Implémentation

En mode « client » la fonctionnalité suivante est disponible :
ServerInfo – afficher la liste de pairs connus par le serveur (noms, adresses)
PeerInfo – afficher l'info sur un pair (nom, adresses, clé, root)
HashesInfo – afficher tous les hashes disponibles pour le pair donné (les hashes de chaque nœud de son arborescence)
DownloadHash – télécharger les données par hash
DownloadPath – télécharger les données par chemin
Traversé de NAT

En mode « server » la fonctionnalité suivante est disponible :

Algorithmes

Analyse du protocole

Le protocole basé sur UDP est sujet à un certain nombre de problèmes potentiels et le protocole est conçu pour résoudre ces problèmes. Parmi eux, on peut distinguer les suivants :
1) Contrôle de l'intégrité des données, par défaut UDP ne garantit pas l'intégrité.
2) Changement l'ordre des datagrammes lors de la réception.
3) Perte de datagrammes lors de l'envoi et de la réception.
4) Fragmentation des paquets, bien que la plupart des équipements prennent en charge MTU 1460, il existe des équipements où MTU 576 octets. En conséquence, un paquet de 1024 octets peut être fragmenté en 2 paquets et les problèmes 1 à 3 peuvent s'appliquer à ces deux paquets.

La conception du protocole permet de résoudre tous les problèmes susmentionnés :
1) Résolu par plusieurs champs : id, taille et signature.
2) L'ordre des datagrammes est résolu par le modèle demande-réponse. Il n'y a qu'une seule réponse pour un demande.
3) Perte de datagramme de/vers le client est résous par le délai d'attente de réponse.
4) Fragmentation est partiellement résolue par le point #1, en partie peut être résolue par concaténation des données entrantes. Étant donné que la taille des paquets ne dépasse pas la taille de 576*2, les fragments peuvent être un maximum de 2, ce qui rend le problème de l'union résolu.

Algo1. Vérification d'intégrité d'UDP paquet

Base sur analyse, nous avons implémenté une méthode de vérification de l'intégrité suivante: on vérifie d'abord que le paquet arrivé est de longueur prévu, puis on vérifie que le type de message correspond au ceux qui est attendu, puis on vérifie l'Id de message (pour les messages reply).

Length	Type	Id	Action
Not ok			Renvoyer Hello
Ok	Not ok		Attendre le message HelloReply suivant
Ok	Ok	Not ok	Renvoyer Hello
Ok	Ok	Ok	Sortir tout heureux

Rien n'est reçu pendant TIMEOUT			Notifier l'utilisateur, puis réessayer de renvoyer Hello (il y a 180 seconds pour ces tentatives, si pendant ce temps on ne réussit pas à recevoir le bon HelloReply, on réenregistre sur le serveur/pair à nouveau)

Algo2. Parcours d'arborescence et téléchargement de données

La fonction **DownloadData** exerce à la fois l'affichage d'arborescence des hashes (**HashesInfo**), le téléchargement par hash (**DownloadHash**) et le téléchargement par chemin (**DownloadPath**), puisque, en principe, dans ces trois cas, nous devons parcourir l'arborescence entier à partir du hash initial.
Le moyen d'utilisation de cette fonction s'est définit par les paramètres OP_PRINT_HASH, OP_DOWNLOAD_HASH et OP_DOWNLOAD_PATH.

La fonction **GetDataByHash** effectue les demandes par hash auprès de serveur/pair, puis on parce les messages reçus avec la fonction **ParceValue** qui sauvegarde temporairement l'information sur l'objet dans la structure **StrObject**, qui a les champs suivants :

- **Type** d'objet, qui peut être CHUNK, BIG_FILE or NODE_UNKNOWN (quand on parce le contenu de DIRECTORY on ne connais pas si ce sera un fichier ou le dossier)
- **Name**, qui est "" quand pour CHUNK ou BIG_FILE
- **NbHash**, nombre de hashes: 1 pour CHUNK et le contenu de DIRECTORY, et > 1 pour BIG_FILE
- **Hash**, hash de données
- **Data**, données, qui sont nil pour BIG_FILE et le contenu de DIRECTORY

Puis, soit on construit le chemin et appelle **DownloadData** pour chaque objet qui se trouve dans de DIRECTORY, soit **DownloadData** pour chaque morceau de BIG_FILE, soit on crée le fichier pour CHUNK.

Pour créer les fichiers, on utilise la structure **DataObject**, qui a les champs suivants :

- **Op**, opération, peut être OP_PRINT_HASH, OP_DOWNLOAD_HASH et OP_DOWNLOAD_PATH
- **Type** d'objet, peut être CHUNK, BIG_FILE or NODE_UNKNOWN (quand on parce le contenu de DIRECTORY on ne connais pas si ce sera un fichier ou le dossier)
- **Name**, nom d'objet qui est "" quand pour CHUNK ou BIG_FILE
- **Path**, chemin distant sur le pair
- **SearchPath**, chemin de recherche pour télécharger
- **HddPath**, chemin pour stocker les données sur le disque dur
- **Handle,** pointeur sur le fichier crée

En cas de OP_PRINT_HASH on juste affiche les chemins obtenus pendant le parcours.

En cas de OP_DOWNLOAD_PATH on vérifie à chaque étape de construction de chemin, si le SearchPath correspond au chemin courant. Si c'est le cas, on met OP_DOWNLOAD_HASH comme opération et donc on commence à télécharger à l'appelle suivant de la fonction **DownloadData**.

Contributions

BRAGINA Natalia :

Enregistrement auprès le serveur
Sujet minimal, i.e. client mode: ServerInfo, PeerInfo, HashesInfo, DownloadHash, DownloadPath
Traversé de NAT coté client
Cryptographie (voir deuxième version du projet)

MESSIKH Abdennour :

Arbre de Merkel
Read de config
Server mode