Le protocole QUIC désormais normalisé

Stéphane Bortzmeyer
<stephane+blog@bortzmeyer.org>

Première rédaction de cet article le 28 mai 2021

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

Le protocole de transport QUIC vient d'être normalisé, sous la forme de plusieurs RFC. QUIC, déjà largement déployé, peut changer pas mal de choses sur le fonctionnement de l'Internet, en remplaçant, au moins partiellement, TCP.

Je vais décrire les RFC en question plus longuement dans des articles spécifiques à chacun mais cet article est consacré à une vision de plus haut niveau : c'est quoi, QUIC, et à quoi ça sert ?

QUIC n'est pas un protocole d'application comme peuvent l'être HTTP ou SSH. Les utilisateurs ordinaires ne verront pas la différence, ils se serviront des mêmes applications, via les mêmes protocoles applicatifs. QUIC est un protocole de transport, donc un concurrent de TCP, dont il vise à résoudre certaines limites, notamment dans le contexte du Web. Quelles limites ? Pour comprendre, il faut revenir à ce à quoi sert la couche de transport (couche 4 du traditionnel modèle en couches). Placée entre les datagrammes d'IP, qui peuvent arriver ou pas, dans l'ordre ou pas, et les applications, qui comptent en général sur un flux d'octets ordonnés, où rien ne manque, la couche transport est chargée de surveiller l'arrivée des paquets, de signaler à l'émetteur qu'il en manque, qu'il puisse réémettre, et de mettre dans le bon ordre les données. Dit comme ça, cela semble simple, mais cela soulève beaucoup de problèmes intéressants. Par exemple, il ne faut pas envoyer toutes les données sans réfléchir : le réseau n'est peut-être pas capable de les traiter, et un envoi trop brutal pourrait mener à la congestion. De même, en cas de pertes de paquet, il faut certes ré-émettre, mais aussi diminuer le rythme d'envoi, la perte pouvant être le signal que le réseau est saturé. C'est à la couche transport de gérer cette congestion, en essayant de ne pas la déclencher, ou en tout cas de ne pas l'aggraver. En pratique, la couche transport est donc très complexe, comme le montre le nombre de RFC sur TCP. La norme de base est le RFC 9293 [1], mais d'autres RFC sont également à prendre en compte.

Maintenant que nous avons révisé les tâches de la couche transport, quelles sont ces limites de TCP dont je parlais, et qui justifient le développement de QUIC ? Notez d'abord qu'elles ont surtout été mentionnées dans le contexte du Web. Celui-ci pose en effet des problèmes particuliers, notamment le désir

1. Pour voir le RFC de numéro NNN, https://www.ietf.org/rfc/rfcNNN.txt, par exemple https://www.ietf.org/rfc/rfc9293.txt

d'une faible latence <https://www.bortzmeyer.org/latence.html> (quand on clique, on veut une réponse tout de suite) et le fait que l'affichage d'une seule page nécessite le chargement de plusieurs ressources (images sans intérêt, vidéos agaçantes, code JavaScript pour faire bouger des trucs, CSS, etc). La combinaison de TCP et de TLS n'est pas satisfaisante question latence, puisqu'il faut d'abord établir la connexion TCP, avant de pouvoir commencer la négociation qui mènera à l'établissement de la session TLS (il existe des solutions partielles comme le "*TCP Fast Open*" du RFC 7413, mais qui n'est pas protégé par la cryptographie, ou la "*session resumption*" du RFC 8446, section 2.2). Ensuite, TCP souffre du manque de parallélisme : quand on veut récupérer plusieurs ressources, il faut soit ouvrir plusieurs connexions TCP, qui ne partageront alors plus d'information (RTT, taux de perte, etc) ou de calculs (cryptographie…), et consommeront potentiellement beaucoup de ports, soit multiplexer à l'intérieur d'une connexion TCP, ce que fait HTTP/2 (RFC 7540) mais, alors, on risque le "*head-of-line blocking*", où la récupération d'une ressource bloque les suivantes, et le fait que la perte d'un seul paquet va faire ralentir tous les téléchargements. Cela ne veut pas dire que TCP est complètement dépassé : largement testé au feu depuis de nombreuses années, il déplace chaque jour d'innombrables octets, pour des connexions qui vont de la courte page HTML à des fichiers de plusieurs giga-octets.

Quels sont les concepts de base de QUIC ? QUIC gère des **connexions** entre machines, comme TCP. Par contre, contrairement à TCP, ces connexions portent ensuite des **ruisseaux** ("*streams*") séparés, ayant leur propre contrôle de congestion, en plus du contrôle global à la connexion. Les ruisseaux peuvent être facilement et rapidement créés. Ce n'est pas par hasard que le concept ressemble à celui des ruisseaux de HTTP/2 (RFC 7540), dans les deux cas, le but était de s'adapter au désir des navigateurs Web de charger toutes les ressources d'une page en parallèle, sans pour autant « payer » l'établissement d'une connexion pour chaque ressource.

