Documentation technique de mon résolveur DoH

Stéphane Bortzmeyer
<stephane+blog@bortzmeyer.org>

Première rédaction de cet article le 30 septembre 2019

https://www.bortzmeyer.org/doh-mon-resolveur.html

Cet article documente le fonctionnement interne du résolveur DoH https://doh.bortzmeyer.fr/ et du résolveur DoT dot.bortzmeyer.fr. Il concerne donc essentiellement les technicien[Caractère Unicode non montré [1]]ne[Caractère Unicode non montré]s. Si vous vous intéressez plutôt aux conditions d'utilisation de ce service, voyez l'article décrivant la politique <https://www.bortzmeyer.org/doh-bortzmeyer-fr-policy.html>.

Pour comprendre les protocoles DoH et DoT, il faut relire les normes qui les décrivent, respectivement les RFC 8484 [2] et RFC 7858.

C'est le même résolveur qui gère les deux protocoles. Si une version initiale de ce résolveur DoH utilisait une mise en œuvre en Python que j'avais écrite lors d'un hackathon de l'IETF <https://www.bortzmeyer.org/hackathon-ietf-101.html>, la version « de production » se sert de l'excellent logiciel dnsdist <https://dnsdist.org/>. dnsdist est un frontal pour serveurs DNS, assurant des fonctions telles que la répartition de charge et, ce qui nous intéresse surtout ici, le relais entre différents protocoles. Ici, dnsdist a été configuré pour accepter des connexions DoH et DoT (mais **pas** le traditionnel DNS sur UDP, ni sur TCP en clair, d'ailleurs) et pour relayer les requêtes DNS vers le « vrai » résolveur.

La machine est une Arch Linux tournant chez OVH, sur l'offre nommée « VPS ».

La configuration de dnsdist se trouve dans un fichier de configuration, dnsdist.conf. N'hésitez pas à consulter la documentation très complète de dnsdist <https://dnsdist.org/reference/config.html>. Voyons les points essentiels.

On fait du DoH donc on lance un service DoH, en IPv4 et en IPv6 :

1. Car trop difficile à faire afficher par LaTeX
2. Pour voir le RFC de numéro NNN, https://www.ietf.org/rfc/rfcNNN.txt, par exemple https://www.ietf.org/rfc/rfc8484.txt

```
addDOHLocal("0.0.0.0:443", "/etc/dnsdist/server-doh.pem", "/etc/dnsdist/server-doh.key", {"/", "/rfc", "/abo
addDOHLocal("[::]:443", "/etc/dnsdist/server-doh.pem", "/etc/dnsdist/server-doh.key", {"/", "/rfc", "/about"
```

Pour améliorer un peu la sécurité, on exige du TLS 1.2 au minimum. Notez qu'il y aurait plein d'autres paramètres à configurer (refuser des algorithmes trop faibles), pour avoir une meilleure note sur SSLlabs <https://www.ssllabs.com/ssltest/analyze.html?d=doh.bortzmeyer.fr&latest>. Et les trucs bizarres qui commencent par `customResponseHeaders` ? Il s'agit d'utiliser la technique du RFC 8631 pour indiquer des méta-informations sur le service, ici, la politique suivie. Cela donne :

```
% curl -v https://doh.bortzmeyer.fr/help
...
< HTTP/2 200
< server: h2o/dnsdist
< link: <https://www.bortzmeyer.org/doh-bortzmeyer-fr-policy.html> rel="service-meta"; type="text/html"
< content-length: 86
<
```

Pour avoir des URL « spéciaux » ne faisant pas du DoH, on ajoute aussi :

```
supportpagemap = { newDOHResponseMapEntry("^/rfc$", 307, "https://www.rfc-editor.org/info/rfc8484"),
                   newDOHResponseMapEntry("^/about$", 307, "https://www.bortzmeyer.org/doh-bortzmeyer-fr-pol
                   newDOHResponseMapEntry("^/policy$", 307, "https://www.bortzmeyer.org/doh-bortzmeyer-fr-po
   newDOHResponseMapEntry("^/help$", 200, "For the server policy, see <https://www.bortzmeyer.org/doh-bortzm
dohFE = getDOHFrontend(0)
dohFE:setResponsesMap(supportpagemap)
dohFE6 = getDOHFrontend(1)
dohFE6:setResponsesMap(supportpagemap)
```

Ces instructions disent à dnsdist de rediriger `/policy` vers l'article décrivant la politique <https://www.bortzmeyer.org/doh-bortzmeyer-fr-policy.html>, et d'afficher un court message pour `/help`.

