Goatsoft Corporation présente :

GLOSTER©

Rapport de soutenance finale

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)

Juin 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

Le groupe et ses motivations

Cela va faire la 2eme année que nous travaillons tous ensemble. Nous avons appris à nous connaître et à travailler ensemble. Nous connaissons nos faiblesses et nos points forts ce qui aide énormément pour la répartition des tâches.

Au cours de l'année dernière, nous avons eu l'occasion de développer un jeu, ce qui nous a permis de progresser énormément d'un point de vue programmation. Nous en gardons donc un souvenir assez positif. Cependant, il manquait quelque chose à ce projet : une utilité.
En effet, nous avions un jeu avec un game-play et des graphismes acceptables, mais pas suffisament pour faire jouer nos amis à ce jeu. C'est pourquoi, cette année, nous nous sommes orientés vers un logiciel de partage de fichiers. Ce type de logiciel est particulièrement à la mode en ce moment et le concept auquel nous pensions restait assez marginal par rapport aux logiciels déjà sortis.

Cette année, notre but principal était donc de faire un logiciel utile,utilisable et surtout UTILISE !

Le projet

Gloster est un logiciel d'échange de fichiers paires à paires, autrement appelé peer to peer. Les echanges sont entièrement cryptés grâce à un système de cryptage symétrique et asymétrique. De plus, il est décentralisé, c'est à dire sans serveur. Ainsi, il est possible de pratiquer la causette sans être importuné par des arroseurs ou pire, des fouineurs et ceci même sur un réseau ASFI.[1]

Pour la traduction de cette phrase, consultez l'adresse réticulaire suivante : http ://ensmp.net/cstic/

[1] Phrase offerte par la " Commission Spécialisée de Terminologie et de Néologie de l'Informatique et des Composants Electroniques"

2 Le réseau

2.1 Les graphes

Pour pouvoir diffuser des informations à tous les clients du réseau, ou passer outre des problémes de clients passifs, il fallait avoir une représentation du réseau. Etant donné qu'à partir du moment où une connexion est établie entre deux clients, l'échange de données peut se faire dans les deux sens, la structure qui nous a semblée la plus adaptée était les graphes non orienté.

2.1.1 La structure

Il existe deux représentations pour les graphes : les matrices et les listes d'adjacences.la représentation par matrice est lourde et n'est pas adaptée du tout à nos besoins. Elle utilise une structure statique, obligeant à réallouer les tableaux dès que l'on souhaite ajouter un nouveau sommet. De plus, une grande partie des informations stockées dans la matrice ne sert à rien. Cette méthode reste donc coûteuse en temps et en mémoire.

Notre choix s'est donc porté sur la représentation par liste d'adjacence. Une fois de plus, il nous fallait faire un choix : une représentation entièrement dynamique ou semi-dynamique. Etant donné que nous avions besoin d'accéder rapidement aux informations des différents sommets du graphes, une liste statique de sommet était plus adéquate.

Bien évidemment, après le choix de cette structure, nous avons implémenté les fonctions de base : ajout et suppression.

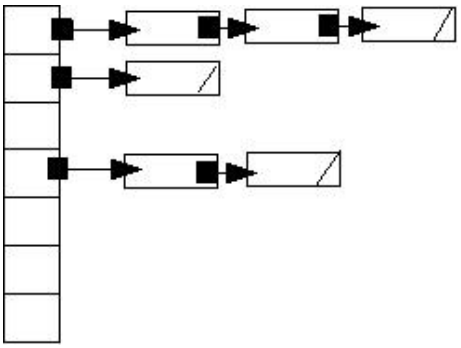


FIG. 1 – Représentation semi-dynamique d'un graphe

2.1.2 Algorithme du plus court chemin

Lorsque nous avions notre graphe, nous nous sommes demandé comment nous allions faire lorsqu'il faudrait envoyer des données entre deux nodes. Etait-il nécessaire de faire une connexion directe si elle n'existait pas, ou y avait-il moyen de faire autrement ? En fait nous avons pensé nous servir du graphe lorsque nous voulions diffuser de petite informations sur le réseau ou pour communiquer avec des clients passifs.En gros, il fallait que les nodes relayent l'information jusqu'à sa destination.

C'est là que l'algorithme de plus court chemin intervient. Afin d'optimiser l'envoi d'informations sur le réseau, nous ne passons pas par n'importe quelle node. En effet, nous avons doublement valué notre graphe. Ainsi chaque node contient une information sur son type de connxion et sur sa vitesse d'émission. Nous pouvons donc choisir de privilégier un bon ping ou une bonne bande passante lorsqu'on recherche le meilleur chemin.

Il existe plusieurs algorithmes de plus court chemin. Les deux plus connus sont les algorithmes de Bellman et de Dijkstra. L'algorithme de Bellman est un peu plus lent que celui de Dikjstra car il teste les distances de toutes ses connexions pour ne retenir que les meilleurs. Il a une complexité, pour une graphe G = <S,A>, de l'ordre de Θ(SA). En revanche l'algorithme de Diksta est un algorithme dit "glouton", c'est à dire que localement il considère que les résultats qu'il obtient sont les meilleurs. Cela permet donc de réduire le nombre de test et donc, selon l'implémentation, de faire mieux que Bellman : il a une complexité d'ordre Θ(Slog(A)). Enfin l'argument de choc, qui nous a permis de nous décider, était le nom des algos. Et oui, "algorithme de Dijkstra" fait plus professionel que "algorithme de Bellman" ; en plus on a le sentiment de faire partie d'une élite lorsqu'on arrive à prononcer son nom.

En bref, nous avons donc retenu l'algorithme de Dijkstra, nous allons maintenant aborder le principe...

Pour expliquer l'algorithme de Dijkstra, nous travaillerons dans un graphe G=<S,A> où S correspond à l'ensemble de sommets et A l'ensemble d'arètes. Nous noterons également |S| et |A| respectivement le nombre de sommets et le nombre d'arètes total. Enfin adj[i] correspond à la liste des nodes en connexion avec avec la node i.

Tout d'abord, pour utiliser cet algorithme, il est nécessaire d'avoir un tableau de taille |S| contenant dans chaque case, correspondant à la node i, son prédecesseur p[i] ainsi que sa distance d[i] par rapport à la node 0. La node 0 correspond au client qui exécute le programme. Au lancement de l'algorithme, tous les d[i] sont initialisés à $+\infty$ et les p[i] à -1 dans une

procédure que nous appellerons init_tab.

Passons maintenant à l'algorithme lui même :

```
1 init_tab()
2 F ← S[G]\{0}
3 tant que F≠ø
4     faire u ← EXTRAIRE-Min(F)
5         pour chaque sommet v ∈ adj[u]
6             faire RELACHER(u,v)
```

A la ligne 1, nous effectuons l'appel à la fonction d'initialisation du tableau contenant les distances et les predecesseurs. Ensuite, ligne 2, nous remplissons une liste contenant tous les sommets sauf le 0, c'est à dire nous. Puis tant que cette liste n'est pas vide, on extrait la node qui a la meilleur distance par rapport à 0. Sur chacune des nodes qui sont dans sa liste d'adjacence, on effectue ensuite un relâchement grâce à la fonction RELACHER().

