BOUBAKER Monia
LELION Ludovic
PIERRON Aurélien

Février 2005
IR3

Exposé réseau :

Peer-to-Peer – KaZaa – FreeNet - JXTA

Professeur responsable :

Etienne DURIS

Sommaire

I Le Peer to Peer

I.1 Historique et présentation du Peer To Peer

A l'origine, l'informatique a été conçue pour réaliser des traitements mathématiques complexes. A ce moment le concept de communication n'existait pas encore entre différents ordinateurs.

Rapidement, le concept de terminaux communiquant avec un ordinateur central, s'est développé. C'est à partir de ce concept qu'est né celui du « client - serveur ».

En 1969, l'ARPA met au point le premier réseau mondial : ARPANET. Suite à diverses modifications dans les années 70 et 80, ce réseau devient le réseau INTERNET au début des années 90.

Aujourd'hui, ce réseau mondial fonctionne intégralement sur le concept de client – serveur.

I.1.1 Le concept de client – serveur

Avec ce concept, les données partagées sur le réseau sont situées sur un ordinateur et il faut donc s'y connecter pour récupérer les données. Le serveur est ainsi le composant central de l'échange de données.

Les utilisateurs voulant télécharger un fichier à partir d'un de ces serveurs doivent se partager la bande passante de celui-ci. Plus le fichier est populaire, plus la charge est importante sur le serveur. Et à partir d'un certain point, ce fichier devient de moins en moins accessible pour les utilisateurs. Le distributeur peut évidemment investir dans un/des serveur(s) plus puissant(s) et/ou dans une augmentation de sa bande passante, mais ces investissements sont coûteux et la limite d'accessibilité n'en serait repoussée que de peu.

Aujourd'hui de plus en plus de personnes utilisent INTERNET, et la taille des fichiers échangés est de plus en plus importante. On s'aperçoit donc que ce type d'architecture commence à atteindre ses limites pour le partage de fichiers de tailles importantes.

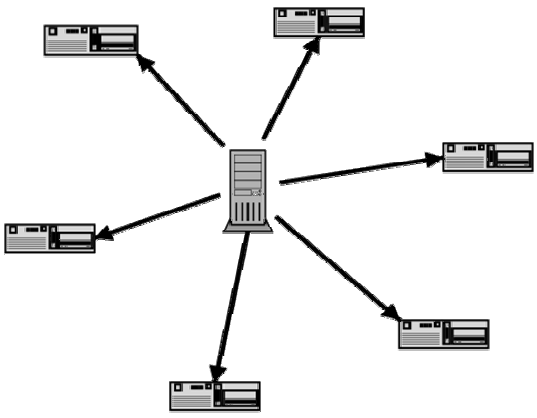


Schéma 1 : Architecture client - serveur

I.1.2 Evolution vers le Peer To Peer

Fin 1998, Shawn Fanning, un étudiant américain passionné d'informatique alors âgé de 19 ans vient bouleverser le monde bien établi du client/serveur. Il décide de quitter l'Université et se lance dans l'écriture d'un logiciel permettant l'échange de fichiers musicaux.

La raison d'être de ce logiciel repose sur le constat suivant : rechercher des MP3 sur les moteurs de recherche habituels conduit à une perte de temps énorme et les réponses sont souvent inappropriées. De plus les temps de transfert sont assez longs suite à la taille conséquente des fichiers et à l'encombrement des serveurs.

Après quelques mois de travail acharné, une première version du logiciel est disponible. Fanning décide de tester une première version le 1er juin 1999 et appelle son logiciel Napster. Le logiciel qui ne devait être testé que par quelques-uns de ses amis remporte un succès des plus rapides. Il conquiert notamment les universités.

En septembre 2000, Napster atteint un nombre de téléchargement record. 1,39 milliard serait le nombre de chansons échangées par ses utilisateurs (source : Webnoize, Cabinet d'études américain).

C'est à partir de ce moment que l'architecture « Peer To Peer » s'est démocratisée et que des logiciels utilisant cette architecture commencent à émerger.

I.1.3 Description du Peer to Peer

« Peer To Peer » signifie littéralement « pair à pair ». Ce concept introduit ainsi une relation d'égal à égal entre deux ordinateurs. L'informatique « pair à pair » se définit comme le partage des ressources et des services par échange direct entre systèmes. Ces échanges peuvent porter sur des informations de différents types, des cycles de traitement, de la mémoire ou encore du stockage sur disque de fichiers.

Contrairement au modèle client/serveur, chaque système est une entité réseau complète qui remplit à la fois le rôle de serveur et celui de client. Le Peer To Peer désigne donc une classe d'applications qui tirent partie des ressources matérielles ou humaines qui sont disponibles sur le réseau Internet.

I.1.4 Différents types d'applications Peer To Peer

Il existe aujourd'hui différents types d'applications utilisant le concept du Peer To Peer.

- calcul distribué (ex : le projet SETI qui consiste à utiliser une partie des ressources de chaque ordinateur connecté sur INTERNET pour réaliser des calculs complexes) ;
- échange de fichiers (généralement des fichiers multimédia (musique, vidéo...)) ;
  - Napster ;
  - Gnutella ;
  - Emule ;

  - Kazaa ;
  - Bittorrent.
- stockage distribué (FreeNet) ;
- téléphonie (Skype).

I.1.5 Différentes architectures Peer To Peer

I.1.5.1 Architecture centralisée

Théoriquement, un tel dispositif représente actuellement la solution la plus confortable pour échanger des fichiers de taille importante dans une communauté (musique, films...). Mais en réalité, ce type d'architecture exige un tel investissement en ressource que les services restent rarement de bonne qualité (lenteur, disponibilité...). Soit, ils sont saturés ; soit, ils sont limités en termes d'utilisateurs simultanés autorisés.

Concrètement, dans toute architecture centralisée, un dispositif exclusivement serveur se charge de mettre en relation directe tous les utilisateurs connectés. L'intérêt de cette technique réside dans l'indexation centralisée de tous les répertoires et intitulés de fichiers partagés par les abonnés sur le réseau. En général, la mise à jour de cette base s'effectue en temps réel, dès qu'un nouvel utilisateur se connecte ou quitte le service.

Au niveau des clients, la recherche se fait à l'aide d'un moteur de recherche classique. A partir d'une requête, le serveur renvoie une liste d'utilisateurs actuellement connectés au service et dont les fichiers partagés correspondent au terme recherché. Le transfert des fichiers se fera entre les utilisateurs et non par le serveur. Dans ces conditions, à aucun moment les fichiers se retrouvent stockés sur le serveur central.

Cette architecture est assez proche de celle du « client serveur », mais la grosse différence qui la distingue de cette dernière consiste en le fait que seuls les informations concernant les utilisateurs sont stockées sur le serveur. Les fichiers étant répartis sur chacun des utilisateurs et téléchargés par connexion directe entre chacun d'eux.

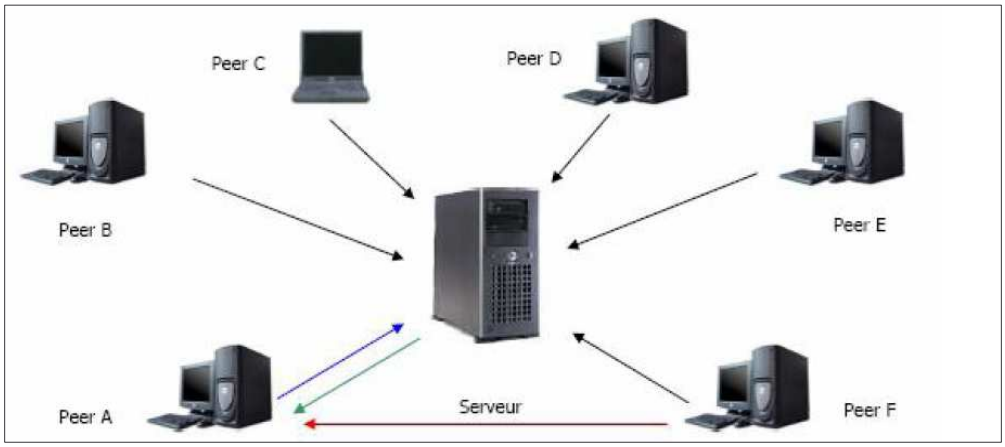


Schéma 2 : Architecture centralisée

Ce type d'architecture est notamment utilisé par les systèmes Napster et Edonkey – Emule.

Les avantages de cette architecture concernent notamment le système de recherche dans le cas où de nombreux utilisateurs partagent des données et sont connectés au serveur.

Les principaux défauts de cette architecture sont :

- la sécurité, une architecture de ce type s'avère particulièrement vulnérable. Elle ne propose qu'une seule porte d'entrée, son serveur centralisé, ce qui constitue le talon d'Achille de tout le réseau. Il suffirait effectivement de bloquer ce serveur pour déconnecter tous les utilisateurs et stopper le fonctionnement de l'ensemble du réseau ;
- Le manque d'anonymat. Le fait de s'enregistrer sur un serveur pour pouvoir accéder aux services, ne garantit bien évidemment aucun anonymat. Le service connaît l'adresse IP de votre machine et le type de fichiers qui sont partagés.

I.1.5.2 Architecture centralisée améliorée

Pour résoudre les problèmes de robustesse et améliorer la qualité de connexion avec le serveur, le serveur central de l'architecture centralisée est remplacé par un anneau de serveur. Ceci permet d'éviter la chute du réseau si une panne se produit sur un serveur, car il y a toujours un point de connexion valide aux serveurs.

De plus l'utilisation de plusieurs serveurs permet de mieux répartir les demandes de connexions et donc de limiter la chute de bande passante. Chaque serveur peut avoir accès aux informations des clients connectés sur les autres.

L'accès aux données partagées est donc totalement transparent pour les utilisateurs.

