Goatsoft Corporation présente :

GLOSTER©

Rapport de 3eme soutenance

Par les créateurs de Tuxout :

Guillaume "CrYpToNyM" MOISSAING (moissa_g)
François "Meteoryte" GOUDAL (goudal_f)
Freddy "Loki" SARRAGALLET (sarrag_f)
Guillaume "Z" LAZZARA (lazzar_g)

Mai 2005

Bitou@InfoSpe-Promo2008

"Où que nous mène la baisse de confiance qui nous occupe, il est nécessaire d'imaginer toutes les voies de bon sens" (un visionnaire)

Table des matières

1 Introduction

Nous avions commencé le developpement des differents composant du reseau ainsi que l'intégration pour la soutenance précédente. Pour celle-ci, il nous fallait, pour la partie reseau, finaliser le chat pour l'échange d'informations lors de la connexion d'un node, et faire une ébauche de l'interface web. La partie transfert de fichiers a aussi été commencée, avec la gestion des fichiers partagés, la recherche de fichiers, et la gestion de la cmpression. Il nous fallait aussi fixer le design de la gui et poursuivre l'intégration.

Tout ceci ayant été réalisé, nous sommes donc a peu près dans les temps par rapport à ce que nous avions planifié dans le cahier des charges.

2 Reseau

2.1 Partage de fichier

2.1.1 XML

Pour stocker les informations sur les fichiers partagés et les fichiers en cours de téléchargement, il a fallu qu'on décide d'un format facilement exportable, notre logiciel devant être multi-plateforme. Il se trouve que notre choix s'est tourné vers le standard d'exportation de données du moment : le XML.

Les fichiers XML sont assez facile à réaliser : la syntaxe se compose d'un ensemble de balises juxtaposées à l'image du HTML. Il suffit de connaître quelques règles, seulement, pour faire un fichier correct et lisible par un lecteur reconnaissant nos balises. L'avantage de tels fichiers, c'est qu'il suffit de faire un simple parser pour traiter les données de la manière qu'on veut. Ainsi, si par exemple, vous avez plusieurs fichiers XML générés par un éditeur de texte, vous pouvez très facilement faire un utilitaire qui va tous les parser et vous donner des statistiques sur l'ensemble de vos fichiers.

Nous avons donc décidé d'utiliser le XML pour les fichiers contenant la liste de fichiers partagés, ainsi que pour le fichier d'information sur les fichiers téléchargés.

Informations en XML pour un fichier en cours de téléchargement
Pour ces fichiers la, nous avons créé un nouveau type de document XML : le Gloster-Temp-File. Dans ce fichier, on retrouve trois balises différentes : file,chunk et part.
Chacune d'elles nous apporte des informations sur l'etat du telechargement du fichier. Voici donc quelques détails supplémentaires sur notre type de fichier :

- Balise <file> :
  - size : taille du fichier
  - priority : priorité du téléchargement avant la fermeture du programme
  - isDownloading : Indique si le téléchargement avait déjà commencé ou pas
  - name : nom du fichier en cours de téléchargement

  - hash : hash du fichier complet

- Balise <chunk> :

  - n : numéro du chunk

  - state : état du chunk - peut prendre les valeurs 0,1,2 pour respectivement vide,en cours de téléchargement,téléchargé

  - hash : hash du chunk

- Balise part :

  - p<i> : état du part numero i; peut prendre les même valeur que l'état des chunks.

Un chunk correspond à un morceaux du fichier d'une taille infèrieur ou égale à 9Mo. Un part, en revanche, correspond à un morceaux de chunk de taille infèrieur ou égale à 180Ko. Un fichier posséde donc au moins un chunk et un part lors de son téléchargement.
Voici un exemple de fichier XML pour un fichier en cours de telechargement :

```
<!DOCTYPE Gloster-Temp-File>
<file size="6136031" priority="2" isDownloading="0"
name="imovie.psd" hash="9B2228A7C21BDFC22304E5CD4E4035D8" >
 <chunk n="0" state="0" hash="9B2228A7C21BDFC22304E5WURJCE95D8" >
  <part p0="0" p1="0" p2="0" p3="0" p4="0" p5="0" p6="0" p7="0" p8="0" p9="0" p10="0"
    p11="0" />
 </chunk>
</file>
```

2.1.2 Liste XML de fichiers partagés

Le format XML nous permet également de stocker la liste de fichier partagés par l'utilisateur. L'intérêt est certain : les utilisateurs de Gloster pourront, à terme, quelque soit leur plate-forme, parcourir la liste des fichiers partagés d'une personne sur le réseau. Pour l'instant cette fonctionnalité n'est pas encore implémentée mais elle le sera pour la prochaine soutenance.

