Goatsoft Corporation présente :

GLOSTER©

Rapport de 1ere 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)

Novembre 2004

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

Gloster est un logiciel de Peer-to-Peer conçu pour des communautés d'une cinquantaine de personnes. Nous avons choisi une architecture décentralisée pour éviter d'avoir à installer des serveurs dédiés. Les communications seront cryptées pour garantir la confidentialité des échanges.

Nous avons choisi de représenter le réseau sous forme d'un graphe, cette structure étant la plus appropriée à notre connaissance. Pour faciliter le portage nous utilisons les librairies Qt.

Notre planning nous semble réalisable et, à ce jour, nous n'avons aucune modification à y apporter.

2 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 donnés peut se faire dans les deux sens, la structure qui nous a semblée la plus adaptée était les graphes non orienté.

2.1 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.

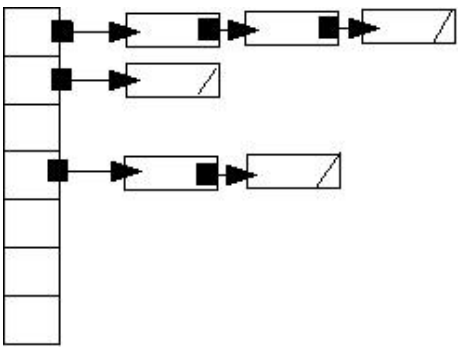

FIG. 1 – Représentation semi-dynamique d'un graphe

2.2 Algorithme du plus court chemin

Lorsque nous avions notre graphe, nous nous sommes demandé comment nous allions faire lorsqu'il faudait 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'algo de plus cours chemin intervient. Afin d'optimiser l'envoi d'information 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 Dikstra 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 totals. 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 :

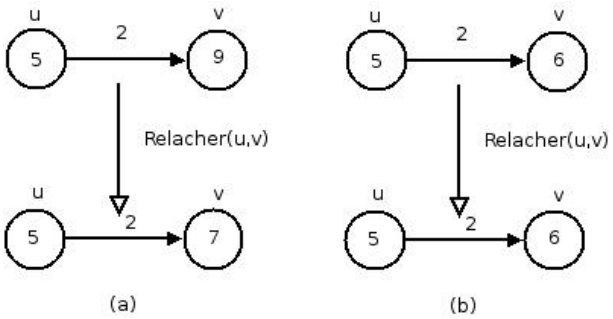

FIG. 2 – L'estimation du plus court chemin de chaque sommet est affichée dans le sommet. **(a)**Comme d[v] > d[u]+2 avant relâchement, la valeur de d[v] décroît. **(b)**Ici, d[v] < d[u]+2 avant le relâchement, donc d[v] ne change pas de valeur.

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'algo 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.

2.3 Les cas qui restent à gérer

Bien que nous ayons déjà une implémentation assez complète des graphes, il nous reste encore quelques fonctions essentielles qui serviront pour les envois d'informations grâce à la couche réseau. Notamment :

- **Une fonction de broadcast** : elle utilisera massivement la fonction de calcul de plus court chemin. Elle permettera d'établir des listes d'envoi pour les connexions directes à une nodes qui souhaite émettre des informations en broadcast.

- **Une fonction de multicast** : elle utilisera elle aussi massivement la fonction de calcul de plus court chemin. Elle permettera d'établir des listes d'envoi pour émettre une même information à plusieurs nodes dans le graphe.

- **Une fonction qui met à jour l'état d'une node** : lors de modificatoin du réseau, ou lors d'une nouvelle connexion, le client recevra des listes de chaque nodes du graphe. Ces listes contiennent les ids des nodes avec lesquelles la node émettrice est en connexion directe. Cette fonction permettera donc de mettre à jour les informations du graphe.

- **Une fonction de parcours en profondeur** : dans le cas où une node qui se déconnecterait couperait la communication entre deux groupes de nodes, il sera nécessaire de mettre à jour le graph en conséquence, en supprimant les nodes qui ne sont plus accessibles. Ainsi à chaque suppression, nous lancerons cette fonction qui nous renverra la liste de nodes à supprimer de notre graphe.

