Projet de programmation réseau
Groupe de discussion par inondation fiable

Juliusz Chroboczek

17 avril 2019

1 Introduction

Le but de ce projet est d'implémenter un protocole de discussion de groupe, de style *IRC*, mais de façon complètement décentralisée : il n'y a pas de serveur central, et les données sont communiquées à tous les pairs par inondation.

2 Définition du protocole

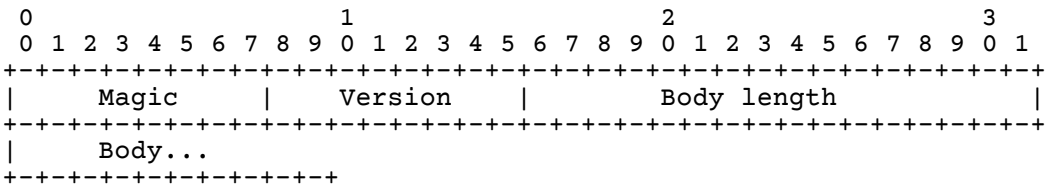
Le protocole est encapsulé dans UDP. Chaque datagramme contient un *message*, qui consiste lui-même d'une suite de triplets type-longueur-valeu (TLV).

Le protocole est un protocole pair-à-pair pur, dans lequel chaque pair est identifé par un *Id*, une suite globalement unique de 64 bits (8 octets).

2.1 Format des messages

Un message est encapsulé dans un datagramme UDP. Toute implémentation doit être capable de recevoir des datagrammes ayant une charge de 4096 octets ou moins, mais ne doit pas envoyer de datagrammes de taille supérieure au PMTU. Un message consiste d'un entête de message suivi d'une suite de messages.

Le corps du datagramme (message de couche application) a le format suivant :

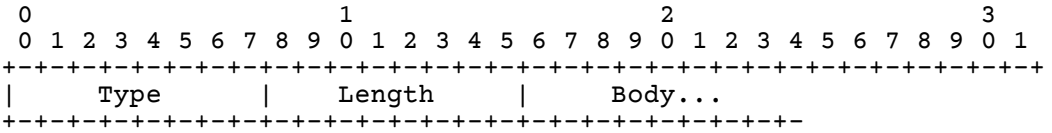


Les champs sont définis comme suit :

Magic : cet octet vaut 93. Tout message qui ne commence pas par un octet valant 93 sera ignoré par le récepteur.
Version : cet octet vaut 2. Tout message qui ne contient pas un champ version valant 2 sera ignoré par le récepteur.
Body length : ce champ indique la longueur des données qui suivent (sans compter les champs *Magic*, *Version* et *Body length*). Si la charge du datagramme UDP est plus grande que *Body Length*+4, les données supplémentaires sont ignorées.
Body : ce champ contient le corps du message, une suite de TLV.

2.2 Format des TLV

Le corps d'un message consiste d'une suite de TLV, des triplets (*type*, *longueur*, *valeur*). À l'exception du TLV *Pad1*, chaque TLV a la forme suivante :

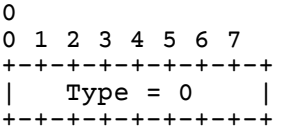


Les champs sont définis comme suit :

Type : le type du TLV, un entier.
Length : la longueur du corps, sans compter les champs *Type* et *Length*.
Body : une suite d'octets de longueur *Length*, dont l'interprétation dépend du TLV.

