RFC 7574 : Peer-to-Peer Streaming Peer Protocol (PPSPP)

Stéphane Bortzmeyer
<stephane+blog@bortzmeyer.org>

Première rédaction de cet article le 14 juillet 2015

Date de publication du RFC : Juillet 2015

https://www.bortzmeyer.org/7574.html

Le pair-à-pair est un succès énorme de l'Internet et du contenu en quantités impressionnantes est échangé tous les jours avec des techniques fondées sur ce principe. Toutefois, il s'agit surtout de téléchargement (obtenir un contenu statique) alors qu'on voudrait pouvoir utiliser le même concept, le pair-à-pair, pour du ruissellement ("*streaming*"), afin de distribuer du contenu dynamique, comme un évenement en train de se produire et qu'on filme. C'est ce que permet le nouveau protocole PPSPP ("*Peer-to-Peer Streaming Peer Protocol*"), que normalise ce RFC. Son but est de fournir l'équivalent de BitTorrent pour du contenu dynamique, et beaucoup de concepts de PPSPP sont très proches de BitTorrent (cf. son cahier des charges, dans le RFC 6972 [1]).

Dans PPSPP, le contenu est « auto-certifié ». L'identificateur d'un contenu permet sa vérification. Contrairement au contenu statique, on ne peut pas utiliser un simple condensat cryptographique (méthode de BitTorrent) puisqu'on ne connait pas tous les octets à l'avance. PPSPP se sert donc d'un arbre de Merkle calculé récursivement au fur et à mesure que le contenu est disponible (comme cela avait été proposé pour BitTorrent <http://bittorrent.org/beps/bep_0030.html>, voir aussi la section 5.5). L'identificateur du contenu est le condensat cryptographique de la racine de cet arbre de Merkle. Grâce à cette auto-certification, un pair malveillant ne pourra pas modifier le contenu (il reste à s'assurer de l'authenticité de l'identificateur mais c'est une autre histoire... BitTorrent a le même problème.)

PPSPP est un protocole applicatif qui peut tourner sur plusieurs protocoles de transport. À l'heure actuelle, le seul normalisé est LEDBAT (RFC 6817), le protocole « durable », qui n'envoie des octets que lorsque le tuyau est libre. Ainsi, l'usage de PPSPP ne va pas perturber les autres applications. (La section 8 détaille l'usage de LEDBAT sur UDP.)

1. Pour voir le RFC de numéro NNN, https://www.ietf.org/rfc/rfcNNN.txt, par exemple https://www.ietf.org/rfc/rfc6972.txt

PPSPP peut découvrir la liste des pairs de plusieurs façons : via un "*tracker*" centralisé, ou via une DHT. Ce nouveau RFC suppose la liste des pairs déjà connue et ne décrit que le protocole pour récupérer le contenu. L'obtention de la liste fera l'objet d'autres travaux.

La section 2 de notre RFC présente le protocole, à travers trois exemples. D'abord, un pair qui rejoint un essaim ("*swarm*"). Un utilisateur humain regarde une page Web contenant un élément `<video>`. Son navigateur Web lui a présenté une image avec les contrôles habituels comme un bouton "*Play*". L'utilisateur clique sur ce bouton. L'URL derrière la vidéo indique du PPSPP. Le navigateur va alors récupérer la liste des pairs (rappelez-vous que ce point n'est pas couvert dans ce RFC). Supposons qu'on utilise un "*tracker*" centralisé. Le logiciel PPSPP va s'enregistrer auprès de ce "*tracker*". Il a une liste de pairs, il peut commencer à parler le protocole PPSPP avec eux. D'abord, le message `HANDSHAKE` (section 3.1 de notre RFC). Ensuite, il recevra des messages `HAVE` (section 3.2) indiquant quels sont les parties du flux de données que ses pairs connaissent déjà et peuvent donc fournir à leurs pairs. Il est maintenant un membre de l'essaim.

Deuxième exemple, on veut échanger des données. Notre logiciel PPSPP va envoyer des messages `REQUEST` (section 3.7) indiquant ce qui l'intéresse, en tenant compte des `HAVE` reçus. Les pairs envoient des messages `DATA` (section 3.3) contenant les octets de la vidéo, ainsi que des messages `INTEGRITY` (section 3.5) contenant les éléments de l'arbre de Merkle et permettant de vérifier que des intermédiaires n'ont pas modifié la vidéo (les détails sur cette vérification sont dans la section 5).

Une fois que notre logiciel a des données, il peut téléverser ("*to upload*"). Il va à son tour envoyer des messages `HAVE` aux pairs, et répondre à des requêtes `REQUEST`.

Le départ « propre » de l'essaim se fait en envoyant un message `HANDSHAKE` (et en se désincrivant du "*tracker*", s'il y en a un). Mais, évidemment, dans un réseau pair-à-pair, il est fréquent que des machines ne partent pas proprement : coupure du réseau, plantage de la machine ou du logiciel, etc. Les pairs doivent donc être prêts à gérer le cas de pairs qui ne répondent plus.

