Interconnexion et routage efficaces pour des procédures de recherche décentralisées dans les systèmes pair-à-pair

Philippe Gauron

N° d'ordre: 8404

THÈSE

présentée

devant l'Université Paris-Sud 11

pour obtenir

le grade de : Docteur en Sciences de l'Université Paris-Sud
Mention Informatique

par

Philippe Gauron

Équipe d'accueil : GraFComm - LRI
École Doctorale : Informatique
Composante universitaire : Faculté des Sciences d'Orsay

Titre de la thèse :

Interconnexion et routage efficaces pour des procédures de recherche décentralisées dans les systèmes pair-à-pair

Soutenue le 28 septembre 2006 devant la commission d'examen

M. :	Joffroy	BEAUQUIER	Président
MM. :	Pascal	FELBER	Rapporteurs
	Laurent	VIENNOT	
MM. :	Marie-Christine	ROUSSET	Examinateur
	Pierre	FRAIGNIAUD	Directeur de thèse

« Ils ne savaient que c'était impossible, alors ils l'ont fait. »
par Mark Twain

« Je m'efforce de tout comprendre et de ne rien condamner. »
Le Baron de Charlus, dans « A l'ombre des jeunes filles en fleur », par Marcel Proust.

Remerciements

Ces pages de remerciements sembleront peut-être longues. Non que je ne veuille oublier personne –car je ne pourrais y couper– mais je suis conscient que de petites causes ont de grands effets. Je suis donc heureux de pouvoir rendre hommage par la plume en attendant la voix vive, à ces papillons qui par leurs battements d'ailes ont du déchaîner des tempêtes...

Tout d'abord, je souhaite remercier Pierre Fraigniaud pour ces années de thèse. Elles furent enrichissantes grâce à lui à bien des niveaux. D'abord pour m'avoir fait découvrir plus avant le monde de la recherche au cours d'un stage de D.E.A. passionnant sur un sujet d'actualité ; mais aussi durant mon doctorat sur un sujet tout aussi passionnant et plus large, grâce à nos discussions et aux différentes réunions de recherche auxquelles j'ai pu participer, motivantes et sources d'un foisonnement d'idées. Il m'a aussi intégré à la communauté de recherche et m'a montré comment renforcer ma rigueur. Pour tout cela, mais aussi pour la patience et la disponibilité dont il a fait preuve au cours de ces années, et la confiance qu'il m'a témoigné pour ce chemin parcouru ensemble, je le remercie du fond du coeur.

Je tiens à remercier Pascal Felber et Laurent Viennot, pour avoir accepté de relire mon rapport de thèse avec attention et fait part de leurs remarques le concernant, ainsi que pour leurs contributions qui m'ont fait découvrir des problématiques proches de mon sujet et aux solutions élégantes qu'ils ont proposé. Merci aussi à Joffroy Beauquier et Marie-Christine Rousset qui ont bien voulu faire partie de mon jury.

Matthieu Latapy m'a fait confiance dès le début de mon D.E.A. au sein du groupe G.R.M., qui a su motiver comme moi bien des doctorants et jeunes docteurs. Il m'a aussi donné une ouverture sur les disciplines connexes à mon domaine de recherche, et plus largement sur les grands réseaux d'interactions. Enfin, je le remercie pour nos différentes réunions de travail à la fin de mon doctorat. Ce furent des échanges riches et agréables qui furent vinrent compléter à-propos mon travail avec Pierre. Au cours

de G.R.M. ou d'autres groupes de discussion, j'ai pu participer à des échanges riches et formateurs, je tiens à citer Simon Bliudze, Christian Destrée, Anh-Tuan Gai, Jean-Loup Guillaume, Fabien Mathieu, Nathalie Mitton, Étienne Rivière, avec qui j'ai pu travailler plus longuement, et Gwendal Simon.

Parmi les différentes personnes que j'ai côtoyé dans mon bureau, je remercie David Ilcinkas et Nicolas Nisse, qui se sont fait tour à tour complices de nos records au C.E.S.F.O., relecteurs de mes articles, auditeurs de mes présentations, testeurs de BN aux couleurs bien peu rassurantes, et victimes souriantes des coups de téléphone reçus ou de ma vie associative. J'ai aussi eu le plaisir de voir régulièrement au début de ma thèse Grzegorz Gancarzewicz et Rafael Castro de Andrade et de manière plus générale les doctorants de l'équipe Moaiz Ben Dhaou, Wadie Benajam, Taoufik Faik, Lynda Gastal, Mathieu Le-Coz, Rafaël Lopez, Lehuy Nguyen, David Soguet. Ils ont contribué à des années de thèse conviviales et parfois bruyantes. Merci en particulier pour les relectures de mes chapitres de manuscrit à Camille, David I., Morgoth, Nicolas N. et Puck, qui m'ont permis d'améliorer sa lisibilité et sa clarté.

De manière plus générale, j'ai pu apprécier la vie de l'équipe GraFComm avec son coin café, ses séminaires réguliers et toujours différents qui m'ont fait découvrir des problèmes de recherche différents et des réponses originales et élégantes.

Le L.R.I. qui m'a accueilli de mon D.E.A. à la fin de mon doctorat m'a fait profiter d'un environnement de recherche remarquable, de par son cadre agréable à vivre. L'équipe administrative remarquablement efficace qui a pu répondre parfois à des demandes dans des délais très courts, en particulier Martine Lelièvre, une équipe système toujours attentive et présente en plus d'être compétente et sympathique. Bien sûr, la présence d'enseignants-chercheurs et de chercheurs dont la culture et l'ouverture d'esprit ont répondu à bien des questions que je me posais et/ou m'ont soutenu, je pense entre autres à Jean-Paul Allouche, Joffroy Beauquier, Stéphane Boucheron, Alain Denise, Sandrine-Dominique Gouraud, Sébastien Tixeuil. Enfin, je tiens à remercier les enseignants-chercheurs du département informatique qui m'ont accompagné dans mes enseignements, contribuant à les rendre agréables, entre autres Nicolas Baskiotis, Pascal Berthomé, Cécile Balkanski, Sylvain Conchon, ma tutrice Sylvie Delaët, Laurence Devillers, Christine Kéribin, Hélène Maynard, Lydia Nicaud, Julien Signoles, Romain Texier, Frédéric Voisin, Fatiha Zaidi. Les étudiants qui m'ont fait face sont aussi du lot des remerciés.

Il me faut aussi remercier l'équipe du C.I.E.S. qui a donné énormément et souvent plus que ce qui leur était demandé, tout en gérant cette responsabilité de manière intelligente et constructive pour nous aider dans notre rôle d'enseignant, en particulier Alain Sarfati et Nouari Kébaïli, mais aussi tous les doctorants qui ont contribué à rendre ces stages enrichissants tant aux plans professionnel que personnel.

Si je veux remercier mes parents qui m'ont donné la chance de faire des études longues,

pour leur soutien, avec le coût que cela impliquait pour eux, et ma famille en général qui de par son cadre a su maintenir en moi et canaliser la curiosité, me permettant ainsi de découvrir le monde de la recherche, en particulier mes deux oncles Pierre.

Parce que les idées qui me sont venues dans mes recherches sont intimement liées à l'état d'esprit du moment, et que leur inspiration a pu venir d'une conversation anodine, d'une soirée arrosée, ou même de silences complices, je me dois aussi de remercier celles et ceux qui ont fait un bout de chemin avec moi. Il y a eu d'abord les membres des associations qui ont su m'accueillir et me transformer en me donnant un peu confiance en moi : les scouts qui m'ont fait faire mes premiers pas de pédagogue ; la F.F.J.d.R. qui m'a permis de faire mes armes en matière de gros projets ; les secouristes de la C.R.R. qui m'ont appris à rire malgré tout ; Bluebird et Blueold pour m'avoir désorganisé ; Rêves de jeux qui m'a permis de concilier animation, organisation et imaginaire ; la F.A.S.E.C.O qui m'a fait découvrir le monde de l'Université et qui m'a permis des rencontres enrichissantes ; l'A.F.N.E.U.S. et la F.A.G.E. pour l'ouverture sur les enjeux de l'Enseignement Supérieur, ses débats houleux mais enrichissants, et pour m'avoir permis des choix dont je suis fier ; A.D.D.O.C. et la C.J.C. qui m'a permis de retrouver dans un cadre mûr des débats nationaux et internationaux.

J'ai eu la chance de profiter de plusieurs personnalités lors de soirées philo, de folies, de vacances merveilleuses, de congrès retentissants, ou simplement de quelques heures éphémères à rêver :

- Jojo, malgré la différence des voies que l'on a choisi, qui est resté attaché aux mêmes valeurs de l'engagement ;
- les scouts Djé et Anne-So, Moustique et Amanda, Titi, Kim, Anne, et mes scouts modèles Max et Olivier C., ainsi que Pierre-Yves D., ;
- Patrice M., « l'archiviste », de la FFJdR ;
- les secouristes Alain, Caro, Carter et Vivi, Déb, Julie et Bastien P., JMeu, Lol, Mitch, Paul, Ruddy, Titi, Tina et Treze, Véro, et Wedge ;
- les informaticiens et assimilés Alex C., Billou, Bozo, Carbone, Cougar, Derfy, Diftong, Guignol, kobk, Lews, Marcheur, Mor, Next, Nils, Rico, Sinese, STF, Sylvia K., Taz, Vinz, et Yann P. ;
- les animateurs Anno, BerTRand, Cédric, Gawel, JPB, Kuma, Loac, Lolo, Maroussia, Pim, SkyMatt, Tôma, Yann B., Zocat ;
- les associatifs engagés Adeline D., ChrisPit, End, Fabrice H., Flo, Moby, Ptit'Rom, Rémi, Sobrette, Tom, Week, et l'équipe du Gala en général.
- les divers Fréd M., Jérémy M., Jyby, Morgann, Olivier C., Sandrine et Manu C., Séverine C..

La vie en cité universitaire m'a appris énormément au niveau humain, entre autres grâce à mes voisins et amis Carine É., Chafikha, Éva, El'Pingouino, Flo, Gaëlle, Maud P., Rahid, Romain T., Vanessa B., ...

Enfin, parce qu'ils m'ont aidé à quitter 8 années de vie associative, qu'ils ont été là en un début d'année agité, parce qu'ils ont supporté mes hauts et de mes bas, qu'il ont su me soutenir en silence ou juste été là, je tiens à remercier Leia, Dread, Hack, Lyam,

Morgoth, Pat, Puck, Bebs, Camille, Cha. Ils ont été la source de bien des changements et découvertes, bien que je ne les en ai pas toujours remercié.

En espérant qu'ils continueront de me supporter pendant longtemps encore...

J'ai sûrement oublié plusieurs personnes qui ont contribué de près ou de loin à la réussite de ma thèse, qu'ils n'en prennent pas ombrage mais qu'ils acceptent en compensation mon sourire comme preuve de ma gratitude lors de nos rencontres futures.

Table des matières

Introduction

Les systèmes pair-à-pair ont pour but la *mise en relation d'utilisateurs* (personnes ou machines) afin de *mutualiser des ressources* (processeurs, espace mémoire, fichiers). Ils sont apparus dans Internet à la fin des années 90, et ont été depuis en développement continuel. Ils ont suscité un intérêt et une passion qui ont débordé la communauté scientifique. Ces systèmes sont parfois appelés *réseaux* pair-à-pair par référence au réseau d'interconnexion des utilisateurs du système. D'abord apparus à des fins de partage de fichiers, les systèmes pair-à-pair sont utilisés depuis quelques années pour des applications variées nécessitant une décentralisation (grilles, réseaux *ad-hoc*, etc.). Depuis les débuts d'Internet, le modèle client serveur était le modèle de référence pour la mise à disposition de ressources. Dans ce modèle un serveur stable donne accès à des ressources ; il peut s'agir :

- de services, comme le fait un DNS (Domain Name System) qui traduit les noms de domaines en adresses IP ;
- de fichiers, comme le font les serveurs FTP (File Transfert Protocol) ;
- d'un contenu multimédia diffusé en continu comme les radios et télévisions en ligne ;
- ou encore de contenus plus complexe comme le font les sites qui donnent accès à des pages hypertextes ou des bases de données.

Un exemple simple met cependant en exergue la faiblesse inhérente des systèmes clients-serveur d'Internet face à une augmentation de charge. À la fin des années 1990 est apparu l'effet «Slashdot» [27] suite à la création du site d'information `slashdot.org`. Ce site publie des articles ayant trait à internet et au monde informatique, accompagnés de liens vers les sites concernés. Étant très visité, ce site engendra une publicité qui eu des effets secondaires pour les sites référencés, qui ont vu soudain leur charge habituelle démultipliée à cause d'une grande quantité de lecteurs. Lors de l'utilisation du modèle client-serveur, il est ainsi nécessaire de prévoir une augmentation potentiellement rapide de la charge. Ceci nécessite toutefois des ressources importantes, et impose des contraintes d'entretien, de bande passante, de sécurité, de charge et de rapidité d'accès à la ressource qui rendent de tels systèmes coûteux.

Les problèmes de bande passante, de sécurité et de charge peuvent être réduits en multipliant les serveurs, ou par la mise en place de la redondance des objets dans le réseau. Cette redondance permet un accès aux ressources partagées plus rapide, car les serveurs peuvent être répartis judicieusement en fonction de la localisation des utilisateurs (si on la connaît). Cependant, cette solution accroît les coûts d'entretien car le

nombre de serveurs nécessaires augmente. De ces constatations est né le besoin de migrer d'un modèle client-serveur à un modèle plus décentralisé : le paradigme pair-à-pair.

Dans un contexte où le nombre de personnes reliées par des réseaux comme Internet augmentait, ainsi que le nombre de ressources virtuellement disponibles sur ces réseaux, le premier objectif des systèmes pair-à-pair était de partager des fichiers. En effet, bien que les ressources potentiellement disponibles sur les réseaux aient augmenté en même temps que le nombre d' utilisateurs, la situation était souvent comparable à celle d'une grande bibliothèque sans index. Comme sur la «Toile», les ressources n'y étaient pas indexées de manière unique, et étaient donc difficilement accessibles. Il s'agissait pour les premiers systèmes pair-à-pair de permettre à chaque utilisateur de mettre à disposition des fichiers sans avoir à les entreposer sur des serveurs stables. C'est de là que vient le nom de «pair» : chaque utilisateur de ces systèmes peut y jouer le rôle de client ou de serveur.

Les systèmes pair-à-pair firent leur apparition durant l'été 1999 avec Napster [73]. Ce système propose le téléchargement de fichiers musicaux, mais il n'est pas encore formellement pair-à-pair, puisqu'il repose sur l'utilisation d'un serveur stable pour la mise en relation des utilisateurs. Le créateur de Napster fut pour cela facilement et rapidement mis en procès par de grands distributeurs de musique comme Time Warner, qui ont fait courir le bruit que toute utilisation des systèmes pair-à-pair était illégale.

Le second système pair-à-pair à être déployé, Gnutella [39], était parfaitement un système décentralisé. Les raisons de la décentralisation des premiers systèmes pair-à-pair ont donc pour origine les poursuites judiciaires dont les utilisateurs ont été les cibles. Cette décentralisation permet à Gnutella de fonctionner encore aujourd'hui. Gnutella symbolise bien l'arrivée des systèmes pair-à-pair dans le paysage d'Internet : par la petite porte mais remarquée. Comme nous allons le voir, le contexte de la création de Gnutella montre en particulier la crainte engendrée par la décentralisation des systèmes pair-à-pair chez certaines compagnies. Le procès possible grâce au serveur central de Napster était rendu impossible du fait de la décentralisation de Gnutella.

Le modèle pair-à-pair va bien plus loin que les applications de partage de fichiers. Il permet en effet de décentraliser des services et de mettre à disposition des ressources dans un réseau. Tout utilisateur d'un réseau pair-à-pair peut alors proposer des ressources et en obtenir sur le réseau. Les systèmes pair-à-pair permettent donc de faciliter le partage d'informations. Ils rendent aussi la censure ou les attaques légale ou pirate plus difficiles. Ces atouts font des systèmes pair-à-pair des outils de choix pour décentraliser des services qui doivent assurer une haute disponibilité tout en permettant de faibles coûts d'entretien. Des propositions sont applicables à plus ou moins long terme pour ne plus utiliser de serveurs, entre autres pour :

- les DNS,
- la mise à disposition de logiciels (distributions linux comme la Mandriva, mises-à-jour Microsoft, etc.),
- les logiciels de messagerie en ligne (skype).

L'application la plus connue actuellement reste cependant le partage de fichiers par le biais de logiciel client-serveur comme eDonkey/eMule (protocole originel eDonkey) [26], FastTrack (utilisé par KaZaA) [59], etc.

Toutefois, les systèmes pair-à-pair décentralisés ne peuvent faire appel à un serveur pour coordonner l'interconnexion des utilisateurs et assurer des faibles délais aux requêtes. C'est pourquoi sont apparus des systèmes pair-à-pair qui imposent une structure entre les utilisateurs, afin de garantir un faible diamètre : il s'agit des systèmes décentralisés *s*tructurés. Ces systèmes s'inspirent de structures de graphes statiques pour interconnecter les utilisateurs, représentés sous forme de nœuds. Ils ont ainsi pu se passer de serveurs pour assurer une répartition de la charge parmi les nœuds en terme :

- de trafic de contrôle reçu et envoyé par chaque nœud, ce qui revient à limiter leur degré ;
- de nombre de requêtes transmis à un nœud, ce qui nécessite une répartition des chemins entre les nœuds ;
- de responsabilité pour l'accès aux ressources partagées dans le réseau.

Enfin, ces systèmes ont souvent pu utiliser un routage proche de celui du graphe dont ils s'inspiraient, diminuant ainsi le nombre de messages de requêtes transitant dans le réseau.

Par ailleurs, l'étude des échanges dans les systèmes pair-à-pair a permis de proposer des alternatives pour l'amélioration de ces systèmes. Des études ont été effectuées au moyen de graphes d'échanges qui représentent les utilisateurs par des nœuds et les échanges entre deux nœuds par une arête. Plusieurs propriétés ont été observées, en particulier une distribution des degrés en loi de puissance, ce qui signifie qu'une grande partie des nœuds échangent avec peu de nœuds, tandis qu'il existe toujours des nœuds qui échangent avec un nombre important d'autres nœuds. Par ailleurs, une agrégation des nœuds dont les intérêts sont proches a été relevée, ce qui sous-entend que les échanges se font en majeure partie au sein de communautés d'intérêts. L'étude de ces propriétés a permis différentes propositions permettant :

- soit un nouveau routage efficace dans les graphes à distribution de degrés en loi de puissance ;
- soit l'amélioration de systèmes pair-à-pair déjà existants, en ajoutant à chaque utilisateur des voisins susceptibles de répondre à leurs requêtes.