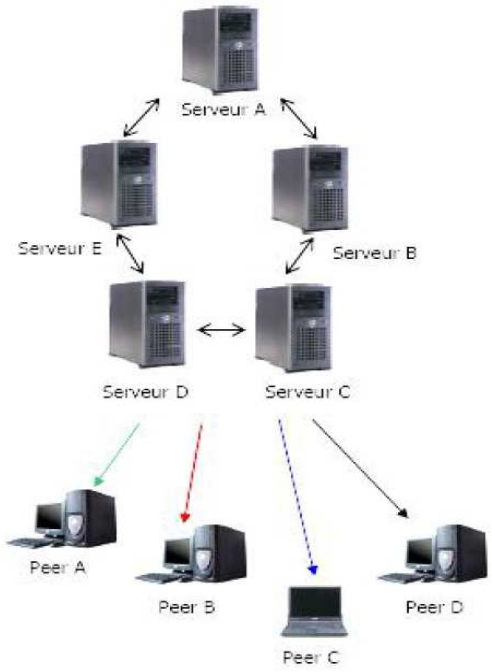


Schéma 3 : Architecture centralisée avec un anneau de serveurs

I.1.5.3 Architecture décentralisée

Nous avons vu que l'architecture centralisée pose des problèmes de sécurité, robustesse, et de limitation de la bande passante. Les problèmes sont directement issus de l'utilisation de serveurs dont le seul but est de posséder l'annuaire des clients.

Si on désire supprimer les serveurs centraux il faut donc trouver le moyen de constituer un annuaire sur chaque client, puis de les faire communiquer. C'est sur ces mécanismes que sont basés les réseaux Peer To Peer décentralisés. Il n'y a donc plus de serveurs centraux, ce sont tous les éléments du réseau qui vont jouer ce rôle. Chaque machine va jouer le rôle de serveur et de client. Chaque machine dans ses rôles est identique à une autre, c'est pour cela que l'on appelle ces types de réseaux Peer To Peer pur.

Un grand avantage de ce nouveau type de réseaux, est en théorie le total anonymat qu'il procure. En effet en évitant de communiquer avec une machine centralisant les demandes et les annuaires, on évite les problèmes de récupération des données utilisateurs.

Ce type d'architecture est utilisé dans les réseaux tels que Gnutella et FreeNet.

Les défauts de ce type d'architecture sont :

- la lenteur de la recherche. En effet, ne disposant pas de système de recherche, les recherches se font en routant de clients à clients les requêtes de recherches. Ainsi la récupération des résultats est assez longues et aléatoires (des chemins différents étant choisis à chaque requête) ;

- Nécessité de récupérer un ensemble de clients connectés sur le réseau pour pouvoir l'intégrer ;
- Diffusion en mode Broadcast des requêtes ayant comme principal inconvénient de ralentir les échanges de données entre machines.

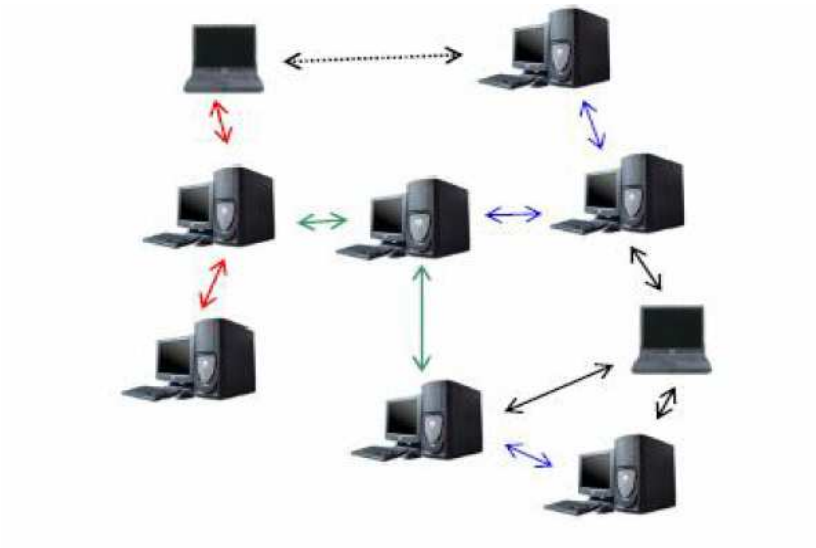


Schéma 4 : Architecture décentralisée

I.1.5.4 Architecture hybride ou décentralisée - centralisée: les réseaux à super noeuds

Le modèle super noeud a pour but d'utiliser les avantages des 2 types de réseaux (centralisé et décentralisé). En effet sa structure permet de diminuer le nombres de connexions sur chaque serveur, et ainsi d'éviter les problèmes de bandes passantes.

D'autre part le réseau de serveurs utilise un mécanisme issu des réseaux décentralisés pour tenir à jour un annuaire client et un index des fichiers à partir des informations provenant des autres serveurs. Un serveur peut donc proposer à n'importe quel client toutes les informations contenues sur le réseau.

Le réseau n'est plus pollué par les trames de Broadcast. Mais la contrepartie est que l'anonymat n'est plus assuré.

Ce type d'architecture est utilisé pour les réseaux FastTrack (KaZaa) et Gnutella 2.

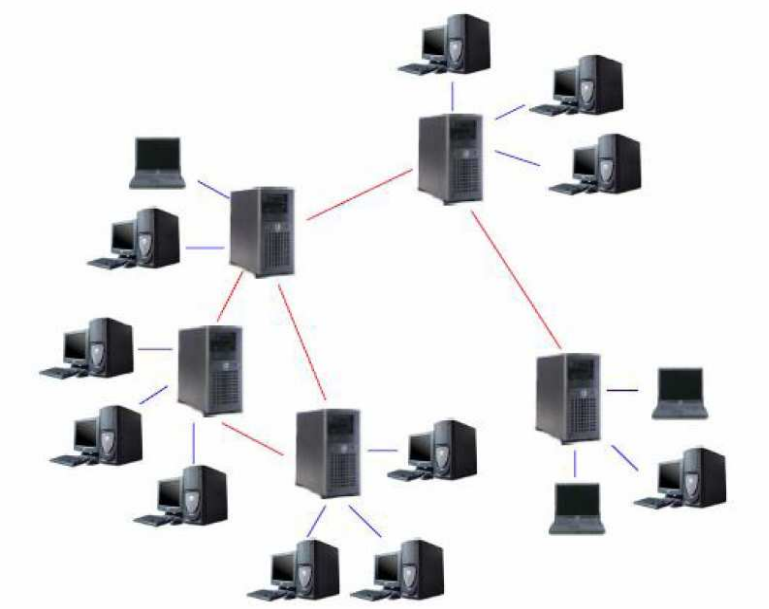


Schéma 5 : Architecture hybride avec notions de super noeuds

II Le réseau FastTrack (Kazaa)

II.1 Présentation

Le protocole réseau FastTrack, dont les clients les plus connus du grand public sont Kazaa, Morpheus, Grokster est un protocole propriétaire appartenant à la société américaine Sherman Networks (http://www.sharmannetworks.com/).

Ce protocole se base sur une architecture hybride (décentralisée – centralisée) présentée précédemment. Ce protocole permet comme les protocoles Gnutella, Emule ... de partager des fichiers multimédia (musique, films ...). Il est aujourd'hui principalement utilisé pour le téléchargement de MP3.

Ce protocole a déjà fait ses preuves aujourd'hui avec une moyenne d'un million et demi d'utilisateurs connectés.

II.2 Architecture

L'architecture de FastTrack suit un système 2-tiers dans lequel la première rangée se compose des raccordements rapides au réseau (Câble/DSL et plus rapide) et la deuxième rangée se compose des raccordements plus lents (modem 56K et plus lent).

Des clients sur la première rangée sont connus en tant que SuperNodes et les clients sur la deuxième rangée sont connus comme étant des noeuds standards. Lors du raccordement au réseau ce qui se produit est que le client décide si le noeud convient pour devenir un SuperNode ou pas en fonction des paramètres de la connexion.

Dans le cas d'un SuperNode, le client est alors relié à d'autres SuperNodes et commence à prendre des raccordements avec d'autres noeuds ordinaires.

Dans le cas d'un noeud, le client va trouver un SuperNode auquel se relier.

Ceci produit une topologie à deux niveaux dans laquelle les noeuds au centre du réseau sont plus rapides et produisent donc une épine dorsale plus fiable et plus stable. Ceci permet à plus de messages d'être conduits et permet donc une plus grande scalability.

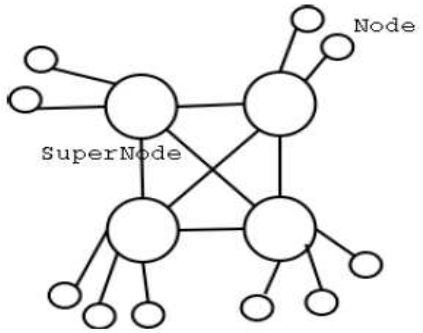


Schéma 6 : Architecture hybride de Kazaa

II.3 Cheminement des requêtes

Le cheminement sur FastTrack est accompli par des annonces entre chaque SuperNodes.

Par exemple, quand un noeud publie une demande de recherche au SuperNode auquel il est relié, celui-ci publie cette demande à tous les SuperNodes auxquels il est actuellement relié. La recherche continue de cette façon jusqu'à ce que le TTL (Time To Live) de la requête ait atteint zéro.

Il faut savoir que chaque SuperNode contient un index de tous les dossiers de ses noeuds reliés, ainsi lorsqu'une requête arrive à un SuperNode, il sait s'il doit la rediriger vers un des nœuds reliés. A chaque fois qu'un nouveau noeud se connecte à un SuperNode, il lui fournit la liste de ses fichiers partagés

Malheureusement puisque les recherches produisent des requêtes en mode Broadcast, ceci génère beaucoup de trafic entre les SuperNodes.

