Université de Mons
Faculté des Sciences
Institut d'Informatique

Étude du problème de copier/coller multi-plateformes et implémentation d'une solution

Directeur : M^r^ Olivier DELGRANGE

Projet réalisé par
Maëlick CLAES

Rapporteurs : M^r^ Bruno QUOITIN
M^r^ Sylvain DEGRANDSART

Année académique 2010-2011

Remerciements

Je remercie premièrement mon directeur Olivier Delgrange pour m'avoir proposé un sujet me permettant de résoudre un problème personnel et m'avoir ainsi donné la motivation de travailler sur celui-ci. Je le remercie, ainsi que les rapporteurs, Bruno Quoitin et Sylvain Degrandsart pour les commentaires qu'ils ont fait sur mon pré-rapport.

Je remercie aussi Bruno Quoitin pour son aide, ses réponses et ses conseils qui m'ont permis d'améliorer et de simplifier le protocole expliqué dans ce rapport.

Enfin je tiens aussi à remercier Pierre Declercq pour le temps passé à relire ce rapport, ainsi qu'Antoine Georis, Gregory Laus, Patrick Demeire et Pierre Jaradin pour les tests qu'ils ont effectués sur Clipsync.

Table des matières

Introduction

Lorsque l'on travaille sur un ordinateur, il est souvent plus agréable de travailler en utilisant plusieurs écrans. Cela permet par exemple d'afficher des informations sur un écran tout en prenant note sur un autre. De plus lorsqu'un utilisateur dispose de deux ordinateurs, par exemple un PC fixe et un portable, il en arrive à travailler en utilisant plusieurs écrans.

Seulement une différence majeure existe entre le fait de travailler avec plusieurs écrans sur un ordinateur et le fait de travailler avec plusieurs ordinateurs ayant chacun leur écran. Dans le premier cas il est aisé de faire transiter de l'information d'un écran à un autre, ceux-ci étant des périphériques reliés au même ordinateur. Dans le second cas cela est impossible sans passer par un moyen de communication tel que l'e-mail ou un support externe tel qu'une clé USB. Or cela est fastidieux s'il faut transmette peu d'information fréquemment.

Le but visé ici est donc de résoudre ce problème grâce à un système de copier/coller entre plusieurs plateformes *i.e.* entre plusieurs machines connectées sur le même réseau. Pour cela un premier chapitre présente le problème et énonce les objectifs et les contraintes du projet. Ensuite un second chapitre permet d'analyser les solutions existantes qui offrent la possibilité de résoudre le problème. Ces solutions sont présentées et leurs avantages et inconvénients sont donnés pour permettre de les comparer.

Un troisième chapitre définit une architecture réseau, ainsi qu'un protocole de communication, à utiliser pour implémenter un ensemble de logiciels permettant de résoudre le problème de copier/coller multi-plateformes. Suit un quatrième chapitre présentant l'implémentantion de cette solution, décrivant et justifiant le choix des outils utilisés pour la mettre en oeuvre. Une évaluation des performances est aussi faite et les fonctionnalités à améliorer ou à ajouter dans le futur sont identifiées.

Enfin une conclusion permet de rendre compte de l'avancement du travail réalisé et donne les grandes lignes qui guideront l'évolution future du logiciel nouvellement implémenté.

Chapitre 1

Problème

1.1 Objectifs

Le premier objectif de ce projet est de donner à un utilisateur la possibilité de faire du copier/coller entre plusieurs ordinateurs allumés. Le logiciel créé doit être une solution, simple et légère, à installer sur chaque poste client. C'est-à-dire qu'il doit être facilement configurable, afin de pouvoir rapidement fournir le service désiré, et il doit avoir une charge minimale sur le système. Il doit permettre de faire facilement l'opération de copier/coller sans avoir à passer par un échange de mails ou de fichiers. Il doit aussi viser le monde Unix en premier lieu, mais doit être adaptable à d'autres systèmes d'exploitation.

Afin de fixer certains termes plusieurs définitions sont nécessaires.

Définition 1.1.1. *Un* presse-papier *est une zone mémoire dans laquelle une ou plusieurs données sont stockées. Celles-ci peuvent être de différents types,* e.g. *du texte, une image, un fichier.*

Remarque 1.1.1. *Bien qu'habituellement un presse-papier ne permet de stocker qu'une seule donnée à la fois, il existe un grand nombre de logiciels permettant de gérer un presse-papier contenant un ensemble de données, ce qui permet d'avoir ainsi un* historique *de ce qui a été copié dedans.* e.g. *GNU Emacs [***emacs***] permet de cycler sur cet historique en revenant au début de celui-ci lorsqu'il a été entièrement parcouru. L'environnement de bureau Xfce inclut un plugin permettant de gérer l'historique du presse-papier de X Window [***xfce-clipman***]*

Définition 1.1.2. Copier *est l'action d'écrire une donnée dans le presse-papier.*

Définition 1.1.3. Coller *est l'action de récupérer la donnée présente dans le presse-papier.*

Définition 1.1.4. Copier/coller *résume le concept permettant de copier quelque chose dans le presse-papier et le coller ensuite.*

Définition 1.1.5. *Copier/coller* multi-plateformes *décrit la possibilité de faire du copier/coller entre deux ordinateurs connectés en réseau. L'abréviation* CCMP *sera parfois utilisée dans la suite de ce rapport* [1].

1.2 Contraintes

La priorité est de fournir un système léger. Celui-ci doit être facilement installable et configurable, discret et l'*overhead* sur le système doit être minimisé. Il ne doit donc pas reposer sur un autre système plus lourd comme le partage de fichiers. Dans un premier temps, il ne doit gérer que le texte mais il doit être extensible. Ceci afin de permettre facilement l'ajout du support pour d'autres formats de données comme les images. Une autre contrainte importante à prendre en considération est l'aspect réseau. Celui-ci introduit de potentiels problèmes de sécurité. Il faudra donc trouver un moyen de réduire ceux-ci *e.g.* en chiffrant la connexion réseau et en obligeant l'utilisateur à s'authentifier afin d'utiliser le logiciel. De même, le logiciel devant tourner sur plusieurs systèmes d'exploitations différents dont Linux, il est plus que préférable que le logiciel soit libre de droits et se base sur des protocoles et technologies ouverts.

1. Un protocole de chiffrement de la norme IEEE 802.11i est aussi abrégé CCMP, cependant celui-ci n'ayant aucun lien avec le sujet du projet, l'utilisation de cette abréviation ne créera pas d'ambiguïté.

Chapitre 2

Analyse des solutions existantes

Cette section détaille l'analyse des solutions existantes *i.e.* la présentation et comparaison de ces différentes solutions. Celles-ci sont divisées en deux catégories. La première reprend les solutions dites *lourdes*, *i.e.* qui reposent sur des logiciels permettant de faire beaucoup plus que du copier/coller *e.g.* les *bureaux virtuels*. La seconde reprend les solutions *potentiellement légères* i.e. qui sont des logiciels qui *a priori* conviendraient comme solution au problème.

Tout d'abord sont étudiés un ensemble de logiciels et protocoles donnés comme mots-clés par le directeur de projet au début de celui-ci. Ceux-ci sont principalement liés à des technologies lourdes et sont majoritairement ceux entrant dans la partie des solutions lourdes. Les autres solutions lourdes sont surtout des solutions plus conventionnelles comme l'échange d'e-mails ou le partage de fichiers.

2.1 Solutions lourdes

Les solutions étudiées dans cette section sont d'abord l'échange d'e-mails, l'utilisation d'un serveur *FTP* (File Transfer Protocol), le partage de fichiers et les outils de travail collaboratif. Les autres solutions étudiées sont *Remote Desktop Service*, *VNC*, *Citrix XenApp*, la technologie *NX* et *ClusterSSH*. La comparaison de ces solutions est résumée dans la table **??**.

2.1.1 Échange d'e-mails

Une première solution qui vient rapidement à l'esprit est l'utilisation du courrier électronique. Il est en effet facile d'échanger du texte ou un fichier quelconque entre deux machines en utilisant celui-ci. Par exemple il suffit simplement d'envoyer à sa propre adresse le texte à copier/coller ou bien de joindre le fichier que l'on désire transférer. Cette solution à l'avantage d'être utilisable sur n'importe quelle machine (et n'importe quel système d'exploitation) sans l'installation d'un logiciel particulier autre qu'un client mail.

Cependant ceci reste très fastidieux. Outre l'utilisation d'un outil qui n'est pas prévu pour faire du copier/coller, ceci nécessite l'installation d'un serveur mail sur le réseau local, ou bien un accès à Internet. Cet accès à Internet pose d'ailleurs un problème important à l'heure actuelle où les connexions fournies aux particuliers ne proposent pas une vitesse d'envoi très élevée. Un réseau intranet câblé en *Fast Ethernet* permettra généralement un débit symétrique d'au moins 100 Mbits/s, alors qu'une connexion ADSL standard dépasse rarement les quelques Mbits/s en upload. Les différences de latence entre l'internet et l'intranet risquent aussi d'être fort importantes.