Ces propositions ont pour objectif principal la réduction des délais des requêtes, et ce sans avoir à maintenir une structure entre les nœuds.

Dans cette thèse, nous nous intéressons à l'amélioration des structures d'interconnexion pour les systèmes pair-à-pair décentralisés dans le but d'assurer de *faibles délais de recherche*. En plus du temps de recherche, nous voulons permettre l'arrivée et le départ des utilisateurs au cours du temps. Nous nous appliquons aussi à permettre une recherche exhaustive, afin de toujours trouver au moins une réponse s'il en existe une dans le réseau. Pour cela, nous utilisons deux types de systèmes pair-à-pair, tous deux totalement décentralisés :

- les réseaux à contenu adressable, pour les garanties de performances qu'ils permettent ;
- les réseaux décentralisés structurés, pour assurer une recherche évoluée.

Organisation de la thèse

Cette thèse est divisée en trois parties afin de présenter les systèmes pair-à-pair dans leur ensemble, et d'aborder ensuite les deux axes de mes recherche que sont les réseaux à contenus adressable et les systèmes basés sur les propriétés observées des échanges dans les systèmes pair-à-pair.

La première partie du document présente le principe des systèmes pair-à-pair, en essayant de délimiter les grandes catégories de systèmes qui existent. En particulier :

- dans le premier chapitre, nous nous attachons à expliquer le modèle utilisé pour représenter les systèmes pair-à-pair. Nous détaillons ensuite la problématique et les propriétés recherchées dans un système idéal.
- le deuxième chapitre présente les catégories de systèmes existants actuellement, et donne des exemples pour chacune de ces catégories.

La seconde partie du document approfondit la catégorie des systèmes décentralisés structurés dans leur ensemble, ainsi que leurs applications. Plus spécifiquement :

- le troisième chapitre explique les principes des systèmes décentralisés qui maintiennent une structure de connexion entre les nœuds en abordant leurs fondements théoriques. Cela permet de poursuivre par un état de l'art des travaux représentatifs du domaine, détaillant les spécificités de chacun des protocoles et le graphe qui l'a inspiré. En particulier, nous verrons quelles furent les propositions pour améliorer :
  * le diamètre, et ainsi diminuer le temps d'une recherche ;
  * le degré des nœuds, pour limiter la charge du nombre de messages reçus et le trafic de contrôle nécessaire au maintien de la structure logique ;
  * la répartition de la charge supportée par les nœuds du réseau, afin de ne pas rendre certains nœuds plus susceptibles d'être défaillants.
- le quatrième chapitre détaille le travail qui a mené à la proposition du système *D2B*. Ce protocole permet d'assurer un diamètre logarithmique en fonction du nombre de nœuds du réseau, tout en permettant un degré moyen constant et une répartition de la charge équilibrée .

Cette thèse est clôturée par une troisième partie qui s'intéresse aux propriétés observées dans les échanges entre les utilisateurs de systèmes pair-à-pair. Nous y décrivons différentes propositions d'utilisation de ces propriétés pour améliorer la recherche dans les systèmes pair-à-pair.

Le cinquième chapitre décrit les propriétés observées dans les graphes d'échanges : distribution des degrés en loi de puissance et agrégation des nœuds en communauté. Nous verrons ensuite quelles ont été les propositions d'utilisations de chacune de ces deux propriétés, que ce soit pour l'amélioration du routage dans les graphe à distribution de degrés en loi de puissance, ou l'utilisation des communautés pour trouver plus rapidement des réponses aux requêtes des nœuds.

Nous proposons dans le sixième chapitre une méthode baptisée *QRE*, qui tire avantage des deux propriétés de distribution des degrés en loi de puissance et d'agrégation des nœuds en communautés, jusque là exploitées séparément. Cette méthode ne requiert pas l'utilisation d'un quelconque système sous-jacent. Elle permet d'obtenir des temps de recherches et un diamètre moyen comparables aux réseaux à contenu adressable tout en ne demandant pas le maintien d'une structure spécifique entre les nœuds et en autorisant des requêtes bien plus évoluées.

Nous concluons cette thèse par la présentation de différentes possibilités d'utilisation des systèmes pair-à-pair dans des domaines plus larges que les systèmes actuels de partages de fichiers, selon la faisabilité et l'état de l'art actuel.

Première partie

Vue d'ensemble des systèmes pair-à-pair

Dans cette partie, nous allons présenter un aperçu des systèmes pair-à-pair. Nous aborderons d'abord les différentes tâches que peuvent remplir ces systèmes, puis nous verrons comment ils peuvent être modélisés. Nous terminerons enfin cette partie en décrivant les critères d'efficacité permettant d'atteindre les objectifs de ces systèmes. Nous verrons en particulier l'impact que peuvent avoir ces critères les uns sur les autres afin de choisir le meilleur compromis selon l'objectif visé.

Chapitre 1

La problématique des systèmes pair-à-pair

Dans ce chapitre, nous allons présenter brièvement les systèmes pair-à-pair. Nous énoncerons les objectifs de ces systèmes, leur organisation et leurs caractéristiques. Nous verrons ensuite comment modéliser les systèmes pair-à-pair. Enfin, nous aborderons la problématique liée à ces systèmes au moyen du modèle, en listant point par point les difficultés posées par les systèmes pair-à-pair.

Les systèmes pair-à-pair sont des systèmes visant à permettre à des utilisateurs la mise en commun d'objets par des utilisateurs et leur recherche en vue de leur récupération (pour des fichiers) ou de leur utilisation (pour des ressources de calcul). Dans la suite, le terme *objet* sera utilisé pour nommer indistinctement une ressource de calcul, un fichier, ou une autre information (entrée DNS par exemple), et le terme *récupération* sera utilisé qu'il s'agisse d'obtenir un accès à une ressource de calcul, de télécharger un fichier, ou d'obtenir une autre information.

Les systèmes pair-à-pair ont plusieurs caractéristiques les distinguant des autres systèmes de partage d'objets.

- Ils permettent de représenter des échanges sociaux au sens où ils lient des utilisateurs qui interagissent de manière humaine. Les objets sont échangés selon les intérêts des utilisateurs.
- Le nombre d'utilisateurs d'un système pair-à-pair peut être très important, de l'ordre du millier au million selon les systèmes [93].
- ils sont dynamiques, car ils doivent permettre à chaque instant l'arrivée et le départ d'utilisateurs.
- Ils sont décentralisés, au moins en partie : la récupération des objets, voire la recherche d'objets, ne nécessite pas l'utilisation de serveurs stables.

Le sujet de cette thèse est la recherche de données dans les systèmes pair-à-pair totalement décentralisés, et l'interconnexion nécessaire pour rendre cette recherche efficace. Le terme de *réseau* pair-à-pair sera utilisé pour désigner un réseau d'utilisateurs créé par un système pair-à-pair. La dénomination de *système* pair-à-pair sera utilisée pour parler du fonctionnement des protocoles de recherche et de publication, et non uniquement de

l'interconnexion des utilisateurs. La récupération des objets sera à l'inverse peu abordée dans cette thèse.

Dans ce document, l'étude des réseaux pair-à-pair s'effectue essentiellement au travers d'une modélisation par des graphes. Nous allons voir dans la suite comment utiliser le formalisme de la théorie des graphes dans le cadre des systèmes pair-à-pair.

1.1 Modèle utilisé

Permettre la recherche d'objets dans les systèmes pair-à-pair demande de répondre à plusieurs questions. Comment les nœuds sont-ils connectés entre eux ? Comment sont envoyés les messages dans le réseau ainsi créé ? Afin d'utiliser une terminologie claire, les notations utilisées dans cette thèse sont détaillées ci-dessous.

Un système pair-à-pair fait évoluer des machines connectées selon un protocole propre. Un réseau *logique* est l'interconnexion qui relie virtuellement les utilisateurs, au dessus d'un réseau *physique* permettant à toute machine de communiquer avec toute autre machine. Le réseau logique repose sur des connexions logicielles maintenues grâce à un protocole de communication (par exemple TCP/IP).

Un *nœud* représente un processus client-serveur exécuté sur une machine et permettant à un utilisateur d'utiliser le système pair-à-pair. Chaque nœud peut être tour à tour demandeur ou fournisseur d'un objet du système. Il peut jouer pour chaque requête le rôle de :

- client, lorsqu'il cherche un objet dans le réseau ;
- serveur, lorsqu'il fournit un objet cherché par un autre nœud ;
- routeur, lorsqu'il reçoit une requête qui devra être dirigée vers d'autres nœuds.

On nommera *logiciel client-serveur* un logiciel permettant d'accéder à un système pair-à-pair via un protocole donné (et permettant d'être client *et* serveur). Dans un système pair-à-pair, chaque nœud peut se servir d'un logiciel client-serveur différent, tant que ce logiciel est compatible avec le protocole utilisé. Afin de préciser le rôle d'un nœud à un instant donné, on pourra caractériser un nœud *client*, pour un nœud effectuant une recherche, un nœud *serveur*, pour un nœud fournissant un objet, et un nœud *routeur* pour un nœud transmettant un message dans le réseau.

Un *lien* entre deux nœuds symbolise une connexion logicielle permettant la communication entre ces deux nœuds. Ces liens ont plusieurs caractéristiques.

- Un lien passe par un ou plusieurs liens physiques.
- La traversée de chacun de ces liens peut prendre un temps arbitrairement long car les réseaux physiques (et donc les liens physiques) au dessus desquels sont créés les réseaux pair-à-pair sont souvent asynchrones.
- Enfin, si certains systèmes pair-à-pair utilisent des liens orientés pour le réseau logique, la plupart de ces systèmes sont basés sur le protocole réseau IP, qui permet de connaître l'origine d'un message. Il est alors possible et facile de connaître les liens entrants, même dans des systèmes pair-à-pair utilisant des liens orientés.

Les systèmes pair-à-pair sont *dynamiques*, c'est-à-dire que chaque nœud peut arriver et repartir au cours du temps. Les utilisateurs doivent en effet pouvoir s'y connecter

ou s'en déconnecter à volonté. La propriété précédente (forte connexité) doit bien sûr rester vérifiée au cours du fonctionnement du système, donc en cas de déconnexion de plusieurs nœuds.

Un envoi d'informations d'un nœud à un autre au sein du réseau sera nommé *message*. Lorsque ce message correspond à une recherche d'objets, il sera nommé *requête*.

1.2 La problématique des systèmes pair-à-pair

Les systèmes pair-à-pair ont été créés pour répondre à différents objectifs. Nous allons détailler dans ce chapitre chacun de ces objectifs tout en montrant quels moyens sont mis en œuvre dans quel systèmes pour les atteindre. Certains de ces moyens sont parfois incompatibles. Cependant, selon le type d'application du système pair-à-pair, certains objectifs sont moins prioritaires, ils peuvent donc être écartés pour en favoriser d'autres.

1.2.1 Gérer de nombreux utilisateurs

Les systèmes pair-à-pairs sont des systèmes à *grande échelle* qui doivent permettre un nombre d'utilisateurs variant sur de grands intervalles. Cette gestion peut s'effectuer au moyen de serveurs ou de manière décentralisée.

Dans le cas de systèmes pair-à-pair partiellement centralisés, le nombre d'utilisateurs gérés est limité du fait des capacités (de calcul, de mémoire, d'espace disque, ou encore de communication) des matériels et des logiciels utilisés par chaque serveur (voir chapitre 1.7 de [61]). Augmenter le nombre d'utilisateurs et d'objets au delà de cette limite entraîne alors des coûts financiers en matériels et logiciels importants. C'est pourquoi les systèmes partiellement centralisés doivent faire un compromis entre les capacités nécessaires au fonctionnement des serveurs gérant un grand nombre d'utilisateurs, et le coût de ces serveurs et de leur maintenance. L'une des possibilités d'augmentation du nombre d'utilisateurs et des objets dans un système partiellement centralisé est la mise en commun de serveurs. Cela pose alors les problèmes de la répartition de ces serveurs, de leur interconnexion, de la cohérence des listes d'objets, etc. Notons que les solutions trouvées pour résoudre ces problèmes dans le cadre des architectures de super-calculateurs peuvent être réutilisées. D'autres systèmes de recherche comparables aux systèmes pair-à-pair partiellement centralisés illustrent l'inconvénient que nous venons de décrire. Les deux premiers moteurs de recherche les plus utilisés, Google et Yahoo, bien que restant très secrets sur leur infrastructure, annonçaient respectivement 100.000 serveurs et 95.000 serveurs pour gérer leur système de recherche en 2003, ce qui représente un coût certain, tant à l'achat qu'en maintenance.

Toutefois, dans les systèmes pair-à-pair partiellement centralisés, la charge imposée aux serveurs permet de décharger les autres nœuds de tâches comme le maintien du réseau logique ou de la recherche des objets. En effet, la charge supportée par ces nœuds est alors proportionnelle au nombre d'objets que proposent ces nœuds, et à la popularité de ces objets : les messages reçus par un nœud proviennent de nœuds demandant accès à ses objets, ou du serveur (pour le maintien du réseau). À l'inverse, les systèmes

décentralisés ne nécessitant pas de serveur, ils n'ont donc aucun coût d'infrastructure centrale.

1.2.2 Tolérance aux pannes et charge du réseau

Un système pair-à-pair doit continuer de fonctionner en cas de pannes de certains nœuds.

Origine des pannes

Les causes de ces pannes peuvent être naturelles, par exemple un afflux de messages plus important que d'habitude. C'est ce qui arrive lors de l'effet «Slashdot» [27] en cas de publicité importante pour un site HTTP qui engendre un trafic trop important pour le serveur hébergeur. Mais ces pannes peuvent également être dues à des attaques pirates, telles que le déni de service dont fut victime le service de DNS d'Internet. Le service DNS a été organisé en hiérarchie afin de ne pas dépendre d'un seul serveur, tout en permettant une recherche simple. Cette réorganisation n'a cependant pas suffit à éliminer ses points faibles. Les DNS fonctionnent comme suit : des serveurs racines administrent les extensions principales (`.fr, .com, .org., .net`, etc.), tandis que chaque nœud à l'intérieur de l'arbre DNS gère un ou plusieurs noms de domaine (`wikipedia.org, atilf.fr, u-psud.fr`, etc.). Les feuilles de cet arbre correspondent à des machines hébergeant le site HTTP, pour une adresse hébergeant un serveur HTTP (par exemple `fr.wikipedia.org, atilf.atilf.fr, www.u-psud.fr`, chacune de ces adresse correspond **directement** à un site HTTP). Cet arbre des DNS n'est pas équilibré, c'est-à-dire que certains noms de domaines sont très long donc la hiérarchie comprend beaucoup de DNS, tandis que d'autre sont très courts, et comprennent peu de DNS. De manière grossière, lorsqu'un navigateur demande la traduction d'un nom de domaine à un DNS de sous-réseau (du laboratoire, de l'entreprise, etc.), celui-ci répond si le nom de domaine (et l'adresse IP correspondante) est dans son cache. Sinon, il retransmet la requête à son DNS « père ». En haut de cette hiérarchie de serveurs se trouvent 13 serveurs racines (`http ://www.root-servers.org`). Le 21 octobre 2003 eût lieu sur ces 13 serveurs une attaque généralisée de type déni de service réparti, basée sur des requêtes de ping. Durant une heure, des requêtes inondèrent les serveurs racines, en faisant ainsi tomber en panne 9 serveurs sur les 13. Sans le service des DNS, Internet est inutilisable. Cet événement a donc achevé de convaincre de la nécessité de décentraliser l'architecture de ce service et de le répartir géographiquement (réflexion entamée après les attentats du World trade center) : en effet, 10 des 13 serveurs se trouvaient alors aux États-Unis, dont certains dans les mêmes villes.

Cela illustre la sensibilité des systèmes en partie centralisé face à une charge importante sur quelques nœuds ou les arêtes attenantes, de même que l'effet «Slashdot». Les attaques judiciaires ont, elles, rappelé la faiblesse inhérente des systèmes basés sur certains nœuds essentiels pour le système. Ces attaques, bien que ne révélant pas une faiblesse technique mais un problème politique et économique, ont influencé les attentes des utilisateurs, et donc la façon dont sont conçus les systèmes pair-à-pair.

Tolérance aux pannes

Pour nommer les pannes qui peuvent intervenir, nous utiliserons un classement en trois types :

- la *panne d'arrêt* : il s'agit d'un nœud quittant le réseau sans prévenir, par accident ou volontairement ;
- la *panne intermittente* : un nœud victime d'une telle panne ne répond aux requêtes qui lui sont envoyées que durant certains intervalles de temps ;
- la *panne byzantine* survient lorsqu'un adversaire, qui peut avoir une connaissance globale du réseau et du fonctionnement du système, tente de rendre le fonctionnement du système incorrect.

Un système pair-à-pair devrait idéalement gérer ces trois types de pannes.

Les pannes d'arrêt : ces sont les dysfonctionnements les plus simples à gérer : il suffit de s'assurer que si des nœuds tombent en panne, le réseau pair-à-pair continuera de permettre la recherche d'objets parmi les nœuds restés connectés. En particulier, les messages doivent continuer à être correctement reçus par les destinataires, le réseau doit donc toujours rester connexe.

Afin que le système pair-à-pair continue de fonctionner, les nœuds indispensables à la recherche d'objets doivent rester connectés. Dans des systèmes partiellement centralisés, il faut donc s'assurer que les serveurs centraux ne puissent être déconnectés. Cette propriété est difficile à assurer.

Un système décentralisé n'a en revanche aucune précaution particulière à prendre vis-à-vis de nœuds spécifiques, puisqu'aucun nœud particulier n'est nécessaire à son fonctionnement. Dans ce cas, pour assurer que l'on puisse toujours envoyer des messages, une solution classique consiste à vérifier régulièrement que les voisins de chaque nœud soient toujours connectés au réseau. L'une des méthodes les plus utilisées pour cela est de demander à chaque voisin d'envoyer régulièrement un message signalant sa présence. Il est aussi possible d'imposer aux nœuds du réseau un nombre de voisins suffisamment important pour que la probabilité qu'un nœud soit déconnecté du réseau du fait de pannes de ses voisins puisse être considérée comme négligeable. En cas de déconnexion d'un voisin, ses voisins le remplacent rapidement par un autre nœud, afin de diminuer le risque de déconnexion du réseau.