Au final, nous avons deux listes de fichiers partagés qui sont complémentaires : une première liste,filedb.xml, qui sert à être envoyée vers les autres utilisateurs et qui contient le minimum d'informations, et une deuxième liste identique à la première en ce qui concerne la structure, filelist.xml, mais qui contient, en plus, les chemins absolus locaux vers les fichiers.

Voici donc un exemple d'un fichier filedb.xml qui sera envoyé aux autres utilisateurs.

```
<!DOCTYPE Gloster-Shared-Files>
<dir name="/" >
 <file size="2391144" name="Aretha Franklin - Respect.mp3"
    hash="EB1114EBC1C0C91259A46EEF0592BEDD" />
</dir>
```

Ici, en revanche, nous avons un fichier filelist.xml, qui, on peut le constater, est plus complet.

```
<!DOCTYPE Gloster-File-List>
<dir name="/" >
 <file size="2391144" p0="727261260E0631F2CA80FF50F0908EC7"
path="/home/z/Download/Aretha Franklin - Respect.mp3" type="2"
name="Aretha Franklin - Respect.mp3" hash="EB1114EBC1C0C91259A46EEF0592BEDD" />
</dir>
```

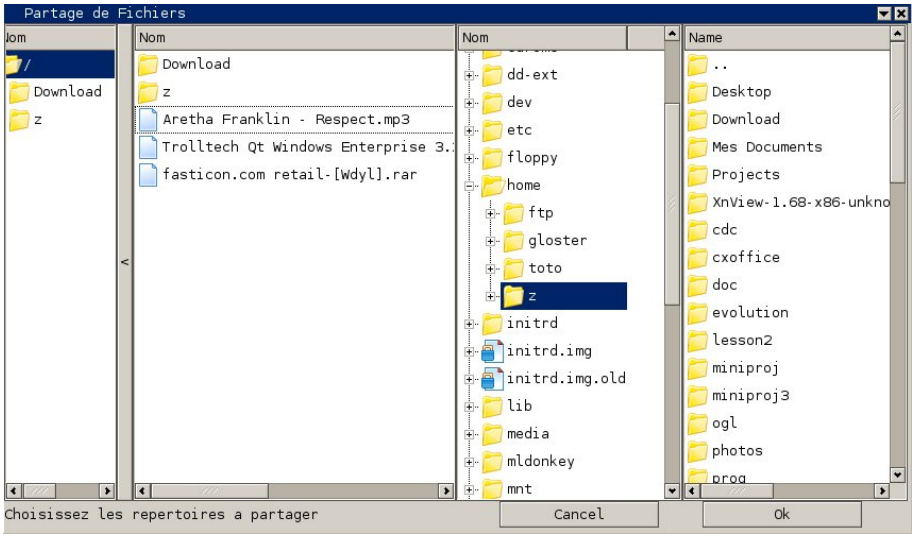
On notera la présence d'attributs supplémentaires à la balise file, notamment "p0" et "path". Les attributs de la forme p¡i¿, correspondent en fait aux hash de chaque chunk i. Le fichier d'exemple étant petit, nous n'avons qu'un chunk reprénsté par l'attribut "p0". La présence des hashes des chunks est importante car le calcul de hash est assez lent sur de gros fichiers. De plus, pour éviter des mauvaises surprises après un tél'echargement d'un gros fichier, nous sommes amenés à faire des tests d'intégrité sur chaque chunk au cours du téléchargement. De ce fait le stockage des hashs dans un xml nous est indispensable.

2.1.3 Selection des fichiers à partager

Pour la sélection de fichier à partager, plusieurs solutions s'offraient à nous : soit nous demandions simplement à l'utilisateur d'entrer le chemin vers les repertoires, soit nous lui faisions sélectionner des repertoires dans une arborescence, soit, enfin, nous lui proposions de choisir les repertoires et les fichiers à partager de manière indépendante et modulable. C'est finalement la dernière solution que nous avons retenue.

