RFC 8484 : DNS Queries over HTTPS (DoH)

Stéphane Bortzmeyer
<stephane+blog@bortzmeyer.org>

Première rédaction de cet article le 22 octobre 2018

Date de publication du RFC : Octobre 2018

https://www.bortzmeyer.org/8484.html

Voici un nouveau moyen d'envoyer des requêtes DNS, DoH ("*DNS over HTTPS*"). Requêtes et réponses, au lieu de voyager directement sur UDP ou TCP sont encapsulées dans HTTP, plus exactement HTTPS. Le but ? Il s'agit essentiellement de contourner la censure, en fournissant un canal sécurisé avec un serveur supposé digne de confiance. Et le chiffrement sert également à préserver la vie privée du client. Toutes ces fonctions pourraient être assurées en mettant le DNS sur TLS (RFC 7858 [1]) mais DoH augmente les chances de succès puisque le trafic HTTPS est rarement bloqué par les pare-feux, alors que le port 853 utilisé par DNS-sur-TLS peut être inaccessible, vu le nombre de violations de la neutralité <https://www.bortzmeyer.org/neutralite.html> du réseau. DoH marque donc une nouvelle étape dans la transition vers un Internet « port 443 seulement ».

La section 1 du RFC détaille les buts de DoH. Deux buts principaux sont décrits :

— Le premier, et le plus important, est celui indiqué au paragraphe précédent : échapper aux "*middleboxes*" qui bloquent le trafic DNS, ou bien le modifient. De telles violations de la neutralité <https://www.bortzmeyer.org/neutralite.html> du réseau sont fréquentes <https://labs.ripe.net/Members/stephane_bortzmeyer/dns-censorship-dns-lies-seen-by-atlas-pr imposées par les États ou les entreprises à des fins de censure, ou bien décidées par les FAI pour des raisons commerciales. Dans la lutte sans fin entre l'épée et la cuirasse, le DNS est souvent le maillon faible : infrastructure indispensable, il offre une cible tentante aux attaquants. Et il est utilisable pour la surveillance (cf. RFC 7626). DNS-sur-TLS, normalisé dans le RFC 7858, était une première tentative de protéger l'intégrité et la confidentialité du trafic DNS par la cryptographie. Mais il utilise un port dédié, le port 853, qui peut être bloqué par un intermédiaire peu soucieux de neutralité du réseau. Dans de nombreux réseaux, hélas, le seul protocole qui soit à peu près sûr de passer partout est HTTPS sur le port 443. D'où la tendance actuelle à tout mettre sur HTTP.
— Il y a une deuxième motivation à DoH, moins importante, la possibilité de faire des requêtes DNS complètes (pas seulement des demandes d'adresses IP) depuis des applications JavaScript tournant dans le navigateur, et tout en respectant CORS.

1. Pour voir le RFC de numéro NNN, https://www.ietf.org/rfc/rfcNNN.txt, par exemple https://www.ietf.org/rfc/rfc7858.txt

L'annexe A de notre RFC raconte le cahier des charges du protocole DoH de manière plus détaillée :

— Sémantique habituelle de HTTP (on ne change pas HTTP),
— Possibilité d'exprimer la totalité des requêtes et réponses DNS actuelles, d'où le choix de l'encodage binaire du DNS et pas d'un nouvel encodage, par exemple en JSON, plus simple mais limité à un sous-ensemble du DNS,
— Et aussi des « non-considérations » : DoH n'avait pas à traiter le cas des réponses synthétisées, comme DNS64, ni celui des réponses à la tête du client, ni le HTTP tout nu sans TLS.

Passons maintenant à la technique. DoH est un protocole très simple. Au hackathon de l'IETF en mars 2018 à Londres, les sept ou huit personnes travaillant sur DoH avaient très vite réussi à créer clients et serveurs, et à les faire interopérer (même moi, j'y étais arrivé <https://www.bortzmeyer.org/hackathon-ietf-101.html>). Vous pouvez lire le compte-rendu du hackathon <https://github.com/IETF-Hackathon/ietf101-project-presentations/tree/master/DoH>, et la présentation quelques jours après <https://datatracker.ietf.org/meeting/101/materials/slides-101-doh-hackathon-feedback-01> au groupe de travail DoH.)

