Université Abou Bekr Belkaid
Tlemcen Algérie
جامعة أبي بكر بلقايد
تلمسان الجزائر

République Algérienne Démocratique et Populaire

Université Abou Bakr Belkaid– Tlemcen

Faculté desSciences

Département d'Informatique

Mémoire de fin d'études

Pour l'obtention du diplôme de Master en Informatique

Option: *Réseaux et Systèmes Distribués (R.S.D.)*

Thème

Mise au point d'une Application de téléchargement en Peer-to-Peer

Réalisé par : Kalule Joseph

Présenté le 11 Juin 2015 devant le jury composé de

- *Mme Iles…….* *(Président)*
- *Mme Didi Fedoua……..* *(Encadreur)*
- *Mr Benmammar Badr……..* *(Examinateur)*

Année universitaire : 2014-2015

Remerciements

J'adresse mes remerciements aux personnes qui m'ont aidé dans la réalisation de ce mémoire. Je souhaite avant tout remercier mon tuteur Mme. Didi Fedoua pour le temps qu'elle a consacré à m'apporter les outils méthodologiques indispensables à la conduite de cette recherche. Son exigence m'a grandement stimulée. L'enseignement de qualité dispensé par l'équipe d'enseignants a également su nourrir mes réflexions et a représenté une profonde satisfaction intellectuelle, merci donc aux enseignants.

Enfin, un grand merci à ma famille qui m'a supporté émotionnellement pendant toute la durée de la recherche et la rédaction de ce mémoire.

Je n'oublie pas de remercier les membres du jury pour avoir pris le temps d'examiner mon PFE, qu'il trouve ici l'expression de mon profond respect et mes plus chaleureux remerciements.

Dédicaces

Ce modeste travail est dédié à:

A celle qui a été toujours la source de grande affection et le goût de la vie .....ma MERE

A tous les enseignants qui m'ont transmis des connaissances depuis ma première année de Licence jusqu'à la deuxième année de Master

A tous mes frères et sœurs

A toute ma famille et mes proches.

A tous mes amis sans exception.

A mon collègue Aly

A toute la promotion informatique RSD 2014/2015 que je leurs souhaite un bon avenir.

A tous ceux qui m'ont aidé de près ou de loin à la réalisation de ce modeste travail.

Résumé

Le partage de fichiers pair-à-pair a émergé en tant que composant de l'utilisation dominante de la bande passante Internet. Commençant par le phénomène Napster de la fin des années 1990, la popularité de P2P a considérablement augmenté le volume de données transférées entre les utilisateurs d'Internet. Pourtant, l'énorme popularité du partage de fichiers et la largeur de protocoles concurrents rend le blocage du trafic P2P une impossibilité pratique. Ce PFE traite du téléchargement de fichiers en mode pair à pair, d'une manière simple et intuitive, l'application écrite en java, permet donc de partager toutes sortes de fichiers entre utilisateurs disparates d'un réseau local donné sans passer par des serveurs.

Abstract

Sharing peer-to-peer file emerged as a component of the dominant use of the Internet bandwidth. Beginning with the Napster phenomenon of the late 1990s, the popularity of P2P has dramatically increased the amount of data transferred between Internet users. Yet the huge popularity of file sharing and width of competitors protocols makes blocking P2P traffic a practical impossibility. This Project of end of Graduation studies treats downloading files in peer to peer mode, a simple and intuitive way, the application is written in Java, so let's share all kinds of files between disparate users of a local area network without going through servers.

ملخص

ظهرت تقاسم الند للند ملف كعنصر من عناصر الاستخدام الأوسع من عرض النطاق الترددي الإنترنت. بدءا من ظاهرة نابستر في أواخر 1990s، وشعبية P2P زاد بشكل كبير من كمية البيانات المنقولة بين مستخدمي الإنترنت. ومع ذلك فإن شعبية ضخمة من تبادل الملفات وعرض بروتوكولات المنافسين يجعل عرقلة حركة المرور P2P استحالة العملية. هذا المشروع التخرج يعامل تحميل الملفات في الند للند واسطة، وسيلة بسيطة وبديهية، وتطبيق مكتوب بلغة جافا، لذلك يتيح حصة جميع أنواع الملفات بين المستخدمين تباين في الكوكب بأسره دون قبل ملقمات.

Table des matières

Chapitre 1 : Introduction Générale

Le Peer-to-peer (P2P) est une architecture d'applications distribuées qui partitionne des tâches ou des charges de travail entre pairs. Les pairs sont également privilégiés, les participants équipotents dans l'application. On dit alors qu'ils forment un réseau de nœuds pair-à-pair (*peer-to-peer*), contrairement au modèle client-serveur, dont ci-joint une comparaison voir Tableau1 ci-dessous.

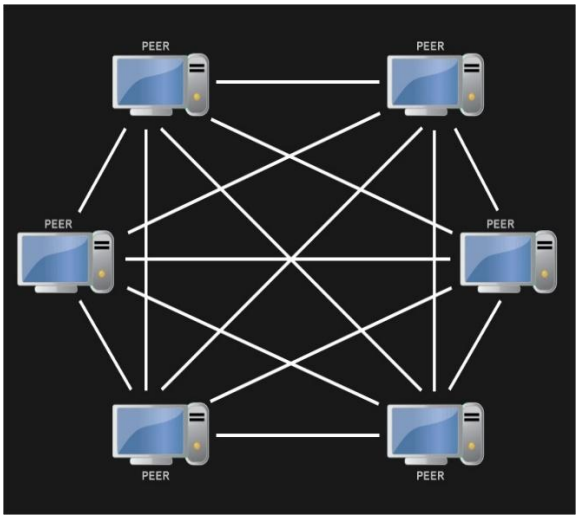


Figure 1.1 : Un réseau Pair à Pair

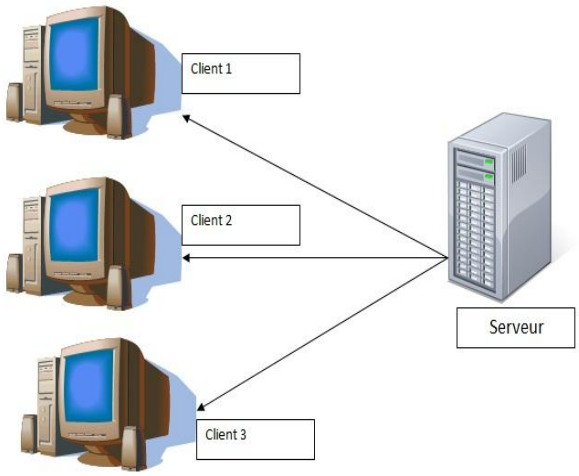


Figure 1.2 : Un réseau Client-Serveur

Les pairs rendent une partie de leurs ressources, telles que la puissance de traitement, de stockage sur disque ou bande passante du réseau, directement accessible aux autres participants du réseau, sans la nécessité d'une coordination centrale par les serveurs ou hôtes stables. Les pairs sont les fournisseurs et les consommateurs de ressources, en contraste avec le modèle client-serveur classique dans lequel la consommation et la fourniture de ressources sont divisées. Des systèmes P2P collaboratifs émergents vont au-delà de l'ère de pairs qui font des choses similaires tout en partageant des ressources, et sont à la recherche de divers pairs qui peuvent apporter des ressources et des capacités uniques à la communauté virtuelle

donnant ainsi la possibilité de s'engager dans de plus grandes tâches autres que celles qui peuvent être accomplies par les pairs individuels, mais qui sont bénéfiques pour tous les pairs.

Client-Serveur	Peer-to-Peer
Serveur présent. Plus puissant que les clients	Serveur absent. Chaque machine est généralement un simple ordinateur personnel
Nécessite une expertise technique pour mettre en place et maintenir	Exige des compétences techniques de base pour mettre en place et maintenir
Les données et la gestion des applications sont centralisées	Chaque machine a ses propres données et applications
La sécurité est gérée centralement sur le serveur	Le réseau est aussi sécurisé que la machine la moins sécurisée (plus vulnérable) sur le réseau
L'accès de l'utilisateur et l'authentification sont gérés de manière centralisée.	Authentification de connexion assez basique pour chaque utilisateur.
Du matériel spécialisé et donc coûteux impliqué, en particulier concernant le serveur	Pas de matériel spécialisé – il n'y a que des machines ordinaires
Les utilisateurs peuvent se déplacer de machine en machine et toujours avoir accès à leurs données, les applications et la configuration de bureau.	Les données de l'utilisateur doivent être dans un dossier partagé si elles doivent être visibles à partir des autres machines

Tableau 1.1 : Comparaison entre le modèle Client-Serveur et le modèle Peer-to-Peer

Objectif

Ce document explore une grande partie de l'ensemble des technologies peer to peer qui existent aujourd'hui. On y donne aussi un tableau comparatif des deux modèles en concurrence le P2P et « client/serveur ».Puis on se propose d'implémenter une application qui permet des communications peer to peer lors de transferts de fichiers.

Structure du mémoire

Ce mémoire est organisé en quatre chapitres. Chacun de ces chapitres aborde des points spécifiques, et il est structuré comme suit:

Le premier chapitre présente une introduction aux réseaux pair-à-pair, et les différents concepts liés à ces réseaux.

Les deuxièmes et troisièmes chapitres expliquent respectivement le fonctionnement de plusieurs protocoles des réseaux non-structurés et réseaux structurés existants dans le domaine du partage des fichiers pair-à-pair.

Le dernier chapitre est réservé à l'implémentation d'une application pair-à-pair simple mais fonctionnelle. Il explique en détails comment les différents composants de cette application travaillent ensemble pour délivrer sa fonctionnalité à l'utilisateur.

Enfin, le manuscrit se termine par une conclusion générale dans laquelle est résumé l'essentiel de ce travail et quelques orientations des travaux futurs sont données.

Chapitre 2: Les réseaux non-structurés

1. Introduction à Napster

Napster était le nom donné à deux services en ligne axés sur la musique. Il a été fondé à l'origine comme un service Internet pionnier de partage et/ou téléchargement de fichiers en peer-to-peer (P2P), qui mettaient l'accent sur le partage de fichiers audio, musique, généralement encodés au format MP3. La société d'origine a connu des difficultés juridiques [5], plus des violations de copyright, a cessé ses activités et a finalement été acquis par Roxio. [4] Dans sa seconde incarnation Napster est devenu un magasin de musique en ligne jusqu'à ce qu'il soit acquis par Rhapsody de Best Buy le 1er Décembre 2011.

Les entreprises et les projets ultérieurs suivirent avec succès son partage de fichiers P2P tels que par exemple Gnutella, Freenet, Kazaa, et bien d'autres. Certains services, comme LimeWire, Scour, Grokster, Maître, et eDonkey2000, ont été ramenés ou modifiées en raison de circonstances similaires.

a) Histoire

Bien qu'il y ait déjà des réseaux qui facilitent la distribution de fichiers sur Internet, tels que IRC, Hotline et Usenet, Napster spécialisée dans les fichiers MP3 de musique et une interface conviviale. À son apogée, le service Napster avait environ 80 millions d'utilisateurs enregistrés.

Napster fait qu'il est relativement facile pour les amateurs de musique à télécharger des copies de chansons qui étaient difficiles à obtenir autrement, comme des chants anciens, enregistrements inédits, et des chansons à partir d'enregistrements de concert bootleg (non-publiés officiellement par l'artiste concerné). Certains utilisateurs se sont sentis justifiés dans le téléchargement de copies numériques d'enregistrements qu'ils avaient déjà achetés dans d'autres formats, tels que LP et cassette, avant que le disque compact ait émergé comme le format dominant pour les enregistrements musicaux.

Les réseaux à haute vitesse dans les dortoirs de collèges sont devenus surchargés, avec 61% du trafic de réseau externe constitué de transferts de fichiers MP3. De nombreux collèges ont bloqué son utilisation pour cette raison, avant même que les problèmes de responsabilité et de violation de droits d'auteur n'apparaissent.

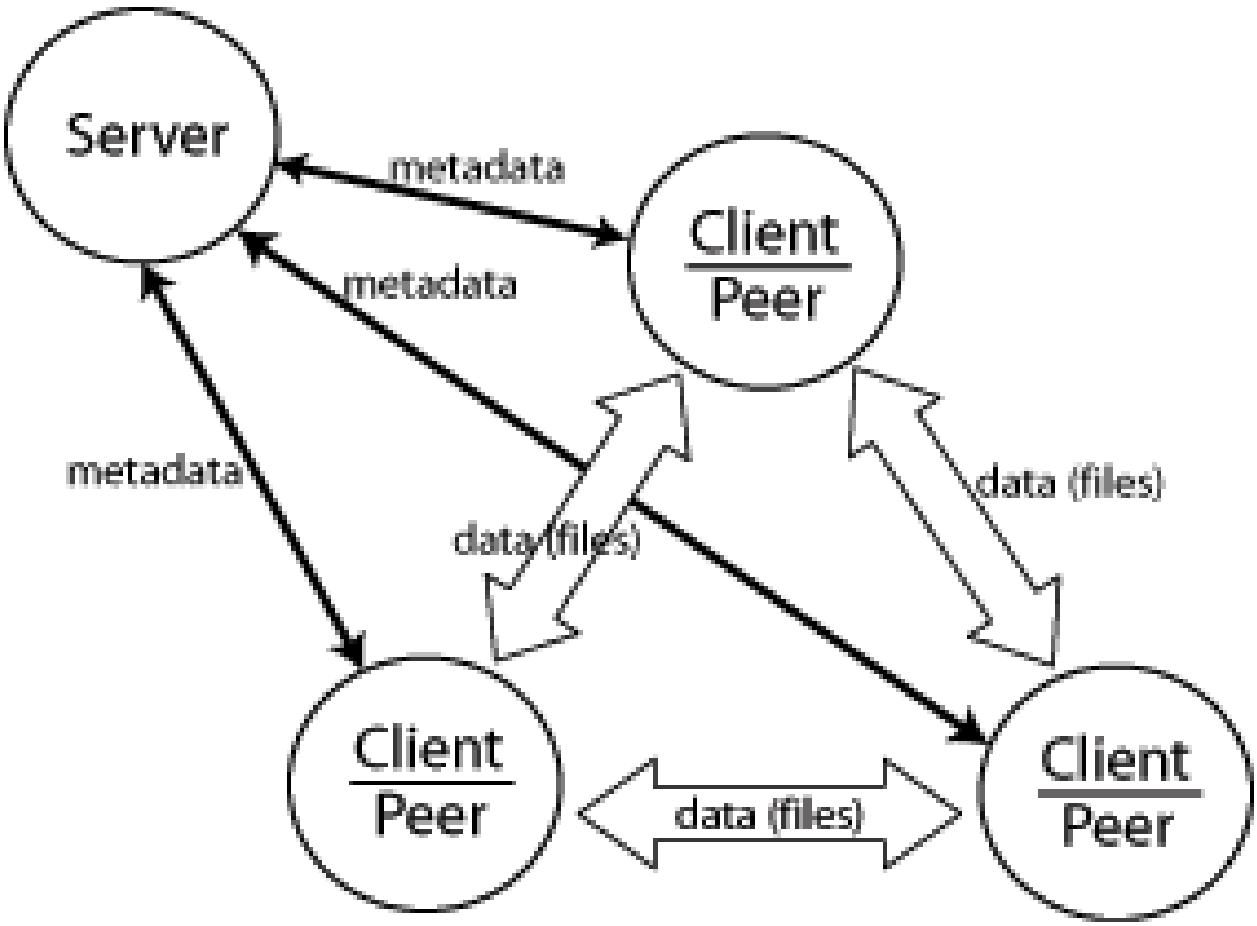


Figure 2.1: Le fonctionnement de Napster

Comme on peut le voir via la figure ci-dessus, l'architecture de Napster reste basée sur les serveurs pour retrouver l'information recherchée, donc elle n'est pas entièrement peer to peer.

En principe Napster a combiné trois fonctionnalités [3]

1) Un moteur de recherche
2) Le partage de fichiers mp3
3) Internet Relay Chat (IRC) – la messagerie instantanée

2. Gnutella

Gnutella est un grand réseau peer-to-peer. Ce fut le premier réseau décentralisé peer-to-peer en son genre, conduisant à d'autres réseaux, et plus tard l'adoption du modèle. Il célèbre une décennie d'existence, le 14 Mars 2010 et a une base de millions d'utilisateurs pour le partage de fichiers peer-to-peer.

À la fin de 2007, c'était le réseau de partage de fichiers le plus populaire sur Internet avec une part de marché estimée à plus de 40%.

a) Conception et fonctionnement

Pour imaginer d'abord comment Gnutella fonctionne, imaginez un grand cercle d'utilisateurs (appelés nœuds), chacun d'eux disposant du logiciel client Gnutella. Au démarrage initial, le logiciel client doit amorcer et trouver au moins un autre nœud. Diverses méthodes ont été utilisées pour cela, y compris une liste d'adresse préexistante de nœuds éventuellement de travail fournis avec le logiciel, en utilisant des caches Web mis à jour de nœuds connus (appelés Gnutella caches Web), ou UDP Host Caches et, plus rarement, même IRC. Une fois connecté, le client demande une liste d'adresses de travail. Le client tente de se connecter aux nœuds de la liste, puis il reçoit d'autres clients, jusqu'à ce qu'il atteigne un certain quota. Il se connecte à seulement quelques nœuds, et met en cache localement les adresses qu'il n'a pas encore essayées, et rejette les adresses qu'il a essayées qui étaient invalides.

Lorsque l'utilisateur veut faire une recherche, le client envoie la demande à chaque nœud connecté activement. Dans la version 0.4 du protocole, le nombre de nœuds connectés et actifs pour un client était assez petite (environ 5), de sorte que chaque nœud ensuite transmet la demande à tous ses nœuds connectés et actifs, qui à leur tour, transmettent la demande, et ainsi de suite, jusqu'à ce que le paquet atteigne un nombre prédéterminé de houblon de l'expéditeur (maximum 7).

