Goatsoft Corporation présente :

GLOSTER©

Rapport de 2eme 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

Pour la dernière soutenance nous avions commencé à implémenter les graphes et à développer l'interface. Nous avions également définit le protocol réseau de notre logiciel. Cependant, il nous fallait encore développer nos composants rséaux.
Pour cette soutenance, c'est chose faite. Nous sommes dorénavant capables de connecter 2 clients.
Pour préparer au mieux la soutenance suivante, nous avons également commencé l'intégration. En d'autre terme nous sommes parfaitement dans les temps et nous remplissons nos objectifs définis dans le cahier des charges.

2 Les graphes

Pour cette soutenance, nous disposions déjà des fonctionnalités de base pour utiliser un graphe ainsi que des algorithmes de plus court chemin. Il nous manquait donc les fonctions nécessaires aux envois de paquets sur le réseau, ainsi que des fonctions pour mettre à jour des parties du graphe.
C'est donc chose faite pour cette soutenance.

2.1 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 :

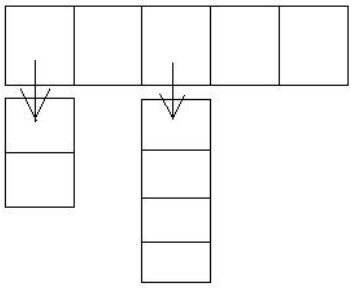
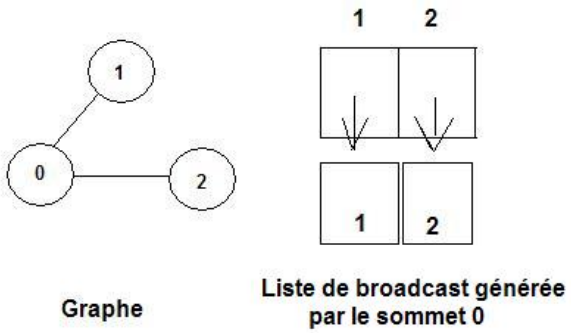
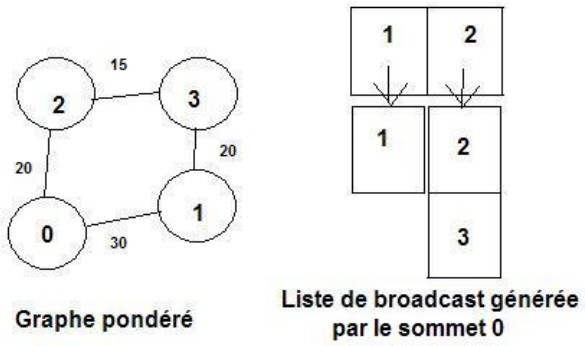


FIG. 1 – Exemple de structure d'une liste de broadcast

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.



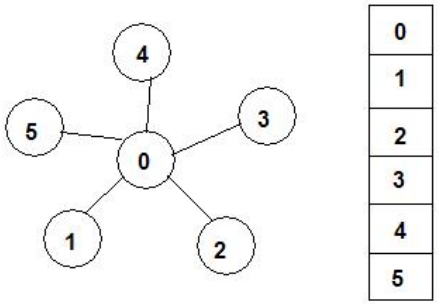
Ensuite nous parcourons l'intégralité de notre vecteur de sommet. A chaque fois si le sommet existe et que ce n'est pas une connection directe nous l'ajoutons dans notre liste de broadcast. On notera l'utilisation de la fonction gotonode(). Celle-ci est en fait une fonction de plus court chemin qui nous 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.

2.2 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. Pour cette soutenance, 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 qu'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 connections 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 existait 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. La complexité de cette dernière fonction est évidemment assez conséquente, de l'ordre du $\Theta(\text{Nlog(N)})$. Il serait donc préférable que l'on cherche des optimisations pour les soutenances suivantes.

2.3 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 voyait mais ne pouvait pas communiquer, ce qui était complètement stupide... (ça arrive même aux meilleurs...)
La solution que nous avons retenu 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.

2.4 Portage en objet Qt

Afin de faciliter l'intégration des graphes avec les objets réseaux et l'interface, nous avons transformé l'objet graphe en QObject qui est un objet QT. Ainsi, l'héritage des fonctions de cet objet nous permette de gérer les signaux au sein des graphes et de mettre a jour l'interface en temps réel.