Les pannes d'intermittence : les pannes d'intermittence sont gérées trivialement lorsque le voisin d'un nœud ne répond plus aux messages. Ce voisin muet est alors considéré comme déconnecté par ce nœud. On se ramène ainsi au cas de la panne d'arrêt. C'est actuellement la méthode utilisée par la plupart des systèmes pair-à-pair.

Les pannes byzantines : afin de détecter et résoudre les pannes byzantine, on rencontre deux cas :

- dans un système partiellement centralisé, il est nécessaire de disposer de serveurs incorruptibles si l'on veut assurer une recherche d'objets correcte. Cela permet

d'assurer des réponses correctes lors d'une recherche, mais c'est une hypothèse très forte.
- dans des systèmes décentralisés, il est possible de tolérer les pannes byzantines dans une certaine limite comme le permettent les travaux d'algorithmique répartie. Le coût est alors une augmentation du trafic moyen dans le réseau

En pratique, seules les pannes d'arrêt sont réellement prises en charge par les systèmes pair-à-pair. Les pannes d'intermittences sont en fait gérées en les assimilant à des arrêts, et les pannes byzantines ne sont pas gérées. De manière générale, des attaques sur un nombre constant de nœuds ou d'arêtes choisis aléatoirement ne doivent pas gêner l'efficacité du réseau de manière significative, puisque ce nombre reste très inférieur au nombre de nœuds composant le réseau. Le système doit aussi continuer de fonctionner correctement tant que des nœuds y sont connectés. Entre autres, le réseau ne doit pas se retrouver fractionné en plusieurs réseaux déconnectés entre eux. Idéalement, la seule façon de rendre un système pair-à-pair inutilisable devrait être d'attaquer une fraction du nombre de nœuds composant le réseau.

1.2.3 Équilibrage de la charge

De manière générale, lorsque le nombre de nœuds connectés à un système pair-à-pair augmente, le nombre de messages transitant dans le réseau augmente d'autant. Pour éviter de rendre les nœuds du système sensibles à une surcharge de messages (et engendrer des pannes d'arrêt ou d'intermittence), un système pair-à-pair peut limiter la charge supportée par les nœuds et les arêtes. Un nœud ou une arête faiblement chargé sera en effet moins sensible à l'arrivée massive d'un grand nombre de messages supplémentaires qu'un nœud recevant en temps normal de nombreux messages. De plus, puisque tout nœud peut être victime de cet événement, il est nécessaire de répartir la charge des messages parmi tous les nœuds du réseau, et d'éviter les points faibles comme les serveurs.

La charge d'un nœud peut être de plusieurs types, car un nœud reçoit des messages à plusieurs titres :

- des messages de contrôle, nécessaire au maintien de l'interconnexion du réseau ;
- des messages de recherche d'objets (qu'il reçoit en tant que routeur) ;
- des messages de demande pour des objets qu'il partage (qu'il reçoit en tant que serveur) ;
- et des messages de réponse à ses recherches d'objets (qu'il reçoit en tant que client).

Par ailleurs, pour que la décentralisation de la charge des messages soit efficace, le nombre total de messages reçus par un nœud doit croître moins vite asymptotiquement que le nombre de nœuds du réseau. Il en va de même pour le nombre de nœuds impliqués dans le routage d'une requête, et le nombre de messages nécessaire à l'insertion ou au départ d'un nœud.

Trafic de contrôle

Si l'interconnexion des systèmes pair-à-pair centralisés est gérée par des serveurs, les systèmes pair-à-pair décentralisés doivent être auto-organisés puisqu'aucun point central ne peut organiser le réseau. Ils maintiennent donc leur infrastructure de manière locale, ce qui entraîne un trafic de contrôle qui doit idéalement engendrer un faible nombre de messages et n'impliquer que peu de nœuds. Ainsi, c'est souvent le proche voisinage des nœuds qui est mis à contribution pour les arrivées et les départs des nœuds dans le système.

Demande d'accès aux objets

Un serveur reçoit des messages pour avoir accès aux objets qu'il propose. Ce nombre de messages variant selon le nombre et la popularité des objets que chaque nœud propose. On dira d'un nœud proposant de nombreux objets ou des objets populaires qu'il est *populaire*. Si les modèles utilisés pour l'évaluation des systèmes pair-à-pair ont longtemps supposé un nombre d'objets et une popularité identique pour tous les nœuds, plusieurs travaux [29, 53] se sont appesantis entre autres sur :

- l'étude de la popularité des fichiers ;
- leur rareté ;
- leur répartition ;
- les nœuds qui hébergent les fichiers.

D'autres contributions ont proposé des méthodes de réplication [36, 65] des objets afin de diminuer la rareté des objets populaires et la charge des nœuds populaires tout en augmentant la disponibilité de ces objets. Ces différentes propositions pourraient être utilisées pour choisir ou influencer l'interconnexion des nœuds, selon la popularité des objets qu'ils proposent.

Messages de recherche

Un système pair-à-pair dont les requêtes nécessitent un grand nombre de sauts chargera plus de nœuds routeurs qu'un système dont les requêtes nécessitent peu de sauts dans le réseau. La minimisation du nombre de sauts par requête permet donc aussi de diminuer la charge totale du réseau.

Par ailleurs, si l'on suppose que tous les nœuds sont également populaires, on peut considérer que tous les chemins possibles entre un nœud client et un nœud serveur sont également susceptibles d'être empruntés pour une recherche. Le nombre de messages de recherche reçus par un nœud est donc directement proportionnel au nombre de chemins qui le traversent. Un système peut alors équilibrer le nombre de messages de recherche reçus par chaque nœud en s'assurant que chaque nœud est situé sur un même nombre de chemins. Cette répartition des chemins est triviale dans les systèmes pair-à-pair centralisés et se révèle faisable dans des systèmes pair-à-pair imposant une structure aux nœuds [33, 68].

Dans les systèmes décentralisés qui n'imposent aucune structure, les solutions adoptées pour diminuer le nombre de messages de recherche sont basées sur la diminution

du nombre de sauts des requêtes [15, 65, 102]. La connaissance des chemins qui peuvent passer par un nœud est en effet plus difficile à obtenir dans des systèmes ne maintenant pas de structure spécifique. De plus, considérer que tous les nœuds ont une popularité égale n'est pas une simplification réaliste [50].

Enfin, de manière locale, les messages reçus par un nœud en tant que routeur dépendent de l'interconnexion des nœuds dans le système. Un nœud de degré important sera plus susceptible de recevoir des messages de recherche, alors qu'un degré faible permet de diminuer le nombre de messages moyens reçus par un nœud. Toutefois, le nombre de requêtes de recherche envoyées par les nœuds doit être supposé du même ordre de grandeur, et le trafic doit être équitablement réparti sur les nœuds du système pour espérer une diminution du nombre de messages de recherche dans le réseau.

Messages de réponses à des recherches

Enfin, le nombre de messages reçus comme client, c'est-à-dire comme demandeur d'objets, dépend essentiellement du nombre de demandes effectuées. Il est difficile d'influer sur ces demandes. La plupart des systèmes pair-à-pair n'étudient pas ce type de messages, et considèrent implicitement que ces messages ne déséquilibrent pas la charge supportée par chaque nœud. Afin de gérer la différence de charge engendrée par les nœuds actifs (effectuant un grand nombre de recherche) sur leurs voisins par rapport aux autres nœuds, on peut imaginer plusieurs solutions. Il est d'abord possible d'augmenter le nombre de voisins des nœuds actifs, afin qu'ils répartissent l'envoi des messages de recherche entre un plus grand nombre de voisins. On peut aussi diminuer la charge des voisins des nœuds de manière indirecte, en diminuant leur degré, ou directe en imposant à leurs autres voisins de leur envoyer un faible nombre de messages.

1.2.4 Autonomie et envoi à des sous-réseaux

Dans certains cas, un nœud peut désirer choisir les nœuds ou les liens du réseau physique qui peuvent être utilisés par ses requêtes, il en va de même pour le réseau logique. Les raisons de ces choix peuvent être diverses :

- la confiance limitée dans certaines ressources du réseau dans les cas où l'on veut assurer la confidentialité en plus de l'arrivée des messages ;
- le droit limité à l'accès à certaines ressources ;
- l'optimisation à partir d'informations locales. Il peut s'agir d'utiliser certains nœuds qui donneront des réponses plus pertinentes, des liens peu utilisés, ou de n'utiliser que le réseau physique local ;
- des choix politiques.

Pour ces différents systèmes, et en particulier le cas du manque de confiance, des algorithmes auto-stabilisants peuvent être utilisés en plus du chiffrement des messages.

Le dynamisme des nœuds du système rend difficile l'identification des nœuds à utiliser ou non, car un nœud peut changer d'identifiant entre deux connexions au réseau, une identification non falsifiable et sûre aidera alors grandement. L'interdiction des liens *physiques* est toutefois plus difficile à mettre en œuvre, car elle nécessite l'utilisation d'un

protocole de routage permettant de forcer ou d'interdire l'utilisation de liens physiques à chaque nœud routeur.

À ce jour, aucun système étudié ne permet à ma connaissance de préciser les nœuds ou les liens physiques qui peuvent être empruntés par des messages dans un systèmes pair-à-pair, cette partie de la problématique ne sera pas traitée dans cette thèse.

1.2.5 Accès au réseau

Le problème de l'accès au réseau est essentiel pour les systèmes pair-à-pair. En effet, pour se connecter à un système pair-à-pair, un nouveau nœud doit disposer d'un point d'entrée dans ce réseau, comme l'adresse d'un nœud déjà inséré au réseau, ou l'adresse d'une site HTTP disposant de telles adresses. Un système pair-à-pair étant au moins partiellement décentralisé, il ne dispose pas forcément d'un serveur permettant cet accès au réseau.

Dans un système centralisé, quelques serveurs sont stables, ils restent toujours connectés et jouent le rôle de point d'entrée dans le réseau. Il est donc facile d'inclure dans chaque logiciel client-serveur pair-à-pair l'adresse physique d'un de ces serveurs, ou un moyen de la retrouver, par exemple une adresse DNS. De cette manière, lorsqu'un nouveau nœud désire s'insérer dans un système pair-à-pair, il lui est facile de contacter un serveur. Dans un système semi-décentralisé, où les serveurs sont stables, il est aussi facile de contacter un serveur pour se connecter au réseau.

L'accès aux systèmes décentralisés a été étudié dès 2000 par le premier système totalement décentralisé, Gnutella [48, 51]. Dans un système où tous les nœuds sont *a priori* volatiles, il n'est pas possible d'intégrer aux logiciels clients-serveurs une adresse de nœud à contacter qui serait valable quelque soit le moment où le nœud voudra rejoindre le système. En effet, quelque soit l'adresse de nœud qui serait choisie pour le logiciel client-serveur, le nœud correspondant est susceptible d'être déconnecté du système lorsque le nouveau nœud se connecte. C'est Gnutella qui a le premier dû résoudre un réel problème d'accès au réseau. Les premiers moyens utilisés furent des moyens de communication standards indépendants du protocole pair-à-pair, comme IRC (Internet Chat Relay, système de communication hiérarchique), ou un site HTTP. Ces médias pérennes servaient à diffuser aux nouveaux nœuds des adresses physiques pour de se connecter. Les nœuds s'étant déjà connectés une fois au système peuvent aussi tenter de se reconnecter aux adresses physiques de leurs voisins des précédentes connexions, au cas où ces voisins seraient encore connectés au système.

Afin de rendre les systèmes autonomes, il a ensuite été proposé de mettre à disposition des nouveaux nœuds des serveurs de cache inter-connectés. Ces serveurs ne sont pas des nœuds du système mais des serveurs qui maintiennent une liste d'adresses physiques de nœuds connectés au système, pour faciliter l'insertion des nouveaux nœuds. Un serveur de cache fournit aux nouveaux nœuds des adresses physiques de nœuds qui se sont connectés au système, afin de permettre aux nouveaux arrivants de s'insérer dans le système. Chaque nouveau nœud contactant un serveur de cache voit son adresse ajoutée à la liste des adresses de nœuds connus de ce serveur. C'est de cette liste que le serveur tire les adresses qu'il fournit aux nouveaux nœuds. Ce système fut d'abord

appliqué à Gnutella sous le nom de «gnuCache», puis «Gwebcache» [51], proposés par Gnucleus.

Dans un système pair-à-pair, les points d'entrée dans le réseau (cache ou autres) doivent être être répartis afin d'augmenter la tolérance aux pannes (voir chapitre 1.2.2), limiter la charge du réseau physique (voir chapitre 1.2.1), et diriger les nouveaux nœuds vers des serveurs proches physiquement. Cette dernière optimisation présente l'intérêt de connecter les nouveaux nœuds à des nœuds eux-mêmes proches dans le réseau physique, afin de réduire les délais de communication (voir chapitre 1.2.9).

Certains systèmes pair-à-pair dédiés à des communautés restreintes d'utilisateurs demandent une identification à chaque connexion, grâce à un serveur qui peut servir de point d'accès. Dans ce cas, le premier accès se fait exclusivement par parrainage, grâce à un nœud qui est déjà admis dans le système et connaît donc le serveur.

Il est à noter que certains protocoles comme giFT-FastTrack [38], tentent de se connecter à des adresses physiques tirées aléatoirement dans un sous-ensemble d'adresses possibles au cas où les autres méthodes d'insertion ont échoué. Cette méthode inonde partiellement le réseau, ce qui engendre donc un grand nombre de messages. De plus, si les méthodes qu'elle est censée remplacer ne fonctionnent pas, il est probable que le réseau physique soit déjà surchargé. En effet, si ni les nœuds des précédentes sessions, ni le site HTTP ne répondent aux messages, il faut envisager que le réseau est surchargé. Cette méthode d'accès au système est donc à éviter pour trouver des nœuds auxquels se connecter.

1.2.6 Protection des utilisateurs

Depuis les débuts des systèmes de télécommunication, des organismes exercent ou tentent d'exercer un contrôle, qu'il s'agisse de censure de contenu ou d'espionnage des utilisateurs, et ce de manière pirate ou légale. Un contrôle peut être mis en place par des organismes pour surveiller les comportements des utilisateurs ou empêcher l'accès à des objets dans des pays non démocratiques. La Chine a par exemple fait pression sur Google pour que certains sites ne soient pas présentés dans les résultat de la version chinoise du moteur de recherche. Mais des gouvernements pourraient aussi désirer lutter contre la diffusion d'incitations au racisme, à la pédophilie, etc. Enfin, depuis quelques années, quelques grandes entreprises de disques luttent contre la diffusion de fichiers soumis à droits d'auteur et demandent pour cela un contrôle des systèmes pair-à-pair.

Ce chapitre concerne les conséquences d'un comportement politique et économique sur le développement des systèmes pair-à-pair. Internet n'est actuellement pas anonyme, puisque TCP/IP impose que pour toute communication bidirectionnelle entre deux nœuds, chacun connaisse l'adresse physique de l'autre. Les communications dans les systèmes pair-à-pair sont souvent bidirectionnelles puisque ces systèmes sont basés sur l'échange d'informations ou la demande d'objets et leur transfert. Dans ce contexte, la complexité du recensement des adresses physiques qui accèdent à un objet dépend du système dans lequel est effectué ce recensement. Dans un système centralisé, il suffit de s'introduire dans le serveur central pour savoir qui partage quoi. L'obtention de cette information par les nœuds peut selon les systèmes être autorisée ou pas. Dans

un système décentralisé, un espion pourra ouvrir les requêtes transitant par lui pour obtenir cette information. Dans un système hybride, les deux méthodes seront possibles. Dans la quasi-totalité des systèmes (à part les systèmes anonymes comme Freenet [17]), les nœuds peuvent aussi surveiller quel nœud récupère un objet qu'il partage.

Bien que savoir *qui cherche quoi* ne signifie pas savoir *qui récupère quoi*, cette information semble servir de moyen de pression à des entreprises de distribution de disques et des associations [85] pour lutter contre les utilisateurs de systèmes pair-à-pair. Des organismes se sont donc spécialisés dans la récupération d'informations sur les utilisateurs [23], parfois avec un objectif offensif affiché comme l'association Retspan [85]. En effet, la correspondance entre les adresses physiques (adresses IP) et l'identité de l'utilisateur est connue par le F.A.I. (Fournisseur d'Accès à Internet, comme Wanadoo, Télé2 ou Free). La correspondance entre des informations personnelles et l'identité d'un utilisateur est donc évidemment accessible à un gouvernement, et à d'autres organismes dans certains pays.

Un système basé sur des serveurs voyant toutes les requêtes des utilisateurs ne peut assurer la confidentialité des utilisateurs que si les serveurs sont sûrs et incorruptibles. C'est une contrainte très forte, et en fait impossible à assurer. En effet, prendre le contrôle d'un système pair-à-pair basé sur des serveurs ne nécessite que la prise de contrôle, de manière légale ou pirate, d'un petit nombre de nœuds (les serveurs). Ce contrôle permet ensuite par exemple de censurer ou d'espionner les utilisateurs.

Or les attaques de serveurs ne sont pas rares et les précédents ne manquent pas. Des fichiers de dizaines de milliers de numéros de cartes bancaires furent piratées dans des banques des États-Unis il y a une dizaine d'années. DoubleClick, site de publicité en ligne, fut probablement piraté dès 1999, permettant ainsi l'accès à des informations personnelles des visiteurs de 13.000 sites Internet [24]. Concernant spécifiquement les systèmes pair-à-pair, de grandes entreprises luttant contre ces systèmes ont réussi à racheter les informations des utilisateurs enregistrées sur des serveurs, à la suite de pressions juridiques. Cette insécurité a contribué à renforcer les inquiétudes concernant la mémorisation d'informations sur les utilisateurs, et à encourager le développement de systèmes ne nécessitant pas de mémorisation centralisée des informations. Les utilisateurs désirent rester *anonymes* tout en proposant et en demandant des objets au système. Toutefois, dans un système décentralisé, il est toujours possible d'espionner les requêtes en transit. C'est pourquoi Freenet [17] a proposé l'utilisation de méthodes cryptographiques pour rendre l'identification des utilisateurs difficile, et se déclarer réellement anonyme. Toutefois, nous verrons que ce n'est pas sans coût en terme de rapidité et de souplesse de recherche. Freenet fait le choix de passer par des intermédiaires, afin d'anonymiser la recherche et le transfert d'objets, ce qui entraîne une augmentation des temps de recherche, donc une diminution de la *rapidité* et de la *vivacité*.