3 Qt

Nous avons choisi de développer notre projet en C++ pour bénéficier de la programmation objet. Pour ne pas "réinventer la roue" nous utilisons les librairies Qt.
Qt propose différents objets pour les sockets, chaînes de caractères, fichiers et beaucoup d'autre utilisables de manière indépendante de la plateforme. Nous souhaitons porter notre projet sous linux et windows, Qt facilitera énormément ce portage.
Qt Designer permet de créer facilement une interface en plaçant les widgets graphiquement au lieu de spécifier les coordonnées dans le code. Les interfaces crées sont compilables sous windows, linux et mac.
Qt permet d'étendre les possibilités du C++ avec les méta-objets. Un système de signaux et de slots permet l'interaction entre les objets. Sans Qt l'interaction est possible avec des pointeurs de fonction mais la manipulation de ceux-ci est beaucoup plus complexe. Il est possible de connecter des signaux avec des slots ainsi que des signaux entre eux et ce sans limitation du nombre de connexion par slot ou signal. Ce système prend 4 fois plus de ressources que les traditionnels pointeurs de fonction ce qui, pour nous, reste tout a fait acceptable.

4 Reseau

4.1 Architecture du Reseau

Gloster est un programme permettant de créer des reseaux d'échange de fichiers décentralisés, c'est à dire sans serveur. Il s'agit donc de nodes qui sont connectes entre-eux, sans pour autant que tout le monde ait une connexion avec tous les autres. Ainsi des données sont susceptibles de pouvoir circuler de nimporte quel node vers nimporte quel autre node, en passant, eventuellement, par d'autres nodes intermédiaires s'il n'existe pas de connexion directe entre ces deux nodes.
Pour cela, chaque paquet circulant sur le reseau contiendra dans son entête le destinataire du paquet, ainsi, si un node recoit un paquet dont il n'est pas le destinataire, il devra alors le faire suivre en le réexpediant de telle sorte que celui-ci se rapproche de sa destination finale.
Le réseau ainsi constitué peut être assimilé à un graphe non-orienté dont les sommets correspondent aux nodes et les arrêtes correspondent aux connexions qui sont établies entre les nodes.

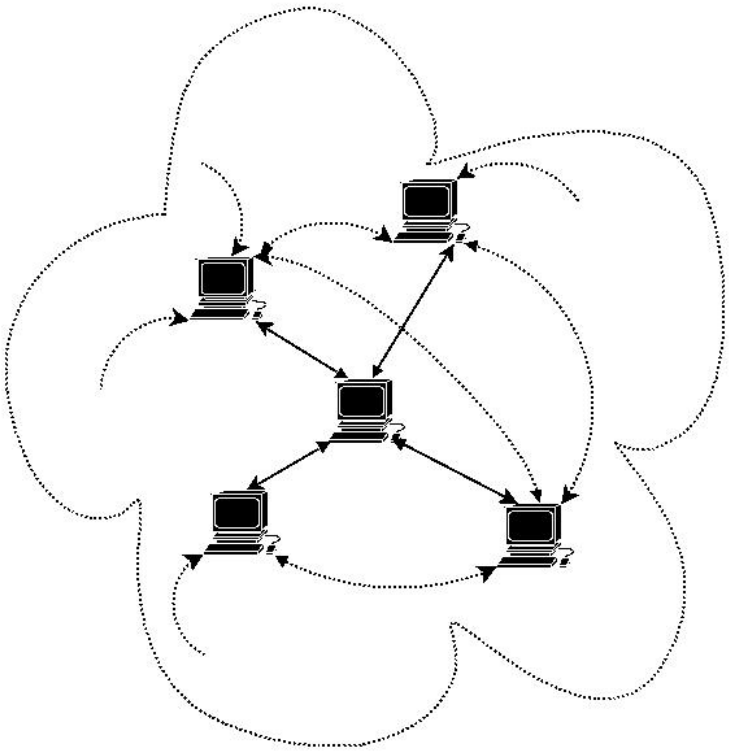

FIG. 3 – Structure du reseau

Chaque node tient donc à jour une représentation complète du graphe du réseau afin qu'il puisse déterminer le chemin à emprunter pour envoyer des données à un autre node. Pour cela, il utilisera un algorithme de plus court chemin qui lui dira quel est le node, parmi ceux auquel il est directement connecté, auquel il doit renvoyer le paquet. Le node suivant, s'il n'est pas non plus le destinataire final, recommencera le calcul de plus court chemin car entre temps, la structure du graphe peut éventuellement avoir changé