Et on veut faire du DoT, pas seulement du DoH, donc on met aussi :

```
addTLSLocal("0.0.0.0:853", "/etc/dnsdist/server-dot.pem", "/etc/dnsdist/server-dot.key", {minTLSVersion="tls
addTLSLocal("[::]:853", "/etc/dnsdist/server-dot.pem", "/etc/dnsdist/server-dot.key", {minTLSVersion="tls1.2
```

Le résolveur est public, donc les ACL autorisent tout le monde :

```
addACL('0.0.0.0/0')
addACL('[::]/0')
```

Mais on se méfie quand même des clients trop enthousiastes donc on les limite à cent requêtes par seconde :

```
addAction(MaxQPSIPRule(100), DropAction())
```

Au passage, il faut dire un mot de `addAction` car il sert souvent. dnsdist permet de définir des politiques à appliquer lors du traitement d'une requête DNS (la documentation dit « paquet », mais, apparemment, il s'agit bien d'une requête). La syntaxe générale (cf. la documentation <https://dnsdist.org/rules-actions.html>) est `addAction(critère, action)` avec de nombreux critères possibles (ici, « a fait plus de 100 r/s ») et plein d'actions disponibles (ici, ignorer la requête).

dnsdist est un répartiteur de charge, il n'est pas un résolveur DNS. Il faut donc un ou plusieurs « vrais » résolveurs derrière. Le principal résolveur, pour le service `https://doh.bortzmeyer.fr`, est un Unbound qui tourne sur la même machine. Au cas où quelque chose irait mal, un second résolveur, complètement indépendant, est fourni par OVH. dnsdist permet de choisir quel résolveur utiliser <https://dnsdist.org/guides/serverselection.html> et j'ai mis :

```
setServerPolicy(firstAvailable)
newServer({address="127.0.0.1:53", name="Local-Unbound"})
newServer({address="213.186.33.99:53", name="OVH"})
```

Pourquoi `firstAvailable` ? Parce que je voulais éviter autant que possible d'envoyer des requêtes à ce second résolveur que je ne contrôle pas et dont je ne connais pas la politique en matière de vie privée. Comme voulu, la quasi-totalité des requêtes arrivent donc au premier résolveur :

```
> showServers()
#   Name                Address                         State     Qps    Qlim Ord Wt    Queries   Drops Drate   L
0   Local-Unbound       127.0.0.1:53                       up     0.0       0   1  1       2509       0   0.0  70
1   OVH                 213.186.33.99:53                   up     0.0       0   1  1          0       0   0.0   0
All                                                            0.0                       2509       0
```

Pour pouvoir utiliser la console de dnsdist, comme ci-dessus, il faut l'activer dans le fichier de configuration :

```
controlSocket('[::1]:5199')
setKey("kg...=")
```

(La documentation explique <https://dnsdist.org/guides/console.html> quelle valeur indiquer à `setKey()`.) L'accès à la console se fait ensuite avec `dnsdist -c` :

```
%  dnsdist -c
> dumpStats()
acl-drops                        0    latency0-1                    5058
cache-hits                    4429    latency1-10                    351
cache-misses                  2571    latency10-50                   785
cpu-sys-msec                 12929    latency100-1000                388
cpu-user-msec                47305    latency50-100                  280
...
```

dnsdist dispose également d'un serveur Web minimal, permettant de regarder l'état du service (ce n'est pas un site Web d'administration, c'est juste pour jeter un coup d'œil). Voici une configuration typique :

```
webserver("[::1]:8082", "2e...", "a5..")
```