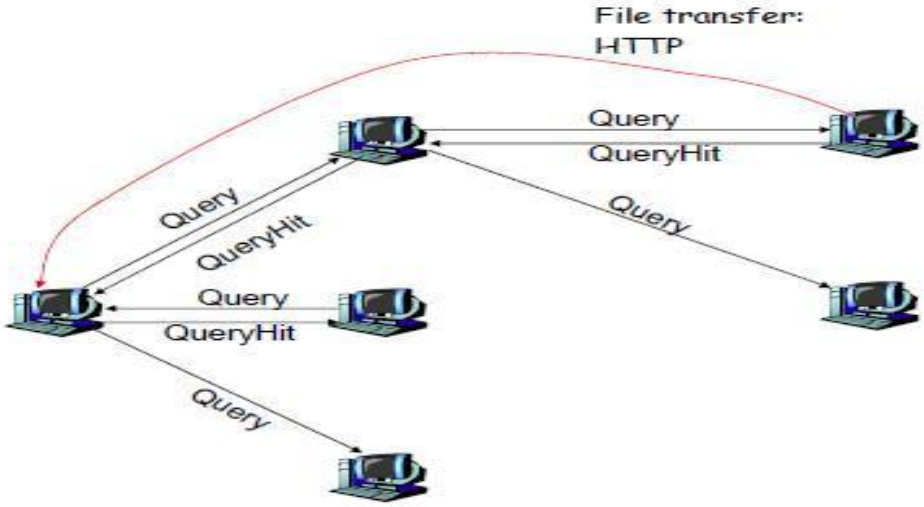


Figure 2.2 : La recherche et téléchargement éventuels avec Gnutella

Depuis la version 0.6 (2002), Gnutella est un réseau composite fait de nœuds feuilles et ultra nœuds (également appelés UltraPeers). [6] Les nœuds terminaux sont reliés à un petit nombre d'UltraPeers (généralement trois), tandis que chaque ultrapeer est connecté à plus de 32 autres

UltraPeers. Avec ce degré sortant supérieur, le nombre maximal de sauts d'une requête a été abaissé à 4.

Les feuilles et UltraPeers utilisent le protocole de routage de requêtes pour échanger une table de requête de routage (QRT), une table de 64 Ki-slots et jusqu'à 2 Mi-slots composées de mots hachés. Un nœud feuille envoie son QRT à chacun des UltraPeers auxquels il est connecté, et UltraPeers fusionne le QRT de toutes leurs feuilles (Downsized 128 Ki-slots) en plus de leur propre QRT. Le routage des requêtes se fait alors en hachant les mots de la requête et en vérifiant s'ils matchent dans la QRT. Les UltraPeers font cette vérification avant de transmettre une requête à un nœud de feuille, et également avant de transmettre la requête à un autre ultra pair, prévu que c'est la dernière hop la requête peut voyager.

Dans le protocole Gnutella classique, les messages de réponse ont été renvoyés le long de la route par laquelle est venue la requête. Ce régime a été révisé par la suite, de sorte que les résultats de la recherche sont maintenant livrés via UDP (User) directement au nœud qui a initié la recherche, généralement un ultrapeer du nœud. Ainsi, dans le protocole actuel, les requêtes portent l'adresse et numéro de port IP de l'un des nœuds. Cela permet de réduire la quantité de trafic acheminé par le réseau Gnutella, qui la rend beaucoup plus évolutive.

Si l'utilisateur décide de télécharger le fichier, il négocie le transfert du fichier. Si le nœud qui a le fichier demandé n'est pas derrière un pare-feu, le nœud interrogeant peut se connecter directement. Cependant, si le nœud est derrière un pare-feu, le client désirant télécharger le fichier envoie une demande PUSH au client distant pour initier la connexion à la place (pour pousser le fichier). Au début, ces demandes ont été acheminées le long de la chaîne d'origine, utilisée pour envoyer la requête. C'était peu fiable, car les routes étaient souvent cassées et les paquets routés sont toujours soumis au contrôle de flux. Par conséquent des procurations poussoirs ont été introduites. Ceux sont généralement les UltraPeers d'un nœud feuille et ils sont annoncés dans les résultats de recherche. Le client se connecte à l'un de ces procurations poussoirs en utilisant une requête HTTP et le proxy envoie une demande de pousser à la feuille sur le compte du client. Normalement, il est également possible d'envoyer une demande de poussée sur UDP au proxy push qui est le plus efficace que d'utiliser TCP. Les procurations push ont deux avantages: d'abord, les connexions ultrapeer feuilles sont plus stables que les routes, ce qui rend les demandes Push beaucoup plus fiables. Deuxièmement, il réduit la quantité de trafic acheminé par le réseau Gnutella.

Enfin, quand un utilisateur se déconnecte, le logiciel client enregistre la liste de nœuds auxquels il était connecté activement et ceux recueillies à partir des Ping pour une utilisation.

En pratique, cette méthode de recherche sur le réseau Gnutella était souvent peu fiable. [10] Chaque nœud est un utilisateur régulier du réseau p2p; en tant que tels, ils sont constamment en connexion et déconnexion, de tel sorte que le réseau n'est jamais complètement stable. En outre, le coût de la bande passante de la recherche sur Gnutella a connu une croissance exponentielle du nombre d'utilisateurs connectés, les connexions saturant souvent et rendent les nœuds lents inutiles. Par conséquent, les demandes de recherche étaient souvent abandonnées, et la plupart des requêtes ont atteint seulement une très petite partie du réseau. Cette observation a identifié le réseau Gnutella comme un système distribué infranchissable,

et a inspiré le développement de tables de hachage distribuées, qui sont beaucoup plus évolutives, mais ne supportent que exact-match, plutôt que mot-clé, lors de la recherche.

Pour résoudre les problèmes de goulets d'étranglement, les développeurs de Gnutella ont mis en place un système à plusieurs niveaux d'UltraPeers et de feuilles. Au lieu que tous les nœuds soient considérés comme égaux, les nœuds qui entrent dans le réseau ont été conservés au «bord» du réseau comme une feuille, pas responsables de tout routage, et les nœuds qui étaient capables de routage des messages ont été promus à UltraPeers, qui accepterait des connexions avec des feuilles et des recherches d'itinéraires et les messages de maintenance du réseau. Cela a permis aux recherches de se propagent encore à travers le réseau, et a permis de nombreuses modifications dans la topologie qui a amélioré l'efficacité et l'évolutivité grandement.

En outre Gnutella a adopté un certain nombre d'autres techniques pour réduire le trafic frais généraux et faire des recherches plus efficaces. Les plus notables sont le protocole de requête de routage (QRP) et l'interrogation dynamique (DQ). Avec QRP une recherche atteint seulement les clients qui sont susceptibles d'avoir les fichiers, et avec DQ la recherche s'arrête dès que le programme a acquis suffisamment de résultats de recherche, ce qui réduit considérablement la quantité de la circulation causé par des recherches populaires.

Un des avantages d'avoir Gnutella décentralisé est de rendre très difficile de fermer le réseau vers le bas et d'en faire un réseau dans lequel les utilisateurs sont les seuls qui peuvent décider quel contenu sera disponible. Contrairement à Napster, où l'ensemble du réseau s'appuyait sur le serveur central, Gnutella ne peut pas être arrêté en fermant un quelconque nœud et il est impossible pour une entreprise de contrôler le contenu du réseau, en raison des nombreux clients libres et open sources qui partagent le réseau.

b) Caractéristiques du protocole et extensions

Gnutella est basé sur les inondations de requêtes. La version 0.4 du protocole Gnutella emploie cinq types de paquets différents, à savoir [13]

PING: découvrir (activement) les hôtes sur le réseau

PONG: réponse au PING

QUERY: rechercher un fichier

QUERY HIT: réponse au QUERY

PUSH: message envoyé à un servent derrière un pare-feu leur demandant d'initier le téléchargement

Les transferts de fichiers sont traités en utilisant le protocole HTTP.

Le développement du protocole Gnutella est actuellement dirigé par le Forum des développeurs Gnutella (GDF, *Gnutella Developers Forum* en Anglais). De nombreuses extensions de protocoles ont été et sont en cours d'élaboration par les fournisseurs de logiciels et développeurs de Gnutella libres de la GDF. Ces extensions comprennent le routage intelligent des requêtes, total de contrôle SHA-1, la transmission des QueryHit via UDP, l'interrogation via UDP, requêtes dynamiques via TCP, les transferts de fichiers via UDP, XML des métadonnées, l'échange de source (appelé aussi la maille de téléchargement) et le téléchargement parallèle en tranches (essaimage).

Il y a des efforts pour finaliser ces extensions de protocole dans la spécification Gnutella 0,6 sur le site de développement de protocole Gnutella. La norme Gnutella 0,4, bien qu'elle soit toujours la dernière spécification de protocole depuis tous les postes, n'existe que comme des propositions à ce jour, et n'est pas à jour. En fait, il est difficile, voire impossible, de se connecter aujourd'hui avec la poignée de main 0,4 et selon les développeurs dans la GDF, la version 0.6 est ce que de nouveaux développeurs devraient poursuivre en utilisant les spécifications des travaux en cours.

Le protocole Gnutella reste en développement et en dépit des tentatives de faire une rupture nette avec la complexité héritée de l'ancien Gnutella 0,4 et de concevoir une nouvelle architecture de message propre, il est toujours l'un des protocoles de partage de fichiers qui a le plus de succès à ce jour.

3. FastTrack

FastTrack est un (P2P) protocole peer-to-peer qui a été utilisé par les programmes de partage de fichiers Kazaa, Grokster, iMesh, et Morpheus. FastTrack est le réseau de partage de fichiers le plus populaire en 2003, et utilisé principalement pour l'échange de fichiers de musique MP3. Le réseau comptait environ 2,4 millions d'utilisateurs simultanés en 2003. On estime que le nombre total d'utilisateurs était supérieure à celle de Napster à son apogée. Les protocoles FastTrack et Kazaa ont été créés et développés par des programmeurs estoniens de BlueMoon Interactive dirigés par Jaan Tallinn, la même équipe qui a créé plus tard Skype.

a) Technologie

FastTrack utilise des super-nœuds pour améliorer l'évolutivité.Pour permettre le téléchargement à partir de plusieurs sources, FastTrack emploie l'algorithme de hachage UUHash. Alors que UUHash permet à de très gros fichiers d'avoir leur totale de contrôle (checksum) vérifié en peu de temps, même sur les ordinateurs lents, il permet aussi à de la corruption massive d'un fichier de passer inaperçue. Beaucoup de gens, ainsi que la RIAA,

ont exploité cette vulnérabilité pour propager des fichiers corrompus et faux sur le réseau. [15]

Le protocole FastTrack utilise le cryptage et n'a pas été documenté par ses créateurs, et les premiers clients étaient tous les logiciels à code source fermé. Cependant, les données d'initialisation pour les algorithmes de chiffrement est envoyé en clair et aucun chiffrement à clé publique n'est utilisé, l'ingénierie inverse a été faite de manière relativement facile. En 2003, les programmeurs open source ont réussi à inverser la conception de la partie du protocole traitant la communication client-supernœud, mais le protocole de communication supernœud - supernœud reste largement inconnu.

4. BitTorrent

BitTorrent est un protocole pour le partage de fichiers peer-to-peer, utilisé pour distribuer de grandes quantités de données sur Internet. BitTorrent est un des protocoles les plus courants pour transférer des fichiers volumineux, et les réseaux peer-to-peer ont été estimés à environ 43% à 70% de tout le trafic Internet (en fonction de l'emplacement géographique). En Novembre 2004, BitTorrent était responsable de 35% de tout le trafic Internet. [14] En Février 2013, BitTorrent était responsable de 3,35% de toute la bande passante à travers le monde, plus de la moitié des 6% de la bande passante totale dédiée au partage de fichiers.

Pour envoyer ou recevoir des fichiers, l'utilisateur doit avoir un client BitTorrent; un programme informatique qui met en œuvre le protocole BitTorrent. Certains clients BitTorrent populaires sont Xunlei transmission, uTorrent, MediaGet, Vuze et BitComet. Les Trackers BitTorrent fournissent une liste de fichiers disponibles pour le transfert, et aident dans le transfert et la reconstruction des fichiers. Le tracker BitTorrent le plus connu est The Pirate Bay. Des clients BitTorrent sont disponibles pour une variété de plates-formes informatiques et systèmes d'exploitation, publié par BitTorrent, Inc.

En Janvier 2012, BitTorrent est utilisé par 150 millions d'utilisateurs actifs (selon BitTorrent, Inc.). Sur la base de ce chiffre, le nombre total d'utilisateurs mensuels BitTorrent peut être estimé à plus d'un quart de milliard

a) Description

Le protocole BitTorrent peut être utilisé pour réduire l'impact des serveurs et du réseau de distribution de gros fichiers. Plutôt que de télécharger un fichier à partir d'un serveur source unique, le protocole BitTorrent permet aux utilisateurs de se joindre à un "essaim" d'hôtes et à télécharger simultanément. Le protocole est une alternative à la source unique, plusieurs sources (technique de miroir) pour distribuer des données, et pour travailler efficacement sur des réseaux avec une bande passante inférieure. En utilisant le protocole BitTorrent, plusieurs ordinateurs de base, tels que les ordinateurs personnels, peuvent remplacer des grands serveurs tout en distribuant efficacement les fichiers à plusieurs destinataires. Cet usage de

la bande passante inférieure permet également d'éviter de grands pics dans le trafic Internet dans une zone donnée, en gardant à internet des vitesses plus élevées pour tous les utilisateurs en général, indépendamment de savoir si oui ou non ils utilisent le protocole BitTorrent.

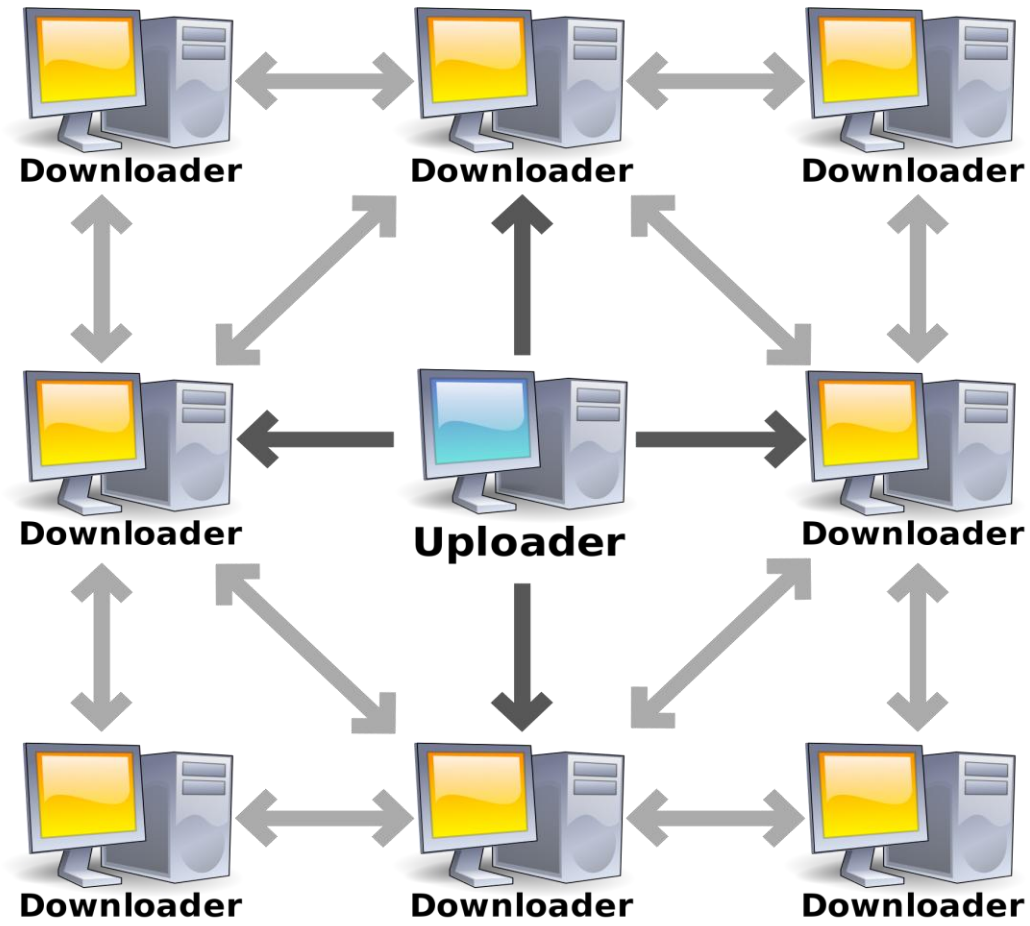


Figure 2.3 : Réseau BitTorrent

Un utilisateur qui veut télécharger un fichier crée d'abord un petit fichier descripteur de torrent qu'il distribue par des moyens classiques (web, email, etc.).

b) Mise en œuvre sectorielle du transfert de fichier

Le fichier à distribuer est divisé en segments appelés pièces. A chaque fois qu'un pair reçoit un nouveau morceau du fichier, il devient une source (de ce morceau) pour les autres pairs, et cela va soulager « la semence originelle » d'avoir à envoyer ce morceau à chaque ordinateur ou utilisateur souhaitant une copie. Avec BitTorrent, la tâche de distribuer le fichier est partagée entre ceux qui le souhaitent; il est tout à fait possible que la semence soit envoyée une seule fois par l'origine et pourtant être distribuée à un nombre illimité de pairs. Chaque pièce est protégée par un hachage cryptographique contenue dans le descripteur de torrent. [20] Cela garantit que toute modification de la pièce peut être détectée de manière fiable, et empêche ainsi les deux modifications accidentelles et malveillantes de l'une des pièces reçues par d'autres nœuds. Si un nœud commence avec une copie authentique du descripteur de torrent, il peut vérifier l'authenticité de l'ensemble du dossier qu'il reçoit.

Les pièces sont généralement téléchargés de façon non-séquentielle et sont réarrangés dans le bon ordre par le client BitTorrent, qui surveille de quelles pièces il a besoin, et de quelles pièces il dispose et peut télécharger des autres pairs. Les pièces sont de la même taille à travers un seul téléchargement (par exemple un fichier de 10 Mo peut être transmis que dix morceaux de 1 Mo ou quarante pièces de 256 Ko). En raison de la nature de cette approche, le téléchargement d'un fichier peut être stoppé à tout moment et être repris à une date ultérieure, sans perte d'informations précédemment téléchargées, qui à son tour rend BitTorrent particulièrement utile dans le transfert de fichiers volumineux. Cela permet également aux clients de rechercher des pièces facilement disponibles et de les télécharger immédiatement, plutôt que d'arrêter le téléchargement et d'attente la prochaine (et peut-être indisponible) pièce successive, ce qui réduit généralement le temps global du téléchargement.

Une fois qu'un pair a téléchargé un fichier complètement, il devient un seeder supplémentaire. Cette transition éventuelle des pairs « à semences » détermine la «santé» globale du fichier (tel que déterminé par le nombre de fois qu'un fichier est disponible dans sa forme complète).