Cependant puisque le SuperNodes ont la garantie d'être raisonnablement rapides, ceci ne pose pas un aussi gros problème que sur le réseau Gnutella.

Le cheminement des requêtes est le même que sur le réseau Gnutella. En effet tant que le TTL de la requête n'a pas atteint 0, chaque noeud redirige la requête aux autres noeuds auxquels il est relié.

Le problème de Gnutella, dans ce type de système, est que les réponses aux requêtes passent par les mêmes noeuds qu'à l'aller. Ainsi il n'est pas garanti de recevoir l'intégralité des résultats car si un des noeuds de la chaîne se déconnecte, les résultats de cette partie ne peuvent arrivés.

Ce problème se produit moins sur FastTrack, en effet les SuperNodes étant censés être des noeuds rapides, le chemin est normalement plus fiable et plus stable.

Lors de la recherche les fichiers sont identifiés à la fois par leur nom mais surtout par leur code de hachage (hachage SHA-1). En effet il est possible d'avoir 2 fichiers identiques mais avec 2 noms différents (ex : fichierA.txt et fichier_A.txt). Dans ce cas le code de hachage sera identique.

II.4 Description du protocole

FastTrack étant un protocole propriétaire de la société Sharman Network, il n'existe pas de description publique du protocole. La description présentée correspond à un protocole récupéré par un procédé de « Reverse Engineering ». Il se peut donc qu'il ne s'agisse pas de la dernière version du protocole.

Il est important de savoir que pour tous les entiers présentés dans la description du protocole sont encodés sur 4 octets à la norme Big-Endian.

Le protocole prévoit l'utilisation de paquet TCP et UDP pour les échanges.

Description des paquets TCP :

1. Handshake request (demande d'ouverture de session)

Rand (4)	outgoing cipher seed (4)	outgoing encryption type (4)

Il s'agit du 1er paquet envoyé servant à initialiser une connexion.

2. Handshake response (réponse à l'ouverture d'une session)

incoming cipher seed (4)	incoming encryption type (4)

Il s'agit de la réponse à un paquet d'ouverture de session. A partir de ce point les requêtes sont encryptées.

3. Server Network Name (nom du réseau pour le serveur)

Nom du réseau du serveur (n)

Paquet envoyé par l'hôte auquel le client cherche à se connecter. Permet de déterminer le nom du réseau sur lequel on se connecte (« Kazaa », « Grokster », ...)

4. Client Network Name (nom du réseau pour le client)

Nom du réseau du client (n)

Paquet envoyé par le client indiquant sur quel réseau se situe le client hôte auquel le client cherche à se connecter. Si le nom de réseau ne correspond pas à celui envoyé par le serveur, la connexion échoue.

A partir de ce point la connexion est effectuée.

5. Ping Request

0x50 (1)

Ce paquet permet de tester qu'un client est toujours connecté. A ce type de paquet le client doit renvoyer un paquet « Pong »

6. Pong response

0x52 (1)

Ce paquet est envoyé suite à la réception d'un paquet « Ping »

7. Packet

0x4B (1)	Message type and length (4)

Tous les autres paquets du protocole sont englobés dans ce type de paquet. Le type du message et sa longueur sont indiqués dans un entier. Le type est codé sur le 1er octet, la longueur sur les 2 octets suivants et il y a un 0 sur le dernier octet.

Les différents types de message sont :

Type	*Longueur*	*Nom*	*Description*
0x00	8	Supernode list	Décrit les informations d'un SuperNode (IP, port). Paquet répété autant de fois qu'il y a des SuperNodes à récupérer
0x01	26	Extended SuperNode information	Pour indiquer à un client qu'on est SuperNode. Contient en plus d'autres informations sur les SuperNode
0x02	8 + n	User information	Donne des informations sur un utilisateur. n étant la longueur du nom de l'utilisateur
0x05	20 + n	Unshare file	Pour enlever le partage sur un fichier. Les 20 octets correspondent au code de hachage du fichier. Les n autres octets contiennent d'autres informations sur le fichier
0x06	9 + n	Search query	Correspond aux paquets décrivant une recherche. Ce paquet contient des informations telles que l'ID de la recherche, le nombre maximum de résultats attendus et englobe d'autres paquets contenant les termes de la recherche ainsi que les types.
0x07	10 + n	Search reply	Correspond au résultat d'une recherche. Contient l'identifiant de la recherche, le nombre de résultats et englobe d'autres paquets décrivant chacun des résultats. Chaque résultat va contenir pour les coordonnées de l'utilisateur, le fichier trouvé (nom + code de hachage) ...
0x09	12	Network stats	Permet d'obtenir des statistiques sur le réseau. Ces statistiques définissent le nombre d'utilisateurs, le nombre des fichiers, la taille totale des fichiers partagés
0x0D	22 + n	Push request	Permet de demander à un autre client sur le réseau de forcer celui ci à envoyer le fichier. Nécessaire dans le cas où un des clients de la connexion est derrière un firewall et donc ne peut pas être joint directement. Ce paquet contient les informations nécessaires pour que l'utilisateur soit contacté
0x15	6	Attempt UDP connection	Message échangé entre SuperNode. Permet à un SuperNode de demander à un autre s'il y a la possibilité d'établir une connexion UDP
0x1D	n	Network name	Indique le nom du réseau. On ne sait pas dans quel cadre ce message est utilisé, car le nom est passé lors du handshake

Type	*Longueur*	*Nom*	*Description*
0x1F	276 + n	Network update	Indique lorsqu'il y a une modification sur le réseau. Utilisé pour notifier le changement de l'encryption ....
0x20	5	Random files request	Permet de demander une liste des fichiers de l'utilisateur
0x21	18 + n	Random files	Réponse d'un utilisateur au paquet précédent. Indique ses coordonnées ainsi que la liste des tous ses fichiers et de leurs types
0x22	24 + n	Share file	Permet à un utilisateur de signaler au SuperNode, qu'il partage un fichier. Ce paquet va contenir toutes les informations concernant le fichier (type, code de hachage, taille ...)
0x2C	4	External IP	Permet de connaître l'adresse IP globale, celle qui est visible de l'extérieur (utilisé par exemple dans le cas d'une translation d'adresse NAT).

Pour les messages UDP la structure est différente. Seul le type du message est envoyé sur le 1er octet.

Les différents types de message UDP sont :

Type	*Longueur*	*Nom*	*Description*
0x27	6 + n	Node ping	Permet de pinger un noeud. Utilisé par Kazaa afin de pinger d'abord le SuperNode avant d'essayer de s'y connecter. Contient des informations sur l'encryption, le nom du réseau ...
0x28	11 + n	Node pong	Kazaa ne se connecte seulement aux SuperNodes qui renvoient ce type de pong. (0x28) et non à ceux qui renvoient le pong2 (0x29).
0x29	13 + n	Node pong 2	Voir description du paquet précédent

III Freenet

III.1 Qu'est-ce que FREENET ?

Freenet est un réseau P2P (*Peer-to-Peer*) fonctionnant au-dessus d'Internet. Contrairement aux réseaux P2P habituels, Freenet assure l'anonymat des échanges et assure une répartition intelligente des données sur le réseau. Bien qu'il soit encore en plein développement, il permet déjà aujourd'hui de consulter des sites Web sur divers sujets, de dialoguer dans des forums et d'échanger des fichiers, le tout de façon anonyme.

Open source, la définition de son protocole est basée sur les idées de Ian Clarke alors étudiant à l'Université d'Edinburgh en Ecosse. Le projet a été initialisé par Clarke en Juillet 1999 et il continue actuellement à coordonner le projet.

Freenet a beaucoup fait parler de lui en 2000, mais son excessive lenteur et de nombreux problèmes techniques ont déçu et détourné de nombreux utilisateurs potentiels à cette époque. Ce qui a contribué à faire de Freenet un réseau réservé à une élite d'utilisateurs.

Aujourd'hui Freenet a largement évolué et son utilisation a été facilitée. Même s'il reste encore peu rapide et moins simple d'utilisation que les autres logiciels P2P, Freenet continue à être amélioré chaque jour par une équipe de développeurs motivés. Le réseau offre déjà aujourd'hui une alternative pour faire face au filtrage que se préparent à mettre en place les fournisseurs d'accès sous la pression des Majors.

Freenet n'a pas pour vocation immédiate de remplacer Emule ou Kazaa par exemple. Son but principal est avant tout de fournir un moyen vraiment fiable d'échanger des informations, sans que personne ne puisse s'y opposer. Il est évidemment possible d'échanger des fichiers musicaux ou des films via Freenet, mais son but premier est surtout d'offrir une liberté d'expression plutôt textuelle, éventuellement illustrée, comme sur un site Web classique, avec des petits fichiers tels que des documents PDF dont la publication pourrait faire l'objet de censure ou de plaintes dans la vie réelle ou sur le Web.

Freenet implémente des stratégies visant à protéger l'intégrité des données, à empêcher la divulgation des activités des utilisateurs, et enfin à fournir une disponibilité redondante des données. Le système est également conçu afin de s'adapter aux habitudes d'utilisation des participants, répliquant et supprimant automatiquement des fichiers en fonction de la demande, afin de faire le meilleur usage possible du stockage disponible.

Pour résumer, Freenet a pour vocation d'être un vecteur d'informations fiable, efficace et anonyme.

