A dynamic experimental DNS server, just for fun

Stéphane Bortzmeyer
<stephane+blog@bortzmeyer.org>

First publication of this article on 31 May 2022. Last update on of 13 June 2022

https://www.bortzmeyer.org/drink.html

Let me introduce you to a new program, an experimental authoritative DNS server intended for dynamic answers (answers depending, for instance, on the client). It is just for fun and it does not pretend to replace existing programs. But you may want to read its source code, or use its online demo, at dyn.bortzmeyer.fr.

My main goal was to have fun. A secondary goal was to have a service to get the IP address used by resolvers to query authoritative name servers. So, as said above, this program is not intended for mission-critical uses.

The program is named Drink, for no special reason. Its source code is available online <https://framagit.org/bortzmeyer/drink>, under a free-software licence. It uses the Elixir programming language and I'll talk about it later.

Let's first use the program. An instance is installed on the domain dyn.bortzmeyer.fr. The tests are done with dig, the links in this article are through the DNS looking glass <https://www.bortzmeyer.org/dns-lg-usage.html>. You can send TXT queries to get help <https://dns.bortzmeyer.org/dyn.bortzmeyer.fr/TXT> :

```
% dig dyn.bortzmeyer.fr TXT
...
;; ANSWER SECTION:
dyn.bortzmeyer.fr. 0 IN TXT "Possible queries: " "hello/TXT to have a greeting. " "ip/TXT,A,AAAA to have the IP
```

A query for the subdomain hello will produce a greeting and the version numbers of the programs used <https://dns.bortzmeyer.org/hello.dyn.bortzmeyer.fr/TXT> :

```
% dig hello.dyn.bortzmeyer.fr TXT
...
;; ANSWER SECTION:
hello.dyn.bortzmeyer.fr. 0 IN TXT "Hello, this is the Drink DNS server, version 0.1.0 and I use the dns libr
```

The most useful service today, in my opinion, is at the subdomain `ip`, returning the IP address of the DNS client :

```
% dig ip.dyn.bortzmeyer.fr AAAA
...
;; ANSWER SECTION:
ip.dyn.bortzmeyer.fr. 0 IN AAAA 2001:860:de02:102::d2
```

Two important things about this service :

— If you do A (IPv4) or AAAA (IPv6) requests, and your resolver uses the other address family, you'll get a response but not of the queried type, and it will be put in the Additional section of the DNS message. Some resolvers will filter out this unexpected response. Use TXT requests if you want to avoid this.

— The IP address returned is not your IP address, nor it is the one you use to talk to your resolver. Here, we query through Quad9 <https://quad9.net/> :

```
% dig @2620:fe::9 ip.dyn.bortzmeyer.fr A
...
;; ANSWER SECTION:
ip.dyn.bortzmeyer.fr. 0 IN A 66.185.123.250
```

As you can see, we queried Quad9 over IPv6 but Quad9 queried our dynamic server over IPv4, using one of PCH addresses.

If you want to know the ECS option sent by your resolver, you can query the `ecs` service (TXT query).

You can get the date and time of the DNS server <https://dns.bortzmeyer.org/date.dyn.bortzmeyer.fr/TXT>, in RFC 3339[1] syntax :

```
% dig TXT date.dyn.bortzmeyer.fr
...
;; ANSWER SECTION:
date.dyn.bortzmeyer.fr. 0 IN TXT "2022-05-31T11:05:36.507343Z"
```

1. Pour voir le RFC de numéro NNN, `https://www.ietf.org/rfc/rfcNNN.txt`, par exemple `https://www.ietf.org/rfc/rfc3339.txt`

(Obviously, it is less efficient to synchronize clocks than NTP.) You can also get a random number <https://dns.bortzmeyer.org/random.dyn.bortzmeyer.fr/TXT>, or IP address <https://dns.bortzmeyer.org/random.dyn.bortzmeyer.fr/AAAA> :

```
% dig A random.dyn.bortzmeyer.fr
...
;; ANSWER SECTION:
random.dyn.bortzmeyer.fr. 0 IN A 149.154.12.82
```

It's probably a useless service but you can use it to explore randomly the Internet, with commands such as `whois $(dig +short random.dyn.bortzmeyer.fr A)`. **Beware**, there is a security weakness in this command. Can you spot it ? If you are the sort of person who does `curl http://random-site.example/sc | sudo bash`, don't worry, this is not worse. Also, this command is less fun with IPv6, since a great part of the IPv6 address space is unallocated.