Ceci permet de montrer que, non seulement il n'est pas nécessaire d'utiliser Internet pour échanger des données entre deux machines qui sont normalement situées dans la même pièce, mais que cela peut en plus apporter une perte de performances si l'on dispose d'une connexion à faible débit [1]. L'utilisation d'Internet est donc écartée par la suite pour ces raisons et seule l'utilisation sur un réseau local est considérée.

2.1.2 Utilisation d'un serveur FTP

Un moyen de faire du copier/coller entre plusieurs plateformes est l'utilisation d'un serveur FTP. Pour cela il faut qu'un serveur soit installé sur le réseau local. De même chaque client doit disposer d'un client FTP. Sur Unix il est aussi possible de se passer d'un serveur FTP et d'utiliser un serveur *SSH* (Secure Shell Client), le protocole *SFTP* (SSH File Transfer Protocol) permettant d'utiliser un serveur SSH à la manière d'un serveur FTP sécurisé.

La solution du serveur FTP a donc comme contrainte l'installation du logiciel serveur. Celle-ci peut être en partie résolue sous Unix grâce à SSH. En effet il peut être assez fréquent d'avoir un serveur SSH installé en local pour l'utilisateur Unix utilisant plusieurs machines simultanément. Cependant le fait de créer un fichier pour copier/coller du texte engendre une lourdeur non désirée. Cette dernière remarque s'applique aussi pour le partage de fichiers.

2.1.3 Partage de fichiers

Le partage de fichier permet d'accéder à un répertoire se trouvant sur une machine A à partir d'une machine B se situant sur le même réseau. Celui-ci peut être mis en oeuvre de plusieurs manières. Sur Windows il est implémenté nativement via le protocole SMB. Sous Unix le protocole *NFS* (Network File System) est sans doute le plus répandu. Une alternative peut aussi être l'utilisation de SSH et *SSHFs* (SSH Filesystem qui permet de monter un dossier sur une machine distante de manière semblable à un périphérique et qui repose sur SFTP). Le partage

1. Ce qui est actuellement le cas en Belgique ainsi que dans la plus grande partie du monde où la fibre optique et les connexions symétriques sont peu répandues

de fichiers de Windows n'étant pas compatible sous Unix et *vice versa*, l'utilisation de *Samba* [**samba**] reste obligatoire s'il est nécessaire de supporter ces deux mondes. Cette solution présente aussi les mêmes désavantages que l'utilisation de FTP.

2.1.4 Outils de travail collaboratif

Ces outils permettent de travailler à distance sur un document de manière collaborative. Bien souvent ces outils s'utilisent à travers une interface web et permettent de voir les modifications apportées au document en temps réel. Ces outils peuvent donc être utiliés comme méthode pour faire du CCMP en utilisant un document collaboratif comme presse-papier. Etherpad[**etherpad**] est un de ces outils et a l'avantage d'être open-source depuis son rachat par Google. Cependant, aussi bien Etherpad que tout autre logiciel de ce genre, ne permettent pas de faire du CCMP avec une configuration minimale et sont clairement une solution lourde pour celui qui n'a pas besoin d'utiliser un outil de travail collaboratif.

2.1.5 Remote Desktop Services

Remote Desktop Services, autrefois *Terminal Services* est un composant de Microsoft Windows permettant l'utilisation d'un ordinateur à distance tournant sous Windows[**wiki:rds**]. Ce système est basé sur un modèle client-serveur. Le serveur est appelé Terminal Server et est inclus dans Windows. Il est à noter que seule la version serveur de Windows permet une configuration avancée du programme serveur. Le protocole utilisé est appelé *RDP* (Remote Dekstop Protocol) et peut être transporté dans un tunnel *TLS* (Transport Layer Security, anciennement *SSL*, Secure Sockets Layer) afin d'améliorer la sécurité du protocole. L'utilisation de ce protocole est possible sous les systèmes d'exploitation basés sur Unix grâce à l'implémentation libre *rdekstop* [**rdesktop**] du client. Le protocole et le serveur supportent le partage du presse-papier, cependant il est évident que cette solution est inadéquate pour être utilisée comme copier/coller multi-plateformes.

2.1.6 VNC

VNC (Virtual Network Computing) [**wiki:vnc**] est un système logiciel permettant d'utiliser un ordinateur à distance. Il a comme avantage sur le protocole de Microsoft d'être libre de droits et d'être indépendant du système d'exploitation. Bien que non sécurisé par défaut, il existe différents moyens de le sécuriser *e.g.* via une connexion SSH ou *VPN* (réseau privé virtuel). Cette solution souffre des mêmes problèmes que Remote Desktop Services, c'est-à-dire qu'il permet de faire bien plus que du copier/coller et allourdi le système.

2.1.7 Citrix XenApp

Citrix XenApp [**wiki:xenapp**] est un ensemble de produits permettant de virtualiser des applications sur différentes machines. Il distribue des services tournant généralement sur un ou plusieurs serveurs à des clients dits légers, *i.e.* qui n'ont pas besoin d'avoir une grande quantité de ressources matérielles disponibles pour exécuter les applications, vu que celles-ci sont exécutées sur un serveur central. Contrairement à VNC qui ne distribue que ce qui est affiché, XenApp fonctionne de manière semblable à *X11* [2]. Cependant ceci ne l'empêche pas de souffrir de problèmes déjà évoqués, tels un overhead important lorsque l'on veut faire du copier/coller et l'utilisation de licence propriétaire.

2.1.8 Technologie NX

NX [**wiki:nx**] est un protocole d'accès distant à X11 reposant sur un modèle client-serveur et utilisant SSH pour la sécurité. L'implémentation de base *NoMachine NX* est propriétaire mais une implémentation libre *FreeNX* [**freenx**] existe. Tout comme les solutions précédentes, NX permet bien plus que le copier/coller et souffre donc des mêmes problèmes.

2.1.9 ClusterSSH

ClusterSSH[**clusterssh**] est un outil permettant de gérer plusieurs sessions SSH dans un seul terminal. Il lance ainsi sur chaque hôte la même commande ce qui permet par exemple de configurer plusieurs serveurs de la même manière. Son premier désavantage est la nécessité d'installer et de configurer un serveur SSH sur chaque machine. Deuxièmement il ne permet pas directement de faire du copier/coller, il faut pour cela passer par un outil tel que xclip[**xclip**] afin d'interragir avec le presse-papier de X Window. Ces deux défauts impliquent que ClusterSSH est peu pratique afin de fournir un service de CCMP.

2.1.10 Comparaison

En résumé, toutes ces solutions lourdes présentées reposent sur un modèle client-serveur et sont conçues, soit pour fournir un service qui n'est pas prévu pour être utilisé afin de réaliser du copier/coller, soit pour être utilisées comme bureau virtuel distant. Certaines sont plus simples que d'autres à mettre en œuvre ; d'autres sont plus sécurisées, tandis que d'autres sont propriétaires ou visent un système d'exploitation particulier. Mais elles sont toutes inadaptées pour effectuer un copier/coller multi-plateformes simple. Leurs caractéristiques sont résumées dans la table **??**.

2. X Window, X11 ou X est le système graphique standard de Unix fonctionnant sous forme de serveur et où chaque application graphique est un client

Solution	Service rendu	Architecture	Sécurité	Indépendance de la plateforme	Ouverture de la solution
E-mails	E-mails	Client-serveur	Dépend du protocole	Oui	Protocoles ouverts
FTP	Transfert de fichiers	Client-serveur	SSH grâce à SFTP	Oui sauf SFTP	Protocoles ouverts
Partage de fichiers	Partage de fichiers	Client-serveur	SSH sous Unix	Oui grâce à Samba	Libre sous Unix, fermé sous Windows
Etherpad	Outil de collaboration en ligne	Client-serveur	HTTPS + Mot de passe	Oui	Logiciel libre
RDS	Bureau distant	Client-serveur	Tunnel TLS possible	Windows mais clients Unix existants	Protocole propriétaire
VNC	Bureau distant	Client-serveur	Possibilité d'utiliser SSH ou un VPN	Oui	Logiciel libre
XenApp	Bureau distant	Client-serveur	Possibilité d'utiliser HTTPS	MS Windows Server, HP-UX, Solaris, AIX	Logiciel propriétaire
NX	Bureau distant	Client-serveur	Oui	Vise Unix	Implémentation libre FreeNX
ClusterSSH	Mutli-SSH	Client-serveur	SSH	Vise Unix	Logiciel libre

TABLE 2.1 – Comparaison des solutions lourdes

2.2 Solutions *a priori* légères

Les solutions présentées dans cette section sont pour la majorité des solutions trouvées en faisant des recherches sur Google sur base de mots clés français et anglais. Celles-ci ont permis de trouver des logiciels qui conviendraient au premier abord en fournissant un moyen simple de faire du copier/coller en réseau. Les logiciels présentés seront *CL1P*, *ClipboardMultiSharer*, *Clipboard Share*, *The Network Clipboard* et *Remote Clip*. La comparaison de ces solutions est résumée dans la table **??**.