III.2 Principes de fonctionnement de FREENET

III.2.1 Principes généraux

Chaque participant de Freenet fait fonctionner un noeud qui fournit au réseau un peu d'espace disque. Pour ajouter un nouveau fichier, l'utilisateur envoie au réseau un message d'insertion contenant le fichier et un identifiant global unique (clé GUID) généré indépendamment de son emplacement géographique. Le fichier est ensuite stocké sur plusieurs noeuds. Pendant sa durée de vie, un fichier peut migrer d'un nœud à un autre ou être répliqué sur plusieurs noeuds. Afin de récupérer un fichier, l'utilisateur envoie une requête contenant la clé GUID. Lorsque la requête atteint un des noeuds sur lequel le fichier est stocké, ce noeud renvoie les données au demandeur.

Freenet est particulièrement novateur car :

- Il ne repose sur aucune forme d'administration ou de contrôle centralisée ;
- Il est virtuellement impossible de retirer par la force un document de Freenet ;
- Aussi bien les auteurs que les lecteurs des informations stockées sur ce système peuvent rester anonymes s'ils le désirent ;
- Les informations peuvent être réparties sur le réseau Freenet de telle manière qu'il soit difficile de déterminer où elles sont stockées ;
- Tout le monde peut publier des informations : il est inutile d'acheter un nom de domaine, ni même de posséder une connexion internet permanente ;
- La disponibilité de l'information augmente en proportion de la demande ;
- Une information peut se déplacer d'une partie d'Internet où elle est faiblement consultée pour un autre lieu où la demande est plus grande.

Freenet permet donc de protéger l'intégrité des données, d'empêcher la divulgation des activités des utilisateurs et de fournir une disponibilité redondante des données. Il est également conçu afin de s'adapter aux habitudes d'utilisation des participants, répliquant et supprimant automatiquement des fichiers en fonction de la demande. Il fait donc un usage optimisé de l'espace de stockage disponible.

III.2.2 Principe de clé

Les clés GUID de Freenet sont calculées en utilisant le hachage sécurisé SHA-1 (Secure Hash Algorithm – codage sur 160 bits).

Le réseau emploie deux principaux types de clés : les clés de hachage des contenus (CHK), utilisées pour le stockage primaire des données et les clés signées de sous-espaces (SSK), destinées à l'utilisation humaine à plus haut niveau. Les deux types sont analogues pour les noeuds et les noms de fichiers dans un système de fichiers conventionnel.

III.2.2.1 Les clés de hachage des contenus (Content-Hash Keys ou CHK)

La CHK est une clé de bas niveau pour le stockage des données, elle est générée en analysant le contenu du fichier devant être stocké. Le processus attribue à chaque fichier un identifiant unique absolu (les collisions SHA-1 sont considérées comme étant presque impossibles) qui peut être vérifié rapidement. A la différence des URL une référence CHK pointera exactement sur le fichier attendu. Les clés CHK permettent également à des copies identiques d'un fichier insérées par différentes personnes d'être automatiquement fusionnées, car chaque utilisateur calculera la même clé pour un même fichier.

Exemple de clé :

CHK@BSI9FMrG7YpKrZXX3jm--nF6mesLAwI,7aKDjQmNNqbwIP9ah~l0fg

Pour des raisons de confort d'utilisation, les clés SSK sont le plus souvent utilisées.

Les clés sont obtenues à l'aide de la commande suivante :

```
java -cp <freenet directory>/freenet.jar freenet.client.cli.Main computechk <file>
```

III.2.2.2 Les clés signées de sous-espaces (Signed-Subspace Keys ou SSK)

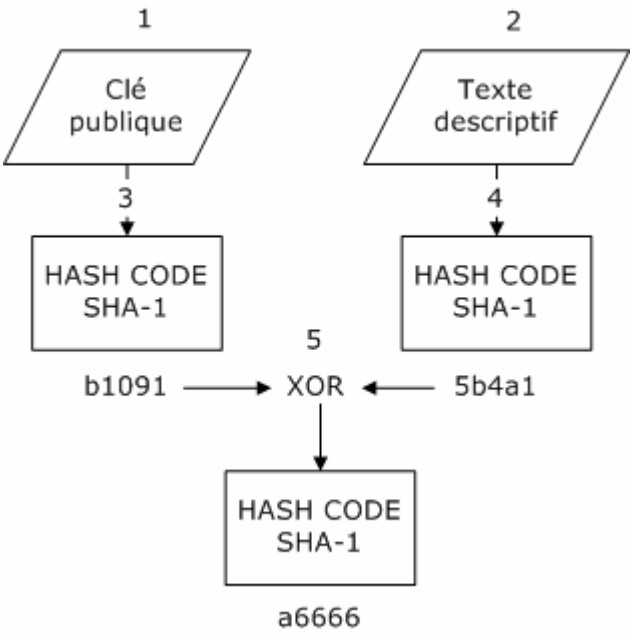
La clé SSK met en place un espace nommé que tout le monde peut lire mais que seul son auteur peut modifier.

Déroulement du calcul d'une clé SSK

Voici les étapes consistant à calculer une clé SSK :

1. Créer un sous-espace pour y placer un document en générant au hasard une paire de clés publique / privée pour l'identifier ;
2. Rédiger une courte description de ce document (5 ou 6 mots environ) ;
3. Hacher la moitié publique de la clé du sous-espace ;
4. Hacher le descriptif du sous-espace ;

5. Combiner et hacher l'ensemble pour obtenir la clé SSK.



Il est ensuite possible de vérifier l'intégrité du fichier en le signant avec la moitié privée de la clé, car chaque nœud qui manipule un fichier d'un sous-espace signé vérifie sa signature avant de l'accepter.

Seule une personne possédant le clé privée pourra modifier le contenu de cet espace de stockage. La clé publique permet quant à elle d'accéder au contenu.

Les clés SSK sont typiquement utilisées pour stocker des fichiers indirects qui ne contiennent que des pointeurs vers des CHK, au lieu de contenir directement des données. Elles permettent aux données d'êtres mises à jour tout en préservant l'intégrité référentielle. Tel qu'il est prévu dans le protocole (la fonction n'étant pas encore implémentée), pour effectuer une mise à jour, le propriétaire des données insère d'abord une nouvelle version des données qui génèreront une nouvelle clé de hachage de contenu (CHK) puisque les données du fichier ont été modifiées. Le propriétaire met ensuite à jour la SSK afin de pointer sur la nouvelle version de son fichier. Ce dernier sera accessible par la SSK et son ancienne version restera accessible par l'ancienne CHK. Dans ce cas, il est nécessaire de garder une trace de l'ancienne CHK, sinon il n'y aura plus de moyen de retrouver l'ancienne version du fichier. Les anciennes versions du fichier finiront par disparaître du réseau en raison d'une demande de plus en plus faible au cours du temps.

Les clés sont obtenues à l'aide de la commande suivante :

```
java -cp <freenet directory>/freenet.jar freenet.client.cli.Main genkeys
```

III.2.3 Le routage

Le routage des requêtes constitue l'élément le plus important de Freenet. Il se différencie des autres protocoles de *Peer-to-Peer* qui maintiennent un index centralisé des fichiers afin que les utilisateurs puissent envoyer leurs requêtes directement aux possesseurs des données ; ou qui consistent à envoyer des requêtes en *broadcast*.

En effet, il utilise un système de recherche ascensionnelle. Chaque nœud transfert les requêtes vers le nœud qu'il suppose être le plus proche du destinataire.

Pour cela, chaque nœud détient une table de routage. A chaque requête fructueuse qu'un nœud retransmet, il stocke la donnée contenue dans le massage dans son cache local avant de faire suivre le message. Il ajoute également une nouvelle entrée dans sa table associant la source de la donnée et la clé de la requête. En général, un nœud stocke des fichiers ayant des clés assez proches.

III.2.4 Récupération de données

Chaque nœud maintient donc une table de routage contenant la liste des adresses des autres nœuds et des clés GUID qu'il pense qu'ils possèdent (l'explication de cette phrase au conditionnel se trouve dans la suite du chapitre).

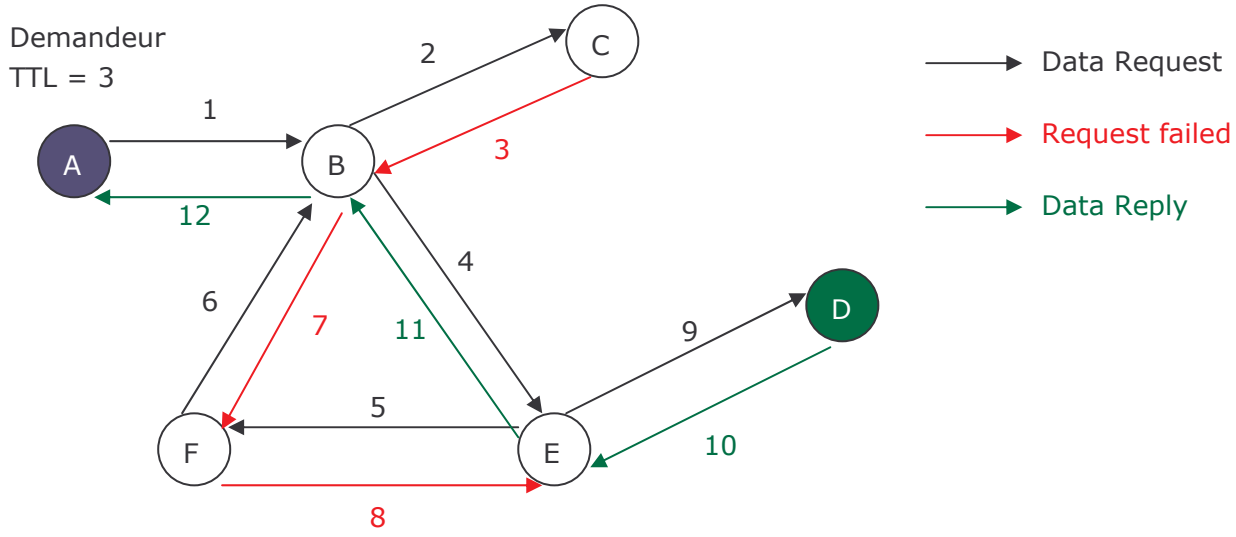
Lorsqu'un nœud reçoit une requête (*Data request*), il vérifie d'abord dans son espace disque local et, s'il trouve le fichier, il le renvoie avec *tag* indiquant qu'il est l'hébergeur des données. Sinon le nœud transfert la requête au nœud dans sa table dont la clé est la plus proche de celle demandée. Cet autre nœud vérifie alors dans son espace disque local et ainsi de suite. Si la requête aboutie, chaque nœud de la chaîne transfert le fichier en retour et créer une nouvelle entrée dans sa table de routage en associant l'hébergeur des données avec le GUID demandé. En fonction de sa distance avec l'hébergeur, chaque nœud peut également conserver une copie locale des données transférées. Il peut donc y avoir duplication de l'information à plusieurs niveaux sur réseau.