La nature distribuée de BitTorrent peut conduire à une inondation semblable à la diffusion d'un fichier dans de nombreux nœuds d'ordinateurs par les pairs. Comme plusieurs de ses pairs peuvent rejoindre l'essaim, la probabilité d'un téléchargement complètement réussi par tout nœud particulier augmente, relativement aux régimes traditionnels de distribution sur Internet, ce qui permet une réduction significative de matériel et de ressources de bande passante.

Les Protocoles de téléchargement distribués en général assurent la redondance contre les problèmes de système, réduisent la dépendance du distributeur d'origine et fournissent des sources pour le fichier qui sont généralement transitoires et donc plus difficiles à retracer par ceux qui voudraient bloquer la distribution par rapport à la situation prévue en limitant la disponibilité des fichiers à une machine hôte fixe (ou même plusieurs).

Un tel exemple de BitTorrent utilisé pour réduire le coût de distribution de transmission de fichier, est dans le système client-serveur BOINC. Si un calcul distribué BOINC application doit être mis à jour (ou simplement envoyé à un utilisateur), il peut le faire avec peu d'impact sur le serveur BOINC. [18]

c) Opération

Un client BitTorrent est un programme qui implémente le protocole BitTorrent. Chaque client est capable de préparer, et de transmettre tout type de fichier informatique sur un réseau, en utilisant le protocole. Un pair est un ordinateur qui exécute une instance d'un client. Pour partager un fichier ou groupe de fichiers, un pair crée d'abord un petit fichier appelé un "torrent" (par exemple MyFile.torrent). Ce fichier contient des métadonnées sur les fichiers à partager et sur le tracker, l'ordinateur qui coordonne la distribution de fichiers. Les pairs qui veulent télécharger le fichier doivent d'abord obtenir un fichier torrent pour lui et se connecter au tracker spécifié, qui leur dit à partir de quels autres pairs télécharger les pièces du fichier.

Pris ensemble, ces différences permettent à BitTorrent d'atteindre un coût bien inférieur au fournisseur de contenu, la redondance beaucoup plus élevé, et une plus grande résistance à des abus. Toutefois, cette protection, en théorie, a un coût: les téléchargements peuvent prendre du temps avant de passer à pleine vitesse, car il peut prendre du temps pour établir des connexions pairs suffisantes, et il peut prendre du temps pour qu'un nœud reçoive des données suffisantes pour devenir un uploader efficace. Cela contraste avec des téléchargements réguliers (comme à partir d'un serveur HTTP, par exemple) qui, malgré qu'ils sont plus vulnérables à la surcharge et les abus, s'élèvent à pleine vitesse très rapidement et maintiennent cette vitesse tout au longue du téléchargement.

En général, les méthodes de téléchargement non-contigu de BitTorrent ont empêché de soutenir le téléchargement progressif ou «lecture en continu». En 2013, un nouveau protocole de streaming BitTorrent est disponible pour le test bêta.

d) Création et publication de torrents

Le pair distributeur de fichier de données traite le fichier comme un certain nombre de morceaux de taille identique, généralement avec des tailles d'octets d'une puissance de 2, et généralement entre 32 et 16 kB chacun. Le pair crée un hachage pour chaque pièce, en utilisant la fonction de hachage SHA-1, et l'enregistre dans le fichier torrent. Quand un autre pair reçoit plus tard, une pièce en particulier, le hachage de la pièce est comparé au hachage enregistré pour vérifier que la pièce est sans erreur. Les Peers qui fournissent un dossier complet sont appelés *seeders*, et le pair qui fournit la copie initiale est appelé le *seeder* initial.

Les informations exactes contenues dans le fichier torrent dépendent de la version du protocole BitTorrent. Par convention, le nom d'un fichier torrent a le suffixe .torrent. Les Fichiers torrent ont une section "annoncer", qui spécifie l'URL du tracker, et une section "info", contenant les noms des fichiers, leurs longueurs, la longueur de la pièce utilisée, et un code de hachage SHA-1 pour chaque pièce, qui sont utilisés par les clients pour vérifier l'intégrité des données qu'ils reçoivent.

Les Fichiers torrent sont généralement publiés sur des sites Web ou ailleurs, et enregistrés avec au moins un tracker. Le tracker conserve des listes de clients qui participent actuellement dans le torrent. Alternativement, dans un système de traqueur (de suivi décentralisé) tous les pairs agissent comme un tracker. Azureus est le premier des clients BitTorrent pour mettre

en œuvre un tel système par la méthode table de hachage distribuée (DHT). Un système de remplacement et DHT incompatibles, connu sous le nom de Mainline DHT, a ensuite été élaboré et adopté par le BitTorrent (réseau principal), uTorrentTransmission, rTorrent, KTorrent, BitComet, et les clients Déluge.

e) Téléchargement de fichiers torrents et Partage

Les utilisateurs trouvent un torrent d'intérêt, en naviguant sur le Web ou par d'autres moyens, le télécharge et l'ouvre avec un client BitTorrent. Le client se connecte au traqueur(s) spécifié(s) dans le fichier torrent, dont il reçoit une liste de pairs partageant actuellement des morceaux du fichier spécifié dans le torrent. Le client se connecte à ces pairs pour obtenir les différentes pièces. Si l'essaim ne contient que le semoir initial, le client se connecte directement à lui et commence à demander des pièces.

L'arrivée des semences, à leur tour, risque d'être longue à se produire (appelé le problème de la promotion du semoir). Depuis le maintien des semences pour le contenu impopulaire entraîne une bande passante élevée et les coûts administratifs, ce qui va à l'encontre des objectifs des éditeurs qui valorisent BitTorrent comme une alternative pas cher à une approche client-serveur. Cela se produit à grande échelle; des mesures ont montré que 38% de tous les nouveaux torrents deviennent indisponibles dans le premier mois. Une stratégie adoptée par de nombreux éditeurs qui augmente considérablement la disponibilité du contenu impopulaire compose de regrouper plusieurs fichiers dans un seul essaim. Des solutions plus sophistiquées ont également été proposées; généralement, ils utilisent des mécanismes inter-torrent à travers lequel plusieurs torrents peuvent coopérer pour mieux partager du contenu.

BitTorrent n'offre pas à ses utilisateurs l'anonymat. Il est possible d'obtenir les adresses IP de tous les participants actuels et éventuellement précédentes dans un essaim de tracker. Cela peut exposer les utilisateurs avec des systèmes non sécurisés à des attaques. Il peut également exposer les utilisateurs au risque d'être poursuivi, si elles distribuent des fichiers sans la permission du détenteur (s) de droit d'auteur. Cependant, il y a des moyens de promouvoir l'anonymat; par exemple, les couches du projet OneSwarm préservant la vie privée des mécanismes de partage sur le dessus du protocole BitTorrent originale.

5. Conclusion

Les réseaux non-structurés sont (comparativement) faciles à mettre en place, mais cette facilité est délivrée avec un cout. Ces réseaux ne sont pas très efficaces, étant donné leur façon de localiser des ressources/pairs. A cause de cela la mise à l'échelle n'est pas supportée. Dans le chapitre suivant, nous allons passer en revue des architectures de réseaux P2P plus performantes.

Chapitre 3: Les réseaux structurés

1. Introduction

Pour lutter contre le manque d'efficacité des réseaux non-structurés, des chercheurs ont implémenté une interface générique et très révolutionnaire – la table de hachage distribuée (ou *Distributed Hash Table [DHT]* en Anglais). Grace à cette technologie, la mise à l'échelle peut être réalisée dans les systèmes de partage de fichiers pair-à-pair. D'autres avantages par rapport aux réseaux non-structurés sont apportés par cette technologie comme on le verra. Avant de commencer de détailler les systèmes se basant sur la DHT, nous allons d'abord donner un exemple pour comprendre son fonctionnement global.

Supposons qu'un grand nombre d'utilisateurs (5 millions) aient lancé leur logiciel de partage de fichiers en pair à pair (Peer-to-Peer, P2P) sur leur ordinateur. Chacun partage ses fichiers (audio, images, vidéo, multimédia, etc.). Un utilisateur (Luc) possède, par exemple, l'album (fictif) « *42 mon amour* ».

Supposons qu'un autre utilisateur (Pierre) souhaite télécharger cet album. Comment son logiciel de P2P peut-il trouver l'ordinateur de Luc ? Le logiciel de Pierre pourrait éventuellement demander aux 5 millions d'ordinateurs si par hasard ils possèdent cet album. Le logiciel de Luc répondrait alors : « Je le possède et je peux commencer à le transférer. » Il serait cependant très long et très lourd en consommation de ressources de demander aux *5 millions* d'ordinateurs s'ils ont cet album, car il y aurait en permanence des millions de questions du type « Je cherche tel album, l'as-tu ? », entraînant des millions de réponses : « Non, désolé ! ».

Un grand annuaire archivant les noms des fichiers partagés par *tous* les utilisateurs résoudrait la question : Il suffirait de demander à ce « grand annuaire » (la *table de hachage*) l'album de musique *42 mon amour* pour obtenir la réponse : « Il est disponible sur l'ordinateur de Luc. (et celui de Mathieu, de Paul, etc.) ». C'est ainsi que fonctionnait la première génération de réseaux P2P. Il y avait un serveur central qui servait de « grand annuaire. » (Exemples : Napster, Audiogalaxy, Edonkey, Kazaa). Cette solution est de moins en moins utilisée en raison de sa fragilité ; en effet si le serveur central n'est plus disponible, on ne peut plus faire aucune recherche sur les fichiers partagés du réseau.

La table de hachage distribuée apporte donc, par rapport aux techniques précédentes l'indépendance au serveur central en le distribuant sur les différents nœuds.

Par exemple, l'utilisateur Jean-Claude va être responsable de tous les fichiers qui commencent par A, Toto va être responsable de tous les fichiers qui commencent par B, etc... Lorsqu'un

nouvel utilisateur se connecte au réseau, la première chose que le logiciel va faire est de dire quels fichiers il peut partager. S'il possède par exemple le film Big Buck Bunny, il va dire à l'utilisateur Toto (qui est responsable des fichiers qui commencent par B) : « J'ai le film Big Buck Bunny. Si des gens le veulent, il est disponible chez moi. » Les recherches deviennent donc très rapides. Si on cherche Big Buck Bunny, on va directement demander à la personne responsable de la lettre « B ».

La réalité est un peu plus complexe : il ne faut pas qu'une seule personne soit responsable des mots qui commencent par "B", car si elle éteint son ordinateur on perd une partie de l'annuaire. Il faut donc introduire une certaine redondance dans l'annuaire, et donc plusieurs ordinateurs sont simultanément responsables des mêmes listes. De plus, vu qu'il y a des centaines de millions de fichiers partagés, le principe de division de l'annuaire n'est pas basé sur les lettres de l'alphabet mais sur une table de hachage des mots des titres des fichiers ou de mots clé.

Enfin, *chaque* ordinateur n'a pas besoin de connaître *tous* les ordinateurs qui archivent des mots. Il connaîtra typiquement une centaine d'ordinateurs. Si l'utilisateur fait une recherche sur Big Buck Bunny et ne connaît pas l'ordinateur qui archive les fichiers commençant par B, alors :

- il demandera à l'ordinateur le plus proche (par exemple l'ordinateur qui archive les fichiers commençant par C) : « Connais-tu l'ordinateur s'occupant des mots commençant par B ? »
- celui-ci répondra « Parmi mes voisins, je connais les ordinateurs qui s'occupent des B, et même je connais des ordinateurs qui s'occupent des fichiers commençant par BA, BI, BO, BU, donc tu ferais bien de demander à celui qui connaît les fichiers commençant par BI s'il a par hasard le fichier que tu cherches. »
- on interroge le responsable des « BI » et il dira : « Oui, je connais les ordinateurs qui ont le film que tu veux » ou alors, s'il ne les connaît pas, il répondra « Je ne connais pas ton fichier, par contre je connais un ordinateur qui s'occupe des fichiers commençant par BIG, donc demande-lui. »

Et ainsi de proche en proche, Pierre va trouver l'information qu'il cherche et les plusieurs endroits où elle est stockée, de manière rapide, efficace et sure, sans inonder le réseau et le surcharger.

2. Chord

Chord est un protocole et algorithme pour une table de hachage distribuée Pair-à-Pair. Une table de hachage distribuée stocke des paires clé-valeur en attribuant des clés pour différents ordinateurs (appelés «nœuds»); un nœud va stocker les valeurs de toutes les clés pour lesquelles il est responsable. Chord spécifie comment les clés sont attribuées aux nœuds, et la manière dont un nœud peut découvrir la valeur pour une clé donnée en localisant d'abord le nœud responsable de cette clé. [21]

a) Aperçu

Les nœuds et les clés sont affectés à un identificateur à m bits en utilisant le hachage cohérent. L'algorithme SHA-1 est la fonction de hachage de base pour le hachage cohérent. Le hachage cohérent fait partie intégrante de la robustesse et les performances de Chord parce que les clés et les nœuds (en fait, leurs adresses IP) sont toutes les deux uniformément réparties dans le même espace d'identification avec une possibilité négligeable de collision. Ainsi, il permet également aux nœuds de rejoindre et quitter le réseau, sans interruption. [21] Dans le protocole, le terme nœud est utilisé pour désigner à la fois un nœud lui-même et son identifiant (ID) sans ambiguïté. Le terme clé également.

b) La Fonction de distance de Chord

Chord calcule la distance entre deux pairs en soustrayant le GUID numériquement l'une de l'autre. L'espace GUID est fait pour rouler, ce qui signifie que dans un espace GUID à 4 bits (valeurs de 0 à 15), la distance de 15 à 0 est défini comme 1. La distance 15-1 est 2, etc. La distance de 0 à 15 est 15, en revanche. Comme nous pouvons voir la distance de A à B est pas la même que la distance de B à A.

La fonction de distance Chord peut être définie un peu plus formellement comme ceci:

Distance (A, B) = B - A (B> = A)

= B - A + 2^N (B <A)

Ceci peut également être calculé comme

Distance (A, B) = (B - A +2^N) mod 2^N

Par exemple:

Distance (0, 11) = (11 - 0 + 16) mod 16 = (27) mod 16 = 11

Distance (11, 0) = (0-11 + 16) mod 16 = (5) mod 16 = 5

Comme nous pouvons le voir, la distance (A, B) n'est pas égale à la distance (B, A).

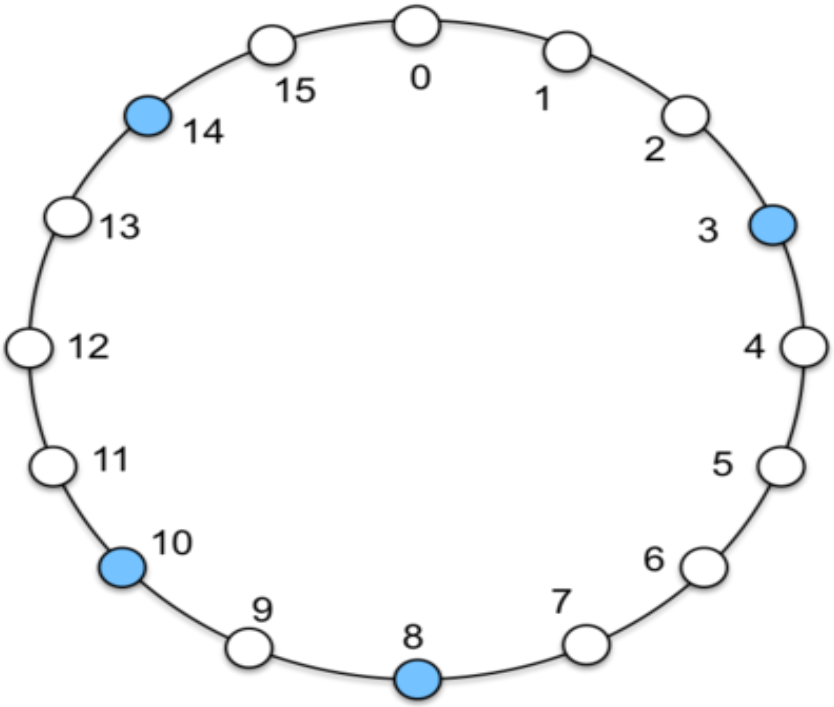


Figure 3.1 : Réseau Chord

En utilisant le protocole de recherche Chord, les nœuds et les clés sont agencées dans un cercle d'identificateur qui a au plus 2^m nœuds, allant de 0 à 2^m - 1. (m devrait être suffisamment grand pour éviter la collision.)

Chaque nœud a un successeur et un prédécesseur. Le successeur d'un nœud est le nœud suivant dans le cercle d'identificateurs dans le sens des aiguilles d'une montre. Le prédécesseur est le sens antihoraire. S'il y a un nœud pour chaque ID possible, le successeur du nœud 0 est le nœud 1, et le prédécesseur du nœud 0 est nœud 2^m - 1 ; Toutefois, normalement il y a des "trous" dans la séquence. Par exemple, le successeur du nœud 153 peut être le nœud 167 (et les nœuds 154-166 n'existent pas); dans ce cas, le prédécesseur du nœud 167 sera le nœud 153.

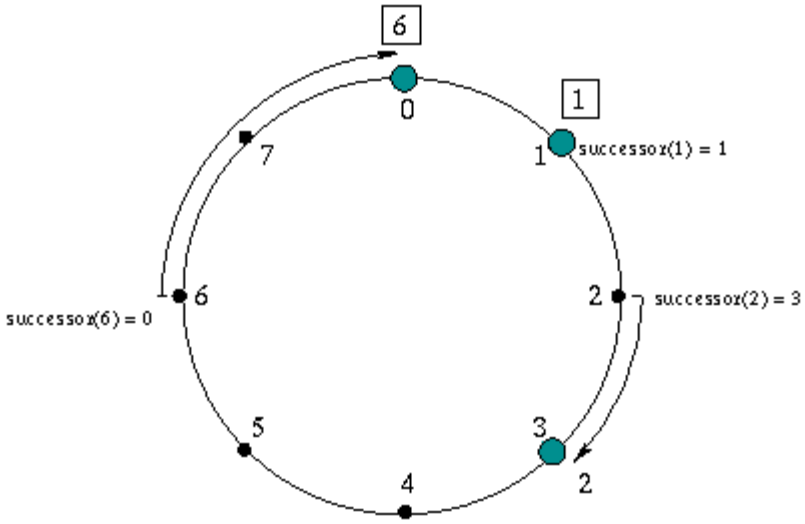


Figure 3.2 : Concept de successeur dans Chord

Le concept de successeur peut être utilisé pour les clés aussi bien. Le nœud successeur d'une clé k est le premier nœud dont l'identifiant est égal à k ou suit k dans le cercle d'identificateur, noté *successeur(k)*. Chaque clé est affectée à (stocké à) son nœud successeur, afin regardant une clé k est d'interroger *successeur(k)*.

