RFC 6887 : Port Control Protocol (PCP)

Stéphane Bortzmeyer
<stephane+blog@bortzmeyer.org>

Première rédaction de cet article le 1 mai 2013

Date de publication du RFC : Avril 2013

https://www.bortzmeyer.org/6887.html

Aujourd'hui, l'utilisateur de l'Internet ne bénéficie que rarement d'une connexion neutre, d'un simple tuyau le connectant à toutes les autres machines de l'Internet. La plupart du temps, il est au contraire bloqué derrière des engins comme les routeurs NAT ou les pare-feux. Par défaut, ces engins bloquent les connexions entrantes, limitant l'utilisateur à un usage type Minitel. Dans ce cas, on a souvent besoin de dire à l'engin sur le trajet « laisse entrer des connexions vers tel port » (par exemple pour le pair à pair <https://www.bortzmeyer.org/emule-ports-linux.html>). Cela se fait aujourd'hui en général par une interface spécifique (par exemple une page Web d'administration du routeur NAT) ou par le protocole privé UPnP. Il y avait donc une demande pour un protocole standard, adapté aux mécanismes d'aujourd'hui comme les CGN : c'est ce nouveau protocole, PCP ("*Port Control Protocol*").

Voici un exemple de configuration manuelle comme on fait souvent aujourd'hui, via une interface Web, ici celle d'une Freebox : Avec PCP, ces interfaces vont-elles disparaître complètement ?

Le principe est simple : un serveur PCP tourne sur l'engin bloquant (routeur NAT ou pare-feu) et autorise des clients PCP à lui parler et à demander des services comme l'ouverture d'un port entrant, ou comme une durée de vie plus longue pour les sessions en cours. Redonnant du contrôle à la machine de l'utilisateur, il rétablit un peu de neutralité dans le réseau. Avec PCP, on a un moyen propre de gérer un serveur (acceptant des connexions entrantes) derrière un routeur NAT et même derrière un CGN. Rappelez-vous qu'accepter des connexions entrantes n'est pas seulement utile si on a un serveur Web à la maison, c'est également une fonction demandée par les protocoles de voix sur IP comme SIP ou bien par le pair-à-pair.

Une fois que le client PCP a obtenu une réponse positive du serveur PCP (« OK, j'envoie les connexions entrantes sur le port 8080 à `172.19.1.1`, port 80, comme tu me l'as demandé »), il peut alors prévenir le reste du monde. Cette information sur le rendez-vous n'est pas gérée par PCP : on se sert de fonctions spécifiques au protocole (`SIP REGISTER` auprès du mandataire, pour SIP, par exemple), ou bien du

DNS (éventuellement avec mise à jour dynamique du RFC 2136 [1] pour cela). Les enregistrements SRV du DNS (RFC 2782) sont la solution idéale, puisqu'ils permettent d'indiquer le numéro de port.

On notera que PCP diminue le besoin d'ALG dans les engins intermédiaires. Une nouvelle application n'aura pas besoin d'attendre que des ALG apparaissent, elle pourra contrôler les connexions dont elle a besoin elle-même, via PCP.

Même si on n'accepte pas de connexions entrantes, qu'on est un pur client, PCP peut être utile. Les routeurs NAT et les pare-feux coupent typiquement la session au bout de N minutes d'inactivité. Pour empêcher cela, les applications qui doivent rester connectées longtemps (SSH, par exemple), envoient régulièrement des paquets "*keepalive*". Avec PCP, ce n'est plus nécessaire, l'application peut dire au serveur PCP « augmente N, s'il te plait ».

PCP est conçu, au contraire d'UPnP, pour une large gamme de besoins : du NAT traditionnel (RFC 3022), au NAPT commun aujourd'hui dans les "*boxes*" (cf. RFC 3022, section 2.2), en passant par le CGN (RFC 6888) et des choses plus exotiques comme DS-Lite (RFC 6333 et RFC 6619), NAT64 (RFC 6146), et même NAT66/NPT (RFC 6296).

PCP est prévu pour fonctionner avec les protocoles de transport qui ont la notion de port (UDP, TCP, SCTP, DCCP, etc). Pour les autres (ICMP, RSVP, ESP...), PCP ne fournit que des services partiels.