1.2.7 Dynamisme des nœuds

Contrairement aux graphes statiques, où les nœuds et les arêtes sont fixes et ne changent pas avec le temps, un réseau pair-à-pair doit permettre les nombreuses connexions et déconnexions de nœuds au fil du temps, et ce sur de faibles intervalles de temps. Les

nœuds sont donc très dynamiques, on retrouve donc là encore la problématique des algorithmes auto-stabilisant.

Le délai de connexion d'un nœud dépendant directement du nombre de voisins auxquels il doit se connecter, un *degré faible* est un avantage pour permettre un fort dynamisme des nœuds. Par ailleurs, tout comme pour la durée d'une requête, le délai de connexion d'un nœud sera à mettre en rapport avec le temps durant lequel ce nœud restera connecté au système.

Enfin, pour éviter une trop forte charge des nœuds, le coût (en nombre de messages) de l'arrivée ou du départ d'un nœud doit croître moins rapidement que le nombre de nœuds dans le réseau.

Systèmes pair-à-pair en partie centralisés. Dans ce cas, un nœud n'est connecté qu'à un seul serveur donc les procédures de connexion et de déconnexion sont rapides.

Systèmes pair-à-pair décentralisés. Dans ces systèmes où les nœuds doivent se connecter à des nœuds spécifiques (système décentralisé non-structurés), l'arrivée et le départ dans le réseau sont rapides. Dans les systèmes structurés, un nœud doit maintenir des voisins dont l'identité répond à certaines contraintes (systèmes décentralisés structurés), et le dynamisme s'en ressent. Parmi ces systèmes, la plupart [33, 70, 83, 96, 100] ont un degré qui évolue avec le nombre de nœuds dans le réseau. Cela les rend plus efficaces, notamment au démarrage, que des systèmes [88, 105] dont le degré dépend du nombre de nœuds *maximum* dans le réseau.

1.2.8 Réactivité des nœuds

Pour assurer un service rapide, un système pair-à-pair doit pouvoir transmettre des messages en permanence, que ce soit pour envoyer des requêtes, y répondre, ou en acheminer. Tout nœud doit évidemment rester accessible tant qu'il ne désire pas quitter le réseau, comme nous l'avons vu pour la tolérance aux pannes (voir chapitre 1.2.2), et il doit pouvoir remplacer rapidement ses voisins défaillants. Un nœud doit donc être informé rapidement des défaillances de ses voisins afin de les remplacer et rester connecté au réseau. La méthode employée de manière générale pour détecter les défaillances des voisins est que chaque nœud envoie régulièrement des messages de contrôle à ses voisins. Chaque nœud vérifie ainsi que ses voisins assurent toujours le fonctionnement du système (*via* le logiciel approprié) et qu'ils sont toujours connectés au nœud. Cependant, comme nous l'avons vu au chapitre 1.2.3, le trafic de contrôle engendré dans le réseau logique, et donc dans le réseau physique, est d'autant plus important que les voisins d'un nœud sont nombreux. C'est pourquoi dans les systèmes centralisés ou semi-décentralisés, les serveurs ne peuvent vérifier que tous les nœuds sont toujours connectés. Ils remettent à jour leur liste de nœuds connectés lorsqu'un message ne peut être envoyé à un nœud. Vérifier que les voisins sont connectés est coûteux en terme de messages, et les systèmes pair-à-pair limitent au maximum ce contrôle de vivacité des nœuds. Par exemple, eMule autorise aux nœuds non serveurs un maximum de $0,2$ messages UDP par seconde pour vérifier qu'un serveur est toujours accessible, et le serveur n'effectue aucun contrôle [61].

Dans un système hybride comme giFT-FastTrack [38], un message de contrôle n'est envoyé qu'après plusieurs minutes de silence du voisin. Enfin, les systèmes décentralisés comme Gnutella [39] ont laissé le taux d'envoi à la discrétion du logiciel client-serveur. Après avoir observé que certains nœuds faibles ralentissaient le système Gnutella, une autre modification fût toutefois proposée. Il s'agit pour les nœuds de prendre pour voisins les nœuds qui ont le plus fort débit et qui sont les plus réactifs. Ceci eut pour effet de limiter le degré des nœuds peu réactifs et les liaisons bas-débit (comme les modems), donc de les repousser vers les extrémités du réseau.

De manière générale, il semble que le contrôle de la vivacité encourage la *décentralisation*. D'autre part, le trafic de contrôle envoyé aux voisins est une charge de plus sur ces nœuds, et il ralentit les délais de réponse des nœuds. Le trafic de contrôle dépendant du nombre de voisins, les nœuds de faible degré sont plus réactifs. Notons que favoriser ainsi la réactivité va à l'encontre de certaines recommandations faites pour la tolérance aux pannes (voir chapitre 1.2.2) telle qu'assurer un fort degré pour limiter le risque de déconnexion d'un nœud.

1.2.9 Rapidité de traitement des requêtes

Dans un système pair-à-pair, un utilisateur qui cherche un objet souhaite obtenir une réponse rapidement. Nous avons vu qu'un système pair-à-pair ne garantissait souvent (dans les cas des réseaux physiques asynchrones) aucune borne sur le temps de traversée d'une arête. Cela signifie que, même dans un système créé pour assurer un temps de réponse faible, le temps mis par une requête pour obtenir une réponse ne peut pas être borné. Toutefois, une bonne approximation du temps mis par une requête pour obtenir une réponse est le nombre de sauts logiques nécessaires pour aller du nœud demandeur au nœud connaissant la réponse. C'est pour cela que le terme de «temps» mis par une requête est parfois utilisé pour des raisons de simplicité, même si l'on ne fait que se référer au nombre de sauts.

La manière de trouver un objet dans un réseau change selon que le système comporte une centralisation ou pas. La solution la plus efficace est d'emprunter le plus court chemin entre le demandeur et le nœud qui peut fournir l'objet recherché. Toutefois, cette méthode n'est pas toujours utilisable dans un système décentralisé où la structure globale du réseau n'est pas forcément connue, un plus grand nombre de sauts que le diamètre peut alors se révéler nécessaire. Le diamètre du réseau est donc une borne inférieure du nombre de sauts nécessaires à une requête pour trouver un objet dans un système.

Cependant, comme le relève [29], le temps (donc le nombre de sauts) mis par une requête pour trouver un objet doit être mis en rapport avec celui mis pour le récupérer. Dans le cas de certains objets, un gros fichier par exemple, la récupération sera beaucoup plus longue et l'on pourra donc négliger le temps mis pour trouver ce fichier par rapport au temps nécessaire à le récupérer. À l'inverse, dans le cas d'objets récupérables rapidement, comme un petit fichier, le temps mis à trouver l'objet est critique car il est souvent plus long que le temps mis à récupérer l'objet (comme dans le cas des DNS, où l'objet prend quelques octets). Notons que le transfert de l'objet vers le nœud respon-

sable est possible dans le cas de transferts rapides et peu coûteux, comme des fichiers de faibles tailles par exemple.

Dans un système pair-à-pair, les nœuds sont connectés dans un réseau logique *via* un réseau physique sous-jacent. Afin de diminuer le délai d'acheminement d'un message et donc accélérer les recherches d'objets, il existe donc deux possibilités :

- diminuer le nombre de sauts dans le réseau logique ;
- diminuer le nombre de sauts dans le réseau physique.

Rapprocher le réseau logique du réseau physique

Rappelons d'abord qu'un nœud est connecté à un voisin s'il connaît l'adresse physique de ce voisin. Pour l'envoi d'un message entre deux voisins dans le réseau logique, un système pair-à-pair passe alors par le réseau physique, sur lequel le système pair-à-pair n'a pas prise : il n'est en effet pas possible au réseau logique de modifier le réseau physique afin de diminuer les délais entre chacun des voisins dans le réseau logique. De plus, les réseaux pair-à-pair sont dynamiques, donc une modification du réseau physique dans le but d'accélérer le réseau logique à un instant donné pourrait devenir inutile l'instant d'après, à la suite de la déconnexion d'un nœud par exemple.

Toutefois, le problème peut être vu sous un autre angle : il est possible d'inspirer le réseau logique du réseau physique. Un système pair-à-pair dont le réseau logique est proche du réseau physique signifie que des nœuds voisins dans le réseau logique seront proches dans le réseau physique, assurant ainsi un délai plus court entre chaque saut dans le réseau logique. Il n'existe pas de méthode universelle pour estimer la mesure de *proximité* entre deux nœuds. Celle-ci peut être estimée à partir de la rapidité de réponse d'un message de contrôle (ping) entre ces deux nœuds [88, 105] ou encore la longueur du plus long préfixe commun de leur adresse IP [37] par exemple. Un délai plus court pour chaque saut du réseau logique permet de raccourcir le délai total d'un message envoyé dans le réseau logique. Le risque d'une telle proximité physique est alors qu'en cas de déconnexion d'un sous-réseau physique, des nœuds qui auraient dû être à distance raisonnable voient tous les chemins les reliant grandement rallongés, dans le cas où les plus courts chemins sont coupés, ou inutilisables. Dans ce dernier cas, le réseau logique peut être déconnecté en deux composantes distinctes. Il se peut aussi que les chemins connus entre deux nœuds cherchant à communiquer aient disparu, et que d'autres chemins existent sans être connus de ces deux nœuds, ce qui a le même effet. Si le réseau logique ne peut influer sur le réseau physique, il peut donc néanmoins tenter de connecter les nœuds du réseau logique selon leur proximité physique.

Allant dans le sens du rapprochement du réseau logique et du réseau physique, des travaux ont commencé à tenir compte de la topologie physique lorsqu'ils connectaient les nœuds entre eux. Les systèmes décentralisés structurés Tapestry [105] et Pastry [88] ont ouvert la voie en 2001, en choisissant pour chaque nœud les voisins les plus proches lorsque plusieurs choix existent.

TOPLUS [37] a poussé le travail d'optimisation commencé par Tapestry et Pastry en créant en 2003 un réseau décentralisé structuré organisé en fonction de la topologie physique. Il utilise pour cela la longueur du préfixe commun de deux adresses physiques IP

comme indication de mesure de proximité physique.

LAND [2, 3] a proposé la même année, un protocole qui permet d'assurer que, pour faire communiquer deux nœuds, le rapport entre le nombre de sauts nécessaires pour les relier dans le réseau logique et le nombre de sauts nécessaires pour les relier dans le réseau physique soit inférieur à $1 + \epsilon$, quelque soit $\epsilon > 0$. Toutefois, LAND assure ce rapport sous certaines conditions de répartition physique des nœuds, en particulier pour l'insertion d'un nœud.

Rapprocher le réseau logique des intérêts des utilisateurs

Ce rapprochement a du sens si l'on considère que les utilisateurs vont souvent demander des objets qui font partie de leurs centres d'intérêts. Cela signifie que, pour un nœud donné, certains nœuds sont plus susceptibles que d'autres de pouvoir fournir les objets qu'il recherche, du fait de leurs intérêts proches. Dans le cas de réseaux où les utilisateurs sont regroupés en communautés et effectuent essentiellement des requêtes sur des objets proches de leurs centres d'intérêts, la connexion à des voisins ayant les mêmes intérêts pourrait donc se révéler intéressante. Cette partie sera développée dans le chapitre 5.

Notons que dans le cas de systèmes permettant de rapprocher le réseau logique du réseau physique, certains systèmes se basent sur une mesure de proximité. Si une mesure de proximité d'intérêts peut être définie, alors on pourrait remplacer la mesure de proximité physique par cette mesure et rapprocher ainsi le réseau logique du réseau des intérêts des nœuds.

1.2.10 Expressivité des requêtes

Les systèmes pair-à-pair ont été créés sur le modèle de systèmes de recherche basés sur des serveurs. Ayant des utilisateurs habitués à la souplesse des systèmes centralisés, les systèmes décentralisés structurés se doivent d'offrir les mêmes fonctionnalités et la même souplesse, en particulier concernant les possibilités de recherche.

Différents modes de recherche existent et, de manière générale, un compromis est à faire : plus la recherche dans le système sera souple, c'est-à-dire plus elle permettra des recherches complexes, plus elle sera longue. De manière non-exhaustive, les différents types de recherche rencontrés dans les systèmes pair-à-pair sont :

- la *recherche par intervalle*. Cette recherche permet de rechercher tout objet dont le nom est dans un intervalle. Pour une donnée numérique par exemple, la requête `[1-20]` peut être effectuée pour trouver toutes les données comprises entre 1 et 20.
- La *recherche par expression rationnelle* permet une recherche fine, puisqu'elle permet des recherches d'expressions. Par exemple, une recherche peut être effectuée pour trouver toutes les chaînes composées de *4 lettres minuscules suivies d'un chiffre*, par l'expression `[a-z]{4}[0-9]`.
- La *recherche approximative*. Celle-ci permet les recherches proposées dans les interfaces de type unix/linux, comme par exemple `p?ngou*` si l'on recherche *pingouin*. Cette recherche permet de placer les caractères d'approximation * et ? partout

dans la chaîne, et plus seulement au début ou à la fin de la chaîne.

– La *recherche exacte* est la recherche la plus simple à effectuer. Elle est aussi appelée recherche par clé : pour cela, chaque objet est associé à une clé unique, par hachage par exemple. Cette méthode ne permet cependant de trouver l'objet que si l'on dispose du nom exact de l'objet ou de sa clé.
  Il est possible d'étendre la recherche exacte à un sous-ensemble de la recherche approximative : les recherches préfixe et suffixe. La requête `(pingou*)` peut alors être effectuée pour une trouver un objet de nom *pingou*`in` sans être sûr de la syntaxe de la fin de son nom. Afin de permettre des recherches sur des mots morphologiquement similaires [13], il suffit d'ôter quelques caractères à la fin de la chaîne. Une autre possibilité est d'enregistrer un objet sous plusieurs noms, en enlevant un ou plusieurs caractères à la fin du nom de cet objet. Cela permet accessoirement de considérer indifféremment les pluriels et les singuliers, les féminins et les masculins.

Suivant l'application, la recherche nécessaire peut être plus ou moins fine. Par exemple, pour les DNS, une recherche exacte suffit car le nom de domaine cherché est connu exactement. Pour d'autres types de systèmes où l'on ne sait pas exactement ce que l'on recherche, des requêtes plus élaborées sont nécessaires : par exemple la recherche par expression rationnelle ou la recherche approximative pour les systèmes de partages de fichiers et les bases de données (mot-clé *like* en SQL), ou la recherche par intervalle pour les bases de données (mot-clé *between... in* en SQL) et les systèmes de partages de ressources de calcul.

L'idéal est d'arriver à des recherches multicritères expressives, comme le permettent des langages de bases de données tel que SQL. Pour les recherches non exactes, il est possible d'affiner la recherche en s'aidant de méta-informations associées à un objet lors de la mise en partage par un nœud. Ces méta-informations permettent de préciser la nature et la sémantique de l'objet. Dans le cas de fichiers, ce peuvent être des caractéristiques comme la taille, la date de dernière modification, le type de fichier, etc. Dans le cas de ressources de calcul, ce sera le type de processeur, sa vitesse, la quantité de mémoire cache, de mémoire, d'espace disque, etc. Afin d'accélérer le traitement des requêtes sur les nœuds, l'extension HUGE de Gnutella2 propose l'utilisation d'un hachage pour rechercher rapidement un fichier parmi tous ceux partagés par un nœud. Le hachage rend alors l'utilisation de recherches fines (comme les recherches par expressions rationnelles) plus difficiles à utiliser. Remarquons qu'il se peux aussi que la hachage soit utilisé dans des protocoles fermés comme KaZaA ou dans les serveurs eDonkey/eMule.

1.2.11 Exhaustivité des réponses

Un utilisateur peut souhaiter obtenir au moins une réponse, plusieurs réponses, un nombre minimum ou donné de réponses, ou toutes les réponses disponibles correspondant à la requête. La méthode de recherche et le système ne seront évidemment pas les mêmes selon l'exigence de l'utilisateur. Ainsi, l'obtention de toutes les réponses du réseau peut se faire très simplement au moyen d'un système de publication dans un système centralisé ou un système décentralisé structuré. Dans un système décentralisé

ou hybride, il nécessitera l'envoi de la requête à un nombre linéaire de nœuds en fonction du nombre de nœuds du réseau. Un système de recherche exhaustif permettra d'assurer de toujours trouver au moins une réponse si elle existe.

Afin de garantir que chaque nœud puisse trouver n'importe quel objet, il faut assurer qu'il puisse envoyer un message à n'importe quel autre nœud du réseau. Le protocole de connexion doit donc assurer que le graphe d'interconnexion soit *fortement connexe*.

D'autre part, les nœuds d'un système pair-à-pair se connectent et se déconnectent régulièrement. Dans un système d'archivage, ou lorsque les nœuds sont régulièrement reconnectées, un utilisateur peut désirer obtenir seulement des objets accessibles (donc en ligne à cet instant), ou il peut accepter aussi des objets de nœuds déconnectés (espérant que ces nœuds se reconnecteront plus tard), comme [30].

Il peut être important qu'une recherche soit la plus complète possible, afin que plusieurs réponses différentes soient obtenues si elles existent. C'est en particulier utile :

- si le format des objets a une importance ;
- si la qualité de l'objet peut varier et que cette qualité a une importance sur l'utilisation que l'on peut en faire.

Pour un mélomane, un fichier sonore compressé avec une grande perte ne sera pas satisfaisant.

Dans d'autres type de systèmes, une recherche incomplète suffit. C'est ainsi le cas pour des objets dont seul le fond a une importance, et non la forme. C'est le cas des textes qui peuvent être indifféremment en texte pur ou mis en forme (pdf, tex, etc.) si seul le contenu importe au lecteur. Cet aspect se retrouve aussi dans les systèmes dédiés au partage d'objets d'un format et une qualité fixés, comme Napster qui ne permettait que le partage de fichiers au format mp3.

Dans tous les cas, selon que les réponses sont situées sur le même nœud ou pas, et selon la méthode d'envoi des réponses, l'utilisateur peut recevoir ces réponses en même temps ou les unes après les autres. Des plus, selon les méthodes, l'utilisateur peut savoir ou pas quand toutes les réponses qu'il va recevoir lui sont parvenues.

1.2.12 Authentification des objets

Nous avons vu dans le chapitre 1.2.2 que le comportements des nœuds byzantins n'était pour l'instant pas géré pas les systèmes pair-à-pair. C'est pourquoi, dans les systèmes qui doivent garantir la correction des objets partagés, l'authentification est une fonctionnalité indispensable puisque des nœuds byzantins peuvent sinon tenter de saboter le comportement du système. En effet, si aucune authentification n'est permise, il est possible d'introduire des objets trompeurs : un nom erroné peut être donné à un objet, ou un objet inutilisable peut être partagé (fichier illisible, processeur inaccessible ou effectuant des calculs erronés).