DoH peut s'utiliser de plusieurs façons : c'est une technique, pas une politique. Néanmoins, son principal usage sera entre un résolveur simple, situé sur la machine de l'utilisateur ou quelque part dans son réseau local, et un résolveur complet situé plus loin dans le réseau (section 1 du RFC). Ce résolveur simple peut être un démon comme stubby <https://dnsprivacy.org/wiki/display/DP/DNS+Privacy+Daemon+-+Stubby> ou systemd, ou bien l'application elle-même (ce qui me semble personnellement une mauvaise idée, car cela empêche de partager configuration et cache). À l'heure actuelle, les serveurs faisant autorité ne parlent pas DoH et il n'est pas prévu qu'ils s'y mettent à brève échéance. Le schéma suivant montre l'utilisation typique de DoH : l'application (le client final) parle à un résolveur simple, en utilisant le protocole DNS (ce qui évite de mettre du DoH dans toutes les applications), le résolveur simple parlera en DoH avec un résolveur de confiance situé quelque part dans l'Internet, et ce résolveur de confiance utilisera le DNS pour parler aux serveurs faisant autorité (soit il sera lui-même un résolveur complet, soit il parlera à un résolveur classique proche de lui, et le serveur DoH sera alors un "*proxy*" comme décrit dans le RFC 5625) :

Les requêtes et réponses DNS (RFC 1034 et RFC 1035) ont leur encodage habituel (le DNS est un protocole binaire, donc je ne peux pas faire de copier/coller pour montrer cet encodage), la requête est mise dans le chemin dans l'URL ou dans le corps d'une requête HTTP (RFC 9113), la réponse se trouvera dans le corps de la réponse HTTP. Toute la sécurité (intégrité et confidentialité) est assurée par TLS (RFC 8446), via HTTPS (RFC 2818). Un principe essentiel de DoH est d'utiliser HTTP tel quel, avec ses avantages et ses inconvénients. Cela permet de récupérer des services HTTP comme la négociation de contenu, la mise en cache, l'authentification, les redirections, etc.

La requête HTTP elle-même se fait avec les méthodes GET ou POST (section 4 du RFC), les deux devant être acceptées (ce qui fut le sujet d'une assez longue discussion à l'IETF.) Quand la méthode utilisée est GET, la variable nommée `dns` est le contenu de la requête DNS, suivant l'encodage habituel du DNS, surencodée en Base64, plus exactement la variante base64url normalisée dans le RFC 4648. Et, avec GET, le corps de la requête est vide (RFC 7231, section 4.3.1). Quand on utilise POST, la requête DNS est dans le corps de la requête HTTP et n'a pas ce surencodage. Ainsi, la requête avec POST sera sans doute plus petite, mais par contre GET est certainement plus apprécié par les caches.

On l'a dit, DoH utilise le HTTP habituel. L'utilisation de HTTP/2, la version 2 de HTTP (celle du RFC 9113) est très recommandée, et clients et serveurs DoH peuvent utiliser la compression et le remplissage que fournit HTTP/2 (le remplissage étant très souhaitable pour la vie privée). HTTP/2 a également l'avantage de multiplexer plusieurs ruisseaux ("*streams*") sur la même connexion HTTP, ce qui évite aux requêtes DoH rapides de devoir attendre le résultat d'une requête lente qui aurait été émise avant. (HTTP 1, lui, impose le respect de l'ordre des requêtes.) HTTP/2 n'est pas formellement imposé, car on ne peut pas forcément être sûr du comportement des bibliothèques utilisées, ni de celui des différents relais sur le trajet.

Requêtes et réponses ont actuellement le type MIME `application/dns-message`, mais d'autres types pourront apparaitre dans le futur (par exemple fondés sur JSON et non plus sur l'encodage binaire du DNS, qui n'est pas amusant à analyser en JavaScript). Le client DoH doit donc inclure un en-tête HTTP `Accept:` pour indiquer quels types MIME il accepte. En utilisant HTTP, DoH bénéfice également de la négociation de contenu HTTP (RFC 7231, section 3.4).