Comme le successeur (ou son prédécesseur) d'un nœud peut disparaître du réseau (en raison de l'échec ou de départ), chaque nœud enregistre tout un segment du cercle à côté de lui, à savoir les r nœuds qui le précède et les r nœuds qui suivent. Cette liste se traduit par une forte probabilité qu'un nœud est capable de localiser correctement son successeur ou son prédécesseur, même si le réseau en question souffre d'un taux d'échec élevé.

c) Les détails du protocole

Dans un réseau de 16 nœuds, les nœuds sont agencés en un cercle. Chaque nœud est relié à d'autres nœuds à des distances 1, 2, 4, 8.

Le Finger table

Pour éviter la recherche linéaire ci-dessus, Chord met en œuvre une méthode de recherche plus rapide en demandant à chaque nœud de maintenir une table de doigt contenant jusqu'à m entrées. Le ième entrée de nœud n contiendra successeur $((n+2^{i-1}) \bmod 2^m)$. La première entrée de la table de doigt est effectivement successeur immédiat du nœud (et donc un champ successeur supplémentaires ne sont pas nécessaires). Chaque fois qu'un nœud veut récupérer une clé k, Il va passer la requête au plus proche successeur de k dans sa table de doigt (le

"plus grand" un sur le cercle dont l'ID est plus petit que k), Jusqu'à ce qu'un nœud découvre que la clé est stockée chez son successeur immédiat.

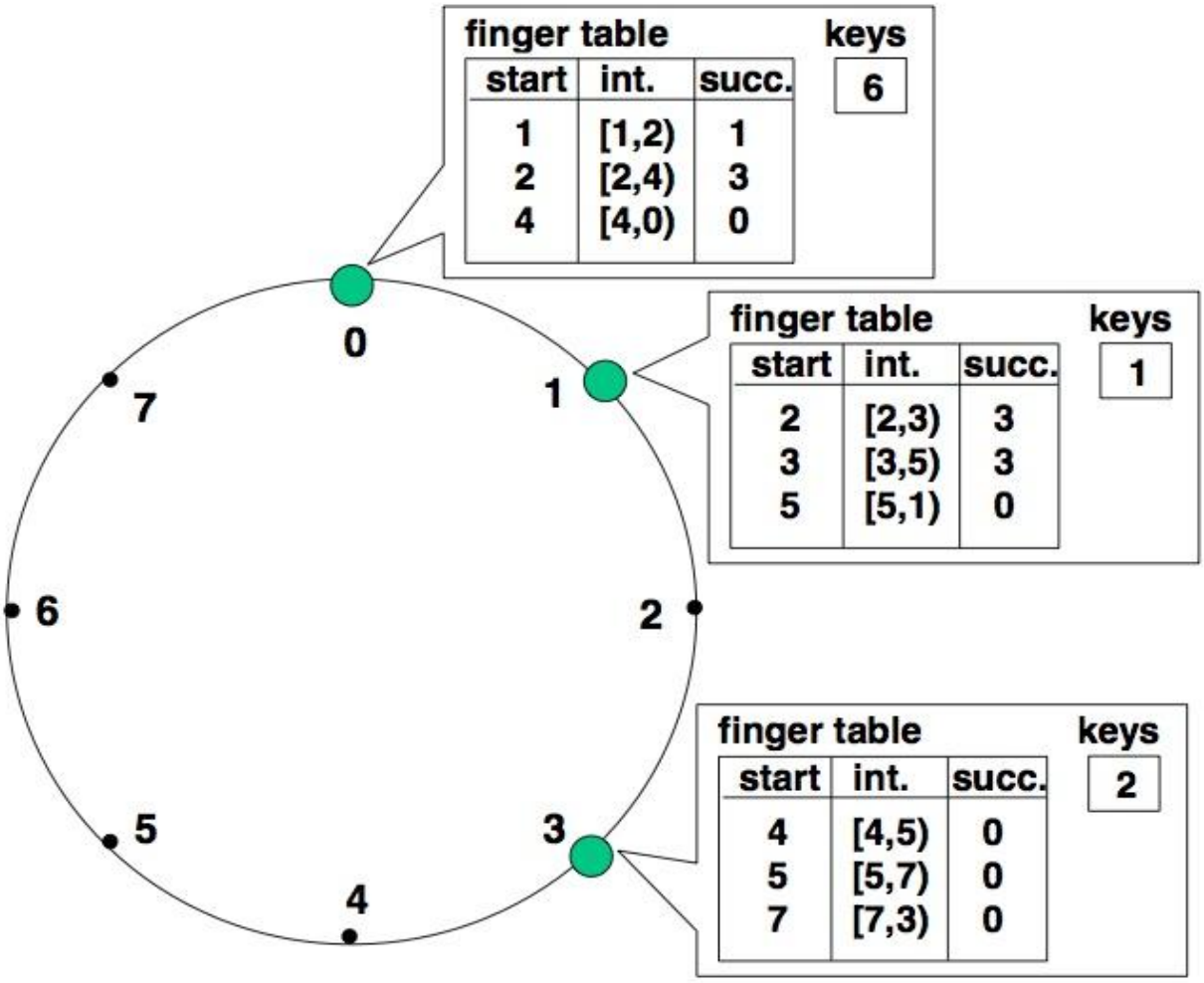


Figure 3.3 : Finger Table

Avec un tel tableau de doigt, le nombre de nœuds qui doivent être contactés pour trouver un successeur dans un réseau à N nœuds est O(log N).

d) Départ de nœuds et tolérance aux pannes

Lors du départ d'un nœud du système, il remet ses clés à son successeur et il informe également son prédécesseur pour que celui-ci puisse changer son pointeur successeur. Le processus de stabilisation met à jour les tables de repérage comme pour le cas de l'ajout.

Le problème est un peu plus corsé lorsqu'un nœud tombe en panne. Si le nœud k tombe en panne, les nœuds contenant k dans leur table de repérage doivent pouvoir retrouver le successeur de k. Notamment, le prédécesseur de k n'a plus de successeur valide.

L'idée consiste à conserver, en plus de son successeur immédiat, une liste des r successeurs directs. Cela permet à un nœud de sauter par-dessus r – 1 nœuds contigus défaillants et de continuer sa participation sur le cercle. Alors, la probabilité que r nœuds soient simultanément en panne est $\boldsymbol{p^r}$, p étant la probabilité qu'un nœud soit en panne.

On a alors les propriétés suivantes (preuves dans les articles de recherche).

Soit r = O(logN) et un système initialement stabilisé. Si chaque nœud tombe en panne avec une probabilité$\frac{1}{2}$, alors

– avec une forte probabilité la recherche renvoie le plus proche successeur en vie de la clé recherchée;

– le temps d'exécution pour effectuer cette recherche est en O(logN).

L'utilisation de la liste de successeurs permet aussi à un logiciel de plus haut niveau de dupliquer les données. Les applications généralement dupliquent les informations sur k nœuds suivant la clé (pour éviter la perte d'information lors de panne), et la liste des successeurs permet de savoir quand des nœuds partent et arrivent, pour pouvoir générer au besoin des nouveaux répliquas de l'information. [21]

3. Kademlia

Kademlia est une table de hachage distribuée pour les réseaux informatiques décentralisés peer-to-peer, conçu par Petar Maymounkov et David Mazières en 2002. [1] Il spécifie la structure du réseau et l'échange d'informations à travers la consultation de nœuds. Les nœuds Kademlia communiquent entre eux en utilisant UDP. Un réseau virtuel ou superposition est formé par les nœuds participants. Chaque nœud est identifié par un numéro identifiant - l'ID du nœud. L'ID de nœud sert non seulement d'identification, mais l'algorithme Kademlia l'utilise pour localiser des valeurs (généralement des hachages ou de mots-clés). En fait, l'ID de nœud fournit une carte directe aux haches et ce nœud stocke les informations sur l'emplacement actuel du fichier ou de la ressource.

Lors de la recherche d'une certaine valeur, l'algorithme a besoin de connaître la clé associée et il explore le réseau en plusieurs étapes. Chaque étape trouvera les nœuds qui sont plus près de la clé jusqu'à ce que le nœud recherché renvoie la valeur. Ceci est très efficace: Comme beaucoup d'autres DHT, Kademlia ne contacte que O (log(n)) nœuds au cours de la recherche sur un total de n nœuds dans le système.

D'autres avantages se trouvent en particulier dans la structure décentralisée, ce qui augmente la résistance contre un déni de service. Même si un ensemble de nœuds est inondé, cela aura

un effet limité sur la disponibilité du réseau, car le réseau lui-même se récupérera en tricotant le réseau autour de ces "trous".

a) Détails du système

Les réseaux peer-to-peer de partage de fichiers de la première génération, tels que Napster, s'appuyaient sur une base de données centrale pour coordonner les recherches sur le réseau. Les réseaux peer-to-peer de deuxième génération, tels que Gnutella, utilisaient l'inondation pour localiser des fichiers, cela en consultant chaque nœud sur le réseau. Les réseaux peer-to-peer de la troisième génération utilisent des tables de hachage décentralisées pour rechercher les fichiers dans le réseau. Les Tables de hachage distribuées stockent les adresses de ressources à travers le réseau. Un critère majeur pour ces protocoles est de localiser rapidement les nœuds souhaités.

Kademlia utilise un calcul "distance" entre deux nœuds. Cette distance est calculée en faisant le **ou exclusif** des ID des deux nœuds, et en prenant le résultat sous forme de nombre entier. Les clés et ID nœud ont le même format et la même longueur, de sorte que la distance peut être calculée entre eux exactement de la même façon. L'ID du nœud est généralement un grand nombre aléatoire qui est choisi dans le but d'être unique pour un nœud particulier (voir UUID). Il se peut que les nœuds qui sont géographiquement très éloignés : l'Allemagne et l'Australie, par exemple, soient des «voisins» si elles ont choisi l'ID de nœuds aléatoire similaires.

Le ou exclusif a été choisi car il agit comme une fonction de distance entre l'ensemble des identifiants de nœuds. Plus précisément:

- La distance entre un nœud et lui-même est égale à zéro
- Il est symétrique: les "distances" calculées à partir de A vers B et de B vers A sont les mêmes
- Il respecte l'inégalité triangulaire: donnée A, B et C sont des sommets (points) d'un triangle, puis la distance de A à B est plus courte que (ou égale) à la somme de la distance de A à C, et la distance à partir de C pour B.

Ces trois conditions sont suffisantes pour assurer que le « ou exclusive » capture toutes les caractéristiques essentielles, importantes d'une fonction de distance "réelle", tout en étant pas cher et simple à calculer. [1] Chaque itération de recherche Kademlia arrive un bit plus près de la cible. Un réseau Kademlia de base avec 2^n nœuds ne prendra que n étapes (dans le pire des cas) pour trouver ce nœud.

b) Les Tables de routage

Les tables de routage Kademlia sont constituées d'une liste pour chaque bit de l'identificateur de nœud. (Par exemple, si un ID de nœud est constitué de 128 bits, un nœud va garder 128 listes). Une liste a de nombreuses entrées. Chaque entrée dans une liste contenant les données nécessaires pour localiser un autre nœud. Les données de chaque entrée de la liste est généralement l'adresse IP, le port et l'ID d'un autre nœud. Chaque liste correspond à une

distance spécifique à partir du nœud. Les Nœuds qui se trouvent dans la nième liste doivent avoir un bit différent de la nième ID du nœud; les n-1 premiers bits de l'ID du candidat doivent correspondre à ceux de l'ID du nœud. Cela signifie qu'il est très facile de remplir la première liste, car seulement 1/2 des nœuds dans le réseau sont loin du candidat. La liste suivante ne peut utiliser que 1/4 des nœuds dans le réseau (un peu plus près que la première), etc.

Avec un ID de 128 bits, chaque nœud du réseau classe les autres nœuds dans l'un des 128 distances différentes, une distance spécifique par bit.

Dès qu'on rencontre des nœuds sur le réseau, ils sont ajoutés à la liste. Cela nécessite un magasin et les opérations de récupération et même aider les autres nœuds à trouver une clé. Chaque nœud rencontré sera considéré pour une inclusion dans les listes. Par conséquent, la connaissance que possède un nœud du réseau est très dynamique. Cela permet de maintenir le réseau constamment mis à jour et ajoute une grande résistance aux pannes ou attaques.

Dans la littérature de Kademlia, les listes sont appelées k-seau, k est un paramètre système, un grand nombre comme 20. Chaque k-seau est une liste comportant jusqu'à k entrées à l'intérieur; c'est à dire que pour un réseau avec k = 20, chaque nœud aura des listes contenant jusqu'à 20 nœuds pour un bit particulier (une distance particulière de lui-même).

Comme les nœuds possibles pour chaque k-seau diminue rapidement (car il y aura très peu de nœuds qui sont si près), les k-seau des bits inférieurs mapperont pleinement tous les nœuds dans cette section du réseau. Comme la quantité d'identifiants possibles est beaucoup plus grande que toute population de nœuds ne pourrait jamais être, certains des k-seaux pour de très courtes distances resteront vides.

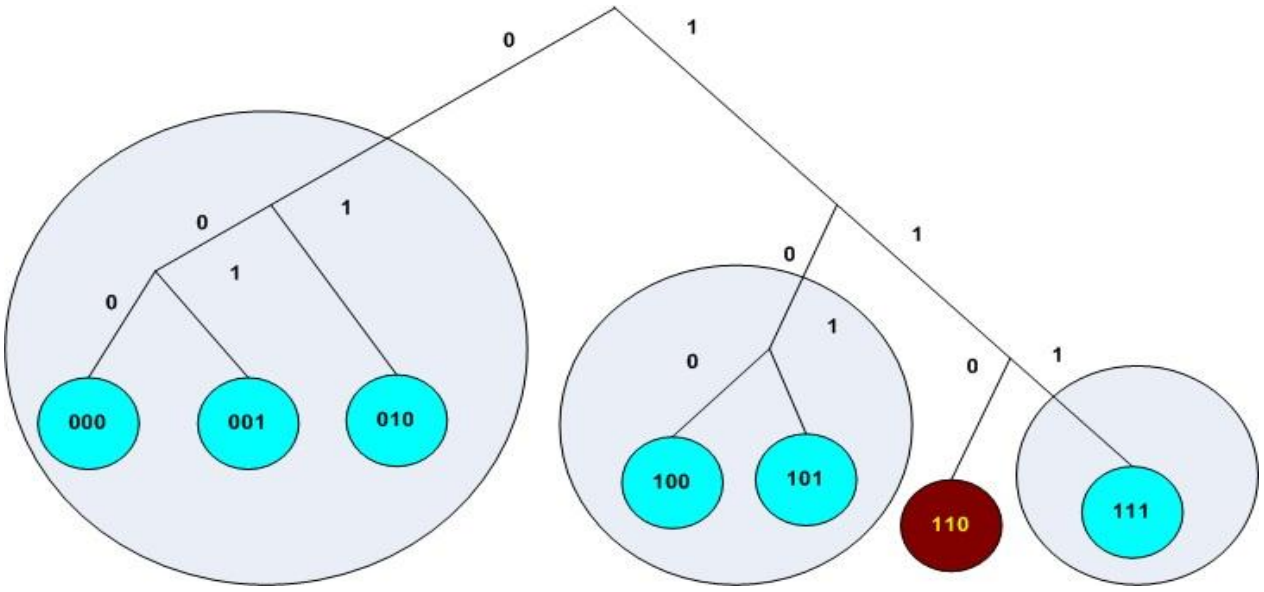


Figure 3.4 : Partition du réseau pour le nœud 110

Considérons le réseau simple ci-dessus. La taille du réseau est égale à 2^3ou 8 clés et nœuds au maximum. Il y a sept nœuds participants - les petits cercles au fond. Le nœud que nous considérons c'est le nœud six (en binaire 110) en rouge. Il a trois k-seau pour chaque nœud de ce réseau. Les Nœuds zéro, un et deux (en binaire 000, 001 et 010) sont des candidats à la k-seau la plus éloignée. Le Nœud trois (en binaire 011, non illustré) ne participe pas dans le réseau. Dans le k-seau du milieu, les nœuds quatre et cinq (binaire 100 et 101) sont placés. Enfin, le troisième k-seau ne peut contenir que le nœud sept (binaire 111). Chacune des trois k-seau est enfermée dans un cercle gris. Si la taille de la k-seau était deux, puis le plus loin 2-seau ne peut contenir que deux des trois nœuds. Par exemple si le nœud six avait deux dans la plus éloignée 2-seau, il faudrait demander une recherche d'ID de nœud à ces nœuds pour trouver l'emplacement (adresse IP) du nœud zéro. Chaque nœud sait bien son quartier et est en contact avec quelques nœuds loin qui peuvent l'aider à localiser les autres nœuds éloignés.

Il est connu que les nœuds qui se sont connectés pendant une longue période dans un réseau resteront probablement connectés pendant une longue période dans le futur. [19] En raison de cette répartition statistique, Kademlia sélectionne les nœuds longuement connectés pour rester stockés dans les k-seau. Cela augmente le nombre de nœuds valides connus à un moment donné dans l'avenir et prévoit un réseau plus stable.

Quand une k-seau est pleine et un nouveau nœud est découvert pour cette k-seau, le nœud moins récemment vu dans la k-seau est remplacé. Si le nœud se trouve être toujours en vie, le nouveau nœud est placé dans une liste secondaire, un cache de remplacement. Le cache de remplacement est utilisé uniquement si un nœud dans la k-seau cesse de répondre. En d'autres termes: les nouveaux nœuds ne sont utilisés que lorsque les nœuds anciens disparaissent.

c) Les messages du protocole

Kademlia a quatre messages :

1) PING - utilisé pour vérifier qu'un nœud est toujours vivant.
2) STORE - Enregistre une paire (clé, valeur) dans un nœud.
3) FIND_NODE - Le destinataire de la demande retournera les nœuds k dans ses propres sauts qui sont les plus proches à la clé demandée.
4) FIND_VALUE - Même que FIND_NODE, mais si le destinataire de la demande a la clé demandée dans son magasin, il retournera la valeur correspondante.