La fonction RELACHER(u,v) permet de verifier si d[v] est plus ou moins important par rapport à d[u]+W(u,v), où W(u,v) correspond au poids de l'arc (u,v). Si d[v] est supèrieur à d[u]+W(u,v) alors on le remplace par cette valeur. De plus, dans ce cas, cela veut dire qu'il est plus rapide de passer par u pour aller à v que de passer par une autre node. Donc on remplace le prédecesseur de v par u.

```
1 RELACHER(u,v)
2 si d[v] ¿ d[u]+W(u,v)
3     alors d[v] ← d[u]+W(u,v)
4         p[v] ← u
```

Voici un exemple un peu plus concret du relâchement :
Apres l'exécution de l'algorithme de Dijkstra, on obtient un tableau tab[] avec des valeurs de prédecesseurs et de distances pour chaque node. Il est donc aisé d'obtenir le plus court chemin d'une node i : il suffit d'aller à la case tab[i] de notre tableau, puis d'aller sur la case tab[p[i]] et ainsi de suite jusqu'à arriver à un prédécesseur égal à 0.
L'algorithme de Dijkstra étant lancé uniquement lors d'une modification du graphe, on peut dire que la recherche d'un plus court chemin est très rapide.

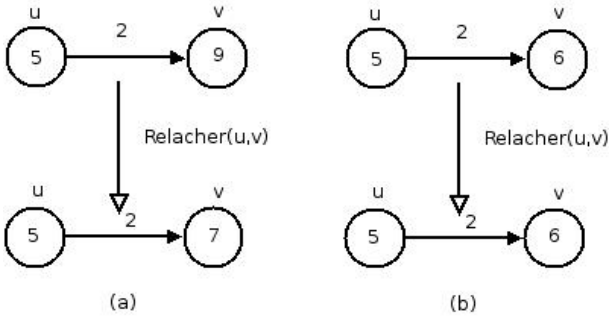


FIG. 2 – L'estimation du plus court chemin de chaque sommet est affiché dans le sommet. **(a)**Comme d[v] > d[u]+2 avant relâchement, la valeur de d[v] décroît. **(b)**Ici, d[v] < d[u]+2 avant le relâchement, donc d[v] ne change pas de valeur.

2.1.3 Les fonctions de broadcast,multicast et unicast

Etant donné que notre réseau est décentralisé, il est nécessaire que nous ayons une repréntation de celui-ci ; c'est pourquoi nous utilisons les graphes. Ainsi lorsque nous voulons envoyer le moindre paquet par les objets réseau, nous sollicitons les algos de plus court chemin.
les fonctions de broadcast, et assimilées, exploitent donc en permanence les algos de plus court chemin. Leur particulirité est qu'elles s'en servent intelligemment. En l'occurence, prenons comme exemple la fonction de broadcast dont voici le principe :

```
1 init_broadcast_list()
2 pour i=0 jusqu'a |S| faire
3     si node[i] existe alors
4         si i¡¿gotonode(i) alors
5             update_bcastlist())
```

L'appel à la fonction init_broadcast_list() à la ligne 1 permet, comme son nom l'indique, d'initialiser les listes de broadcast. En effet, ces dernières sont en réalités des listes de listes. Voici la représentation mémoire d'une liste quelconque :

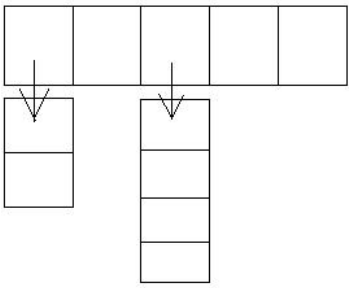
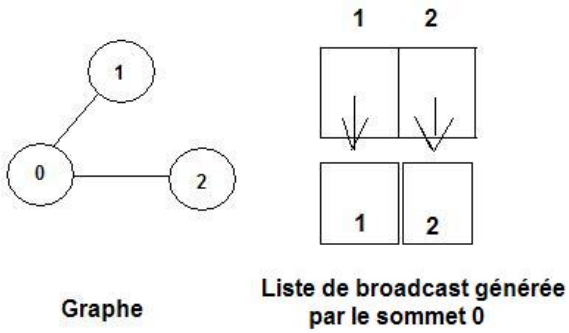
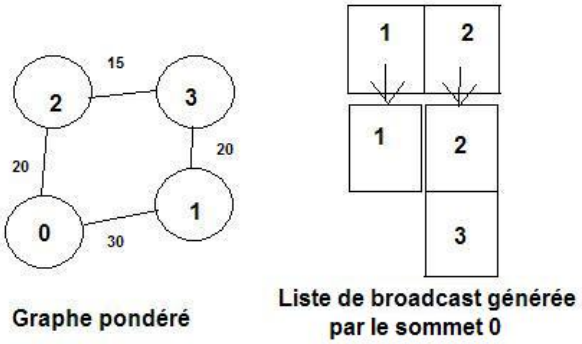


FIG. 3 – Exemple de structure d'une liste de broadcast

Init_broadcast_list crée donc la structure à partir de la liste des arètes du sommet 0, correspondant au client sur lequel est exécuté le programme. Etant donné qu'ici c'est une fonction de broadcast, nous voulons envoyer l'information à tous les sommets du graphe, y compris à ceux qui rerouterons ensuite le paquet à d'autres, j'ai nommé les sommets ayant une aréte avec le sommet 0 (connections directes). Les listes contenues dans la première liste, correspondent aux sommets auxquels chaque connection directe devra rerouter l'information. La fonction init_broadcast_list() ajoute donc également le numéro de chaque connection directe dans chaque liste. Le schéma suivant devrait éclairer un peu plus ces explications.



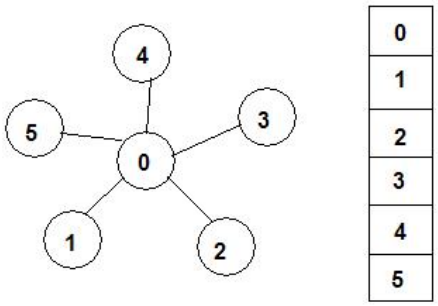
Ensuite nous parcourons l'intégralité de notre vecteur de sommet. A chaque fois si le sommet existe et que ce n'est pas une connexion directe nous l'ajoutons dans notre liste de broadcast. On notera l'utilisation de la fonction gotonode(). Celle-ci est en fait une fonction de plus court chemin qui nous retourne le numero de la connection directe, dans le graphe, qui correspond au premier sommet du meilleur chemin vers la node i ;



En ce qui concerne le multicast, le principe reste le même. Les 2 seuls différences sont lors de l'appel à ini_broadcast_list() : les connections directes ne sont pas ajoutées aux listes d'envois et la construction des listes d'envoi se fait á partir d'une liste de destinataires.

2.1.4 La mise a jour du graphe : les étoiles

Pour pouvoir mettre à jour le graphe sur tout le réseau nous avons dû trouver des moyens efficaces de transmettre les informations de connection entre les sommets. La meilleure solution s'est avérée être un envoi d'étoiles. Concrètement, dès que nous souhaitons transmettre les connections actuelles d'une node sur le réseau, nous envoyons une liste comportant les informations de la node émettrice ainsi que celles de ses connections directes. Nous avons donc codé une fonction qui construit cette étoile.



