RFC 9114 : Hypertext Transfer Protocol Version 3 (HTTP/3)

Stéphane Bortzmeyer
<stephane+blog@bortzmeyer.org>

Première rédaction de cet article le 7 juin 2022

Date de publication du RFC : Juin 2022

https://www.bortzmeyer.org/9114.html

Le protocole de transport QUIC <https://www.bortzmeyer.org/quic.html>, bien que permettant plusieurs protocoles applicatifs au-dessus de lui, a été surtout conçu pour HTTP. Il est donc logique que le premier protocole applicatif tournant sur QUIC et normalisé soit HTTP. Voici donc HTTP/3, la nouvelle version d'HTTP, et la première qui tourne sur QUIC, les précédentes tournant sur TCP (et, souvent, sur TLS sur TCP). À part l'utilisation de QUIC, HTTP/3 est très proche de HTTP/2.

Un petit point d'histoire sur HTTP : la version 1.1 <https://www.bortzmeyer.org/http-11-reecrit.html> (dont la norme précédente datait de 2014) utilise du texte, ce qui est pratique pour les humains (on peut se servir de netcat ou telnet comme client HTTP), mais moins pour les programmes. En outre, il n'a aucun multiplexage, encourageant les auteurs de navigateurs à ouvrir plusieurs connexions TCP simultanées avec le serveur, connexions qui ne partagent pas d'information entre elles et sont donc inefficaces. La version 2 de HTTP (RFC 9113 [1]) est binaire et donc plus efficace, et elle inclut du multiplexage. Par contre, les limites de TCP (une perte de paquet va affecter toutes les ressources en cours de transfert, une ressource lente peut bloquer une rapide, etc) s'appliquent toujours. D'où le passage à QUIC <https://www.bortzmeyer.org/quic.html> pour HTTP/3. QUIC améliore la latence <https://www.bortzmeyer.org/latence.html>, notamment grâce à la fusion avec TLS, et fournit du « vrai » multiplexage. Autrement, HTTP/3 est très proche de HTTP/2, mais il est plus simple, puisque le multiplexage est délégué à la couche transport.

QUIC <https://www.bortzmeyer.org/quic.html> offre en effet plusieurs fonctions très pratiques pour HTTP/3, notamment le multiplexage (complet : plus de "*head-of-line blocking*"), le contrôle de débit par ruisseau et pas par connexion entière, et l'établissement de connexion à faible latence <https://www.bortzmeyer.org/latence.html>. QUIC a été conçu en pensant à HTTP et aux problèmes spécifiques du Web.

1. Pour voir le RFC de numéro NNN, https://www.ietf.org/rfc/rfcNNN.txt, par exemple https://www.ietf.org/rfc/rfc9113.txt

Nous avons maintenant toute une panoplie de versions de HTTP, du HTTP/1.1 du RFC 9112 et suivants, au HTTP/2 du RFC 9113, puis désormais à notre HTTP/3. De même que HTTP/2 n'a pas supprimé HTTP/1, HTTP/3 ne supprimera pas HTTP/2, ne serait-ce que parce qu'il existe encore beaucoup de réseaux mal gérés et/ou restrictifs, où QUIC ne passe pas. Toutes ces versions de HTTP ont en commun les mêmes sémantiques, décrites dans le RFC 9110.

La section 2 du RFC fait un panorama général de HTTP/3. Quand le client HTTP sait (on verra plus tard comment il peut savoir) que le serveur fait du HTTP/3, il ouvre une connexion QUIC avec ce serveur. QUIC fera le gros du travail. À l'intérieur de chaque ruisseau QUIC, il y aura un seul couple requête/réponse HTTP. La requête et la réponse seront placées dans des trames QUIC de type <https://www.iana.org/assignments/quic/quic.xml#quic-frame-types> STREAM (les trames de contenu dans QUIC). À l'intérieur des trames QUIC, il y aura des trames HTTP <https://www.iana.org/assignments/http3-parameters/http3-parameters.xml#http3-parameters-frame-types>, dont deux types sont particulièrement importants, HEADERS et DATA. QUIC étant ce qu'il est, chaque couple requête/réponse est séparé et, par exemple, la lenteur à envoyer une réponse n'affectera pas les autres requêtes, mêmes envoyées après celle qui a du mal à avoir une réponse. (Il y a aussi la possibilité pour le serveur d'envoyer des données spontanément, avec les trames HTTP de type PUSH_PROMISE et quelques autres.) Les requêtes et les réponses ont un encodage binaire comme en HTTP/2, et sont comprimées avec QPACK (RFC 9114), alors que HTTP/2 utilisait HPACK (qui nécessitait une transmission ordonnées des octets, qui n'existe plus dans une connexion QUIC).