Pour rendre les choses plus fiables, le graphe est étiqueté afin de pondérer les connexions selon leur vitesse, afin de privilegier le chemin qui sera le plus rapide, sans pour autant être le plus court.
Certains paquets peuvent avoir plusieurs destinataires, comme par exemple pour les paquets de Chat. Il s'agit alors de Multicast. Ainsi, l'entête d'un paquet ne contient pas en realite le destinataire du paquet mais plutôt une liste de destinataires précedée du nombre de destinataires. Ainsi, lorsqu'un paquet ne contient qu'un seul destinataire, c'est un paquet normal, sinon c'est du multicast ou du broadcast.

Pour le cas de paquets devant être diffusés à tous les nodes (broadcast), on utilise le même principe que le multicast, en mettant dans l'entête du paquet, la liste de tous les nodes présents sur le réseau.
Les entêtes des paquets contiennent enfin l'expediteur du paquet car cette information est utile dans presque tous les cas.

Dans l'entête d'un paquet, un node est representé par un identifiant qui sera composé de son adresse IP, de son port Client, et de son adresse MAC. En effet, si deux nodes qui sont chacun sur un reseau local indépendant et qui sont reliés via Internet, et que ces deux nodes ont, sur leur réseau local respectif, la même adresse IP et utilisent le même port, ils auraient le meme ID alors que ce sont pourtant bien 2 nodes différents ce qui poserait problême, ainsi on rajoute leur adresse MAC afin de les identifier correctement.

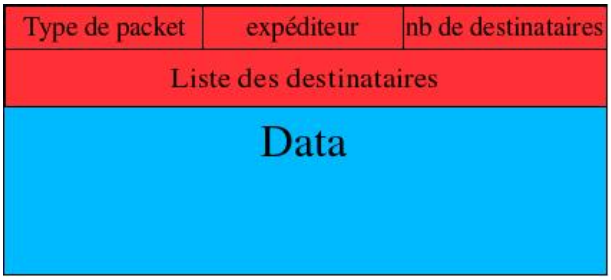

FIG. 4 – Structure générale d'un paquet

4.2 Types de packets

Structure Génerale d'un paquet

Tout paquet est constitué d'une partie entête et d'une partie données. L'entête à la même structure pour tous les paquets alors que la partie données est spécifique à chaque type de paquet.
L'entête est tout d'abord constituée d'un octet permettant de déterminer le type de paquet.

Ceci est immédiatement suivi par l'identifiant du node qui a émis le paquet codé sur 12 octets (Adresse IP : 4 Octets + Port Client : 2 Octets + Adresse MAC : 6 Octets).
Vient ensuite un Unsigned Short Int (2 Octets) donnant le nombre de destinataires du paquet.
Enfin, une liste de destinataires de taille n*12 (n étant le nombre de destinataires) constitue des identifiants consécutifs de chaque node devant recevoir le paquet.
Viennent ensuite les données dont la structure dépend donc directement du type de paquet.

Gestion des Connexions/Deconnexions

Pour que tout le monde puisse à tout moment maintenir son graphe du réseau, il faut qu'à chaque fois qu'un node se connecte a un réseau, tout le monde soit mis au courant, sans oublier que ce node n'est pas forcément isolé, c'est à dire qu'il pouvait lui mème être dans un autre réseau qui était isolé du réseau auquel il vient de se connecter. Autrement dit, cette connexion relierait deux réseaux qui étaient auparavent indépendants. Il faut donc que chaque membre des réseaux initiaux soient prévenus de l'arrivée dans le réseau de tous les membres de l'autre réseau. Pour cela, lorsq'un node a initié une connexion, il envoie au node, lorsque cette connexion est établie, un paquet contenant la liste des autres nodes auquels il est directement connecté.

