TP10 : Recherche de pairs en UDP
pour un protocole pair-à-pair

Matthieu Boutier *(boutier@pps.univ-paris-diderot.fr)*

Une des problématiques des protocoles pair-à-pair, comme BitTorrent, est de toujours garder sous la main un certain nombre de pairs. La recherche de nouveaux pairs se fait généralement en demandant aux pairs déjà connus la liste de leurs pairs. Nous implémentons dans ce TP un protocole de recherche de pairs, le premier étant renseigné par l'utilisateur.

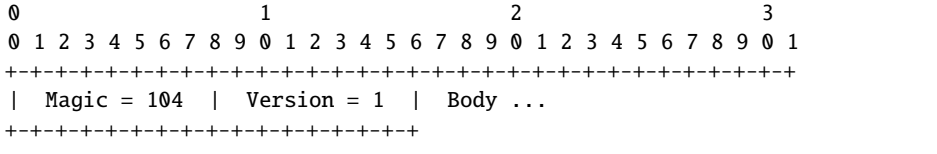
(Les questions sont en deuxième partie de ce sujet.)

1 Description du protocole

Nous enverrons toutes les 10 secondes un *Peer Request* à chacun de nos pairs, et toutes les secondes un *Hello*. Si un pair ne nous donne pas de *Hello* dans les 4 secondes, il sera considéré inactif, et pourra être supprimé.

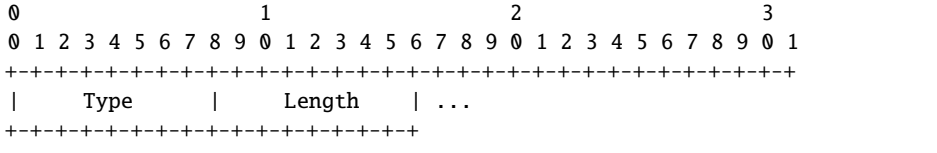
1.1 En-tête

Tout message doit commencer par cet en-tête. Lorsqu'un message est reçu avec un en-tête différent, il doit être ignoré. Le champ *Body* est une liste de TLV, décrits ci-après.



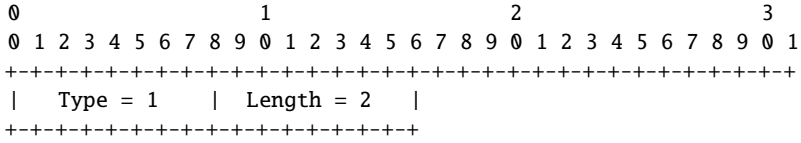
1.2 Format général des TLVs

Les TLV sont constitués d'au moins deux octets : le premier, *Type*, identifie le type de message, tandis que le deuxième, *Length*, donne la taille totale du TLV (y inclus les deux champs *Type* et *Length*). Les messages de type inconnus sont silencieusement ignorés. Si un TLV a des octets supplémentaires par rapport à la présente spécification, ils seront simplement ignorés.



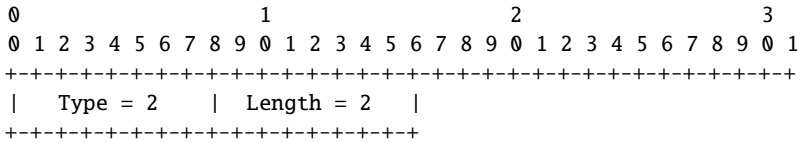
1.3 Bonjour (Hello TLV)

Ce TLV doit être envoyé toutes les secondes à chaque pair. Un pair qui ne reçoit pas de Hello pendant 4 secondes d'un pair donné considèrera que ce dernier est déconnecté. À la réception de ce TLV, la date de dernier message reçu de ce pair est mis à jour (si le pair n'est pas référencé, il est ajouté).



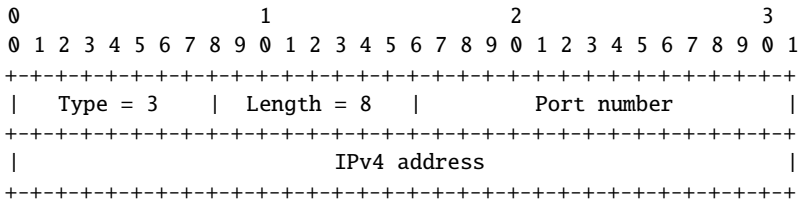
1.4 Demande de pair (Peer Request TLV)

À la réception d'un tel TLV, un pair doit envoyer la liste complète de tous les pairs auquel il est connecté, à l'aide du *Peer Address TLV*. Il devrait être envoyé à une fréquence régulière, disons toutes les 10 secondes (mais ce choix est laissé à l'implémentation).



1.5 Envoi d'adresse (Peer Address TLV)

Ce message contient l'adresse d'un pair. À la réception d'un tel TLV, l'adresse est ajoutée à la liste des pairs connus. Aucun autre traitement particulier n'est requis : la demande de la liste de ses pairs, et l'envoi d'un *Hello* sera faite en temps et en heure, en même temps que les autres demandes.



2 Guide de programmation

Vous trouverez un fichier `server.c` (sur Didel ou ma page web) qui implémente déjà un certain nombre de fonctions [1].

Code fourni Mon programme a pour arguments le port local, l'adresse d'un pair distant, et son numéro de port. Par exemple, `./server 4201 127.0.0.1 4200` se lie au port 4201, et envoie des messages à `127.0.0.1:4200`.

1. Vous admirerez le style et l'élégance du code, ainsi que l'humilité de l'auteur.

2.1 Envoi régulier de messages

Envoyer régulièrement des messages à nos pairs n'est pas évident : lorsque nous attendons un message avec `recvfrom`, tout est bloqué. Voilà ce que nous allons faire : nous envoyer de "faux" messages régulièrement, à l'aide d'autres processus.

1. Complétez la fonction `send_regular_value`.
2. Utilisez cette fonction pour vous envoyer un message de valeur 1 toutes les secondes.
3. Récupérez ce message dans votre boucle principale, et afficher qu'il faudrait envoyer des Hellos.
4. Complétez la fonction `format_header`, qui écrit l'en-tête du protocole dans le buffer, et renvoie la zone pointée juste après.
5. Complétez la fonction `send_hellos`, puis utilisez-là à la place de votre affichage.
6. Commencez à compléter `parse_msg` pour qu'elle reconnaisse l'arrivée d'un Hello, ajoute le pair s'il n'existe pas, et mette à jour sa date de dernière réception autrement.

À ce stade, vous devriez pouvoir établir le contact entre deux pairs.

2.2 Plus on est de fous, plus on rit

Allé, il est temps de partager ses connaissances !

1. Comme précédemment, utilisez `send_regular_value` avec une valeur de 2 et un intervalle de 4 secondes, écrivez `send_requests`, et envoyez des requêtes régulières à vos pairs.
2. Modifiez `parse_msg` pour recevoir les requêtes.
3. Écrivez `send_peers` pour envoyer tous vos pairs dans des messages. Attention à la taille maximale des messages. On utilisera un buffer d'au plus 1400 octets.
4. Finissez le traitement des requêtes dans `parse_msg`.
5. Utilisez `remove_old_peers` pour libérer la mémoire des vieux pairs (ou des pairs donnés par des implémentations boguées).

2.3 Et maintenant ?

Il y a toujours moyen d'être plus malin : garder des pairs inactifs en mémoire lorsqu'on n'en a pas d'autres (garder un minimum de pairs), préférer les pairs les plus stables, etc. À vous de voir !