3 Réseau

La partie implémentation du réseau s'est concretisée par l'intégration avec les graphes, ce qui a permis de pouvoir établir les premières connexions entre des clients.
Un changement majeur a été décidé, à savoir l'utilisation d'un protocole texte plutôt qu'un protocole binaire. En effet, le protocole TCP utilise des buffers aussi bien à l'émission qu'à la réception des données. Ainsi, on ne sait pas lorsque les données partent réellement et elles peuvent aussi arriver de manière groupée. Ainsi nous avions plusieurs paquets concaténés dans un seul bloc de données et il était impossible de différencier ces différents paquets. De plus, certains paquets se retrouvaient tronqués, la fin du paquet arrivant dans le bloc de données suivant. L'utilisation d'un protocole texte permet de contourner ce problème en introduisant un caractère de séparation entre les paquets, permettant de bien distinguer les paquets à la réception.
Ceci fait, les connexions fonctionnent maintenant correctement entre les nodes, l'établissement de la connexion se fait comme il faut, les clients s'échangent leurs étoiles afin de propager le graphe sur tous les nodes, les clients sont capables d'émettre des paquets en unicast, multicast et broadcast. Ils peuvent également router les paquets afin que deux nodes n'étant pas en connexion directe puissent communiquer.

3.1 Chat

Le systeme de chat fonctionne selon un principe relativement similaire à IRC, à savoir plusieurs Chans dans lesquels on peut discuter, nimporte qui pouvant créer un chan. Il y a donc deux types de paquet destinés à cela.
Tout d'abord la commande CUEV contient toutes les informations nécessaires a la gestion des évenements au sein du Chat (Connexion / Déconnexion d'un utilisateur d'un Chan). Ce type de paquet contient bien entendu l'entête classique, suivi du nom du Chan concerné puis enfin un booleen indiquant s'il s'agit d'une connexion ou bien d'une déconnexion. Ce type de paquet est envoyé systématiquement sous forme de broadcast, ainsi lorsqu'un utilisateur rejoint ou quitte un Chan, tout le monde sur le réseau est averti. Dès l'ors, si un utilisateur génère un paquet CUEV indiquant une connexion sur un Chan inconnu, tous les autres doivent donc considérer qu'il s'agit d'une création d'un nouveau Chan.

Le client gère une liste de Chans contenant, pour chaque Chan, la liste des utilisateurs présents dans ce Chan. Ainsi, le client met en permanence à jour cette liste grâce aux paquets CUEV qu'il recoit. Deux cas restent alors non déterminés : à la connexion et lorsqu'un utilisateur quitte le réseau.

Dans le cas de la connexion, le node auquel on se connecte va nous transmettre sa propre liste de Chans, ainsi on aura une liste à jour même si on arrive alors que des Chans existent déjà sur le réseau. Dans le cas où un utilisateur quitte le réseau, le graphe émet un signal déclenchant une fonction permettant de supprimer, dans la liste des chans, toutes les occurrences de l'utilisateur en question.
Lorsqu'on supprime le dernier utilisateur d'un Chan, le chan est alors également supprimé.
La deuxième commande concernant les Chans est CMSG. Celle-ci contient tout simplement un message d'un des Chans existant. Dans le paquet, après l'entête standard, on peut trouver le nom du Chan dans lequel s'inscrit ce message, suivi du contenu du message proprement dit. Ce paquet est émis en multicast, et donc seuls les utilisateurs faisant partie du Chan recoivent le message. Dans le cas où un client reçoit un message d'un Chan dans lequel il n'est pas, le message est tout simplement ignoré.
Il existe aussi un système de messages Privés, pour la discussion entre deux utilisateurs uniquement. Pour cela, un message CMPV est utilisé. Le paquet ne contient que l'entête standard et le contenu du message. Il est émis en unicast, directement de l'expéditeur à l'utilisateur cible.

3.2 RFC

3.2.1 Introduction

Ce document décrit la base du protocole Gloster qui permettra, à terme, 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.

3.2.2 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 routage 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.

3.2.3 Structure d'une Constellation

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

3.2.4 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.

3.2.5 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, l'ip publique du node, son port d'écoute, son type de connexion et sa vitesse de connexion (informations dans l'ordre et séparées par des espaces).