Chaque message RPC comprend une valeur aléatoire de l'initiateur. Cela garantit que lorsque la réponse est reçue, elle correspond à la demande envoyée précédemment.

d) Localisation des nœuds

La recherche de nœuds peut procéder de manière asynchrone. La quantité de recherches simultanées est désignée par α et est généralement de trois. Un nœud lance une demande de FIND_NODE en interrogeant les α nœuds dans ses propres k-seau qui sont les plus proches à la clé souhaitée. Lorsque ces nœuds destinataires reçoivent la demande, ils vont chercher dans leurs k-seau et retourner les k nœuds les plus proches à la clé désirée dont ils ont connaissance. Le demandeur met à jour une liste de résultats avec les résultats (nœud de ID) qu'il reçoit, en gardant les k meilleurs (les k nœuds qui sont plus près de la clé recherché) qui répondent aux requêtes. Ensuite, le demandeur choisira ces k meilleurs résultats et émettre la demande à eux, et itérera encore et encore ce processus. Parce que chaque nœud a une meilleure connaissance de son propre environnement que tout autre nœud, les résultats reçus seront d'autres nœuds qui sont chaque fois plus proche de la clé recherchée. Les itérations se poursuivent jusqu'à ce qu'il n'y ait pas de nœuds retournés qui sont plus proches que les meilleurs résultats précédents. Lorsque les itérations arrêtent, les meilleurs k nœuds dans la liste des résultats sont ceux de l'ensemble du réseau qui sont les plus proches de la clé souhaitée.

Quand une requête atteint son RTT, une autre requête peut être lancée, ne dépassant jamais les requêtes α dans le même temps.

e) Localisation des ressources

L'information est localisée en la mappant à une clé. Un hachage est généralement utilisé pour ce mappage. Les nœuds stockeurs auront des informations grâce au un message STORE précédent. La localisation d'une valeur suit la même procédure que la localisation des nœuds les plus proches d'une clé, sauf que la recherche se termine quand un nœud a la valeur demandée dans son magasin et renvoie cette valeur.

Les valeurs sont stockées dans plusieurs nœuds (k nœuds) pour permettre les nœuds d'aller et de venir tout en assurant que la valeur reste toujours disponible dans un nœud. Périodiquement, un nœud qui stocke une valeur va explorer le réseau pour trouver les k nœuds qui sont proches de la valeur de clé et de reproduire la valeur sur eux. Cela compensera les nœuds disparus.

Aussi, pour des valeurs populaires qui pourraient avoir de nombreuses demandes, la charge dans les nœuds stockeurs est diminuée en stockant cette valeur dans un nœud près, mais en dehors des k plus proches de sa clé. Ce nouveau stockage est appelé un cache. De cette manière, la valeur est stockée plus en plus loin de la clé, en fonction de la quantité de demandes. Cela permet aux recherches populaires de trouver un stockeur plus rapidement. Comme la valeur est renvoyée à partir des nœuds plus éloignés de la clé, cela évite les "points chauds" possibles. Les nœuds de cache supprimeront la valeur après un certain temps en fonction de leur distance de la clé.

Certaines implémentations (par exemple Kad) ne disposent pas de réplication, ni la mise en cache. Le but de cela est de supprimer les informations anciennes rapidement du système. Le

nœud qui fournit le fichier actualisera périodiquement les informations sur le réseau (effectuer messages FIND_NODE et STORE). Lorsque tous les nœuds possédant le fichier sont hors-ligne, personne ne rafraîchira encore ses valeurs (sources et mots-clés) et l'information finira par disparaître du réseau.

f) Rejoindre le réseau

Un nœud qui aimerait se connecter au réseau doit d'abord passer par un processus de bootstrap. Dans cette phase, le nouveau nœud a besoin de connaître l'adresse IP et le port d'un autre nœud – le nœud de bootstrap (obtenu à partir de l'utilisateur, ou à partir d'une liste stockée) - qui participe déjà dans le réseau Kademlia. Si le nœud joignant n'a pas encore participé dans le réseau, il calcule un numéro d'identification aléatoire qui est censé ne pas être déjà affectée à un autre nœud. Il utilise cet ID jusqu'à ce qu'il quitte le réseau.

Le nouveau nœud insère le nœud bootstrap dans l'un de ses k-seau. Puis le nouveau nœud lance un FIND_NODE de son propre ID vers le nœud bootstrap (le seul autre nœud qu'il connait). [17] Le «self-lookup" peuplera les k-seau d'autres nœuds avec l'ID du nouveau nœud, et peuplera les k-seau du nouveau nœud avec les nœuds dans le chemin entre lui et le nœud bootstrap. Après cela, le nouveau nœud actualise tous les k-seau plus loin que celui dans lequel se trouve le nœud bootstrap. Cette actualisation est juste la recherche d'une clé aléatoire qui est dans les environs de ces k-seau.

Initialement, les nœuds ont une k-seau. Lorsque la k-seau est pleine, elle peut être divisée. La scission se produit si la gamme de nœuds dans le k-seau couvre l'ID du nœud (des valeurs à gauche et à droite dans un arbre binaire). Kademlia utilise même cette règle pour trouver les nœuds k-seaus les « plus proches", parce que généralement un saut va contenir tous les nœuds qui sont le plus proche de lui, ils peuvent être plus de k, et nous voulons qu'il les connaisse tous. Si k est égal à 20, et il y a plus que 21 nœuds avec un préfixe "xxx0011 ....." et le nouveau nœud est "xxx000011001", le nouveau nœud peut contenir plusieurs k-seau pour les +21 autres nœuds. Cela permet de garantir que le réseau est au courant de tous les nœuds dans la région la plus proche.

g) Les recherches accélérées

Kademlia utilise une métrique XOR pour définir la distance. Deux identifiants de nœud ou un ID de nœud et une clé sont « XORés » et le résultat est la distance entre eux. Pour chaque bit, la fonction XOR retourne zéro si les deux bits sont égaux et un si les deux bits sont différents. Les distances métriques XOR maintiennent l'inégalité triangulaire: soient A, B et C des sommets (points) d'un triangle, alors la distance de A à B est plus courte que (ou égale) à la somme de la distance de A à C à B.

La métrique de XOR permet à Kademlia d'étendre les tables de routage pour dépasser des bits uniques. Des groupes de bits peuvent être placés dans des k-seaux. Le groupe de bits est nommé un préfixe. Pour un préfixe à m bits, il y aura 2^m-1 k-seaux. La k-seau manquant est une extension de l'arbre de routage qui contient l'ID de nœud. Un préfixe à m bits réduit le nombre maximal de recherches de log_2 n à n log_{2^m}n. Ce sont des valeurs maximales et la

valeur moyenne sera beaucoup moins, ce qui augmente les chances de trouver un nœud dans un k-seau qui partage plus de bits que juste le préfixe avec la clé cible.

Les nœuds peuvent utiliser des mélanges de préfixes dans leurs tables de routage, tels que le Réseau Kad utilisés par eMule. Le réseau Kademlia pourrait même être hétérogène dans les implémentations de la table de routage, au détriment de la complexité d'analyse des recherches.

h) Utilisation dans les réseaux de partage de fichiers

Kademlia est utilisé dans les réseaux de partage de fichiers. En faisant des recherches par mots clés, on peut trouver de l'information dans le réseau de partage de fichiers de sorte qu'elle peut être téléchargée. Comme il n'y a pas d'instance centrale pour stocker un index des fichiers existants, cette tâche est divisée de façon égale entre tous les clients: Si un nœud veut partager un fichier, il traite le contenu du fichier, et il calcule de celui-ci un certain nombre (hachage) qui identifiera ce fichier au sein du réseau de partage de fichiers. Les hachages et les identifiants de nœuds doivent être de la même longueur. Il recherche ensuite plusieurs nœuds dont l'ID est proche de la valeur de hachage, et leurs envoie sa propre adresse IP pour y être stockée. C'est à-dire qu'il se publie en tant que source de ce fichier. Un client qui recherche utilisera Kademlia pour rechercher sur le réseau le nœud dont l'ID a la plus petite distance au hachage de fichier, puis il récupérera la liste des sources qui est stockée dans ce nœud.

Etant donné qu'une clé peut correspondre à plusieurs valeurs, par exemple de nombreuses sources d'un même fichier, chaque nœud de stockage peut avoir des informations différentes. Dans ce cas, les sources sont demandées de tous les k nœuds proches de la clé.

La hache de fichier est généralement obtenu à partir d'un lien Internet «magnet» spécialement formé et trouvé ailleurs, ou inclue dans un fichier d'indexation obtenu à partir d'autres sources.

Les Recherches de noms de fichier sont mises en œuvre en utilisant des mots-clés. Le nom de fichier est divisé en ses mots constitutifs. Chacun de ces mots-clés est haché et stocké dans le réseau, ainsi que le nom du fichier correspondant et le fichier hachage. Une recherche consiste à choisir l'un des mots-clés, en contact avec le nœud avec un ID plus proche de ce mot clé de hachage, et la récupération de la liste des noms de fichiers qui contiennent les mot-clé. Étant donné que chaque nom de fichier dans la liste ci-jointe a son hachage, le fichier choisi peut alors être obtenu de la manière habituelle.

i) Implémentations

Les réseaux publics utilisant l'algorithme Kademlia sont:

KadNetwork, le Réseau Overnet, BitTorrent utilise une DHT basée sur une implémentation de l'algorithme Kademlia, pour les torrents de traqueur.

SPS OsirisRetroshare

Gnutella DHT

j) Comparaison entre les tables de routage de Chord et de Kademlia

Peer GUID		**2^0=1**		**2^1=2**		**2^2=4**		**2^3=8**	
Chord	**Kad**	**Chord**	**Kad**	**Chord**	**Kad**	**Chord**	**Kad**	**Chord**	**Kad**
0	**0**	1	1	2	2	4	4	8	8
1	**1**	2	0	3	3	5	5	9	9
2	**2**	3	3	4	0	6	4	10	10
3	**3**	4	2	5	1	7	7	11	11
4	**4**	5	5	6	6	8	0	12	12
5	**5**	6	4	7	7	9	1	13	13
6	**6**	7	7	8	4	10	2	14	14
7	**7**	8	6	9	5	11	3	15	15
8	**8**	9	9	10	10	12	12	0	0
9	**9**	10	8	11	11	13	13	1	1
10	**10**	11	11	12	8	14	14	2	2
11	**11**	12	10	13	9	15	15	3	3
12	**12**	13	13	14	14	0	8	4	4
13	**13**	14	12	15	15	1	9	5	5
14	**14**	15	15	0	12	2	10	6	6
15	**15**	0	14	1	13	3	11	7	7

Tableau 2.1 : Chord vs Kademlia

Le tableau ci-dessus montre les tables de routage complètes de pairs dans un réseau P2P ayant un espace GUID à 4 bits, en utilisant soit Chord soit Kademlia pour calculer les distances. Les lignes montrent le GUID du pair en gras, puis le contenu de sa table de routage (4 entrées par table)

L'organigramme ci-dessous montre l'algorithme général de localisation des nœuds dans les DHTs :

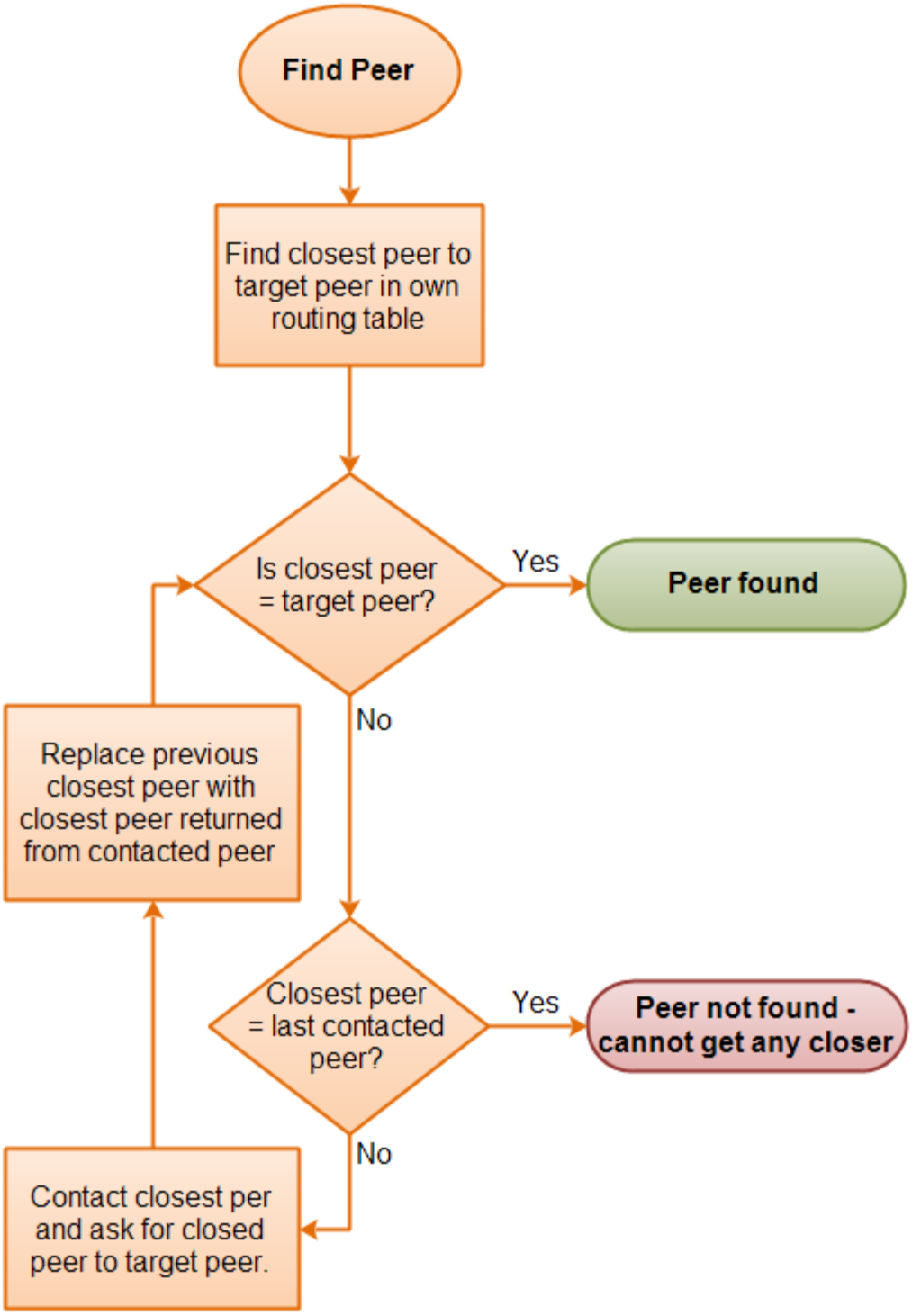


Figure 3.5: Organigramme du processus de localisation des pairs

4. CAN (Content Addressable Network)

CAN est une infrastructure P2P décentralisée et distribuée qui fournit des propositions des tables de hachage distribuées sur une échelle comme celle de l'internet. Il était l'une des quatre propositions originales des tables de hachage distribuées, introduite concurremment avec Chord, Pastry, et Tapestry.

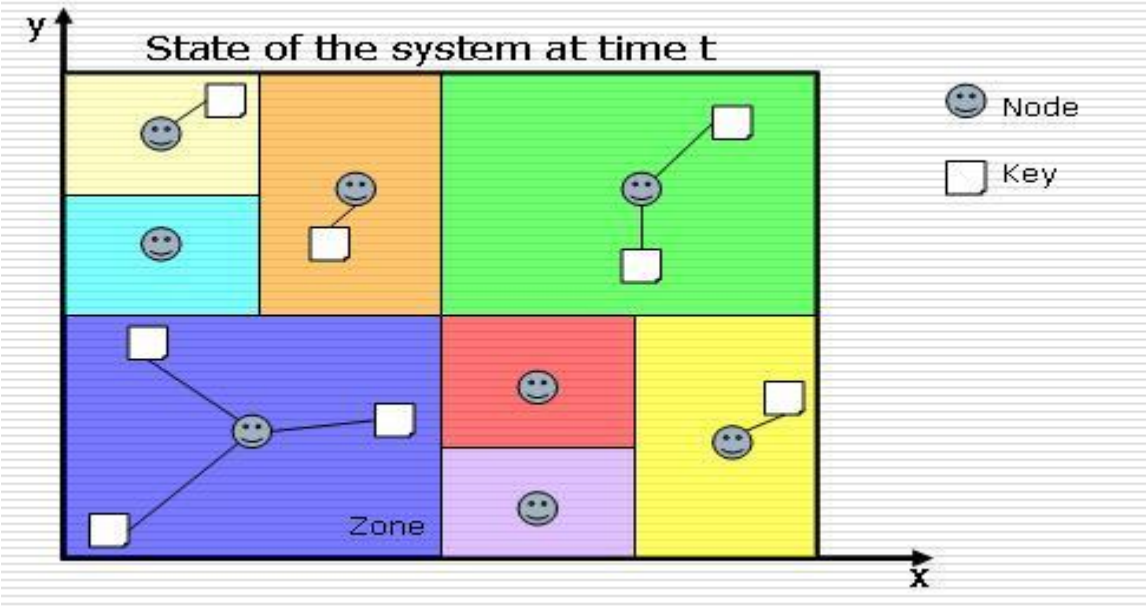


Figure 3.6 : Un réseau CAN

a) Routage

Un nœud CAN maintient une table de routage qui contient l'adresse IP et la zone cartésienne virtuelle de chacun de ses voisins. Un nœud route un message vers un point destinataire dans l'espace cartésien. Le nœud détermine d'abord quelle zone voisine est plus proche du point de destination, puis regarde l'adresse IP du nœud de cette zone via la table de routage. [16]