Les deux derniers paramètres indiquent le mot de passe à utiliser pour les accès au site Web et la clé pour l'API. Ici, le serveur n'écoute que sur une adresse locale. Si vous voulez le rendre accessible de l'extérieur, comme il ne gère pas HTTPS, il vaut mieux le mettre derrière un relais. J'ai utilisé stunnel, avec cette configuration :

```
[dnsdist]
accept  = 8083
connect = localhost-ipv6:8082
cert = /etc/stunnel/stunnel.pem
key = /etc/stunnel/stunnel.key
```

Cela me permet de regarder de l'extérieur :

Et pour l'API de ce serveur interne <https://dnsdist.org/guides/webserver.html#dnsdist-api>, qui renvoie des résultats en JSON :

```
% curl -s --header "X-API-Key: XXXX" https://doh.bortzmeyer.fr:8083/api/v1/servers/localhost/statistics | jq
[
  {
    "name": "responses",
    "type": "StatisticItem",
    "value": 2643
  },
  {
    "name": "queries",
    "type": "StatisticItem",
    "value": 3698
  },
...
```

Il y a aussi quelques paramètres liés aux performances du serveur. Celui-ci active la mémoire de dnsdist, qui gardera au maximum 100 000 enregistrements DNS :

```
pc = newPacketCache(100000)
getPool(""):setCache(pc)
```

Pour l'instant, cette valeur est énorme pour ce modeste serveur. Dans la console :

```
> getPool(""):getCache():printStats()
Entries: 84/100000
Hits: 1127
Misses: 2943
...
```

Cette mémoire explique pourquoi, plus haut, le serveur indiquait davantage de requêtes que de réponses (il ne compte comme réponse que ce qui a été traité par les vrais résolveurs <https://dnsdist.org/statistics.html>). Autres réglages, portant sur le réseau <https://dnsdist.org/advanced/tuning.html>, qu'il me reste à ajuster dès que le serveur recevra du trafic abondant :

```
setMaxUDPOutstanding(65535)     -- Nombre maximum de requêtes en attente pour un résolveur
setMaxTCPClientThreads(30)  -- Nombre maximum de fils d'exécution TCP (chacun pouvant traiter plusieurs clients)
setMaxTCPConnectionDuration(1800) -- Après trente minutes, on raccroche
setMaxTCPQueriesPerConnection(300) -- Après trois cents requêtes, on raccroche
setMaxTCPConnectionsPerClient(10) -- Dix connexions pour un seul client, c'est déjà beaucoup, mais il faut pense
```

dnsdist permet d'enregistrer chaque requête DNS, par exemple dans un fichier. Cette fonction n'est **pas** activée sur `https://doh.bortzmeyer.fr` car elle serait contraire aux promesses faites quant au respect de la vie privée <https://www.bortzmeyer.org/doh-bortzmeyer-fr-policy.html>. Mais c'est un bon exemple d'utilisation de `addAction` pour **toutes** les requêtes (`AllRule`) :

```
-- addAction(AllRule(), LogAction("/tmp/dnsdist.log", false, true, false))
```

(Attention si vous utilisez systemd, avec l'option `PrivateTmp=true`, le fichier journal sera mis quelque part sous `/tmp/systemd-private-XXXXXXX-dnsdist.service-Ijy8yw`, pour être plus difficile à trouver.) Les lignes journalisées sont du genre :

```
Packet from [2001:db8::a36b::64]:44364 for www.ietf.org. AAAA with id 8283
```

Pour la même raison de vie privée, je n'utilise pas la TeeAction <https://dnsdist.org/advanced/teeaction.html> (qui permet de copier les requêtes ailleurs) ou dnstap <http://dnstap.info/>.