Garantir qu'un objet est correct peut avoir trois significations.

- L'objet n'a pas été modifié depuis sa mise en ligne. Par exemple, si un auteur de roman diffuse un essai dans un système de partage de fichiers, il est nécessaire de pouvoir vérifier que l'œuvre récupérée est bien l'originale.

- L'objet correspond aux informations qui lui sont associées, par exemple son nom. Ainsi, si un roman a pour titre « Voyage au centre de la Terre », l'utilisateur voudra s'assurer avant de récupérer ce texte qu'il s'agit bien de ce roman écrit par Jules Verne, et non d'un autre contenu sans rapport. Cette possibilité est d'autant plus utile que le coût de récupération est important (temps, argent, bande passante, etc.). Dans le cas d'une ressource de calcul, il peut être nécessaire de s'assurer que les informations relatives à ses performances sont exactes, ou s'assurer qu'elle n'a pas la réputation d'envoyer de faux résultats.
- Dans le cas où différentes versions d'un objet peuvent coexister, l'avis d'autres utilisateurs peut être le bienvenu avant de récupérer l'objet. Ainsi, dans le cas d'un extrait sonore ou d'une bande vidéo, il est préférable de s'assurer de la qualité de cet extrait.

Le risque de diffusion de faux a pris forme dès les débuts des systèmes pair-à-pair. Toutefois, il s'est concrétisé avec le début de l'opposition d'organismes aux systèmes de partage de fichiers permettant la distribution de fichiers sous droits d'auteurs [79, 80, 85]. Des faux, c'est-à-dire des fichiers dont le contenu ne correspondait pas au nom, sont apparus en masse dans les systèmes de partage de fichiers grand public comme Gnutella, KaZaA, et eMule. L'insertion de faux a rendu le fonctionnement de ces systèmes problématique puisque les réponses renvoyées par le système étaient fausses pour beaucoup. Il est donc nécessaire de limiter ce genre de dysfonctionnement, voire de pouvoir les interdire dans des systèmes pair-à-pair dont la justesse des objets est essentielle.

Certains logiciels client-serveur pair-à-pair hybrides et semi-centralisés comme eMule ou eDonkey [26, 61] ont réagi en ajoutant pour chaque nœud une évaluation des objets qu'il récupère (nommé *ghost rating*). Une autre possibilité d'estimer l'authenticité de fichiers serait d'identifier les nœuds qui les proposent. En effet, il n'est pas déraisonnable de considérer que les différents objets erronés ou incorrects sont insérés par les mêmes nœuds. Les systèmes utilisant des serveurs regroupant des communautés d'utilisateurs (voir chapitre 1.2.5) se connaissant, et accédant au service par identification, ont plus facilement résisté à ces insertions de faux fichiers et, les organismes insérant des faux se sont rapidement vus identifiés et exclus. Toutefois, si ce type d'authentification à l'insertion fonctionne dans des systèmes centralisés, elle est bien plus difficile à mettre en place dans les réseaux décentralisés.

Toujours en identifiant les nœuds, il est possible de chercher à savoir quelle est la crédibilité d'un nœud. Dans un système centralisé ou semi-décentralisé, il est simple de confier la récolte des critiques sur un serveur, afin de les rendre disponibles aux nœuds. Il est toutefois possible à un agresseur disposant de nombreux nœuds de fausser ce système d'évaluation.

1.2.13 Accessibilité des objets et «consommateurs égoïstes»

Afin de satisfaire les nœuds du réseau, un système pair-à-pair doit permettre la recherche comme la récupération d'objets. Toutefois, un grand nombre d'utilisateurs recherchant un trop petit nombre d'objets a rapidement rendu impossible la récupération de tous les objets demandés par tous les nœuds. En effet, trop de sollicitations étaient

envoyées à quelques nœuds partageant ces fichiers populaires. Ces derniers ne pouvaient répondre à toutes les demandes à la fois. Cela a incité des logiciels client-serveur comme LimeWire et Bearshare à proposer des modifications à la première version de Gnutella pour faciliter l'accès aux objets. Ces deux logiciels proposèrent de placer les fichiers à récupérer dans des files d'attente du côté du nœud client, et les nœud clients dans des files d'attente du côté du nœud serveur. Cela permit de télécharger plus tard des fichiers alors impossible à récupérer (par exemple lorsque l'hébergeur qui proposait la donnée envoyait déjà des données et ne pouvait en envoyer plus).

Par ailleurs, un système pair-à-pair est d'autant plus utile qu'il met d'objets à disposition des utilisateurs, et qu'il permet au plus grand nombre d'y accéder. Pourtant, depuis le lancement des systèmes pair-à-pair, il a été observé que bien des utilisateurs refusent de partager des données dans les systèmes pair-à-pair, jusqu'à 69% des utilisateurs de Gnutella en 2000 d'après [5] : nous nommerons ces utilisateurs «consommateurs égoïstes», ou nœuds égoïstes selon le cas. Ces résultats sont confirmés par d'autres travaux : le protocole eDonkey semble comporter 68% de consommateurs égoïstes en 2003 [29]. Les raisons pour qu'un nœud ne partage rien peuvent être, par exemple, de ne pas vouloir consommer de la bande passante montante, la peur de se faire prendre à distribuer une donnée sans en avoir le droit, etc. Certains utilisateurs utilisent aussi des passerelles (pages HTTP) sur la toile pour effectuer dans Gnutella des recherches et des téléchargements. Ils passent donc par des nœuds qui ne partagent rien : des pages HTTP. Il peut s'agir aussi de personnes qui ne peuvent utiliser un logiciel client-serveur à cause d'un pare-feu.

Le comportement égoïste des nœuds a tendance à gêner la *tolérance aux pannes* et à limiter l'intérêt des systèmes pair-à-pair en augmentant le nombre de consommateurs d'objets sans augmenter le nombre d'objets à disposition des nœuds du réseau. Ainsi, [5] montre que seuls 1% des utilisateurs répondent à 37% des requêtes, et que 5% des utilisateurs répondent à 70% des requêtes. Une étude [41] relate que le nombre de nœuds partageant des données dans Gnutella serait descendu de 40% à 15% après la rupture du réseau, début août 2000, au moment du procès contre Napster. Ce nombre serait cependant remonté à partir d'octobre suivant. Si les utilisateurs qui hébergent un grand nombre d'objets sont la cible d'attaques, le système pair-à-pair perdra une grande partie de ses objets (les données hébergées par les nœuds disparus peuvent toutefois exister en d'autres endroits), ce qui le rendra moins intéressant pour ses utilisateurs.

Afin d'encourager le partage d'objets, et limiter le nombre de recherches par des nœuds qui n'apportaient rien au réseau, il a été proposé d'imposer aux nœuds de partager des objets. Cependant, pour les systèmes décentralisés, la plupart des logiciels client-serveur et des protocoles étant des initiatives de la communauté libre. Il était donc facile, après rétro-ingénierie si nécessaire, de modifier les logiciels client-serveurs pair-à-pair. Il fut ensuite proposé de bloquer les nœuds égoïstes sur la base des informations transmises par les nœuds. Là encore, des logiciels client-serveur peuvent être modifiés ou créés afin de ne pas diffuser d'informations qui les pénaliserait voire les excluraient pour non-partage de fichier, mauvais débit, etc. Notons que la solution naïve consistant à vérifier la présence réelle des données annoncées par chaque nœud n'est pas réaliste. Elle engendrerait en effet une charge trop importante sur le réseau et les nœuds

vérificateurs, et une diminution de la bande passante utilisable chez ces derniers.

LimeWire et Bearshare ont enfin proposé un mécanisme empêchant la diffusion de requêtes provenant de navigateurs sur le réseau, effectuant leurs recherches par le biais de site HTTP, en identifiant ces requêtes. Cela limite le trafic purement égoïste, puisqu'il provient de site HTTP qui ne mettent aucun objet à disposition du système, donc ne lui apportent rien.

Favoriser le partage d'objets dans les systèmes pair-à-pair n'est pas directement liée au routage et ne sera donc pas traitée dans la thèse. Toutefois, si la solution coercitive n'est pas applicable pour obliger le partage de données dans ces systèmes, des techniques ont été développées pour encourager la mise à disposition d'objets. DirectConnect [21] permet d'obliger le partage d'un certain nombre de fichiers. eMule (chapitre 4.2.1 de [61]) permet à chaque nœud d'attribuer à tout autre nœud une note selon le nombre de fichiers qu'il lui a envoyé. BitTorrent [18], un protocole pair-à-pair de transfert de fichiers, utilise aussi une technique permettant de favoriser la diffusion des objets. Dans sa première version, il commence par mettre en relation, comme eMule (chapitre 4.3.1 de [61]), les sources et les demandeurs. Ensuite, un nœud favorise en majorité l'envoi de données au nœud qui lui envoie des données au meilleur débit. Chaque nœud permet tout de même à quelques nœuds choisis aléatoirement, donc ayant potentiellement un faible débit, de télécharger chez lui. BitTorrent favorise enfin la diffusion des données : les nœuds envoient en priorité une donnée à un nœud ayant un bon débit. Cela permet une augmentation rapide du nombre de sources d'une donnée dans le système. Pour finir, signalons qu'il existe des proposition pour encourager le partage basées sur la théorie des jeux, comme [6].

1.2.14 Réplication des objets

Comme nous l'avons vu dans le chapitre 1.2.13, dans les systèmes dédiés à l'entrepôt permanent de données ou de ressources, il ne faut pas qu'un nœud soit le seul fournisseur pour une fraction importante d'objets, ou pour un objet très populaire. En effet, ce nœud deviendrait un point faible pour le réseau, car une grande part des objets ou de l'intérêt du système reposerait sur lui. Dans un système où les objets doivent rester accessibles en cas de départ du nœud qui en était responsable ou encore de déconnexions de certains voisins, plusieurs solutions existent.

- La mise en cache est une copie de manière transparente pour le nœud qui partage l'objet.
- La réplication, qui consiste à copier un objet sur un certain nombre de nœuds, automatiquement ou selon sa popularité. Le nombre de copies peut aussi varier suivant la probabilité qu'un nœud disparaisse du réseau.
- L'utilisation d'algorithme auto-stabilisant pour assurer qu'un chemin pourra être trouvé à terme vers tout objet. Nous ne nous appesantirons pas sur cette méthode.

La récupération normale d'objets par des nœuds constitue en elle-même une mise en cache (ou réplication passive) si cette copie est mise à disposition du réseau une fois l'objet récupéré. Certains logiciels client-serveur de systèmes pair-à-pair, comme le logiciel eDonkey [26], partagent par défaut les objets récupérés.

La réplication a été utilisée pour augmenter la disponibilité et l'accessibilité des objets dans des systèmes n'utilisant pas de méthode de recherche exhaustive, comme Gnutella [39]. Lorsque trop peu de copies restent dans le réseau, l'objet est copié à nouveau. Pour cela, un mécanisme de réplication actif est nécessaire. Le contrôle et la réplication des objets ayant un coût parfois non négligeable, ce mécanisme s'assure que les objets ne seront pas répliqués plus que nécessaire afin de ne pas surcharger inutilement les nœuds et les liens. Il est aussi possible d'estimer la probabilité que tous les nœuds hébergeant une copie d'un objet soient déconnectés au bout d'une certaine période. Pour cela, une estimation de la demi-vie des nœuds dans le réseau peut être utilisée.

La mise en cache et la réplication permettent également d'augmenter le nombre de sources d'un objet, et peuvent donc diminuer le délai nécessaire pour en trouver une copie si les copies sont judicieusement placées dans le réseau. La réplication permet aussi d'augmenter la vitesse de récupération d'un objet en multipliant le nombre de sources potentielles. Il est en effet possible de récupérer en parallèle des parties différentes d'un même fichier, ou de lancer un calcul parallélisé sur plusieurs nœuds. Pour les systèmes de partage de fichiers, il faut noter que la réplication n'accélère la récupération que si la somme des débits entrants des nœuds demandeurs est supérieure ou égale à la somme des débits sortants des nœuds sources. Or la plupart des utilisateurs actuellement ont un débit entrant limité, en raison de la bande passante asymétrique de l'ADSL. La réplication rend donc aussi le réseau *plus rapide*. Cette dernière amélioration du temps de récupération ne sera cependant pas directement discutée dans cette thèse, car elle concerne moins l'interconnexion et la recherche d'objets que le transfert de ces objets. Cette technique est utilisée dans eDonkey [26], eMule [61] et BitTorrent [18]. Toutefois, la première version de BitTorrent repose sur des serveurs pour trouver une première source, le nœud hébergeant cette source redirige ensuite le nouveau nœud vers d'autres copies dans le réseau. C'est pourquoi des protocoles comme BitTorrent, très performants en terme de récupération d'objets, gagneraient à être couplés à un protocole de recherche pair-à-pair décentralisé efficace comme des réseaux à contenu adressable. Cependant, cette centralisation a posé des problèmes de censure puisque de nombreux serveurs français hébergeant des liens vers des sources ont été contraints de fermer en 2005. Le lecteur intéressé pourra consulter la littérature concernant le protocole de téléchargement BitTorrent [18].

1.2.15 Pare-feux et détection d'utilisation de systèmes pair-à-pair

Un grand nombre d'organismes donnant accès à Internet utilisent des pare-feux pour gérer le trafic de requêtes et se protéger de certains messages indésirés. Les règles sur lesquelles sont basées ces pare-feux laissent souvent passer les requêtes HTTP. C'est pourquoi certains systèmes pair-à-pair ont opté pour l'utilisation de ce protocole pour le transfert de fichiers lorsqu'un nœud derrière un pare-feu veut récupérer une donnée de l'extérieur (comme Gnutella et FastTrack). De même, un nœud qui utilise un service de traduction d'adresse (NAT) ou un proxy a des difficultés à utiliser un système pair-à-pair. En effet, ces services servent souvent de passerelle entre un réseau privé et Internet,

et ne laissent pas passer tous les messages pour des raisons de sécurité. Ce sont des nœuds filtrés ou « protégés ».

Pour résoudre le problème posé par un tel nœud lorsqu'un nœud non connecté souhaite échanger avec un nœud protégé, il faut trouver un intermédiaire connecté au nœud protégé. La demande de connexion est alors dite *poussée* (de l'anglais *pushed*). Cet intermédiaire va demander au nœud protégé (via sa connexion) de se connecter au nœud qui désire échanger avec lui. Toutefois, cette méthode est coûteuse en ressources nécessaire (en nombre de messages et en temps de calcul). Certains protocoles comme eMule l'ont donc désactivée par défaut (chapitre 2.4 de [61]). Ainsi, dans Gnutella (voir chapitre *Descriptor Routing* de [45]), une réponse à une recherche est renvoyée en suivant le même chemin que la demande. Si un nœud reçoit une réponse mais ne peut récupérer le fichier en se connectant directement au nœud source, c'est que le nœud source est donc probablement « protégé ». Le nœud demandeur va donc renvoyer, le long du chemin qu'a pris la réponse, une demande de téléchargement au nœud source, qui va se connecter au nœud demandeur. S'il n'y arrive pas, c'est que le nœud demandeur est « protégé », l'échange est alors impossible par ce moyen. Cette méthode demande un nombre d'intermédiaires important. À la fin de l'année 2005, eDonkey2000 annonçait avoir trouvé une méthode pour faire communiquer deux nœuds utilisant une traduction d'adresse (NAT) [26]. Des protocoles comme STUN [98] peuvent être utilisés pour cela.

Enfin, la plupart des systèmes permettent l'utilisation de n'importe quel port réseau afin de permettre plus de flexibilité et ainsi pouvoir changer de port si le port par défaut est interdit par un pare-feu. Cela rend difficile le contrôle, voire l'interdiction, d'un système pair-à-pair par des organismes gérant un sous-réseau physique. Toutefois, lorsqu'un organisme souhaite détecter l'utilisation de systèmes pair-à-pair par une machine interne de son réseau, il est possible de mettre en relation le grand nombre de connexions réseaux ouvertes (TCP), le nombre de paquets entrés et sortis, et le débit entrant et sortant.

1.3 Conclusion

Nous avons vu dans ce chapitre différents objectifs et les difficultés qu'ils présentent pour les systèmes pair-à-pair, selon qu'ils sont centralisés ou pas. Certains ont été unanimement visés par les systèmes pair-à-pair, comme :

- la gestion de nombreux utilisateurs (vu au chapitre 1.2.1) ;
- l'accès au réseau (vu au chapitre 1.2.5) ;
- le dynamisme des nœuds (vu au chapitre 1.2.7) ;
- la rapidité de traitement des requêtes (vu au chapitre 1.2.9).

D'autres objectifs ont été traités de manière incomplète ou ont fait l'objet d'attention de la part d'une partie seulement des protocoles développés. Parmi ceux-ci, on trouve :

- la tolérance aux pannes (vu au chapitre 1.2.2) ;
- l'équilibrage de la charge (vu au chapitre 1.2.3) ;
- la protection des utilisateurs (vu au chapitre 1.2.6) ;
- l'expressivité des requêtes (vu au chapitre 1.2.10) ;

- l'exhaustivité des réponses (vu au chapitre 1.2.11) ;
- l'authentification des objets (vu au chapitre 1.2.12) ;
- l'accessibilité des objets et les «consommateurs égoïstes» (vu au chapitre 1.2.13) ;
- la réplication des objets (vu au chapitre 1.2.14) ;
- les pare-feux et la détection d'utilisation de systèmes pair-à-pair (vu au chapitre 1.2.15).

Enfin, l'autonomie et l'envoi à des sous-réseaux 1.2.4 se place dans un cadre d'un échange d'objets secrets et dans un réseau moins sûr. Elle n'a jusque là pas été considérée comme suffisamment intéressante pour motiver le développement de systèmes pair-à-pair spécifiques. Parmi les différentes problématiques que nous avons vu, nous avons soulevé qu'il était difficile de faire cohabiter certaines approches, comme la tolérance aux pannes et la réactivité des nœuds par exemple, ou encore la rapidité de traitement des requêtes face à la gestion de nombreux utilisateurs. Nous allons voir dans la suite de cette thèse comment les systèmes pair-à-pair ont abordé ces différentes problématiques, lesquelles ont pu être combinées, et à quel prix.