Autre limitation : PCP suppose que la machine de M. Toutlemonde n'a qu'une seule connexion à l'Internet, passant par l'équipement qui héberge le serveur PCP. Le but est de garantir que l'équipement qui répond en PCP verra bien passer **tous** les paquets. Ce modèle convient à la maison ou à la petite entreprise actuelle, avec son réseau local connecté via une unique "*box*". Le dessin 1 en section 4 du RFC illustre ce modèle.

Les sections suivantes fournissent les détails du protocole. Celui-ci manipulant souvent des adresses IP, la représentation de celles-ci (section 5) va servir souvent : PCP utilise systématiquement un champ de taille fixe de 128 bits (même pour des adresses IPv4, qui sont alors représentées préfixées du `::ffff:0:0/96`), car c'est plus simple.

Une description de haut niveau du protocole est en section 6. À première vue, PCP pourrait être traité comme un protocole requête/réponse (un peu comme le DNS) mais, en fait, comme il n'est pas synchrone (on peut envoyer plusieurs requêtes à la suite et lire ensuite plusieurs réponses), il vaut peut-être mieux le considérer comme un protocole **suggestion/notification**. Le client PCP fait des suggestions au routeur ou au pare-feu (qui sont donc libres de refuser) et le serveur PCP (le routeur ou pare-feu) envoie des notifications indiquant son état. Dans ce modèle, il y a donc deux flux de données indépendants, un du client PCP vers le serveur et un du serveur PCP vers le client. Bien sûr, l'envoi d'une notification est en général corrélé à une suggestion (elle indique indirectement l'acceptation de la suggestion) mais, je le répète, ce n'est pas synchrone. (Une des raisons de ce choix est de permettre au serveur de résister aux attaques par déni de service en ignorant les suggestions lorsque la charge est trop élevée, laissant le client ré-émettre si nécessaire.) PCP tourne sur UDP (ports 5350 et 5351) et n'a pas de notion de transaction (les messages ne portent pas d'identificateur de transaction.) La notification donne l'état actuel du routeur ou pare-feu. Si le client tient à savoir à quelle suggestion est liée telle notification, il doit examiner le contenu du message et le corréler aux suggestions qu'il a envoyées.

1. Pour voir le RFC de numéro NNN, `https://www.ietf.org/rfc/rfcNNN.txt`, par exemple `https://www.ietf.org/rfc/rfc2136.txt`

La section 7 indique le format standard de tous les paquets : un numéro de version (aujourd'hui 2), un bit indiquant s'il s'agit d'une question ou d'une réponse, un code indiquant l'opération (la plus importante est sans doute `MAP`, décrite en section 11), l'adresse IP du client PCP, la durée de vie demandée ou obtenue (pour certaines opérations) et quelques autres informations, qui dépendent de l'opération. Parmi elles, les options, encodées en TLV et enregistrées dans un registre IANA <https://www.iana.org/assignments/pcp-parameters/pcp-parameters.xml#option-rules>.

Les réponses ont un champ indiquant le code de retour d'une requête. 0 est un succès, tous les autres codes indiquent une erreur. Par exemple, 2 est la réponse à une requête non autorisée (rappelez-vous que le serveur PCP a sa propre politique et n'obéit pas aveuglément à tout ce que demande le client), 3 la réponse à une requête syntaxiquement incorrecte, etc. Ces deux erreurs sont permanentes (renvoyer le même paquet PCP ne changera rien). Mais il existe aussi des erreurs temporaires comme 8 qui indique que le serveur PCP aurait bien voulu répondre favorablement mais qu'il manque de ressources (par exemple de ports libres) pour cela. Ré-essayer automatiquement, après un délai raisonnable, peut donc être une bonne solution.