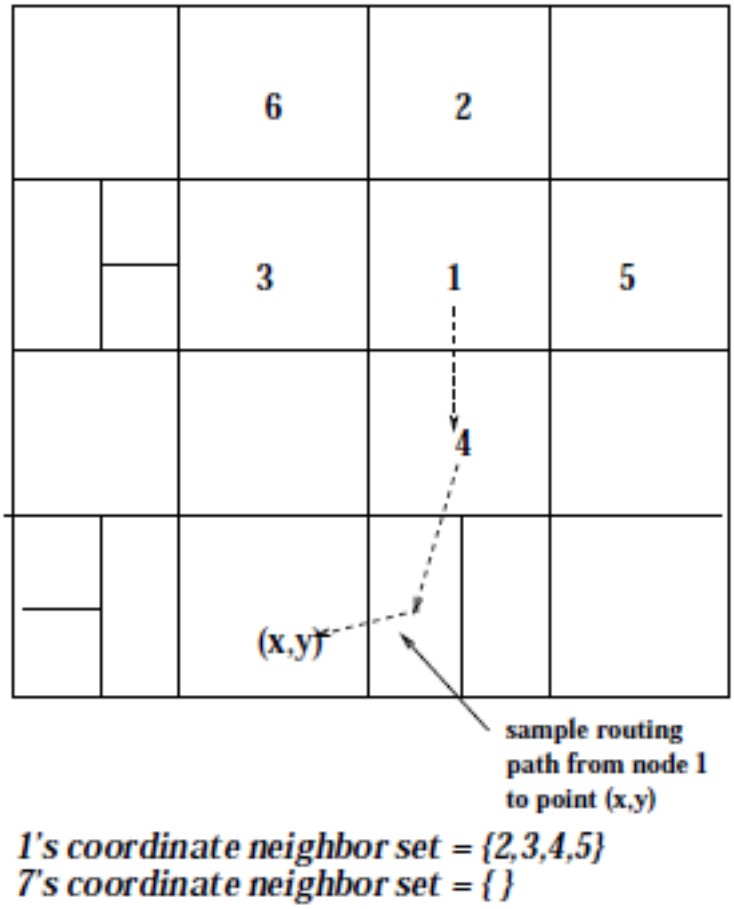


Figure 3.7 : Routage CAN

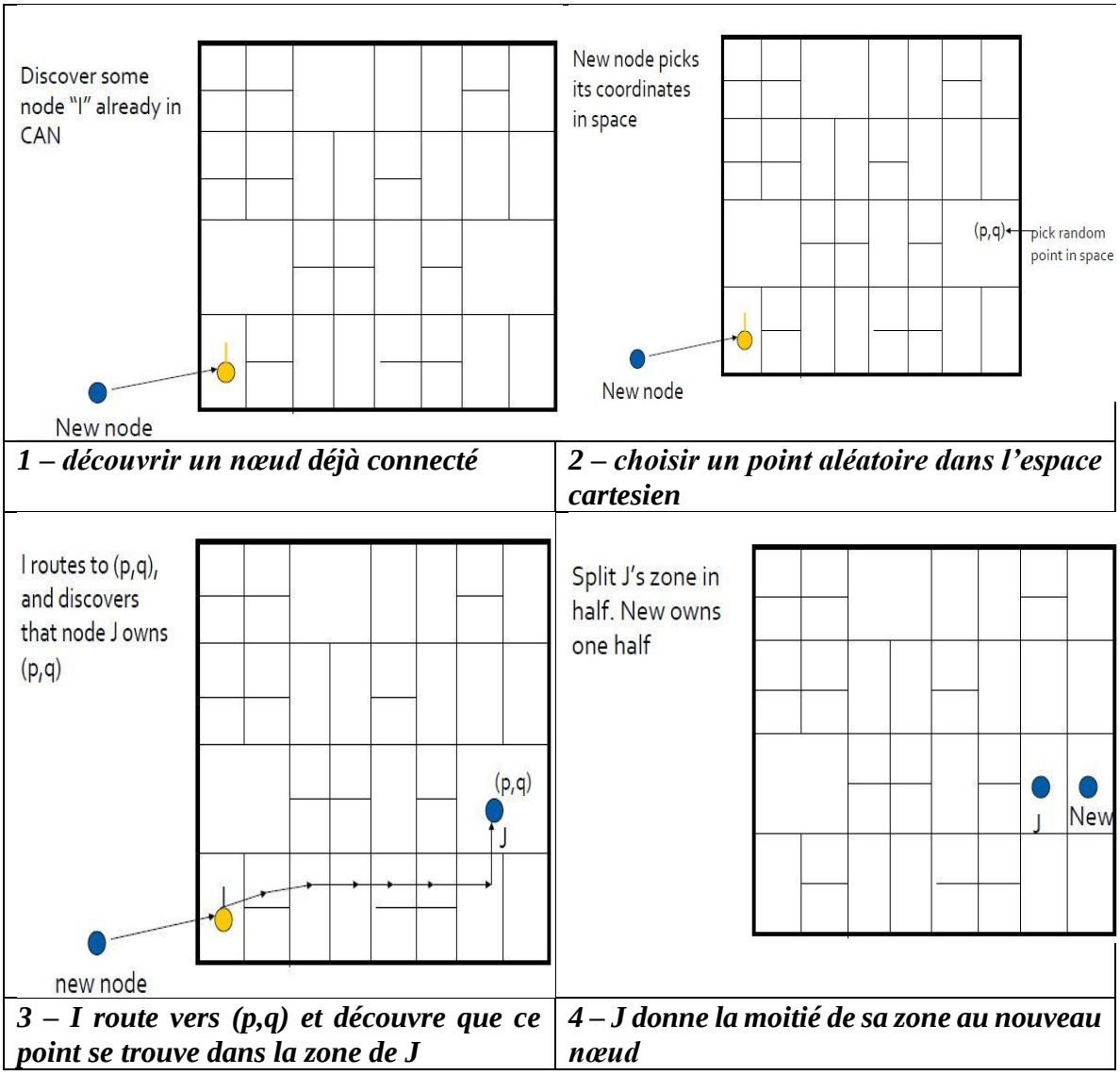
b) Entrée d'un nœud

Pour participer à une CAN, un nouveau nœud doit:

- Trouver un nœud déjà dans le réseau de recouvrement.
- Identifier une zone qui peut être divisée
- Mettre à jour les tables de routage des nœuds voisins de la zone nouvellement divisée.

[16]

Pour trouver un nœud déjà dans le réseau de recouvrement, les nœuds d'amorçage peuvent être utilisés pour fournir le nouveau nœud des adresses IP des nœuds actuellement dans le réseau de recouvrement.



1 – découvrir un nœud déjà connecté

2 – choisir un point aléatoire dans l'espace cartesien

3 – I route vers (p,q) et découvre que ce point se trouve dans la zone de J

4 – J donne la moitié de sa zone au nouveau nœud

Figure 3.8 : comment un nœud se connecte au réseau CAN

Après avoir reçu une adresse IP d'un nœud déjà dans la CAN, le nouveau nœud peut tenter d'identifier une zone pour lui-même. Le nouveau nœud choisit au hasard un point dans l'espace de coordonnées et envoie une requête JOIN, dirigée vers le point aléatoire, à l'une des adresses IP reçues. Les nœuds déjà dans l'itinéraire de réseau de recouvrement acheminent la demande vers le nœud approprié en utilisant leurs tables de routage zone. Une fois que le nœud de gestion de la zone du point de destination reçoit la demande rejoindre, il peut honorer la demande d'inscription en scindant sa zone en deux, en s'attribuant lui-même la première moitié, et en allouant au nouveau nœud la seconde moitié. Si elle ne respecte pas la demande rejoindre, le nouveau nœud continue à sélectionner des points aléatoires dans l'espace de coordonnées et d'envoyer des requêtes JOIN adressées à ces points aléatoires jusqu'à ce qu'il rejoint le réseau avec succès. [16]

Après la scission de la zone et allocation des nouvelles zones créées, les nœuds voisins sont mis à jour avec les coordonnées des deux nouvelles zones et les adresses IP correspondantes. Les tables de routage sont mises à jour et les mises à jour sont propagées à travers le réseau.

c) Départ d'un nœud

Pour gérer le départ d'un nœud, la CAN doit

- identifier le nœud au départ
- fusionner sa zone avec celle d'un nœud voisin
- mettre à jour les tables de routage à travers le réseau. [16]

La détection du départ d'un nœud peut être faite, par exemple, par l'intermédiaire de messages de pulsation qui diffusent régulièrement des informations de table de routage entre voisins. Après une période prédéterminée de silence d'un voisin, ce nœud voisin est déterminé comme ayant échoué et est considéré comme un nœud au départ. Alternativement, un nœud qui aimerait se déconnecter volontairement peut diffuser un tel avis à ses voisins.

Après l'identification d'un nœud au départ, sa zone doit être soit fusionnée soit reprise. Tout d'abord la zone du nœud parti est analysée pour déterminer si elle peut être fusionnée avec celle d'un nœud voisin pour former une zone valide. Par exemple, une zone dans un espace de coordonnées 2d doit être un carré ou un rectangle et ne peut pas être en forme de L. Le test de validation peut parcourir toutes les zones voisines afin de déterminer si une fusion valide peut se produire. Si l'une des fusions potentielles est considérée comme une fusion valide, les zones sont ensuite fusionnées. Si aucune des fusions potentielles n'est jugée valable, alors le nœud voisin avec la zone plus petite prend le contrôle de la zone du nœud au départ. [16] Après une prise de contrôle, le nœud de prise en charge peut périodiquement tenter de fusionner ses zones contrôlées avec des zones voisines respectives.

Si la fusion est réussie, les tables de routage des nœuds de zones voisines sont mises à jour pour refléter la fusion. Le réseau verra cette section du réseau de recouvrement comme une zone unique après une fusion et traitera tout routage avec cet état d'esprit. Pour effectuer une prise de contrôle, le nœud ayant pris le contrôle d'une nouvelle zone met à jour les tables de routage des nœuds voisins, de sorte que toute requête à l'une des deux zones se dirige au nœud responsable. Mais le réseau voit toujours la sous-section du réseau de recouvrement comme deux zones distinctes et traite tout routage avec cet état d'esprit.

5. Pastry

Pastry est un réseau de recouvrement et de routage pour la mise en œuvre d'une table de hachage distribuée (DHT) similaire à Chord. Les paires clé-valeur sont stockées dans un réseau redondant peer-to-peer d'ordinateurs connectés à Internet. Le protocole est initialisé en lui fournissant l'adresse IP d'un pair déjà dans le réseau et à partir de là par l'intermédiaire de la table de routage dynamique qui est construite et réparée. En raison de sa nature décentralisée redondante, il n'y a aucun point de défaillance unique et tout nœud peut quitter le réseau à tout moment sans avertissement et avec peu ou pas de risque de perte de données. Le protocole est également capable d'utiliser une métrique de routage fournie par un programme externe, tel que *ping* ou *traceroute*, afin de déterminer les meilleurs itinéraires à stocker dans sa table de routage.

a) Généralités

Bien que la fonctionnalité de table de hachage distribuée de Pastry est presque identique à d'autres DHT, ce qui le distingue c'est le réseau de routage de recouvrement construit sur le concept de DHT. Cela permet à Pastry de réaliser l'évolutivité et la tolérance aux pannes bien mieux que les autres réseaux, tout en réduisant le coût global de l'acheminement d'un paquet d'un nœud à un autre en évitant la nécessité d'inonder de paquets le réseau. Parce que la métrique de routage est fournie par un programme externe basée sur l'adresse IP du nœud cible, la métrique peut être facilement activée au plus court nombre de sauts, la plus faible latence, la plus haute bande passante, ou même une combinaison générale de métriques.

L'espace des clés de la table de hachage est pris comme étant circulaire, comme l'espace de clés dans le système Chord et les identifiants de nœuds sont des entiers non signés à 128 bits représentant leurs positions dans l'espace de clés circulaire. Les ID de nœud sont choisis aléatoirement et uniformément afin que les pairs qui sont adjacents dans l'ID de nœud puissent être géographiquement éloignés. Le réseau de recouvrement de routage est formé sur le dessus de la table de hachage par chaque pair en découvrant et en échangeant des informations d'état comprenant une liste de nœuds feuille, une liste de voisins, et une table de routage. La liste de nœuds feuille comprend les L/2 pairs les plus proches à l'ID du nœud dans chaque direction autour du cercle.

En plus des nœuds feuille, il y a aussi la liste de voisinage. Cela représente les M pairs les plus proches en termes de la métrique de routage. Bien qu'il ne soit pas utilisé directement dans l'algorithme de routage, la liste de voisinage est utilisée pour maintenir les directeurs de la localité dans la table de routage.

Enfin, il y a la table de routage elle-même. Elle contient une entrée pour chaque bloc d'adresses qui lui est attribué. Pour former les blocs d'adresses, la clé de 128 bits est divisée en chiffres avec chaque chiffre étant de b bits de long, ce qui donne un système de numérotation avec la base$\mathbf{2^b}$. Cela partitionne les adresses en niveaux distincts du point de vue du client, le niveau 0 représentant un préfixe commun à zéro chiffres entre deux adresses, niveau 1 un préfixe commun à un chiffre, et ainsi de suite. La table de routage contient

l'adresse du pair le plus proche connu pour chaque chiffre possible à chaque niveau de l'adresse, sauf pour le chiffre qui appartient au pair lui-même à ce niveau particulier. Cela se traduit par le stockage de $\mathbf{2^b - 1}$ contacts par niveau, avec le nombre de niveaux à l'échelle de $\mathbf{(log_2 N)/b}$. Les valeurs de $\mathbf{b \approx 4}$, $\mathbf{L \approx 2^b}$ et $\mathbf{M \approx 2^b}$ représentent des valeurs opérationnelles sur un réseau typique. [17]

b) Routage

Un paquet peut être routé à une adresse dans l'espace de clés s'il y a un pair ou pas avec cet ID de nœud. Le paquet est routé vers sa place sur l'anneau circulaire et le pair dont l'ID nœud est plus proche de la destination souhaitée recevra le paquet. Chaque fois qu'un poste reçoit un paquet de route ou veut envoyer un paquet, il examine d'abord son ensemble de feuilles et l'envoie directement au bon nœud s'il s'y trouve. Si cela échoue, le prochain pair consulte sa table de routage dans le but de trouver l'adresse d'un nœud qui partage un préfixe plus long avec l'adresse de destination que le pair lui-même. Si l'homologue n'a pas de contacts avec un préfixe plus ou le contact est mort il reprendra un pair de sa liste de contact avec le même préfixe de longueur dont l'ID nœud est numériquement plus proche de la destination et envoie le paquet à ce pair. Comme le nombre de chiffres corrects à l'adresse doit toujours augmenter ou rester le même - et si elle reste la même la distance entre le paquet et sa destination devient plus petit - le protocole de routage converge. [17]

c) Les applications construites sur Pastry

Pastry lui-même spécifie comment les clés sont réparties entre les nœuds et comment le nœud responsable de la tenue d'une clé peut être trouvé. Pastry est utilisé pour implémenter des fonctionnalités comme un système de fichiers distribués, un système d'abonnement et d'édition, ou tout autre système qui peut être réduit à stocker des valeurs et de les récupérer plus tard.

d) PAST

PAST est un système de fichiers distribué construit au-dessus de Pastry. Un fichier est stocké dans le système par le calcul du hachage de son nom. Puis Pastry envoi le contenu du fichier au nœud dans l'espace de clés circulaire plus proche de la valeur de hachage obtenue à partir du nom de fichier. Ce nœud enverra alors des copies du fichier aux k nœuds les plus proches de la clé réelle, dont la plupart sont susceptibles d'être des nœuds feuilles de ce nœud et donc directement accessibles. La demande peut être satisfaite par l'un des k nœuds qui ont des copies des données.

e) SCRIBE

Scribe est un système décentralisé publish / subscribe qui utilise Pastry pour sa gestion d'itinéraire et la recherche sous-jacente d'hôtes. Les utilisateurs créent des sujets auxquels les autres utilisateurs peuvent souscrire. Une fois que le sujet a été créé, le propriétaire du sujet peut publier de nouvelles entrées sous le thème qui sera distribué dans un arbre multicast à

tous les nœuds SCRIBE qui ont souscrit à ce sujet. Le système fonctionne en calculant le hachage du nom de la rubrique concaténé avec le nom de l'utilisateur qui possède le sujet. Cette empreinte est ensuite utilisée comme une clé de Pastry, et l'éditeur, puis achemine les paquets vers le nœud le plus proche de la clé en utilisant le protocole de routage de Pastry pour créer le nœud racine du sujet sur ce nœud. Les gens vont ensuite s'abonner au sujet en calculant la clé du sujet et le nom de l'éditeur, puis en utilisant Pastry pour acheminer un message « vous abonner à ce sujet » vers le nœud racine. Lorsque le nœud racine reçoit le message d'abonnement d'un autre nœud, il ajoute l'ID de nœud à sa liste des enfants et commence à agir comme un transitaire du sujet.

6. Tapestry

a) Introduction

Tapestry est une infrastructure extensible qui fournit la localisation d'objets et le routage décentralisés, en se concentrant sur l'efficacité et la minimisation de la latence des messages. Ceci est réalisé car Tapestry construit localement des tables de routage optimales dès l'initialisation et les maintient dans le but de réduire le tronçon de routage. En outre, Tapesry permet la détermination de la distribution de l'objet en fonction des besoins d'une application donnée. De même Tapestry permet aux applications de mettre en œuvre la multidiffusion dans le réseau de recouvrement.

b) Algorithme **API**

A chaque nœud est attribué un nodeID unique uniformément répartie dans un grand espace d'identifiants. Tapestry utilise l'algorithme SHA-1 pour produire un espace d'identifiant de 160 bits représenté par une clé hexagonale de 40 chiffres. L'Application de points limites spécifiques à GUID est pareillement affectée comme des identificateurs uniques. NodeIDs et les GUID sont à peu près également répartis dans le réseau de recouvrement à chaque nœud stocké, plusieurs identifiants différents sont affectés. A partir de plusieurs expérimentations, il a été remarqué que l'efficacité de Tapestry augmente avec la taille du réseau. Pour différencier entre les applications, un identifiant d'application unique est utilisé. Tapestry utilise le best-effort pour publier les objets et l'itinéraire.

c) PublishObject

Les messages utilisés par ce système sont : UnPublishObject, RouteToObject, RouteToNode (au lieu de la correspondance exacte, on utilise la correspondance la plus proche)

d) Routage

Maille de routage

Chaque identifiant est mappé à un nœud en direct appelé la racine. Si la nodeID d'un nœud est G, alors il est la racine pour d'autres utilisateurs, les nodeIDs de la table de routage et les adresses IP permettent de trouver les nœuds voisins. A chaque hop un message est acheminé

progressivement plus près de G par routage incrémentale de suffixe. Chaque carte de voisinage a des niveaux multiples où chaque niveau contient des liens vers les nœuds correspondant jusqu'à une certaine position de chiffres de l'ID. La -ième entrée primaire dans la j-ème niveau est l'ID et l'emplacement du nœud le plus proche qui commence par le préfixe (N, j-1) + i. Cela signifie que le niveau 1 a des liens vers les nœuds qui n'ont rien en commun, le niveau 2 a le premier chiffre en commun, etc. Pour cette raison, l'acheminement prend environ $\boldsymbol{log_b N}$ sauts dans un réseau de taille N et les ID de base B (hex : B = 16). Si un ID exact ne peut être trouvé, la table de routage volonté route vers le nœud de correspondance le plus proche. Pour la tolérance de panne, les nœuds gardent ces liens secondaires tels que la table de routage a une taille **c * B *** $\boldsymbol{log_b N}$.

e) La publication de l'objet et l'emplacement

Les participants au réseau peuvent publier des objets en acheminant régulièrement « publier un message » vers le nœud racine. Chaque nœud le long du chemin stocke un pointeur cartographiant l'objet. Plusieurs serveurs peuvent publier des pointeurs vers le même objet. Les liens redondants sont priorisés par la latence et / ou de la localité.