De même qu'on ne journalise pas, on ne ment pas <https://www.bortzmeyer.org/doh-bortzmeyer-fr-polic
html>. dnsdist peut le faire via `SpoofAction()` ou bien avec un script Lua spécifique, pour faire des choses horribles comme bloquer un type de données particulier :

```
luarule(dq) if (dq.qtype==dnsdist.NAPTR) then return DNSAction.Nxdomain, "" else return DNSAction.Allow, "" end
addLuaAction(AllRule(), luarule)
```

DoH et DoT fonctionnent tous les deux sur TLS (RFC 8446) et utilisent donc son mécanisme d'authentification via des certificats PKIX (RFC 5280). Il faut donc obtenir des certificats raisonnablement reconnus par les clients donc, comme tout le monde, j'ai utilisé Let's Encrypt. dnsdist n'inclut pas de client ACME (RFC 8555), j'ai donc utilisé certbot <https://certbot.eff.org/> que je fais tourner en mode autonome ("*standalone*"). Cela marche car dnsdist n'écoute que sur le port 443 alors qu'ACME n'utilise que le 80. Pour créer le certificat initial :

```
certbot certonly -n --standalone --domain doh.bortzmeyer.fr
```

Et pour le renouveler :

```
certbot renew --standalone --reuse-key --deploy-hook /usr/local/sbin/restart-dnsdist
```

(`restart-dnsdist` contient `dnsdist -e 'reloadAllCertificates()'`.) Et voici les liens symboliques configurés pour pointer vers les répertoires utilisés par Let's Encrypt :

```
% ls -lt
total 28
lrwxrwxrwx  1 root root   51 Sep 24 15:50 server-dot.key -> /etc/letsencrypt/live/dot.bortzmeyer.fr/privkey.
lrwxrwxrwx  1 root root   53 Sep 24 15:50 server-dot.pem -> /etc/letsencrypt/live/dot.bortzmeyer.fr/fullchai
lrwxrwxrwx  1 root root   51 Sep 15 16:31 server-doh.key -> /etc/letsencrypt/live/doh.bortzmeyer.fr/privkey.
lrwxrwxrwx  1 root root   53 Sep 15 16:31 server-doh.pem -> /etc/letsencrypt/live/doh.bortzmeyer.fr/fullchai
...
```

Avec gnutls-cli, on peut voir le certificat utilisé :

```
% gnutls-cli dot.bortzmeyer.fr:853
...
- Certificate type: X.509
- Got a certificate list of 2 certificates.
- Certificate[0] info:
 - subject 'CN=dot.bortzmeyer.fr', issuer 'CN=Let's Encrypt Authority X3,O=Let's Encrypt,C=US', serial 0x047
Public Key ID:
sha1:696c94f0f093a3b1037ffcb2fcd9d23859d539bc
sha256:787005b3173d1c95bc425241ea40e5474b644f00fded7fd35d87350331644c56
Public key's random art:
+--[ RSA 2048]----+
|      o          |
|       = o       |
|    . . O     ...|
|     o B +    .+.|
|      = S    .  o|
|       = .  .  E |
|        . .=     |
|       . o=o.    |
|        o.oo.    |
+-----------------+

- Certificate[1] info:
 - subject 'CN=Let's Encrypt Authority X3,O=Let's Encrypt,C=US', issuer 'CN=DST Root CA X3,O=Digital Signatu
- Status: The certificate is trusted.
- Description: (TLS1.2)-(ECDHE-RSA-SECP256R1)-(AES-256-GCM)
...
```

Pour pouvoir authentifier le serveur par d'autres moyens que la chaîne de confiance PKIX (par exemple par épinglage de la clé, ou par DANE), je demande à certbot de ne pas changer la clé lors des renouvellements de certificats, avec l'option `--reuse-key` (cf. mon précédent article sur la question <https://www.bortzmeyer.org/letsencrypt-certbot-keep-key.html>).

À propos de DANE (RFC 6698), j'ai créé les enregistrements nécessaires avec hash-slinger <http://people.redhat.com/pwouters/hash-slinger/> :

```
% tlsa --create --selector 1  --port 853 dot.bortzmeyer.fr
Got a certificate with Subject: /CN=dot.bortzmeyer.fr
_853._tcp.dot.bortzmeyer.fr. IN TLSA 3 1 1 787005b3173d1c95bc425241ea40e5474b644f00fded7fd35d87350331644c56
```

