Developing DNS-over-HTTPS clients and servers

Stéphane Bortzmeyer
<stephane+blog@bortzmeyer.org>

First publication of this article on 23 March 2018

https://www.bortzmeyer.org/hackathon-ietf-101.html

The weekend of 17-18 March 2018, I participated to the IETF 101 <https://www.ietf.org/how/meetings/101/> hackathon in London. The project was DoH, aka DNS-over-HTTPS, and the idea was to develop clients, servers, and to test that they interoperate.

DoH (DNS-over-HTTPS) is not yet published as a RFC. One of the goals of IETF hackathons <https://www.ietf.org/how/runningcode/hackathons/> is precisely to test Internet-Drafts before they become RFC, to be reasonably sure they are not wrong, too complicated, or useless. DoH is developed in the DoH working group <https://datatracker.ietf.org/wg/doh/> and currently has one Internet-Draft, the specification of DNS-over-HTTPS, `draft-ietf-doh-dns-over-https`. Why creating DoH? DNS privacy is the main factor behind this project. Issues with DNS privacy are documented in RFC 7626[1]. One of them is that traffic is sent in clear and therefore can be read by any sniffer. To prevent that, there is a standard, in RFC 7858, to run DNS over TLS, using a dedicated port, 853. But this port may be easily blocked by an hostile middlebox. The only port which is always open is 443, because it's used by HTTPS. Of course, DNS-over-TLS could use port 443 but you may have DPI devices checking that it is actually HTTPS running (yes, the trafic is encrypted but think of things like TLS' ALPN). And HTTPS gives us other things : proxies, caching, availability from JavaScript code...

So, DNS-over-HTTPS. This technique allows a stub resolver to talk to a DNS resolver over a secure transport. Let's see if we can implement the draft and make this implementation work with other implementations. My personal idea was to modify the excellent getdns <https://getdnsapi.net/> library to add DoH as a possible transport (DNS-over-TLS is already there). But it was too complicated for me and, moreover, Willem Toorop decided to refactor the code, to make easier to add new transports, so getdns was too "in flux" for me. (Willem worked on it during the hackathon.) Instead, I developed first a server in Python, then developed a client in Python to test my server, then tested them against other clients and servers, then developed a second client in C. Let's see the issues.

1. Pour voir le RFC de numéro NNN, https://www.ietf.org/rfc/rfcNNN.txt, par exemple https://www.ietf.org/rfc/rfc7626.txt

DoH requires (I know, the actual rules are more complicated than a simple requirement) HTTP/2 (RFC 7540). One of the reasons is that DNS requests can take a very variable time. You don't want your requests for `datatracker.ietf.org` to be delayed by a previous request for `brokendomain.allserversdown.e` standing in the queue. HTTP/2, with its streams, allow requests to be run in parallel. But HTTP/2 is recent, and many libraries and servers don't support it yet, specially on stable releases of operating systems. For the Python server, I choose the Quart framework `<https://gitlab.com/pgjones/quart>`, which relies itself on hyper `<https://pypi.python.org/pypi/h2>`, an implementation of HTTP/2 in Python. Because these were recent libraries, not always available as a package for Ubuntu, I created a LXC container with the "unstable" (very recent) version of Debian. I installed Quart with pip, as well as dnspython `<http://www.dnspython.org/>`. dnspython is required because DoH uses the DNS wire format, a binary format (other systems running DNS over HTTPS, not yet standardized, use JSON). So, I needed to pack DNS packets from data and to unpack them at the other end, hence dnspython.

Like many HTTP development frameworks for Python, Quart allows you to define code to be run in response to some HTTP methods, for a given path in the URI. For instance :

```
@app.route('/hello')
async def hello():
    return 'Hello\n'
```

The decorator `@app.route` routes requests to `https://YOURDOMAIN/hello` to the `hello` routine, which executes asynchronously (people used to Flask will recognize the syntax ; those who don't know Flask should learn it, in order to be able to use Quart). More complicated :

```
@app.route('/dns', methods=['POST'])
async def index():
      ct = request.headers.get('content-type')
      if ct != "application/dns-udpwireformat":
          abort(415)
      data = await request.get_data()
      r = bytes(data)
      message = dns.message.from_wire(r)
      # get the DNS response from the DNS message, see later...
      return (response
           {'Content-Type': 'application/dns-udpwireformat'})
```

Here, we handle only POST requests, we check the `Content-Type:` HTTP header, we parse the body of the request with dnspython (`dns.message.from_wire(...)`) and we return a response with the proper content type.

How do we get the answer to a specific DNS request ? We simply give it to our local resolver, with dnspython :

```
resolver = "::1"
raw = dns.query.udp(message, resolver)
response = raw.to_wire()
```

The biggest goal of DoH is privacy, so we need to activate encryption :

```
tls_context = ssl.create_default_context(ssl.Purpose.CLIENT_AUTH)
tls_context.options |= ssl.OP_NO_TLSv1 | ssl.OP_NO_TLSv1_1 | ssl.OP_NO_COMPRESSION
tls_context.set_alpn_protocols(['h2', 'http/1.1'])
app.run(host=bind, port=port, ssl=tls_context)
```

(We accept HTTP/1.1, also, because we're tolerant.) To get a certificate (because, unfortunately, few programs and libraries support DANE), we use Let's Encrypt. The server I wrote cannot handle the ACME challenge. But one call to `certbot certonly`, choosing the option "Spin up a temporary webserver" (with my own server stopped, of course) was enough to get a nice certificate. I then load it :

```
tls_context.load_cert_chain(certfile='le-cert.pem', keyfile='le-key.pem')
```

Putting every together, we have the complete code . You run it with simply :

```
% ./quart-doh.py -c -r ::1
```

Obviously, this is not a successful hackathon if you don't discover at least one bug <https://gitlab.com/pgjones/quart/issues/72> in the library. Note it was fixed by the author even before the end of the event.