Après cette rapide présentation, voyons les détails, en commençant par le commencement, l'établissement des connexions entre client et serveur. D'abord, comment savoir si le serveur veut bien faire du HTTP/3 ? Le client HTTP a reçu consigne de l'utilisateur d'aller en https://serveur-pris-au-hasard.example/, comment va t-il choisir entre HTTP/3 et des versions plus classiques de HTTP ? Il n'y a rien dans l'URL qui indique que QUIC est possible mais il y a plusieurs méthodes de découverte, permettant au client une grande souplesse. D'abord, le client peut simplement tenter sa chance : on ouvre une connexion QUIC vers le port 443 et on voit bien si ça marche ou si on reçoit un message ICMP nous disant que c'est raté. Ici, un exemple vu avec tcpdump :

```
11:07:20.368833 IP6 (hlim 64, next-header UDP (17) payload length: 56) 2a01:e34:ec43:e1d0:554:492d:1a13:93e4
11:07:20.377878 IP6 (hlim 52, next-header ICMPv6 (58) payload length: 104) 2001:41d0:302:2200::180 > 2a01:e3
```

Si ça rate, on se rabat en une version de HTTP sur TCP (les premiers tests menés par Google indiquaient qu'entre 90 et 95 % des utilisateurs avaient une connectivité UDP correcte et pouvaient donc utiliser QUIC). Mais le cas ci-dessus était le cas idéal où on avait reçu le message ICMP et où on avait pu l'authentifier. Comme il est possible qu'un pare-feu fasciste et méchant se trouve sur le trajet, et jette silencieusement les paquets UDP, sans qu'on reçoive de réponse, même négative, il faut que le client soit plus intelligent que cela, et essaie très vite une autre version de HTTP, suivant le principe des globes oculaires heureux (RFC 8305). L'Internet est en effet farci de "*middleboxes*" qui bloquent tout ce qu'elles ne connaissent pas, et le client HTTP ne peut donc jamais être sûr qu'UDP passera. C'est même parfois un conseil explicite de certains vendeurs <https://knowledgebase.paloaltonetworks.com/KCSArticleDetail?id=kA10g000000ClarCAC>.

Sinon, le serveur peut indiquer explicitement qu'il gère HTTP/3 lors d'une connexion avec une vieille version de HTTP, via l'en-tête Alt-Svc: (RFC 7838). Le client essaie d'abord avec une vieille version de HTTP, puis est redirigé, et se souviendra ensuite de la redirection. Par exemple, si la réponse HTTP contient :

```
Alt-Svc: h3=":443"
```

Alors le client sait qu'il peut essayer HTTP/3 sur le port 443. Il peut y avoir plusieurs services alternatifs dans un `Alt-Svc:` (par exemple les versions expérimentales de HTTP/3). Dernière possibilité, celle décrite dans le RFC 8164. (Par contre, l'ancien mécanisme `Upgrade:` et sa réponse 101 n'est plus utilisé par HTTP/3.)

À propos de port, j'ai cité jusqu'à présent le 443 car c'est le port par défaut, mais on peut évidemment en utiliser un autre, en l'indiquant dans l'URL, ou via `Alt-Svc:`. Quant aux URL de plan `http:` (tout court, sans le S), ils ne sont pas utilisables directement (puisque QUIC n'a pas de mode en clair, TLS est obligatoire) mais peuvent quand même rediriger vers du HTTP/3, via `Alt-Svc:`.