En effet, dans Gloster nous voulons aller plus loin que dans un logiciel de Peer to Peer classique, nous voulons proposer en plus la possibilité de choisir fichier par fichier ce que l'on souhaite partager. De plus, comme à terme il sera possible d'explorer les listes de fichiers des autres utilisateurs, nous voulons également proposer la possibilité de modifier l'arborescence des fichiers partagés. Ainsi l'avantage peut-être énorme : si un utilisateur a un disque dur particulièrement désordonné, l'arborescence des repertoires partagés, envoyée aux autres utilisateurs, pourra être largement plus ordonnée et présentable.

Pour faciliter la gestion des fichiers partagés, nous avons dû créer une interface qui permette de les manipuler facilement. En l'occurence, la fenêtre ci-dessous peut se découper en deux parties. La partie gauche correspond à l'arborescence virtuelle (à gauche) et aux fichiers contenus dans le repertoire courant (à droite). La partie droite correspond, quant à elle, à l'arborescence du disque dur local. Toutes les manipulations nécessaires à l'organisation de l'arborescence virtuelle et à l'ajout de fichier à partager, se font par des drag and drop entre les différentes arborescences.



Une fois que l'utilisateur à terminé de choisir les fichiers à partager, il lui suffit alors de valider ses choix par le bouton OK. Se lance alors la génération des deux fichiers XML abordés plus haut : filelist.xml et filedb.xml.

2.1.4 Recherche de fichier

Pour trouver des fichiers sur notre réseau, nous avons besoin d'interroger à chaque fois tous les utilisateurs. Par conséquent, la recherche doit etre performante et ne pas ralentir de manière significative la machine de l'utilisateur. Nous distingons donc deux types de recherches : la recherche par hash et la recherche par nom.

Recherche par hash Lors du chargement de l'application, nous chargeons en mémoire les hash et les chemins absolus des fichiers partagés dans un objet. Ce dernier associe les hash aux chemins absolus et optimise la recherche : La liste de hash est classée.
Ainsi, en très peu de temps, à partir d'un hash, nous pouvons savoir si un fichier appartient à notre liste de fichier partagés.
Ce type de recherche nous sert essentiellement lorsqu'on recherche des sources pour un fichier en cours de téléchargement.

Recherche par nom La recherche par nom est une recherche qui demande largement plus de ressource étant donné que l'on cherche une chaîne au sein des noms de TOUS les fichiers partagés. Le problème est que ce type de recherche sera particulierement utilisé puisqu'il sert à trouver un fichier sur le réseau dans le cas où on ne connait pas le hash.
L'appel à la fonction de recherche dans le thread principal aurait donc pour effet de bloquer complètement l'interface. Pour remédier à ça, nous avons donc dû mettre la fonction de recherche sur un deuxième thread.
Chaque demande de recherche est stockèe dans une file et dès que la file contient au moins un élement, le thread lance la recherche. Aussi, pour ne pas qu'il y ait des conflits d'accès sur la file d'attente entre les deux threads, nous avons utilisé l'objet QT QMutex qui permet de blocker l'accès aux variables utilisées dans le deuxième thread pendant son exécution. L'utilisation de plusieurs threads nous permet donc d'effectuer, à la fois, des tâches nécessaires au fonctionnement globale du réseau et des tâches nécessaires à l'utilisation du logiciel lui-même.

2.2 Reconnaissance des sockets

Nous avons décidé que Gloster n'utiliserait qu'un seul port d'écoute pour les connexions entrantes, ceci afin de simplifier l'utilisation pour des personnes qui utilisent des Firewalls ou bien des Routeurs. Cependant, il y a plusieurs types de sockets : les sockets de gestion du réseau (GSOCKET), les sockets de transfert de fichiers (GDLSOCK) et les sockets de demande de transfert de fichiers utiles quand la connexion dans l'autre sens est impossible (GULSOCK). Le socket d'ecoute, lorsqu'il recoit une connexion, doit donc avant tout determiner a quel type de socket il a affaire afin de le traiter comme il se doit.
Ainsi, a chaque fois qu'un client Gloster crée un socket pour une connexion sortante, dès que celui-ci a reussi à établir la connexion, il envoie un paquet de 8 octets indiquant son type.
Ainsi, lorsqu'une connexion entrante arrive, on attend d'avoir recu au moins 8 octets, puis on lit 8 octets. Ainsi on regarde le type indiqué. Si cela correspond à l'un des 3 types connus, on passe le socket dans le mode correspondant afin qu'il fasse ce pour quoi il est prévu.
Si l'identifiant ne correspond à aucun des 3 types de paquets connus, c'est soit que c'est un client qui utilisait une mauvaise clé de cryptage, soit une connexion provenant d'un logiciel n'étant pas un client Gloster. Dans tous les cas, ce socket n'est pas exploitable et on ferme la connexion.