Afin de protéger l'identité de l'hébergeur des données, les noeuds altèrent occasionnellement les messages de réponses en se déclarant eux mêmes hébergeur dans le *tag* correspondant, avant de passer les messages au reste de la chaîne. Les requêtes suivantes pourront tout de même continuer à localiser les données, puisque le noeud qui a altéré le message a retenu la vraie identité de l'hébergeur des données dans sa propre table de routage. Il transférera lors la requête vers le bon hébergeur. Les tables de routage ne sont jamais révélées aux autres noeuds.

Afin de limiter l'utilisation des ressources, le demandeur donne à chaque requête, une durée de vie (*Time To Live*) qui est décrémenté à chaque noeud. Si le TTL expire, la requête échoue. L'utilisateur pourra cependant réessayer avec un TTL plus élevé, jusqu'à un certain maximum. L'ensemble des nœuds parcourus forme une chaîne.

Si un noeud envoie une requête à un destinataire qui est déjà dans la chaîne, ce dernier répondra avec un message d'erreur (*Request failed*) le noeud essaiera alors d'utiliser un autre noeud avec la clé la plus proche. Si un noeud n'a plus de candidats à essayer, il renvoie l'erreur à son prédécesseur dans la chaîne, qui essaiera alors un second choix et

ainsi de suite. Une fois le fichier trouvé, le nœud possesseur répondra avec une requête de succès (*Data reply*).



Sur ce schéma, A est à l'origine de la demande. Son plus proche « voisin » est le nœud B, il lui transfert donc sa requête qu'il transfert à son tour vers C. C ne possédant pas le fichier et ses voisins non plus, le TTL arrive à expiration. Il envoie donc un message « Request failed » à B. Le nœud B essaie alors de transmettre la requête vers F qui le retransmet vers B. Ce dernier détecte une boucle et envoie un message d'erreur qui sera propagé jusqu'à E. Le nœud E transmet alors la requête à D qui détient le fichier. Il répond donc avec un message « Data reply ». Sur le chemin, les nœuds E, B et A vont stocker le fichier.

Pour des requêtes sur des clés similaires, A transférera directement la requête vers B qui la transféra directement a E et à son tour la transmettra à D. Les nœuds qui répondent de façon fiable aux requêtes seront ajoutés à d'avantages de tables de routages et seront donc contactés plus souvent que les nœuds qui ne l'ont pas fait.

Grâce à ce mécanisme, les noeuds vont se spécialiser dans la localisation de clés similaires puisqu'un nœud présent dans une table de routage avec une clé particulière recevra les requêtes pour des clés similaires.

III.2.5 L'insertion de données

Pour insérer un fichier, un utilisateur lui attribue une clé GUID et envoie un message d'insertion (*DataInsert*) à son propre nœud, contenant la clé et une valeur de TTL. Ensuite ce message est envoyé aux « voisins » de l'utilisateur. En recevant une insertion, un nœud vérifie dans son espace disque local pour voir si la clé existe déjà. Si c'est le cas, l'insertion échoue, soit parce que le fichier est déjà sur le réseau, soit parce que l'utilisateur a déjà inséré un autre fichier avec la même description. Dans ce dernier cas, l'utilisateur devra réaliser une mise à jour de son fichier. Toutefois, les mises à jour ne sont toujours pas implémentées. Les développeurs travaillent actuellement sur un mécanisme assurant que toutes les copies soient remplacées.

Si un noeud reçoit un message *DataInsert* et qu'il ne dispose pas localement d'un fichier identifié par la clé contenue dans ce message, il va chercher dans sa table de routage la clé la plus proche et fera suivre le message au noeud correspondant. Si aucune collision n'a été rencontrée lorsque le TTL du message atteint la valeur nulle, un résultat confirmant l'insertion de la donnée sera alors envoyée en direction du noeud à l'origine de l'insertion.

La donnée nouvellement insérée sera enfin propagée le long du chemin établit par la requête d'insertion. Grâce au mécanisme de routage, elle sera stockée sur un noeud possédant des clés similaires, ce qui renforce le mécanisme de regroupement de clés.

III.2.6 Cryptage des données

Pour plusieurs raisons, les développeurs de Freenet ont décidés de faire circuler des données cryptées. Ainsi les possesseurs de nœuds ne peuvent pas avoir accès aux données présentent sur leur nœud.

Le réseau proprement dit ne sait rien de ce niveau de cryptage, il ne fait que transférer des données déjà cryptées lors de leur insertion.

III.2.7 Le protocole

Freenet utilise la communication par paquets (TCP, UDP, etc.). Un message contient un identifiant unique de 64 bits, une durée de vie (nombre de sauts aller, *hops-to-live*) et deux adresses, celle du destinataire et celle du l'émetteur. Un nœud stocke l'ID des requêtes qu'il a transmit afin d'intercepter les messages de réponse ou d'erreur et éviter les boucles.

Une communication commence par un message de type *Request.Handshake*. Si le nœud cible est actif, il répond avec un message *Reply.Handshake* spécifiant la version du protocole à utiliser. Une fois ouverte, elle active pendant quelques heures.

III.2.7.1 Recherche de données

Une fois la connexion établie, l'émetteur transmet un message *Request.Data* contenant une clé de recherche (représentant le fichier désiré). Le nœud distant vérifie alors dans son espace de stockage si la clé est présente. Si elle ne l'est pas, il transmettra alors le message au nœud possédant une clé la plus proche de celle recherchée. Selon la durée de vie du message, l'émetteur déclenche un *timer* correspondant au temps d'aller retour. Une fois écoulé, il considérera que sa requête a échouée. Pendant le transit de la requête, les nœuds distants peuvent envoyer des messages *Reply.Restart* à l'émetteur afin de lui indiquer qu'ils sont en attente de réponse et qu'il doit augmenter son *timer*.

Une fois que la requête est arrivée à destination, le nœud distant répond avec un message *Send.Data* contenant les données attendues ainsi que l'adresse du nœud qui l'a renvoyé (probablement pas le véritable nœud). Si la requête n'aboutie pas et qu'elle a atteint sa durée de vie, le nœud distant transmet alors un message *Reply.NotFound* qui sera transmit jusqu'à l'émetteur.

III.2.7.2 Insertion de données

Pour insérer des données, l'émetteur envoie un message *Request.Insert* en spécifiant un ID, une durée de vie et une proposition de clé pour les données. Le nœud distant regarde alors dans son espace de stockage et s'il ne trouve pas la clé, transmet alors le message. Les *timer* et les messages *Reply.Restart* sont également utilisés.

Si il y a une collision, c'est-à-dire si les données sont déjà présentes sur l'un des nœuds, le nœud distant répond soit avec un message *Send.Data* contenant les données soit avec un message *Reply.NotFound* si les données existes mais qu'elles ne sont pas encore trouvées. Temps qu'il n'y a pas de collision, les nœuds distants transmettent des messages *Request.Continue*. Si le temps de vie de la requête expire sans collision, le nœud distant répond alors avec un message *Reply.Insert*. Il attend alors un message *Send.Insert* contenant les données de la part de l'émetteur.

III.2.8 Le réseau Freenet

Plus le nombre de requêtes transmises est élevé et plus la connaissance des autres nœuds dans le réseau s'améliore et plus les routes s'adaptent de façon à être plus précises, sans nécessiter d'annuaire global.

III.2.8.1 Connexion

Pour rejoindre le réseau, un nouveau noeud génère d'abord une paire de clé publique / privée pour lui-même. Cette paire est utilisée pour identifier logiquement le noeud et pour signer une adresse physique de référence.

Le noeud envoie ensuite un message d'annonce incluant sa clé publique et son adresse physique à un noeud existant, localisé par un moyen externe tel que les communications personnelles ou une liste de noeuds publiée sur le Web, avec un TTL spécifié par l'utilisateur. Le noeud qui reçoit ce message note les informations d'identification de ce nouveau noeud et transfère l'annonce à d'autres noeuds choisis au hasard dans sa table de routage. L'annonce continue de se propager jusqu'à ce que le TTL expire. A ce moment-là, les noeuds dans la chaîne assignent collectivement un GUID au hasard dans l'espace de clés.

III.2.8.2 Gestion du stockage

L'espace de stockage des données est limité au nombre d'utilisateur et à l'espace qu'ils fournissent. Le système doit parfois décider quels fichiers il faut conserver ou non. Il alloue de l'espace en fonction de la popularité, mesurée par la fréquence du nombre de requêtes par fichier. Chaque noeud classe les fichiers qu'il possède par ordre de date de dernières requêtes. Lorsqu'un nouveau fichier arrive et qu'il ne peut pas être placé dans l'espace disque disponible, le nœud supprime les fichiers les moins demandés jusqu'à ce qu'il y ait assez de place.

III.2.8.3 Analyse des performances

Plusieurs analyses montrent que le réseau Freenet supporte assez bien les montées en charge et qu'il admet une tolérance aux erreurs honorable.