La section 3 de notre RFC présente tous les messages qui peuvent être échangés (rassurez-vous : je n'en couvre qu'une partie mais ils ne sont pas si nombreux, seulement douze types). Les messages ne demandent pas de réponse : une absence de nouvelles de la part du pair indique qu'il y a un problème. Par exemple, si on envoie un message `HANDSHAKE` à des pairs qui ne veulent pas de nous, ils ne répondront pas avec un message d'erreur, ils s'abstiendront simplement de répondre. Il y a donc deux sortes de pairs : ceux qui répondent vite et bien et ceux qui répondent peu ou pas, et qu'on va donc petit à petit décider d'ignorer. On n'insiste pas avec les pairs non répondants.

À noter que PPSPP ignore le format des données envoyées : il transmet des octets, sans savoir si c'est du MPEG nu, ou bien un container rassemblant plusieurs fichiers, comme le permettent des formats comme AVI. PPSPP est un protocole de transfert de données, pas une solution complète de télé-diffusion.

Parmi les points à noter dans les messages : chaque essaim transmet un flot d'octets et un seul (cf. l'avertissement ci-dessus sur les containers). Un essaim a un identificateur ("*swarm ID*"), échangé dans les messages `HANDSHAKE`. Ces messages contiennent également d'autres paramètres (cf. section 7) comme la taille des morceaux ("*chunks*") ou comme la fonction exacte utilisée pour l'arbre de Merkle. Le flot d'octets est en effet découpé en morceaux, chacun ayant un identificateur ("*chunk ID*"). Cet identificateur est un paramètre des messages `HAVE`, `REQUEST` et `DATA`. Ainsi, un message `DATA` contient un morceau, et l'identificateur de ce morceau. Cela permet au pair de savoir où il en est et s'il lui manque quelque chose.

J'ai parlé plus haut du message INTEGRITY qui contient un condensat cryptographique d'un sous-arbre de l'arbre de Merkle correspondant au contenu transmis. Les messages INTEGRITY ne sont pas signés donc ils ne protègent que contre les accidents, pas contre un pair malveillant. Il existe des messages SIGNED_INTEGRITY qui, eux, sont signés (cf. sections 5 et 6.1). À noter que, dans ce cas, la clé publique est dans l'identificateur de l'essaim, encodée en suivant le format des DNSKEY de DNSSEC.

Les messages REQUEST, mentionnés plus haut, servent à demander une partie du flot de données. Contrairement à BitTorrent, ils ne sont pas indispensables : un pair peut vous envoyer des morceaux que vous n'avez pas explicitement demandé. C'est logique pour PPSPP, qui est prévu pour le ruissellement : PPSPP est plus "*push*" que "*pull*".

La section 4 explique le mécanisme d'adressage des morceaux. On n'est en effet pas obligé de les désigner un par un. On peut utiliser des intervalles (« du morceau N1 au morceau N2 »), des octets (« tous les morceaux qui composent le premier million d'octets ») ou bien des "*bin numbers*". Un "*bin number*" est un mécanisme très astucieux permettant de désigner un intervalle d'octets presque quelconque avec un seul nombre. Je vous laisse lire la section 4.2 du RFC pour une explication complète. (Si quelqu'un de courageux peut ensuite en faire un article sur les "*bin numbers*" pour Wikipédia... Actuellement, il n'y en a pas.)

La section 12 de notre RFC intéressera tout particulièrement les administrateurs système et réseau. Elle couvre les problèmes opérationnels avec PPSPP. Par exemple, comme avec tout protocole pair-à-pair, un pair PPSPP a intérêt à choisir des pairs « proches » (entre guillemets car il n'est pas évident de définir « proche »). Le protocole ALTO du RFC 7285 peut aider à mieux choisir ses pairs. Mais il faut bien dire que, l'expérience concrète avec PPSPP manquant encore, on est un peu ici dans des zones inexplorées.

La section 13 du RFC, elle, se consacre aux questions de sécurité. D'abord, la vie privée : comme BitTorrent, PPSPP expose à chaque pair l'adresse IP des autres pairs (sauf si on utilise Tor, ce qui ne sera pas forcément simple) et on ne peut donc pas tellement cacher à ses pairs son intérêt pour tel ou tel contenu. À noter que PPSPP ne fournit pas de mécanisme pour protéger la confidentialité du contenu. Si on veut distribuer du contenu à accès restreint, on le chiffre avant, ou bien on utilise IPsec ou DTLS.

Ensuite, le risque d'attaques par réflexion <https://www.bortzmeyer.org/attaques-reflexion.html>, puisque PPSPP tournera typiquement sur UDP. Si un attaquant ment sur son adresse IP, peut-il déclencher l'envoi de données massives vers un innocent ? A priori, non, en raison de l'usage d'un mécanisme analogue aux ISN ("*Initial Sequence Number*") de TCP. La communication nécessite la connaissance de nombres nommés "*channel ID*" et ils sont choisis aléatoirement (RFC 4960). Un attaquant ne peut donc pas les connaitre (cf. section 5.1.3) s'il a triché sur son adresse IP (puisqu'il ne recevra pas le message lui communiquant le "*channel ID*" à utiliser : en d'autres termes, PPSPP teste la « routabilité »).

Une version, apparemment assez expérimentale, de ce protocole a été mise en œuvre en logiciel libre, dans libswift <https://github.com/libswift/libswift>. Une autre, écrite en Erlang est le projet Swirl <http://www.swirl-project.org/>.