Petit détail DNS : le champ ID (identification de la requête) doit être mis à zéro. Avec UDP, il sert à faire correspondre une requête et sa réponse mais c'est inutile avec HTTP, et cela risquerait d'empêcher la mise en cache de deux réponses identiques mais avec des ID différents.

En suivant la présentation des requêtes HTTP du RFC 9113 (rappelez-vous que HTTP/2, contrairement à la première version de HTTP, a un encodage binaire), cela donnerait, par exemple :

```
:method = GET
:scheme = https
:authority = dns.example.net
:path = /?dns=AAABAAABAAAAAAAAA3d3dwdleGFtcGxlA2NvbQAAAQAB
accept = application/dns-message
```

(Si vous vous le demandez, `AAABAAABAAAAAAAAA3d3dwdleGFtcGxlA2NvbQAAAQAB` est une requête DNS pour l'adresse IPv4 de `www.example.com`.)

Et la réponse HTTP ? Aujourd'hui, elle est forcément de type MIME `application/dns-message`, mais d'autres types pourront apparaitre. En attendant, le corps de la réponse HTTP est une réponse DNS avec son encodage binaire habituel normalisé dans le RFC 1035, section 4.1 (tel qu'utilisé pour UDP ; notez que HTTP permettant d'indiquer la longueur du message, les deux octets de longueur utilisés par le DNS au-dessus de TCP ne sont pas nécessaires et sont donc absents).

Le serveur DoH doit mettre un code de retour HTTP (RFC 7231). 200 signifie que la requête HTTP a bien été traitée. Mais cela ne veut pas dire que la requête DNS, elle, ait connu un succès. Si elle a obtenu une erreur DNS NXDOMAIN (nom non trouvé) ou SERVFAIL (échec de la requête), le code de retour HTTP sera quand même 200, indiquant qu'il y a une réponse DNS, même négative. Le client DoH, en recevant ce 200, devra donc analyser le message DNS et y trouver le code de retour DNS (NOERROR, NXDOMAIN, REFUSED, etc). Le serveur DoH ne mettra un code d'erreur HTTP que s'il n'a pas du tout de réponse DNS à renvoyer. Il mettra 403 s'il refuse de servir ce client DoH, 429 si le client fait trop de requêtes (RFC 6585), 500 si le serveur DoH a une grosse bogue, 415 si le type MIME utilisé n'est pas connu du serveur, et bien sûr 404 si le serveur HTTP ne trouve rien à l'URL indiqué par exemple parce que le service a été retiré. Dans tous ces cas, il n'y a **pas** de réponse DNS incluse. La sémantique de ces codes de retour, et le comportement attendu du client, suit les règles habituelles de HTTP, sans spécificité DoH. (C'est un point important et général de DoH : c'est du DNS normal sur du HTTP normal). Par exemple, lorsque le code de retour commence par un 4, le client n'est pas censé réessayer la même requête sur le même serveur : elle donnera forcément le même résultat.

Voici un exemple de réponse DoH :

```
:status = 200
content-type = application/dns-message
content-length = 61
cache-control = max-age=3709
[Les 61 octets, ici représentés en hexadécimal pour la lisibilité]
   00 00 81 80 00 01 00 01  00 00 00 00 03 77 77 77
   07 65 78 61 6d 70 6c 65  03 63 6f 6d 00 00 1c 00
   01 c0 0c 00 1c 00 01 00  00 0e 7d 00 10 20 01 0d
   b8 ab cd 00 12 00 01 00  02 00 03 00 04
```

La réponse DNS signifie « l'adresse de `www.example.com` est `2001:db8:abcd:12:1:2:3:4` et le TTL est de 3709 secondes [notez comme il est repris dans le `Cache-control:` HTTP] ».

Comment un client DoH trouve-t-il le serveur ? La section 3 du RFC répond à cette question. En gros, c'est manuel. DoH ne fournit pas de mécanisme de sélection automatique. Concevoir un tel mécanisme et qu'il soit sécurisé est une question non triviale, et importante : changer le résolveur DNS utilisé par une machine revient quasiment à pirater complètement cette machine. Cela avait fait une très longue discussion au sein du groupe de travail DoH à l'IETF, entre ceux qui pensaient qu'un mécanisme automatique de découverte du gabarit faciliterait nettement la vie de l'utilisateur, et ceux qui estimaient qu'un tel mécanisme serait trop facile à subvertir. Donc, pour l'instant, le client DoH reçoit manuellement un gabarit d'URI (RFC 6570) qui indique le serveur DoH à utiliser. Par exemple, un client recevra le gabarit `https://dns.example.net/{?dns}`, et il fera alors des requêtes HTTPS à `dns.example.net`, en passant `?dns=[valeur de la requête]`.