Pour pouvoir entrer dans une constellation, il faut d'abord obtenir l'accord d'un 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 123456789012345678901234 192.168.0.69 4242 2 123
A<B : ACPT Luc 123456789012345678901234 192.168.0.42 4242 3 51
```

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 A234567890123456789012345 12345678901234AS789012345,123456789012345678901234
Z A234567890123456789012345 192.168.0.1 4242 1 50,Fanfwe 12345678901234AS789012345 192.168.0.42 32817 1 50,loki 123456789012345678901234 192.168.0.4 33470 1 50

La personne recevant ce packet doit se débrouiller pour le renvoyer au travers de ses connexions directes, en direction de 12345678901234AS789012345 et 123456789012345678901234. 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. 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 12345678901234AS789012345,12345678901234567890123 48Z A2345678901234567890123 192.168.0.1 4242 1 50,Fanfwe 12345678901234AS789012345 192.168.0.42 32817 1 50,loki 12345678901234567890123 48 192.168.0.4 33470 1 50

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 fg3456789012345678901234 A2345678901234567890123Z #chan 1

Exemple de message au sein du Chan #chan :
CMSG fg3456789012345678901234 A2345678901234567890123Z #chan Coucou

Exemple de message privé entre deux nodes :
CMPV fg3456789012345678901234 A2345678901234567890123Z 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.

4 Gui et intégration

4.1 Gui

La menuBar a été modifiée : il y a maintenant une rubrique Menu qui donne accès à differentes fenêtres (Statut, Creation de chan...) ainsi qu'aux commandes de réorganisation des fenêtres (Cascades et Mosaique). Le menu Fichier s'est enrichit d'un item Connexion, qui ouvre une fenêtre contenant differents paramètres à indiquer, pour créer une connexion et/ou pour se mettre en écoute ; cette fenêtre est temporaire, l'utilisateur final n'aura pas à saisir certains parametres comme "Node id". Un nouveau menu fait son apparition : Fenetres, qui liste l'ensemble des fenêtres ouvertes. Bien sur, si on clique sur un nom de fenêtre dans cette liste, ladite fenêtre prend le focus. La mise à jour est faite à chaque ouverture de ce menu :

