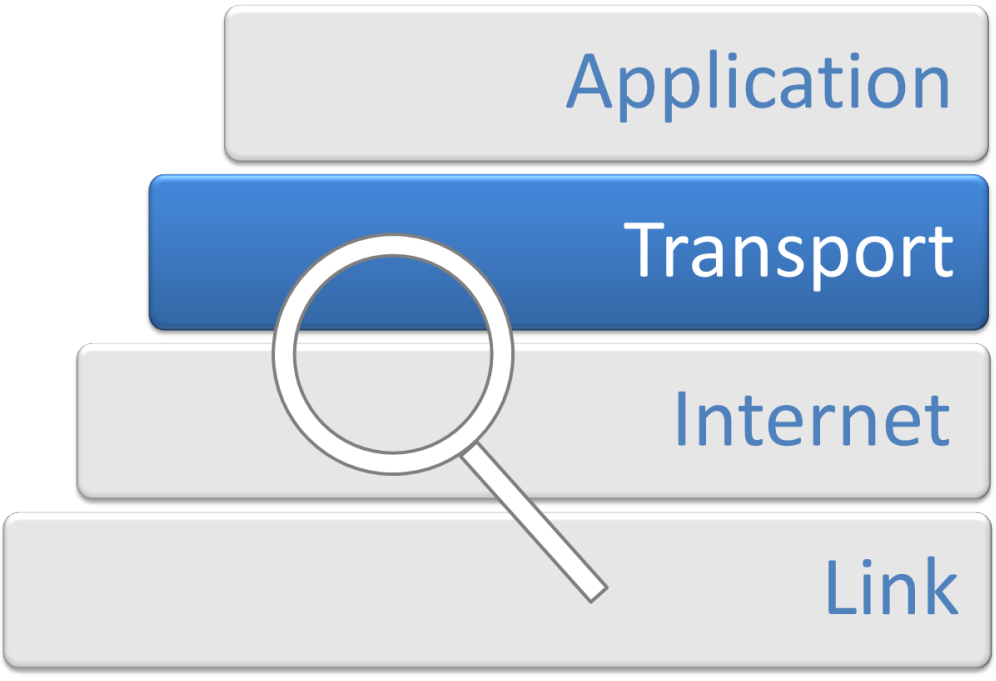
# Le protocole QUIC, ou la nouvelle couche de transport (1/13)

Stéphane Bortzmeyer
stephane+capitoledulibre@bortzmeyer.org

Capitole du Libre, Toulouse, 16 novembre 2019



## Retour sur le modèle en couches




Dave Braunschweig [CC BY-SA 3.0 (https ://creativecommons.org/licenses/by-sa/3.0)], via Wikimedia Commons

## Ce que fait une couche Transport

- Dissimuler aux applications les fantaisies du réseau (paquets perdus, paquets réordonnancés...),
- Tout en contrôlant le flux de données (ne pas étouffer le récepteur),
- Et en évitant la congestion.



## Les limites de TCP

- TCP est un énorme succès, propulse la grande majorité des échanges, s'adapte à des cas très différents,
- Mais...
- Il est mal adapté au Web, avec ses pages composées de plusieurs objets,
- Si on met tout sur une connexion, l'objet le plus lent bloque les autres (*Head-of-Line blocking*),
- Même si on utilise les ruisseaux de HTTP/2, un paquet perdu, et tout le monde attend,
- Si on fait une connexion TCP par objet, on ne peut pas partager les mesures de RTT, ou l'établissement de TLS,
- Avec TLS, l'ouverture de connexion est lente car TLS doit attendre TCP. Or, la **latence** est cruciale.

## Les limites des mises en œuvre de TCP

- TCP est typiquement dans le noyau (ce n'est pas obligatoire mais c'est l'habitude),
- Il peut être intéressant d'avoir la couche transport dans l'application (mises à jour plus faciles et plus fréquentes),
- Il y a aussi une histoire politique : l'application n'a pas forcément le même fournisseur que le système d'exploitation.



