

RFC 5405 : Unicast UDP Usage Guidelines for Application Designers

Stéphane Bortzmeyer

<stephane+blog@bortzmeyer.org>

Première rédaction de cet article le 18 Novembre 2008

Date de publication du RFC : Novembre 2008

<http://www.bortzmeyer.org/5405.html>

La grande majorité des applications Internet tourne sur le protocole de transport TCP. Mais son concurrent UDP, normalisé dans le RFC 768¹, prend de l'importance avec le multimédia et les jeux en ligne pour lesquels il est souvent bien adapté. Mais, contrairement à TCP, UDP ne fournit aucun mécanisme de contrôle de la congestion. C'est donc aux applications de fournir ce contrôle, suivant les règles expliquées par ce RFC.

UDP est apprécié pour certaines applications car il est simple et léger et le fait qu'il ne garantisse pas l'acheminement de la totalité des paquets n'est pas forcément un problème dans les applications multimédia : si on perd quelques secondes d'une communication téléphonique RTP, il vaut mieux passer à la suite que de perdre du temps à la retransmettre comme le ferait TCP. Mais UDP ne fournit pas non plus de contrôle de la congestion. C'est pourtant nécessaire (RFC 2914 et RFC 2309), à la fois pour assurer que le réseau continue à être utilisable et également pour assurer une certaine **équité** entre les différents flux de données, pour éviter qu'une seule application gourmande ne monopolise le réseau pour elle.

UDP ne le faisant pas, il faut bien que l'application le fasse et, pour cela, qu'elle mette en œuvre les conseils de ce RFC. (Notre RFC contient également des conseils pour d'autres aspects de l'utilisation d'UDP que le contrôle de congestion : mais c'est le plus important.)

Le gros du RFC est dans la section 3 qui détaille ces conseils (la section 5 contient un excellent résumé sous forme d'un tableau des conseils à suivre). Le premier est qu'il vaut peut-être mieux ne pas utiliser UDP. Beaucoup de développeurs d'applications pensent à UDP en premier parce qu'il est simple et facile à comprendre et qu'il est « plus rapide que TCP ». Mais, rapidement, ces développeurs se rendent

1. Pour voir le RFC de numéro NNN, <http://www.ietf.org/rfc/rfcNNN.txt>, par exemple <http://www.ietf.org/rfc/rfc768.txt>

compte qu'ils ont besoin de telle fonction de TCP, puis de telle autre, ils les mettent en œuvre dans leur application et arrivent à une sorte de TCP en moins bien, d'avantage bogué et pas plus rapide. Notre RFC conseille donc d'abord de penser aux autres protocoles de transport comme TCP (RFC 793), DCCP (RFC 4340) ou SCTP (RFC 4960). Ces protocoles sont d'autant plus intéressants qu'ils ont souvent fait l'objet de réglages soigneux depuis de nombreuses années et qu'il est donc difficile à un nouveau programme de faire mieux. D'autant plus qu'il existe souvent des réglages spécifiques pour les adapter à un usage donné. Par exemple, on peut dire à TCP de donner la priorité à la latence (paramètre `TCP_NODELAY` de `setsockopt`) ou bien au débit.

Si on ne suit pas ces sages conseils, et qu'on tient à se servir d'UDP, que doit-on faire pour l'utiliser intelligemment? La section 3.1 couvre le gros morceau, le contrôle de congestion. Celui-ci doit être pris en compte dès la conception de l'application. Si cette dernière fait de gros transferts de données (section 3.1.1, c'est le cas de RTP, RFC 3550), elle doit mettre en œuvre TFRC, tel que spécifié dans le RFC 5348, donc faire à peu près le même travail que TCP. Et ce mécanisme doit être activé par défaut.

Si l'application transmet peu de données (section 3.1.2), elle doit quand même faire attention et le RFC demande **pas plus d'un datagramme par RTT**, où le RTT est un cycle aller-retour avec la machine distante (calculé selon le RFC 2988; si le calcul n'est pas possible, le RFC demande une durée de trois secondes). L'application doit également détecter les pertes de paquet pour ralentir son rythme si ces pertes - signe de congestion - sont trop fréquentes.

Le cas où l'application est un tunnel au dessus d'UDP est également couvert (section 3.1.3).