Notez que l'URL dans le gabarit peut comporter un nom de domaine, qui devra lui-même être résolu via le DNS, créant ainsi un amusant problème d'œuf et de poule (cf. section 10 de notre RFC). Une solution possible est de ne mettre que des adresses IP dans l'URL, mais cela peut poser des problèmes pour l'authentification du serveur DoH, toutes les autorités de certification n'acceptant pas de mettre des adresses IP dans le certificat (cf. RFC 6125, section 1.7.2, et annexe B.2).

La section 5 du RFC détaille quelques points liés à l'intégration avec HTTP. D'abord, les caches. DNS et HTTP ont chacun son système. Et entre le client et le serveur aux extrémités, il peut y avoir plusieurs caches DNS et plusieurs caches HTTP, et ces derniers ne connaissent pas forcément DoH. Que se passe-t-il si on réinjecte dans le DNS des données venues d'un cache HTTP ? En général, les réponses aux requêtes POST ne sont pas mises en cache (elles le peuvent, en théorie) mais les requêtes GET le sont, et les implémenteurs de DoH doivent donc prêter attention à ces caches. La méthode recommandée est de mettre une durée de validité explicite dans la réponse HTTP (comme dans l'exemple plus haut avec `Cache-control:`), en suivant le RFC 9111, notamment sa section 4.2. La durée de validité doit être inférieure ou égale au plus petit TTL de la section principale de la réponse DNS. Par exemple, si un serveur DoH renvoie cette réponse DNS :

```
ns1.bortzmeyer.org. 27288 IN AAAA 2605:4500:2:245b::42
ns2.bortzmeyer.org. 26752 IN AAAA 2400:8902::f03c:91ff:fe69:60d3
ns4.bortzmeyer.org. 26569 IN AAAA 2001:4b98:dc0:41:216:3eff:fe27:3d3f
```

alors, la réponse HTTP aura un `Cache-Control: max-age=26569`, le plus petit des TTL.

Si la réponse DNS varie selon le client, le serveur DoH doit en tenir compte pour construire la réponse HTTP. Le but est d'éviter que cette réponse adaptée à un client spécifique soit réutilisée. Cela peut se faire avec `Cache-Control: max-age=0` ou bien avec un en-tête `Vary:` (RFC 7231, section 7.1.4 et RFC 9111, section 4.1) qui va ajouter une condition supplémentaire à la réutilisation des données mises en cache.

S'il y a un en-tête `Age:` dans la réponse HTTP (qui indique depuis combien de temps cette information était dans un cache Web, RFC 9111, section 5.1), le client DoH doit en tenir compte pour calculer le vrai TTL. Si le TTL DNS dans la réponse est de 600 secondes, mais que `Age:` indiquait que cette réponse avait séjourné 250 secondes dans le cache Web, le client DoH doit considérer que cette réponse n'a plus que 350 secondes de validité. Évidemment, un client qui veut des données ultra-récentes peut toujours utiliser le `Cache-control: no-cache` dans sa requête HTTP, forçant un rafraichissement. (Il est à noter que le DNS n'a aucun mécanisme équivalent, et qu'un serveur DoH ne saura donc pas toujours rafraichir son cache DNS.)

La définition formelle du type MIME `application/dns-message` figure en section 6 de notre RFC, et ce type est désormais enregistré à l'IANA <https://www.iana.org/assignments/media-types/application/dns-message>.