Le node sur lequel on vient de se connecter recoit donc une liste d'ID qu'il ne connait pas. Il va donc les ajouter a son graphe et lancer en broadcast un paquet contenant sa propre liste de connexions directes.

Ainsi, a chaque fois qu'un node recoit un paquet contenant une liste de nodes contenant au moins une node qu'il ne connait pas, il la rajoute a son graphe et proadcaste a son tour sa liste de connexions directes.
Ainsi, on est certain que l'arrivée de nouveaux nodes dans un réseau soit correctement propagée à tous les autres nodes, même dans le cas de regrouppement entre deux rêseaux isolés.

Pour les deconnexions, le principe est exactement le même : Quand une connexion est supprimée, les deux nodes qui étaient reliés par cette connexion broadcastent leur nouvelle liste de connexions directes afin que tous les clients puissent mettre à jour leur graphe.

Il y à donc un seul type de paquet permettant de gerer les Connexions et les Déconnexions. Les données transmises dans ce paquet sont constituées tout d'abord d'un Unsigned Short Int (2 Octets) donnant le nombre de nodes qui vont être listés. Vient alors une liste de nodes accompagnées d'informations les concernant. Pour chaque node, on a les informations suivantes :

- 16 Octets réservés au stockage du pseudo.
- 12 Octets pour le stockage de l'identifiant du node.
- 15 Octets pour le stockage de l'adresse ip du node sous forme chaine de caractère.
- 2 Octets pour stocker le port d'écoute du node.
- 1 Octet pour stocker le type de connexion dont dispose le node.
- 1 Octet pour stocker la bande passante de la connexion du node.

Le premier node de la liste correspond au node ayant émis le paquet et les autres nodes sont ceux étant directement connectés a lui.

Gestion du Chat

Il existe 3 types de paquets pour la gestion du Chat : Les paquets contenant les évènements propres au chat (Arrivée ou départ d'un utilisateur dans un Chan, Création ou Suppression de Chan).
Les paquets contenant les messages des chans.
Les paquets contenant les messages privés.

Les données transmises pour le premier type de paquet sont constituées de deux chaines de caractère terminées par Zero. La première contient le nom du Chan, la deuxième contient une chaine correspondant au type d'évènement, ce qui permet de pouvoir éventuellement rajouter des types d'évènements plus tard.

Pour le deuxième type de paquet, il s'agit d'un multicast fait à tous les participants du Chan. Le paquet contient donc deux chaines de caractère

terminées par Zero. La première contient le nom du Chan auquel appartient le message et la deuxième contient le message proprement dit.

Le troisième type de paquet, quand à lui, est plus simple puisqu'il ne contient qu'une simple chaine de caractère terminée par Zero et qui contient le message privé.

4.3 Implementation

Nous avons bien avancé l'implémentation de cette architecture réseau. Celle-ci dépend de l'implémentation des graphes et ne pourra être fonctionnelle qu'une fois cette dernière achevée.
Nous avons créé plusieurs objets hérités d'objets Qt bénéficiant ainsi du système de slots et signals.

L'objet central est la class Gnetwork. Très simple d'utilisation, celle-ci donne accès à toutes les fonctionnalités du réseau :

- Connexion a un node a partir de son adresse IP
- Connexion a un node a partir de son identifiant unique
- Déconnexion d'un node
- Ecouter sur un port pour attendre des connexions entrantes
- Rejoindre/Quitter un salon de chat, y envoyer un message ou envoyer un message privé
- Demander/Rechercher/Envoyer un fichier
- Récupérer la liste des fichiers partages d'un utilisateur
- ...

Les fonctions de plus bas niveau permettant d'envoyer directement des packets sur le reseau sont accessibles aux class héritant de Gnetwork pour permettre à d'autres développeur de se resservir facilement de cette dernière. Le Gnetwork émet des signaux pour chaque action dans le réseau (réception de messages, nouveau node dans le réseau,...).

Si on demande l'écoute sur un port, le Gnetwork se chargera de créer un GlistenSocket, class héritant du QserverSocket de Qt. Le GlistenSocket accepte automatiquement les connexions entrantes et créé un Gsocket, socket cliente héritant du Qsocket.