Le client a donc découvert le serveur, et il se connecte. Il doit utiliser l'ALPN TLS (RFC 7301, qui est quasi-obligatoire avec QUIC, et indiquer comme application `h3` (cf. le registre IANA des applications <https://www.iana.org/assignments/tls-extensiontype-values/tls-extensiontype-values.xml#alpn-protocol-ids>). Les réglages divers qui s'appliqueront à toute la connexion (la liste des réglages possibles est dans un registre IANA <https://www.iana.org/assignments/http3-parameters/http3-parameters.xml#http3-parameters-settings>) sont envoyés dans une trame HTTP de type `SETTINGS`. La connexion QUIC peut évidemment rester ouverte une fois les premières requêtes envoyées et les premières réponses reçues, afin d'amortir le coût de connexion sur le plus grand nombre de requêtes possible. Évidemment, le serveur est autorisé à couper les connexions qui lui semblent inactives (ce qui se fait normalement en envoyant une trame HTTP de type `GOAWAY`), le client doit donc être prêt à les réouvrir.

Voilà pour la gestion de connexions. Et, une fois qu'on est connecté, comment se font les requêtes (section 4 du RFC) ? Pour chaque requête, on envoie une trame HTTP de type `HEADERS` contenant les entêtes HTTP (encodés, je le rappelle, en binaire) et la méthode utilisée (`GET`, `POST`, etc), puis une trame de type `DATA` si la requête contient des données. Puis on lit la réponse envoyée par le serveur. Le ruisseau est fermé ensuite, chaque ruisseau ne sert qu'à un seul couple requête/réponse. (Rappelez-vous que, dans QUIC, envoyer une trame QUIC de type `STREAM` suffit à créer le ruisseau correspondant. Tout ce qui nécessite un état a été fait lors de la création de la connexion QUIC.)

Comme expliqué plus haut, les couples requête/réponse se font sur un ruisseau, qui ne sert qu'une fois. Ce ruisseau est bidirectionnel (section 6), ce qui permet de corréler facilement la requête et la réponse : elles empruntent le même ruisseau. C'est celui de numéro zéro dans l'exemple plus loin, qui n'a qu'un seul couple requête/réponse. La première requête se fera toujours sur le ruisseau 0, les autres seront que les ruisseaux 4, 8, etc, selon les règles de génération des numéros de ruisseau de QUIC.

Voici, un exemple, affiché par tshark d'un échange HTTP/3. Pour pouvoir le déchiffrer, on a utilisé la méthode classique, en définissant la variable d'environnement `SSLKEYLOGFILE` avant de lancer le client HTTP (ici, curl), puis en disant à Wireshark d'utiliser ce fichier contenant la clé (`tls.keylog_file: /tmp/quic.key` dans `/.config/wireshark/preferences`). Cela donne :

```
% tshark -n -r /tmp/quic.pcap
1   0.000000    10.30.1.1 → 45.77.96.66  QUIC 1294 Initial, DCID=94b8a6888cb47e3128b13d875980b557d9e415f0, SCID=
2   0.088508  45.77.96.66 → 10.30.1.1    QUIC 1242 Handshake, DCID=57e2ce80152d30c1f66a9fcb3c3b49c81c98f329, SCI
3   0.088738    10.30.1.1 → 45.77.96.66  QUIC 185 Handshake, DCID=3bd5658cf06b1a5020d44410ab9682bcf277610f, SCID
4   0.089655  45.77.96.66 → 10.30.1.1    QUIC 1239 Handshake, DCID=57e2ce80152d30c1f66a9fcb3c3b49c81c98f329, SCI
5   0.089672    10.30.1.1 → 45.77.96.66  QUIC 114 Handshake, DCID=3bd5658cf06b1a5020d44410ab9682bcf277610f, SCID
6   0.090740  45.77.96.66 → 10.30.1.1    QUIC 931 Handshake, DCID=57e2ce80152d30c1f66a9fcb3c3b49c81c98f329, SCID
7   0.091100    10.30.1.1 → 45.77.96.66  HTTP3 257 Protected Payload (KP0), DCID=3bd5658cf06b1a5020d44410ab9682b
8   0.091163    10.30.1.1 → 45.77.96.66  HTTP3 115 Protected Payload (KP0), DCID=3bd5658cf06b1a5020d44410ab9682b
9   0.189511  45.77.96.66 → 10.30.1.1    HTTP3 631 Protected Payload (KP0), DCID=57e2ce80152d30c1f66a9fcb3c3b49c
10  0.190684  45.77.96.66 → 10.30.1.1    HTTP3 86 Protected Payload (KP0), DCID=57e2ce80152d30c1f66a9fcb3c3b49c
11  0.190792    10.30.1.1 → 45.77.96.66  QUIC 85 Protected Payload (KP0), DCID=3bd5658cf06b1a5020d44410ab9682bc
12  0.192047  45.77.96.66 → 10.30.1.1    HTTP3 86 Protected Payload (KP0), DCID=57e2ce80152d30c1f66a9fcb3c3b49c
13  0.193299  45.77.96.66 → 10.30.1.1    HTTP3 604 Protected Payload (KP0), DCID=57e2ce80152d30c1f66a9fcb3c3b49
14  0.193421    10.30.1.1 → 45.77.96.66  QUIC 85 Protected Payload (KP0), DCID=3bd5658cf06b1a5020d44410ab9682bc
```