Il fallait ensuite prendre en compte ces étoiles. Deux cas sont alors à distinguer : celui où le sommet émetteur existe déjà dans le graphe et celui où il n'existe pas.

```
1 si non node_existe(i) alors
2     Ajouter(node i)
3     pour j=1 jusqu'a |S| faire
4         si node_existe(j) alors
5             Ajouter_arete(i,j)
6         sinon
7             Ajouter(node j)
8 sinon
9     pour chaque arete L de i faire
10        si est_dans_etoile(L) alors
11            supprimer_connection(etiquette(L),i)
12    pour j=0 jusqu'a |S| faire
13        si non ajoute(j) faire
14            Ajouter(node j)
```

A chaque fois que l'on récupère une étoile, on vérifie si le sommet émetteur existe déjà. Dans le cas où il n'existe pas, les choses sont assez simples : on crée le nouveau sommet et on traite les connexions directes une par une. Si elles existent on les crée, sinon on se contente de rajouter une arète.
Dans le cas où la node émettrice existe déjà dans le graphe, le traitement est un peu plus compliqué. Il est nécessaire de voir si ses connexions directes sont déjà dans le graphe et si des connexions existaient déjà avant l'arrivée de l'étoile. Enfin on peut se permettre d'ajouter dans le graphe les sommets qui n'ont pas été encore ajoutés.

2.1.5 Suppression de sommets

Jusqu'ici nous gérions déjà la suppression de sommets, cependant il manquait un aspect important. Etant donné que nos graphes ont très peu de chance d'être connexes, dans le cas de la suppression d'un sommet, on pouvait se retrouver rapidement avec plusieurs composantes connexes. De ce fait, nous avions alors plusieurs réseaux qui se voyaientt mais ne pouvaient pas communiquer, ce qui était complètement stupide... (cela arrive même aux meilleurs...)
La solution que nous avons retenue et qui semble la plus optimisée est un parcours profondeur du graphe aprés suppression d'un sommet. Ainsi, au fur et à mesure du parcours nous marquons les sommets visités. Après le parcours, il suffit de vérifier si nous sommes passés sur chaque sommet. Si nous ne sommes pas passés dessus c'est donc qu'il fait parti d'un autre réseau et donc qu'il faut le supprimer du graphe actuel.

2.2 RFC

2.2.1 Introduction

Ce document décrit la base du protocole Gloster qui permet à des machines distantes d'échanger des fichiers, d'en rechercher, d'en partager. Ce protocole est adapté a de petites communauté et permet également de chatter ; il ne necessite aucun serveur dédié. Ce document fournit un ensemble cohérent de méthodes, entêtes, formats de corps de requêtes et réponses qui permettent de faire les opérations suivante :

- En ce qui concerne la connexion : établir une connexion directe avec un node pour pouvoir entrer dans sa constellation ainsi que la mise a jour de la constellation (découverte de nouvelles étoiles).
- En ce qui concerne le chat : Entrer dans un chan, envoyer des messages, en sortir et envoyer des messages privés.
- En ce qui concerne la gestion d'erreur : Notifier quelqu'un qui nous envoie un packet non conforme.
- En ce qui concerne le transfert de fichiers : Rechercher des fichiers, demander des listes de fichiers partagés et établir des connexions binaires.
- En ce qui concerne le cryptage : Echanger des clef.

2.2.2 Terminologie

- node : machine possèdant un client Gloster fonctionnel.
- étoile : liste contenant l'ensemble des connexions directes d'un node.
- constellation : représente l'ensemble d'un groupe de machine connectées grace au protocole Gloster.
- ligne : liste de caracteres imprimables suivie d'un retour a la ligne (\n).
- commande : ligne conforme au protocole Gloster.
- nodeid : identifiant unique (doit être différent chez chaque node) il est composé de 25 caractères imprimables.

– type de connexion : détermine la réactivité de la connexion d'un node. représenté par un entier :
1=ls
2=dsl
3=isdn
4=rtc
5=sat

– vitesse de connexion : détermine la bande-passante de la connexion d'un node disponible pour le tranfert de données. représenté par un entier :
0 : pas de routage
1 : illimite
2 : >1000 Ko/s
3 : 500<x<=1000 Ko/s
4 : 100<x<=500 Ko/s
5 : 50<x<=100 Ko/s
6 : 25<x<=50 Ko/s
7 : 10<x<=25 Ko/s
8 : 5<=10 Ko/s
9 : 1<x<=5 Ko/s

– unicast : envoi d'une commande a une seule node de la constellation.

– multicast : envoi d'une commande a plusieurs nodes de la constellation.

– broadcast : envoie d'une commande a toutes les nodes de la constellation.

2.2.3 Structure d'une Constellation

La structure d'une constellation est complexe et variable car une constellation ne contient pas de serveur dédié. Tous les nodes communiquent pour propager les modifications de la constellation à laquelle elles appartiennent. Le réseau est donc facilement représentable sous forme d'un graphe non orienté.

2.2.4 Généralités sur le protocole

Gloster utilise le protocole TCP pour sa sécurité. Le TCP n'intègre pas de notion de packet, il est buffurisé et sert à transmettre un flux. Nous avons donc opté pour un protocole en mode texte, chaque commande étant sur une nouvelle ligne. Ceci permet également une portabilité plus aisée.

2.2.5 Procédures Gloster

Initialisation de session

Dans l'initialisation, les commandes commencent par une serie de 4 caracteres definissant la type de commande, un espace suivit d'informations d'identification : le pseudo, le nodeid,son type de connexion, sa vitesse de connexion, l'ip publique de la node, son port d'écoute, et un compteur de modifications (informations dans l'ordre et séparées par des espaces). Le compteur de modification sert, sur une grande échelle, à ne prendre en compte que les dernières modifications.

Pour pouvoir entrer dans une constellation, il faut d'abord obtenir l'accord d'une des nodes. Pour se faire, il faut ouvrir une connexion vers un node et lui envoyer un HELO. Si les informations sont correctes, celui-ci répond par un ACPT.

Exemple :

```
A>B : HELO Timmy 12345678901234567890123345 2 123 192.168.0.69 4242 5
A<B : ACPT Luc 12345678901234567890123346 3 51 192.168.0.42 4242 124
```

Utilisation une fois la session ouverte

Routage

Une fois la session ouverte, les commandes commencent toujours par une serie de 4 caracteres permettant d'identifier la commande suivi d'un espace et du nodeid de l'expediteur de la commande suivi d'un autre espace et de la liste des destinataires.
La liste des destinataires est une liste de nodeid séparée par des virgules (',').

Exemple :

STAR A2345678901234567890123 45 12345678901234AS789012345,123456789012345678901234 8
Z A2345678901234567890123 45 192.168.0.1 4242 1 50,Fanfwe 12345678901234AS789012345 192.168.0.42
32817 1 50,loki 123456789012345678901234 8 192.168.0.4 33470 1 50 ;9

La personne recevant ce packet doit se débrouiller pour le renvoyer au travers de ses connexions directes, en direction de 12345678901234AS789012345 et 123456789012345678901234 8. Le packet sera coupé si necessaire.

Mise a jour de la constellation

La constellation peut être soumise à de nombreuses modifications, il est important que tout le monde connaisse ces modifications pour effectuer un routage correct et pour pouvoir communiquer avec toutes les nodes de la constellation. C'est là que le concept d'étoile intervient.
A chaque modification de son étoile, un node broadcast sa nouvelle étoile grace a la commande STAR et incrémente son compteur de modifications. Les nodes qui découvrent de nouvelles étoiles au sein de leur constellation peuvent ainsi découvrir de nouveaux nodes ou être informés de la disparition de certains.