Drink has EDNS (RFC 6891) and recognizes a few options, such as the maximum size of the answer :

```
% dig @ns1-dyn.bortzmeyer.fr +bufsize=50 date.dyn.bortzmeyer.fr TXT
;; Truncated, retrying in TCP mode.
...
;; ANSWER SECTION:
date.dyn.bortzmeyer.fr. 0 IN TXT "2022-05-31T11:24:09.318345Z"
...
;; MSG SIZE  rcvd: 91
```

As you can see from the "Truncated, retrying in TCP mode.", Drink returned a response with the truncation flag, and dig retried with TCP (which is of course supported by Drink). Another thing you can do with EDNS, is to query NSID (Name Server Identifier, RFC 5001) :

```
% dig @ns1-dyn.bortzmeyer.fr +nsid date.dyn.bortzmeyer.fr TXT
...
;; OPT PSEUDOSECTION:
; EDNS: version: 0, flags:; udp: 1200
; NSID: 6e 73 31 2d 64 79 6e 2e 62 6f 72 74 7a 6d 65 79 65 72 2e 66 72 ("ns1-dyn.bortzmeyer.fr")
...
;; ANSWER SECTION:
date.dyn.bortzmeyer.fr. 0 IN TXT "2022-05-31T11:31:43.240583Z"
```

As you can see, the name server identifier is returned (not terribly useful in that case, but much more interesting if you use anycast).

Another service is at the subdomain `bgp`, returning not only the IP address of the DNS client, but also the IP prefix announced in the DFZ, and the AS number which originates this prefix :

```
% dig bgp.dyn.bortzmeyer.fr TXT
...
;; ANSWER SECTION:
bgp.dyn.bortzmeyer.fr. 3597 IN TXT "2605:4500:2:245b::bad:dcaf" "2605:4500::/32" "46636"
```

(It uses the `bgp.bortzmeyer.org` HTTPS service.) Here, the resolver `2605:4500:2:245b::bad:dcaf` is part of the prefix `2605:4500::/32`, announced by AS 46636 <https://search.arin.net/rdap/?query=AS46636>. This is specially interesting when you ask many separate machines to query this name. For instance, with RIPE Atlas probes <https://atlas.ripe.net/> and the Blaeu program <https://labs.ripe.net/author/stephane_bortzmeyer/creating-ripe-atlas-one-off-measurer>, we can ask 20 probes to do a DNS resolution :

```
% blaeu-resolve --requested 20 --type TXT bgp.dyn.bortzmeyer.fr
["195.211.77.68" "195.211.76.0/23" "49825"] : 1 occurrences
["162.158.85.93" "162.158.84.0/22" "13335"] : 1 occurrences
["173.194.169.12" "173.194.0.0/16" "15169"] : 1 occurrences
["2001:610:1:40ba:145:100:185:17" "2001:610::/29" "1103"] : 1 occurrences
["192.87.106.106" "192.87.0.0/16" "1103"] : 1 occurrences
["45.32.149.90" "45.32.144.0/21" "20473"] : 1 occurrences
["185.73.24.170" "185.73.24.0/22" "201454"] : 1 occurrences
["66.96.115.242" "66.96.112.0/20" "715"] : 1 occurrences
["2a02:908:2:110b::24" "2a02:908::/33" "3209"] : 1 occurrences
["2404:4408:5::b9" "2404:4400::/28" "9790"] : 1 occurrences
["2607:fb90:c13e:fff0:b77b:5ed3:0:aaaa" "2607:fb90:c13e::/48" "22140"] : 1 occurrences
["2001:1438:2:14::11" "2001:1438::/32" "8881"] : 1 occurrences
["2a04:e4c0:14::69" "2a04:e4c0:14::/48" "36692"] : 1 occurrences
["195.121.117.200" "195.121.64.0/18" "8737"] : 1 occurrences
["66.185.123.252" "66.185.123.0/24" "42"] : 1 occurrences
["185.22.47.150" "185.22.44.0/22" "60294"] : 1 occurrences
["162.158.201.67" "162.158.200.0/22" "13335"] : 1 occurrences
["212.142.48.77" "212.142.32.0/19" "6830"] : 1 occurrences
["184.83.74.52" "184.83.0.0/16" "11232"] : 1 occurrences
["213.13.28.71" "213.13.0.0/16" "3243"] : 1 occurrences
Test #41687133 done at 2022-06-11T10:17:45Z
```

Which gives us a glimpse of the resolvers they use (AS 12335 <https://search.arin.net/rdap/?query=AS13335> is Cloudflare and AS 15169 <https://search.arin.net/rdap/?query=AS15169> is Google, both operating public DNS resolvers used by some probes).

OK, let's assume you're convinced and you want to install Drink on your machines. Since it is written in the Elixir programming language, which requires a runtime, you need to install Elixir, and also the Erlang sources, to compile the DNS library (on Debian, this is `apt install elixir erlang-dev erlang-src`, on Arch, `pacman -S elixir erlang-nox`). Then, get the code with git and just type `mix deps.get` then `mix run drink.exs` (see the `README.md` file for details). You will probably need to create a configuration file or to provide options on the command line, see again `README.md` for details.