On y voit au début l'ouverture de la connexion QUIC. Puis, à partir du datagramme 7 commence HTTP/3, avec la création des ruisseaux nécessaires et l'envoi de la trame SETTINGS (ruisseau 3) et de HEADERS (ruisseau 0). Pas de trame DATA, c'était une simple requête HTTP GET, sans corps. Le serveur répond par son propre SETTINGS, une trame HEADERS et une DATA (la réponse est de petite taille et tient dans un seul datagramme). Vous avez l'analyse complète et détaillée de cette session QUIC dans le fichier (en ligne sur https://www.bortzmeyer.org/files/http3-get-request.txt).

La section 7 décrit les différents types de trames définis pour HTTP/3 (rappelez-vous que ce ne sont pas les mêmes que les trames QUIC : toutes voyagent dans des trames QUIC de type STREAM), comme :

— HEADERS (type 1), qui transporte les en-têtes HTTP, ainsi que les « pseudo en-têtes » comme la méthode HTTP. Ils sont comprimés avec QPACK (RFC 9114).
— DATA (type 0) qui contient le corps des messages. Une requête GET ne sera sans doute pas accompagnée de ce type de trames mais la réponse aura, dans la plupart des cas, un corps et donc une ou plusieurs trames de type DATA.
— SETTINGS (type 4), qui définit des réglages communs à toute la connexion. Les réglages possibles figurent dans un registre IANA <https://www.iana.org/assignments/http3-parameters/http3-parameters.xml#http3-parameters-settings>.
— GOAWAY (type 7) sert à dire qu'on s'en va.

Les différents types de trames QUIC sont dans un registre IANA <https://www.iana.org/assignments/http3-parameters/http3-parameters.xml#http3-parameters-frame-types>. Ce registre est géré selon les politiques (cf. RFC 8126) « action de normalisation » pour une partie, et « spécification nécessaire » pour une autre partie de l'espace des types, plus ouverte. Notez que les types de trame HTTP/3 ne sont pas les mêmes que ceux de HTTP/2 <https://www.iana.org/assignments/http2-parameters/http2-parameters.xml#frame-type>, même s'ils sont très proches. Des plages de types sont réservées (section 7.2.8) pour le graissage, l'utilisation délibérée de types non existants pour s'assurer que le partenaire respecte bien la norme, qui commande d'ignorer les types de trame inconnus. (Le graissage et ses motivations sont expliqués dans le RFC 8701.)

Wireshark peut aussi analyser graphiquement les données et vous pouvez voir ici une trame QUIC de type STREAM (ruisseau 0) contenant une trame HTTP/3 de type HEADERS. Notez que cette version de Wireshark ne décode pas encore la requête HTTP (ici, un simple GET) :

Et une fois qu'on a terminé ? On ferme la connexion (section 5), ce qui peut arriver pour plusieurs raisons, par exemple :

— La connexion n'a pas été utilisée depuis longtemps (« longtemps » étant défini par les paramètres de la connexion) et une des deux parties décide de la fermer,
— une des deux parties décide de fermer une connexion car son travail est terminé, et envoie donc une trame de type GOAWAY,
— une erreur a pu se produire, empêchant de continuer la connexion, par exemple parce que le réseau est coupé.

Bien sûr, des tas de choses peuvent aller mal pendant une connexion HTTP/3. La section 8 du RFC définit donc un certain nombre de codes d'erreurs pour signaler les problèmes. Ils sont stockés dans un registre IANA <https://www.iana.org/assignments/http3-parameters/http3-parameters.xml#http3-parameters-error-codes>.