## Le problème des middleboxes

- L'Internet est truffé de *middleboxes*, des boitiers intermédiaires qui tripotent la couche Transport,
- NAT, pare-feux... (pas les routeurs, qui s'arrêtent à la couche Réseau),
- Les *middleboxes* bloquent tout ce qu'elles ne connaissent pas. En pratique, seuls TCP, UDP et ICMP passent. Et encore : les options sont souvent bloquées.
- Cette **ossification** fait qu'on ne peut plus déployer un nouveau protocole de transport.
- Tout nouveau protocole de transport doit donc tourner sur UDP. C'est le cas par exemple d'un autre concurrent de TCP, SCTP (RFC 6951).

# QUIC

- *Quick UDP Internet Connection* (en fait, on n'utilise plus ce développé, juste le nom QUIC). QUIC est un concurrent de TCP, ce n'est **pas** « faire tourner HTTP sur UDP »,
- QUIC fait donc tout ce que fait TCP (authentification faible de l'adresse IP source, retransmission des perdus, contrôle de congestion...),
- QUIC gère une connexion composée de plusieurs ruisseaux (*streams*), comme dans HTTP/2, le contrôle de flux se fait par ruisseau,
- Connexions identifiées par le *Connection ID*, pas le port, car un routeur NAT a pu changer le port source.
- À l'intérieur d'un paquet QUIC, plusieurs trames (qui peuvent être dans des ruisseaux différents),
- Les trames ont un type (`STREAM`, `CRYPTO`, `ACK`, `PING`,



# Le chiffrement

- Systématiquement chiffré ; pas de QUIC sans chiffrement (Bas les pattes, les *middleboxes* !),
- Et avec AEAD (chiffrement intègre),
- Le chiffrement se fait avec TLS **mais**,
- Plus de protocole des enregistrements (*Record Layer*), on ne garde que celui de salutation. TLS est à côté de QUIC, plus au dessus, il chiffre avec le matériel fourni par TLS.
- QUIC permet le remplissage, pour gêner l'analyse des tailles de paquets.

## La visibilité

- Avec TCP, même en utilisant TLS ou SSH, le fonctionnement du protocole de Transport était **visible** aux *middleboxes*,
- RFC 8546 : la vue depuis le réseau (*wire image*),
- La visibilité permet des attaques comme les faux RST TCP (utilisé par certains FAI pour casser BitTorrent),
- Au contraire, QUIC chiffre presque tout. Les *middleboxes* ne doivent pas voir la couche Transport.
- Certains FAI ne vont pas être contents de ce renforcement du modèle de bout en bout (RFC 8404),
- Exemple du débat sur le *spin bit*, un bit non chiffré pour permettre aux observateurs de mesurer le RTT. Le *spin bit* est optionnel, pour protéger l'intimité.



## HTTP sur QUIC, ou HTTP/3

- QUIC est très optimisé pour HTTP (multiplexage des ruisseaux, contrôle de flux par ruisseau, latence à l'établissement de la connexion le plus faible possible...),
- HTTP/3 a la même sémantique que HTTP (`GET`, `POST`, `User-Agent:`, `Content-Type:`...),
- Son encodage ressemble à celui de HTTP/2 (donc binaire),
- Une requête/réponse par ruisseau. Chaque ruisseau ne sert qu'une fois,
- Trames QUIC `HEADERS` et `DATA`,
- On utilise `Alt-Svc:` (RFC 7838) pour savoir s'il sait faire du `h3` (risque que QUIC soit bloqué). Ou ALPN (RFC 7301).

## La normalisation

- À l'origine développé par Google vers 2012,
- Passage à l'IETF en 2015-2016,
- Principale différence avec la version Google, le protocole de cryptographie spécifique a été remplacé par TLS,
- Autre différence, HTTP/3 défini séparement de QUIC,
- RFC publiés. . . plus tard.



## Les mises en œuvre

- NGTCP2 (en C) `https://github.com/ngtcp2/ngtcp2` et NGHTTP3,
- Quiche (en Rust) `https://github.com/cloudflare/quiche`,
- aioquic (en Python) `https://github.com/aiortc/aioquic`,
- Quicly (en C) `https://github.com/h2o/quicly`,
- Déjà dans curl (option de compilation).

# Conclusion

- QUIC est déjà largement déployé, dans sa version Google,
- La version IETF avance mais c'est long,
- Mon pari est qu'elle sera un succès, porté par les fortes demandes de HTTP.
- Si vous ne lisez qu'une chose sur QUIC : le livre de Daniel Stenberg `https://http3-explained.haxx.se/`.