2.3 Recherche de fichier

Pour rechercher un fichier, un client broadcaste un paquet sur tout le réseau pour indiquer la chaine qu'il recherche.
Tous les clients reçoivent donc ce paquet. Ils recherchent donc chacun de leur coté cette chaine dans leur liste locale de fichiers partagés et construisent une liste de résultats. Quand un client a fini de rechercher et donc de générer la liste de résultats, elle répond en Unicast a la personne ayant généré la recherche.
Ainsi, la personne ayant fait la recherche reçoit les réponses au fur et à mesure et les affiche dans une liste.
Les réponses sont accompagnées de la chaine de recherche, ainsi le client ayant fait la recherche peut classer les réponses des clients sous plusieurs onglets et donc lancer plusieurs recherches simultanément, les réponses pouvant donc être triées par la chaine de recherche et placées dans le bon onglet.

2.4 Transfert de fichier

Le transfert de fichiers est encore au stade experimental. Nous sommes entrain de le debugger, c'est pourquoi seulement une partie de ce dernier sera présenté lors de la soutenance. Cet avancement reste néanmoins conforme au planing du cahier des charges qui prévoyait une ébauche de transfert de fichier pour cette soutenance.

2.4.1 Hash de fichier

Nous avons choisi de proposer du multisourcing dans Gloster car les logiciels alternatifs ne le proposent pas. Or la necessité de ce dernier se fait grandement ressentir sur des réseaux de plus de 10 utilisateurs.
Un hash MD4 (cf. Cryptage) de chaque partie est calculé puis on les concat'ene dans un buffer que l'on hash (toujours en MD4). Ce dernier hash et la taille du fichier servent à identifier le fichier de manière universelle. Nous pouvons ainsi trouver toutes les sources disponibles sur le réseau, indépendamment du nom du fichier.
Pour parvenir à télécharger un fichier depuis plusieurs sources en même temps nous découpons les fichiers en morceaux de taille égale. Ces morceaux sont nommés chunks et nous avons fixé leur taille à environ 9Mo.
Les chunks sont eux mêmes découpés en parts (nous avons fixé la taille des parts a environ 480ko ce qui donne une vingtaine de parts par chunk).

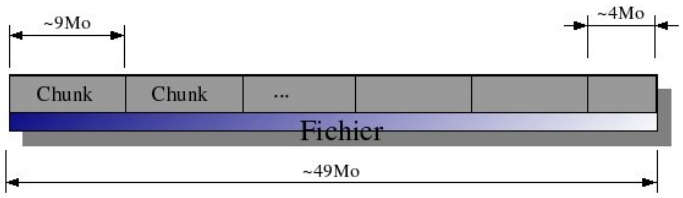


FIG. 1 – Découpe du fichier en chunk

Lors du téléchargement, une recherche de fichier par hash est lancée sur le reseau. Nous demandons á chaque source un part différent. Si un transfert est interrompu, le part en cours de transfert, et uniquement lui, est perdu. Nous avons un algorithme pour choisir le part que nous allons demander dont le but est de completer les chunks.
Quand un chunk est complet, son intégrité est vérifiée à l'aide de son hash MD4 et, s'il est valide, il est partagé sur le réseau, sinon, il est évidement supprimé. Cette technique est très puissante puisqu'elle nous permet, en plus du multisourcing, de partager un fichier pendant son téléchargement. Ceci favorise la diffusion des nouveaux fichiers.

2.4.2 Compression

Pour optimiser les taux de transfert, nous compressons part à l'aide de la zlib. Compresser uniquement par blocs de 480ko nous assure une certaine rapidit'e et une faible occupation mémoire. Nous utilisons la zlib car contrairement à certaines alternatives (LZW, ...), elle nous assure que le buffer compressé à une taille inférieure ou égale au buffer d'écompressé. De plus, elle intègre un contrôle d'intégrité et est librement utilisable.