La section 8 du RFC est consacrée aux questions de vie privée. C'est à la fois un des principaux buts de DoH (empêcher l'écoute par un tiers) et un point qui a fait l'objet de certaines polémiques, puisque DoH peut être utilisé pour envoyer toutes les requêtes à un gros résolveur public auquel on ne fait pas forcément confiance. Le RFC 7626 traite séparément deux problèmes : l'écoute sur le réseau, et l'écoute effectuée par le serveur. Sur le réseau, DoH protège : tout est chiffré, via un protocole bien établi, TLS. Du fait que le serveur est authentifié, l'écoute par un homme du milieu est également empêchée. DNS sur TLS (RFC 7858) a exactement les mêmes propriétés, mais pour principal inconvénient d'utiliser un port dédié, le 853, trop facile à bloquer. Au contraire, le trafic DoH, passant au milieu d'autres échanges HTTP sur le port 443, est bien plus difficile à restreindre.

Mais et sur le serveur de destination ? Une requête DNS normale contient peu d'informations sur le client (sauf si on utilise la très dangereuse technique du RFC 7871). Au contraire, une requête HTTP est bien trop bavarde : "*cookies*" (RFC 6265), en-têtes `User-Agent:` et `Accept-Language:`, ordre des en-têtes sont trop révélateurs de l'identité du client. L'utilisation de HTTP présente donc des risques pour la vie privée du client, risques connus depuis longtemps dans le monde HTTP mais qui sont nouveaux pour le DNS. Il avait été envisagé, pendant la discussion à l'IETF, de définir un sous-ensemble de HTTP ne présentant pas ces problèmes, mais cela serait rentré en contradiction avec les buts de DoH (qui étaient notamment de permettre l'utilisation du code HTTP existant). Pour l'instant, c'est donc au client DoH de faire attention. Si la bibliothèque HTTP qu'il utilise le permet, il doit veiller à ne pas envoyer de "*cookies*", à envoyer moins d'en-têtes, etc.

Notez que la question de savoir si les requêtes DoH doivent voyager sur la même connexion que le trafic HTTPS normal (ce que permet HTTP/2, avec son multiplexage) reste ouverte. D'un côté, cela peut aider à les dissimuler. De l'autre, les requêtes HTTP typiques contiennent des informations qui peuvent servir à reconnaitre le client, alors qu'une connexion servant uniquement à DoH serait moins reconnaissable, le DNS étant nettement moins sensible au "*fingerprinting*".

Comme TLS ne dissimule pas la taille des messages, et qu'un observateur passif du trafic, et qui peut en plus envoyer des requêtes au serveur DNS, peut en déduire les réponses reçues, le RFC recommande aux clients DoH de remplir les requêtes DNS selon le RFC 7830.

Le choix de Mozilla d'utiliser DoH pour son navigateur Firefox (voir un compte-rendu de la première expérience <https://blog.nightly.mozilla.org/2018/08/28/firefox-nightly-secure-dns-experimen>) et le fait que, dans certaines configurations, le serveur DoH de Cloudflare était systématiquement utilisé a été très discuté (cf. cette discussion sur le forum des développeurs <https://groups.google.com/forum/#!msg/mozilla.dev.platform/_8OAKUHso0c/QUhjVYz3CAAJ> et cet article du Register <https://www.theregister.co.uk/2018/03/20/mozilla_firefox_test_of_privacy_mechanism_prompts_privacy_worries/>). Mais cela n'a rien à voir avec DoH : c'est le choix d'utiliser un résolveur public géré par un GAFA qui est un problème, pas la technique utilisée pour accéder à ce résolveur public. DNS-sur-TLS aurait posé exactement le même problème. Si Mozilla a aggravé les choses avec leur discours "*corporate*" habituel (« nous avons travaillé très dur pour trouver une entreprise de confiance »), il faut rappeler que le problème de la surveillance et de la manipulation des requête et réponses DNS par les FAI est un problème réel (essayez de demander à votre FAI s'il s'engage à ne jamais le faire). On a vu plus haut que DoH ne prévoit pas de système de découverte du serveur. Il faut donc que cela soit configuré en dur (un travail supplémentaire pour les utilisateurs, s'il n'y a pas de résolveur par défaut). En tout cas, le point important est que DoH (ou DNS-sur-TLS) ne protège la vie privée que si le serveur DoH est honnête. C'est une limitation classique de TLS : « TLS permet de

s'assurer qu'on communique bien avec Satan, et qu'un tiers ne peut pas écouter ». Mais DoH n'impose pas d'utiliser un serveur public, et impose encore moins qu'il s'agisse d'un serveur d'un GAFA.