Avec QUIC, le chiffrement est obligatoire. Il n'y a pas de version de QUIC en clair. Non seulement les données applicatives sont chiffrées, comme avec TLS ou SSH mais une bonne partie de la couche transport l'est également. Pourquoi ?

— En raison de la prévalence de la surveillance massive sur l'Internet (RFC 7258) ; moins on expose de données, moins il y a de risques.
— Afin d'éviter que les "*middleboxes*" ne se croient capables d'analyser le fonctionnement de la couche transport, et ne se croient autorisées à y intervenir, ce qui mène à une ossification de l'Internet. Si tout est chiffré, on pourra faire évoluer le protocole sans craindre l'intervention de ces fichus intermédiaires.
— Certains FAI ont déjà exploité le caractère visible de TCP (RFC 8546) pour des attaques, par exemple en envoyant des paquets RST ("*ReSeT*") pour couper le trafic BitTorrent. TLS et SSH ne protègent pas contre ce genre d'attaques. (Mais QUIC ne protège pas complètement ; le problème est difficile car il faut bien permettre aux machines terminales de réclamer une coupure de la connexion.)
— L'observation de la couche transport peut permettre d'identifier les services utilisés et d'en déprioritiser certains. Le chiffrement du transport peut donc aider à préserver la neutralité du réseau <https://www.bortzmeyer.org/neutralite.html>.

On peut prévoir que les habituels adversaires du chiffrement protesteront d'ailleurs contre QUIC, en l'accusant de gêner la visibilité (ce qui est bien le but). Voir par exemple le RFC 8404 et même le RFC 9065. On voit ainsi un fabricant de produits de sécurité qui conseille carrément de bloquer QUIC <https://knowledgebase.paloaltonetworks.com/KCSArticleDetail?id=kA10g000000ClarCAC>. Qu'est-ce que vont dire ces adversaires du chiffrement lorsqu'on aura des VPN sur QUIC, comme ce sur quoi travaille le bien nommé groupe MASQUE <https://datatracker.ietf.org/wg/masque/> !

QUIC utilise le classique protocole TLS (RFC 8446) pour le chiffrement **mais** pas de la manière habituelle. Normalement, TLS fait la poignée de main initiale, l'échange des clés **et** le chiffrement des données. Avec QUIC, TLS ne fait que la poignée de main initiale et l'échange des clés. Une fois les clés obtenues, QUIC chiffrera tout seul comme un grand, en utilisant les clés fournies par TLS.

Puisqu'on a parlé des "*middleboxes*" : déployer un nouveau protocole de transport dans l'Internet d'aujourd'hui est très difficile, en raison du nombre d'équipements qui interfèrent avec le trafic (les routeurs NAT, par exemple). Conceptuellement, QUIC aurait pu tourner directement sur IP. Mais il aurait alors été bloqué dans beaucoup de réseaux. D'où le choix de ses concepteurs de le faire passer sur UDP. (Le protocole de transport SCTP avait eu ce problème et a dû se résigner à la même solution, cf. RFC 6951.) QUIC utilise UDP comme il utiliserait IP. On lit parfois que QUIC, ce serait « HTTP au-dessus d'UDP », mais c'est une grosse incompréhension. QUIC est une couche de transport complète, concurrente de TCP, dont le fonctionnement sur UDP n'est qu'un détail nécessaire au déploiement. QUIC n'a aucune des propriétés d'UDP. Par exemple, comme TCP, mais contrairement à UDP, QUIC teste la réversibilité <https://www.bortzmeyer.org/returnability.html> du trafic, ce qui empêche l'usurpation d'adresses IP <https://www.bortzmeyer.org/usurpation-adresse-ip.html> et toutes les attaques UDP qui reposent dessus.

QUIC est parfois présenté comme « tournant en mode utilisateur et pas dans le noyau (du système d'exploitation) ». En fait, ce n'est pas spécifique à QUIC. Tout protocole de transport peut être mis en œuvre en mode utilisateur ou noyau (et il existe des mises en œuvre de TCP qui fonctionnent en dehors du noyau). Mais il est exact qu'en pratique la plupart des mises en œuvre de QUIC ne sont pas dans le noyau du système, l'expérience prouvant que les mises à jour du système sont souvent trop lentes, par rapport au désir de faire évoluer en permanence la couche transport. Ceci dit, la norme ne l'impose pas et n'en parle même pas. (On peut ajouter aussi que, dans beaucoup de systèmes d'exploitation, il est plus facile à un programme utilisateur de tourner sur UDP que directement sur IP. Par exemple, sur Unix, tourner directement sur IP nécessite d'utiliser les prises brutes et donc d'être root.) Et puis, sur Windows, Google n'aime pas dépendre de Microsoft pour les performances de son navigateur.