Les objets sont situés en acheminant un message vers la racine de l'objet. Chaque nœud le long du chemin vérifie la cartographie et redirige la demande appropriée. L'effet de routage est la convergence des chemins à proximité se dirigeant vers la même destination.

f) Nœuds dynamiques

Insertion d'un nœud

Le nouveau nœud devient la racine pour son nodeID. La racine trouve la longueur du plus long préfixe de l'ID qu'elle partage. [17] Puis il envoie un message de multidiffusion qui atteint tous les nœuds existants qui partagent le même préfixe. Puis ces nœuds ajoutent le nouveau nœud à leurs tables de routage. Le nouveau nœud peut se charger d'être la racine pour certains des objets de la racine. Les nœuds contacteront le nouveau nœud pour fournir une liste de voisins temporaire. Le nouveau nœud effectue ensuite une recherche itérative du plus proche voisin pour remplir tous les niveaux dans sa table de routage.

Départ d'un nœud

Pour quitter le réseau, un nœud diffuse son intention de quitter et transmet le nœud de remplacement pour chaque niveau dans les tables de routage des autres nœuds. Les objets stockés par ce nœud sont redistribués ou reconstitués à partir des copies redondantes.

L'échec de nœud

La défaillance inattendue d'un nœud est gérée grâce à la redondance du réseau et des pointeurs de sauvegarde pour rétablir des liens endommagés.

Les Applications

Tapestry fournit un réseau de routage de recouvrement qui est stable dans une grande variété de conditions du réseau. Cela offre une infrastructure idéale pour les applications et services distribués. Les applications basées sur le protocole Tapestry sont:

- OceanStore - utilitaire de stockage distribué sur PlanetLab
- Mnémosyne - système de fichiers stéganographique
- Bayeux - Auto-organisation demande la multidiffusion
- SpamWatch - filtre anti-spam décentralisée

Pour finir, nous avons mappés toutes ces techniques dans un tableau de comparaison pour avoir une idée globale sur les avantages et inconvénients de chacune des techniques citées.

Tableau 2.1 : Comparaison entre quelques systèmes utilisant une DHT

DHT	Diamètre	Taille de la table de routage	Flexibilité Voisinage	Flexibilité Routage	Robustesse	Type de réseau
Tapestry	O(log_{2^b} N)	O((2^{b-1})*log_{2^b} N)	$N^{\frac{\log N}{2}}$	1	++	maille
Pastry	O(log_{2^b} N)	O((2^{b-1})*log_{2^b} N)	$N^{\frac{\log N}{2}}$	2^b	++	anneau
Chord	O (log N)	O(log N)	$N^{\frac{\log N}{2}}$	C log N	+/-	Anneau
Kademlia	O(log_{2^b} N)	O(k*log_{2^b}N)	$N^{\frac{\log N}{2}}$	2^b	+/-	Arbre
CAN		O(d)	d	c log N	+++	d-dimensionnel

Avec :

- b : nombre de chiffres dans l'alphabet utilisé
- N : nombre de nœuds du réseau
- k : (pour Kademlia) le nombre de voisins dans chaque sous arbre d'un nœud
- d : (pour CAN) le nombre de dimensions (d = 2^b)
- diamètre : la plus grande distance, en nombre de sauts, entre deux pairs
- une constante du système, choisie par les concepteurs

D'après le tableau ci-dessus, on voit les avantages et inconvénients de chacun des systèmes utilisant une DHT, et on peut en déduire que aucun n'est parfait pour toutes les applications qu'on peut en faire, mais chacun est mieux adapté à certaines applications plutôt que d'autres. Dans notre cas, nous voulons mettre au point un système permettant le téléchargement de fichiers en P2P, et donc il faudrait utilisé un système permettant de trouver l'endroit où est

stocké le fichier recherché de manière rapide, efficace et sans submerger le réseau de demandes inutiles, donc pour nous le système idéal serait CAN, chose malheureusement qu'on n'aura pas le temps de développer du moins dans ce PFE, suite à la complexité de la tâche, peut-être plus tard pour une thèse de doctorat.

7. Conclusion

Dans ce chapitre, nous avons passé en revue avec des détails, les systèmes deuxième génération après Napster et Gnutella, qui sont fragiles face à l'utilisation d'un serveur de localisation des ressources (qui peut tomber en panne mettant hors service tout le réseau) et le passage à l'échelle qui est complexe à mettre en place. Donc Torrent, Chord, Pastry et Tapestry sont venus pallier aux inconvénients de leurs prédécesseurs, en utilisant comme annuaire décentralisé et distribué une table de hachage (DHT). Alors bien évidemment chacun a un fonctionnement différent de l'autre, étant plus adapté à telle application par rapport à une autre. Une comparaison est faite à la fin du chapitre pour en déterminer les avantages et inconvénients.

Dans le chapitre suivant, nous développons notre application avec son fonctionnement et son architecture, basée sur une hybridation entre les architectures première et deuxième génération, car la complexité des systèmes deuxième génération est telle qu'il faudrait une année entière pour arriver à bout. Donc notre architecture est un LAN, utilisant une table de hachage pour localiser les ressources recherchées.

Chapitre 4 : Application

1. Introduction

L'application développée offre à l'utilisateur la fonctionnalité de téléchargement des fichiers à partir d'autres utilisateurs ayant aussi lancé une instance. J'ai pris comme modèle les réseaux non-structurés, et donc l'application n'impose aucune structure sur le réseau actuel. Elle ne marche que sur des LAN (réseaux locaux). Grace à ce dernier, les pairs participants voient la topologie entière du réseau. Le choix de cette architecture pour notre application se justifie par le fait que la solution basée sur les réseaux structurés est trop compliquée à faire, comme nous avons pu le voir dans le chapitre trois. Vu le temps imparti pour mener à bien notre PFE, c'était un choix obligé.

2. Caractéristiques / propriétés

Notre architecture P2P choisie a les caractéristiques suivantes :

1. tout pair voit tout autre pair connecté au réseau
2. pour qu'un fichier soit retourné dans le résultat d'une recherche, toute la chaine cherchée doit être contenue dans son nom
3. les recherches sont exhaustives – tout le monde est contacté lors d'une recherche donc si le fichier existe il sera trouvé
4. un téléchargement donné implique exactement deux pairs – celui qui donne (le serveur dans ce cas) et celui qui reçoit (le client dans ce cas)
5. des téléchargements simultanés sont supportés
6. le protocole UDP est utilisé pour la découverte et TCP pour tout le reste

Le tableau ci-dessous résume l'environnement dans lequel l'application a été développée :

Paramètre	Valeur
Système d'Exploitation	Windows 7
Environnement de Développent Intégré	NetBeans 8.1
Langage	Java
Version JDK	1.6
Mémoire Vive	4 Go

Tableau 3.1

3. L'interface Utilisateur

La fenêtre principale comprend 5 onglets. L'onglet 'Options' est utilisé comme panneau de configuration pour l'application.

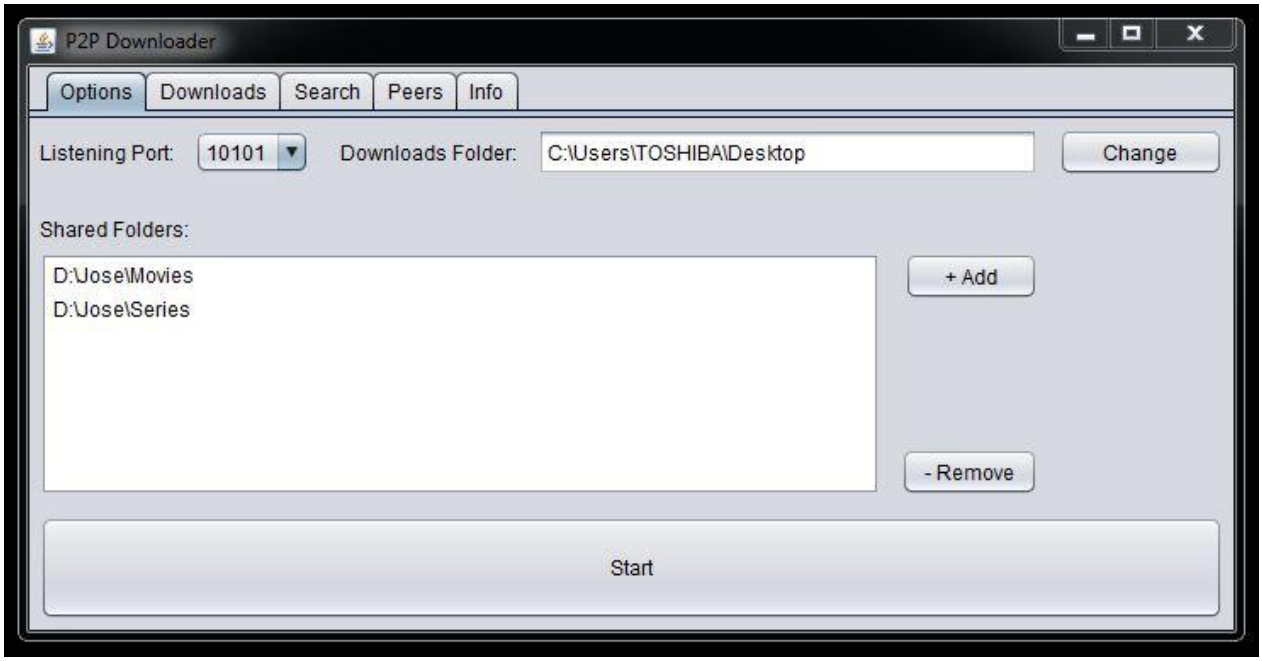


Figure 4.1 : L'onglet de configuration. Ici on peut choisir le port pour l'écoute, le dossier destinataire des téléchargements, et les dossiers à partager sur le réseau

L'application offre à l'utilisateur la possibilité de choisir :

- Le port sur lequel faire l'écoute
- Le répertoire pour enregistrer les fichiers téléchargés
- L'ensemble de répertoires qui seront consultés lors d'une recherche. Tout utilisateur de l'application sur le réseau sera capable de télécharger le contenu de ces dossiers.

Lorsque le bouton ‘Start’ est cliqué, l’application lance un Socket sur le port sélectionné pour écouter le réseau. Au même temps elle balaie le réseau pour trouver les autres pairs.

L’onglet « Info » contient un tableau qui sert à afficher des informations utiles et pour la plupart les techniques de fonctionnement du programme. Cela permet par exemple de savoir la cause d’une erreur quant il s’en produit.

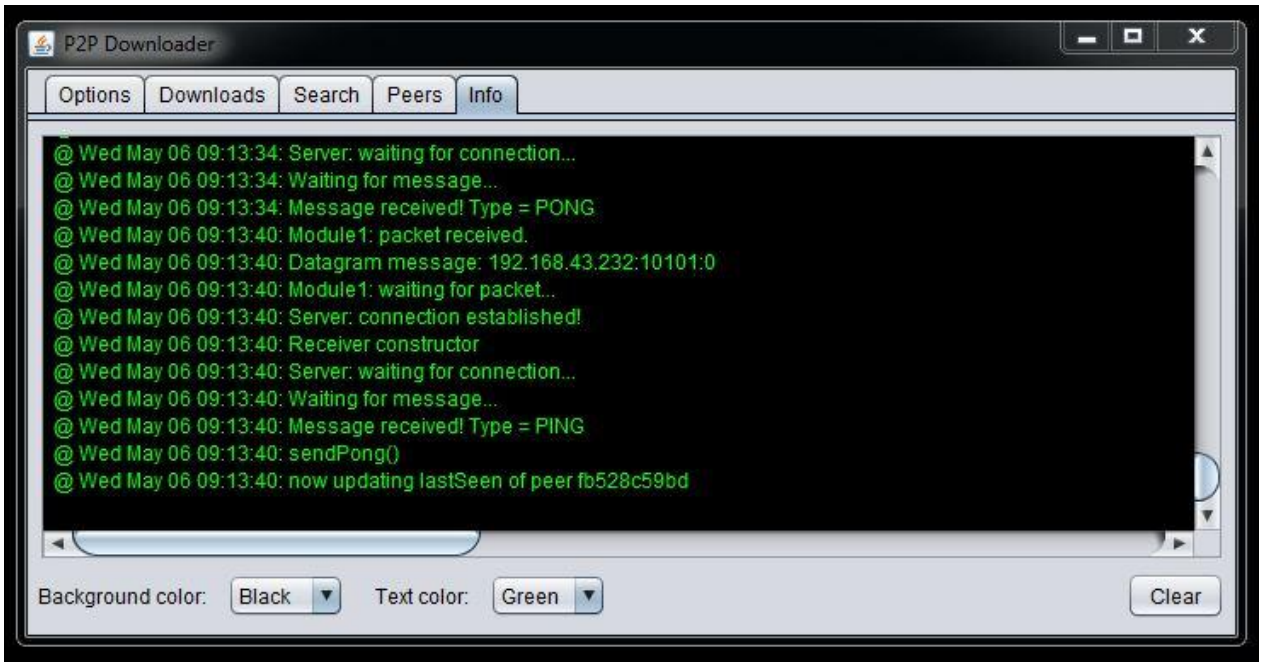


Figure 4.2 : Le panneau d’informations

L’onglet « Peers » contient une table dans laquelle les pairs connectés sont affichés. On peut donc savoir l’état du réseau dans lequel on est. Si cette table est vide, ça veut dire qu’il n’y a pas d’autres utilisateurs de l’application sur notre réseau, ou que le bouton « Start » n’a pas été cliqué pour lancer l’application localement.

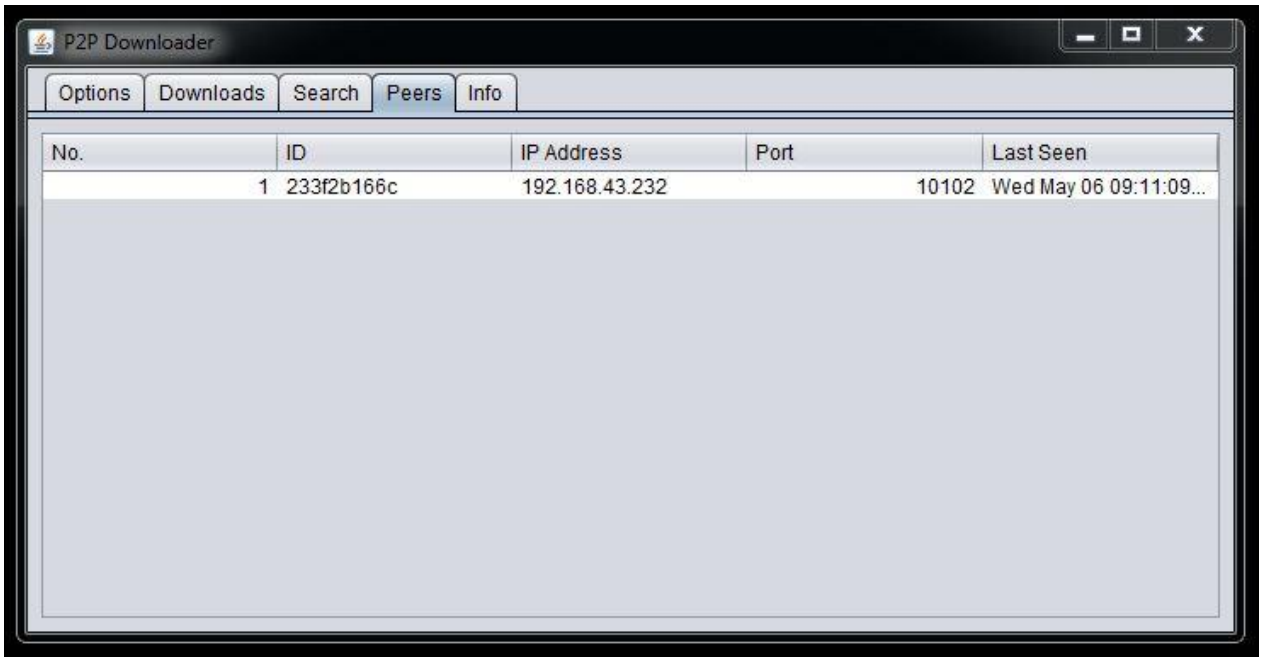


Figure 4.3 : La liste des pairs connectés

L'onglet « Search » permet à l'utilisateur de lancer des recherches et de visualiser éventuellement les résultats. En cliquant sur un résultat et puis sur le bouton « Download » on peut donc lancer un téléchargement du fichier sélectionné.