For the `ip` service, you can find similar existing services :

— `resolver.00f.net` (A and AAAA, if the request is over IPv4, it creates an IPv4-mapped IPv6 address, see RFC 4291, section 2.5.5.2) ; besides Drink, it seems to be one of the few whose source code is available <https://github.com/jedisct1/whatsmyresolver>,

- `dns.toys` is a bit different, it is not a zone in the usual DNS tree, you need to query the authoritative server directly (and so you need a clean DNS path, something which is not always available, for instance at WiFi hotspots, and you won't learn the IP address of your resolver); try `dig ip @dns.toys` (and see their other funny services <https://www.dns.toys/>); its source code (in Go) is available <https://github.com/knadh/dns.toys>,
- `_country.pool.ntp.org` (TXT queries) displays the IP address of the client, its port, the country, and ECS information.
- `whoami.v4.powerdns.org` and `whoami.v6.powerdns.org` (TXT, A and AAAA),
- `o-o.myaddr.l.google.com` (only TXT queries),
- `whoami[Caractère Unicode non montré [2] ].[Caractère Unicode non montré ]fastly[Caractère Unicode non montré ].[Caractère Unicode non montré ]net` and `whoami[Caractère Unicode non montré ]6.[Caractère Unicode non montré ]fastly[Caractère Unicode non montré ].[Caractère Unicode non montré ]net` (A, AAAA, and TXT queries, the TXT queries also give you ECS information).
- `whoami.akamai.net` (A and AAAA),
- `resolver-identity.cloudfront.net` (A and AAAA),
- `whoami.ultradns.net` (only A requests, even if the servers have IPv6),

If you know other similar DNS services on the Internet, don't hesitate to report them. (They tend to be short-lived, many old ones no longer work, or return broken results.)

If you have ideas about services implemented on Drink, or bugs to report, please create a ticket <https://framagit.org/bortzmeyer/drink/-/issues>.

Now, a few words about the source code. One of the reasons of the choice of the programming language Elixir is its excellent support of parallelism (or, rather, the excellent support of parallelism by the Erlang virtual machine). This is priceless for Internet servers. One process (an Elixir process, not an operating system process) is created for each connection (a connection being an UDP request or a "real" connection, with TCP) and the parallelism between clients is efficiently handled by the virtual machine. This isolation of processes (they share nothing, not even memory) also protects the server against malicious or broken clients, that can wait for a long time sending anything, or can send badly crafted DNS messages (something which is quite common on the Internet). A process may crash (for instance when attempting to decode a malformed DNS message) but the server will continue to serve the other clients.

The Drink server handles, of course, UDP and TCP (which is mandatory, see RFC 7766 and RFC 9210, but often forgotten in "custom" DNS servers). This is also something that is relatively easy with Elixir parallelism.

The DNS library I used <https://github.com/tungd/elixir-dns/> saved me a lot of work, thanks to its author. But it also has some limitations. For instance, it has no support for EDNS options <https://github.com/tungd/elixir-dns/issues/48> so I had to add it (not a big deal, but still annoying).

If you know Elixir, and its culture, you may be surprised to see that Drink does not use many things common in the Elixir world, such as OTP supervisors or protocols like GenServer. This is because, for this specific case, it seems to me they were not adding a lot of value. I may change my mind later.

One word about performance, testing with the excellent dnsperf tool <https://github.com/DNS-OARC/dnsperf>. On a PC with two 2.6 Ghz cores, running dnsperf with one hundred parallel sending threads yields 10,000 requests per second and zero failure. (This is with Drink's logging disabled, if you log each query, the DNS server will be much slower.)

2. Car trop difficile à faire afficher par LaTeX

```
% cat data
ip.test A
hello.test TXT
test SOA

%   ./src/dnsperf -n 10000 -c 100  -s 127.0.0.1 -p 3553 -d data
DNS Performance Testing Tool
Version 2.9.0

[Status] Command line: dnsperf -n 10000 -c 100 -s 127.0.0.1 -p 3553 -d data
[Status] Sending queries (to 127.0.0.1:3553)
[Status] Started at: Tue May 31 21:12:09 2022
[Status] Stopping after 10000 runs through file
[Status] Testing complete (end of file)

Statistics:

  Queries sent:         30000
  Queries completed:    30000 (100.00%)
  Queries lost:         0 (0.00%)

  Response codes:       NOERROR 30000 (100.00%)
  Average packet size:  request 25, response 90
  Run time (s):         2.974810
  Queries per second:   10084.677677

  Average Latency (s):  0.009701 (min 0.000258, max 0.028758)
  Latency StdDev (s):   0.002650
  Latency StdDev (s):   0.022502
```