Chapitre 2

Un survol des systèmes pair-à-pair existants

Dans ce chapitre, nous allons présenter les grands types de systèmes pair-à-pair et nous préciserons en quoi ils diffèrent entre eux. Pour chacun de ces types, nous présenterons des exemples de systèmes existants et nous verrons comment ils abordent les problématiques énoncées au chapitre précédent.

Les systèmes pair-à-pair peuvent être classés en 5 grandes familles, que nous verrons dans un ordre décroissant de centralisation :

- Les *systèmes centralisés* sont les premiers systèmes pair-à-pair. Nés en juin 1999, ils sont composés d'un serveur indexant les objets disponibles dans le réseau, et de nœuds qui peuvent demander et proposer ces objets par le biais de ce serveur. Seule la récupération d'objets est décentralisée ; la recherche reste, elle, basée sur un serveur.
- Les *systèmes semi-décentralisés* arrivent en juin 2000. Ils utilisent plusieurs serveurs stables inter-connectés. Ces serveurs permettent de répartir la responsabilité des nœuds du système sur plusieurs machines. Les serveurs sont stables et ils n'ont pour rôle que la coordination des recherches. Ils ne partagent aucun objet. Chaque nœud du système est connecté à un serveur qui, contrairement aux systèmes centralisés, n'a qu'une connaissance partielle des objets du système. Un serveur ne connaît en effet que les objets partagés par les nœuds connectés à lui, et non plus tous les objets présents dans le système.
- Les *systèmes hybrides* ont vu le jour en mars 2001. Basés sur des nœuds partageant tous des objets, ils font toutefois la différence entre plusieurs niveaux de responsabilité. Des nœuds sont connectés entre eux afin de constituer l'infrastructure de base du réseau, formant ainsi un premier niveau logique. D'autres nœuds se connectent à un nœud de premier niveau et forment le second niveau de responsabilité. Les nœuds peuvent être promus du second niveau au premier niveau et *vice versa* d'une manière que nous décrirons plus loin. Le premier niveau du système joue un rôle prépondérant dans le système, car il permet la mise en relation des nœuds fournisseurs avec les nœuds demandeurs.

- Les *systèmes décentralisés non-structurés*, créés en mars 2000, ne reposent plus sur des serveurs ayant un rôle prépondérant mais sur des nœuds jouant tous le même rôle. Ces nœuds sont connectés entre eux sans organisation particulière.
- Les *systèmes décentralisés structurés*, dont les premiers travaux sont publiés en septembre 2001. Sans aucun serveur central, les nœuds sont connectés entre eux suivant certaines contraintes liées à leur identifiant, afin de maintenir une architecture spécifique. C'est cette dernière qui permet d'assurer de bonnes propriétés sur les temps de publication et de recherche.

Nous détaillons ci-dessous ces différents systèmes pair-à-pair dans le même ordre décroissant de décentralisation. Cet ordre n'est donc pas l'ordre chronologique d'apparition de ces systèmes. Dans la suite, nous verrons pour chaque type de système les raisons qui ont poussé à sa création.

2.1 Les systèmes centralisés ou les débuts du pair-à-pair

Les systèmes centralisés sont les premiers systèmes pair-à-pair à avoir vu le jour, en juin 1999, lorsque fût créé Napster [73]. Il fut suivi d'autres protocoles du même type, comme DirectConnect (qui a évolué en DC++ [21]).

2.1.1 Les choix de fonctionnement de Napster

Napster est un système de partage de fichiers, où les objets partagés, des fichiers, sont gérés par un serveur index. Il s'agit en fait de 50 à 150 serveurs index, un méta-serveur réorientant les nouveaux nœuds vers un de ces serveurs [89]. Un serveur contient les titres de fichiers hébergés par le réseau, et l'adresse des nœuds qui les mettent à disposition. Les serveurs peuvent aussi jouer le rôle de censeurs : par exemple, dans Napster, seul était autorisé le partage de fichiers audio de format mp3. Napster n'a pas été officiellement diffusé, et les analyses du protocole [74] sont donc basées sur la rétro-ingénierie. Ces analyses ont permis la création de logiciels client-serveur libres, comme Lopster, Xnap ou Teknap, mais aussi des logiciels client-serveur non libres comme WinMX. Des serveurs libres ont aussi vu le jour comme OpenNap [76] et OpenNap-ng [77]. Ces derniers serveurs permettent de choisir les fichiers partagés, ôtant ainsi la limitation aux fichiers mp3.

Dans Napster, lorsqu'un nœud cherche un objet, il envoie au serveur une requête sous forme de chaîne de caractères, pour trouver tous les fichiers dont le titre contient cette chaîne. Le serveur répond alors par une liste de nœuds hébergeant un fichier correspondant à cette recherche. La recherche peut être limitée au serveur auquel est connecté le nœud demandeur ou être lancée sur tous les serveurs. L'échange du fichier s'effectue ensuite directement entre le pair qui héberge le fichier et le pair qui le recherche. L'étape du transfert du fichier est donc la seule différence avec le modèle *client-serveur*. L'architecture de ce type de système permet ainsi de partager les mêmes fichiers sur des nœuds différents.

2.1.2 Comportement des systèmes centralisés

Au delà de Napster, nous allons décrire ici les forces et les faiblesses des systèmes pair-à-pair centralisés par rapport à chaque problématique pair-à-pair vue au chapitre 1.2.

Gérer de nombreux utilisateurs

La charge des nœuds du système est concentrée sur les quelques serveurs, qui sont responsables de maintenir à jour les listes d'objets partagés. La charge des liens est donc concentrée sur les liens attenants aux serveurs. Dans le réseau physique, les liens physiques composant les liens logiques supportent donc cette charge. Par ailleurs, il est impossible à un tel système d'accepter des utilisateurs au-delà d'un certain seuil, qui dépend des capacités physiques du serveur. Ces limites ne sont pas théoriques, comme cela a été observé dans le chapitre 1.7 de [61]. Ces types de systèmes sont en particulier sensibles à des attaques de type «déni de service».

Tolérance aux pannes et charge du réseau

Les systèmes centralisés sont intrinsèquement sensibles aux pannes, en particulier aux attaques ciblées. En effet, bien que plusieurs serveurs existent pour assurer une redondance de l'index des objets, leur nombre reste faible au regard du nombre de nœuds. Une panne sur les serveurs, dès la panne d'arrêt (voir le chapitre 1.2.2), a donc une forte incidence sur le fonctionnement du système puisque tous les nœuds jusqu'alors connectés au serveur sont expulsés du réseau. Ceux-ci se reconnectent alors au méta-serveur au même instant, risquant de le surcharger momentanément, puis de surcharger les serveurs restants. Cela peut aller jusqu'à rendre le système inutilisable. C'est cette centralisation qui a fourni un point d'attaque contre Napster : il a suffit de viser les quelques serveurs (leur propriétaire en fait) pour causer la fin de ce système, grâce à des attaques juridiques.

Autonomie et envoi à des sous-réseaux

Dans les réseaux centralisés, cette partie de la problématique est triviale. En effet, si un nœud effectue une recherche sur le serveur, il obtiendra entre autre une liste de nœuds proposant l'objet. Si l'un des nœuds renvoyés en réponse est considéré comme non sûr, il suffit de ne pas faire appel à lui pour accéder à l'objet. Dans le cas où un nœud demandeur souhaite éviter l'utilisation de certains nœuds du réseau, ou au contraire n'utiliser que certains nœuds qu'il connaît, l'utilisation d'un serveur (en lequel le nœud a confiance) permet de transmettre directement la requête aux nœuds jugés dignes de confiance. Cependant, cette possibilité n'a pas été proposée par Napster.

Accès au réseau

Puisque les systèmes pair-à-pair centralisés utilisent des serveurs stables, ils ont une adresse physique stable ou bénéficient des services d'un DNS permettant de la

retrouver. Le problème du point d'entrée ne les concerne pas, car chaque nouveau nœud peut facilement retrouver l'adresse d'un serveur pour s'insérer.

Protection des utilisateurs

Dans les systèmes centralisés, toutes les requêtes passant par le serveur, il est aisé de contrôler et/ou censurer les objets mis en commun par les nœuds. Ainsi, le contrôle des fichiers partagés par Napster a été accentué durant ses difficultés judiciaires : un filtre fut mis en place afin que le réseau n'accepte plus le partage de fichiers soumis à des droits d'auteurs. Ces filtres, bien que rudimentaires, illustrent la difficulté des systèmes centralisés à résister à la censure. La protection contre la censure et la défense de l'anonymat des utilisateurs ne peuvent être assurées correctement par des systèmes centralisés puisqu'il suffit de contrôler le serveur pour censurer les nœuds ou les identifier.

Dynamisme des nœuds

Les nœuds peuvent arriver et repartir de manière simple dans les systèmes centralisés car aucune topologie n'y est maintenue. Les nouveaux nœuds, à leur arrivée, n'ont donc qu'à se connecter au serveur, stable et connu, pour être insérés dans le système.

Réactivité des nœuds

Le degré moyen d'un serveur centralisé est grand. Le serveur doit utiliser ses capacités pour traiter les requêtes de recherche, de partage d'objets, et d'effacement d'objets qu'il reçoit. Ces requêtes viennent s'ajouter aux messages nécessaires à la gestion des arrivées et des départs des nœuds. Il est donc difficile d'assurer des réponses rapides lorsque le nombre de nœuds approche la capacité maximale du serveur. Il est alors nécessaire d'augmenter les capacités du serveur, jusqu'à une certaine mesure et à un certain prix.

Rapidité de traitement des requêtes

Un système centralisé permet d'assurer un faible nombre de messages et de sauts pour chaque requête. En effet, une requête va directement vers le serveur, en un saut, et ce serveur répond en un saut. Un aller-retour vers le serveur est ainsi suffisant pour localiser un objet.

Expressivité des requêtes

Les systèmes centralisés peuvent traiter des recherches complexes, puisque tous les objets partagés par le système sont listés de manière centralisée. Le serveur, disposant de la liste exhaustive de tous les objets, peut effectuer de manière centralisée et rapide tout type de recherche, comme des recherches approchées, par intervalles, par expressions rationnelles, ainsi que, évidemment, des recherches par mots-clés.

Exhaustivité des réponses

L'architecture centralisée permet de fournir des résultats exhaustifs en réponse aux requêtes car tous les objets partagés dans le système sont enregistrés sur le serveur. Si un nœud choisissait de n'effectuer une recherche que parmi les objets d'un sous-ensemble de nœuds, le coût de cette recherche serait donc sensiblement le même que parmi toutes les objets du système. En effet, la requête serait dans les deux cas envoyée au serveur. La différence se ferait alors dans la recherche locale effectuée par le serveur. En cas de recherche sur tous les serveurs, les résultats arrivent groupés par serveur d'origine. D'autres initiatives comme FFSS [30] ont proposé des fonctionnalités telles que l'enregistrement des objets hors-ligne, dans le cadre de réseaux dont les nœuds se reconnectaient régulièrement.

Authentification des objets

L'existence d'un serveur facilite grandement la mise en place d'une identification des objets, par hachage du contenu pour les fichiers par exemple. Il est aussi possible que chaque nœud évalue les objets en associant une note à chaque objet qu'il a récupéré. Une note globale peut être ainsi donnée à des objets rigoureusement identiques. Dans ce cas, tout nœud peut déduire la qualité d'un nœud du système selon la qualité des objets proposés par ce nœud. Il est aussi possible de juger si un objet est intéressant selon les notes globales qui lui sont attribuées. Pour un système centralisé, il est simple de gérer globalement ces notes pour qu'elles soient lisibles et compréhensibles par chaque utilisateur, tout en étant non falsifiables. Un système de notation des nœuds eux-mêmes peut aussi être mis en place, afin de rendre accessible à chaque nœud le succès de chacun (nombre d'utilisations de ses objets) ou la satisfaction que chacun donne (attribuée après utilisation de l'objet par un nœud).

Accessibilité des objets et «consommateurs égoïstes»

Un serveur peut aisément constater quels nœuds ne mettent à disposition aucun objet, et donc mettre en place une politique adaptée afin d'éviter les comportements de «consommateurs égoïstes». Puisque toutes les requêtes passent par lui, il suffit au serveur d'interdire toute requête émise par un nœud qui n'a annoncé aucun objet (cependant, ce type de gestion n'a pas été mis en place par Napster), mais cela ne réglerait pas le problème posé par un utilisateur qui annoncerait des objets fictifs. En effet, le coût d'une vérification systématique des objets proposés par les nœuds serait trop important. Une solution simple serait de gérer localement sur le serveur (ou sur chaque nœud, mais ce serait moins exhaustif) une liste de nœuds considérés comme honnêtes, c'est-à-dire annonçant des objets effectivement accessibles. Notons que la mise en place d'une vérification par le serveur qu'un nœud partage bien des objets pourrait être effectuée sur dénonciation des nœuds du réseau. Elle devrait toutefois faire l'objet de précautions : il faudrait en effet éviter qu'il soit possible de surcharger un nœud en demandant continuellement la vérification de ses objets au serveur.

Réplication des objets

Dans les systèmes centralisés, il est aisé de maintenir pour chaque objet une liste de tous les utilisateurs qui proposent cet objet. Dès qu'une recherche est effectuée par un nœud, il trouve alors tout de suite les copies de l'objet correspondant à sa requête.

Pare-feux et détection d'utilisation de systèmes pair-à-pair

Puisqu'il n'y a qu'un faible nombre de serveurs, un administrateur de sous-réseau relié à Internet peut placer les adresses physiques de ces quelques serveurs dans une liste de machines dont l'accès sera interdit par le pare-feu du sous-réseau. Cela permet d'interdire à toute machine du sous-réseau l'accès au système pair-à-pair par les machines de son domaine, puisque le ou les serveurs sont des intermédiaires obligatoires pour utiliser le système pair-à-pair.

Toutefois, le problème est différent si un nœud u est connecté à travers un pare-feu. Si u souhaite se connecter au serveur du système pair-à-pair, la connexion peut être acceptée par le pare-feu. Le nœud u et le serveur peuvent alors communiquer malgré la présence du pare-feu. Le pare-feu interdira toutefois toute communication vers le nœud u qui sera initiée par un autre machine que le serveur. Ainsi, si un nœud v du système pair-à-pair souhaite avoir accès à un objet proposé par le nœud u, il lui est seulement possible de passer par le serveur qui relaiera au nœud u la demande d'accès à l'objet. Le nœud u se connectera alors au nœud v pour lui envoyer le fichier, lui donner accès aux ressources de calcul, etc.

2.2 Les systèmes semi-décentralisés : vers une dissémination des serveurs

À la suite des systèmes centralisés (vus au chapitre 2.1) et un peu avant les systèmes hybrides (que nous verrons au chapitre 2.3) sont apparus des systèmes créés pour rendre les recherches d'objets plus efficaces. En effet, si les systèmes hybrides permettent de décentraliser la recherche et la récupération d'objets, ils peinent à rendre des résultats lorsque le nombre d'utilisateurs augmente. En effet, dans ces systèmes, la méthode de recherche est basée sur une inondation bornée. Au fur et à mesure que le nombre de nœuds du réseau augmentent, ainsi que le nombre d'objets dans le réseau, l'inondation bornée touche proportionnellement de moins en moins de nœuds du réseau et donc de moins en moins d'objets du réseau. Cela diminue la probabilité de trouver une réponse. La solution consistant à augmenter la borne de l'inondation n'est pas satisfaisante car elle augmente la charge du réseau de manière exponentielle. Or la philosophie d'un système pair-à-pair repose *a priori* sur une augmentation des performances et de l'efficacité au fur et à mesure que le nombre de nœuds participants, donc que le nombre d'objets, augmente. C'est pourquoi des systèmes basés sur des interconnexions de serveurs stables, comme eDonkey en juin 2000, ou eMule en mai 2002 (utilisant initialement le protocole eDonkey), ont été conçus pour améliorer les résultats de recherche. Chaque nœud non serveur, appelé feuille, est connecté à un serveur. Ces serveurs sont stables, et gèrent un

très grand nombre de nœuds (jusqu'à un million de nœuds pour eMule par exemple). Ils n'ont pas besoin de communiquer entre eux, et ne partagent aucun fichier.

Les objets partagés par une feuille sont enregistrés sur le serveur responsable de cette feuille. Lorsqu'une feuille recherche un objet, elle envoie sa requête à son serveur. Celui-ci effectue alors la recherche parmi les objets des nœuds qui lui sont connectés. Le rôle de l'interconnexion peut varier selon les protocoles. Il n'a pas vocation à servir pour diffuser des requêtes, mais a plutôt un rôle de calcul de statistiques, ou de maintien de la liste des serveurs en fonctionnement par exemple. Ce système perd en décentralisation ce qu'il gagne en finesse et en nombre de résultats. En effet, la centralisation permet de bénéficier de recherches plus fines sur des serveurs regroupant un grand nombre de nœuds donc de fichiers. Plusieurs dizaines de serveurs permettent d'assurer une continuité dans le service.

2.2.1 eDonkey et eMule

eDonkey [26] est un logiciel client-serveur permettant le partage de fichiers. Il a donné son nom au protocole [25, 55] qui utilise des serveurs reliés entre eux, afin que chacun sache quel serveur est en fonctionnement. Ce protocole est aussi à la base des logiciels client-serveur eMule [61] (qui a proposé des extensions au protocole), eMule+, et MLDonkey. Les nœuds se connectent à un seul serveur (bien que rien n'interdise de se connecter à plusieurs comme l'a proposé MLDonkey). Les nœuds peuvent effectuer des recherches sur leur serveur mais aussi à l'ensemble des serveurs connus, afin de maximiser leurs chances de trouver un fichier satisfaisant. La distribution du logiciel client-serveur à l'origine du protocole eDonkey a cessé en septembre 2005 suite à des attaques judiciaires.

Chaque nœud se voit attribuer un identifiant unique. S'il n'est pas connecté à travers un pare-feu, un nœud se voit attribuer un identifiant dépendant directement de son adresse physique (adresse IP). Lorsqu'un nœud lance un requête de recherche, il obtient des réponses contenant chacune des informations sur le fichier trouvé et l'identifiant du nœud qui héberge la donnée. S'il souhaite télécharger cette donnée, il peut soit calculer l'adresse physique de la source à partir de son identifiant, et cela signifie que la source est accessible directement, soit le calcul de l'adresse physique est impossible. Dans le second cas, c'est que la source n'est pas accessible directement, cela arrive quand le port de l'application (TCP 4662) n'est pas utilisable. Cela peut être dû au fait que la source est connectée au système pair-à-pair *via* un pare-feu, un proxy, une traduction d'adresse (NAT), ou qu'elle est occupée. Le protocole permet alors l'envoi d'une requête au serveur auquel est connectée la source afin qu'elle se connecte elle-même au demandeur. Le problème d'un nœud derrière un pare-feu cherchant à communiquer avec un autre nœud lui aussi derrière un pare-feu persiste et n'est pas résolu par cette méthode. De toute façon, cette fonctionnalité augmente beaucoup trop la charge du serveur et a été désactivée de la plupart des serveurs (voir le chapitre 2.4 de [61]).