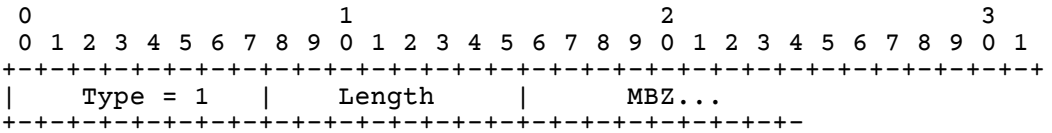
2.3 Détails des TLV

Pad1



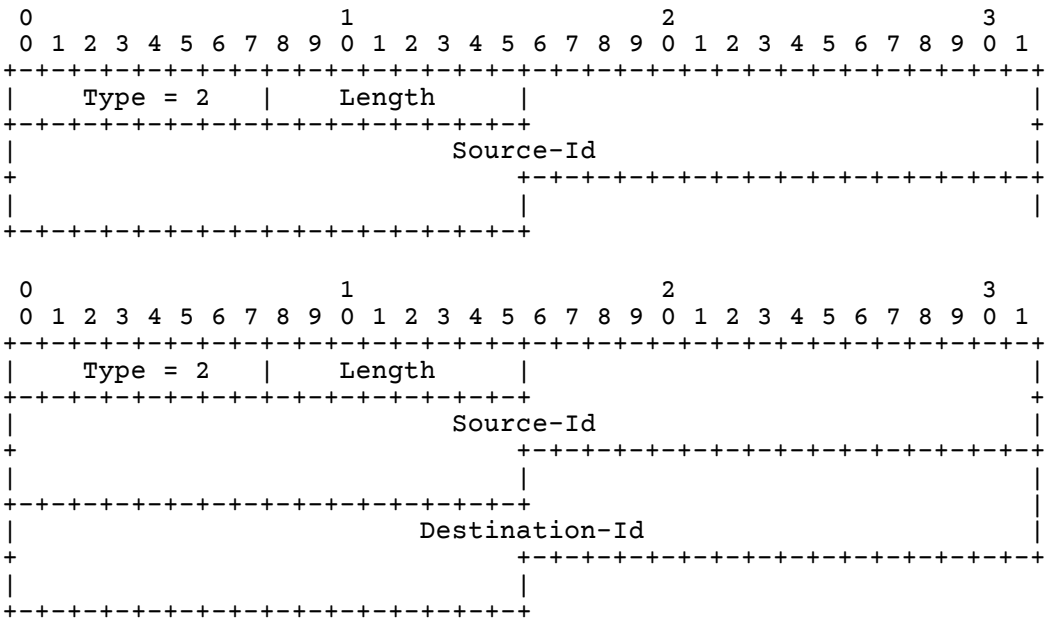
Ce TLV est ignoré à la réception.

PadN



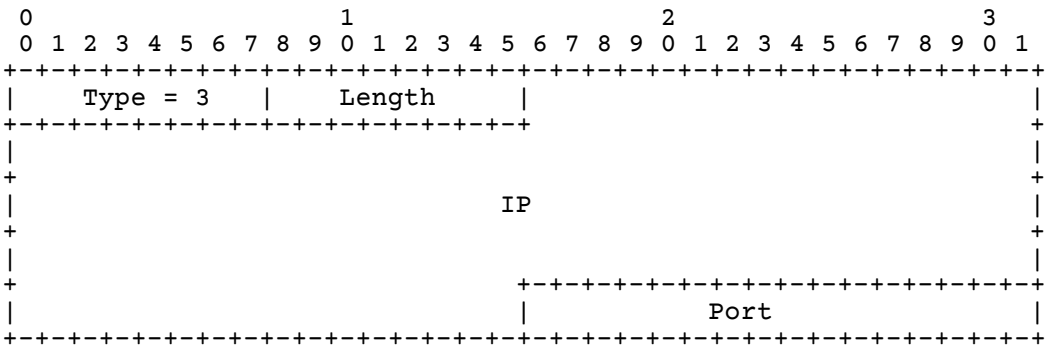
Ce TLV est ignoré à la réception. Le champ MBZ est une suite de zéros.

Hello



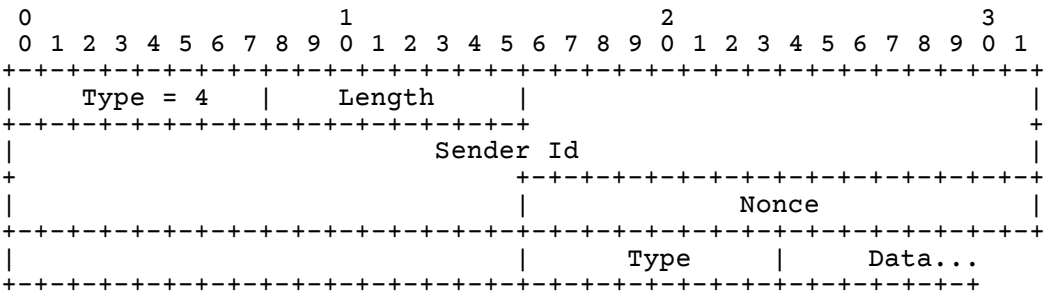
Ce TLV a deux formats : le format court (8 octets, sans compter l'entête) et le format long (16 octets). Un TLV *Hello* court est envoyé à un pair potentiel dont on ne connaît pas forcément l'*Id*. Un TLV *Hello* long est envoyé à chaque voisin toutes les 30 secondes environ, et sert à confirmer l'accessibilité bidirectionnelle. Le champ *Source-Id* contient l'*Id* de l'émetteur. Le champ *Destination Id* contient l'*Id* du destinataire.