Puis je les mets dans le fichier de zone DNS, et on peut voir les enregistrements avec dig :

```
% dig TLSA _853._tcp.dot.bortzmeyer.fr
...
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 15141
;; flags: qr rd ra ad; QUERY: 1, ANSWER: 2, AUTHORITY: 4, ADDITIONAL: 11
...
;; ANSWER SECTION:
_853._tcp.dot.bortzmeyer.fr. 86400 IN TLSA 3 1 1 (
787005B3173D1C95BC425241EA40E5474B644F00FDED
```

Et on peut les vérifier :

```
% tlsa --verify --resolvconf="" doh.bortzmeyer.fr
SUCCESS (Usage 3 [DANE-EE]): Certificate offered by the server matches the TLSA record (193.70.85.11)
SUCCESS (Usage 3 [DANE-EE]): Certificate offered by the server matches the TLSA record (2001:41d0:302:2200::180)
```

Et superviser le tout depuis Icinga <https://www.bortzmeyer.org/monitor-dane.html>.

On a vu qu'il y avait deux résolveurs DNS derrière dnsdist, le principal, géré par moi, et un résolveur de secours. Le résolveur principal utilise Unbound. Pour mieux protéger la vie privée, il utilise la "*QNAME minimisation*" du RFC 7816. Dans le `unbound.conf`, on a :

```
server:
  qname-minimisation: yes
```

Vous pouvez vérifier que c'est bien activé avec le domaine de test `qnamemintest.internet.nl` et l'outil test-doh présenté plus loin :

```
% test-doh https://doh.bortzmeyer.fr qnamemintest.internet.nl TXT
...
a.b.qnamemin-test.internet.nl. 10 IN TXT "HOORAY - QNAME minimisation is enabled on your resolver :)!"
```

Alors qu'avec un autre résolveur DoH, moins soucieux de vie privée :

```
% test-doh https://dns.google/dns-query qnamemintest.internet.nl TXT
...
a.b.qnamemin-test.internet.nl. 9 IN TXT "NO - QNAME minimisation is NOT enabled on your resolver :("
```

J'ai aussi mis quelques autres paramètres Unbound, typiquement pour durcir la résolution face à diverses menaces :

```
  harden-glue: yes
  harden-below-nxdomain: yes
  harden-dnssec-stripped: yes
  harden-referral-path: yes
  aggressive-nsec: yes
```

Avec tout ça, je crois ne pas avoir dit comment j'avais installé dnsdist. Comme il n'existe pas de paquetage dnsdist dans Arch Linux (mais il existe dans AUR <https://aur.archlinux.org/packages/dnsdist/>), j'ai téléchargé le source et compilé moi-même, selon les instructions <https://dnsdist.org/install.html>. D'abord, j'ai téléchargé et installé la bibliothèque libh2o <https://h2o.example.net/>, qui permet à dnsdist de parler HTTP/2 (RFC 7540) :

```
cmake .
make
make install
```

Puis on peut installer dnsdist :

```
PKG_CONFIG_PATH=/usr/local/lib/pkgconfig  ./configure --enable-dns-over-tls --enable-dns-over-https
make
make install
```

Testons maintenant, d'abord avec kdig (qui fait partie de Knot, et notez la première ligne de la réponse) :

```
% kdig  +tls @dot.bortzmeyer.fr nextinpact.com
;; TLS session (TLS1.3)-(ECDHE-SECP256R1)-(RSA-PSS-RSAE-SHA256)-(AES-256-GCM)
;; ->>HEADER<<- opcode: QUERY; status: NOERROR; id: 56403
;; Flags: qr rd ra; QUERY: 1; ANSWER: 2; AUTHORITY: 0; ADDITIONAL: 1

;; EDNS PSEUDOSECTION:
;; Version: 0; flags: ; UDP size: 4096 B; ext-rcode: NOERROR

;; QUESTION SECTION:
;; nextinpact.com.      IN A

;; ANSWER SECTION:
nextinpact.com.      118 IN A 45.60.122.203
nextinpact.com.      118 IN A 45.60.132.203

;; Received 75 B
;; Time 2019-10-03 17:34:29 CEST
;; From 2001:41d0:302:2200::180@853(TCP) in 14.1 ms
```

