Projet de Programmation Réseau
Parcours de la DHT BitTorrent

Guillaume Matheron, Yann Ramusat

1 Description du programme

Le programme effectue un parcours de la DHT le plus exhaustif possible (environ un million de noeuds) en partant du bootstrap "router.bittorrent.com" de BitTorrent.

Le réseau ainsi découvert est stocké au format csv pour pouvoir être ensuite étudié à l'aide d'outils tels que Octave ou Matlab.

Comme la taille des données que le programme traite est conséquente, le code optimise le plus possible les accès au réseau et la place occupée par les données en vol.

1.1 Choix du langage

Notre programme est écrit en C++.

Ce choix est motivé par plusieurs raisons, la première est sa proximité avec le langage C que nous avons utilisé en cours. Ceci nous a permis d'approfondir et de réutiliser le plus efficacement possible les codes que nous avions eu à faire en TD.

Le C++ propose aussi de nombreuses structures de données implémentées efficacement (set, list, vector, ...) et des primitives d'entrées-sorties très pratiques (cout, streams) qui nous ont permis de plus nous concentrer sur le sujet en lui-même et moins sur la gestion des structures de données annexes.

A noter que les threads sont plus faciles à manipuler en C++ qu'en C et l'utilisation de l'orienté objet permet de plus facilement collaborer à deux sur un projet.

Le C++ est un langage standard, très répandu, ce qui apporte une certaine sécurité quant à la portabilité du code.

Finalement, le C++ semble être le langage le plus approprié pour mener à bien ce projet, nous permettant le plus de liberté possibles tout en restant dans la lignée du cours.

1.2 Architecture générale

Le programme a accès à un pool de threads (200) pouvant chacun gérer indépendamment une liste de 300 noeuds distants à contacter.

L'objectif de la classe explore est de dispatcher les noeuds distants (et non encore visités !) à contacter aux threads. Chacun sera ensuite chargé (fonction work()) de contacter les noeuds, de récupérer leur éventuelle réponse et de la traiter.

Les données partagées par les différents threads sont sécurisées à l'aide de mutex et de conditions.

C'est notamment le cas de la queue où résident les noeuds distants en attente d'être traités. Tous les threads y rajoutent les nouveaux voisins qu'ils viennent de recevoir. Si la queue est vide, retirer un objet fait faire une attente passive. Ceci permet d'optimiser l'utilisation du processeur.

1.3 Partie réseau de l'implémentation

Chaque thread utilise sa propre socket IPv4 pour contacter les noeuds qui lui ont été attribués et retourne les informations reçues au fur et à mesure qu'il les obtiens.

La requête utilise le champ adéquat pour détecter les voisins disponibles implémentant la version BEP32 du protocole.

Nous avons créé plusieurs classes destinées à manipuler les données du réseau :

1. CNI (Compact Node Information) qui contient les informations de la DHT associées à chaque noeuds
2. IP pour rendre la manipulation des IP plus simples.
3. UDP qui permet d'effectuer toutes les primitives d'envoi de messages.

1.4 Stockage des informations

Etant donné que le programme parcours en moyenne un million de noeud de la DHT à chaque parcours, régulièrement, les données collectées sont stockées dans le fichier de sortie généré automatiquement au début du programme en fonction de la date actuelle.
Exemple de sortie : 2015_11_18_19_56_20.csv.

En effet, le programme maintient en mémoire un ensemble (via la structure de données set) de statistiques de chaque noeud déjà contacté.

Ces informations ne sont plus utiles pour le programme et comme les statistiques sont vouées à être étudiées grâce à un script matlab, périodiquement, ces données sont stockées dans le fichier de sortie.

2 Statistiques obtenues

2.1 Méthodes d'exploration

Méthode 1 On envoie exactement une requête à chaque noeud découvert. Il s'agit d'un find_nodes avec un identifiant aléatoire. Les noeuds restants sont ajoutés à la file. Le programme se termine quand la file est vide.

2.2 Statistiques temporelles

Nous avons lancé un script qui explore la DHT via la méthode 1, avec un cadençage de 200 threads contactant 300 noeuds chacun, acceptant des réponses durant 2 secondes avant de fermer le socket.