Le téléchargement multisource est aussi permis par eDonkey. L'extension eMule permet que, lorsqu'une source est trouvée, et dans le cas où le fichier est en cours de récupération, la source qui héberge la donnée fournit les autres sources qu'elle connaît

afin d'accélérer la récupération de la donnée à partir de plusieurs sources.

Afin de décourager les nœuds égoïstes, eMule a mis en place une extension permettant un système de crédits, utilisant l'algorithme de cryptographie à clé publique RSA. Ce système permet aux nœuds qui attendent de télécharger un fichier d'avancer plus vite dans la file d'attente des nœuds fournisseurs auxquels il a lui-même fourni des fichiers, et donc de récupérer le fichier demandé plus rapidement. Il s'agit donc d'un système de récompense *local*. Ce système de crédit est basé sur un *identifiant d'utilisateur* de 16 bits (différent de l'identifiant du nœud) qui est engendré aléatoirement, et dépendant de la machine associée au nœud. Cet identifiant reste donc le même au cours des différentes connexions du nœud au réseau.

eDonkey utilise des hachages de contenu de fichiers pour les différencier de manière sûre. Il décompose les fichiers en blocs récupérables indépendamment, ce qui permet d'accélérer la récupération des fichiers en parallélisant les téléchargements. Ces blocs sont identifiés au moyen de la fonction de hachage SHA1, ce qui limite la probabilité d'avoir une collision entre deux hachages. En particulier, cette fonction se révèle bien plus efficace que la fonction UUHash utilisée par FastTrack (système qui sera décrit plus loin au chapitre 2.3.1). De plus, eDonkey identifie aussi chaque fichier dans son intégralité par la concaténation de hachages (utilisant la fonction MD4).

Dans les protocoles eDonkey et eMule, il n'est pas possible d'attribuer plus de responsabilité à un nœuds ayant des capacités supérieures aux autres. De plus, deux limites au nombre d'utilisateurs existent : une limite matérielle interdit toute nouvelle connexion lorsque le nombre d'utilisateurs maximum permis est atteint. Une limite logicielle interdit toute nouvelle arrivée de nœud connecté derrière un pare-feu (nœud protégé) au delà de cette borne. Depuis 2004, certains serveurs eDonkey censurent les requêtes lorsqu'elles concernent certains mots-clés (sex, xxx, etc.), et interdisent le partage de données lorsqu'elles sont de certains types (mp3, vidéos, etc.). Cela rappelle encore une fois l'une des faiblesse des systèmes centralisés : la sensibilité à la censure. Toutefois, afin de diminuer l'un des autres problèmes posés par la centralisation et limiter la charge des serveurs, eDonkey a intégré à son logiciel client-serveur l'utilisation d'une table de hachage répartie baptisée Overnet, et basée sur Kademlia [70, 78]. Le logiciel client-serveur est devenu à cette occasion eDonkey2000. eMule a aussi intégré cette table de hachage répartie sous le nom de «Kad !». Le protocole Kademlia sera étudié dans le chapitre 3.1. Notons que l'administration du réseau eDonkey2000 et la propriété du logiciel client-serveur associé par la société MetaMachine ont mis fin à l'utilisation de ce protocole en septembre 2005, suite à des attaques judiciaires (voir le paragraphe *Grande échelle et tolérance aux pannes*). Le protocole eMule, ne dépendant d'aucune entreprise, continue toutefois de fonctionner.

2.2.2 Comportement des systèmes semi-décentralisés

Gérer de nombreux utilisateurs

Ce type de système n'est évidemment pas décentralisé. En effet, il repose sur un nombre limité de serveurs qui supportent toute la charge du réseau. Si un serveur se

déconnecte, tous ses utilisateurs doivent se reconnecter à un autre serveur. Les serveurs, et donc les nœuds qui y sont connectés, présentent donc l'inconvénient d'être à la merci d'une attaque ou d'une surcharge. C'est ce qui s'est passé avec le serveur Razorback au début de l'année 2006. Un autre problème du protocole eDonkey est que l'entreprise l'ait maintenu secret en ne distribuant qu'un logiciel client-serveur. Cela n'a probablement pas aidé à sa diffusion en limitant le nombre de logiciels client-serveur initiaux, donc le nombre de nœuds connectés au réseau. La diffusion gratuite du logiciel client-serveur eDonkey a dû cesser en septembre 2005 suite à une attaque judiciaire menée par le RIAA. Ici, le secret du code source non libre semble avoir eu le même effet que la centralisation. Il a mené à la fin du protocole eDonkey par manque de disponibilité du logiciel client-serveur, et a réorienté ses utilisateurs vers des protocoles proches comme eMule. Le protocole eMule, légèrement différent, continue en effet de fonctionner.

Tolérance aux pannes et charge du réseau

Les serveurs ayant un rôle prépondérant dans les systèmes semi-décentralisés, le réseau est particulièrement sensible à une attaque sur ceux-ci. La panne d'un serveur obligerait les nœuds à se connecter à un autre serveur. Ce nombre de serveurs étant constant par rapport à un nombre de nœuds qui varie, attaquer ces serveurs de manière légale ou logicielle n'est pas impossible et rendrait le réseau inutilisable. Les pannes d'arrêt ne sont pas gérées, rendant donc impossible le traitement des autres pannes.

Autonomie et envoi à des sous-réseaux

Le seul choix permis par les systèmes semi-décentralisés pour choisir les nœuds par lesquels peuvent passer les requêtes est le choix des serveurs auxquels un nœud enverra ses requêtes. Des sites existent actuellement afin de maintenir à jour des informations concernant ces serveurs. En particulier, ils indiquent les serveurs peu sûrs ou soupçonnés d'espionner les requêtes des nœuds qui y sont connectés.

Accès au réseau

La première connexion à ces systèmes se fait par la récupération de fichiers de serveurs sur un site HTTP, ce qui accentue encore la centralisation.

Protection des utilisateurs

Du fait de la centralisation, certains serveurs ont été lancés afin d'espionner les utilisateurs. En effet, plusieurs sociétés et associations se sont spécialisées dans la lutte contre l'utilisation des systèmes pair-à-pair. Celles-ci peuvent très facilement identifier l'adresse physique d'un utilisateur en se connectant à un système pair-à-pair semi-décentralisé. Ces organismes peuvent même faire fonctionner des serveurs afin de savoir ce qui est partagé par les utilisateurs connectés à ces serveurs. L'identification des nœuds eMule étant basée sur l'adresse physique, il est possible de connaître l'adresse physique d'un nœud à partir de son identifiant. Puisque chaque résultat de recherche pour un

objet est accompagné des identifiants des sources, il suffit d'effectuer une recherche sur un objet pour obtenir des adresses physiques d'utilisateurs de systèmes pair-à-pair hébergeant cet objet. Les systèmes semi-décentralisés sont donc peu apte à la protection des utilisateurs.

Dynamisme des nœuds

Dans un système semi-décentralisé, comme dans les systèmes centralisés, on bénéficie de l'avantage de la centralisation. L'arrivée et le départ des nœuds sont effectués en se connectant à un serveur, ce qui est immédiat. Le serveur, lui, ne peut cependant quitter le réseau puisque tous les nœuds connectés au réseau dépendent de lui.

Réactivité des nœuds

Un nœud est connecté à un seul serveur, donc il est facile pour un nœud «normal» d'assurer une bonne réactivité face à chaque message.

Rapidité de traitement des requêtes

Comme dans n'importe quel système centralisé, il suffit de deux sauts pour obtenir une réponse, c'est-à-dire que chaque serveur répond directement à la requête qui lui est envoyée. En effet, pour une requête de recherche qui lui est envoyée, un serveur ne cherche des réponses que parmi les nœuds qui sont connectés à lui. L'utilisation de hachage sur ce serveur permet d'accélérer la recherche locale parmi les objets partagés par ses feuilles. L'interconnexion, qui pourrait être utilisée pour accélérer la recherche, ne sert effectivement qu'à la gestion du réseau, comme le maintien à jour d'une liste des serveurs en fonctionnement.

Expressivité des requêtes

Bien que les systèmes semi-décentralisés permettent une expressivité maximale, jusqu'au expression rationnelles par exemple, la pratique montre que les serveurs eMule et eDonkey maintiennent une liste de hachage plutôt que des noms de fichiers. Cela leur permet d'accélérer les recherches, car une grande quantité d'objets est partagé par chaque serveur : cent à cent cinquante millions de fichiers sont partagés par un million de nœuds pour eMule par exemple [84]. Bien que la centralisation permette l'utilisation de tout type de recherche, elle empêche les plus complexes en raison de la charge qu'elles engendrent ! Une recherche par expression rationnelle n'est en effet pas faisable suffisamment rapidement pour répondre à toutes les requêtes arrivant à un serveur.

Exhaustivité des réponses

La centralisation permet à un nœud soit d'effectuer la recherche sur le serveur auquel il est connecté (en TCP), soit d'envoyer directement une requête à tous les serveurs connus (en UDP). Néanmoins, si la centralisation permet l'exhaustivité, les nœuds hébergeant des objets populaires reçoivent beaucoup de demandes. Les serveurs eMule ne

renvoient donc pas la totalité des adresses des nœuds hébergeant un objet demandé par une requête. Ils ne renvoient qu'un sous-ensemble ne comprenant pas les nœuds dont l'adresse a déjà été envoyée dans une réponse moins d'une minute auparavant.

Un second problème est la possibilité de censure de certaines requêtes et d'objets par certains serveurs. Cela diminue de beaucoup le nombre de réponses qui peuvent être obtenues par les nœuds connectés à ces serveurs par rapport aux nœuds connectés sur des serveurs non-censeurs.

Authentification des objets

Les objets peuvent être identifiés selon leur contenu, par hachage, afin de les comparer et permettre le téléchargement multisource.

Accessibilité des objets et «consommateurs égoïstes»

Chaque nœud est identifié de manière unique. Cela permet à chaque utilisateur de noter les autres utilisateurs selon les objets qu'il a récupérés (voir le chapitre 4.2.1 de [61]). Un système semi-décentralisé permet à un utilisateur de donner la priorité à un utilisateur qui lui a fourni beaucoup d'objets par le passé, lorsque plusieurs nœuds demandent à accéder à l'un de ses objets.

Réplication des objets

Aucun mécanisme de réplication n'existe dans les systèmes eDonkey et eMule, la diffusion des fichiers se fait uniquement *via* leur téléchargement.

Pare-feux et détection d'utilisation de systèmes pair-à-pair

Une fonction permet au serveur de jouer le rôle de tiers pour entamer une connexion vers un nœud protégé par un pare-feu, mais elle a été désactivée car elle s'est révélée coûteuse en capacité pour le serveur.

2.3 Les systèmes hybrides ou la prise en compte des différences entre les nœuds

Les systèmes hybrides ont pour but de limiter la charge trop importante dans les systèmes décentralisés non-structurés (décrits plus tard au chapitre 2.4) comme Gnutella. Ils permettent de palier la baisse d'efficacité due à l'augmentation du nombre d'utilisateurs des systèmes pair-à-pair en tenant compte de l'hétérogénéité des nœuds. On les retrouve parfois dans la littérature sous le nom de « réseaux de seconde génération ». C'est ainsi que sont apparus FastTrack [28] (utilisé par KaZaA [59] et Grokster) en mars 2001, puis WinMX qui intègre le protocole WPNP (WinMX Peer Networking Protocol) en mai 2001, Ares qui désavoue le protocole Gnutella 0.4 en 2002, et la nouvelle version de Gnutella [48] en mars 2003. Dans les systèmes hybrides, des nœuds

de haute responsabilité, nommés super-nœuds sont inter-connectés. Les nœuds « normaux », appelés feuilles, sont connectées à un seul super-nœud. Lorsqu'une recherche est effectuée, elle est envoyée à certains super-nœuds, qui la renvoient à certaines ou toutes leurs feuilles. Les recherches se font, comme pour les systèmes décentralisés, à partir de mots-clés et, parfois, de hachage de noms ou de contenus. Des différences existent évidemment entre les responsabilités des super-nœuds et des feuilles selon le protocole. Le protocole giFT-FastTrack, par exemple, propose un rapport de 1 super-nœud pour 100 feuilles. Ce type de systèmes permet de diminuer le nombre de messages reçus par une grande partie des nœuds (les feuilles) en augmentant la charge des super-nœuds mais sans imposer la maintenance de quelques serveurs comme les systèmes centralisés ou semi-centralisés. Cela permet aussi d'augmenter l'efficacité de la recherche sur les super-nœuds puisqu'ils sont responsables de bien plus d'objets qu'un nœud quelconque d'un système décentralisé non-structuré comme Gnutella (voire chapitre 2.4). Aucune structure n'aidant à la recherche, il peut se révéler utile de diffuser des indices aidant à la localisation des objets sur les nœuds si l'on veut limiter le nombre de messages. Notons que le choix des super-nœuds n'est pas trivial, et la problématique est similaire à celle que l'on retrouve dans les réseaux de censeurs et pour la domination en théorie des graphes.

2.3.1 FastTrack (KaZaA)

Le protocole FastTrack [28] est introduit en mars 2001. C'est un protocole propriétaire fermé, ce qui empêche de consulter son code source. Les communications sont chiffrées, et l'analyse du trafic est complexe. FastTrack n'est pas non plus documenté. Toutefois, plusieurs personnes ont analysé ce système, ce qui a permis de comprendre les communications entre les nœuds « normaux »et les super-nœud [28, 38]. Les fonctionnalités des communications entre super-nœuds restent cependant inconnues. Plusieurs logiciels client-serveur utilisent le protocole FastTrack, les plus connus sont KaZaA [59], Grokster, iMesh, Morpheus (jusqu'en 2002), giFT, Kazaa lite, K-Lite et MLDonkey. Des études sur le trafic engendré par KaZaA, comme [63], ont permis d'expliquer les comportements des utilisateurs et leur dynamique, et aussi de mieux comprendre les raisons du bon fonctionnement du protocole FastTrack. Les réseaux créés par les différents logiciels client-serveur FastTrack sont globalement compatibles. Dans ce protocole, les super-nœuds sont promus selon des critères non spécifiés. Un nœud peut cependant s'auto-promouvoir super-nœud. À son arrivée, un nœud s'insère dans le réseau grâce à une liste de serveurs pré-enregistrés dans le logiciel client-serveur. Ceux-ci lui permettent de se connecter à un super-nœud auquel il envoie alors la liste des fichiers qu'il souhaite partager sur le réseau. Les requêtes de recherche sont envoyées à ce même super-nœud. Un nœud récupère directement une donnée cherchée des nœuds qui la partagent. Le protocole de transfert est HTTP, afin de passer plus facilement les pare-feux. FastTrack est le protocole qui a introduit le téléchargement multisource, c'est-à-dire qu'il permet de récupérer le fichier à partir de plusieurs nœuds en parallèle, exploitant ainsi l'asymétrie de la bande passante (voir le chapitre 1.2.14). Pour identifier les données, FastTrack utilise la fonction de hachage UUHash, rapide, mais engendrant des

collisions en grand nombre. Cela a permis l'introduction facile de fausses données dans les réseaux FastTrack, car elles étaient identifiées comme similaires aux données authentiques. Malgré ces inconvénients, FastTrack reste le protocole pair-à-pair le plus utilisé dans le monde, jusqu'en 2004. Les utilisateurs du réseau FastTrack ayant été la cible d'attaques juridiques aux États-Unis, le nombre d'utilisateurs utilisant ce protocole a moins augmenté que celui de ses concurrents. En décembre 2005, après une attaque juridique en Australie, KaZaA a été interdit d'utilisation en Australie, ce qui a encore réduit sa communauté et les objets disponibles sur ce système.

2.3.2 Gnutella2

Gnutella, après avoir été confronté aux limites inhérentes à l'inondation, a proposé des évolutions vers un modèle hybride. Gnutella2 permet de diminuer le nombre de messages en limitant le nombre de connexions de chaque nœud. Les super-nœuds jouent le rôle de passerelle vers le réseau, et filtrent les requêtes du réseau vers les feuilles. Plusieurs contributions [44, 87, 91] ont approfondi le travail commencé avec Reflector [47]. Cela a fait évoluer Gnutella vers une seconde version du protocole, nommée 0.6 [48]. La version 0.6 est compatible avec la première version de Gnutella 0.4, et chacune des extensions qui y est proposée est utilisable séparément. Chaque super-nœud est connecté au plus à 10 autres super-nœuds, et il a entre 10 et 100 nœuds normaux connectés à lui. Un nœud peut être nommé super-nœud suivant des critères qui peuvent différer selon les logiciels client-serveur : cela peut dépendre du fait qu'il soit protégé par un pare-feu (qui rend difficile les connexions), de son système d'exploitation, de sa bande passante, du temps depuis lequel il est connecté au réseau, ou de ses ressources matérielles (puissance de calcul, capacité mémoire) [48, 91]. Une liste de fichiers est régulièrement mise à jour sur chaque super-nœud, contenant la liste des fichiers partagés par ses feuilles. Une recherche est effectuée en comparant les mots-clés de la recherche avec les noms des fichiers partagés de chaque nœud.

Afin d'accélérer les temps de recherche dans les listes des voisins des super-nœuds, l'extension QRP [87] propose d'utiliser des filtre de Bloom [12]. Ce procédé permet de savoir directement si un mot-clé apparaît dans le nom d'un fichier détenu par une feuille, plutôt que de devoir parcourir la liste de tous les fichiers partagés. Cependant, le hachage interdit la recherche approximative ou par expression rationnelle. De plus, QRP crée aussi des mots composé tirés du mot originel, auquel on a ôté la première lettre, les deux premières, ou les trois premières lettres. On fait de même avec la dernière, les deux dernières, et les trois dernières lettres du mot. Seuls les mots d'au moins trois lettres sont enregistrés dans la table de hachage, ces variations permettent par exemple d'ôter les pluriels. Lorsqu'une requête est reçue par un super-nœud, il hache les mots recherchés, et vérifie grâce à sa table de hachage si parmi certaines de ses feuilles un fichier correspond. Selon que le nœud effectue la recherche en appliquant un ET ou un OU entre les mots-clés, tous les mots devront être dans le nom du fichier ou un seul mot suffira.