Neighbour



Ce TLV indique que l'émetteur a une relation de voisinage symétrique avec un pair à l'adresse *IP* et écoutant sur le port UDP *Port*. Les adresses IPv6 sont représentées telles quelles, les adresses IPv4 sont représentées sous forme *IPv4-Mapped* (dans le préfixe `::FFFF:0:0/96`).

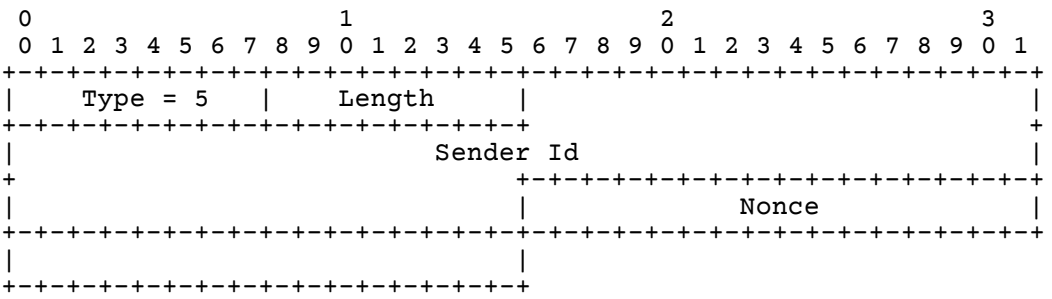
Data



Ce TLV contient des données envoyé au groupe de discussion. Le champ *Sender Id* contient l'*Id* de l'originateur du message. Le champ *Nonce* contient une chaîne de 32 bits arbitraire choisie de façon à ce que la paire (*Sender Id*, *Nonce*) soit globalement unique ; on peut par exemple utiliser un compteur séquentiel, la tirer au hasard, ou y coder l'heure d'émission avec une granularité suffisante pour qu'elle soit unique (mais attention si vous utilisez des *threads*).

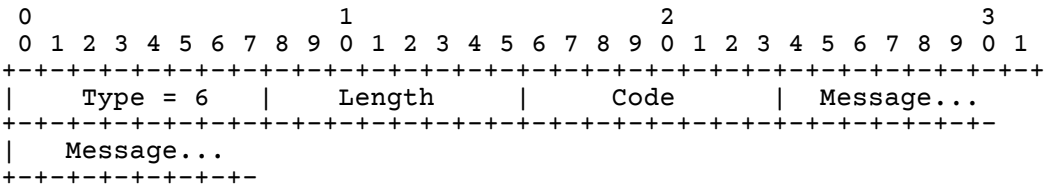
Le champ *Type* indique le type de données contenues dans le champ *Data*. Si *Type* vaut 0, alors le champ *Data*, de longueur arbitraire, contient les données à afficher dans le groupe de discussion ; il est de la forme « *nick* **:** *message* », où *nick* et *message* sont des chaînes de caractères codées en UTF-8 (mais attention — il faut l'inonder même si les données contenues n'obéissent pas à ce format). Si *Type* ne vaus pas 0, alors le champ *Data* contient des données inconnues — il faut alors inonder les données sans les afficher.

Ack



Ce message acquitte la réception d'un message *Data*. Les champs *Sender Id* et *Nonce* sont recopiés depuis le message *Data* correspondant.

GoAway



Ce message sert à supprimer explicitement une relation de voisinage, ce qui évite de passer du temps à inonder vers un voisin qui n'existe plus. Lorsqu'il reçoit un TLV *GoAway*, le récepteur marque l'émetteur comme un voisin non-symétrique, ou alors le supprime carrément de sa liste de voisins (mais il peut le conserver dans sa liste de voisins potentiels). Le champ *Code* indique la raison de la suppression, et il peut avoir les valeurs suivantes :