III.2.8.4 Sécurité - Anonymat

Lors d'une requête, le nœud situé immédiatement après le demandeur n'est pas capable de dire si son prédécesseur était l'initiateur du message ou s'il ne faisait que transférer un message venu d'un autre nœud. De la même façon, le nœud situé immédiatement avant le destinataire n'est pas capable de dire si son successeur est le véritable destinataire ou s'il continuera à transférer le message.

Cette organisation est destinée à protéger non seulement les producteurs d'informations et les consommateurs au début de chaque chaîne, mais aussi les hébergeurs à la fin de ces chaînes. Il est donc impossible de savoir précisément où sont stockées les informations. Du fait de l'architecture décentralisée et de la redondance des données, il est impossible d'empêcher la diffusion des informations.

De plus, une tentative d'attaque visant à supplanter un fichier déjà existant en créant délibérément des collisions par l'insertion de mauvais fichiers sous une clé existante, aura de grandes chances d'échouer et de disséminer le vrai fichier. En effet, lorsqu'une collision apparaît (c'est-à-dire que deux clés identiques sont rencontrées), le fichier déjà présent sur le réseau et qui est définit par la clé en question sera envoyé aux émetteurs des messages d'insertion, propageant ainsi le fichier déjà présent sur le réseau.

III.3 Freenet pas à pas

III.3.1 Installation

Lors des tests, Freenet a été installé sur un PC sous Windows XP relié à l'ADSL avec un débit de 1024Kb/s.

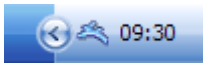
L'installation est détaillée sur le site officiel de Freenet. Il est nécessaire de télécharger un « installeur » via le web. Une fois lancé, il se connecte à un serveur afin de débuter l'installation.

Cependant, la version téléchargée posant problème, il a fallu télécharger une autre version de l' « installeur » sur un autre site.

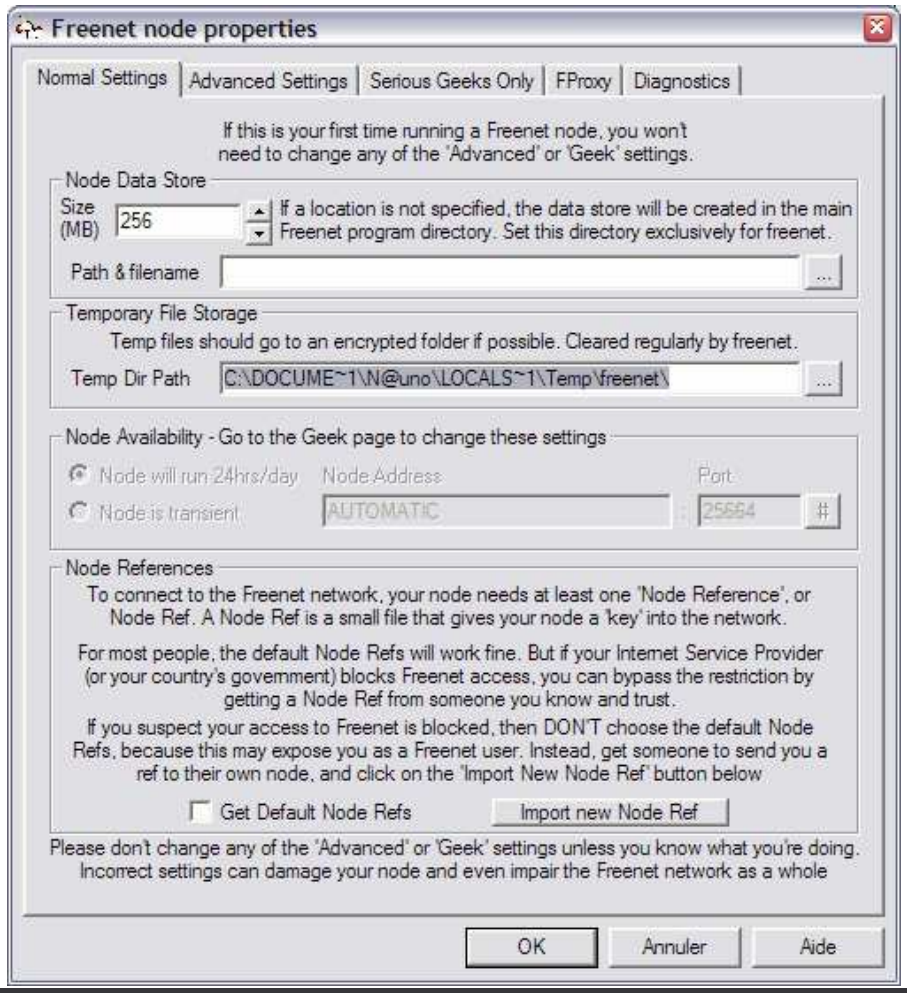
L'installation est assez rapide et très simple.

III.3.2 Configuration

Une fois lancé, Freenet est iconisé dans la barre Windows. Il fait office de serveur.



Il est possible de configurer Freenet en effectuant un clic droit sur l'icône puis sur *Configure…* (le logiciel n'est disponible que dans une seule langue). La fenêtre suivante s'ouvre :



III.3.2.1 Le dossier de stockage

La première chose à faire est de choisir où Freenet doit stocker ses données. Il faut ensuite indiquer l'espace disque qui lui est alloué. La valeur par défaut est 256Mo ; son minimum est de 10Mo et n'a pour seul maximum que la taille des disques durs.

III.3.2.2 Adresse IP et port

Par défaut, Freenet utilise le port 8888 de l'ordinateur. Etant accessible par un navigateur, il faut donc ouvrir un navigateur Web avec l'URL suivante : http://127.0.0.1:8888/ pour lancer son interface Web. Il est possible de la modifier.

III.3.2.3 Bande passante

Afin que la navigation sur Freenet ne sature pas celle sur Internet, il est possible de limiter la bande passante à lui allouer.

III.3.2.4 Autres options

Il est possible d'affecter à Freenet une priorité d'utilisation (*CPU priority*). Il est également possible de limiter le nombre de connexions ainsi que le nombre de *Thread*.

III.3.3 L'utilisation

L'utilisation de Freenet est différente des logiciels de P2P tels que Emule ou Kazaa. Ce n'est pas un logiciel permettant l'accès à des données ; il se contente de fournir un accès au réseau crypté fonctionnant au-dessus d'Internet. Un navigateur Web est nécessaire afin d'accéder à des dernières.

Aux vus des spécifications du protocole de Freenet, il n'est pas possible d'utiliser son réseau tout suite après son installation. Il doit dans un premier temps trouver ses « voisins » et ainsi établir une carte du réseau qui l'entour. Il faut alors lancer Freenet et attendre quelques instants afin qu'il découvre un minimum du réseau.

Une jauge présente sur la page d'accueil indique le niveau de charge du réseau.

Une fois sur cette page d'accueil, la navigation se fait de façon normale. Il faut naviguer de lien en lien pour obtenir une information. Seules les URL sont différentes. En effet, il n'y a pas de notion d'URL à proprement parler avec Freenet. L'accès à un site se fait par sa clé publique. Voici un exemple de clé :

http://127.0.0.1:8888/SSK@rjYFfgPHfolmcStiaoxESFfBXz8PAgM/FreenetHelp//

SSK représente le type de clé, rj….gM représente la clé publique et FreenetHelp est l'espace de noms.

Voici l'interface Web lancée lorsque l'utilisateur double-clic sur l'icône de Freenet (ou clic-droit sur l'icône puis *Open Gateway* :

III.3.3.1 Mode expert

Il est également possible d'obtenir une interface en mode « expert ». Ce mode présente un menu plus complet ainsi qu'un formulaire afin de pouvoir poster des fichiers. Dans ce mode, il est possible de visualiser les requêtes qui ont échouées, les tâches en cours, des informations sur l'environnement de Freenet, etc.

III.3.3.2 Le téléchargement de fichiers

L'absence de moteur de recherche est une limitation gênante dans l'utilisation de Freenet. Il faut parcourir la liste des sites présents sur les annuaires Freenet afin d'avoir une idée du contenu qu'on peut y trouver. De plus, de nombreux sites ne doivent pas être référencés et leur adresse doit circuler exclusivement sur les fora ou les courriels Freenet.

Il est donc assez fastidieux de trouver un fichier ; cela est dû à la grande importance qu'attachent les développeurs de Freenet à la confidentialité des échanges entre utilisateurs.

Pendant nos recherches, nous avons pu constater la présence d'un logiciel, *Fuqid*, spécialisé dans le téléchargement de fichiers sous le réseau Freenet. Une fois installé, il a besoin d'un accès au réseau Freenet. Ensuite il se comporte comme les logiciels de P2P standard, permettant de mettre en pause, de reprendre ultérieurement les téléchargements, de gérer plusieurs téléchargements, etc.

III.3.4 Observations

Durant les essais, nous avons pu constater la lenteur de recherche des pages ainsi que leur affichage. Cependant, nous n'avons peut être pas attendu un temps nécessaire afin qu'acquérir une connaissance du réseau suffisante.

III.3.4.1 Le contenu

Le contenu y est vaste. Nous avons trouvé (grâce aux annuaires des sites) tous types de documents comme des vidéos ou des fichiers musicaux à télécharger, des sites ouverts à tous traitant de sujets d'actualité, des jeux, des sites à caractère pornographique, etc.

Beaucoup de sites sont référencés mais leur résumé reste vague et ne semblent pas être intéressants. Devant les lenteurs rencontrées, seul un nombre restreint de sites a été visité.

La majorité des sites indexés sont en anglais. Nous avons cependant rencontré des sites en allemand et en français.

III.3.4.2 Lors d'un échec d'accès aux données

Lors de nos essais, nous avons rencontrés plusieurs types d'erreur.