Le fonctionnement de base de PCP figure en section 8. Le client PCP doit d'abord trouver le serveur. Il a pu le connaître par configuration statique, ou tout simplement l'apprendre via une option DHCP (RFC 7291). Sinon, il tente le routeur par défaut, qui sera en général le serveur PCP. Le client va ensuite générer un message, où l'adresse IP du client est sa propre adresse IP (qui sera souvent une adresse privée, du RFC 1918). La section 16.4 explique comment connaître cette adresse grâce à `getsockname()`. Le client doit être prêt à retransmettre (section 8.1.1 pour les détails) : UDP ne garantit pas l'acheminement.

Le serveur reçoit la requête, vérifie qu'elle arrive sur la bonne interface (le CPE typique de M. Toutlemonde, par exemple, n'acceptera pas les requêtes PCP venant de l'Internet, uniquement celles venant du réseau local), vérifie que l'adresse du client dans le paquet correspond à l'adresse IP source (autrement, c'est qu'un NAT supplémentaire se trouvait accidentellement sur le chemin) et répond. Il inclus dans la réponse un champ "*Epoch*" qui est un compteur s'incrémentant une fois par seconde. Si le client voit ce compteur diminuer, cela indiquera que le routeur a probablement redémarré, et donc perdu toutes les correspondances enregistrées, qu'il va falloir re-créer.

C'est quoi, ces « correspondances » ("*mappings*" dans la langue de George Martin) ? Le routeur NAT maintient une table des correspondances entre un couple {adresse IP interne, port interne} et un couple {adresse IP externe, port externe}. Cette table est en partie remplie automatiquement lors des connexions sortantes mais peut aussi être partiellement gérée par PCP. Par exemple, le fonctionnement classique du NAT d'un petit routeur à la maison, lorsqu'un paquet venant de {`192.168.10.10`, 5623} sort, est de réserver un port pour cette session (mettons le port 7891) et de mémoriser une **correspondance** {`203.0.113.254`, 7891} ¡=¿ {`192.168.10.10`, 5623} (on suppose que l'adresse IPv4 externe du routeur est `203.0.113.254`). Avec cette correspondance en tête, le paquet de réponse destiné à {`203.0.113.254`, 7891} sera automatiquement NATé et dirigé vers {`192.168.10.10`, 5623}. PCP permet de manipuler de telles correspondances et, par exemple, pour reprendre un exemple donné plus haut, de dire « envoie les connexions entrantes sur le port 8080 à `192.168.10.1`, port 80 », ce qui se traduira par la création d'une correspondance ("*mapping*") {`203.0.113.254`, 8080} ¡=¿ {`192.168.10.11`, 80}. On notera que le client PCP n'a pas indiqué l'adresse externe `203.0.113.254`. PCP permet de le faire mais, dans le cas le plus courant, le client PCP ne connait pas cette adresse et laisse le serveur la choisir.

La section 10 résume les opérations les plus importantes de PCP, `MAP` (détaillée en section 11) et `PEER` (détaillée en section 12). `MAP` permet de créer les correspondances statiques, pour gérer un serveur, `PEER` permet notamment de manipuler les correspondances dynamiques, celles crées automatiquement par le routeur NAT. La liste de toutes les opérations est dans un registre IANA <https://www.iana.org/

assignments/pcp-parameters/pcp-parameters.xml#opcode-rules> (cf. section 19.2, sur la politique d'affectation des codes numériques correspondants.)

Prenons l'exemple d'un serveur du réseau local (comme {192.168.10.11, 80} dans mon exemple), qui va appeler MAP pour « ouvrir » le routeur ou pare-feu aux connexions entrantes. En pseudo-code :

```
/* Les variables précédées de & sont des pointeurs, pour pouvoir
écrire un résultat dans la variable. */
externalport = 80;
pcp_send_map_request(myport, myaddr, &externalport, &externaladdr,
                requestedlifetime, &assignedlifetime);

if (!pcp_response_received())
      panic("No answer");
}
else {
    /* Éventuellement mettre à jour le DNS, un serveur de rendez-vous,
    pour dire au reste du monde qu'il peut y aller */
}

/* La suite est le code habituel, sans PCP */
if (received_incoming_connection_or_packet())
            process_it(s);
```