L'un des buts essentiels de QUIC était d'améliorer la sécurité. La section 10 du RFC discute de ce qui a été fait. En gros, la sécurité de HTTP/3 est celle de HTTP/2 quand il est combiné avec TLS, mais il y a quelques points à garder en tête. Par exemple, HTTP/3 a des fonctions, comme la compression d'en-têtes (RFC 9114) ou comme le contrôle de flux de chaque ruisseau qui, utilisées sans précaution, peuvent mener à une importante allocation de ressources. Les réglages définis dans la trame SETTINGS servent entre autres à mettre des limites strictes à la consommation de ressources.

Autre question de sécurité, liée cette fois à la protection de la vie privée, la taille des données. Par défaut, TLS ne cherche pas à dissimuler la taille des paquets. Si on ne sait pas quelle page a chargé un client HTTPS, l'observation de la taille des données reçues, comparée à ce qu'on obtient en se connectant soi-même au serveur, permet de trouver avec une bonne probabilité les ressources demandées (c'est ce qu'on nomme l'analyse de trafic). Pour contrer cela, on peut utiliser le remplissage. HTTP/3 peut utiliser celui de QUIC (avec les trames QUIC de type `PADDING`) ou bien faire son propre remplissage avec des trames HTTP utilisant des types non alloués, mais dont l'identificateur est réservé (les trames HTTP de type inconnu doivent être ignorées par le récepteur).

Toujours question sécurité, l'effort de QUIC et de HTTP/3 pour diminuer la latence <https://www.bortzmeyer.org/latence.html>, avec notamment la possibilité d'envoyer des données dès le premier paquet ("*early data*"), a pour conséquences de faciliter les attaques par rejeu. Il faut donc suivre les préconisations du RFC 8470.

La possibilité qu'offre QUIC de faire migrer une session d'une adresse IP vers une autre (par exemple quand un ordiphone passe de 4G en WiFi ou réciproquement) soulève également des questions de sécurité. Ainsi, journaliser l'adresse IP du client (le `access_log` d'Apache…) n'aura plus forcément le même sens, cette adresse pouvant changer en cours de route. Idem pour les ACL.

Ah, et question vie privée, le fait que HTTP/3 et QUIC encouragent à utiliser une seule session pour un certain nombre de requêtes peut permettre de corréler entre elles des requêtes. Sans "*cookie*", on a donc une traçabilité des utilisateurs.

HTTP/3 a beaucoup de ressemblances avec HTTP/2. L'annexe A de notre RFC détaille les différences entre ces deux versions et explique comment passer de l'une à l'autre. Ainsi, la gestion des ruisseaux qui, en HTTP/2, était faite par HTTP, est désormais faite par QUIC. Une des conséquences est que l'espace des identificateurs de ruisseau est bien plus grand, limitant le risque qu'on tombe à cours. HTTP/3 a moins de types de trames que HTTP/2 car une partie des fonctions assurées par HTTP/2 le sont par QUIC et échappent donc désormais à HTTP (comme `PING` ou `WINDOW_UPDATE`).

HTTP/3 est en plein déploiement actuellement et vos logiciels favoris peuvent ne pas encore l'avoir, ou bien n'avoir qu'une version expérimentale de HTTP/3 et encore pas par défaut. Par exemple, pour Google Chrome, il faut le lancer `google-chrome --enable-quic --quic-version=h3-24` (`h3-24` étant une version de développement de HTTP/3). Pour Firefox, vous pouvez suivre cet article <https://blog.cloudflare.com/how-to-test-http-3-and-quic-with-firefox-nightly/> ou celui-ci <https://hacks.mozilla.org/2021/04/quic-and-http-3-support-now-in-firefox-nightly-and-be>. Quand vous lirez ces lignes, tout sera peut-être plus simple, avec le HTTP/3 officiel dans beaucoup de clients HTTP.