Exemple d'une commande étoile :

STAR A2345678901234567890123 45 12345678901234AS789012345,123456789012345678901234 8
Z A2345678901234567890123 45 192.168.0.1 4242 1 50,Fanfwe 12345678901234AS789012345 192.168.0.42
32817 1 50,loki 123456789012345678901234 8 192.168.0.4 33470 1 50 ;10

Chat

Deux types de commande permettent de gérer les Chans :

- CUEV indique une connexion ou déconnexion d'un node dans un Chan. Il contient le nom du Chan concerné suivi d'un flag indiquant s'il s'agit d'une connexion (1) ou d'une déconnexion (0). S'il s'agit d'une connexion sur un Chan inexistant, on crée le chan.

- CMSG indique un message d'un utilisateur dans un Chan. Il contient le chan concerné suivi du message proprement dit.

Il y a aussi un paquet spécial pour les messages privés :

- CMPV indique un message privé. Il contient uniquement le message étant donné que ce message est envoyé directement d'un node à un autre et ne dépend pas d'un Chan.

Exemple de connexion d'un node au chan #chan :
CUEV fg34567890123456789012345 A23456789012345678901234Z #chan 1

Exemple de message au sein du Chan #chan :
CMSG fg34567890123456789012345 A23456789012345678901234Z #chan Coucou

Exemple de message privé entre deux nodes :
CMPV fg34567890123456789012345 A23456789012345678901234Z Ceci est un PV

Gestion d'erreurs

Deux types de commandes servent à notifier d'une erreur dans un packet reçu :

- UNKN indique un type de packet non reconnu
- WRNG indique un packet reconnu dont un ou plusieurs paramètres sont erronés

Ces commandes peuvent être suivies de l'intégralité du mauvais packet reçu.

Transfert de fichiers

Deux types de recherche sont disponible :

- SRCH permet de rechercher des fichiers par nom (envoyé en broadcast)

```
A>> : SRCH ... 8 toto
A<B : RSRH ... toto tata_toto.avi 654258 AF5848BC7 2;titi_toto.exe 89789 E8CD48 1
A<D : RSRH ... toto tata_toto.avi 654258 AF5848BC7 2;toto.exe 19789 E8HD48 1
```

Pour la commande de reponse on a dans l'odre : id nom fichier taille hash type.

– HSRH permet de rechercher des fichiers par hash (pour le multisourcing)

```
A>> : HSRH ... AF5848BC7 654258
A<B : HRSR ... AF5848BC7 654258 tata_toto.avi
A<C : HRSR ... AF5848BC7 654258 tata_titi.avi
A<D : HRSR ... AF5848BC7 654258 tata_toto.avi
```

Pour la commande de reponse on a dans l'odre : id nom fichier taille hash type.

Il est également possible de faire une demande de liste de fichiers partagés. L'utilisateur nous envoie le hash de sa liste et on compare avec une liste que l'on a éventuellement en local avant de lancer un téléchargement.

```
A>B : FLST ...
A<B : RFLS ... AF5848BC7 654258
```

Dans certaines configuration, il se peut qu'un client soit dans l'incapacité d'ouvrir un port en écoute. Il existe donc une commande pour lui demander de se connecter à nous.

```
A>B : ACON ...
```

Echange de cles

Deux types de paquets permettent les echanges de cles publiques :

– WKEY permet d'effectuer une requète auprès d'un autre utilisateur pour qu'il nous fournisse sa clé publique.

– GKEY permet d'envoyer sa clé publique à un autre utilisateur. Le paquet contient la clé publique encodée en hexadecimal

```
A>> : GKEY ... AF5848BC7FC
```

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.

Le hashage sert aussi à la génération du nodeid. Celui-ci est maintenant généré automatiquement à partir de la clé publique RSA, ainsi ce nodeid est unique puisque les clés de cryptage sont uniques sur un réseau. De plus, le fait de connaitre le nodeid ne révèle pas pour autant la cle publique à tout le monde et ainsi on peut s'arranger pour ne pas la propager à tout le monde

si on ne le désire pas.

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 implémenté : il s'agit du cryptage symétrique. L'algorithme 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.

Un cryptage symetrique DES sur 64 bits est qualifié de relativement faible, cependant pour ce qui concerne les échanges autres que les fichiers eux mêmes, cela n'est pas vraiment grave car ce n'est pas la partie la plus sensible. L'avantage de cet algorithme est que par conséquent, il est relativement rapide et donc n'altère pas la réactivité du logiciel.

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 :

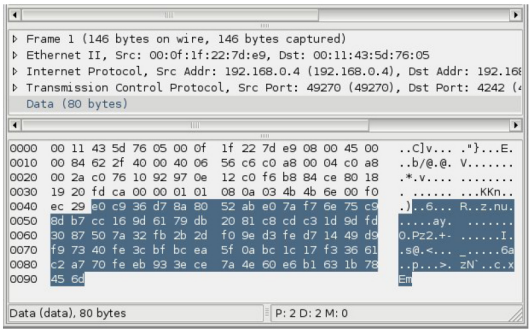


FIG. 4 – Sniff d'un paquet crypté entre deux clients Gloster

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

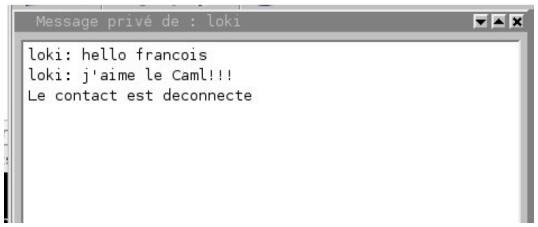


FIG. 5 – Le paquet est bien decrypté

3.3 Cryptage Asymétrique

Le cryptage asymétrique est utilisé pour initier les transferts de fichiers. Il permet d'échanger de manière sécurisée une clé de session DES qui permettra pour tout le reste du transfert de fichiers, de crypter et decrypter avec un algoritme rapide mais tout en conservant un bon niveau de sécurité car pour chaque fichier transféré, une nouvelle clé de session est générée et donc sa durée de vie est tres faible, ce qui réduit de beaucoup la possibilité de décrypter les données de manière frauduleuse.
Pour le cryptage asymétrique, l'algorithme RSA est celui retenu pour Gloster. Nous utilisons des clés de 1024 bits, ce niveau est encore reconnu de nos jours comme sûr.
Chaque utilisateur possède un couple de clés : une clé privée et sa clé publique associée. Quand un utilisateur veut envoyer un paquet crypté par RSA, il utilise la clé publique de la personne vers qui il veut l'envoyer pour crypter le message, quand une personne recoit un paquet crypte par RSA, il utilise sa clé privée pour le décrypter.
Ainsi, pour envoyer une clé de session à une personne, il faut posseder sa clé publique pour pouvoir crypter le message que lui seul pourra déchiffrer avec sa clé privée.
Comme pour le cryptage symétrique, le cryptage asymétrique à été implémenté dans le meme composant qui fait office de socket crypté, ainsi, lors de l'établissement de la connexion, l'échange de clé se fait automatiquement et de manière transparente pour les autres composants de l'application qui utilisent ce socket crypté.

FIG. 6 – Principe du cryptage asymétrique

4 Transfert et partage de fichier

4.1 Multisourcing

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).
Pour ne pas ralentir l'interface, l'écriture dans le fichier et la calcul des hashs des chunks téléchargés se fait sur un thread secondaire.