2.2.1 CL1P

CL1P[**cl1p**] est un site web dont le but est de permettre de partager de l'information entre plusieurs ordinateurs via Internet. Celui-ci peut être vu comme une adaptation des logiciels de travail collaboratif au CCMP. Bien que s'utilisant de manière simple, il ne permet pas de faire du CCMP en synchronisant le presse-papier du système d'exploitation. Il est en effet nécessaire d'avoir un navigateur web ouvert et de copier dans une page web la donnée du presse-papier pour la récupérer sur une autre machine sur la même page web. Une autre critique importante est le manque de décentralisation de la solution : il est absurde d'avoir à se connecter à un serveur présent aux États-Unis pour échanger de l'information entre deux ordinateurs qui sont dans la même pièce. Son dernier défaut majeur est de ne pas être basé sur un logiciel libre que tout le monde pourrait télécharger et installer sur son propre serveur web, ce qui aurait permis de résoudre en partie le problème de la centralisation.

2.2.2 ClipboardMultiSharer

ClipboardMultiSharer [**clipmsharer**] est un logiciel écrit en *Java* et en *C#* qui permet de faire du copier/coller entre plusieurs ordinateurs. Celui-ci supporte le copier/coller de texte et d'image et repose sur l'utilisation d'un fichier partagé en réseau. Ceci en fait donc en réalité une solution lourde vu qu'il requiert l'utilisation du partage de fichiers. De plus, bien que le programme soit écrit en Java, la portabilité du programme dépend du type de partage de fichiers utilisé. Il sera donc sans doute nécessaire d'utiliser Samba s'il est nécessaire de travailler entre Unix et Windows.

2.2.3 Clipboard Share

Clipboard Share [**clipshare**] est un autre logiciel qui permet de faire du CCMP. La connexion est chiffrée et il est possible de recevoir du contenu en provenance d'un envoyeur de confiance. Il est écrit en C#, requiert *Microsoft .NET 3.5*

ainsi que *PNRP* (*Peer Name Resolution Protocol*), un protocole *P2P* [3] propriétaire de Microsoft, ce qui signifie que le logiciel ne tourne que sur Windows XP SP2 ou plus récent [**wiki:pnrp**]. Bien qu'il propose des fonctionnalités intéressantes et réponde au critère de logiciel léger, il ne convient pas car il ne vise que Windows et repose sur des technologies propriétaires.

2.2.4 The Network Clipboard

The Network Clipboard [**netclip**] est lui écrit en *C++* et fonctionne aussi bien sous Windows que sous Linux. Le protocole mis en place permet d'utiliser l'application en P2P grâce au *broadcast IP*. Cependant il possède plusieurs défauts qui l'empêchent d'être un bon candidat. Premièrement le contenu n'est pas chiffré sur le réseau et aucun moyen d'authentification ne semble être mis en oeuvre pour sécuriser le système. Ensuite il ne semble pas certain que le programme tourne sur tous les Unix, *e.g.* il n'est fait mention nulle part d'une compatibilité assurée avec *MacOS*. Enfin, il faut tout de même noter que The Network Clipboard utilise la version 3 de *Qt* [4] et qui, avant la version 4, n'était libre que sous Linux [**wiki:qt**]. Le programme repose donc sur une librairie propriétaire sous les autres systèmes d'exploitation.

2.2.5 Remote Clip

Remote Clip [**remoteclip**] est à la base un outil pour synchroniser du contenu entre Windows et un *PDA Palm*. Celui-ci existe aussi en version Java et lui permet ainsi d'être portable. Il repose sur une architecture P2P dont le fonctionnement est expliqué dans un article de Robert C. Miller et Brad A. Myers [**Miller99syncclips**]. Les connexions sont également chiffrées via TLS à partir de la version 1.4 de Java et l'ajout d'un pair dans un groupe de partage du presse-papier nécessite l'accord explicite de la machine gérant celui-ci. Il permet de gérer aussi bien le texte que les fichiers et est distribué sous licence libre.

Remote Clip semble donc être le logiciel répondant à toutes les exigences requises mais contrairement aux solutions citées plus haut, le projet ne semble plus actif. En effet la dernière version du logiciel est datée de juillet 2002. Cela signifie que de potentielles failles de sécurité ne seront pas corrigées et que le logiciel ne sera plus amélioré. Bien que cela ne soit pas directement handicapant, il peut malgré tout être utile d'avoir un logiciel maintenu. Par exemple le manque d'adresse IPv4 et le passage à l'IPv6 peuvent être une bonne motivation pour avoir une solution à jour.

3. Peer-to-Peer, pair à pair en français, le principe de ce type de réseaux est décrit dans la section **??** à la page **??**

4. Framework C++ utilisé entre autre dans le projet *KDE*

2.2.6 Comparaison

En résumé toutes les solutions présentées ont chacune des qualités et des défauts. CL1P a l'avantage d'être simple d'utilisation mais ne propose cependant aucune vraie syncrhonisation entre le presse-papier du sysème d'exploitation. De même celui-ci est centralisé et propriétaire. ClipboardMultiSharer repose sur du partage de fichiers (et donc un modèle client-serveur). Clipboard Share repose sur des technologies fermées et ne tourne que sous Windows. The Network Clipboard n'est pas sécurisé. Seul Remote Clips répond réellement aux exigences du projet même si celui-ci est quelque peu vieillissant. Finalement, le but de ce projet étant aussi d'implémenter une nouvelle solution, celle qui sera décrite par la suite reprendra plusieurs principes de Remote Clip, mais aussi le broadcast le broadcast utilisé dans The Network Clipboard pour l'auto-configuration. Les caractéristiques de l'ensemble des logiciels sont résumées dans la table **??**.

Solution	Service rendu	Architecture	Sécurité	Indépendance de la plateforme	Ouverture de la solution
CL1P	CCMP Web	Client-serveur	HTTPS + Restrictions à certains utilisateurs possible	Oui	Logiciel propriétaire
ClipboardMultiSharer	CCMP	Client-serveur	Partage de fichiers	Java + partage de fichiers	Logiciel libre
Clipboard Share	CCMP	P2P	Connexion cryptée + envoyeur de confiance	Windows (.NET 3.5 + PNRP)	Logiciel libre mais technologies MS
The Network Clipboard	CCMP	P2P	aucune	Linux + Windows	Logiciel libre
Remote Clip	CCMP	P2P	TLS + validation de connexion d'un pair	Java	Logiciel libre

TABLE 2.2 – Comparaison des solutions légères

Chapitre 3

Solution proposée

Après analyse des différentes solutions existantes pour faire du CCMP dans le chapitre précédent, la solution jugée la plus efficace est Remote Clip, celle-ci proposant une architecture P2P adéquate et un niveau de sécurité correct. Pour ces raisons, certaines idées mises en avant par Remote Clip sont réutilisées ici. De même le principe de messages broadcast utilisé dans le but d'auto-configurer le réseau, comme fait dans The Network Clipboard, est aussi repris.

3.1 Principes d'une architecture P2P

Avant de décrire l'architecture qui utilisée, il est préférable de rappeler quels sont les principes de base d'une architecture P2P. Habituellement, une architecture client-serveur est utilisée lorsqu'il faut fournir un service à un ensemble de machines connectées en réseau, c'est-à-dire que chaque client va se connecter à un (ou parfois plusieurs) serveur central qui se chargera de fournir le service désiré. Par opposition à ce mode de fonctionnement centralisé, un réseau organisé en pair-à-pair permet de se passer de serveur central, chaque client jouant à la fois le rôle de client et de serveur. La figure **??** illustre la différence de topologie réseau existante entre ces deux types d'architectures réseau. De manière plus précise les systèmes P2P sont définis dans [**AS04**] comme étant :

> des systèmes distribués constitués de noeuds interconnectés, capables de s'auto-organiser dans des topologies de réseaux avec comme but le partage de ressources, telles que le contenu, les cycles CPU, le stockage, la bande passante tout en ayant la capacité de s'adapter aux erreurs et de s'accommoder de populations de noeuds transitoires ; tout en maintenant une connectivité et des performances acceptables sans requérir l'intermédiaire ou le support d'un serveur ou d'une autorité centralisée globalement.

Les difficultés potentielles à mettre en évidence dans ce genre de systèmes sont la gestion du va-et-vient de clients et la tolérance aux erreurs sans l'aide de

serveur central, et la minimisation de la charge sur le réseau, due à cette gestion. Il faut cependant noter que l'importance de la tolérance aux erreurs est tout de même assez minime dans le contexte du copier/coller. Les données stockées sont en général destinées à être utilisées à court terme et non pas à être stockées de manière persistante. De même le nombre de pairs est normalement peu élevé, et donc le va-et-vient n'est pas aussi important que sur des systèmes à grande échelle.