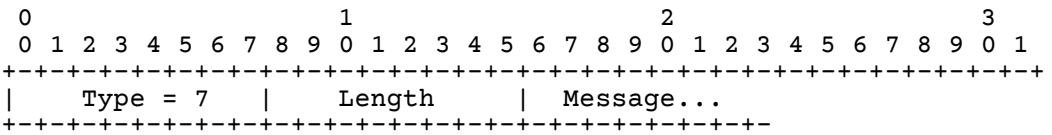
0 raison non-spécifiée ;
1 l'émetteur quitte le réseau (par exemple parce qu'il termine) ;
2 le récepteur n'a pas envoyé un Hello depuis trop longtemps ou alors il n'a pas acquitté un TLV *Data* dans les temps ;
3 le récepteur a violé le protocole ;
4 l'émetteur a déjà trop de voisins.

Les valeurs inconnues du champ *Code* doivent être traitées comme la valeur 0.

Le champ *Message* est optionnel, et il contient un message lisible par un être humain, codé en UTF-8. Il est utilisé par mon pair pour vous aider lors du déboggage.

> *Attention :* UDP est un protocole non-fiable, et par ailleurs un voisin peut très bien quitter le réseau sans l'indiquer à la couche application. L'utilisation du message *GoAway* est une optimisation utile, mais vous ne pouvez pas compter sur sa réception.

Warning



Ce TLV permet d'envoyer un message pour l'implémenteur ; mon pair l'envoie si votre pair fait des choses bizarres mais techniquement légales. Le champ *Message* contient un message lisible par un être humain, codé en UTF-8.

Autres TLV Un TLV ayant un type inconnu est silencieusement ignoré en réception, ce qui permet d'étendre le protocole sans changer de numéro de version.

2.4 Déroulement du protocole

Chaque pair est initialisé avec son *Id*, une chaîne de 64 bits globalement unique ; on peut par exemple la tirer au hasard.

Maintenance de la liste de voisins Chaque pair est initialisé avec une liste de *voisins potentiels* (possiblement vide), indexée par des paires $(IP, Port)$. Il maintient par ailleurs une table de *voisins*, indexée par des paires $(IP, Port)$, de pairs dont il a récemment reçu un TLV Hello (notez que le même pair peut apparaître plusieurs fois dans la table de voisins, par exemple s'il a plusieurs adresses). Chaque entrée de la table de voisins contient l'*Id* du pair, ainsi que deux dates : la date de

réception du dernier *Hello* de n'importe quel type de ce voisin, et la date de réception du dernier *Hello* long (16 octets) de ce voisin.

Un voisin est dit *récent* s'il a envoyé un TLV *Hello* (long ou court) dans les dernières deux minutes. Il est *symétrique* s'il a envoyé un *Hello* long avec notre *Id* dans le champ *Destination-ID* dans les dernières deux minutes.

S'il n'a pas assez de voisins symétriques (par exemple moins de 8) un pair envoie un TLV *Hello* court (8 octets) à un ou plusieurs ses voisins potentiels. Un pair envoie toutes les 30 secondes environ un TLV *Hello* long (16 octets) à chacun de ses voisins (symétriques ou pas).

Lorsqu'il reçoit un *Hello*, court ou long, il insère le pair dans sa table de voisins et note la date de réception. De plus, lorsqu'il reçoit un *Hello* long et que l'*Id* de destination contenue dans le *Hello* long est égale à son *Id*, il note la date de réception du *Hello* long.

Si un voisin n'a pas envoyé de *Hello* depuis plus de deux minutes, on lui envoie un TLV *GoAway* de code 2 avant de le supprimer de la liste de voisins (mais il peut être conservé dans la liste de voisins potentiels). Si un voisin n'a pas envoyé de *Hello* long depuis deux minutes, il est marqué non-symétrique.

Un pair envoie « assez souvent » des TLV *Neighbour* à ses voisins les informant de ses autres voisins symétriques. La politique précise d'envoi de ces TLV est laissée à l'implémentation. Lorsqu'il reçoit un TLV *Neighbour*, le pair insère l'adresse contenue dans sa table de voisins potentiels.

Inondation Chaque pair maintient une liste de données récemment reçues, indexée par des paires de la forme $(Id, Nonce)$. Chaque entrée de cette liste contient une liste de voisins symétriques vers lesquels cette donnée doit encore être inondée et le nombre de fois qu'elle a été envoyée à ce voisin. La liste de données récemment reçues a sa taille limitée, par exemple en horodatant chaque entrée et en la supprimant au bout de quelques minutes, ou en utilisant une liste circulaire de taille fixée.