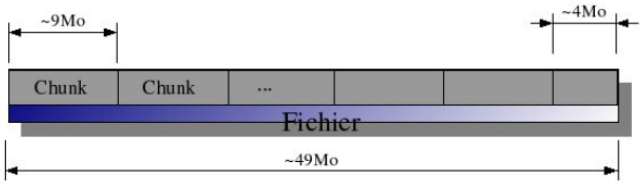


Fig. 7 – Découpe du fichier en chunk

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.

4.2 Protocole

Lorsque la connexion est établie, toutes les communications sont en binaire pour assurer une vitesse maximale.
Le type suivant permet d'identifier les paquets :

```
enum {fdl_getfile,fdl_getpart,ful_hi,ful_havefile,ful_haventfile,ful_ul,ful_new,ful_acceptdl,ful_queue};
```

Le downloader commence par s'identifier avec un ful_hi suivi de son nodeid. Le downloader envoie un fdl_getfile avec le hash et la taille du fichier voulu. L'uploader répond par un ful_havefile (suivi d'un tableau de bit pour specifier quels chunks il possède) s'il possède le fichier dans sa liste de partages ou ful_haventfile s'il ne l'a pas.
Dans l'affirmative, le downloader envoie des fdl_getpart suivi de 2 entiers : le numero de chunk et le numero de part voulu. L'uploader envoie un ful_ul suivi d'un entier definissant la taille du paquet du part demandé puis le part. L'uploader notifie qu'il à un nouveau chunk en partage par un ful_new suivi du numero de chunk. ful_queue sert a indiquer une position dans la file d'attente au downloader.

Il est possible de limiter sa vitesse d'upload. Cette dernière est ensuite répartie intelligement entre les downloaders. C'est à dire que si 20ko/s sont attribués à un socket qui ne monte pas au dessus de 10ko/s (limitations physiques ou autres) un algorithme de retro-controle va baisser la limite a 11ko/s et répartir le surplus aux autres downloaders.
Si la vitesse remonte a 11ko/s, l'algo va ré-équilibrer les vitesses.

4.3 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.

4.4 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'état du téléchargement 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 480Ko. 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>
```

4.5 Liste XML de fichiers partagés

Le format XML nous permet également de stocker la liste de fichiers 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>
<infos nodeid="3aab3e204bf2bd45d2f0408bd" >
<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>
<hashes p0="E3C9729E72497964AC1A25F787E3F66C" hash="3C673C458492C6A027DAC481716BB1AD" >
<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>
```

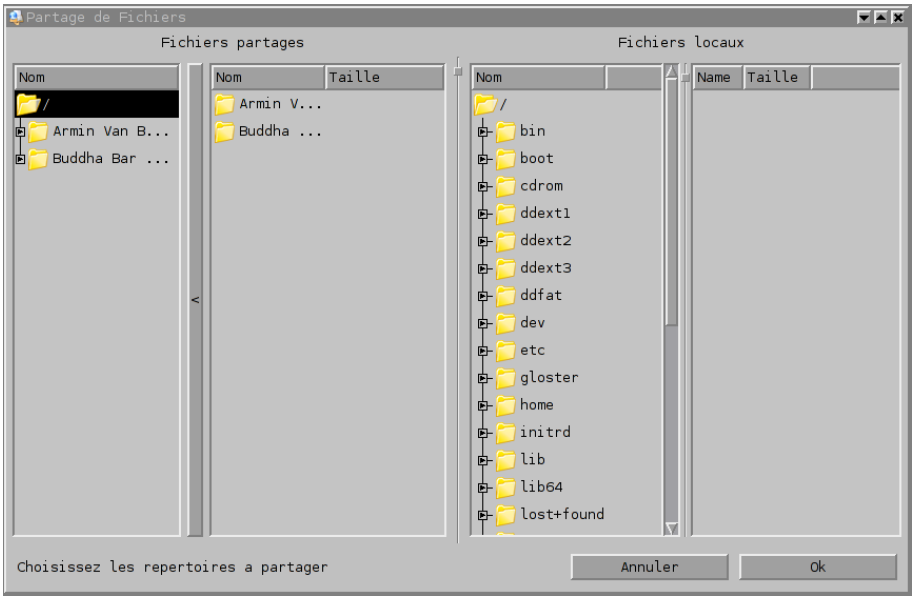
On notera la présence d'attributs supplémentaires à la balise file, notamment "p0" et "path". Les attributs de la forme p¡i¿, correspondent en fait aux hash de chaque chunk i. Le fichier d'exemple étant petit, nous n'avons qu'un chunk reprénsté par l'attribut "p0". La présence des hashes des chunks est importante car le calcul de hash est assez lent sur de gros fichiers. De plus, pour éviter des mauvaises surprises après le 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.
La présence de la balise "hashes" nous permet de vérifier l'integrité de la liste de fichier lorsqu'elle est transférer par le système d'upload de fichier.

4.6 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.

4.7 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.

4.7.1 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.

4.7.2 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.

4.8 Parcours de la liste de fichier d'un utilisateur

En plus de la simple recherche de fichier sur le réseau, comparable à celle de logiciels bien connus comme Edonkey, Gloster tient à rester dans l'optique d'un logiciel communautaire, en proposant de parcourir la liste de fichiers partagés des autres utilisateurs. Il suffit simplement de cliquer droit sur le pseudo de l'utilisateur et de faire "explorer", pour qu'un explorateur s'ouvre. Tout est alors possible : vous pouvez sélectionner plusieurs fichiers et lancer le téléchargement en même temps.

5 GUI

5.1 Première Configuration

Si Gloster est lancé pour la première fois sur un système, une fenêtre de configuration apparait.

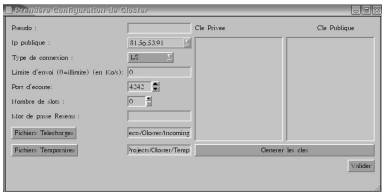


FIG. 8 – Première execution de Gloster

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).Le champ "Ip publique" est rempli automatiquement si la machine est connecté à Internet ; sinon, il doit le compléter à la main. Ces paramètres sont enregistrés pour ne plus avoir à les rentrer de nouveaux mais restent facilement modifiable par la fenêtre "Configuration" du menu Fichier.
Deux champs supplementaires ont été ajoutés : les clés publique et privé (RSA) que l'on peut ajouter manuellement ou generer en cliquant sur "Générer les les clés".

5.2 Vue générale de la GUI

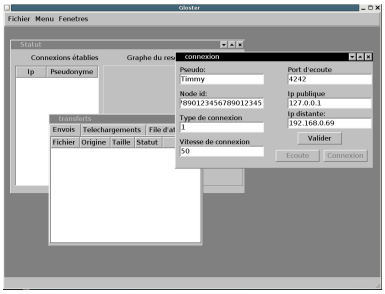
Etant donné qu'un nombre assez important de fenêtres est necéssaire (options, transferts, multiples chan...), une interface SDI devient vite envahissante ! Nous avons donc opté pour du MDI (dont la mise en oeuvre est tout de même mois evidente avec Qt Designer.)