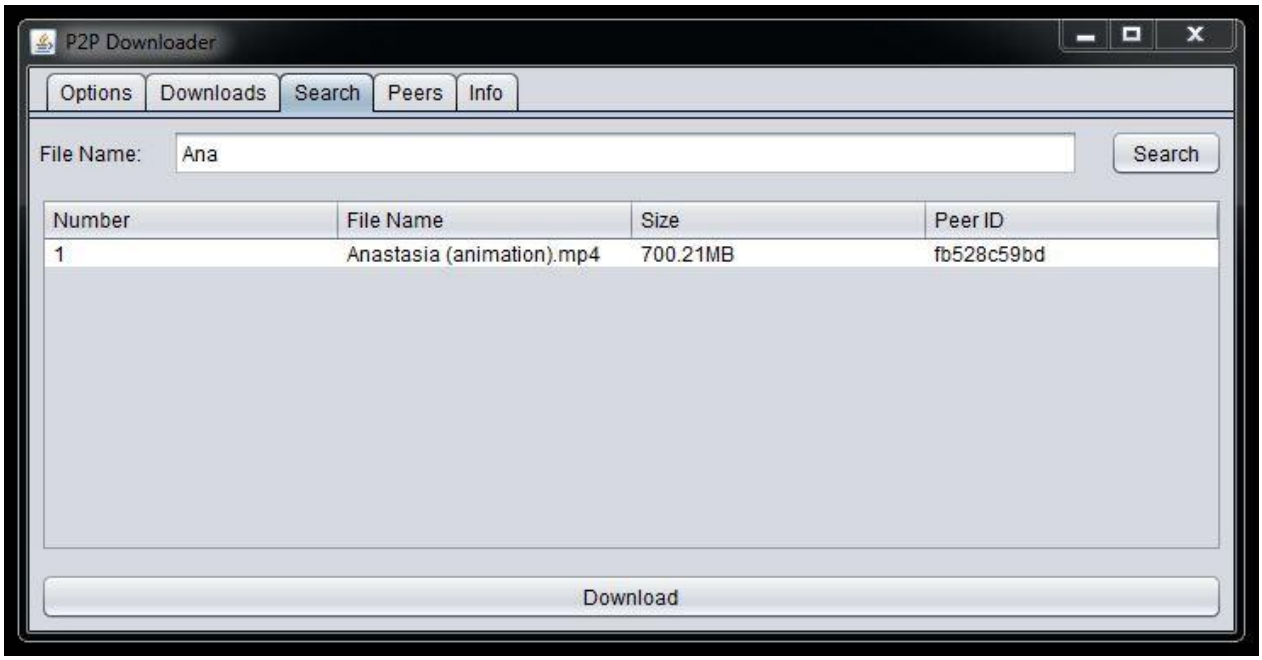


Figure 4.4 : Les résultats d'une recherche du terme « Ana »

L'onglet « Downloads » contient une table dans laquelle sont affichés tous les téléchargements (en cours et terminés). Ça nous permet donc de suivre la progression des téléchargements et de savoir lorsqu'ils se terminent.

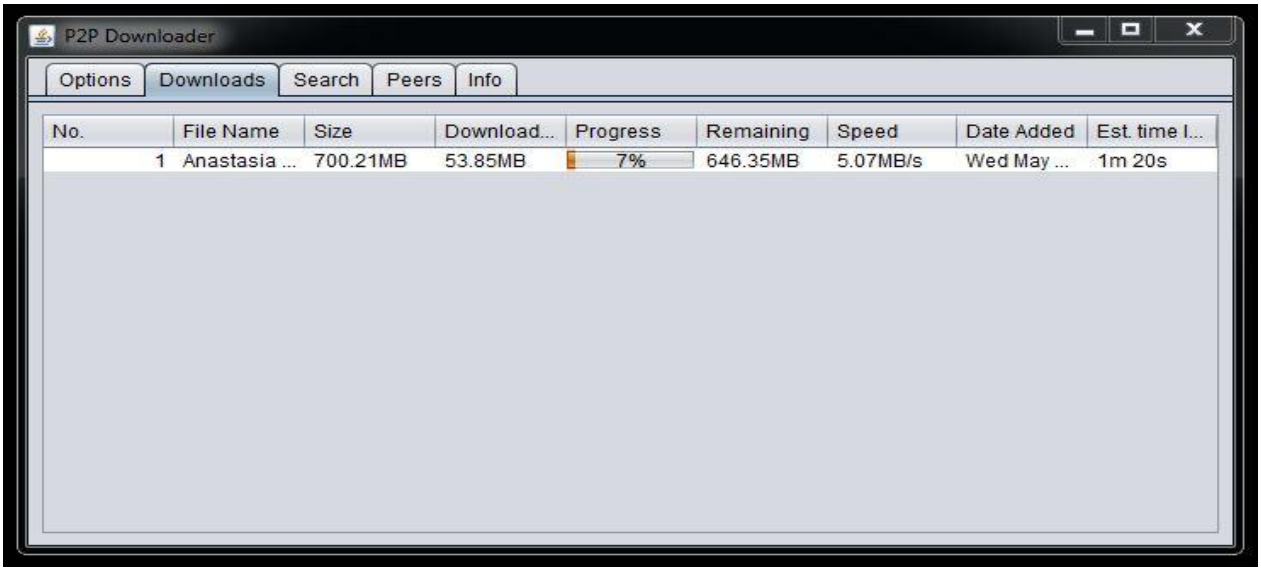


Figure 4.5 : Onglet des téléchargements ; Ici un téléchargement à 7%

Cette application est fondamentalement modulaire, un fait qui a été facilement réalisé étant donné le langage employé. Les modules sont les suivants :

- Un serveur
- Un éclaireur
- Un envoyeur
- Un receveur
- Un coordinateur

Lorsque l'application est lancée, le serveur se met à l'écoute sur le port donné en attendant des connexions éventuelles.

L'éclaireur quant à lui commence à envoyer des paquets UDP vers une adresse multicast pour découvrir les autres machines sur lesquelles s'exécute l'application. Chaque instance de l'application fait l'écoute sur l'adresse multicast 224.10.113.10.

Un envoyeur est créé à chaque fois que l'application souhaite envoyer un message à travers le réseau. Chaque instance n'envoi qu'un seul message durant son cycle de vie.

Un receveur est créé à chaque fois qu'une nouvelle connexion est acceptée par le serveur car le protocole oblige que toute connexion doit être suivi de l'envoi d'un message.

Le coordinateur supervise le fonctionnement de tous les autres modules du programme. Il est créé au lancement de l'application et sa durée de vie est égale à celle du programme.

Lorsque l'utilisateur lance une recherche, l'application contacte tous les pairs connus. Chaque pair cherche dans ses dossiers partagés pour trouver des fichiers ayant des noms qui satisfont la chaine recherchée. Les résultats sont renvoyés sous forme de tableaux (un pair renvoie un tableau vide s'il ne trouve aucun fichier satisfaisant la demande). L'application ensuite présente tous les résultats de tous les pairs à l'utilisateur dans un seul tableau. A ce point il suffit d'en sélectionner un et lancer un téléchargement.

4. Réalisation de l'application

Les packages principales utilisés :

1. java.io
2. java.net

- Et les classes du package java.io utilisées sont:

1. File
2. FileInputStream
3. FileOutputStream
4. FileNotFoundException
5. IOException
6. ObjectInputStream
7. ObjectOutputStream
8. InputStream
9. OutputStream
10. BufferedInputStream
11. BufferedOutputStream

- Les classes du package java.net utilisées sont:

1. DatagramPacket
2. DatagramSocket
3. MulticastSocket
4. ServerSocket
5. Socket
6. SocketException
7. InetAddress
8. UnknownHostException

- Le lancement de l'application crée 3 modules :

1. Serveur de type ServerSocket pour accepter des connexions
2. Objet de type Module1 pour recevoir des datagrammes multicast
3. Objet de type Module2 pour envoyer des datagrammes multicast

Suite au lancement de l'application, celle-ci n'a aucune connaissance des autres applications (pairs) sur le réseau. Les modules les plus importants à ce point sont Module1 et Module2 (j'ai choisi les noms arbitrairement) car ils s'occupent de la découverte des autres pairs. Le Module1, à l'aide d'un Socket de type MulticastSocket, s'inscrit au groupe multicast 224.10.113.10 et il attend de recevoir des paquets multicast envoyés à ce groupe. Le Module2 envoi des datagrammes au même groupe en espérant qu'une (ou plusieurs) instance(s) Module1 d'une autre application(s) les recevra (recevront). C'est de cette façon que la découverte des pairs est accomplie.

Lorsque le Module1 reçoit un datagramme, il extrait le message qu'il porte. Ce message contient l'adresse IP et le numéro de port de la source du datagramme. (Ces informations y sont mises par le Module2 de cette machine source.) Ensuite il appelle la méthode reportAddress(String adresse, int port) de la Classe principale P2P Downloader dans le but de l'informer qu'un pair a été découvert à cet adresse.

Le module P2P Downloader compare l'adresse et le numéro de port avec les valeurs correspondantes de la machine locale. Si elles sont identiques il ne fait rien. Sinon il s'agit d'un autre pair alors il effectue le test suivant : est-ce que ce pair est déjà dans la table de hachage (ConcurrentHashMap) locale ? Si oui, il met à jour son champ lastSeen ; sinon il l'insère dans la table en initialisant le champ lastSeen avec le temps actuel. A ce point la liste des pairs dans l'onglet Peers est actualisée pour montrer les derniers changements.

Mise à part la découverte (et le téléchargement, comme on verra), toute communication entre des pairs utilise des messages du type Message (la classe Message). Un message peut être de l'un de ces types :

1. **PING** – Ce message est envoyé périodiquement à tout pair présent dans la table de hachage locale peerDescriptors. Un pair qui reçoit ce message doit répondre avec un message PONG
2. **PONG** – Ce message est envoyé en tant que réponse à un PING. Un pair qui reçoit ce message met à jour le champ lastSeen (vu dernièrement) du pair qui l'a envoyé
3. **SEARCH** – Ce message est envoyé à tous les pairs dans la table de hachage locale pour effectuer une recherche. Il contient une chaine de caractères indiquant une partie (ou l'entièreté) du nom du fichier recherché. Un pair qui reçoit ce message doit comparer la chaine avec les noms des fichiers dans ses dossiers partagés et construire un tableau contenant ceux qui contient la chaine. Puis il envoie un message RESULT contenant ce tableau à l'expéditeur du message SEARCH.
4. **RESULT** – Ce message est envoyé comme réponse à un message SEARCH. Il contient un tableau de détails des fichiers qui satisfont la recherche effectuée. Ce tableau peut être vide (de longueur 0), et c'est le cas si un pair n'a aucun fichier dont le nom contient la chaine recherchée. Un pair qui reçoit ce message extrait son contenu et le présente à l'utilisateur dans l'Onglet Search.
5. **REQDATA** – Ce message est envoyé suite à la sélection d'une ligne dans la liste des résultats et l'appui sur le bouton Download. Il contient le nom du fichier choisi et il avertit le propriétaire du fichier que ce pair aimerait le télécharger. Un pair qui reçoit ce message crée un FileUploader qui envoie un message DATA en réponse.

6. **DATA** – Ce message est envoyé au pair désirant télécharger un fichier à son PC par le pair qui a le fichier. Il contient des informations supplémentaires comme la longueur du fichier et le numéro de port qui sera utilisé pour le transfert. Un pair qui reçoit ce message crée une instance de FileDownloader pour effectuer son téléchargement.
7. **BYE** – Ce message est envoyé par un pair désirant quitter le réseau à tous les pairs dont il a les adresses dans sa table de hachage. Un pair qui reçoit ce message supprime le pair correspondant de sa table de hachage.

Tout envoi de message se fait à l'aide du module Sender. Chaque instance de cette classe n'envoie qu'un seul message. La première chose faite par un Sender c'est se connecter au Server du pair destinataire. Le module Server est une classe dont une instance fait l'écoute tant que l'application tourne.

Parfois un pair peut quitter de façon imprévu et donc ne pas envoyer le message BYE. Pour assurer qu'un tel pair ne restera pas dans la table de hachage, chaque pair parcourt sa table périodiquement et supprime les pairs dont le champ lastSeen a une valeur plus ancienne que 60 secondes. Un tel pair a dû ne pas répondu à au moins 3 messages PING, d'où la conclusion qu'il n'est plus connecté.

Le module FileUploader lit et envoi 4 kilo octets à la fois.Il fait la lecture (du fichier) à travers la méthode *read()* de la classe BufferedInputStream. Il fait l'envoi (des octets lus) à travers la méthode *write()* de la classe OutputStream. Le module FileDownloader lit et écrit 4 kilo octets à la fois. Il fait la lecture à travers la méthode *read()* de la classe InputStream. Il fait l'écriture à travers la méthode *write()* de la classe BufferedOutputStream. A la fin du transfert, les deux modules (FileUploader et FileDownloader) ferment les flux créés respectivement mais dans l'ordre inverse de leur création.

Toutes ces étapes nous ont permis d'arriver à une application fonctionnelle et efficace du point de vue rapidité et le nombre de recherches abouties. Le tableau ci-dessous récapitule la quantité d'infos échangées entre les pairs d'un réseau P2P, en augmentant graduellement la taille du réseau.

	2 PCs		3 PCs		4 PCs	
Temps	envoyé	reçu	envoyé	reçu	envoyé	reçu
20 s	1998	1903	3894	4001	5607	5411
40 s	3796	3614	7402	7608	10648	10277
60 s	5901	5697	11819	11283	16918	16321
80 s	7846	7469	15461	14763	22121	21339
100 s	9807	9340	1929	18432	27641	26659
120 s	11767	11195	23175	22125	33160	31981

Tableau 3.2 : Analyse de la quantité de trafic (en octets) acheminé par une machine

5. Conclusion

Dans ce chapitre, nous avons présenté en détails les étapes que nous avons suivies dans notre implémentation d'une application de téléchargement en P2P, en utilisant le langage JAVA, qui est un langage multiplateforme, donc portable et contenant tous les outils dont nous avions besoin pour développer une telle application, comme les classes « socket » et « message », entre autre pour faire l'écoute sur un port et le téléchargement et toutes les opérations que nous avons décrites précédemment. Suite à cela, les futurs étudiants intéressés par le développement de telles applications trouveront déjà une base consistante, sur laquelle évoluer. Les différents tests que nous avons effectués se sont avérés positifs et donc notre application remplit parfaitement les objectifs que nous nous étions posés au début. Cependant on peut améliorer un logiciel quel que soit son degré de perfection.

Conclusion Générale

Ce mémoire essaie d'explorer les diverses technologies pair à pair. Il est axé sur le fonctionnement détaillé de ces technologies. Comme le sujet est tellement vaste, cette exploitation n'est pas du tout complète, mais notre espoir est qu'elle puisse donner aux lecteurs une base fondamentale sur la problématique des systèmes P2P.

Le partage de fichiers n'est pas illégal et les réseaux pair-à-pair sont utilisés à des fins légitimes. Les questions juridiques dans le partage de fichiers impliquent la violation des lois du matériel protégé. La plupart des discussions sur la légalité du partage de fichiers impliquent le droit d'auteur sur le matériel uniquement. Beaucoup de pays ont des exceptions d'utilisation équitable qui permettent une utilisation limitée du matériel protégé sans être obligés d'avoir la permission des détenteurs de droits. Ces documents comprennent des commentaires, rapports de nouvelles, de la recherche et de l'érudition. Les lois de copyright sont territoriaux- ils ne dépassent pas le territoire d'un état spécifique que si cet État fait partie d'un accord international. La plupart des pays font aujourd'hui parties d'au moins un tel accord.

Le P2P est en plein essor et la recherche dans ce domaine est exponentielle, mais reste un domaine complexe et pas entièrement abouti. Notre contribution modeste dans ce domaine consiste en une application qu'il faut installer dans tous les pairs, écrite en JAVA. Les pairs doivent appartenir à un réseau local, donc le nombre des pairs ne doit pas dépasser 254 membres, et tous se connaissent grâce à une découverte du réseau et peuvent ensuite rechercher n'importe quelle ressource et la télécharger ensuite de manière rapide et efficace. Les numéros de port sont dynamiques et choisis par l'utilisateur. Nous espérons avoir donnée les bases de cette thématiques pour les futurs étudiants en quête de thème de projet de fin d'études, surtout un état de l'art global et les fonctionnements les plus répandus et les plus utilisés ou du moins les plus aboutis. Cependant des améliorations peuvent toujours être apportées, surtout le passage à l'échelle, que nous nous sommes posés comme perspectives.

Bibliographie

1	Petar Maymounkov et David Mazières. New York University	"Kademlia: A Peer-to-peer information system based on the XOR Metric"
2	Rasti, Stutzbach, Rejaie, 2006	On the Long-term Evolution of the Two-Tier Gnutella Overlay.
3	Tyson, Jeff.	How the Old Napster Worked
4	Menn, Joseph (2003). Crown Business. ISBN 0-609-61093-7.	All the Rave: The Rise and Fall of Shawn Fanning's Napster.
5	Copyright and Peer-To-Peer Music File Sharing:	The Napster Case and the Argument Against Legislative Reform", March 2004.
6	gnutella.sourceforge.net	http://rfc-gnutella.sourceforge.net/src/rfc-0_6-draft.html
7	Watson, Stephanie.	"How Kazaa Works".
8	Babaoglu, Ozalp (2012). Complex Systems. Universitá di Bologna. Retrieved 6 February 2013.	"Introduction to Peer-to-Peer Systems" (PDF).
9	Cohen, Bram (2 July 2001). Yahoo eGroups. Archived from the original on 2 July 2011. Retrieved 15 April 2007.	"BitTorrent — a new P2P app".
10	Jordan Ritter <jpr5@darkridge.com>	Why Gnutella Can't Scale. No, Really.
12	Alejandro Zentner, The B.E. Journal of Economic Analysis & Policy, Vol. 5, Issue 1 (2005)	"File Sharing and International Sales of Copyrighted Music: An Empirical Analysis with a Panel of Countries"
13	Dye, Mark. McDonald, Rick. Rufi, Antoon W.	'Network Fundamentals', Cisco Networking Academy, Cisco Press, Ch 3. p91
14	TorrentFreak. 24 January 2007. Archived from the original on 26 March 2014. Retrieved 7 April 2013.	"BitTorrent: The "one third of all Internet traffic" Myth".
15	Thomas Mennecke. 2005.	How Overpeer was able to corrupt data on the FastTrack network.
16	Ratnasamy et al. (2001)	A Scalable Content-Addressable network

17	Michael Welzl; Institute of Computer Science. University of Innsbruck, Austria	"Peer-to-Peer Systems DHT examples part 2 (Pastry, Tapestry and Kademlia)"
18	Costa, Fernando; Silva, Luis; Fedak, Gilles; Kelley, Ian (2008). IEEE International Symposium on Parallel and Distributed Processing, 2008. IPDPS 2008. IEEE. doi:10.1109/IPDPS.2008.4536446. Retrieved 7 April 2013	"Optimizing the Data Distribution Layer of BOINC with BitTorrent" (PDF).
19	Stefan Saroiu, P. Krishna Gummadi and Steven D. Gribble. Technical Report UW-CSE-01-06-02, University of Washington, Department of Computer Science and Engineering, July 2001.	"A Measurement Study of Peer-to-Peer File Sharing Systems."
20	Cohen, Bram (October 2002). BitTorrent.org. Archived from the original on 8 February 2014. Retrieved 19 April 2013.	"BitTorrent Protocol 1.0".
21	Stoica, I.; Morris, R.; Karger, D.; Kaashoek, M. F.; Balakrishnan, H. (2001). (PDF). ACM SIGCOMM Computer Communication Review 31	"Chord: A scalable peer-to-peer lookup service for internet applications"

Liste des Figures

Liste des Tableaux