Si l'application se moque du port externe attribué, il suffit de ne pas l'indiquer dans la requête et de regarder dans la réponse quel port a été alloué. De toute façon, le routeur ne lui allouera pas forcément le port demandé. Par exemple, pour un CGN, tout le monde voudra sans doute s'allouer le port 80, ce qui n'est pas possible. Soit le CGN le refusera à tous, soit il le donnera au premier arrivé et allouera un autre port pour les suivants. Un client PCP bien fait doit donc être préparé : il doit lire le port effectivement attribué et réagir intelligemment s'il n'a pas eu le port souhaité (ce n'est pas fait dans le programme d'exemple ci-dessus). Bien sûr, le client PCP doit aussi être prêt à ce que sa requête soit complètement rejetée, par exemple parce que la table des correspondances a une taille fixe et que ladite table est pleine.

La lecture de la réponse permet aussi au client PCP de savoir quelle est l'adresse externe allouée, ce qui peut être pratique. (Si on n'est intéressé que par cela, il suffit donc de créer une correspondance avec une très courte durée de vie et de lire la réponse.)

La réponse PCP inclut la durée de vie de la correspondance (champ "*Lifetime*" du paquet, variable assignedlifetime du programme plus haut). C'est la responsabilité du client de renouveller la correspondance avant qu'elle n'expire (cf. section 15).

Si l'application ne sait pas si elle est derrière un routeur NAT ou un pare-feu, ce qui est fréquent, la solution recommandée est quand même d'essayer PCP. S'il n'y a pas de réponse, cela peut vouloir dire que la machine a une connectivité complète (sans NAT) ou malheureusement que le routeur ne gère pas PCP. (Le code plus haut suppose que l'absence de réponse indique qu'on n'a pas de connectivité en tant que serveur, donc panic().)

Le routeur doit créer des correspondances ayant la sémantique "*Endpoint Independent Filtering*", ce qui veut dire que cette correspondance sera la même quelle que soit l'adresse IP de la machine distante. Il va devoir aussi se débrouiller pour que les messages ICMP correspondant à cette correspondance soient bien acheminés à la machine.

Outre les champs génériques communs à tous les paquets PCP, MAP a quelques champs qui lui sont propres notamment le port interne (la variable myport dans le pseudo-code), le port externe **suggéré** (la

variable `externalport` dans le programme en pseudo-code) et l'adresse IP externe suggérée (rappelez-vous que le petit routeur NAT de la maison n'a pas le choix : il n'a en général qu'une seule adresse IPv4 externe, donc il ignorera ce paramètre). Dans la réponse à un `MAP`, la valeur de ces champs indiquera au client ce que le serveur PCP a finalement choisi (pas forcément ce qui était demandé). Une option de la requête permet de dire si les adresses et ports suggérés sont impératifs (on préfère un échec plutôt qu'une correspondance avec d'autres valeurs que celles demandées) ou pas (cf. section 13.2).

`MAP` permet aussi de contrôler un pare-feu, avec son option `FILTER` (section 13.3) où on décrit les adresses IP et ports autorisés, le reste étant bloqué. Séparer la fonction de NAT de celle de pare-feu est essentiel : actuellement, les routeurs NAT typiques mélangent les deux, ce qui a mené certaines personnes à croire que le NAT avait un rôle de protection <`https://www.bortzmeyer.org/nat-et-securite.html`>. Outre la sécurité, `FILTER` peut être utile pour des machines sur batterie : elles éviteront ainsi de gaspiller du courant à traiter des paquets non désirés.

PCP est également utile au cas où on n'a certes besoin que de connexions sortantes mais où on veut influencer des paramètres comme la durée de vie des correspondances créées automatiquement, de manière à ne pas avoir besoin d'envoyer de temps en temps des paquets "*keepalive*". On utilise pour cela l'opération `PEER`. Cette opération a à peu près les mêmes champs que `MAP` mais les correspondances crées peuvent avoir des sémantiques autres que "*Endpoint Independent Filtering*". (`MAP` a le code 1 et `PEER` 2, dans le registre IANA <`https://www.iana.org/assignments/pcp-parameters/pcp-parameters.xml#opcode-rules`>.)