Lorsqu'il reçoit un TLV *Data* d'un voisin symétrique, le pair vérifie si la donnée $(Id, Nonce)$ se trouve dans sa liste de données récemment reçues. Si ce n'est pas le cas, il affiche la donnée à l'utilisateur. Dans ce cas, il insère la donnée dans la table de données récentes et initialise la liste de pairs à inonder à sa liste de voisins symétriques. Dans tous les cas (même si la donnée était déjà récente), il envoie un TLV *Ack* à l'émetteur. Il supprime l'émetteur de la liste de voisins à inonder, puis procède à l'inondation.

Pour inonder le message, le pair envoie le message à chacun des pairs à inonder après un temps aléatoire compris entre 2^{n-1} et 2^n secondes, où n est le nombre de fois que ce message a déjà été envoyé à ce voisin (remarquez qu'un temps d'attente est marqué même au premier envoi, ce qui permet de minimiser la probabilité qu'un message soit inondé simultanément par deux voisins). Lorsque le pair procédant à l'inondation reçoit un TLV *Ack* ou *Data* correspondant à la donnée inondée, ce voisin est supprimé de la liste de voisins à inonder. Le processus termine lorsque tous les voisins ont acquitté la donnée (la liste de voisins à inonder est vide). (Un TLV *Data* est considéré comme un acquittement, ce qui optimise le cas où une donnée est inondée simultanément par les deux voisins.)

Si le compteur de réemissions pour un voisin atteint 5, c'est que ce voisin est mort ou alors très lent ; on lui envoie alors un TLV *GoAway* de code 2 et le voisin est éliminé de la liste de voisins (mais peut être gardé dans la liste de voisins potentiels).

Envoi des données Lorsque l'utilisateur demande à envoyer un message, le message est inséré dans la liste de messages récents et inondé comme au paragraphe ci-dessus.

3 Sujet minimal

Vous devrez me fournir un programme qui implémente le protocole ci-dessus, implémentant notamment la découverte de voisins et l'inondation. La liste des langages de programmation autorisés vous sera communiquée par la suite. Vous êtes libres du choix de l'interface utilisateur, mais ne passez pas trop de temps à faire une interface graphique, vous serez évalués principalement sur la partie réseau.

Votre solution devra *obligatoirement* interopérer avec la mienne, que vous trouverez sur le hôte nommé `jch.irif.fr` sur le port UDP 1212 (une interface *web* tourne sur le port 8082). Votre solution devra *obligatoirement* implémenter la communication au-dessus de IPv6, et pourra optionnellement implémenter la communication au-dessus de IPv4.

Votre solution devra être accompagnée d'un rapport de quelques pages sous format PDF. Votre rapport devra notamment m'indiquer les choix d'implémentation que vous aurez faits, notamment les détails de votre politique de découverte de voisins et les optimisations que vous aurez apportées au processus d'inondation.

4 Extensions

Je m'attends à ce que vous implémentiez des extensions au sujet minimal ci-dessus, et j'évaluerai vos extensions avec intérêt et bienveillance. Cependant, même étendue, votre implémentation devra rester interopérable avec le sujet minimal défini ci-dessus — toutes les extensions devront rester compatibles. Si vous avez besoin d'un numéro de TLV pour votre extension, annoncez votre besoin sur la liste de diffusion et je vous en affecterai un.

Ci-dessous quelques idées d'extensions au sujet.

Gestion des erreurs Comment votre implémentation réagit-elle à une indication de couche inférieure (due à une erreur locale ou à la réception d'un message ICMP) ?

Concurrence Votre implémentation est-elle capable d'effectuer plusieurs inondations simultanément, ou est-ce qu'une indondation bloque le pair jusqu'à ce qu'elle termine ?

Agrégation Le protocole permet d'agréger plusieurs TLV dans un seul message. Votre implémentation agrège-t-elle les TLV de façon optimale ? Qu'est-ce que ça veut dire ? (Clairement, il y a une tension entre latence et efficacité.) Est-ce qu'elle est capable d'agréger des TLV appartenant à plusieurs inondations différentes ? Votre implémentation construit-elle des paquets allant jusqu'au PMTU correspondant au voisin, ou se limite-t-elle à 1024 octets de charge ?