Un Gsocket est également créé quand on établit une connexion à un node. La class Gsocket se charge de filtrer les packets entrants tout en vérifiant leur validité. Un message différent est émis pour chaque type de packet. Ce message est récupéré par le Gnetwork qui le traite.

5 Gui

Pour la GUI de notre projet, nous avons avions plusieurs possibilités : MDI ou SDI (ou encore un système a base d'onglets, mais celui ci ne nous semblait pas vraiment adapté).

5.1 MDI vs SDI

Une interface de type SDI est composé de multiples fenêtres indépendantes les unes des autres alors que pour une ineterface de type MDI les differentes fenêtres sont regroupées au sein d'une fenêtre principale.

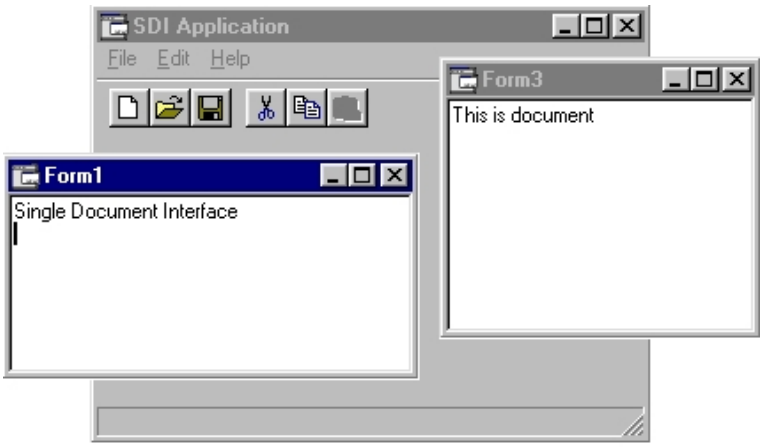

FIG. 5 – Exemple de SDI

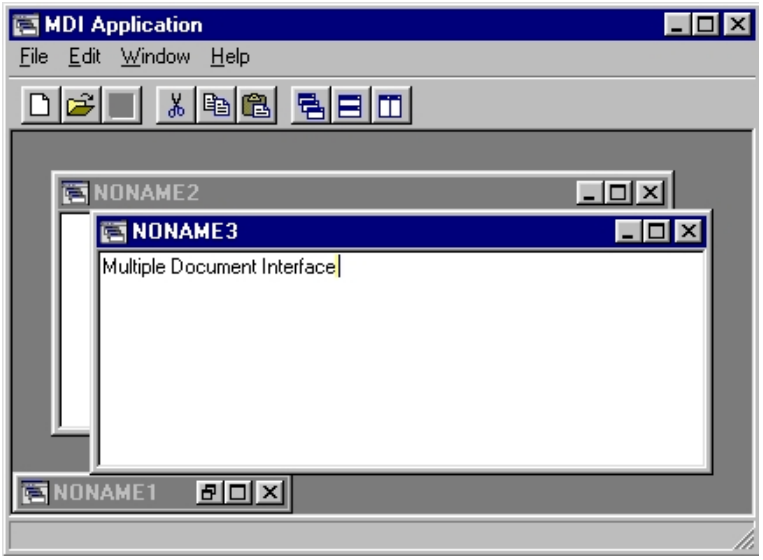

FIG. 6 – Exemple de MDI

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.)

5.2 Création des fenêtres "on the fly" [1]

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 a ce formidable outil qu'est Qt Designer (Ces fenêtres sont de type QmainWindow et non Qdialog...)