La section 9 de notre RFC traite des autres problèmes de sécurité. D'abord, sur la relation entre DoH et DNSSEC. C'est simple, il n'y en a pas. DNSSEC protège les données, DoH protège le canal (une distinction que les promoteurs de DNSCurve n'ont jamais comprise <https://www.bortzmeyer.org/dnscurve.html>). DNSSEC protège contre les modifications illégitimes des données, DoH (ou bien DNS-sur-TLS) protège contre l'écoute illégitime. Ils résolvent des problèmes différents, et sont donc tous les deux nécessaires.

Quant à la section 10 du RFC, elle expose diverses considérations pratiques liées à l'utilisation de DoH. Par exemple, si un serveur faisant autorité sert des réponses différentes selon l'adresse IP source du client (RFC 6950, section 4), utiliser un résolveur public, qu'on y accède via DoH ou par tout autre moyen, ne donnera pas le résultat attendu, puisque l'adresse IP vue par le serveur faisant autorité sera celle du résolveur public, probablement très distincte de celle du « vrai » client. Un exemple similaire figure dans le RFC : une technique comme DNS64 (RFC 6147) risque fort de ne pas marcher avec un résolveur DNS extérieur au réseau local.

Quelles sont les mises en œuvre de DoH ? Le protocole est assez récent donc votre système favori n'a pas forcément DoH, ou alors c'est seulement dans les toutes dernières versions. Mais DoH est très simple à mettre en œuvre (c'est juste la combinaison de trois protocoles bien maitrisés, et pour lesquels il existe de nombreuses bibliothèques, DNS, HTTP et TLS) et le déploiement ne devrait donc pas poser de problème.

Voyons maintenant ce qui existe, question logiciels et serveurs. On a vu que Cloudflare a un serveur public, le fameux `1.1.1.1` étant accessible en DoH (et également en DNS-sur-TLS <https://www.bortzmeyer.org/dns-over-tls-atlas-measures.html>). Je ne parlerai pas ici de la question de la confiance qu'on peut accorder à ce serveur (je vous laisse lire sa politique de vie privée <https://developers.cloudflare.com/1.1.1.1/commitment-to-privacy/privacy-policy/privacy-policy/> et l'évaluer vous-même), qui avait été contestée lors de la polémique Mozilla citée plus haut. Cloudflare fournit également une bonne documentation sur DoH <https://developers.cloudflare.com/1.1.1.1/dns-over-https/>, avec une explication de l'encodage <https://developers.cloudflare.com/1.1.1.1/dns-over-https/wireformat/>. Enfin, Cloudflare fournit un résolveur simple (comme stubby ou systemd cités plus haut) qui est un client DoH, cloudflared <https://developers.cloudflare.com/1.1.1.1/dns-over-https/cloudflared-proxy/>.

Un autre serveur DoH public, cette fois issu du monde du logiciel libre, est celui de l'équipe PowerDNS, `https://doh.powerdns.org/` (cf. leur annonce <https://mailman.powerdns.com/pipermail/pdns-users/2018-August/025495.html>). Il utilise leur logiciel dnsdist <https://github.com/ahupowerdns/pdns/tree/dnsdist-doh>.

Vous trouverez une liste de serveurs DoH publics chez DefaultRoutes <https://doh.defaultroutes.de/> ou bien chez curl <https://github.com/curl/curl/wiki/DNS-over-HTTPS#publicly-available-> ou encore sur le portail dnsprivacy.org <https://dnsprivacy.org/wiki/display/DP/DNS+Privacy+Public+Resolvers#DNSPrivacyPublicResolvers-DNS-over-HTTPS(DOH)>. Testons ces serveurs DoH publics avec le programme (en ligne sur `https://www.bortzmeyer.org/files/doh-nghttp.c`), qui avait été écrit au hackathon IETF 101 <https://www.bortzmeyer.org/hackathon-ietf-101.html>, on lui donne l'URL du serveur DoH, et le nom à résoudre, et il cherche l'adresse IPv4 correspondant à ce nom :

```
% ./doh-nghttp https://doh.powerdns.org/ www.bortzmeyer.org
The address is 204.62.14.153

% ./doh-nghttp  https://1.1.1.1/dns-query www.bortzmeyer.org
The address is 204.62.14.153

% ./doh-nghttp  https://doh.defaultroutes.de/dns-query www.bortzmeyer.org
The address is 204.62.14.153

% ./doh-nghttp   https://mozilla.cloudflare-dns.com/dns-query www.bortzmeyer.org
The address is 204.62.14.153
```