J'ai montré plus haut quelques essais avec curl. Pour avoir HTTP/3 avec curl <https://daniel.haxx.se/blog/2019/09/11/curl-7-66-0-the-parallel-http-3-future-is-here/>, il faut actuellement quelques bricolages, HTTP/3 n'étant pas forcément compilé dans le curl que vous utilisez. Déjà, il faut utiliser un OpenSSL spécial, disponible ici <https://github.com/tatsuhiro-t/openssl/>. Sinon, vous aurez des erreurs à la compilation du genre :

```
openssl.c:309:7: warning: implicit declaration of function ‘SSL_provide_quic_data’ [-Wimplicit-function-declarat
  309 |   if (SSL_provide_quic_data(ssl, from_ngtcp2_level(crypto_level), data,
      |       ^~~~~~~~~~~~~~~~~~~~~
```

ngtcp2 peut se compiler avec GnuTLS et pas le OpenSSL spécial, mais curl ne l'accepte pas (`configure: error: --with-ngtcp2 was specified but could not find ngtcp2_crypto_openssl pkg-config file.`). Ensuite, il faut les bibliothèques ngtcp2 <https://github.com/ngtcp2/ngtcp2> et nghttp3 <https://github.com/ngtcp2/nghttp3>. Une fois celles-ci prêtes, vous configurez curl :

```
% ./configure --with-ngtcp2  --with-nghttp3
...
HTTP2:            enabled (nghttp2)
HTTP3:            enabled (ngtcp2 + nghttp3)
...
```

Vérifiez bien que vous avez la ligne `HTTP3` indiquant que HTTP3 est activé. De même, un `curl --version` doit vous afficher `HTTP3` dans les protocoles gérés. Vous pourrez enfin faire du HTTP/3 :

```
% curl -v --http3 https://quic.tech:8443
*   Trying 45.77.96.66:8443...
* Connect socket 5 over QUIC to 45.77.96.66:8443
* Connected to quic.tech () port 8443 (#0)
* Using HTTP/3 Stream ID: 0 (easy handle 0x55879ffd4c40)
> GET / HTTP/3
> Host: quic.tech:8443
> user-agent: curl/7.76.1
> accept: */*
>
* ngh3_stream_recv returns 0 bytes and EAGAIN
* ngh3_stream_recv returns 0 bytes and EAGAIN
* ngh3_stream_recv returns 0 bytes and EAGAIN
< HTTP/3 200
< server: quiche
< content-length: 462
<
<!DOCTYPE html>

<html>
  ...
```

Pour davantage de lecture sur HTTP/3 :

— La référence est bien sûr le livre libre et gratuit de Daniel Stenberg (l'auteur de curl), "*HTTP/3 explained*" <https://http3-explained.haxx.se/>. Il a une traduction en français. Le source du livre est en ligne <https://github.com/bagder/http3-explained/>.
— Rob Graham a fait un bon résumé de HTTP/3 <https://blog.erratasec.com/2018/11/some-notes-about-http3.html>.
— Et on en trouve un autre sur le blog de Cloudflare <https://blog.cloudflare.com/http-3-from-root>.
— Et il y a une étude détaillée <https://www.smashingmagazine.com/2021/08/http3-core-concepts> de Robin Marx.
— C'est en 2018 que HTTP/3 a reçu son nom, suite à cette annonce <https://mailarchive.ietf.org/arch/msg/quic/RLRs4nB1lwFCZ_7k0iuz0ZBa35s/>

Si vous utilisez Tor, notez que QUIC et HTTP/2 (j'ai bien dit HTTP/2, puisque Tor ne gère pas UDP et donc pas QUIC) peuvent mettre en cause la protection qu'assure Tor. Une analyse sommaire est disponible <https://gitweb.torproject.org/tor-browser-spec.git/plain/position-papers/HTTP3/HTTP3.pdf> En gros, si QUIC, grâce à son chiffrement systématique, donne moins d'infos au réseau, il en fournit peut-être davantage au serveur.

Ah, sinon j'ai présenté HTTP/3 à ParisWeb <https://www.paris-web.fr/2021/conferences/http3-ce-nest-pas-seulement-http2-avec-un-numero-plus-grand.php> et les supports sont disponibles en (en ligne sur https://www.bortzmeyer.org/files/parisweb2021-http3-bortzmeyer.pdf). Et Le blog de curl est désormais accessible en HTTP/3 <https://daniel.haxx.se/blog/2022/05/02/now-on-http-3/>. Voici d'autres parts quelques articles intéressants annonçant la publication de HTTP/3 :

— Celui de Cloudflare <https://blog.cloudflare.com/cloudflare-view-http3-usage/>, avec d'intéressantes statistiques sur l'utilisation des différentes versions de HTTP,
— Celui d'Akamai <https://www.akamai.com/blog/news/the-next-generation-of-http>,
— Celui de Mark Nottingham <https://www.mnot.net/blog/2022/06/06/http-core>.