Un autre usage de l'opération `PEER` est pour re-créer des correspondances perdues. Un des plus gros inconvénients du NAT est qu'il maintient un **état** dans le routeur. Normalement, l'Internet est un réseau de paquets, les routeurs intermédiaires font passer les paquets mais n'ont aucune notion de session associées, il font passer du TCP, de l'UDP ou n'importe quel autre protocole sans faire de différence. Un des avantages de ce principe est que le redémarrage d'un routeur ne change rien : les connexions TCP, par exemple, continueront comme si rien ne s'était passé, l'état des connexions étant entièrement contenu dans les machines terminales. Le NAT change tout cela : si un routeur NAT redémarre, toutes les correspondances sont perdues. Si les machines tentent de continuer la communication, de nouvelles correspondances seront créées, avec de nouveaux numéros de port, et ce changement cassera les connexions TCP en cours. C'est notamment insupportable pour SSH.

Comment PCP peut-il aider ? Une machine qui connait la liste des correspondances (elle a pu l'apprendre avec l'opération `PEER`) peut, lorsqu'elle détecte un redémarrage du routeur, lui renvoyer cette liste (toujours avec `PEER` mais, cette fois, en remplissant les champs "*Suggested Address*" et "*Suggested Port*"), rétablissant ainsi la même table des correspondances (section 14). À noter que, pour accélérer la reconstitution de l'état du routeur, un routeur qui sait qu'il va redémarrer peut aussi envoyer des paquets PCP avec l'opération `ANNOUNCE` (section 14.1.3) pour prévenir les clients. Ces envois se font aux adresses "*multicast*" `224.0.0.1:5350` et `[ff02::1]:5350` (`224.0.0.1` et `ff02::1` voulant dire « toutes les machines du réseau »). Les machines clientes peuvent alors savoir qu'elles vont devoir recréer leurs correspondances (après avoir attendu un délai aléatoire, pour éviter qu'elles ne le fassent toutes en même temps).

`PEER` peut enfin créer une nouvelle correspondance mais cela n'a pas un grand intérêt, puisque tout routeur NAT sait le faire automatiquement au premier paquet d'une nouvelle connexion sortante. (La seule exception est le cas cité plus haut où on restaure l'état d'un routeur NAT qui a redémarré.)

Je ne connais pas l'état actuel de mise en œuvre de PCP dans les routeurs NAT typiques. Pour le programmeur qui veut s'y mettre sérieusement, la section 16 contient un certain nombre de conseils pratiques, aussi bien pour l'écriture de serveurs que pour celle de clients PCP. Quant à la section 17, elle se penche sur les problèmes de déploiement de cette nouvelle technologie. Par exemple, il est important

que le routeur rejette les paquets entrants dont l'adresse source ne correspond pas à celles du réseau interne (RFC 2827) pour éviter qu'un méchant n'usurpe une adresse IP.

Tiens, puisqu'on parle de sécurité, la section 18 lui est entièrement consacrée. PCP permet de contrôler plus finement les correspondances dans le routeur, voire de modifier les filtres d'un pare-feu. Il est donc essentiel que ces nouvelles possibilités ne se fassent pas au détriment de la sécurité. Par exemple, une machine du réseau interne autorisée à créer ou modifier des correspondances peut voler le trafic d'une autre. Ce n'est pas très grave pour un réseau local à la maison mais bien plus embêtant dans le cas d'un grand CGN.

PCP, s'il est correctement mis en œuvre, est protégé contre les attaquants situés en dehors du chemin. Par exemple, si une machine quelque part sur l'Internet envoie un paquet PCP prétendant venir d'une des adresses internes (mettons `192.168.1.1`), le routeur/serveur PCP rejettera ce paquet qui arrive par l'interface externe (il ne doit accepter les requêtes PCP que si elles sont internes). Par contre, un attaquant interne a à peu près "*open bar*" avec PCP. Un serveur PCP doit donc être déployé uniquement si toutes les machines internes sont considérées comme membres de la même bande (une supposition raisonnable pour un réseau domestique) ou si on peut séparer les clients en plusieurs domaines, empêchant M. Michu de voler le trafic de Mme Toutlemonde, située derrière le même CGN.