En suivant toutes ces règles, l'application gère proprement la congestion. Et le reste? La section 3.2 fournit des pistes sur la gestion de la taille des paquets (rester en dessous de la MTU, utiliser la découverte de MTU spécifiée dans des RFC comme le RFC 4821, etc). La 3.3 explique la question de la fiabilité : par défaut, UDP ne retransmet pas les paquets perdus. Si c'est nécessaire, c'est l'application qui doit le faire. Elle doit aussi gérer l'éventuelle duplication des paquets (qu'UDP n'empêche pas). Le RFC note que les retards des paquets peuvent être très importants (jusqu'à deux minutes, normalise le RFC) et que l'application doit donc gérer le cas où un paquet arrive alors qu'elle croyait la session finie depuis longtemps.

La section 3.4 précise l'utilisation des sommes de contrôle (facultatives pour UDP sur IPv4 mais qui devraient être utilisées systématiquement). Si une somme de contrôle pour tout le paquet semble excessive, et qu'on veut protéger uniquement les en-têtes de l'application, une bonne alternative est UDP-Lite (RFC 3828), décrit dans la section 3.4.1.

Beaucoup de parcours sur l'Internet sont encombrés de « *middleboxes* », ces engins intermédiaires qui assurent diverses fonctions (NAT, coupe-feu, etc) et qui sont souvent de médiocre qualité logicielle, bricolages programmés par un inconnu et jamais testés. La section 3.5 spécifie les règles que devraient suivre les applications UDP pour passer au travers sans trop de heurts. Notamment, beaucoup de ces « *middleboxes* » doivent maintenir un **état** par flux qui les traverse. En TCP, il est relativement facile de détecter le début et la fin d'un flux en observant les paquets d'établissement et de destruction de la connexion. En UDP, ces paquets n'ont pas d'équivalent et la détection d'un flux repose en général sur des heuristiques. L'engin peut donc se tromper et mettre fin à un flux qui n'était en fait pas terminé. Si le DNS s'en tire en général (c'est un simple protocole requête-réponse, avec en général moins de deux secondes entre l'une et l'autre), d'autres protocoles basés sur UDP pourraient avoir de mauvaises surprises. Elles doivent donc se préparer à de soudaines interruptions de la communication, si le *"timeout"* d'un engin intermédiaire a expiré alors qu'il y avait encore des paquets à envoyer. (La solution des *"keepalives"* est déconseillée par le RFC car elle consomme de la capacité du réseau et ne dispense pas de gérer les coupures, qui se produiront de toute façon.)

La section 3.6 fera le bonheur des programmeurs qui y trouveront des conseils pour mettre en œuvre les principes de ce RFC, via l'API des prises ("*sockets*", RFC 3493). Elle est largement documentée mais en général plutôt pour TCP que pour UDP, d'où l'intérêt du résumé qu'offre ce RFC, qui ne dispense évidemment pas de lire le Stevens <<http://www.bortzmeyer.org/unix-network-programming.html>>. Par exemple, en l'absence de mécanisme de `TIME_WAIT` (la prise reste à attendre d'éventuels paquets retardés, même après sa fermeture par l'application), une application UDP peut ouvrir une prise... et recevoir immédiatement des paquets qu'elle n'avait pas prévus, qui viennent d'une exécution précédente.

Le protocole ICMP fournit une aide utile, que les applications UDP peuvent utiliser (section 3.7). Mais attention, certains messages ICMP peuvent refléter des erreurs temporaires (absence de route, par exemple) et ne devraient pas entraîner de mesures trop drastiques.

Après tous ces conseils, la section 4 est dédiée aux questions de sécurité. Comme TCP ou SCTP, UDP ne fournit en soi aucun mécanisme d'intégrité des données ou de confidentialité. Pire, il ne fournit même pas d'authentification de l'adresse IP source (authentification fournie, avec TCP, par le fait que, pour établir la connexion, il faut recevoir les réponses de l'autre). L'application doit-elle mettre en œuvre la sécurité seule ? Le RFC conseille plutôt de s'appuyer sur des protocoles existants comme IPsec (RFC 4301) ou DTLS (RFC 6347). En effet, encore plus que les protocoles de gestion de la congestion, ceux en charge de la sécurité sont très complexes et il est facile de se tromper. Il vaut donc mieux s'appuyer sur un système existant plutôt que d'avoir l'"*hubris*" et de croire qu'on peut faire mieux que ces protocoles ciselés depuis des années.

Pour authentifier, il existe non seulement IPsec et DTLS mais également d'autres mécanismes dans des cas particuliers. Par exemple, si les deux machines doivent être sur le même lien (un cas assez courant), on peut utiliser GTSM (RFC 3682) pour s'en assurer.