1 Objectifs

L'objectif de ce TP est de développer une application basique de partage de fichiers sur un réseau P2P utilisant une DHT. Nous prendrons ici la DHT *Kademlia* [1] qui est notamment utilisée au sein du réseau *Bittorent* [2].

1) Récupérez le fichier local /net/ens/vince/dht/chatserver.py.

2 Mise en jambe

2.1 Principe

La version de la bibliothèque *Kademlia* que nous allons utiliser est basée sur un *framework* réseau événementiel *Python* [3] appelé *Twisted* [4].

Nous allons en premier lieu regarder comment ce *framework* fonctionne sur un exemple simple : un *chat server*.

2.2 Le chat

Nous allons partir sur le squelette qui se trouve dans le fichier **chatserver.py**.

2) Dans la fonction **main**, nous allons créer un nouveau serveur vide.

```
factory = protocol.ServerFactory()
factory.clients = []
```

3) Ensuite, nous allons déclarer une classe (en dehors du bloc **main**) qui contiendra le comportement de notre serveur. Notre classe doit hériter de la classe **Protocol** fournie par *Twisted*.

4) Nous indiquons ensuite à notre serveur d'utiliser notre classe (dans la fonction **main**).

```
factory.protocol = <Nom de votre Classe>
```

5) Nous devons ensuite spécifier un port d'écoute et lancer le moteur.

```
reactor.listenTCP(tcpPort,factory)
reactor.run()
```

1. http://pdos.csail.mit.edu/~petar/papers/maymounkov-kademlia-lncs.pdf
2. http://www.bittorrent.org
3. http://python.org
4. http://twistedmatrix.com

Vincent Autefage
autefage@labri.fr

6) Chaque nouvelle instance de notre classe va représenter un nouveau client. Maintenant que le moteur est configuré, nous devons écrire le comportement du serveur. Pour cela, nous allons surcharger 3 fonctions :

```
connectionMade(self)
# Ajoute le nouveau client dans la liste self.factory.clients
connectionLost(self, reason)
# Retire le nouveau client de la liste self.factory.clients
dataReceived(self, data)
# Transfert la data sur chaque clients
# On peut contacter un client au travers de sa socket self.transport
```

7) Lancez votre serveur et testez le avec plusieurs clients.

8) Essayez de supprimer l'effet *écho* pour l'émetteur du message.

Rappel :

```
$ netcat <ip> <port> # permet de se connecter à un serveur et de communiquer avec stdio
```

3 La DHT

3.1 Premier contact

Maintenant que vous avez vu comment *Twisted* fonctionne, nous allons passer à notre programme de partage. Dans celui-ci, vous n'aurez que les parties relatives à la DHT à implémenter. En d'autres termes on s'intéresse aux fonctions **get**, **put** et **remove**.

Nous allons nous intéressez ici au programme **shared.py** ainsi qu'à la bibliothèque de *Kademlia*. Ce programme est un *démon*, c'est à dire qu'une fois lancé, il tourne indéfiniment tant que l'on ne l'arrête pas (sauf si il plante, bien entendu...). Il est manipulable au travers d'une *socket TCP* par un client local.

9) Récupérez le fichier local /net/ens/vince/dht/shared.py.

10) Examinez bien son fonctionnement, et comprenez à quoi sert chacune des classes, méthodes et fonctions.

11) Ajoutez une commande *help* dont la fonction est d'afficher un texte d'aide à l'utilisateur. Servez vous pour ce faire de la fonction **gprint**.

12) Vous remarquez que les méthodes **search**, **publish** et **clean** sont incomplètes. Elles correspondent en réalité aux fonctions **get**, **put** et **remove** de la DHT. Nous allons implémenter ces méthodes dans l'ordre.

3.2 search (get)

13) La première chose à faire et de générer le *hash* du mot clé à l'aide des instructions suivantes.

```
h = hashlib.sha1()
h.update(keyword)
key = h.digest()
```

Vincent Autefage
autefage@labri.fr

14) Il faut ensuite effectuer une requête à la DHT grâce à la fonction suivante.

```
df = self.node.iterativeFindValue(key)
```

15) La variable **df** devient un objet *Twisted* auquel il faut appliquer des *callback*. Un *callback* d'erreur générique est déjà écrit : c'est la méthode **errcallback** de la classe **FileShare**. Il suffit donc de brancher le *callback* d'erreur sur **df**.

```
df.addErrback(self.errcallback)
```

16) Il faut enfin écrire un *callback* interne à la méthode **search** (c'est à dire une sous-méthode) que l'on appellera par example **ok**. Cette sous méthode ne possède qu'un seul argument **result**. Le fonctionnement de cette méthode est simple : si l'argument est de type **dict**, l'élément recherché se trouve sur le nœud **result[key]** (sous forme binaire). Sinon, la clé n'existe pas dans la DHT.

```
def ok(result):
    if isinstance(result, dict):
        gprint(keyword + " found.")
    else:
        gprint("Cannot find " + keyword)
```

17) On ajoute en dernier lieu, le fameux *callback* de succès.

```
df.addCallback(ok)
```

3.3 publish (put)

18) La méthode **publish** est plus complexe. Le but est ici de partager un répertoire. Seule la méthode de publication, c'est à dire **publishNextFile** vous est demandée. Le principe est d'effectuer un **pop** sur la liste **files** tant qu'elle n'est pas vide. Pour chaque élément *popé*, vous devez :

1. Générer son *hash*,
2. Effectuer un **Put** dans la DHT à l'aide de l'instruction suivante.

   ```
   df = self.node.iterativeStore(<hash key>, self.node.id)
   ```

3. Ajouter son nom dans la liste **self.files**,
4. Enregistrer un *callback* sur **publishNextFile**,
5. Enregistrer un *callback* d'erreur sur **self.errcallback**.

Attention : il n'y a pas de boucle à faire, le bouclage s'effectuera automatiquement grâce aux *callback*. Il est donc important de ne pas recréer de *callback* quand la liste **files** est vide (i.e. tous les fichiers ont été publiés).

19) Lorsque la liste **files** est entièrement vidée, transférez au client le nombre de fichiers publiés à l'aide de la fonction **gprint**. Effectuez également un **return** pour stopper la boucle d'appels récursifs.

Vincent Autefage
autefage@labri.fr

3.4 clean (remove)

20) Cette méthode suit la même logique que la fonction de publication mais dans le sens inverse. Implémentez la. La fonction de nettoyage dans la DHT est la suivante.

```
df = self.node.iterativeDelete(<hash key>)
```

4 Test

21) Lancez votre démon.

22) Pour le contacter, utilisez **netcat**.

23) Essayez de lancer plusieurs couples *démon/client* reliés entre eux et effectuez des tests de publication/téléchargement.

24) Essayez maintenant de créer un réseau de partage unique sur toute la salle et partagez, c'est gratuit !

5 Bonus

25) Créez un programme *client* pour communiquer avec votre démon.

Vincent Autefage
autefage@labri.fr