parallelism,
- Every DNS request is a process,
- Every TCP connection is a process,
Parallelism

- Elixir/Erlang favor massive parallelism,
- Every DNS request is a process,
- Every TCP connection is a process,
- Ancillary stuff such as logging periodic statistics is a process,
Parallelism

- Elixir/Erlang favor massive parallelism,
- Every DNS request is a process,
- Every TCP connection is a process,
- Ancillary stuff is a process,
- Consequences: a crashed or stuck request does not block the server,
Parallelism

- Elixir/Erlang favor massive parallelism,
- Every DNS request is a process,
- Every TCP connection is a process,
- Ancillary stuff is a process,
- Consequences: a crashed or stuck request does not block the server,
- For TCP, pipelining and out-of-order replies worked without even thinking of it.
Parallelism

- Elixir/Erlang favor massive parallelism,
- Every DNS request is a process,
- Every TCP connection is a process,
- Ancillary stuff is a process,
- Consequences: a crashed or stuck request does not block the server,
- For TCP, pipelining and out-of-order replies worked without even thinking of it.
- Unlike what many people say, parallel programming is simpler.
Spawning the TCP processes

```elixir
Enum.map(addresses, fn address ->
    socket_result = Socket.TCP.listen(config()[:port],
        [version: version,
         packet: 2, # Automatically add/read a 2-bytes
                   # length before data.
         mode: :binary,
         local: [address: address]])
    socket = socket_open(socket_result)
    tcp_pid = spawn_link(Drink.Server,
                          :tcp_loop_acceptor,
                          [socket, config()[:bases]])
    Process.monitor(tcp_pid)
end)
```
Robustness
Robustness

Everybody loves RFC 9267,
Robustness

- Everybody loves RFC 9267,
- The Internet is a jungle,
Robustness

- Everybody loves RFC 9267,
- The Internet is a jungle,
- Don’t trust the integrity of the incoming packets,
Robustness

- Everybody loves RFC 9267,
- The Internet is a jungle,
- Don’t trust the integrity of the incoming packets,
- Compression pointers are a great source of security bugs,
Robustness

- Everybody loves RFC 9267,
- The Internet is a jungle,
- Don’t trust the integrity of the incoming packets,
- Compression pointers are a great source of security bugs,
- EDNS can be fun, too (had to be done from scratch for Drink).
def extract_edns_opt(bin) do
  <<code::unsigned-integer-size(16)>> =
  Binary.part(bin, 0, 2)
  code_txt =
  case code do
    Drink.EdnsCodes.nsid -> :nsid
    ...
    other -> other
  end
  # Read RFC 6891
  <<length::unsigned-integer-size(16)>> =
  Binary.part(bin, 2, 2)
  data = Binary.part(bin, 4, length)
  [{code_txt, length, data} |
    extract_edns_opt(Binary.part(bin, 4+length, byte_size(bin)-(4+length))]
rescue
  e ->
    raise Drink.EdnsError, inspect(e)
end
A dynamic server requires dynamic signing,
Cryptography is fun: one forgotten bit and everything is
wrong,
Example of a problem: the default encoding of DNS
replies compresses names in NS and SOA messages.
A dynamic server requires dynamic signing,
A dynamic server requires dynamic signing,

Cryptography is fun: one forgotten bit and everything is wrong,
A dynamic server requires dynamic signing,

Cryptography is fun: one forgotten bit and everything is wrong,

Example of a problem: the default encoding of DNS replies compresses names in NS and SOA messages (no way to disable it, I had to rewrite the encoding from scratch).
# RFC 4034, section 3.1.8.1

```elixir
owner_bin = Binary.from_list(Drink_Utils.encode(String.downcase(owner)))
short_rrsig = \<<\ntype::unsigned-integer-size(16),
  @algorithm::unsigned-integer-size(8),
  num_labels::unsigned-integer-size(8),
  ttl::unsigned-integer-size(32),
  expiration::unsigned-integer-size(32),
  inception::unsigned-integer-size(32),
  tag::unsigned-integer-size(16)>>\n |> Binary.append(owner_bin)
encoded_rrset = Drink.Encoding.encode(data)
{:ok, sig} = ExPublicKey.sign(Binary.append(short_rrsig, encoded_rrset), key)
short_rrsig |> Binary.append(sig)
```
Thou shall not lie

Dynamic signing of negative answers requires to ignore the 9th commandment, Drink uses the white lies of RFC 4470, hard to get right and the behaviour of resolvers vary.
Thou shall not lie

Dynamic signing of negative answers requires to ignore the 9th commandment,
Thou shall not lie

- Dynamic signing of negative answers requires to ignore the 9th commandment,

- Drink uses the white lies of RFC 4470 (generating NSEC records going from “a bit before” to “a bit after”),
Thou shall not lie

- Dynamic signing of negative answers requires to ignore the 9th commandment,
- Drink uses the white lies of RFC 4470,
- Hard to get right and the behaviour of resolvers vary.
Generating NSEC bitmaps

```elixir
block = floor(Enum.min(l)/256)
todo = Enum.filter(l, fn type -> type < (256*(block+1)) end)
todo = Enum.map(todo, fn type -> type - (256*block) end)
bits = bits_of(todo, 0)
remainder = rem(length(bits), 8)
pad_size =
  if remainder == 0 do
    0
  else
    8 - remainder
  end
```
Tests

- Internal tests with the Elixir framework,
- External tests from a Python program,
- Important: tests with broken requests.
Tests

Internal tests with the Elixir framework,
Tests

- Internal tests with the Elixir framework,
- External tests from a Python program (for diversity),
Tests

- Internal tests with the Elixir framework,
- External tests from a Python program,
- Important: tests with broken requests.
Generating broken requests

edns_option = struct.pack(">H", nsid) + \\n    struct.pack(">H", 14) # Wrong length
additional_section = struct.pack("B", 0) + \\n    struct.pack(">H", opt) + \\n    struct.pack(">H", bufsize) + \\n    struct.pack(">L", 0) + struct.pack(">H", 4) - \\n    edns_option

data = struct.pack(">HHHHHH", id, misc, 1, 0, 0, 1) + \\n    encode_name(domain) + struct.pack(">H", txt) + \\n    struct.pack(">H", in_class) + \\n    additional_section
s.sendto(data, sockaddr)
rdata, remote_server = s.recvfrom(4096)
resp = dns.message.from_wire(rdata)
assert resp.rcode() == formerr