Une remarque supplémentaire à faire est qu'une différence est souvent faite entre réseaux P2P structurés et non structurés [**AS04** ; **Lua05asurvey**]. Les premiers ont une topologie contrôlée de manière précise permettant de placer chaque donnée à un endroit précis et d'effectuer des recherches efficaces. Le réseau ici étant censé être de taille relativement petite et le contenu changeant très rapidement, il n'est pas nécessaire de structurer le réseau. En effet, le fait de structurer le réseau, en utilisant par exemple une table de hashage distribuée, a comme intérêt d'avoir une complexité algorithmique sous-linéaire en fonction du nombre de pairs présents dans le réseau. Ici la taille du réseau étant fortement réduite, cette complexité a peu d'impact sur les performances du logiciel. Le fait de ne pas utiliser une table de hashage distribuée permet donc d'avoir un protocole plus simple sans perte de performances.

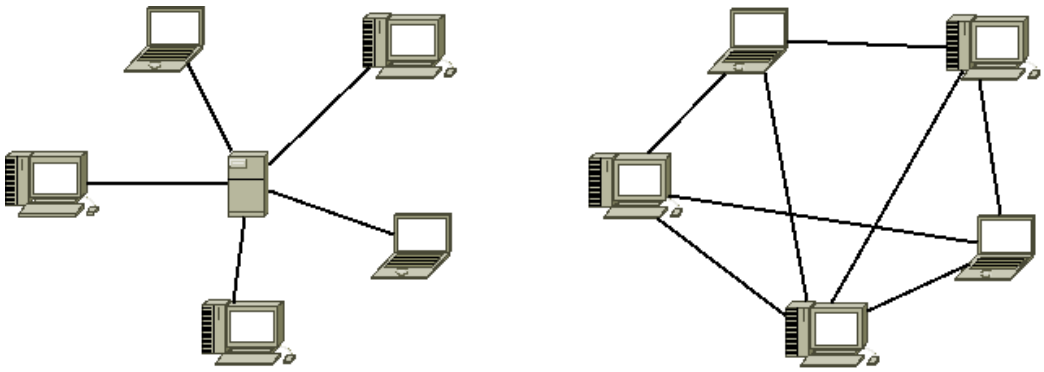

FIGURE 3.1 – Exemple de différence entre une topologie client-serveur et P2P.

3.2 Architecture logicielle

Afin de suivre le principe *KISS* (Keep It Simple Stupid, *cf.* Annexe **??**) et la philosophie Unix, le projet est divisé en plusieurs programmes, chacun d'entre eux s'occupant d'une tâche particulière. Une autre raison justifiant une telle conception est le fait qu'elle introduit la possibilité de développer le projet de manière incrémentale, en se concentrant d'abord sur l'aspect réseau et ensuite sur l'implémentation propre à un environnement particulier.

3.2.1 Client P2P

Premièrement un logiciel s'occupant uniquement de la gestion du presse-papier sur le réseau est nécessaire. Celui-ci a comme rôle de se connecter aux autres pairs et de s'organiser avec ceux-ci quand cela est nécessaire. Sa seule fonctionnalité est en fait de gérer la partie réseau du système, toute interaction locale (*i.e.* avec l'utilisateur) se fait par l'intermédiaire d'autres logiciels qui seront définis par la suite.

Il faut définir de manière plus précise comment l'ensemble des clients P2P vont s'organiser afin de s'informer pour savoir quel client *détient* le presse-papier, comment un pair entre dans le réseau et comment détecter qu'un pair a quitté celui-ci (volontairement ou pas).

Gestion du presse-papier en P2P

Il y a principalement deux possibilités pour gérer le presse-papier sur le réseau. Lorsqu'un copier est effectué en local, le client P2P doit prévenir les autres pairs qu'il détient le presse-papier. La première option est d'envoyer en même temps la donnée copiée à tous les pairs. La deuxième est de n'envoyer cette donnée que lorsque qu'un coller est effectué chez un pair, celui-ci peut alors demander la donnée au pair détenant le presse-papier. La première manière de faire a les avantages et inconvénients suivants :

- Avantages :
  - Possibilité de garder une copie du presse-papier sur chaque pair.
  - Tolérance aux erreurs dans le cas où le pair détenant le presse-papier aurait quitté le réseau.
  - Absence de communications réseaux en cas de multiples collers.
- Inconvénients :
  - Consommation de bande passante accrue dans le cas où des copiers sont effectués fréquemment car il faudra envoyer la même donnée à chacun des pairs qui ne feront peut être pas forcément de coller. De plus la consommation de bande passante dépend du nombre de pair présents sur le réseau (même si celui-ci reste faible).

La seconde a les avantages et inconvénients suivants :

- Avantages :
  - Il n'est pas nécessaire de notifier l'ensemble des pairs à chaque copier : si un pair effectue plusieurs copies d'affilée, il ne faudrait notifier le réseau qu'une seule fois et sans envoyer la moindre donnée.
  - En général le coller ne sera effectué que par un seul pair, il n'a donc qu'à demander lui même au pair détenant le presse-papier de la lui envoyer.
- Inconvénients :
  - Faible tolérance aux erreurs : impossibilité de récupérer le presse-papier d'un pair ayant quitté le réseau.

– Si plusieurs collers ont lieu les uns à la suite des autres, il faudra demander plusieurs fois la même donnée au pair détenant le presse-papier.

X Window

Avant de choisir entre une de ces deux solutions, il est important de savoir ce qu'est X Window et comment ce logiciel fonctionne[**nye1992xlib**]. X est un standard pour les systèmes d'exploitations basés sur Unix permettant d'afficher des éléments graphiques sur un écran plutôt qu'un terminal. Il est principalement responsable de la couche la plus basse des systèmes de fenêtrage de la majorité des systèmes Unix. Son fonctionnement est basé sur une architecture client-serveur où X Window est le logiciel serveur et chaque application graphique est un client. Lorsqu'un copier est réalisé au sein d'une application, le client (*i.e.* la fenêtre) notifie le serveur X qu'un copier a eu lieu. Le serveur retient alors que la dernière application à avoir copié du texte est le client en question.

Lorsqu'une deuxième application effectue un coller, elle demande d'abord au serveur quel client a fait le dernier coller. Il peut ensuite demander à ce client de délivrer le contenu de son presse-papier. Il faut noter que si le client responsable du presse-papier a disparu, *i.e.* si la fenêtre a été fermée, et qu'aucun copier n'a été effectué depuis la fermeture du client, alors il est impossible de récupérer la donnée et le presse-papier est considéré comme vide [1]. En fin de compte, il est clair que la description du fonctionnement de X Window est la deuxième solution décrite précédemment.

Choix d'une solution pour la gestion et synchronisation du presse-papier

Il faut donc faire un choix entre une de ces deux solutions. Un des critères défini au début de ce travail étant de cibler Unix, il semble idéal de se baser sur X Window pour implémenter une nouvelle solution. De même, Remote Clip, qui a été décrit comme la meilleure solution rencontrée, fonctionne de la même manière. Cependant en fin compte, c'est la première solution envoyant le presse-papier lors de la copie qui est choisie. Celle-ci est plus intuitive et a surtout l'avantage d'être tolérante aux erreurs. Et même si Remote Clip a été identifiée comme solution idéale, un intérêt de réimplémenter une nouvelle solution est d'améliorer ses fonctionnalités, en améliorant ici la tolérance aux erreurs. Un dernier avantage qui est discuté dans le chapitre **??** est le fait qu'envoyer la donnée à chaque pair à chaque copier facilite la gestion d'historique synchronisé du presse-papier

1. Il faut cependant noter que si un logiciel d'historique de presse-papier est utilisé, c'est le gestionnaire de presse-papier qui est responsable de la donnée copiée. Il est donc normal si le presse-papier est toujours disponible après avoir fermé la fenêtre dans laquelle la donnée a été initialement copiée.

sans avoir à ajouter cette fonctionnalité directement dans le logiciel nouvellement produit.

Rejoindre le réseau

Outre un moyen d'authentification, il est nécessaire de définir comment un pair peut rejoindre le réseau afin de partager son presse-papier. Une première solution consiste, pour un pair voulant rejoindre le réseau, à connaître l'adresse IP et l'identifiant d'au moins un autre pair déjà présent dans le réseau et connaissant les autres pairs présents dans ce réseau. Il suffit alors de contacter cet autre pair et de synchroniser la liste des pairs de l'ensemble du réseau. Une autre solution envisageable est d'utiliser un mécanisme de broadcast permettant de flooder [2] le réseau pour rechercher les pairs désirant rejoindre le réseau. Cette solution sera celle utilisée, celle-ci permettant de minimiser le nombres de paramètres que devra configurer l'utilisateur.

Pour cela, il faut que chaque pair envoie à intervalle régulier un message en broadcast, sur le réseau local, permettant d'identifier le groupe de pairs (via un nom choisi par l'utilisateur) auquel il appartient ou veut appartenir. Tout pair appartenant au groupe (*i.e.* dont le groupe possède le même nom) et recevant un message doit alors contacter le pair ayant envoyé le message en ouvrant une connexion TCP. L'intérêt de cette connexion TCP est de s'assurer que le presse-papier est reçu dans son intégrité par chaque pair (les messages broadcast étant envoyés via des paquets UDP). De plus, ceci permet à chaque pair de s'identifier afin de s'assurer que seuls les membres du groupe (*i.e.* les machines de l'utilisateur présentes sur le réseau local) ont accès au presse-papier. Pour cela un mécanisme d'authentification est décrit dans la suite de ce chapitre.