2.5 Serveur Web

Pour le contrôle à distance de Gloster, un mini serveur Web à été développé, permettant ainsi de surveiller ses téléchargements, de lancer de nouveaux téléchargements depuis une autre machine étant en réseau avec celle executant Gloster et disposant d'un simple navigateur Web.
Nous mettons donc un socket en écoute sur un port spécifique paramètrable par l'utilisateur, répondant aux commandes de base du protocole HTTP 1.1. Le serveur Web répond donc a la commande GET du protocole HTTP qui est envoyée par le navigateur pour faire la demande d'une page. Il est capable de récupérer le chemin du fichier qui lui est demandé via l'URL afin de pouvoir afficher différentes pages. Il gére aussi la commande HEAD qui est généralement souvent utilisée par les Cache des Proxys afin de ne pas avoir à recharger completement une page si celle-ci n'a pas été modifiée.
Le serveur gére plusieurs formats de données et est donc capable de transferer des pages HTML mais aussi des fichiers binaires tel que des images par exemple, ce qui permettra d'égayer quelque peu l'interface Web de Contrôle à distance de Gloster.
N'importe qui ne doit pas pouvoir accéder à cette interface, c'est pourquoi une authentification par mot de passe s'impose. Un mécanisme d'authentification est décrit dans les RFC du protocole HTTP, nous l'avons donc implémenté dans notre serveur Web. Par ce mécanisme, le navigateur doit rajouter un champ spécial dans l'entête de sa requète GET. Ce champ s'appelle "Authorization" et doit être suivi du login concaténé avec le mot de passe tapé par l'utilisateur, le tout séparé par un " :" et réencode en base64. Ce réencodage est une methode permettant d'encoder n'importe quel caractère de la table ASCII complète (donc de 0 à 255) en un groupe de caractéres imprimables, permettant ainsi de faire passer des données binaires au travers d'un protocole texte.

Notre serveur Web est donc capable de récuperer ce champ, de le réencoder en ASCII afin d'en extraire le login et le mot de passe afin de les comparer aux login et mot de passe corrects. Si le navigateur n'a pas fourni de champ Authorization dans sa requète, le serveur lui retourne un code spécial (401 - Authorization Required) lui disant qu'une authentification est nécessaire

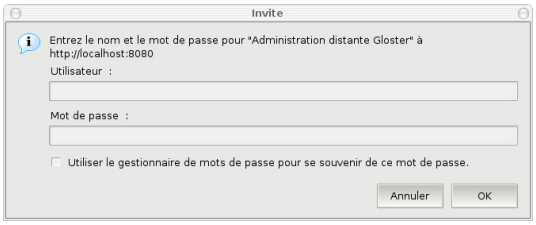


FIG. 2 – Authentification du client

pour accéder à cette page. Ainsi, le navigateur sait qu'il faut interroger l'utilisateur pour avoir un login et un mot de passe et ainsi réiterer la requète en incluant le champ Authorization dans l'entéte, avec les informations que l'utilisateur aura tapées.
De même, si l'authentification échoue, le serveur retourne également ce code, ainsi le navigateur sait qu'il doit redemander à l'utilisateur de s'identifier. Par la suite, c'est le navigateur qui garde en mémoire les paramètres d'authentification et qui va continuer d'inclure le champ Authorization pour toutes les requêtes qu'il fera sur ce même site.
Ainsi ce mécanisme permet de demander à l'utilisateur de s'authentifier lors du premier accès et de conserver une session jusqu'à la fermeture de son navigateur.

3 Cryptage

3.1 Hashage

Le hachage consiste, a partir d'un buffer de données de taille quelconque, de produire un autre buffer de taille constante ne dépendant pas de la taille du buffer d'entrée et dont le contenu dépend directement de celui du buffer d'entrée, tout en perdant une partie des informations de telle sorte qu'il ne soit pas possible de retrouver le buffer d'entrée à partir de son hash. Le hash doit etre calculé de la manière la plus uniforme possible afin d'éviter au maximum les collisions (un même Hash pour deux Buffers d'entrée differents).
Le transfert de fichiers nécessite de générer des Hash de fichiers ou morceaux de fichiers afin de reconnaitre les fichiers notamment pour le multi-sourcing. Pour cela, nous avons donc décidé d'utiliser l'algorithme de Hashage MD4. Celui-ci génère des empreintes sur 128 bits d'un buffer de données binaires quelconques.
Pour conserver la portabilité du projet, nous avons décidé d'utiliser la librairie OpenSSL qui est une librairie Open Source et qui permet notamment de génerer des empreintes MD4. Cette librairie est multiplateforme et assure donc la portabilité de Gloster sur un grand nombre d'architectures.
Il a donc été implémenté une fonction permettant simplement de prendre un buffer de données présent en mémoire ainsi que sa taille et de retourner son Hash. Cependant, cela n'est pas adapté pour générer les empreintes de fichiers complets car ceux ci devraient dans ce cas être intégralement chargés en mémoire avant de pouvoir en faire l'empreinte, ce qui est bien évidemment impossible puisque les fichiers sont susceptibles d'être gros et donc de ne pas tenir en mémoire.
C'est pourquoi, pour le cryptage de fichiers, on utilise un autre jeu de fonctions de la librairie OpenSSL qui permet de calculer une empreinte MD4 "par morceaux". Ainsi on peut fournir les données a hasher en plusieurs morceaux puis dire de finaliser le MD4 aprés les avoir fourni tous les morceaux. Ainsi on charge le fichier par petits blocs pour les fournir à la fonction de hashage.