Puis grâce aux sondes RIPE Atlas <https://atlas.ripe.net/>, le service. Les Atlas savent faire du DoT <https://www.bortzmeyer.org/dns-over-tls-atlas-measures.html>. Voyons d'abord en IPv6 (le choix par défaut) :

```
%  blaeu-resolve --dnssec --displayvalidation --displayrtt --tls --nameserver=dot.bortzmeyer.fr --nsid --sort --
Nameserver dot.bortzmeyer.fr
[ (Authentic Data flag)  2001:4b98:dc0:41:216:3eff:fe27:3d3f] : 956 occurrences Average RTT 1572 ms
[TIMEOUT] : 17 occurrences Average RTT 0 ms
[TUCONNECT (may be a TLS negotiation error)] : 14 occurrences Average RTT 0 ms
Test #22928943 done at 2019-09-30T09:37:15Z
```

Il y a quand même trop d'échecs. Ceci dit, il s'agit parfois de problèmes réseau et pas de problèmes DoT. Essayons ICMP Echo avec les mêmes sondes :

```
% blaeu-reach --old_measurement=22928943 --by_probe 2001:41d0:302:2200::180

493 probes reported
Test #22929082 done at 2019-09-30T09:48:25Z
Tests: 487 successful probes (98.8 %), 6 failed (1.2 %), average RTT: 72 ms
```

Donc, au moins une partie des problèmes n'est pas spécifique à DoT. Pour les autres cas d'échec, on peut aussi imaginer que certains réseaux ne laissent pas sortir les communications vers le port 853, qu'utilise DoT (c'est d'ailleurs une des principales raisons qui a motivé le développement de DoH qui, utilisant HTTPS, est plus difficile à bloquer). En attendant, essayons en IPv4 :

```
%  blaeu-resolve --dnssec --displayvalidation --displayrtt --tls --nameserver=dot.bortzmeyer.fr --nsid --sort --
Nameserver dot.bortzmeyer.fr
[ (Authentic Data flag)  2001:4b98:dc0:41:216:3eff:fe27:3d3f] : 971 occurrences Average RTT 975 ms
[TIMEOUT] : 9 occurrences Average RTT 0 ms
[TUCONNECT (may be a TLS negotiation error)] : 6 occurrences Average RTT 0 ms
Test #22929087 done at 2019-09-30T09:51:59Z

% blaeu-reach --old_measurement=22929087  --by_probe  193.70.85.11
494 probes reported
Test #22929254 done at 2019-09-30T11:35:16Z
Tests: 494 successful probes (100.0 %), 0 failed (0.0 %), average RTT: 78 ms
```

C'est clair, on a moins d'erreurs. L'Internet est hélas moins fiable en IPv6 (surtout vu les difficultés de configuration d'IPv6 chez OVH <https://www.bortzmeyer.org/ipv6-archlinux-ovh.html>).

J'ai utilisé plus haut le script `test-doh` pour tester le serveur. Ce script est écrit en Python et disponible en (en ligne sur `https://www.bortzmeyer.org/files/test-doh.py`). Exemple d'utilisation :

```
% test-doh https://doh.bortzmeyer.fr netflix.com AAAA
id 0
opcode QUERY
rcode NOERROR
flags QR RD RA
;QUESTION
netflix.com. IN AAAA
;ANSWER
netflix.com. 60 IN AAAA 2a01:578:3::3431:7806
netflix.com. 60 IN AAAA 2a01:578:3::22fd:6807
netflix.com. 60 IN AAAA 2a01:578:3::36e5:444d
...
;AUTHORITY
;ADDITIONAL
```

On peut aussi utiliser (en ligne sur `https://www.bortzmeyer.org/files/doh-client.sh`), qui appelle curl, et utilise deux programmes Python, (en ligne sur `https://www.bortzmeyer.org/files/dns-t2b.py`) et (en ligne sur `https://www.bortzmeyer.org/files/dns-b2t.py`) pour créer les messages DNS en entrée et les lire à la sortie (DoH n'utilise pas JSON ou XML, mais le format binaire du DNS.)