Une façon de limiter les risques est que le serveur PCP ne configure explicitement que des correspondances qu'il aurait de toute façon configuré implicitement, lorsqu'une nouvelle connexion sort. Cela veut dire qu'il y a des opérations PCP sûres (créer une correspondance sans spécifier le port et l'adresse externes) et d'autres qui peuvent être dangereuses (créer une correspondance avec un port externe spécifié).

Si on autorise PCP à configurer un pare-feu, il faut évidemment faire encore plus attention. Il n'existe pas à l'heure actuelle de mécanisme de sécurité standard pour PCP : son premier domaine d'application concerne des cas où il n'y a déjà pas de sécurité aujourd'hui. Pour d'autres domaines, il faudra trouver de nouvelles solutions de sécurité.

Quelques autres attaques possibles avec PCP :

— Risque d'attaque par déni de service contre le serveur PCP, par exemple en créant tellement de correspondances qu'on remplit la table qui les garde en mémoire. Le RFC préconise des limites par machine (du genre, « N correspondances PCP maximum par machine »).
— Risque, en cas de redémarrage d'un routeur NAT, qu'un attaquant rapide ne re-crée une correspondance, avant la machine qui l'avait créée, volant ainsi le trafic. L'attaquant doit agir vite (avant que la victime n'ait re-créé les correspondances) mais c'est possible.

PCP est le successeur d'un protocole très proche nommé PMP ("*Port Mapping Protocol*", RFC 6886). L'annexe A décrit la coexistence entre les implémentations de PMP qui restent, et PCP. L'autre protocole qui avait été envisagé par le groupe de travail PCP <http://tools.ietf.org/wg/pcp> était bien sûr UPnP, mais qui était techniquement moins proche de ce qui était souhaité (non routé, peu sécurisé, peu fiable). PMP utilisant les mêmes ports, des dissecteurs de paquets analysent parfois le PCP sous le nom de PMP.

Aujourd'hui, on trouve très peu de mises en œuvre de PCP déployées. Aucun système d'exploitation n'a de client PCP en série. Une solution possible sera peut-être (suggestion de Gunther Ozerito) de faire un relais UPnP ¡=¿ PCP comme décrit dans le RFC 6970. Sinon, Wireshark va bientôt avoir un dissecteur PCP (il vient d'être développé <https://bugs.wireshark.org/bugzilla/show_bug.cgi?id=7714> et sera livré avec la future version 1.10 ; un autre, sous forme d'un greffon en Lua, est disponible <http://sourceforge.net/projects/pcptestingsuits/>). Attention, Wireshark a aussi un dissecteur pour un autre PCP <http://wiki.wireshark.org/PCP> ("*Performance Co-Pilot Protocol*").

Et j'ai bien envoyé des paquets PCP à ma Freebox, mais je reçois toujours une triste réponse ICMP *"port 5351 unreachable"* d'où je déduis qu'elle n'a pas encore de serveur PCP. (Et elle ne diffuse apparemment rien vers le port 5350.)

Au fait, pour les envoyer, j'ai utilisé ce (très grossier) script Scapy, qui est loin de remplir tous les champs mais permet quelques tests :

```
from scapy.all import *

# http://www.secdev.org/projects/scapy/doc/build_dissect.html
class PCP(Packet):
       name = "Port Control Protocol"
       fields_desc = [ ByteField("Version", 2),
                       BitField("R", 0, 1), # 0 = Request / 1 = Response
                       BitField('Opcode', 1, 7), # MAP is 1, PEER is 2
                       ShortField("Reserved", 0),
                       IntField("RequestedLifetime", 0),
                       LongField("ClientAddressUpper", 0),
                       LongField("ClientAddressLower", 0)
                       ]

p = IP(dst="192.168.2.254")/UDP(sport=0,dport=5351)/PCP(RequestedLifetime=5000,
                                                        ClientAddressLower=232236033)
sr1(p)
```

Un bon article d'introduction sur PCP est « *"New Technology Demo : PCP"* » dans le numéro 7.2 de l'IETF Journal <http://isoc.org/wp/ietfjournal/files/2011/10/IETFJ7-2.pdf>.