```
QWidgetList windows = wrk->windowList(QWorkspace::CreationOrder);
    Fenetres->clear();
    for (i = 0; i <int(windows.count()); i++)
     {
     id = Fenetres->insertItem(windows.at(i)->caption(), this, SLOT(winfocus(int)));
     Fenetres->setItemParameter( id, i );
     Fenetres->setItemChecked( id, wrk->activeWindow() == windows.at(i) );
     }
```

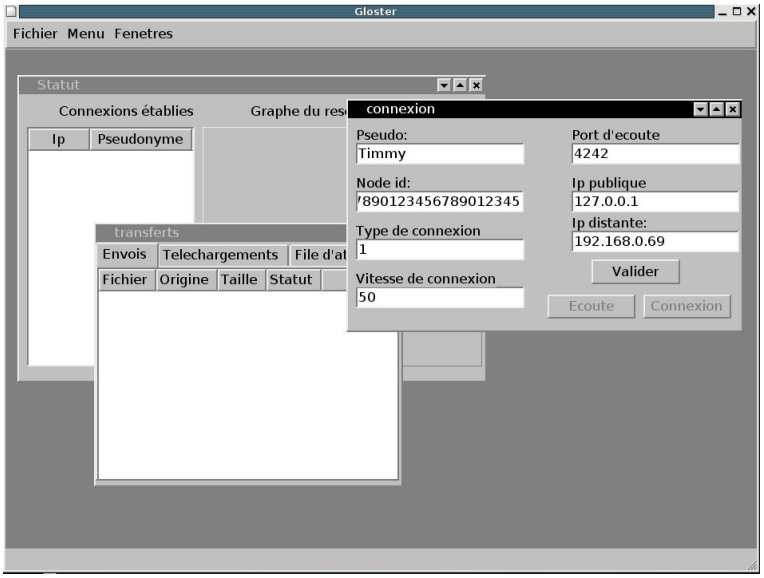


FIG. 2 –

Le menu "créer chan" sert enfin a quelque chose !! En cliquant sur Valider, on crée un nouveau chan :

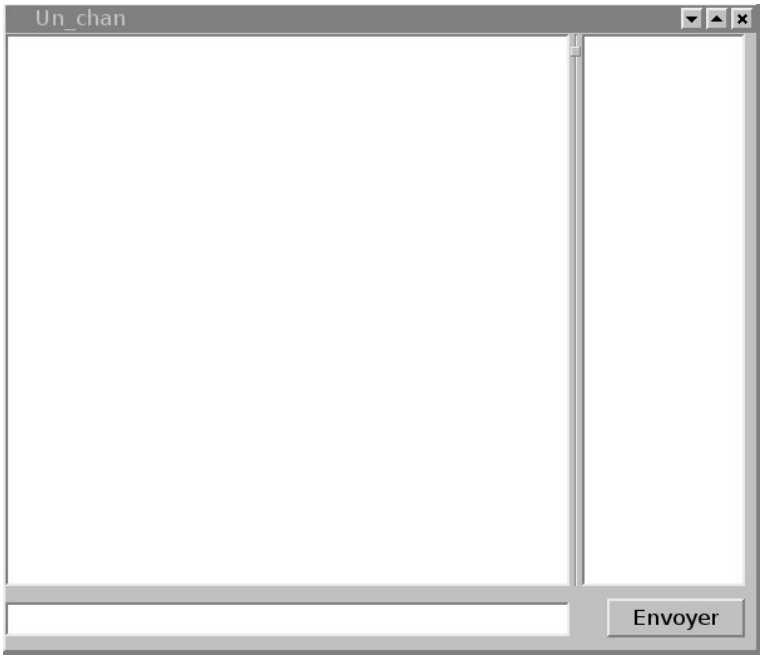


FIG. 3 – basique mais efficace

La gui n'est pas encore totalement terminée ; des fenetres comme Configuration évoluera en fonction de nos besoins (nouvelles options, sauvegarde des parametres dans un fichier...). De plus la fenêtre connexion ne restera pas comme ca ; pour le moment les divers paramètres servent pour des tests.

4.2 Intégration

Etant donné que tout à super bien été fait, pour l'intégration, il a suffit de connecter les différent signaux et slots de la Gui et de la partie reseau. Ce système de signaux et de slots est vraiment très partique ; les signaux sont en quelque sorte des "pointeurs de fonction" et s'utilisent de manière similaire.

5 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 est déjà mise en place et est destinée à accueillir les liens permettant de télécharger les releases du projet lorsque celui-ci sera suffisament avancé pour être utilisable et donc diffusable. Enfin, une page Liens donne une liste de sites Web qui nous tiennent à coeur.

6 Travail à accomplir

Pour la prochaine soutenance, nous avons comme but de finaliser complètement l'interface et le réseau afin de pouvoir commencer à attaquer la fonction principale du logiciel, c'est à dire le transfert de fichiers. Nous entamerons également le cryptage des paquets.
Pour le réseau, il reste à finaliser le chat pour l'échange des informations lors de la connexion d'un node, mettre en place une interface de pilotage à distance.
Pour l'interface, nous avons prévu de fixer le design des différentes fenêtres afin de ne pas avoir à y faire d'importantes modifications par la suite.
Pour le transfert de fichiers, nous devrons attaquer la création d'une connexion binaire lors du transfert de fichiers, la gestion de la liste des fichiers partagés, la gestion de la compression.
Pour le cryptage, nous devrons nous attaquer au système de clé privée / clé publique, ainsi qu'au cryptage proprement dit des données.

7 Conclusion

L'avancement du projet se fait conformément au planning établi dans le cahier des charges et aux modifications de la première soutenance. Les élements nécessaires au moteur du projet sont maintenant stables et assemblés. Nous avons pu tester gloster sur de multiples réseaux.

Maintenant que la base est prête nous allons pouvoir démarrer de nouvelles parties du projet. Ainsi nous obtiendrons certainement un programme utilisable par des Geeks pour la prochaine soutenance.

Nous tenons également à vous faire part de nos découvertes pendant le coding précédent cette soutenance : le tamarin est une plante multifonctions (huile, produit nettoyant, farine, boissons, laxatif, ...) très appréciée dans les pays asiatiques. Une autofellation est généralement impossible pour les hommes mais peut être réalisée grâce à un entraînement intensif (source Wikipedia). La gymnastique et le yoga peuvent aider. Vous pouvez être choqué mais nous vous rappellons que dans le cas particulier de la crise de cette fin de siècle, on ne peut se passer d'examiner chacune des solutions possibles.