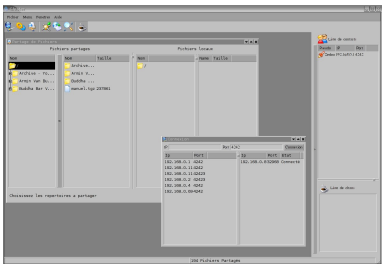
Tout d'abord, il faut creer la fentre principale, (celle qui contiendra toutes les autres fenetres) et lui associer un Qworkspace.

```
MainWin *w=new MainWin();
w->wrk= new QWorkspace(w);
w->wrk->setScrollBarsEnabled( TRUE );
w->setCentralWidget(w->wrk);
```

On crée le Qworkspace associé a la fenêtre principale (w) et on met a sa propriété de barre de défilement à vrai.
Puis, on design les différentes fenêtres grâce à ce formidable outil qu'est Qt Designer.

La GUI avant...
Et la GUI actuelle... :-D





5.3 Enregistrement des paramètres

Tous les paramètres saisis (de même que ceux rentrés dans la fenêtres "Première 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 du 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"));
```

5.4 le Systray

Généralement, un logiciel de peer to peer est très souvent lancé sur une machine et même en permanance pour certain utilisateurs ; c'est pourquoi, il était indispensable d'ajouter un Systray (icone dans la zone de notification).

D'abord, le menu contextuel du Systray :

```
menu.insertItem( QObject::tr("Cacher"),mw, SLOT(hide()) );
    menu.insertSeparator();
    menu.insertItem( QObject::tr("Montrer"),mw, SLOT(showNormal()));
    menu.insertSeparator();
    menu.insertItem(QObject::tr("Quitter"),&a, SLOT(quit()));
```

Et le Systray en lui même :

```
    tray = new TrayIcon(QPixmap(a.applicationDirPath()+"/images/Gloster.png"),
 "Gloster",&menu,0,0);
    QObject::connect(tray,SIGNAL(clicked( const QPoint&,int )),mw,SLOT(hide()));
    tray->show();
```

5.5 Traduction

Pour que Gloster soit utilisé par un plus grand nombre de personne, nous l'avons traduit en Anglais, grâce a QT Linguist : toutes les chaines de caractères doivent êtres "signalées" en les entourant de la commande