Pour résumer les différences entre QUIC et TCP (rappelez-vous qu'ils assurent à peu près les mêmes fonctions) :

— QUIC est systématiquement chiffré,
— QUIC permet un vrai multiplexage ; la requête lente ne bloque pas la rapide, et la perte d'un paquet ne ralentit pas tous les ruisseaux multiplexés,
— QUIC fusionne transport et chiffrement, ce qui permet notamment de diminuer la latence <https://www.bortzmeyer.org/latence.html> à l'établissement de la connexion,
— QUIC permet la migration d'une connexion en cas de changement d'adresse IP (je n'en ai pas parlé ici, voir RFC 9000).

Les RFC normalisant QUIC sont :

— RFC 9000 est la norme principale, décrivant le socle de base de QUIC,
— RFC 9001 normalise l'utilisation de TLS avec QUIC,
— RFC 9002 spécifie les mécanismes de récupération de QUIC, quand des paquets sont perdus et qu'il faut ré-émettre, sans pour autant écrouler le réseau,
— RFC 8999 est consacré aux invariants de QUIC, aux points qui ne changeront pas dans les nouvelles versions de QUIC.

J'ai dit que QUIC, comme TCP, est un protocole de transport généraliste, et qu'on peut faire tourner plusieurs applications différentes dessus. En pratique, QUIC a été conçu essentiellement en pensant à HTTP mais dans le futur, d'autres protocoles pourront profiter de QUIC, notamment s'ils ont des problèmes qui ressemblent à ceux du Web (désir de faible latence, et de multiplexage).

Pour HTTP, la version de HTTP qui tourne sur QUIC se nomme HTTP/3 et a été normalisée plus tard, dans le RFC 9113. Comme HTTP/2 (RFC 7540), HTTP/3 a un encodage binaire mais il ne fait plus de multiplexage, celui-ci étant désormais assuré par QUIC. Pour d'autres protocoles, les curieux pourront s'intéresser à SMB (déjà géré par Wireshark <https://gitlab.com/wireshark/wireshark/-/merge_requests/123>), au DNS (draft-ietf-dprive-dnsoquic) ou SSH (draft-bider-ssh-quic). En moins sérieux, il y a même eu des discussions pour mettre IRC sur QUIC <https://github.com/ircv3/ircv3-ideas/issues/28>, mais ce n'est pas allé très loin.

QUIC a eu une histoire longue et compliquée. À l'origine, vers 2012, QUIC était un projet Google (documenté dans « "*QUIC : Multiplexed Transport Over UDP*" <https://goo.gl/dMVtFi> », article qui ne reflète pas le QUIC actuel). Si les motivations étaient les mêmes que celles du QUIC actuel, et que certains concepts étaient identiques, il y avait quand même deux importantes différences techniques : le QUIC de Google utilisait un protocole de cryptographie spécifique, au lieu de TLS, et il était beaucoup plus marqué pour son utilisation par HTTP uniquement. Et il y a aussi bien sûr une différence politique, QUIC était un protocole privé d'une entreprise particulière, il est maintenant une norme IETF. Le travail à l'IETF a commencé en 2016 <https://www.ietfjournal.org/quic-performance-and-security-at-the-transport-layer/> à la réunion de Séoul <https://www.ietf.org/how/meetings/97/>. Les discussions à l'IETF ont été chaudes et prolongées, vu l'importance du projet. Après tout, il s'agit de remplacer, au moins partiellement, le principal protocole de transport de l'Internet. C'est ce qui explique qu'il se soit écoulé plus de quatre ans <https://datatracker.ietf.org/doc/draft-ietf-quic-transport/> entre le premier projet IETF et ces RFC. Vous pouvez avoir une idée de ce travail en visitant le site Web du groupe de travail <https://quicwg.org/> ou en admirant les milliers de tickets traités <https://github.com/quicwg/base-drafts/issues>.

Questions mises en œuvre de QUIC (elles sont nombreuses), je vous renvoie à mon article sur le RFC 9000. Le déploiement de QUIC a commencé il y a longtemps puisque Google l'avait déjà mis dans Chrome et ses services. En 2017, Google annonçait <https://blog.apnic.net/2017/12/12/internet-protocols-changing/> que QUIC représentait déjà 7 % du trafic vers ses serveurs. Bon, c'était le QUIC de Google, on va voir ce que cela va donner pour celui de l'IETF, mais le point important est que le déploiement n'a pas attendu les RFC.

Quelques lectures sur QUIC :

— 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/>. En dépit de son nom, il couvre également QUIC. Il a une traduction en français.
— Mon exposé à Capitole du Libre <https://www.bortzmeyer.org/capitole-du-libre-2019.html> en 2019 couvrait QUIC, avec un beau résumé en image </images/sketchnotes-cdl2019.png>, fait par Kevin Ottens <https://mamot.fr/@ervin>.
— L'IETF a publié un résumé de QUIC sur son blog <https://www.ietf.org/blog/innovative-new-tech>.
— Il n'y a pas à l'heure actuelle de page QUIC sur le Wikipédia francophone.
— Si vous aimez les détails techniques, outre les RFC cités plus haut, j'ai fait un article détaillant une session QUIC <https://www.bortzmeyer.org/quic-demo.html>.