3.2 Cryptage Symétrique

Une des caracteristiques de Gloster, et pas des moindres, est que tous les échanges doivent être cryptes afin d'assurer la confidentialité des échanges. Un premier niveau de cryptage a donc été implémente pour cette soutenance. Il s'agit du cryptage symétrique. L'algorithme actuellement retenu est le DES. Cet algorithme utilise une clé unique de 64 bits pour le cryptage et le décryptage.
Ici encore, pour ne pas avoir de problèmes de portabilité sur des architectures autres que le PC, l'emploi d'une librairie s'est très vite fait ressentir. Ici encore, la librairie OpenSSL nous a donc été utile puisqu'elle intègre aussi des fonctions de cryptage/décryptage avec l'algorithme DES.
DES crypte les données par bloc de 8 octets, or le réseau s'appuie sur le protocole TCP qui transmet des données sous forme de flux bufferisé. Il faut donc absolument envoyer exclusivement sur le réseau des blocs de 8 octets sans quoi un décalage se produit et les donnés ne peuvent plus être décryptées. Ainsi, lors de l'envoi d'un paquet, on envoie tout d'abord la taille de celui-ci suivi de son contenu. Si sa taille n'est pas un bloc de 8 octets, on le complete pour qu'il le soit en rajoutant des données quelconques qui de toute facon seront ignorées puisque le destinataire connait la taille exacte du paquet.
Le décryptage s'effectue à l'aide de la même clé.
Pour rendre transparent ce processus de cryptage et décryptage, nous avons crée un nouvel objet C++ qui dérive de l'objet Socket que nous utilisions auparavent mais qui redéfinit les méthodes de lecture et d'écriture sur le socket. Ainsi, il a simplement fallu remplacer l'objet Socket anciennement utilisé par le nouveau socket crypté pour que tout soit automatiquement crypté puis décrypté de l'autre coté.
L'objet Socket que nous utilisions auparavent disposaient aussi de méthodes permettant l'envoi et la réception directe de lignes de texte. Ces deux methodes ont également été réimplémentées dans le socket crypté. Ici, il est inutile de spécifier la taille en début de paquet puisque ceux-ci sont délimités par le caractère spécial de retour chariot. Il faut toujours en revanche envoyer des blocs de 8 octets donc quand la taille du paquet n'est pas multiple de 8, on complète le paquet par des retours chariots, generant ainsi des lignes vides qui seront ignorèes à la réception.
La clé de 64 bits est specifiée à la construction de l'objet socket, ainsi il sera possible pour le transfert de fichiers, d'utiliser ces mêmes sockets mais avec une clé différente de celle utilisée pour gèrer le dialogue entre les nodes. Ainsi une clé de session pourra être générée lors de la création d'un socket de transfert de fichiers et échangée entre les deux extrêmités de la liaison de transfert.
La clé de cryptage de 64 bits est générée à partir d'un Hash d'une phrase qui sera la clé réseau et qui est donc facile à retenir et à diffuser et qui pourra

être spécifiée au niveau de l'interface de configuration.
Si une personne tente de se connecter sur un réseau sans avoir la bonne clé, il va alors tenter d'envoyer le paquet lui permettant de se présenter au node à qui il se connecte, cependant comme la clé est mauvaise, l'entête du paquet sera illisible et le node sur qui il se connecte va se rendre compte que cette personne "ne parle pas la bonne langue" et lui fermera la connexion.
DES est un algorithme utilisant une cle de 64 bits et est donc maintenant relativement faible compte tenu de la puissance de calcul des machines actuelles. C'est pourquoi il est prévu de passer au Triple-DES. Il s'agit tout simplement d'une triple passe dans l'algorithme DES avec une clé differente à chaque fois. Ainsi cela revient plus ou moins à un cryptage avec une clé de 192 bits. Le Triple-DES est encore reconnu comme un algorithme de cryptage sûr de nos jours et reste simple à mettre en oeuvre la ou un simple DES est déja en place, c'est pourquoi c'est lui que nous utiliserons par la suite pour le cryptage de données symétrique de Gloster.