Parfait, tout a bien marché. Un autre serveur DoH a la particularité d'être un résolveur menteur (regardez son nom) :

```
% ./doh-nghttp    https://doh.cleanbrowsing.org/doh/family-filter/ www.bortzmeyer.org
The address is 204.62.14.153

% ./doh-nghttp    https://doh.cleanbrowsing.org/doh/family-filter/ pornhub.com
The search had no results, and a return value of 8. Exiting.

% ./doh-nghttp https://doh.powerdns.org/   pornhub.com
The address is 216.18.168.16
```

Bon, et si je veux faire mon propre serveur DoH, on a quelles solutions ? Voyons d'abord le doh-proxy <https://facebookexperimental.github.io/doh-proxy/> de Facebook. On lui indique le résolveur qu'il va utiliser (il n'est pas inclus dans le code, il a besoin d'un résolveur complet, a priori sur la même machine ou le même réseau local) :

```
% doh-proxy --port=9443 --upstream-resolver=192.168.2.254 --certfile=server.crt --keyfile=server.key --uri=/
2018-09-27 10:04:21,997:     INFO: Serving on <Server sockets=[<socket.socket fd=6, family=AddressFamily.AF_INET
```

Et posons-lui des questions avec le même client `doh-nghttp` :

```
% ./doh-nghttp  https://ip6-localhost:9443/ www.bortzmeyer.org
The address is 204.62.14.153
```

C'est parfait, il a marché et affiche les visites :

```
2018-09-27 10:04:24,264:     INFO: [HTTPS] ::1 www.bortzmeyer.org. A IN 0 RD
2018-09-27 10:04:24,264:     INFO: [DNS] ::1 www.bortzmeyer.org. A IN 56952 RD
2018-09-27 10:04:24,639:     INFO: [DNS] ::1 www.bortzmeyer.org. A IN 56952 QR/RD/RA 1/0/0 -1/0/0 NOERROR 374ms
2018-09-27 10:04:24,640:     INFO: [HTTPS] ::1 www.bortzmeyer.org. A IN 0 QR/RD/RA 1/0/0 -1/0/0 NOERROR 375ms
```

Au même endroit <https://facebookexperimental.github.io/doh-proxy/>, il y a aussi un client DoH :

```
% doh-client --domain 1.1.1.1 --uri /dns-query --qname www.bortzmeyer.org
2018-09-27 10:14:12,191:    DEBUG: Opening connection to 1.1.1.1
2018-09-27 10:14:12,210:    DEBUG: Query parameters: {'dns': 'AAABAAABAAAAAAAAA3d3dwpib3J0em1leWVyA29yZwAAHA
2018-09-27 10:14:12,211:    DEBUG: Stream ID: 1 / Total streams: 0
2018-09-27 10:14:12,219:    DEBUG: Response headers: [(':status', '200'), ('date', 'Thu, 27 Sep 2018 08:14:1
id 0
opcode QUERY
rcode NOERROR
flags QR RD RA AD
edns 0
payload 1452
;QUESTION
www.bortzmeyer.org. IN AAAA
;ANSWER
www.bortzmeyer.org. 5125 IN AAAA 2001:4b98:dc0:41:216:3eff:fe27:3d3f
www.bortzmeyer.org. 5125 IN AAAA 2605:4500:2:245b::42
;AUTHORITY
;ADDITIONAL
2018-09-27 10:14:12,224:    DEBUG: Response trailers: {}
```

Ainsi qu'un résolveur simple (serveur DNS et client DoH).

Il ne faut pas confondre ce doh-proxy écrit en Python avec un logiciel du même nom <https://github.com/jedisct1/rust-doh> écrit en Rust (je n'ai pas réussi à le compiler, celui-là, des compétences Rust plus avancées que les miennes sont nécessaires).