Il faut cependant prêter attention à un problème : le nombre de connexions TCP ouvertes peut devenir une charge importante pour le réseau. En effet pour un groupe de n pair, $n - 1$ connexions sera ouverte pour chaque pair. Le nombre de connexions serait donc en $O(n^2)$. Cependant le réseau P2P étant limité à quelques pairs dans le cas du CCMP, ceci n'est pas handicapant. De même afin d'ouvrir plusieurs fois une connexion TCP avec un pairs déjà connu, il faut gérer une table des pairs dans laquelle sera ajouté un pair identifié. Les figures **??** et **??** illustrent par des diagrammes d'états comment un pair se comporte pour contacter les autres pairs du réseaux.

Quitter le réseau

Lorsqu'un pair quitte le réseau, il doit notifier l'ensemble des autres pairs de son départ. Une fois ceci fait, il peut fermer chaque connexion ouverte avec chacun des pairs. En revanche il se peut qu'un pair quitte le réseau sans pouvoir notifier

2. Bien que le fait d'innonder le réseau de messages ait un aspect négatif, la quantité de bande passante reste cependant faible, les messages envoyés étant relativement courts.

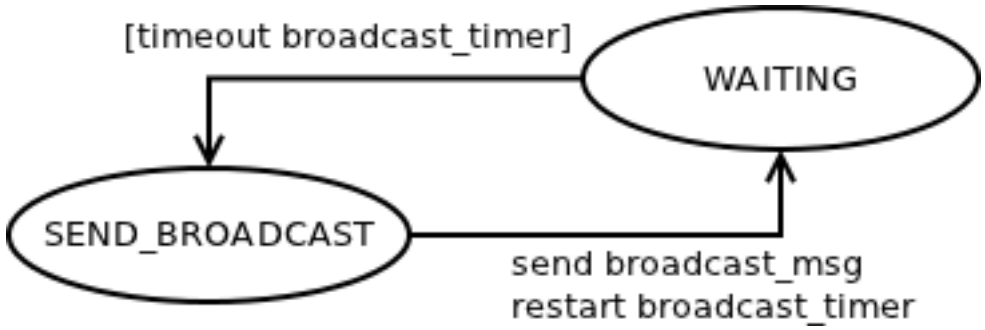

FIGURE 3.2 – Diagramme d'états modélisant l'envoi de messages en broadcast par le client P2P.

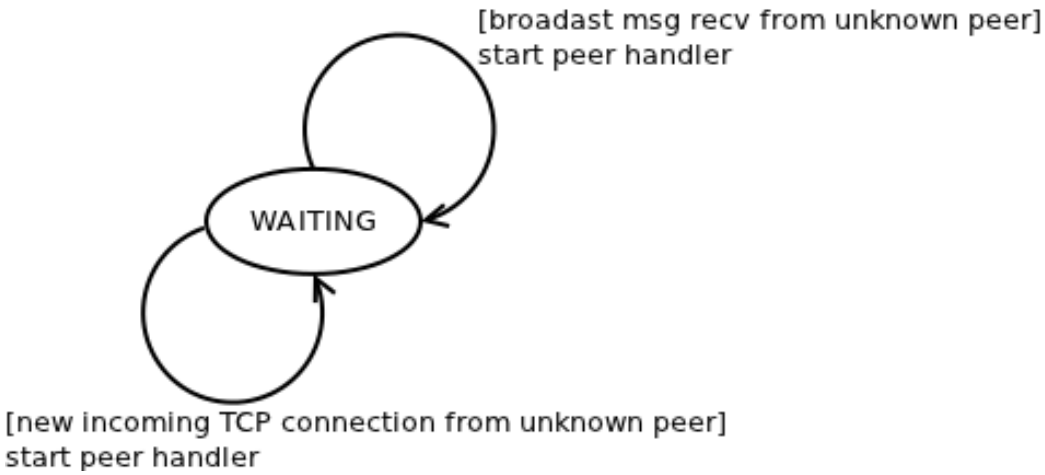

FIGURE 3.3 – Diagramme d'états modélisant le comportement du client P2P lors de la réception d'un message broadcast et lors de l'acceptation d'une connexion TCP.

son départ aux pairs, *e.g.* à cause d'une déconnexion du lien physique. Pour cela un mécanisme de *keep alive* doit être utilisé. Il faut que chaque pair envoie de manière régulière un message sur chaque connexion ouverte, même en cas d'inactivité de la part de l'utilisateur. Lorsqu'un pair n'envoie plus de messages keep alive pendant un certain laps de temps, il faut que les autres pairs ferment la connexion TCP avec ce pair et le considèrent comme ayant quitté le réseau. Si la connexion physique est rétablie, le pair *disparu* peut alors rejoindre à nouveau le réseau grâce au mécanisme de broadcast.

Authentification des pairs

Afin d'assurer que le contenu du presse papier reste confidentiel à son utilisateur et qu'aucune personne présente sur le réseau ne soit capable de modifier son contenu, un système d'authentification doit être mis en place. Celui-ci permet

de s'assurer qu'aucun utilisateur non désiré ne puisse accéder au presse-papier.

Ce système d'authentification est basé sur le protocole de Needham-Schroeder [**1978Needham**] permettant l'échange sécurisé de clés de chiffrement en utilisant une autorité de confiance. Ici cependant l'autorité de confiance est l'utilisateur ayant au préalable configuré le logiciel. Pour cela lors de l'ouverture d'une connexion TCP, chaque pair envoie un message de type JOIN avec son nom de pair et un *nonce* chiffré. C'est un nombre généré de manière pseudo-aléatoire que chaque pair devra renvoyer incrémenté de un. Ceci permet d'avoir un nombre utilisé une seule fois et d'empêcher un utilisateur étranger de rejoindre le réseau de pairs en réutilisant un message qui aurait déjà été utilisé. Le comportement d'un pair ayant été contacté par un autre pair est illustré par un diagramme d'états dans la figure **??**. Le comportement pour le pair ayant initié le contact est le même.

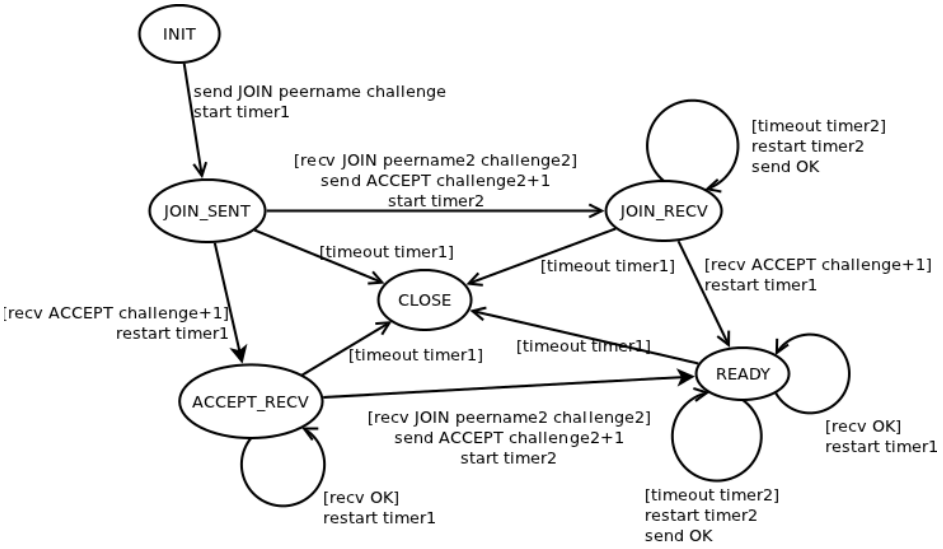

FIGURE 3.4 – Diagramme d'états illustrant l'authentification des pairs lors de l'ouverture d'une session TCP. Le noeud INIT représente l'ouverture de la session et les autres noeuds représentent différents états. Les transitions sont déclenchées si la condition entre crochets et les actions présentes ensuites sont exécutées. Ce diagramme définit aussi bien le comportement du pair ayant initié la connexion TCP que celui du pair l'ayant acceptée.

Envoi du presse-papier

Le presse-papier est envoyé lorsqu'un client local notifie le client P2P qu'un copier a été effectué. Afin d'éviter que les keep alive ne soient plus transmis lors de l'envoi du presse-papier contenant une donnée particulièrement longue, la possibilité de fragmenter le presse-papier est permise dans le protocole mis en place (*cf.* page **??**).