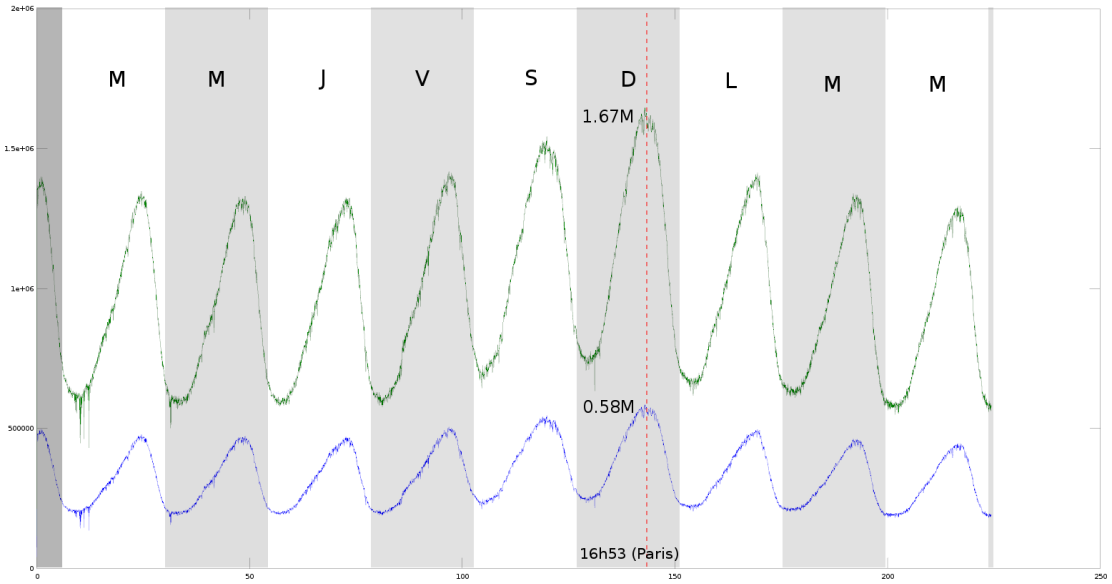

FIGURE 1 – Au cours d'une semaine environ (en moyenne uun échantillon toutes les 4 minutes). En vert : nombre d'identifiants récoltés (= nombre de noeuds contactés). En bleu : nombre de noeuds ayant répondu.

Ce cadençage très rapide a permis d'explorer beaucoup de noeuds très rapidement et de faire des statistiques avec une grande fréquence temporelle. Malheureusement, les buffers UDP ont parfois été saturés sur le serveur VPS utilisé. Il est impossible d'augmenter leur taille, le serveur étant une machine virtuelle.

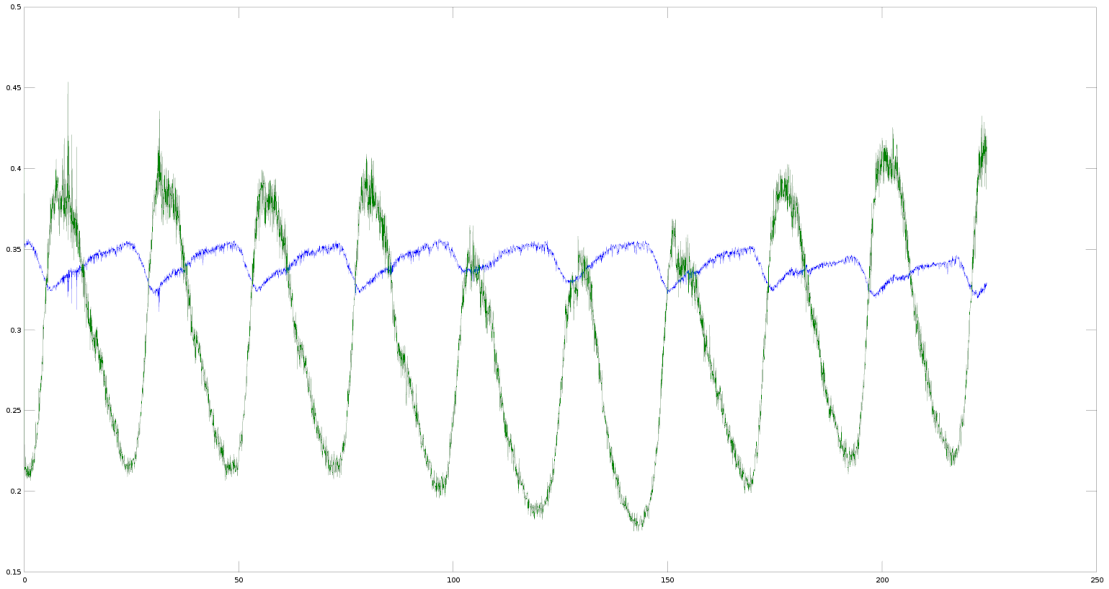

FIGURE 2 – Au cours de la même période que la Figure 2.2. En bleu : fraction de noeuds contactés répondant aux requêtes (environ 35%). En vert : fraction de noeuds contactés implémentant BEP-32 (×100), c'est-à-dire environ 0.3%.