Optimisations Les performances de ce protocole dépendent des détails de l'algorithme d'inondation. En particulier, si une donnée est inondée simultanément vers plusieurs voisins, on risque

de l'envoyer plusieurs fois ; si les temps d'attente sont tirés indépendamment pour les différents voisins, il existe une possibilité de ne l'inonder qu'une fois. Une autre optimisation utile est de détecter qu'un pair participe à plusieurs relations de voisinage, et d'envoyer la donnée à l'une de ses adresses, et, un peu plus tard, un acquittement aux autres.

L'algorithme proposé ci-dessus utilise des *timeouts* fixés. Une optimisation possible est de mesurer le temps que met chaque pair à acquitter le message, et utiliser un *timeout* dynamique ; un algorithme possible est celui utilisé pour le calcul du *RTO* dans la RFC 793. Si on connaît les RTT, on peut peut-être aussi optimiser l'inondation en privilégiant les liens rapides.

Adresses multiples Si votre pair a plusieurs adresses (par exemple parce qu'il a plusieurs interfaces), communique-t-il toujours avec un voisin donné en utilisant la même adresse ? Votre pair devient-il voisin de lui-même ? Que se passe-t-il si un voisin a plusieurs adresses ? (Il faudra probablement ajouter un champ aux entrées de la table de voisins.)

Sécurité de l'implémentation Votre implémentation est-elle robuste face aux pairs boggués ou malicieux ? Par exemple, que se passe-t-il si un message a une longueur supérieure à celle du datagramme UDP reçu, ou si un TLV déborde du message ?

Votre implémentation accepte-t-elle un message injecté par un pair qui n'a pas établi de relation de voisinage par un échange de Hellos ? Est-ce qu'elle rejette correctement les messages de pairs qui ne connaissent pas votre *Id* ? (Comme un *Id* consiste de 64 bits, il est difficile de le deviner sans faire un échange, ce qui complique un petit peu la tâche d'un attaquant.)

Sécurité cryptographique Autentification ? Chiffrage ? Remarquez que l'inondation est opaque (elle n'interprète pas les messages), il est donc possible d'implémenter a sécurité de bout en bout. Si vous préférez la sécurité de saut en saut, vous pouvez soit utiliser DTLS, soit concevoir votre propre protocole. Attention cependant à l'interopérabilité avec les pairs non sécurisés.

Découverte multicast Un Hello court peut être envoyé en multicast. Je vous propose d'utiliser le port 1212 et le groupe multicast `ff12:b456:dad4:cee1:4589:71de:a2ec:e66`. Je l'ai tiré au hasard, il est donc globalement unique. Si vous arrivez à faire fonctionner le multicast au-delà du lien local, racontez-moi comment vous avez fait.

Fragmentation Ce protocole est limité à des messages d'un peu moins de 250 octets. Comment l'étendre à des messages d'une taille illimitée ? Vous pouvez soit utiliser TCP, soit implémenter votre propre protocole de fragmentation, ce qui est plus amusant, par exemple en codant les fragments dans un nouveau TLV ou dans un TLV Data avec un champ Type différent de 0 (dans les deux cas, annoncez votre intention sur la liste de diffusion pour éviter les collisions de numéro de TLV ou Type.)

Données non textuelles Une extension naturelle est de permettre d'envoyer des images (par exemples des avatars). Vous pouvez utiliser un TLV Data avec Type différent de 0 pour coder les données non-textuelles.

Contrôle de la congestion Votre protocole est-il *TCP-Friendly*, c'est-à-dire se limite-t-il à sa part de la capacité du lien lorsqu'il est en concurrence avec TCP, et cela même lorsqu'il inonde une quantité importante de données ?

Transfert de fichiers Étendez le protocole pour permettre le transfert de fichiers. Vous pouvez soit utiliser TCP, soit construire votre propre protocole fiable au-dessus d'UDP. Attention cependant au contrôle de congestion.

Traversée de Firewalls, de NAT En IPv6, le protocole PCP permet de traverser les *firewalls*, mais il est rarement implémenté. En IPv4, PCP ou NAT-PMP permettent de traverser les NAT, mais c'est NAT-PMP qui est généralement implémenté. N'implémentez en aucun cas uPNP, c'est un protocole mal conçu qu'il faut laisser mourir en paix. Regardez aussi si vous avez envie d'implémenter un client STUN pour la traversée de NAT.