3.2.2 Client local

Le client P2P est le *front-end* avec le réseau, il est chargé de communiquer avec les pairs du réseau afin de les découvrir et de savoir lequel détient le presse-papier. Le front-end avec l'utilisateur est en revanche le client local. Celui-ci tourne sur le même ordinateur que le client P2P, se charge d'envoyer le contenu du presse-papier lors d'une copie de l'utilisateur, et lui demande le contenu du presse-papier lorsque que cet utilisateur effectue un coller. De même le client P2P envoie le contenu du presse-papier au client local si ce dernier a envoyé une requête pour l'obtenir ou si un autre client local a changé le contenu du presse-papier.

Cependant, contrairement au client P2P, il peut y avoir plusieurs clients locaux tournant sur le même ordinateur. Chacun d'entre eux étant destiné à un environnement précis. Dans ce projet, seront développés deux types de client différents :

- Un client tournant en tâche de fond (comme *daemon*) et synchronisant le contenu du presse-papier de X Window.
- Un client gérant le copier/coller dans un terminal. L'intérêt de ce client est d'envoyer des résultats de commandes Unix directement via un pipe, ceci étant plus intuitif que la copie via le système de fenêtrage lorsque l'on travaille en console. Il est composé de deux logiciels :
  - Une commande permettant de copier une donnée à partir de l'entrée standard du shell.
  - Une commande permettant de coller une donnée sur la sortie standard du shell.

3.3 Protocoles

Trois protocoles sont à définir afin de faire fonctionner l'application correctement. Le premier est celui permettant de découvrir des pairs grâce au broadcast. Le second est celui utilisé sur le réseau par les clients P2P afin de communiquer entre eux. Le dernier est utilisé entre le client P2P et les clients locaux afin de se notifier mutuellement des changements dans le presse papier. De même ils doivent prendre en compte l'aspect sécurité, en proposant un moyen d'identification. Ces trois protocoles seront appelés respectivement protocole broadcast, protocole P2P et protocole local.

Les protocoles sont caractérisés par un ensemble de types de message. Ceux-ci sont décrits en utilisant la syntaxe suivante :

```
MSG <SP> <VAR1> <SP> <VAR2> ... <SP> <VARN> <CRLF>
```

MSG définit le type du message. Celui-ci contient N variables <VAR1>, <VAR2>, . . ., <VARN>. Chacune est séparée par un espace (<SP>) et le message se termine

par un retour chariot (<CRLF>). Chaque variable est ensuite décrite et le type de messages pouvant être reçu comme réponse est décrit.

3.3.1 Protocole broadcast

JOIN

Un message de type JOIN envoyé en broadcast permet d'informer les autres pairs de l'existence de ce pair ci.

`JOIN <SP> <NAME> <SP> <GROUP> <CRLF>`

Variable NAME : identifiant du pair envoyant le message.

Variable GROUP : nom du groupe de pairs défini par l'utilisateur lors de la configuration du logiciel.

Réponse : si le pair n'est pas déjà présent dans la table des pairs, il faut ouvrir une connexion TCP avec celui-ci en utilisant le protocole P2P.

3.3.2 Protocole P2P

JOIN

Un message de type JOIN sert à authentifier un pair.

`JOIN <SP> <NAME> <SP> <CHALLENGE> <CRLF>`

Variable NAME : identifiant du pair envoyant le message.

Variable CHALLENGE : un nombre généré aléatoirement utilisé comme challenge.

Réponse : un message ACCEPT où le challenge est incrémenté de un.

ACCEPT

Un message de type ACCEPT sert de réponse à un message JOIN.

`ACCEPT <SP> <CHALLENGE> <CRLF>`

Variable CHALLENGE : le challenge reçu dans le message JOIN

Réponse : Si le challenge ne correspond pas (*i.e.* n'a pas été incrémenté de un et l'authentification a échoué), alors un KO est envoyé pour fermer la connexion.

KO

Ce type de message permet de signaler un refus ou une erreur et de fermer la connexion.

`KO <SP> <ERRNO> <CRLF>`

Variable ERRNO : code d'erreur pouvant être :

- **0** fermeture de la connexion.
- **1** fermeture due à un timeout.
- **2** message ACCEPT non valide, échec de l'authentification.
- **3** type de données du message DATA inconnu. Pas de fermeture de la session, le contenu du presse-papier est simplement ignoré par le pair.

OK

Ce type de message est envoyé à intervalle régulier afin de servir de mécanisme de keep alive.

`OK <CRLF>`

DATA

Un message de type DATA permet d'envoyer le contenu du presse-papier. Le contenu presse-papier peut être fragmenté et être envoyé dans plusieurs messages DATA. Il n'est cependant pas nécessaire de numéroter les messages dans le cas de données fragmentées, TCP permettant de s'assurer que les messages sont reçus dans l'ordre dans lequel ils ont été envoyés. De même l'implémentation du protocole doit s'assurer qu'un pair n'envoie pas simultanément deux données à un même autre pair.

```
DATA <SP> <TYPE> <SP> <MORE> <SP> <LENGTH> <SP>
  <CONTENT> <CRLF>
```

Variable TYPE : cette variable permet de préciser le type de donnée et ainsi d'étendre facilement le protocole afin de supporter d'autres types de données. Dans ce cas il faudra préciser comment ce type de données est encodé.

- **0** texte.

Variable MORE : vaut 1 s'il y a encore des données fragmentées qui suivent, 0 sinon. Si la variable vaut 1, alors il faut bufferiser le contenu du message jusqu'à ce qu'à recevoir un message DATA avec cette variable égale à 0.

Variable LENGTH : longueur de la donnée en bytes.

Variable CONTENT : contenu de la donnée dont la longueur doit vérifier la variable LENGTH.

3.3.3 Protocole local

GET

Ce type de message est envoyé par le client local pour demander au client P2P d'envoyer le contenu du presse-papier.

```
GET <CRLF>
```

DATA

Un message de type DATA permet d'envoyer le contenu du presse-papier. Un tel message peut aussi bien être envoyé par le client P2P que par le client local.

```
DATA <SP> <TYPE> <SP> <LENGTH> <SP> <CONTENT> <CRLF>
```

Variable TYPE : cette variable permet de préciser le type de donnée et ainsi d'étendre facilement le protocole afin de supporter d'autres types de données. Dans ce cas il faudra préciser comment ce type de données est encodé.

0 texte.

Variable LENGTH : longueur de la donnée en bytes.

Variable CONTENT : contenu de la donnée dont la longueur doit vérifier la variable LENGTH.

Chapitre 4

Implémentation de la solution proposée

Ce chapitre justifie premièrement les choix fait avant et pendant l'implémentation de de l'architecture baptisée *Clipsync*. Ensuite une présentation de l'utilisation des logiciels produits est faite. Après des vérifications de performances sont effectuées sur ces logiciels et enfin les faiblesses du logiciel sont mises en avant afin de déterminer les éléments prioritaires à développer lors de l'évolution future du logiciel.

4.1 Choix des outils

Le langage de programmation C++ a été choisi afin d'implémenter Clipsync, celui-ci permettant d'avoir un code compilé nativement et ainsi optimisé pour la machine hôte. Bien que ceci soit une source d'erreurs, il permet aussi de gérer la mémoire manuellement, ce qui implique un gain de performances. En revanche les client locaux, à savoir GTKCLip et Shellclip, ont eux été implémentés en Python. Ce langage de haut niveau en comparaison au C++ offre la possibilité d'écrire ces deux clients sous forme de scripts de taille réduite. La perte de performance par rapport au C++ est peu importante vu que ces scripts utilisent un protocole de communication simple et servent uniquement d'interface entre l'utilisateur et le client P2P.

La librairie POCO[**poco**] a été utilisée principalement comme bibliothèque pour la manipulation de sockets portables. Ceci permet d'avoir une compatibilité théorique avec Microsoft Windows, bien que celle-ci n'ait pas été testée. De plus POCO offrant un grand nombre de services, la librairie a aussi été utilisée pour la gestion des threads et des timers, pour la configuration du logiciel grâce à un fichier XML, pour la protection de zones critiques avec mutex, pour la génération de nombres pseudo-aléatoires et pour la cryptographie. L'algorithme de chiffrement utilisé est l'*AES*[**daemen2002the**] (Advanced Encryption Standard, aussi connu sous le nom de *Rijndael*). Étant l'algorithme choisi par le gouverne-

ment américain, cet algorithme utilisant une clé de 256 bits permet un niveau de sécurité très largement acceptable.

Afin d'éviter les erreurs en C++, et en particulier pour détecter les fuites de mémoire, Valgrind[**valgrind**] a été utilisé. Celui-ci est à la fois un debugger, un *memory-checker* et un *profiler*. Celui-ci permet de debugger le programme pendant le développement et est utilisé dans la suite de ce chapitre afin d'analyser les performances du logiciel et pour s'assurer que le programme n'est pas victime de fuites de mémoires. Bazaar[**bzr**] a été utilisé comme gestionnaire de versions décentralisé et Launchpad[**launchpad**] a été choisi comme forge sur laquelle déposer le logiciel produit. Ce dernier est donc disponible sous license GPL à l'adresse `https://launchpad.net/clipsync`.