Having a server is nice but there were not many DoH clients to test it (some were developed during the hackathon). I then developed a client in Python, still with dnspython for the DNS part, but using pycurl <http://pycurl.io/> for HTTP/2. The DNS request is built from a name entered by the user (note that the DNS query type, here, is fixed and set to ANY) :

```
message = dns.message.make_query(queryname, dns.rdatatype.ANY)
message.id = 0 # DoH requests that
```

We use pycurl to establish a HTTP/2 connection :

```
c = pycurl.Curl()
c.setopt(c.URL, url) # url is the URL of the DoH server
data = message.to_wire()
c.setopt(pycurl.POST, True)
c.setopt(pycurl.POSTFIELDS, data)
c.setopt(pycurl.HTTPHEADER, ["Content-type: application/dns-udpwireformat"])
c.setopt(c.WRITEDATA, buffer)
c.setopt(pycurl.HTTP_VERSION, pycurl.CURL_HTTP_VERSION_2)
c.perform()
```

The `c.setopt(pycurl.HTTP_VERSION`, where we require HTTP/2, works only if the libcurl <https://curl.haxx.se/libcurl/> library used by pycurl has been linked with the nghttp2 <https://nghttp2.org/> library. Otherwise, you get a `pycurl.error: (1, '')` which is not very helpful (error 1 is `CURL_UNSUPPORTED_PROTOCOL`). Again, you need recent versions of everything.

We then get the answer in the `buffer` variable, we can parse it and do something with it :

```
body = buffer.getvalue()
response = dns.message.from_wire(body)
```

The complete code is . You can run it this way (here using one of the public DoH servers) :

```
% ./doh-client.py https://dns.dnsoverhttps.net/dns-query gitlab.com
...
;ANSWER
gitlab.com. 300 IN A 52.167.219.168
...
```

I also developed a C client. Because parallel programming in C is very difficult (unlike Go, where it is a pleasure), I wanted an asynchronous HTTP/2 library, in order to make it usable in the future in getdns `<https://getdnsapi.net>`, which is asynchronous. I use nghttp2 `<https://nghttp2.org/>`, already mentioned, and getdns for the DNS packing and unpacking (parsing). The HTTP/2 code was shamelessly copied from a nghttp2 example, so let's focus on the DNS part. getdns provides `getdns_convert_fqdn_to_dns_name` to put names in DNS wire format (if you don't know the DNS, remember the wire format is different from the presentation format `www.foobar.example`; for instance, the wire format do not use dots) and routines like `getdns_dict_set_bindata` to create getdns messages :

```
getdns_convert_fqdn_to_dns_name (session_data->qname, dns_name_wire_fmt);
getdns_dict_set_bindata (dict, "qname", *dns_name_wire_fmt);
getdns_dict_set_int (dict, "qtype", GETDNS_RRTYPE_A);
getdns_dict_set_dict (qdict, "question", dict);
getdns_dict_set_int (rdict, "rd", 1);
getdns_dict_set_dict (qdict, "header", rdict);
```

Yes, building getdns data structures is a pain. In the end, all that was necessary was (as displayed by `getdns_pretty_print_dict(qdict)`) :

```
{
  "header":
  {
    "rd": 1
  },
  "question":
  {
    "qname": <bindata for gitlab.com.>,
    "qtype": GETDNS_RRTYPE_A
  }
}
```

We then put it in DNS wire format with `getdns_msg_dict2wire (qdict, buffer, &size);` and give it to nghttp2. At this time, it works only for GET requests, there is something wrong in the code I used for sending the body in POST requests.

When getting the answer, getdns allows us to search info with the JSON pointer (RFC 6901) syntax (getdns does not use JSON but the data model is the same) :

```
getdns_dict_get_int (msg_dict, "/header/rcode", &this_error);
getdns_dict_get_bindata (msg_dict, "/answer/0/rdata/ipv4_address", &this_address_data);
char *this_address_str = getdns_display_ip_address (this_address_data);
fprintf (stdout, "The address is %s\n", this_address_str);
```

The complete code is (en ligne sur `https://www.bortzmeyer.org/files/doh-nghttp.c`) and can be used this way :

```
% ./doh-nghttp  https://dns.dnsoverhttps.net/dns-query gitlab.com
The address is 52.167.219.168
```

The `-v` option will display a lot more details.

What were the lessons learned during the hackathon ? I let you see that in the presentation I gave at the DoH working group afterwards. `<https://datatracker.ietf.org/meeting/101/materials/slides-101-doh-hackathon-feedback-01>` For the other code developed during the hackathon, see the notes taken during the hackathon `<https://github.com/IETF-Hackathon/ietf101-project-presentat` `blob/master/DoH/README.md>`.

Other reports :
— Sara Dickinson's synthesis at the end of the hackathon `<https://github.com/IETF-Hackathon/ietf101-project-presentations/blob/master/DNS_DOH_presentation.pdf>`,
— Tony Finch's notes, day 1 `<https://fanf.dreamwidth.org/123507.html>` and day 2 `<https://fanf.dreamwidth.org/123738.htm>`.

Many thanks to Charles Eckel for organising this wonderful event, to the other people working on DoH at the same time, making this both a fun and useful experience, and to the authors of the very good libraries I used, Quart, nghttp2, getdns and pycurl.