Valeur du TLL

Lors de la demande d'accès à quelques sites, nous avons eu une page d'erreur indiquant que les données n'avaient été atteintes peut être en raison d'un TTL trop faible. Un formulaire nous proposait alors de saisir une nouvelle valeur de ce TTL et de retenter l'accès aux données. Dans certains cas, ce mécanisme a fonctionné nous donnant la possibilité d'accéder aux données désirées.

Réseau occupé

Nous avons également rencontrés à plusieurs reprises, le message d'erreur indiquant une trop forte demande sur le réseau. Il était alors nécessaire de patienter puis de réitérer notre requête.

III.4 Conclusion sur Freenet

Freenet est encore en développement et nécessite des optimisations, mais les principales caractéristiques voulues lors de sa création fonctionnent déjà. Il remplit donc ses objectifs, possède plusieurs modules annexes (mails, newsgroups). Le concept de mobilité des données est très intéressant et mérite d'être approfondi.

Cependant sa lenteur d'accès et le contenu qu'il propose pour le moment ne font pas de lui un système attrayant.

Une poignée d'adeptes semble pourtant miser sur celui-ci mais face à des réseaux comme Edonkey ou Bitorrent, Freenet ne parviendra pas à s'imposer.

De plus, en juillet 2003 une faille importante du système a été mise à jour suite à la sortie d'une nouvelle version. Il était en effet possible à une personne ayant créé une page Web hébergée sur Freenet de connaître l'adresse IP de tous ses visiteurs en exploitant une faille qualifiée officiellement comme "sérieuse". Sur un réseau *Peer-to-Peer* traditionnel, cela pourrait paraître banal, mais sur Freenet ? Bien sûr le bug a été corrigé dès le lendemain, tous les utilisateurs qui ont pu effectuer la mise à jour et Freenet reste sans conteste le réseau public le plus sûr actuellement. Pourtant, cette faille a peut-être fait réaliser à cette communauté que même Freenet ne garantie pas une sécurité et un anonymat à 100%. D'autres bugs cette fois non corrigés pourraient être exploités par des personnes mal intentionnées…

Tout ceci font de Freenet un réseau encore trop marginal pour certains et trop peu connu du grand public.

Ses développeurs continuent toutefois à miser sur ce dernier et de nombreuses améliorations sont prévues et ce malgré un nouveau projet de Ian Clarke. Ce projet nommé *Dijjer* reprend le concept de mise en cache des informations de Freenet.

Freenet reste un projet trop ambitieux qui n'a pas bénéficié du succès escompté par son auteur. La réplication des données les plus demandées est une technique peu commune et elle pose plusieurs problèmes. En effet, empêcher la redondance d'information est habituellement un enjeux pour un développeur. Aller à l'encontre de cela risque à terme de poser des problèmes de stockage. De plus, les développeurs de Freenet cherchent encore une façon de d'assurer la mise à jour de toutes les copies d'un document lors de sa mise à jour par son auteur.

Pourtant, devant les différentes actions des Majors à l'heure actuelle pour enrayer le phénomène d'échanges illégaux de fichiers musicaux, Freenet pourrait se révéler LA solution. En effet, une augmentation des utilisateurs augmenterait nettement l'espace de stockage et améliorait également la vitesse d'accès aux données (ceci augmenterait la probabilité que la donnée soit proche de notre nœud).

IV JXTA

IV.1 Introduction sur JXTA

Les applications Peer-to-Peer sont en grande expansion. Cependant, la plupart de ces applications sont limitées à une seule plateforme et sont incapables d'échanger directement des données avec des applications similaires (Kazaa/Gnutella, ICQ/AIM/MSIM). C'est dans l'optique de supprimer ces défauts que Sun Microsystem a proposé ce framework Peer-to-Peer Open Source : JXTA. JXTA comme JuXTAposition entre le nouveau et l'existant c'est-à-dire le model client/serveur et le N2N (Node to Node).

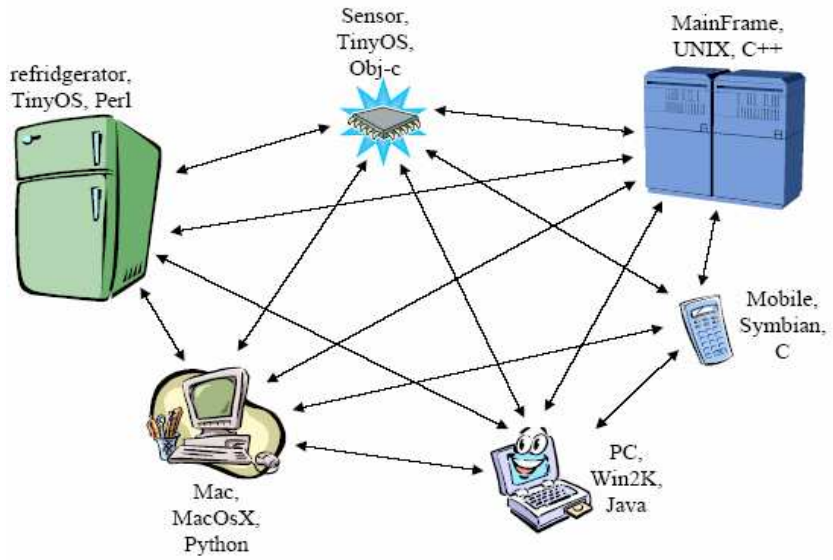
Le framework JXTA a été officialisé par Bill Joy, chef scientifique et CEO de Sun Microsystem, le 15 février 2001 lors de la conférence O'Reilly à San Francisco sur le peer to peer (P2P). Il invitait la communauté à se joindre aux efforts de son équipe afin de réfléchir au développement du P2P. Le site de JXTA (www.jxta.org) a ouvert le 25 Avril 2001 hébergé par collabNet, il héberge une centaine de projets actuellement. Il y a 25000 membres actifs.

Cette partie a pour objectif de présenter un « état de l'art » de JXTA et ses diverses composantes sur base des articles et documentations disponibles. Pour y parvenir, nous présenterons tout d'abord JXTA et ses origines. Et, nous passerons en revue ses objectifs essentiels. Ensuite, nous nous attarderons sur l'architecture proposée par Sun ainsi que sur les concepts et différents protocoles entrant dans sa composition.

IV.2 Qu'est ce que JXTA ?

JXTA est un framework Peer-to-Peer Open Source. C'est une spécification d'un ensemble de 6 protocoles, qui permettent le partage, la communication et la collaboration entre les peers. Les 6 protocoles sont indépendants et nous ne sommes pas tenus de les implémenter tous.

JXTA offre des communications via des messages XML et permet ainsi à n'importe quelle entité sur le réseau (téléphones cellulaires, PDA, serveurs, PCs, …) de communiquer :



IV.3 Les objectifs de JXTA

Le but du framework JXTA est de créer une plateforme qui permette de construire, simplement et facilement, un grand ensemble de services et d'applications distribués pour tout dispositif qui pourrait être un « peer ». JXTA permet, entre autre, aux développeurs de se concentrer sur le développement de leurs applications, en créant des logiciels distribués flexibles, inter opérables, disponibles pour tout « peer » de l'Internet.

Le framework JXTA a 3 objectifs :

- l'interopérabilité ;
- l'indépendance vis-à-vis des plateformes logicielles et matériel ;
- l'ubiquité ;

IV.3.1 Interopérabilité

Les applications de P2P actuelles sont faites pour un seul type de service (échange de musique, de fichiers, ...). Comme il n'y a aucune infrastructure commune, chaque créateur de software P2P crée sa propre communauté et développe les primitives de base des systèmes. Cela rend les systèmes incompatibles et en plus constitue des efforts inutiles ! JXTA tient à résoudre ce problème en proposant des primitives et des services fondamentaux communs.

IV.3.2 Indépendance de la plateforme

La plus part des soft P2P actuels sont basés sur Windows et TCP/IP. Or ceci n'est pas l'unique type de plateforme existant. Si une application doit tourner sur d'autres systèmes et communiquer avec le système de base, soit les services doivent être re-développés, soit des ponts entre les technologies doivent être prévus. JXTA propose d'assurer un système garantissant les échanges entre tous les types de plateformes et de protocoles.

IV.3.3 Ubiquité

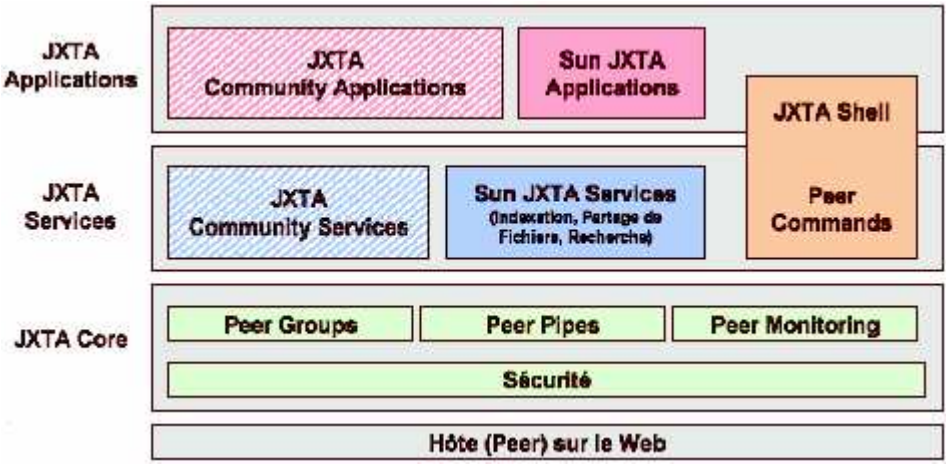
Le dernier objectif est l'ubiquité. Il matérialise le désir que le framework JXTA soit implémentable sur n'importe quel périphérique doté d'un coeur digital ("digital heartbeat"). Cette notion est très générale puisqu'elle comprend les senseurs, l'électronique grand public, les PDA, les GSM, les routeurs réseaux, les ordinateurs de bureau, les serveurs de données ou de traitements, etc.