Et les clients, maintenant ? Commençons par le bien connu curl, qui a DoH est depuis la version 7.62.0 (pas encore publiée à l'heure où j'écris, le développement est documenté ici <https://github.com/curl/curl/pull/2668>.) curl (comme Mozilla Firefox) fait la résolution DNS lui-même, ce qui est contestable (il me semble préférable que cette fonction soit dans un logiciel partagé par toutes les applications). Voici un exemple :

```
% ./src/.libs/curl -v --doh-url  https://doh.powerdns.org/ www.bortzmeyer.org
... [Se connecter au serveur DoH]
* Connected to doh.powerdns.org (2a01:7c8:d002:1ef:5054:ff:fe40:3703) port 443 (#2)
* ALPN, offering h2
...
* SSL connection using TLSv1.2 / ECDHE-RSA-AES256-GCM-SHA384
* Server certificate:
*  subject: CN=doh.powerdns.org
... [Envoyer la requête DoH]
* Using Stream ID: 1 (easy handle 0x5631606cbd50)
> POST / HTTP/2
> Host: doh.powerdns.org
> Accept: */*
> Content-Type: application/dns-message
> Content-Length: 36
...
< HTTP/2 200
< server: h2o/2.2.5
< date: Thu, 27 Sep 2018 07:39:14 GMT
< content-type: application/dns-message
< content-length: 92
... [On a trouvé la réponse DoH]
* DOH Host name: www.bortzmeyer.org
* TTL: 86392 seconds
* DOH A: 204.62.14.153
* DOH AAAA: 2001:4b98:0dc0:0041:0216:3eff:fe27:3d3f
* DOH AAAA: 2605:4500:0002:245b:0000:0000:0000:0042
... [On peut maintenant se connecter au serveur HTTP - le but
```

```
principal de curl - maintenant qu'on a son adresse IP]
* Connected to www.bortzmeyer.org (204.62.14.153) port 80 (#0)
> GET / HTTP/1.1
> Host: www.bortzmeyer.org
> User-Agent: curl/7.62.0-20180927
> Accept: */*
>
```

Pour les programmeurs Go, l'excellente bibliothèque godns <https://github.com/miekg/dns> n'a hélas pas DoH (pour des raisons internes <https://github.com/miekg/dns/issues/707>). Du code expérimental avait été écrit dans une branche <https://github.com/miekg/dns/tree/doh1> mais a été abandonné. Les amateurs de Go peuvent essayer à la place cette mise en œuvre <https://github.com/m13253/dns-over-https> (on notera que ce client sait parler DoH mais aussi le protocole spécifique et non-standard du résolveur public Google Public DNS).

Pour les programmeurs C, la référence est la bibliothèque getdns <https://getdnsapi.net/> (notez que c'est elle qui a été utilisée pour le client doh-nghttp cité plus haut). Le code DoH est, à la parution du RFC, toujours en cours de développement et pas encore dans un dépôt public. Une fois que cela sera fait, stubby, qui utilise getdns, pourra parler DoH.

Voilà, si vous n'êtes pas épuisé·e·s, il y a encore des choses à lire :

— Plein d'informations sur Doh et ses mises en œuvre <https://doh.defaultroutes.de/> chez DefaultRoutes, dont un bon exposé sur DoH <https://doh.defaultroutes.de/The-End-of-DNS-as-we-know.html>,
— Le dépôt git <https://github.com/dohwg> qui avait été utilisé par le groupe de travail DoH,
— Un bon article amusant <https://blog.apnic.net/2017/11/17/ietf-100-singapore-dns-http-doh/> sur les débuts de DoH.
— Un article de Mozilla sur DoH <https://hacks.mozilla.org/2018/05/a-cartoon-intro-to-dns-over> (l'explication du DNS est assez médiocre mais il y a des dessins, donc cela peut être utile pour présenter DoH à l'école élémentaire). Attention, contient de la propagande "*corporate*".
— L'analyse de Sara Dickinson <https://centr.org/news/news/the-dns-community-brought-dns-over-html> sur les conséquences de DoH.
— Un intéressant article de fond <https://www.potaroo.net/ispcol/2018-10/doh.html> sur DoH.
— Mon exposé sur DoH aux JDLL de 2019 <https://www.bortzmeyer.org/doh-jdll.html>.