4.2 Utilisation des logiciels

Avant de pouvoir récupérer, compiler et utiliser le logiciel, il est nécessaire d'avoir les librairies et outils suivants :

- Bazaar : `http://bazaar.canonical.com/`
- GCC : `http://gcc.gnu.org/`
- GNU Make : `http://www.gnu.org/software/make/`
- POCO version 1.3.6 minimum : `http://pocoproject.org/`
- Python 2.6 ou 2.7 : `http://www.python.org/`
- PyGTK et GTK version 2.1 ou supérieur : `http://www.pygtk.org/`
- Doxygen peut être optionnellement installé pour générer la documentation du projet : `http://www.stack.nl/~dimitri/doxygen/`

Tous ces logiciels peuvent êtres installés sous une distribution GNU/Linux basée sur une Debian récente grâce à la commande :

```
sudo apt-get install bzr gcc make libpoco-dev
  python-gtk2 doxygen
```

Le code source peut alors être récupéré grâce à la commande :

```
bzr branch lp:clipsync
```

Clipsync peut alors être compilé grâce à GNU Make dans le dossier *src/clipsync*. L'exécutable produit peut alors être lancé en passant en paramètre un nom de fichier XML servant à configurer le logiciel. Voici la liste des balises et leur description :

- *net_frontend* :
  - *interface* : interface réseau utilisée pour les communications réseau
  - *use_ipv6* : booléen servant à indiquer l'utilisation d'IPv6. Cette fonctionnalité n'est cependant pas disponible pour le moment et l'option a seulement été ajoutée pour l'évolution future du logiciel.

  - *port* : port TCP utilisé pour écouter les connexions TCP entrantes. C'est sur ce port que le pair sera contacté par les autres pairs du réseau.
  - *bcast_port* : port UDP utilisé pour l'envoi et la réception de messages broadcast.
  - *bcast_interval* : intervalle en millisecondes utilisé pour l'envoi de messages broadcast.
  - *peer_name* : nom servant à identifier ce pair sur le réseau et devant être unique pour ce pair.
  - *group* : nom du groupe de pairs. Chaque pair du réseau devra posséder le même nom de groupe dans sa configuration.
  - *passphrase* et *salt* : mot de passe et sel [1] servant à générer la clé pour le chiffrement par AES. Ces deux valeurs doivent être identiques sur chaque pair. Il est possible de supprimer ces deux valeurs, de lancer le logiciel qui va générer des valeurs aléatoires qui peuvent ensuite être copiées sur les autres pairs.
  - *keepalive_delay* : délai en millisecondes après lequel la connexion est fermée si aucun message OK n'est reçu.
  - *keepalive_interval* : interval en millisecondes utilisé pour l'envoi de messages OK.
  - *verbose*, *verbose_bcast*, *verbose_peer* : booléens permettant d'activer ou de désactiver le mode verbeux [2] de Clipsync.
- *local_frontend* :
  - *local_port* : port TCP en local utilisé pour communiquer avec les clients locaux.
  - *verbose* : booléen activant le mode verbeux pour l'interface avec les clients locaux.

Le script *gen_config.py* permet de générer un fichier de configuration en passant en argument dans l'ordre : le nom du pair, le nom du groupe de pairs, l'interface réseau à utiliser et optionnellement : le port, le port broadcast, le port local, la longueur du mot de passe à générer et la longueur du sel à générer.

L'utilisation de Shellclip et GTKClip est possible en exécutant les scripts Python et en passant en paramètre le port TCP local permettant de contacter Clipsync.

4.3 Vérifications de performances

Valgrind, utilisé pendant le développement de Clipsync comme debugger, offre aussi la possibilité d'être utilisé comme profiler. Afin d'analyser les perfor-

1. Le salage en cryptographie permet de renforcer la sécurité d'un algorithme de chiffrement[**schneier-crypto**]

2. Le mode verbeux permet d'afficher sur la sortie standard des informations sur l'activité du logiciel.

mances, il a d'abord été utilisé en tant que vérificateur de mémoire pour s'assurer que le programme n'est pas sujet à des fuites de mémoires. Ceci permet de s'assurer que le programme ne consomme pas, en théorie, de plus en plus de mémoire lors d'une longue utilisation. Dans un second temps, ceci est vérifié en pratique grâce à l'outil Massif de Valgrind, celui-ci permettant d'analyser la taille de la pile d'appel tout au long de l'exécution du programme.

Pour cela deux instances de Clipsync ont été lancées pendant une durée de 6h30. Durant cette exécution, un script envoyant des données aléatoires à Clipsync via Shellclip a été utilisé. Il se connecte régulièrement, avec un intervalle variant entre une et dix secondes, à une des deux instances et envoie un contenu aléatoire (de taille variant entre 10 et 1000 bytes).

La première instance a été lancée en utilisant l'outil Memcheck de Valgrind dont le résultat est présent dans la figure **??**. Celui-ci indique qu'aucun byte n'a été perdu au cours de l'exécution. Notons que les 14914 bytes, qui ont été indirectement perdus, sont dûs au fait que le programme ne s'est pas terminé « naturellement », celui-ci tournant dans une boucle infinie. La seconde instance a été lancée en utilisant Massif et le résultat, montrant une utilisation de la mémoire constante ne dépassant pas 191500 bytes est présent sur la figure **??**.

```
==23131== Memcheck, a memory error detector
==23131== Copyright (C) 2002-2010, and GNU GPL'd, by Julian Seward et al.
==23131== Using Valgrind-3.6.1 and LibVEX; rerun with -h for copyright info
==23131== Command: ./clipsync.exe conf.xml
==23131==
Clipsync started
Peer connected
Accept sent to peer
Accept from peer clover2 verified
Peer clover2 verified.
==23131==
==23131== HEAP SUMMARY:
==23131==     in use at exit: 131,614 bytes in 2,879 blocks
==23131==   total heap usage: 995,956 allocs, 993,077 frees, 857,337,345 bytes
  allocated
==23131==
==23131== LEAK SUMMARY:
==23131==    definitely lost: 0 bytes in 0 blocks
==23131==    indirectly lost: 0 bytes in 0 blocks
==23131==      possibly lost: 14,914 bytes in 341 blocks
==23131==    still reachable: 116,700 bytes in 2,538 blocks
==23131==         suppressed: 0 bytes in 0 blocks
==23131== Rerun with --leak-check=full to see details of leaked memory
==23131==
==23131== For counts of detected and suppressed errors, rerun with: -v
==23131== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 45 from 8)
```

FIGURE 4.1 – Résultats de Valgrind avec Memcheck sur une durée de 6h30.

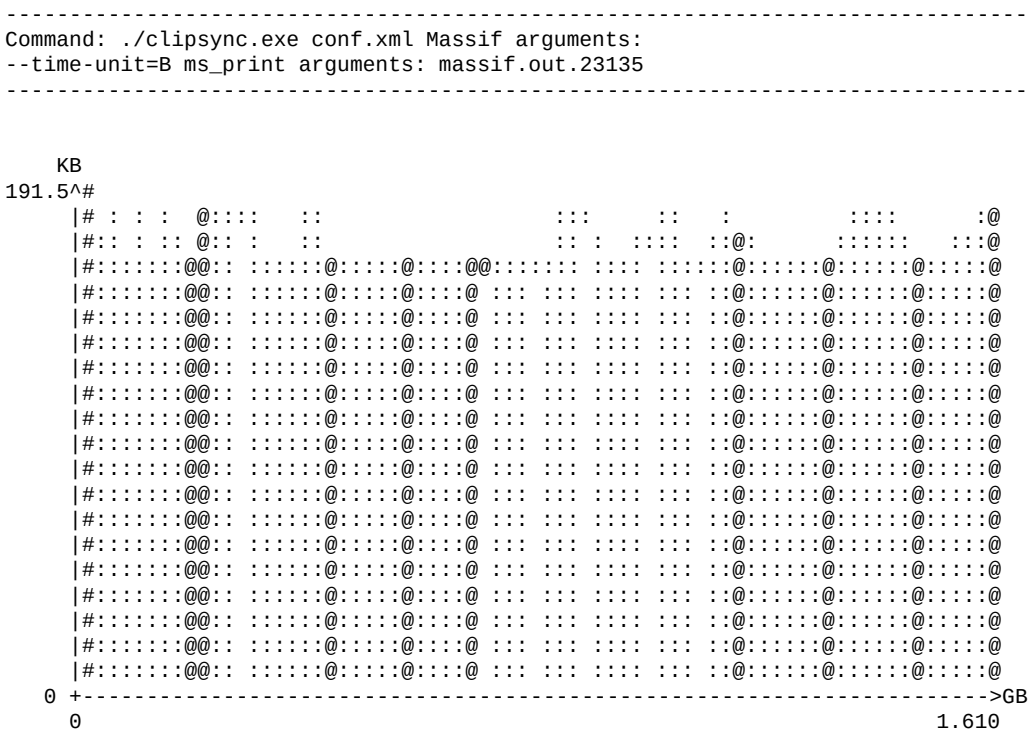

FIGURE 4.2 – Résultats de Valgrind avec Massif sur une durée de 6h30.