Pour tester si un noeud implémente BEP-32, on lui envoie une requête find_nodes avec un champ want demandant des ipv4 et des ipv6. On compte alors les réponses contenant un champ nodes6.

2.3 Statistiques plus exhaustives

Le problème principal des statistiques ci-dessus est le nombre d'erreurs UDP. La vitesse d'envoi rapide cause beaucoup d'erreurs de buffer.

De plus, certains noeuds man connectés peuvent ne pas être trouvés, car on envoie uniquement une requête par noeud.

Pour remédier à ces problèmes, nous avons lancé le programme avec de nouveaux paramètres : cadence plus faible et envoi de 10 requêtes à chaque noeud.

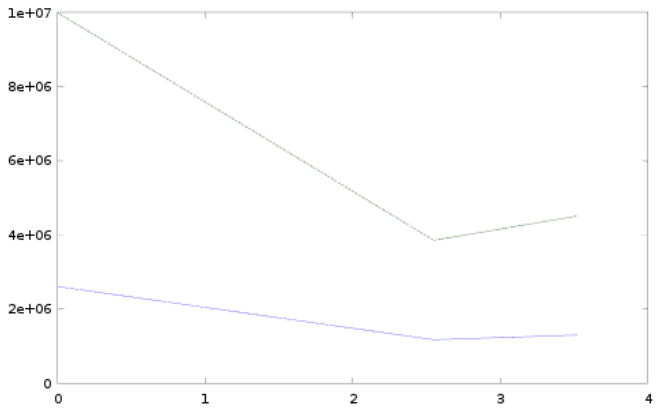

FIGURE 3 – Au cours d'une semaine environ (en moyenne uun échantillon toutes les 4 minutes). En vert : nombre d'identifiants récoltés (= nombre de noeuds contactés). En bleu : nombre de noeuds ayant répondu.

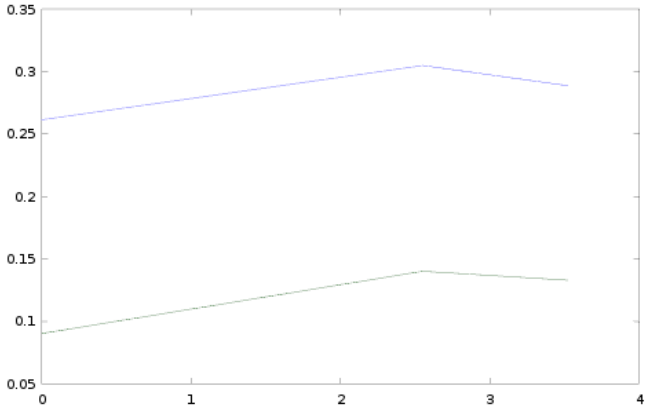

FIGURE 4 – Au cours de la même période que la Figure 2.3. En bleu : fraction de noeuds contactés répondant aux requêtes (environ 30%). En vert : fraction de noeuds contactés implémentant BEP-32 (×100), c'est-à-dire environ 0.1%.

Ces résultats sont proportionnellement proches de ceux obtenus avec les autres paramètres. On en déduit qu'ils sont assez représentatifs.

3 Conclusions

Cette exploration de la DHT de Kadelmia donne plusieurs résultats :

— Le nombre de noeuds actifs est environ doublé entre 5h et 17h (heure de Paris) ;
— Le nombre de noeuds actifs et accessibles à un instant donné est de l'ordre de grandeur de 2 millions ;
— Le nombre de noeuds dont l'identifiant est connu est de l'ordre de grandeur de 10 millions.

— La fraction de noeuds implémentant BEP-32 est de l'ordre de 0.1%. Les noeuds les plus actifs sont plus suceptibles l'implémenter BEP-32 (lors des périodes où peu de noeuds sont actifs, la fraction de noeuds implémentant BEP-32 augmente).
— Le nombre de noeuds implémentant BEP-32 est de l'ordre de 3000.