IV.4 Architecture logicielle de JXTA

IV.4.1 Présentation

La structure possède 3 couches :

- le noyau : JXTA Core ;
- la couche des services : JXTA Services ;
- la couche des applications : JXTA Applications.



IV.4.2 JXTA Core (noyau)

Il permet d'implémenter les services d'administration : création et gestion de groupes de pairs, communication et sécurité. Les pipes sont les canaux de communication entre les pairs. Un message est envoyé à un pair par un pipe, sa structure est en XML et peut contenir des données, du contenu et/ou du code. L'intégrité, la confidentialité et la sécurité peuvent aussi être garanties par le pipe.

IV.4.3 JXTA Shell

Le JXTA Shell est un outil qui se situe à cheval sur les couches services et applications. Il permet aux développeurs ou aux utilisateurs avancés, d'interagir, de contrôler ou de tester l'environnement de pair. Cet outil fonctionne à la manière d'un shell Unix.

IV.4.4 JXTA Services (couche des services)

Le JXTA Services permet de faciliter le développement d'applications. C'est une librairie qui étend les fonctionnalités offertes par le JXTA Core. Cette couche fournit des mécanismes pour rechercher, partager, indexer et mettre en cache du code et du contenu. Le mécanisme de recherche de contenu peut être distribué et/ou parallèle et utilise XML comme représentation des requêtes.

IV.4.5 JXTA Applications (couche des applications)

Le JXTA Applications permet la construction de services ou d'applications pour les pairs. Au sein du JXTA Applications, les applications seront développées sur base des services disponibles dans le JXTA Services accédant aux mécanismes de base fournis dans le JXTA Core.

IV.5 Les concepts

Cette partie a pour objectif de présenter les concepts essentiels du framework JXTA tels que le Peer, le PeerGroup, le Peer Router, le pipe, le Peer EndPoint, l'Advertisment, le Message et le Module.

IV.5.1 Peer

Il s'agit d'une abstraction correspondant à un *device* connecté au réseau et implémentant un ou plusieurs protocoles de JXTA. Il est caractérisé par un comportement indépendant et asynchrone des autres *peers*. Il est spécifié par un identifiant et peut appartenir à un ou plusieurs groupes.

IV.5.2 PeerGroup

C'est un ensemble de *peers* partageant un ou plusieurs services. Il est spécifié par un identifiant. Un *peer* appartient à 1 ou plusieurs groupes. Par défaut, il appartient au NetPeerGroup. Le but d'un groupe peut être :

- Créer un environnement sûr (Secure Environment) ;
- Créer un environnement délimité (Scoping Environment) ;
- Créer un environnement de contrôle (Monitoring Environment) ;

Tout groupe a un parent. Il permet notamment la publication de l'advertisement correspondant à un groupe.

IV.5.3 Peer Router

Il est utilisé pour trouver un chemin de communication pour les *peers* qui sont séparés par un *firewall*.

IV.5.4 Pipe

Il correspond à un canal de communication entre des *peers*. Les caractéristiques minimums d'un *pipe* sont :

- Asynchrone ;
- Unidirectionnel : Il doit donc exister des inputs pipes et des outputs pipes ;
- Virtuel : Les extrémités peuvent être connectées à un ou plusieurs points. Il existe donc des point-to-point pipes (1 to 1) et des propagate *pipes* (1 to n). De plus, les extrémités sont dynamiques. Ainsi, en cas d'erreur, une extrémité peut être changée.

L'implémentation des pipes est fonction des protocoles réseaux que supporte l'implémentation de JXTA utilisée.

IV.5.5 Peer EndPoint

C'est une destination logique à laquelle un *pipe* est lié. Exemple : AdresseIP :Port.

IV.5.6 Advertisement

Il s'agit d'un document en XML qui nomme, décrits et publie l'existence de *peer*, *peer group*, de *pipes* ou de services.

Il y a actuellement 9 types *d'advertisements* définis :

- Peer Advertisement ;
- Peer Group Advertisement ;
- Pipe Advertisement ;
- Module Class Advertisement ;
- Module Spec Advertisement ;
- Module Impl Advertisement ;
- Content Advertisement ;
- Peer Info Advertisement.

IV.5.7 Message

C'est une unité de base d'échanges entre des *peers*. Ils sont envoyés et reçus par le biais des pipes.

Il existe deux moyens de les représenter :

- En XML ;
- En binaire (encodé en Base64 dans le corps d'un message XML).

IV.5.8 Module

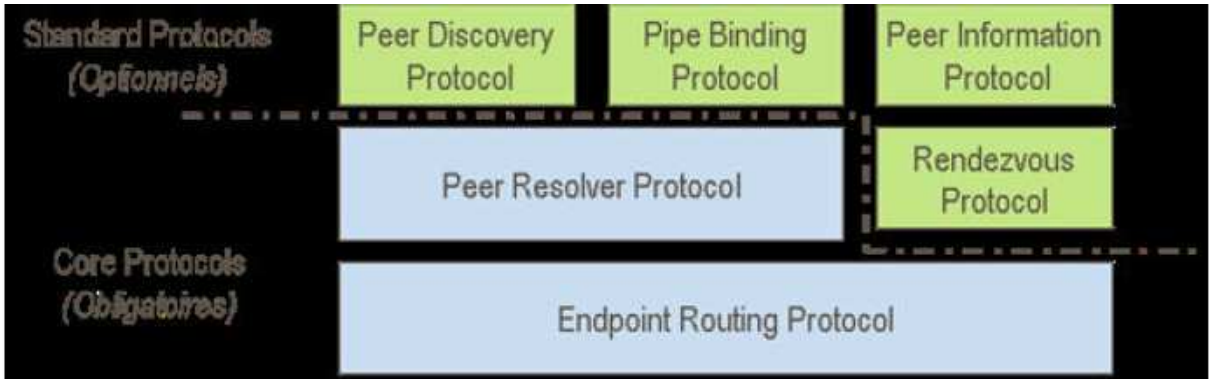
C'est une abstraction utilisée pour représenter du code implémentant un comportement dans le monde de JXTA. Le code peut être du Java, C, des dll, etc. Un *peer* rejoignant un groupe peut y découvrir des nouveaux comportements qu'il désire implémenter.

La représentation des modules est indépendante de la plateforme afin de pouvoir permettre la création de groupe hétérogène. Il existe plusieurs sortes de modules :

- *Module Class :* présente un comportement attendu et l'identifie par un ModuleClassID ;
- *Module Specification :* contient toutes les informations nécessaire pour accéder ou invoquer un module. Il est identifié par ModuleSpecID (contient le ModuleClassID) ;
- *Module Implementation : c*ontient le ModuleSpecID correspondant.

IV.6 Les protocoles

Les protocoles définis par JXTA sont des formats de messages XML permettant la communication entre les *peers*. Il y a en tout 6 protocoles dont 2 seulement doivent obligatoirement être implémentés. On peut représenter l'architecture de ces protocoles ainsi :



Explication de ces 6 protocoles JXTA :

- Endpoint Routing Protocol (**ERP**) : il permet d'obtenir des informations sur les «routes» disponibles pour envoyer un message à un autre peer ;
- Rendezvous Protocol (**PDP**) : il permet de « propager » un message dans un peerGroup et de recevoir des messages propagés ;
- Peer Resolver Protocol (**PRP**) : il permet d'envoyer une requête à un ou plusieurs peers et recevoir une ou plusieurs réponses ;
- Peer Discovery Protocol (**PDP**) : il « publie » ses propres ressources et découvre les ressources d'autres personnes ou groupes ;
- Peer Information Protocol (**PIP**) : il donne des informations sur les capacités et status des autres peers ;

Pipe Binding Protocol (**PBP**) : il établit un canal de communication vers un ou plusieurs *peers*. Permet de diriger l'endroit où envoyer un message

IV.7 Avantages / Inconvénients

IV.7.1 Avantages

Comme dit précédemment, JXTA permet d'implémenter n'importe quel langage sur n'importe quel systèmes d'exploitation (ouverture vers le matériel "non PC" comme les téléphones).

Tout en étant décentralisé (Peer-to-Peer), un module centralisé (comme une application de type messagerie instantanée) pourra être facilement ajouté à une autre application (si les deux applications supportent les protocoles Juxta).

IV.7.2 Inconvénients

JXTA est neuf, change rapidement et est faiblement documenté.

Seul un début de noyau est aujourd'hui disponible.

Il existe des incertitudes liées au potentiel d'évolution de la plate-forme et à son adoption par les utilisateurs-développeurs.

IV.8 Conclusion sur JXTA

JXTA est un framework permettant de créer facilement des applications Peer-to-Peer interopérables.

Ce framework est en constante amélioration. Ainsi, de nouveaux services sont en cours de création, le Shell et la sécurité sont en cours d'amélioration.

Cependant, JXTA a besoin de meilleurs performances et de faire ces preuves a grande échelle.

V Références

Peer to peer

- http://www.p2pwatchdog.com/
- http://www.p2pfr.com/

Kazaa

- http://www.kazaa.com/ (Site officiel)
- http://cvs.berlios.de/cgi-bin/viewcvs.cgi/gift-fasttrack/giFT-FastTrack/PROTOCOL?rev=1.19&content-type=text/vnd.viewcvs-markup
- http://www.slyck.com/ft.php

FreeNet

- http://freenet.sourceforge.net/ (Site officiel)
- http://freenet-france.mondelibre.org/ (Site français)

JXTA

- http://www.jxta.org (Site officiel)
- http://jxta.free.fr (Site français)