Voici un exemple montrant le fonctionnement du cryptage :

Un client envoie le paquet suivant sur le réseau sous forme cryptée :

CMPV 123456789LO21456789012343 fanfwe7890121456789012343 j'aime le Caml!!!

Si on sniffe les paquets transitant sur le réseau, le paquet correspondant à ce message ressemble à cela :

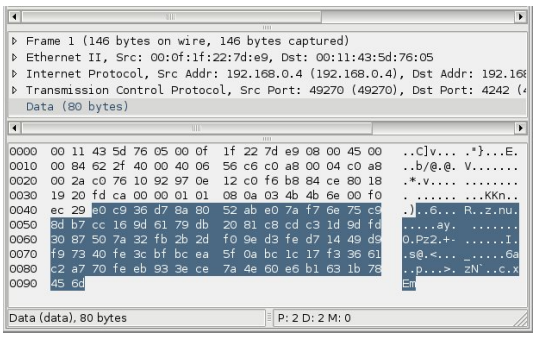


FIG. 3 – Sniff d'un paquet crypté entre deux clients Gloster

On voit donc bien que les données ne passent pas en clair sur le réseau. Le client recoit le paquet et le décrypte correctement :

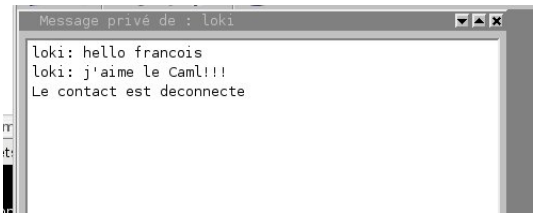


FIG. 4 – Le paquet est bien decrypté

4 GUI

4.1 Première Configuration

Si Gloster est lancé pour la première fois sur un système, une fenêtre de configuration apparait (dont le design sera revu...) ;

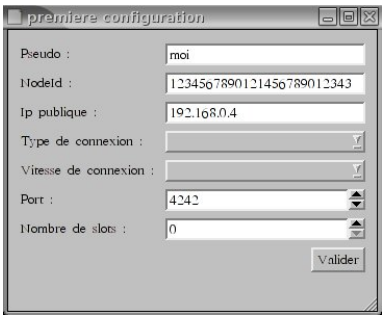


FIG. 5 – Première execution de Gloster

Cette fenêtre permet de saisir les paramètres de base, nécessaires à l'initialisation du reseau (le champ "NodeId" disparaitra par la suite, il est là pour des raisons pratiques lors du developpement). Ceux-ci sont enregistrés pour ne plus avoir à les rentrer de nouveaux mais restent facilement modifiable par la fenêtre "Configuration" du menu Fichier.

4.2 Enregistrement des Paramètres

Les paramètres ainsi saisis (de même que ceux rentrés dans la fenêtres "Configuration" sont enregistrés grâce à des QSettings ; ils ne sont écrits sur le disque qu'une fois detruits.
Exemple :

```
QSettings* settings=new QSettings();
   settings->setPath("Goatsoft","Gloster");
   settings->beginGroup("/Gloster");
   settings->beginGroup("/Config");
   settings->writeEntry("/general/Nick",nick->text());
```

Sous Unix, les paramètres sont stockés dans un fichier glosterrc qui se trouve dans le repertoire .qt dans le home de l'utilisateur ; sous Windows © ils sont stockés dans la base de registre sous la cle HKEY_CURRENT_USER/ Goatsoft/Gloster.
Une fois enregistrés, ces paramètres sont facilement récupérables avec un readEntry :
Exemple :

```
QSettings settings;
   settings.setPath("Goatsoft","Gloster");
   settings.beginGroup("/Gloster");
   settings.beginGroup("/Config");
   w->nickedt->setText(settings.readEntry("/general/Nick"));
```

4.3 Interface

L'interface a été revue et corrigée ; une barre d'outils avec de jolis petits icones pour pouvoir acceder aux differents fenêtres a été ajoutée ; la liste des utilisateurs et celle des chans ont été directement incorporées à la fenêtre principale, dans une QDockwindow retractable.

