RFC 8304 : Transport Features of the User Datagram Protocol (UDP) and Lightweight UDP (UDP-Lite)

Stéphane Bortzmeyer
<stephane+blog@bortzmeyer.org>

Première rédaction de cet article le 8 février 2018

Date de publication du RFC : Février 2018

https://www.bortzmeyer.org/8304.html

Le RFC 8303 [1], du groupe de travail TAPS <https://datatracker.ietf.org/wg/taps/about/>, décrit les **fonctions** et **services** rendus par les protocoles de transport aux applications. Ce RFC 8304 se focalise sur deux protocoles sans connexion, UDP et UDP-Lite. Quels services fournissent-ils ?

UDP est normalisé dans le RFC 768 et UDP-Lite dans le RFC 3828. Tous les deux permettent l'envoi et la réception de datagrammes, sans connexion préalable, sans garantie d'être prévenu en cas de perte du datagramme, sans garantie qu'ils soient distribués dans l'ordre. Une application qui utilise UDP va donc devoir connaitre les services exacts que ce protocole fournit. Aujourd'hui, de nombreuses applications utilisent UDP, l'une des plus connues étant sans doute le DNS, sans compter celles qui utilisent un autre protocole de transport mais qui l'encapsulent dans UDP pour passer les boitiers intermédiaires (cf. par exemple RFC 8261). UDP-Lite est très proche d'UDP et ajoute juste la possibilité de distribuer à l'application des paquets potentiellement endommagés. Pour utiliser UDP intelligement, les programmeurs ont tout intérêt à consulter le RFC 8085.

L'API la plus courante est l'API dite « socket », normalisée par POSIX, et dont la meilleure documentation est évidemment le livre de Stevens <https://www.bortzmeyer.org/unix-network-programming.html>. Les applications peuvent envoyer des données avec `send()`, `sendto()` et `sendmsg()`, en recevoir avec `recvfrom()` et `recvmsg()`. Elles peuvent être dans l'obligation de configurer certaines informations de bas niveau comme la permission de fragmenter ou pas, alors que TCP gère cela tout seul. Des options existent dans l'API "*socket*" pour définir ces options.

Notre RFC 8304 suit la démarche du RFC 8303 pour la description d'un protocole de transport. Plus précisement, il suite la première passe, celle de description de ce que sait faire le protocole.

1. Pour voir le RFC de numéro NNN, `https://www.ietf.org/rfc/rfcNNN.txt`, par exemple `https://www.ietf.org/rfc/rfc8303.txt`

Donc, pour résumer (section 3 du RFC), UDP est décrit dans le RFC 768, et UDP-Lite dans le RFC 3828. Tous les deux fournissent un service de datagramme, non connecté, et qui préserve les frontières de message (contrairement à TCP, qui envoie un flot d'octets). Le RFC 768 donne quelques idées sur l'API d'UDP, il faut notamment que l'application puisse écouter sur un port donné, et puisse choisir le port source d'envoi.

UDP marche évidemment également sur IPv6, et une extension à l'API "*socket*" pour IPv6 est décrite dans le RFC 3493.

UDP est très basique : aucun contrôle de congestion (le RFC 8085 détaille ce point), pas de retransmission des paquets perdus, pas de notification des pertes, etc. Tout doit être fait par l'application. On voit que, dans la grande majorité des cas, il vaut mieux utiliser TCP. Certains développeurs utilisent UDP parce qu'ils ont lu quelque part que c'était « plus rapide » mais c'est presque toujours une mauvaise idée.

Notre RFC décrit ensuite les primitives UDP utilisables par les applications :

— `CONNECT` est très différent de celui de TCP. C'est une opération purement locale (aucun paquet n'est envoyé sur le réseau), qui associe des ports source et destination à une prise ("*socket*"), c'est tout. Une fois cette opération faite, on pourra envoyer ou recevoir des paquets. Comme `CONNECT`, `CLOSE` n'a d'effet que local (pas de connexion, donc pas de fermeture de connexion).
— `SEND` et `RECEIVE`, dont le nom indique bien ce qu'elles font.
— `SET_IP_OPTIONS` qui permet d'indiquer des options IP, chose qui n'est en général pas nécessaire en TCP mais peut être utile en UDP.
— `SET_DF` est important car il permet de contrôler la fragmentation des paquets. TCP n'en a pas besoin car la MSS est négociée au début de la connexion. Avec UDP, c'est le programmeur de l'application qui doit choisir s'il laissera le système d'exploitation et les routeurs (en IPv4 seulement, pour les routeurs) fragmenter, ce qui est risqué, car pas mal de pare-feux mal configurés bloquent les fragments, ou bien s'il refusera la fragmentation (DF = "*Don't Fragment*") et devra donc faire attention à ne pas envoyer des paquets trop gros. (Voir aussi les RFC 1191 et RFC 8201 pour une troisième possibilité.)
— `SET_TTL` (IPv4) et `SET_IPV6_UNICAST_HOPS` permettent de définir le TTL des paquets IP envoyés. (Le terme de TTL est objectivement incorrect, et est devenu "*hops*" en IPv6, mais c'est le plus utilisé.)
— `SET_DSCP` permet de définir la qualité de service souhaitée (RFC 2474).
— `SET_ECN` permet de dire qu'on va utiliser ECN (RFC 3168). Comme les précédentes, elle n'a pas de sens en TCP, où le protocole de transport gère cela seul.
— Enfin, dernière primitive citée par le RFC, la demande de notification des erreurs. Elle sert à réclamer qu'on soit prévenu de l'arrivée des messages ICMP.

Certaines primitives ont été exclues de la liste car n'étant plus d'actualité (cf. le RFC 6633).

Avec ça, on n'a parlé que d'UDP et pas d'UDP-Lite. Rappelons qu'UDP-Lite (RFC 3828) est à 95 % de l'UDP avec un seul changement, la possibilité que la somme de contrôle ne couvre qu'une partie du paquet. Il faut donc une primtive `SET_CHECKSUM_COVERAGE` qui indique jusqu'à quel octet le paquet est couvert par la somme de contrôle.

Notez enfin que le RFC se concentre sur l'"*unicast*" mais, si cela vous intéresse, l'annexe A décrit le cas du "*multicast*".