Le service est supervisé par Icinga. Pour DoT, je me sers d'un programme développé lors d'un hackathon IETF <`https://www.bortzmeyer.org/monitor-dns-over-tls.html`>. (Il y a depuis une version plus propre dans getdns <`https://getdnsapi.net/`>, le programme dans le paquetage Debian `getdns-utils` se nomme `getdns_server_mon`.) Lancé à la main, il donne :

```
 % /usr/local/lib/nagios/plugins/check_dns_with_getdns -H 2001:41d0:302:2200::180 -n nextinpact.com
GETDNS OK - 16 ms, expiration date 2019-12-23, auth. None:  Address 45.60.132.203 Address 45.60.122.203
```

Pour DoH, certains des tests sont les tests génériques des "*monitoring plugins*" <`https://www.monitoring-plugins.org/`>, par exemple le test de l'expiration du certificat <`https://www.bortzmeyer.org/tester-expiration-certifs.html`> (pour ne pas être surpris si les renouvellements Let's Encrypt se sont mal passés). Cela se configure avec :

```
vars.http_vhosts["doh-http"] = {
    http_uri = "/"
    http_vhost = "doh.bortzmeyer.fr"
    http_ssl = true
    http_sni = true
    http_timeout = 15
    http_certificate = "4,2"
    }
```

Ainsi, je suis prévenu s'il reste moins de quatre jours au certificat (et une alarme est levée s'il ne reste que deux jours). Autre test générique, que le serveur renvoie bien un échec si on ne lui parle pas en DoH correctement :

```
vars.http_vhosts["doh-raw-http"] = {
    http_uri = "/"
    http_vhost = "doh.bortzmeyer.fr"
    http_ssl = true
    http_sni = true
    http_timeout = 15
    http_expect = 400
    http_string = "Unable to parse the request"
    }
```

Mais il faut évidemment tester que le serveur répond bien aux requêtes DoH. On utilise pour cela un script dérivé du `test-doh` cité plus haut, (en ligne sur `https://www.bortzmeyer.org/files/check_doh.py`). Utilisé à la main, ça donne :

```
% /usr/local/lib/nagios/plugins/check_doh -H doh.bortzmeyer.fr -n nextinpact.com
https://doh.bortzmeyer.fr/ OK - No error for nextinpact.com/AAAA, 107 bytes received
```

Et pour l'intégrer dans Icinga, on déclare une commande dans `commands.conf` :

```
object CheckCommand "doh_monitor" {
  command = [ PluginContribDir + "/check_doh" ]

  arguments = {
          "-H" = "$address6$",
          "-n" = "$doh_lookup$",
          "-p" = "$doh_path$",
          "-V" = "$doh_vhost$",
          "-t" = "$doh_type$",
          "-p" = "$doh_post$",
          "-i" = "$doh_insecure$",
          "-h" = "$doh_head$"
          }
}
```

Puis un service dans `services.conf` :

```
Apply Service "doh" {
  import "generic-service"

  check_command = "doh_monitor"
    assign where (host.address || host.address6) && host.vars.doh
}
```

On peut alors configurer la machine à superviser :

```
     vars.doh = true
     vars.doh_vhost = "doh.bortzmeyer.fr"
     vars.doh_lookup = "fr.wikipedia.org"
     vars.doh_post = true
```

On supervise également le serveur Web internet :

```
vars.http_vhosts["doh-admin"] = {
    http_uri = "/api/v1/servers/localhost"
    http_port = 8083
    http_vhost = "doh.bortzmeyer.fr"
    http_ssl = true
    http_sni = true
    http_timeout = 15
    http_header = "X-API-Key: 23b..."
    http_string = "dohFrontends"
    }
}
```

Il existe d'autres résolveurs DoH gérés par des associations ou des individus, comme celui du `.cz`, `https://www.nic.cz/odvr/` ou comme celui de l'association 42l <`https://42l.fr/DoH-service`>. Mais je n'en trouve pas (pour l'instant) qui ait documenté en détail leur configuration technique.

Sinon, si vous aimez lire, il y a les très bons supports <`https://www.shaftinc.fr/misc/dot-doh-dns-chiffre pdf`> de l'exposé de Shaft à Pas Sage En Seine (plein de DoT et un peu de DoH.)