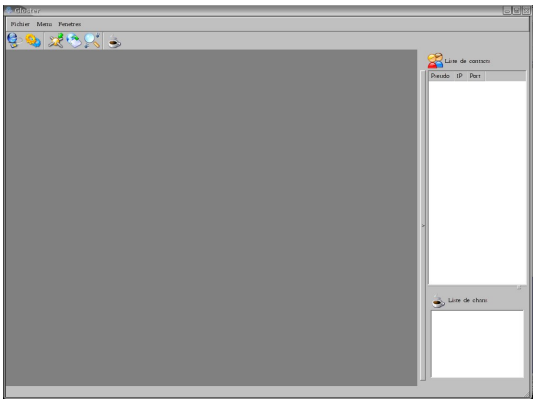


FIG. 6 – Fenêtre principale

Dans la rubrique "Dossiers" de la fenetre "Configuration", le boutton "Partages" lance un explorateur de fichiers, dans lequel il suffit de faire glisser les fichiers ou dossiers à partager dans la partie gauche. Le fait de cliquer sur "OK" lance le hashage des fichiers. Toutes les informations relatives a ces fichiers (nom, taille, hash...) sont stockées dans un .xml (cf section "Reseau").

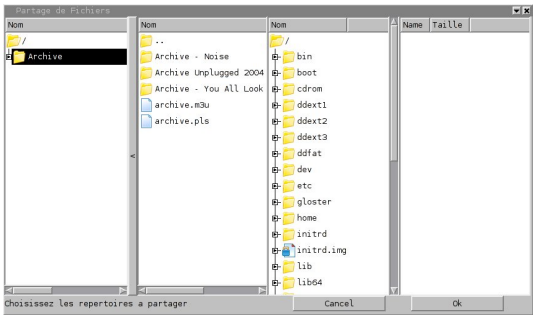


FIG. 7 – L'explorateur permettant de paratger des fichiers

La fenêtre de connexion a été totalement refaite. Le fait de cliquer sur "Connexion" ajoute l'IP et le Port à la liste qui est stockée avec le reste des paramètres, pour ne pas avoir à taper à chaque fois ces informations.

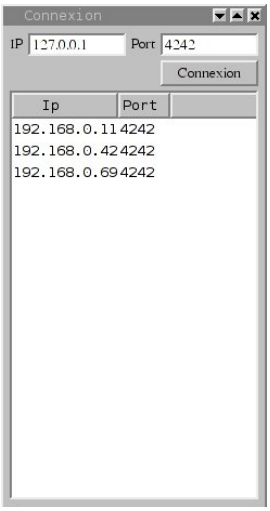


FIG. 8 – Fenêtre de connexion

La fenêtre "Statut" affiche enfin quelque chose : l'état du graphe reseau en 3D et en temps réel!! :-)

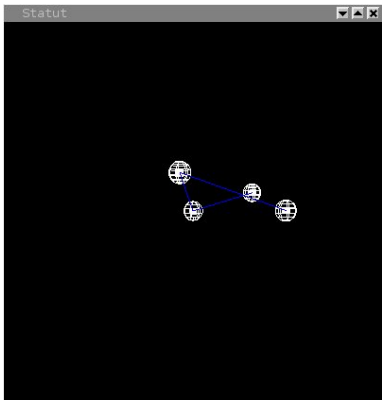


FIG. 9 – L'affichage du graphe du reseau

Il reste 2 fenêtres à revoir : celle de "Recherche" et celle de "Transferts" qui sont totalement hideuses et pas très fonctionnelles!!

5 Travail à accomplir

Pour la dernière soutenance, il nous restera à terminer le transfert de fichiers. Nous aurons aussi à implementer le cryptage assymetrique, et l'échange de clés de session pour les transferts.Pour ce qui est de l'interface, des optimisations y seront apportées (indexation threadée par exemple), et l'interface web realisée. Et puis bien sur, étant donné que ce sera la dernière soutenance, il faudra aussi faire une aide et des packages distribuables. Ainsi, notre projet sera complètement fonctionel.

6 Conclusion

Après cette soutenance, on peut dire que le projet a globalement bien avancé depuis la dernière soutenance. Les fonctionnalités de base du logiciel sont toutes implémentées et quasiment toutes terminées. Il ne reste plus qu'à terminer le transfert de fichier en lui même. Cela semble être une grosse partie, cependant la programmation de cette partie est terminée et il ne reste plus qu'à debugguer. La non présentation du transfert de fichier à cette soutenance s'explique par l'inter-dépendance des différentes parties qui composent le transfert de fichier, ce qui nous a obligé à tout programmer d'un coup pour enfin pouvoir tester.