Pour chaque item devant donner accès a une nouvelle fenêtre, il faut creer ladite fenêtre :

```
chanCreate *w=new chanCreate (wrk,0,WDestructiveClose);
```

Pour une question d'ésthétique,nous supprimons la barre de statut :

```
w->statusBar()->~QStatusBar();
```

Et enfin, la fenêtre est affichée :

```
w->show();
```

Voici à quoi, pour l'instant, ressemble l'interface de Gloster (avec quelques fenêtres ouvertes) :

[1] à la volée

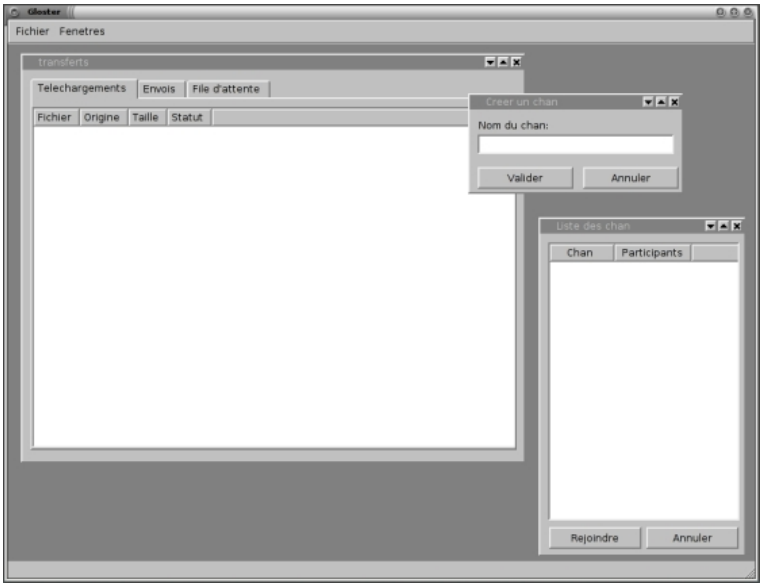

FIG. 7 – Rendu de la GUI sous linux...

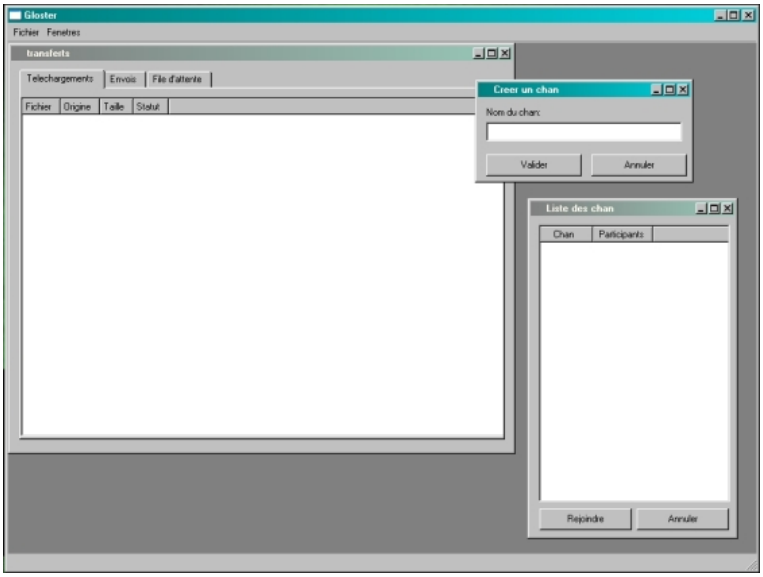

FIG. 8 – ...et sous Windows

6 Travail à accomplir

La principale tâche pour la prochaine soutenance sera de fusionner la partie Graphe et la partie reseau puis lier le tout a la Gui. De plus il reste quelques améliorations à apporter a la Gui. Il faudra aussi commencer le developpement du site web.

7 conclusion

L'avancement du projet se fait conformément au planning établi dans le cahier des charges. Les élements nécessaires pour le moteur du projet devraient être terminés pour la prochaine soutenance et intégrés ce qui devrait permettre de commencer à tester avec un vrai réseau.

Les bases du projet ont éte posées pour cette soutenance, la partie réflexion concernant le protocole réseau a été faite et les methodes choisies ont été suffisemment réfléchies pour pouvoir être fiables, il ne reste plus qu'à les implementer mais le debut du commencement à bien été entamé. L'interface se construit selon le modèle choisi, à savoir le MDI. Le travail de réflexion est donc terminé et notre travail d'ici la prochaine soutenance sera principalement porté sur le code et l'implémentation des principes qui ont été choisis.