```
tr()
```

```
QTranslator translator( 0 );
locale=QTextCodec::locale();
translator.load( QString("gloster_")+locale, 0 );
a.installTranslator( &translator );
```

La ligne locale=QTextCodec : :locale() ;sert a recupérer la "locale" (la langue du système). Si la langue detectée est le français, l'interface sera en francais, dans tous les autres cas elle sera en anglais. La langue de l'interface est modifiable dans l'interface. De plus, grâce au .ts généré, n'importe quel utilisateur peut simplement traduire le logiciel dans sa propre langue.

5.6 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.

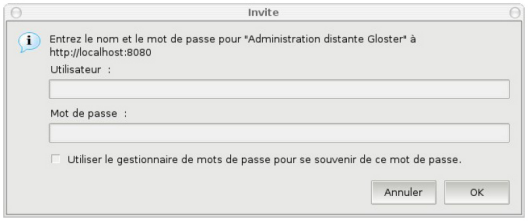


Fig. 9 – Authentification du client

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 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.

Pour le contenu des pages, un fichier de Template à été crée. Il contient du code HTML encadré par des balises spéciales permettant de délimiter des sections de code HTML. De plus, au milieu des pages HTML, des variables de la forme ##VARIABLE## ont été insérés pour définir les éléments dynamiques des pages.

Lors de l'accès à une page, le serveur web va parser le fichier de template pour y extraire les morceaux de code HTML dont il a besoin, puis y remplacer les eventuelles variables par ce qui convient en fonction des données qu'il récupère dans l'application.

Les pages disponibles sont :
- Connexions : Cette page permet de gérer les connexions avec les différents nodes. Il permet d'afficher les différentes connexions établies, ainsi que d'en initier de nouvelles. Cette page est rafraichie automatiquement toutes les 20 secondes.

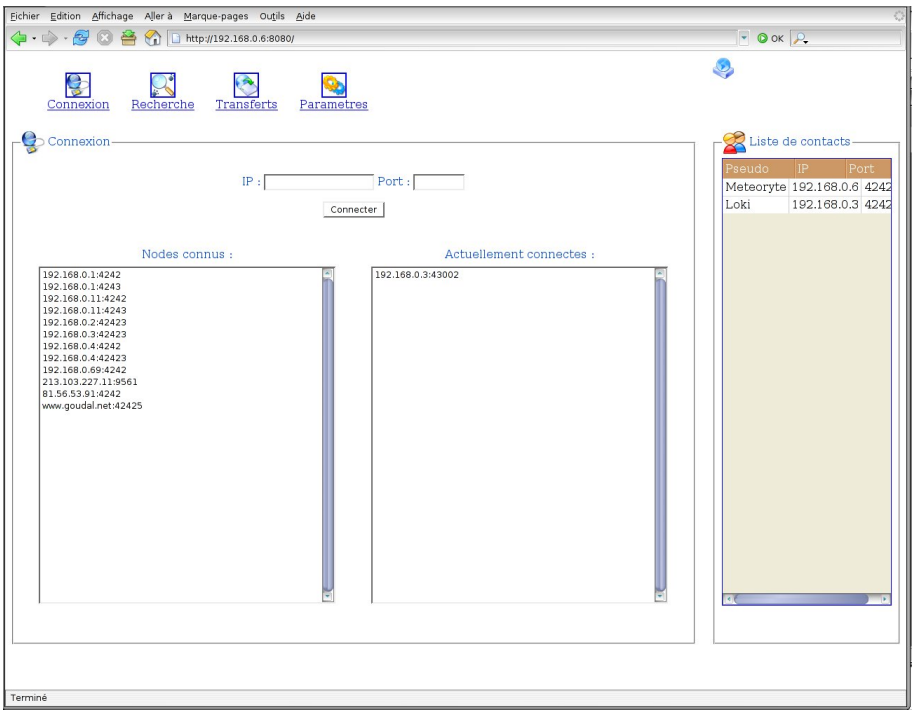


FIG. 10 – Page de connexion

- Recherche : Cette page permet d'effectuer des recherches de fichiers tout comme cela peut se faire dans la GUI. On y met un mot clé et on lance la recherche. Les résultats y arrivent petit à petit grâce au fait que la page se rafraichit automatiquement toutes les 5 secondes. On peut alors cliquer sur un fichier pour débuter son téléchargement.

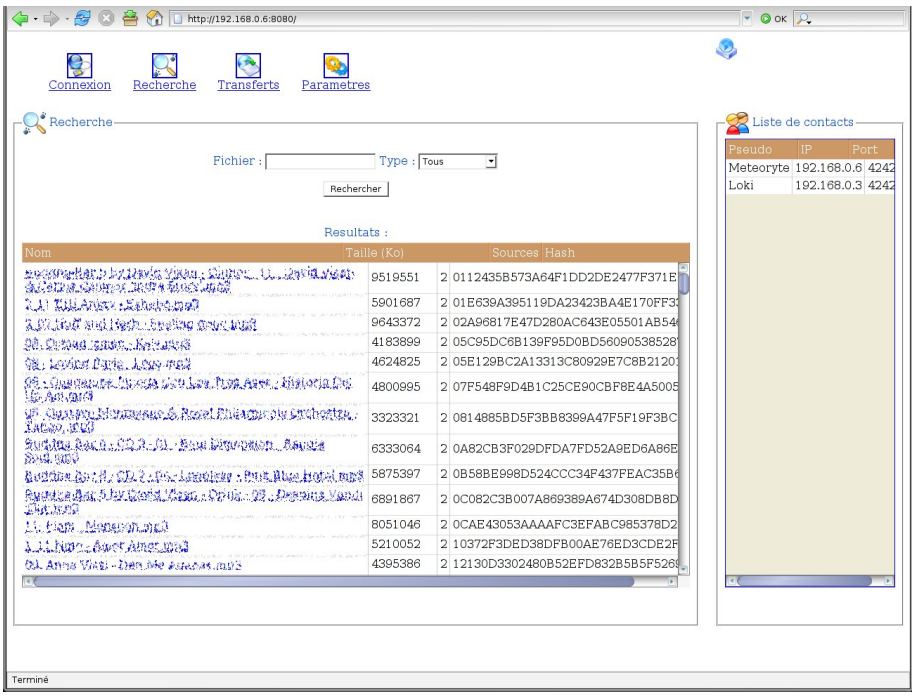


FIG. 11 – Page de recherche

- Transferts : Cette page permet de voir les téléchargements et les envois en cours, ainsi que leur avancement. Elle est elle aussi rafraichie automatiquement pour permettre d'observer l'évolution des transferts sans avoir à recharger manuellement la page.

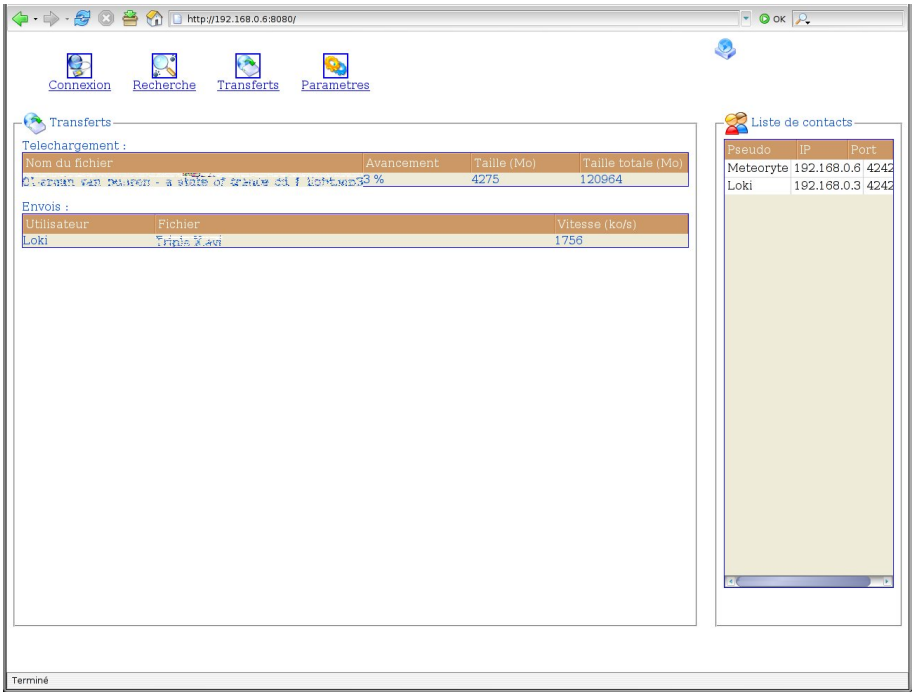


FIG. 12 – Page de transferts

- Parametres : Cette page reprend la majorité des options paramétrables dans la GUI. On peut ainsi modifier des paramètres de Gloster directement depuis l'interface Web.

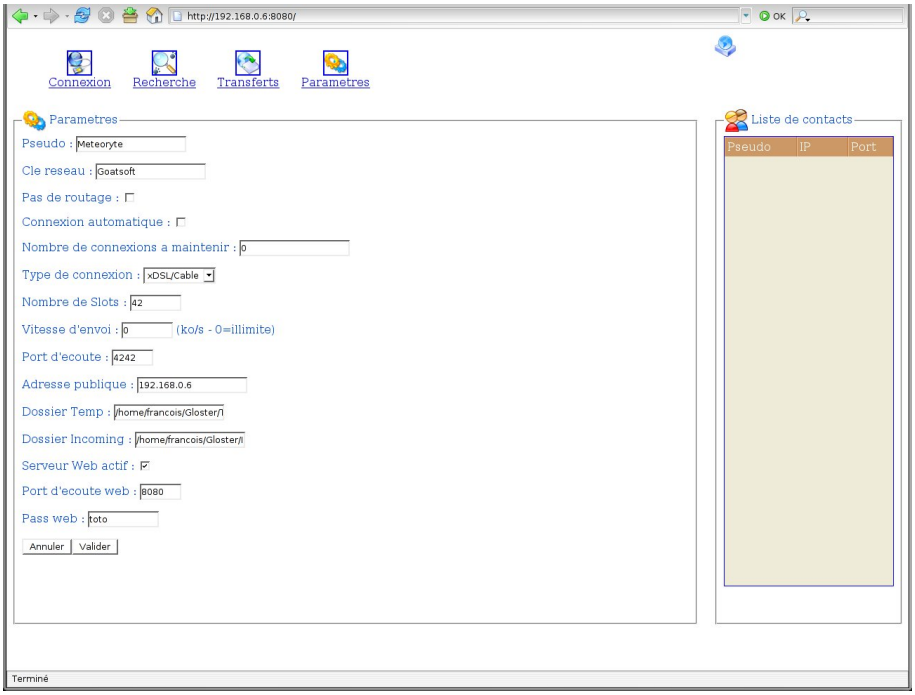


FIG. 13 – Page de paramètres

De plus, en permanence, sur la droite, la liste des contacts est affichée et se rafraichit toutes les 5 secondes, afin de savoir qui parmi nos contacts sur le réseau est présent.

6 Site web

Le site web est hébergé sur Sourceforge, ce qui nous offre la possibilité de faire notre propre site web en parallèle d'une interface réalisée par Sourceforge, disponible pour tous les projets qu'ils hébergent.

Sourceforge offre des services permettant de télécharger les différentes versions du projet, le tout grâce à différents miroirs présents dans de nombreuses parties du globe ce qui offre une très bonne disponibilité des releases du projet.

Il y a également un système de news permettant d'annoncer de manière claire et rapide les nouveautés du projet.
Il y a aussi un forum permettant aux utilisateurs de dialoguer à propos du projet, laisser leurs commentaires, suggestions aux développeurs.
Il y a aussi un système de report de bug permettant aux utilisateurs de signaler très facilement un bug aux développeurs.