4.4 Analyse des fonctionnalités présentes et futures

Le logiciel produit permet de synchroniser du texte présent dans le presse-papier. D'autres fonctionnalités plus avancées avaient été identifiées au début de ce rapport comme la gestion d'autres types de données comme les images ainsi que la gestion de l'historique.

La première de ces deux fonctionnalités, qui peut par exemple servir à synchroniser des morceaux d'images entre deux éditeurs graphiques, est possible grâce au champ type présent dans les messages DATA. Cette intégration nécessite cependant des modifications du code de Clipsync, en effet aucun mécanisme n'a été mis en place pour stocker en mémoire un presse-papier différent d'une chaîne de caractères.

Une autre fonctionnalité découlant du CCMP d'images est la compression du presse-papier. A partir du moment où des images sont transférées sur le réseau, la taille des données envoyées sur le réseau augmente. Il devient alors intéressant de compresser le flux de données afin d'économiser de la bande passante. Ces deux fonctionnalités seront (après la correction de bugs découverts) la première priorité pour l'évolution future du logiciel.

En ce qui concerne la gestion de l'historique du presse-papier, cette fonc-

tionnalité n'est pas prévue afin de respecter au mieux la philosophie UNIX et n'avoir qu'un seul outil par fonctionnalité. Ainsi Clipsync s'occupe de la synchronisation du presse-papier, mais n'a pas pour but de synchroniser l'historique. De plus il existe déjà un grand nombre d'outils gérant l'historique du presse-papier et ceux-ci sont théoriquement compatibles avec Clipsync [3]. Dés lors il est possible de gérer l'historique du presse-papier en installant un tel outil sur chaque machine et d'utiliser Clipsync uniquement pour synchroniser la donnée se situant réellementt dans le presse-papier du système.

Une autre amélioration à prendre en compte assez rapidement est le support de l'IPv6. En effet le fichier configuration permet d'activer IPv6 mais cette option ne fonctionne pas. Ce problème vient du fait que l'adresse de broadcast est récupérée via POCO à partir du nom de l'interface réseau fournie dans le fichier de configuration. Cependant, une interface n'ayant pas d'adresse de broadcast en IPv6, Clipsync sera incapable de trouver cette adresse de broadcast et ne peut donc fonctionner. Il sera donc nécessaire d'adapter le logiciel en utilisant par exemple plutôt le multicast que le broadcast.

Bien que la compression et le support des images soient deux fonctionnalités pouvant améliorer Clipsync et lui donner une plus value importante, le passage à l'IPv6 est cependant un problème d'actualité assez urgent. Pour cette raison le support de l'IPv6 est prioritaire en comparaison à l'ajout des deux fonctionnalités précédemment citées. Ceci permettra ainsi à Clipsync d'avoir un réel avantage technique sur Remote Clip dont le seul réel défaut est de devenir vieillissant et donc de ne pas supporter l'IPv6.

Enfin à plus long terme, il sera certainement intéressant de revoir la structure du logiciel afin qu'il puisse rester facile à maintenir. Une de ces améliorations pourrait être l'utilisation de *Zeroconf* [4] pouvant remplacer le système de broadcast mis en place en utilisant une implémentation libre telle qu'*Avahi*[**avahi**] ou *Bonjour*[**bonjour**].

3. Des tests ont été effectuées avec XFCE-Clipman et confirment cette hypothèse.

4. Ensemble de protocoles dont le but est de fournir des services d'auto-configuration permettant entre autres de découvrir les services disponibles sur un réseau local.

Conclusion

Finalement, après avoir examiné le problème, les solutions existantes pour le résoudre se sont avérées pour la plupart inefficaces. Une solution reprenant un grand nombre d'idées de The Network Clipboard et de Remote Clip a été imaginée pour ensuite être implémentée. Cette solution comprend un mécanisme de broadcast pour la découverte de pairs et un mécanisme d'authentification permettant de s'assurer que le presse-papier ne pourra ni être altéré, ni être lu par un utilisateur du réseau local étranger au groupe d'ordinateurs entre lesquels l'utilisateur souhaite synchroniser le presse-papier.

Clipsync, le logiciel produit, a ainsi pu être publié comme logiciel libre sur Launchpad dans le but de faciliter sa distribution ainsi que son évolution. Pour plannifier cette évolution, les fonctionnalités manquantes ont été mises en avant. Le passage à l'IPv6 étant un problème d'actualité urgent, l'effort sera avant tout concentré sur son support dans Clipsync ainsi que sur la découverte et la correction de bugs. Ceci permettra de stabiliser le logiciel avant l'ajout de nouvelles fonctionnalités.

Ensuite l'ajout de nouvelles fonctionnalités pour l'utilisateur telles que l'implémentation du copier/coller d'images et la compression du presse-papier, seront la seconde priorité des développements futurs du logiciel. A plus long terme, des mécanismes d'auto-configuration standardisés tels qu'Avahi seront utilisés pour remplacer le mécanisme de broadcast.

Annexe A

Codes sources utilisés

Afin de comprendre comment fonctionnent certains logiciels étudiés dans l'étude de faisabilité, certains codes sources disponibles en ligne ont été utilisés. Ceux-ci étant disponibles sous licences libres (GPL et MIT), ils sont non seulement disponibles en ligne mais peuvent aussi être réutilisés librement.

Même si ces codes ne seront pas réutilisés pour des raisons évidentes (utilisation d'un environnement de développement différent), certains principes ont été compris grâce à ceux-ci. De même les principes de base utilisé dans le projet s'inspirent largement de ces logiciels, c'est pour cette raison qu'une annexe leur est consacrée.

ClipboardMultiSharer

Le code source Java de l'application sous licence GPL [1] a été parcouru afin de comprendre comment fonctionnait le logiciel, entre autres afin de voir comment il était possible d'accéder au presse-papier en Java. La révision consultée était la 16 datant du 10 juillet 2010.

Clipboard Share

N'ayant pas de connaissances en C# et en MS .NET, je n'ai ni lu, ni utilisé le code de ce logiciel sous license MIT [2].

1. `http://sourceforge.net/projects/clipboardmshare/develop`
2. `http://clipboardshare.codeplex.com/SourceControl/list/changesets`

The Network Clipboard

Le code source sous licence GPL [3] a été consulté dans le but d'examiner le protocole utilisé. La révision consultée était la 88.

Remote Clip

Le code source sous licence GPL est disponible avec la dernière version du logiciel [4]. Celui-ci a été consulté dans le but d'étudier le protocole réseau utilisé.

Gestionnaires de presse-papier

Le code source de plusieurs gestionnaires de presse-papier ont permis d'aider à la compréhension du fonctionnement du presse-papier via l'utilisation de librairies annexes telles que GTK et Qt. Ces logiciels sont Glipper 2.1 [5], Klipper [6] et parcellite [7].

3. `http://sourceforge.net/projects/netclipboard/develop`
4. `http://www.cs.cmu.edu/~rcm/RemoteClip/RemoteClip-3.1.zip`
5. Version 2.1 : `https://launchpad.net/glipper`
6. Version présente dans KDE 4.6 (`http://www.kde.org`)
7. version 1.0.1 : `http://parcellite.sourceforge.net`

Annexe B

Principe KISS

Le principe *KISS* (Keep It Simple, Stupid !) [**wiki:kiss**] est un principe prônant la simplicité, tout particulièrement dans le monde de la conception. Il est utilisé dans le développement logiciel dans le but d'exprimer le fait que la conception doit être simple et éviter toute complexité inutile.

Celui-ci est à la base même de la philosophie Unix, en effet celle-ci prône l'utilisation de petits utilitaires simples ayant chacun une fonctionnalité bien précise. Douglas McIlroy, inventeur du pipe UNIX, a dit[**quartercentury-unix**] :

> This is the Unix philosophy : Write programs that do one thing and do it well. Write programs to work together. Write programs to handle text streams, because that is a universal interface.

Le projet a donc été développé en utilisant une approche orientée composant[**wiki:poc**], en fournissant de petits logiciels communiquant au travers de sockets de manière textuelle.

Plusieurs auteurs [**Brooks1995** ; **Raymond2001**] prônent cette philosophie sans pour autant y faire référence explicitement. *e.g.* Brooks explique par exemple que le design est d'une importance capitale car il permet de comprendre plus facilement le code source, Raymond dit qu'il faut concevoir les logiciels de manière à ce qu'ils suivent la philosophie d'Unix. Il dit aussi que la perfection en conception n'est pas atteinte lorsque l'on n'a plus rien à ajouter mais plutôt lorsque l'on n'a plus rien à retirer.

Filip Hanik, un ingénieur logiciel membre de l'*Apache Foundation* parle aussi du principe KISS et donne des conseils pour appliquer ces principes en Java (mais ceux-ci sont applicables dans tous les langages de programmation) [**fhanikKISS**].