

RFC 768 : User Datagram Protocol

Stéphane Bortzmeyer

<stephane+blog@bortzmeyer.org>

Première rédaction de cet article le 8 Janvier 2009

Date de publication du RFC : Août 1980

<http://www.bortzmeyer.org/768.html>

Le protocole IP, situé dans la couche 3, fournit un service de datagrammes où presque rien n'est garanti. Les datagrammes peuvent arriver dans le désordre, être dupliqués ou, plus couramment, ne pas arriver du tout. IP ne fournit pas non plus de mécanisme permettant d'identifier un processus particulier sur la machine de destination. Quasiment toutes les applications utilisent un protocole de transport au dessus d'IP. TCP (RFC 793¹) est le plus connu. Il fournit, lui, fiabilité de la transmission mais au prix de délais parfois importants. En effet, tout octet devant être acheminé à bon port, TCP doit souvent attendre les retardataires... ou demander leur réexpédition. De plus, le fonctionnement de TCP nécessite l'établissement préalable d'une connexion et cela prend du temps (trois trajets sont nécessaires, donc la latence est parfois importante).

Il existe des applications pour qui cette stricte sémantique est excessive ou, plus exactement, pour lesquels on n'a pas envie de payer son prix. Ainsi, si on transmet un flux vidéo avec RTSP, et qu'un paquet de données se perd, on préfère en général afficher à l'écran un petit carré noir plutôt que de geler l'affichage en attendant une retransmission.

UDP fournit donc un service minimum, peu coûteux. Si l'application n'a pas besoin de fiabilité, UDP lui permet un débit maximum. Si elle a besoin d'une sémantique plus forte, elle doit la mettre en œuvre elle-même.

Un autre service important fourni par TCP ou ses homologues comme SCTP est le contrôle de congestion. Une application qui utilise UDP bêtement, en envoyant autant de données qu'elle le peut, risque de saturer le réseau, au détriment d'autres applications plus raisonnables. Si on utilise UDP pour des données en quantité importante, on doit assurer ce contrôle de congestion soi-même (RFC 3714, RFC 5348 et RFC 5405).

¹Pour voir le RFC de numéro NNN, <http://www.ietf.org/rfc/rfcNNN.txt>, par exemple <http://www.ietf.org/rfc/rfc793.txt>

UDP n'apporte donc presque rien par rapport à IP seul. Son principal ajout est la notion de ports. Le paquet UDP indique deux nombres, le port de source et le port de destination, qui, avec les adresses IP, servent à identifier les deux processus qui communiquent. L'application a typiquement indiqué le port souhaité en appelant `bind`.

Le RFC est ultra-court, seulement trois pages, reflet d'une époque où les protocoles étaient spécifiés avec quelques grands principes plutôt qu'avec un luxe de détails, qui évoque désormais plutôt des textes juridiques. Les RFC d'il y a vingt-huit ans ne ressemblent guère à ceux d'aujourd'hui... Mais il faut aussi noter que la taille du RFC est le reflet de la simplicité d'UDP. Le RFC se contente essentiellement de spécifier le format (simple) de l'en-tête des paquets UDP. Si on analyse UDP avec `pcap`, il suffit (page 1 du RFC), pour le décoder, de :

```
/* UDP header */
struct sniff_udp {
uint16_t      sport; /* source port */
uint16_t      dport; /* destination port */
uint16_t      udp_length;
uint16_t      udp_sum; /* checksum */
};
...
const struct sniff_dns *dns;
...
udp = (struct sniff_udp *) (packet + SIZE_ETHERNET + size_ip);
source_port = ntohs(udp->sport);
dest_port = ntohs(udp->dport);
...
```

Quels protocoles d'application utilisent UDP ? Le plus connu est le DNS, dont l'essentiel du trafic se fait en UDP. Le mode principal du DNS, « une requête / une réponse », chacune tenant souvent dans un paquet, correspond en effet bien à UDP. Au moment de la sortie de notre RFC 768, l'un des principaux utilisateurs d'UDP était d'ailleurs un ancêtre du DNS, "*Internet Name Service (IEN 116)*". Mais il y a aussi NTP (RFC 1305) et, depuis moins longtemps, les protocoles « multimédia » comme RTP (RFC 3550) qui ont typiquement des grandes quantités de données à faire passer, sans exigence qu'elles arrivent toutes (une trame vidéo en retard n'a plus aucun intérêt). Aujourd'hui, les protocoles de transmission de contenu multimédia (par exemple SIP, RFC 3261 pour la téléphonie sur IP) utilisent en général TCP pour le canal de contrôle et UDP pour les données, pour lesquelles l'acheminement de tous les paquets n'est pas vital.

On peut bien sûr utiliser aussi UDP pour les transferts de fichiers, qui ont besoin d'être parfaitement fiables. Cela suppose de gérer, au dessus d'UDP (qui n'assure pas ce service), un mécanisme permettant de s'assurer que tout a été bien transmis. C'est ce que fait le client BitTorrent de référence depuis fin 2008, une décision qui a suscité pas mal de remous (http://www.theregister.co.uk/2008/12/01/richard_bennett_utorrent_udp/) (souvent exagérés (<http://torrentfreak.com/will-utorrent-really-kill-the-internet-081201/>)).

Vu avec `tcpdump`, commande `tcpdump -vvv -nudp`, voici d'ailleurs un exemple de paquet UDP, en l'occurrence pour le DNS (une question et une réponse) :

```
16:03:11.436076 IP (tos 0x0, ttl 64, id 0, offset 0, flags [DF], proto UDP (17), length 78) 192.134.7.248.15 > 192.134.7.248.53
16:03:11.451313 IP (tos 0x0, ttl 54, id 0, offset 0, flags [DF], proto UDP (17), length 775) 199.6.1.30.53 > 192.134.7.248.15
```

Les champs spécifiques d'UDP (protocole 17) sont le champ Longueur et les ports (ici, 53 et 15301).

UDP a aujourd'hui plusieurs camarades / concurrents dans la catégorie « protocole de transport sans connexion et non fiable ». C'est le cas de DCCP (RFC 4340). Il a aussi des ajouts ou des variantes comme DTLS (RFC 5238) pour la sécurité ou UDP-Lite (RFC 3828) pour encore moins de vérifications.

Et pour le programmeur ? Utiliser UDP n'est pas plus difficile que d'utiliser TCP, sauf si on veut assurer les mêmes services que TCP et qu'on met en œuvre la fiabilité des transferts. Sinon, un client UDP a « juste » à créer les prises avec l'option `SOCK_DGRAM` (au lieu de `SOCK_STREAM`), à appeler `bind` pour obtenir un port et à envoyer les paquets, par exemple avec `sendto`.