Il y a également un accès sous forme HTML à l'arborescence du CVS permettant de télécharger les différentes versions des fichiers et de suivre leur évolution. Le serveur CVS hébergé par Sourceforge nous permet de travailler mutuellement sur le projet sans risque que l'un d'entre nous n'écrase le travail d'un autre. Le CVS est aussi accessible en lecture seule à n'importe qui, ainsi les utilisateurs peuvent directement récupérer la version présente sur le CVS, sans forcément devoir attendre une release, pour disposer des dernières ameliorations.

En ce qui concerne les pages réalisees par nous même pour le site web, elles sont réalisées en PHP et respectent entièrement le standard XHTML.

Le site dispose d'une page d'accueil affichant un bref résume des news les plus récentes, ainsi qu'une description du projet. Il y a une page News qui affiche les news postées sur l'interface de Sourceforge en les récuperant via un système RSS. Une page Downloads avec les liens permettant de télécharger les releases du projet en source et binaire. Enfin, une page Liens donne une liste de sites Web qui nous tiennent à coeur.

7 conclusion

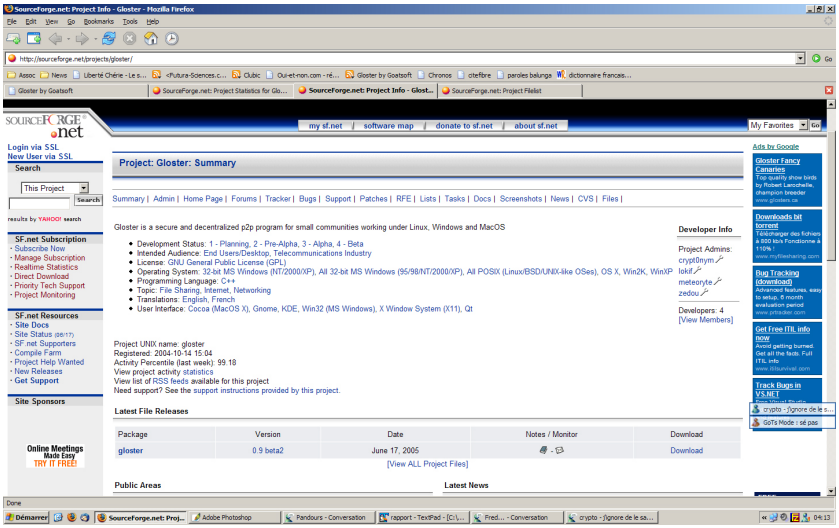
Apres cette année de travail, nous voyons enfin le fruit de nos efforts récompensé. En effet, maintenant que notre logiciel est arrivé dans une phase "stable", nous pouvons exploiter pleinement ses possibilités et déterminer si nos choix était les meilleurs. Bien sûr il y a toujours des choses que l'on voudrait améliorer et il serait malhonnête de dire que nous sommes complètement satisfait de ce que nous avons fait. Cependant, lorsqu'on compare notre logiciel à ceux utiliser tous les jours par des millions d'utilisateurs, nous nous disons qu'au vu du temps que nous avons passé à travailler dessus, le résultat est plus que correct.

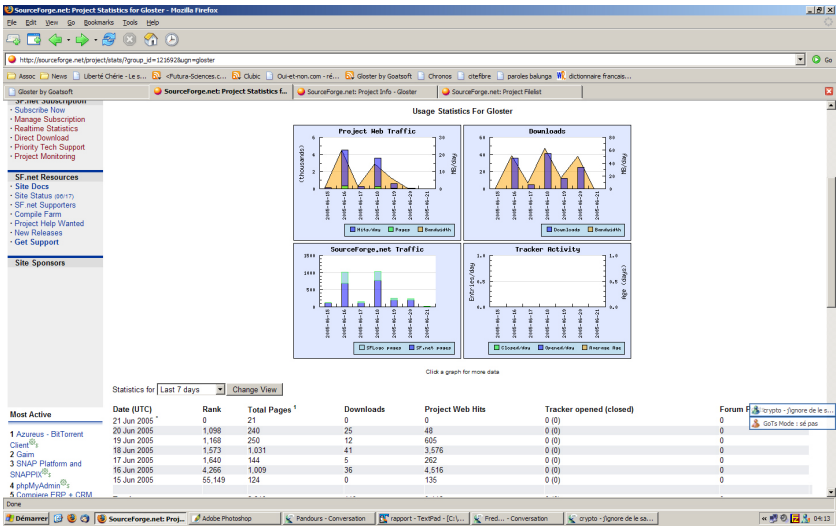
De plus, à l'utilisation notre logiciel reste relativement optimisé : la quantité de mémoire utilisée dépasse rarement les 20Mo et l'utilisation processeur reste raisonnable.

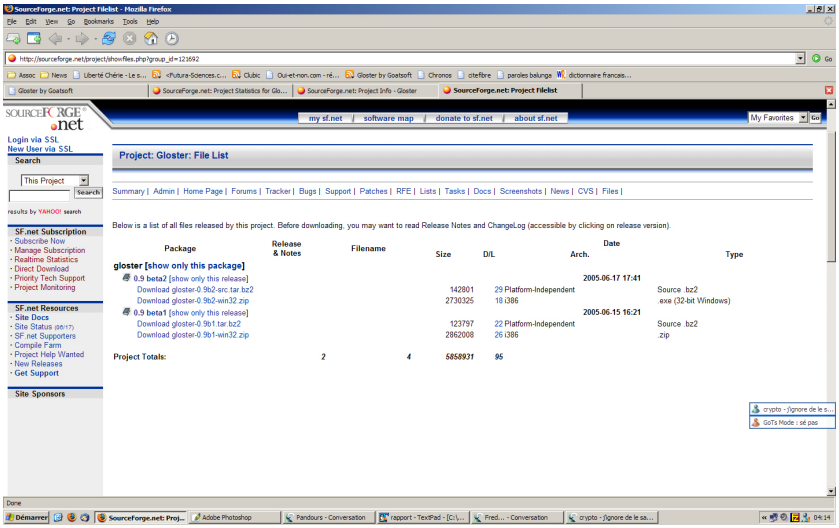
Enfin, grâce à la présence de notre projet sur le portail SourceForge.net, nous avons pu sortir des préversions et trouver des testeurs avant la soutenance. Ces testeurs, de niveaux de connaissance différent, nous ont apporté un regard neuf sur notre travail et nous ont permis de rendre notre interface plus intuitif et plus conviviale. Nous avons d'ailleurs dépassé la centaine de téléchargement en moins d'une semaine !

Nous sommes donc totalement satisfait du résultat obtenu et nous avons atteint notre objectif de départ : faire un logiciel utilisable, utile et utilisé.

Annexes


SourceForge.net: Project Info - Gloster - Mozilla Firefox
Project: Gloster: Summary
Gloster is a secure and decentralized p2p program for small communities working under Linux, Windows and MacOS
Latest File Releases
Package
Version
Date
Notes / Monitor
Download
gloster
0.9 beta2
June 17, 2005


SourceForge.net: Project Statistics for Gloster - Mozilla Firefox
Usage Statistics For Gloster
Project Web Traffic
Downloads
SourceForge.net Traffic
Tracker Activity
Most Active


SourceForge.net: Project Filelist - Mozilla Firefox
Project: Gloster: File List
Summary | Admin | Home Page | Forums | Tracker | Bugs | Support | Patches | RFE | Lists | Tasks | Docs | Screenshots | News | CVS | Files |
Below is a list of all files released by this project. Before downloading, you may want to read Release Notes and ChangeLog (accessible by clicking on release version).
Package
Release & Notes
Filename
Size
D/L
Arch.
Date
Type
gloster [show only this package]
0.9 beta2 [show only this release]
2005-06-17 17:41
Download gloster-0.9b2-src.tar.bz2
142801
29
Platform-Independent
Source .bz2
Download gloster-0.9b2-win32.zip
2730325
18
i386
.exe (32-bit Windows)
0.9 beta1 [show only this release]
2005-06-15 16:21
Download gloster-0.9b1.tar.bz2
123797
22
Platform-Independent
Source .bz2
Download gloster-0.9b1-win32.zip
2862008
26
i386
.zip
Project Totals:
2
4
5858931
95