De plus, une méthode de file d'attente est proposée afin de gérer les messages reçus suivant des priorités, pour que les nœuds ne soient pas surchargés de messages. Par

ailleurs, l'extension Ultrapairs [91] propose que le super-nœud cesse la transmission des requêtes de recherche aux voisins si les réponses trouvées sont en nombre suffisamment important. Enfin, l'extension QRP [87] permet de ne faire suivre une requête arrivée à un nœud qu'aux voisins réputés les plus adaptés.

L'extension HUGE [71] a été créée afin de pouvoir identifier des fichiers de manière unique, avec le but d'être inter-opérable avec les hachages utilisés par les autres systèmes, en utilisant des Tigertree (TTH). Cette possibilité permet à Gnutella 0.6 la récupération multisource d'objets, puisque HUGE permet d'être sûr que deux objets ont le même contenu.

Une méthode de mise en cache des ping a été proposée afin d'enregistrer sur chaque nœud les pings qui sont passés par ce nœud, avec toutes les informations qui y sont attachées. Cela permet, lorsqu'un nœud envoie un ping, de lui envoyer directement des informations sur les nœuds connectés au réseau sans avoir à attendre les réponses à ce ping ni à envoyer de messages pour chaque ping reçu. Les caches sont remis à jour régulièrement par chaque nœud. Cela limite donc le trafic de contrôle qui est, nous l'avons vu, très important dans Gnutella 0.4 [86].

Afin de rendre les recherches plus efficaces dans les réseaux hybrides, il est utile de diffuser des objets sur des nœuds, voisins par exemple. Cependant, il faut que ces objets ne nécessitent pas trop de mémoire. Un important travail [64] propose diverses méthodes inspirées de la sélection d'objets afin de renvoyer les requêtes sans inondation. Ces méthodes permettent de juger vers quel(s) voisin(s) (feuilles ou super-nœuds) envoyer une requête. Le choix peut être basé sur la similitude entre les mots-clés de la requête et les documents hébergés par le voisin : soit selon la présence des mots-clés *dans les titres* de chaque document (recherche par nom), soit selon la fréquence d'apparition des mots-clés *dans les documents* du nœud (recherche par contenu), soit encore selon que les mots-clés *apparaissent* dans au moins un document du nœud (recherche par correspondance). Pour un super-nœud le choix du super-nœud auquel renvoyer une requête peut se faire selon les requêtes auxquelles ont répondu ses voisins précédemment. Dans [64], une étude a été effectuée sur la base d'un système Gnutella 0.6. Les méthodes qu'elle propose, combinées à un élagage, permettent de limiter l'espace nécessaire à chaque super-nœud pour enregistrer les informations de fréquence et de références de chaque document partagé par ses voisins. L'étude comparative fournie par [64] montre, entre autres, que les recherches basées sur le contenu des documents hébergés dans un réseau plutôt que sur les mots contenus dans les titres de ces documents augmente l'efficacité. De plus, l'utilisation des fréquences de présence des mots dans les documents augmente aussi significativement la précision des recherches [64]. D'autre part, la recherche basée sur les seuls termes apparaissant dans les documents d'un nœud, plus simple que la précédente, est aussi bien moins efficace. Ainsi, des expériences montrent que pour un réseau de 35 super-nœuds auxquels sont connectés jusqu'à 1008 feuilles, un nombre de 114 messages suffit à obtenir une *précision* (pourcentage de réponses obtenues qui correspondent à la requête) de 72% . Cette même méthode permet d'obtenir un *rappel* (pourcentage de réponses correctes trouvées parmi le nombre total de réponses existant dans le réseau) de 29%.

Enfin, [48] propose de permettre des requêtes plus fines grâce à des méta-informations

(codées en XML), afin d'obtenir des réponses plus pertinentes. Ce protocole propose aussi d'attacher des méta-informations aux réponses afin de préciser la nature du fichier correspondant à la requête, et ainsi de permettre au demandeur de décider quel est le fichier le plus pertinent sans avoir à télécharger tous les fichiers renvoyés en réponses.

2.3.3 Comportement des systèmes hybrides

Gérer de nombreux utilisateurs

Un système hybride est décentralisé, bien que certains nœuds aient une charge plus lourde que les autres. Les super-nœuds reçoivent en effet les messages de leurs feuilles en plus de ceux de leurs voisins super-nœuds. Le choix des critères permettant à une feuille de devenir super-nœud est donc important. Ces critères peuvent se baser sur la bande passante du nœud, son temps de réponse, l'utilisation d'un pare-feu, sa puissance de calcul, sa capacité mémoire, le temps depuis lequel il est connecté au réseau, mais on pourrait aussi imposer une proportion maximale de super-nœuds par rapport au nombre de feuilles par exemple, que les super-nœuds soient répartis équitablement parmi les feuilles, ou demander un degré de confiance. Ils diffèrent d'un protocole à l'autre, voire d'un logiciel client-serveur à l'autre, mais un nœud peut généralement demander à devenir super-nœud.

L'autre problème face auquel se trouvent ces systèmes est le choix du nombre de feuilles connectées à un super-nœud, qui ne dépend pas du nombre de nœuds, et n'est donc pas adapté au dynamisme des réseaux pair-à-pair. Toutefois, même si une estimation du nombre de nœuds dans le réseau est possible, un super-nœuds n'a pas forcément la capacité d'augmenter le nombre de nœuds qu'il gère en même temps que le nombre de nœuds augmente dans le réseau, la création d'un troisième niveau dans la hiérarchie des nœuds serait alors nécessaire, ce qui suppose une réorganisation coûteuse. En pratique, la charge des super-nœuds est loin d'être négligeable (10 fois plus qu'une feuille dans Gnutella 0.6), d'où les tentatives de réduire le nombre de diffusions en sélectionnant les nœuds auxquels un message est transmis. Le choix de charger plus certains nœuds permet toutefois de décharger les liens menant aux feuilles, et de diminuer ainsi le nombre de messages envoyés dans le réseau physique. En effet, la connaissance des objets des feuilles permet de n'envoyer une requête à une feuille que si elle la concerne. Ainsi, dans Gnutella 0.6, les requêtes reçues par un super-nœud seront transmises aux seules feuilles qui ont les objets recherchés, et aux super-nœuds voisins.

Tolérance aux pannes et charge du réseau

Les systèmes hybrides permettent à des nœuds d'avoir plus de responsabilités que d'autres. Du fait qu'une feuille ne se connecte qu'à un super-nœud, si ce super-nœud est déconnecté du système, toutes ses feuilles le seront aussi. Les pannes d'arrêt ne sont donc sans effet que pour une feuille. Cela représente tout de même 90% à 99% des nœuds dans Gnutella 0.6.

Autonomie et envoi à des sous-réseaux

Si un nœud ne veut utiliser qu'une partie des nœuds ou des liens pour le routage de ses requêtes, les mêmes difficultés existent que pour les systèmes décentralisés non-structurés. Les mêmes solutions sont valables. Ni Gnutella 0.6, ni FastTrack ne permettent de restreindre les nœuds ou les liens utilisés pour le routage de leurs requêtes.

Accès au réseau

giFT-FastTrack [38] propose de contacter des adresses physiques aléatoires lorsqu'aucune autre méthode d'entrée dans le réseau n'est disponible. Cette méthode peut engendrer une surcharge importante. En effet, elle est utilisée dans le cas où le serveur ne répond pas, et où aucun nœud de précédentes sessions éventuelles ne répondent. Cette situation peut donc arriver si le réseau Internet est très chargé, et ce type de méthode peut engendrer un grand nombre de messages, et charger encore plus le réseau. Elle est donc clairement problématique pour la charge du réseau physique. Cette tentative pouvant être considérée comme agressive (vérifier l'activité d'un ou des ports est une méthode servant à préparer une attaque sur une machine), elle est désactivée par défaut.

Gnutella 0.6 bénéficie du même système d'introduction dans le réseau que Gnutella 0.4 (voir le chapitre 2.4.1). Dans Gnutella 0.6, comme dans le logiciel client-serveur giFT-FastTrack, un nœud contacté donnera l'adresse de son super-nœud. giFT-FastTrack fait aussi appel aux nœuds contactés lors d'une précédente session. En cas de première session, il utilise un fichier fourni par un serveur (`sourceforge.net`) recensant les super-nœuds actifs (ces super-nœuds envoient leur liste de nœuds toutes les 4 heures par défaut). Une fois un super-nœud FastTrack contacté, il renverra les adresses des autres nœuds qu'il connaît (autour de 200). Dans le cas où ces méthodes échouent, giFT-FastTrack propose de contacter sur le port FastTrack (port 1214) des adresses physiques aléatoires dans l'intervalle 24.0.0.0/8., en se basant sur l'observation que 1% des utilisateurs de ces adresses IP utilisent FastTrack.

Protection des utilisateurs

Les logiciels client-serveur privés utilisant le protocole FastTrack, comme KaZaA, ne livrant pas leur code source, chiffrant leurs messages, et demandant un accès réseau permanent, il est difficile de vérifier s'ils ne transmettent pas des informations autres que celles nécessaires au partage d'objets. En effet, les nœuds de systèmes décentralisés ou hybrides peuvent être amenés à envoyer des objets à n'importe quel nœud qui le rechercherait. Cependant, les messages étant chiffrés, il est difficile de savoir si le message envoyé ne contient pas d'informations autres que celles partagées par le nœud. Ainsi, KaZaA inclue un logiciel espion, ce qui a mené à la création du logiciel client-serveur Kazaa lite afin de ne pas souffrir de la diffusion d'informations personnelles.

Par ailleurs, de manière générale dans un système hybride, un super-nœud ayant accès à tous les objets partagés par ses feuilles, il est très facile de savoir quels sont les objets partagés par celles-ci. Il est aussi simple de savoir quels sont les objets intéressant une feuille : il s'agit d'un sous-ensemble des objets qu'il recherche (sous-ensemble pour

le cas où il envoie des requêtes aléatoires ou inutiles). Les systèmes hybrides souffrent donc de la même faiblesse que les systèmes centralisés, à moindre échelle puisque le nombre de nœuds connectés à un super-nœud est moindre que pour un serveur unique.

Un moyen simple de cacher les intérêts d'un nœud est de se déclarer comme super-nœud pour plusieurs nœuds virtuels, afin de laisser croire que les requêtes envoyées par ce nœud sont issues de nœuds différents.

Dynamisme des nœuds

Les feuilles des architectures hybrides n'ont qu'un voisin. Elles se connectent donc rapidement, et leur déconnexion n'ayant aucune incidence sur le fonctionnement du réseau, elles peuvent partir sans avoir à se chercher de nœuds remplaçants. Un super-nœud ayant plus de responsabilité, son départ aura plus d'incidence sur le fonctionnement du réseau, car ses feuilles seront déconnectées. Il devra prévenir ses feuilles afin de leur permettre de se reconnecter à un autre super-nœud, ainsi que ses voisins super-nœuds (une dizaine). À sa promotion en super-nœud, un nœud n'a pas de feuilles, cette promotion est donc très rapide puisqu'elle consiste à se connecter à un dizaine de super-nœuds (pour Gnutella 0.6). Idéalement, le nombre de super-nœuds auquel est connecté un super-nœud reste faible, afin de ne pas surcharger les super-nœuds par des requêtes qui sont *a priori* envoyées à tous les voisins.

Réactivité des nœuds

Les systèmes hybrides font clairement le choix de donner à certains nœuds plus de responsabilités pour favoriser les nœuds ayant peu de capacités. Les feuilles ont donc une meilleure réactivité que dans un système décentralisé habituel, puisqu'elles ne sont connectées qu'à un seul super-nœud. En particulier vérifier la connexion avec ce nœud est facile. Les super-nœuds gèrent au contraire plus de nœuds que les nœuds d'un système décentralisé non-structuré, cela leur nécessite donc plus de capacités pour obtenir la même vivacité. C'est une des raisons pour lesquelles le choix des critères permettant à une feuille de devenir super-nœud est important, pour le dynamisme du système cette fois. Un super-nœud doit aussi maintenir les connexions avec les autres super-nœuds par le biais de pings réguliers. Il doit aussi vérifier que ses feuilles sont toujours connectées, bien que l'on puisse alléger la charge des super-nœuds et diviser le trafic réseau par deux en considérant qu'une feuille n'ayant pas envoyé de message depuis longtemps est déconnectée.

Rapidité de traitement des requêtes

Les protocoles de systèmes hybrides laissent les logiciels client-serveur très libres concernant la distance à parcourir pour une requête de recherche. Cela signifie que, pour Gnutella 0.6 par exemple, le nombre de sauts maximum effectué est fixé par le logiciel client-serveur. Toutefois, une requête arrivant à un nœuds avec un nombre de sauts restant à effectuer abusivement grands est supprimée. Dans des systèmes comme Gnutella 0.6, il est possible d'effectuer une première recherche avec un nombre maximum

de sauts faible, puis si les résultats sont trop peu pertinents ou trop peu nombreux, refaire cette recherche en précisant un nombre de sauts maximum supérieur. Bien que le protocole FastTrack et sa méthode de recherche soient fermés, on sait qu'il permet de limiter la recherche en précisant le nombre maximum de résultats demandés (voir message *Search query 0x06* dans [28]), augmentant ainsi la rapidité du système.

Expressivité des requêtes

Un système hybride permet, tant qu'il n'utilise pas de hachage, d'effectuer tout type de recherche, des recherches exactes aux recherches par expressions rationnelles.

Exhaustivité des réponses

N'étant qu'une évolution des systèmes décentralisés non-structurés, les architectures hybrides ont la même limite : elles ne permettent pas d'accomplir des recherches exhaustives sans explorer tout le réseau, ce qui est trop coûteux. Les recherches de Gnutella 0.6, par exemple, se cantonnent aux nœuds à distance bornée, donc aux objets enregistrés sur des super-nœuds proches. Cependant, chaque super-nœud a en charge un grand nombre d'objets et peut donc potentiellement fournir des réponses à de nombreuses requêtes. Le rapport entre le nombre d'objets gérés par un super-nœud Gnutella 0.6 et le nombre d'objets gérés par un nœud Gnutella 0.4 est *a priori* le même que le rapport entre le nombre de feuilles et le nombre de super-nœuds, c'est à dire de 100 pour 1 dans Gnutella 0.6.

Authentification des objets

Aucun système hybride ne permet actuellement d'assurer l'origine d'un objet ni ne permet d'exprimer un degré de contentement des nœuds l'ayant récupéré. La difficulté est la même que pour les systèmes décentralisés non-structurés. Cependant, l'existence de super-nœuds donne déjà des responsabilités à certains nœuds, et permet d'assurer une certaine centralisation dans le cas de notations de fichiers. C'est ce qui est fait dans les logiciels client-serveur KaZaA récents [59], qui permettent la notation de fichiers partagés selon leur intégrité et leur qualité, ces fichiers étant identifiés de manière unique grâce à leur clé de hachage. Les nœuds qui évaluent les fichiers sont favorisés en avançant plus vite dans les files de téléchargement. Le système de crédit de FastTrack ne peut cependant fonctionner que dans le cas où l'on fait confiance aux super-nœuds du système, or un nœud FastTrack peut s'auto-promouvoir super-nœud. Le protocole étant fermé, il est donc impossible de s'assurer que ce système de notation soit sûr.

Accessibilité des objets et «consommateurs égoïstes»

Gnutella 0.6 bénéficie toujours de la méthode d'interdiction des logiciels client passant par une page HTTP. Concernant FastTrack, bien que probable, l'existence d'une méthode d'encouragement au partage est difficile à vérifier.

Réplication des objets

Aucun mécanisme de réplication actif n'existe, ces systèmes n'étant pas fait pour l'entrepôt permanent d'objets. Gnutella 0.6 (avec l'extension HUGE [71]), comme FastTrack, permet le téléchargement multisource. KaZaA est d'ailleurs l'un des premiers à avoir introduit cette possibilité dans les systèmes pair-à-pair.

Par ailleurs, il est possible de répartir la charge due aux réponses renvoyées parmi les différents nœuds qui ont une copie d'un objet en cache. [15] propose ainsi que chaque nœud recevant une requête pour une clé en cache décide, selon une certaine probabilité, de renvoyer la requête à un voisin qui a aussi l'objet et dont le degré est le plus faible. Sinon, il envoie l'objet au demandeur.

Pare-feux et détection d'utilisation de systèmes pair-à-pair

Gnutella 0.4 fût le premier à penser à la difficulté créée par les pare-feux et à avoir tenté de le résoudre. La méthode permettant de contourner cette difficulté n'a pas changé dans la version 0.6, FastTrack utilise d'ailleurs la même méthode, comme bien d'autres protocoles.

2.4 Les systèmes décentralisés non-structurés : vers une égalité entre les nœuds

Les systèmes décentralisés non-structurés font leur apparition à la suite des systèmes centralisés, en réaction aux attaques juridiques dont ces derniers sont la cible. Pour contrer ce danger qui fait fuir les utilisateurs, ils répartissent la totalité des fonctions du système entre les nœuds : la recherche tout comme le routage et la récupération des objets. Tous les nœuds ont alors le même rôle. Ils se connectent entre eux sans contraintes à leur entrée dans le réseau, et émettent ensuite leurs requêtes sans assurer un routage, c'est-à-dire que les requêtes n'ont pas de destinataire précis.

Pour des systèmes de partage de données, la recherche se fait sur la base de mots-clés et parfois de hachage de mots présents dans les noms de fichiers. Plus rarement, dans le cas de textes par exemple, elle utilise les hachages de mots contenus dans les fichiers (dans les textes bruts, pdf, etc.). Cela permet d'augmenter la précision (les mots d'un texte sont plus nombreux) et la pertinence (certains mots du contenu permettent de différencier deux textes dont le titre est identique) de la recherche. Les systèmes pair-à-pair décentralisés non-structurés ne disposant d'aucune topologie d'interconnexion, ni de serveurs centraux, pour faciliter la routage, une requête de recherche doit découvrir les informations disponibles lors de son routage. Il peut donc se révéler utile de diffuser quelques informations concernant les nœuds ou les données afin d'aider au routage si l'on désire limiter le trafic engendré par les requêtes de recherche. Dans le même esprit, la réplication des objets (voir chapitre 1.2.14) permet d'augmenter l'accessibilité des objets dans le réseau, en diminuant la distance à laquelle se trouve un objet d'une partie des nœuds du réseau.

Gnutella, dans sa première version [39], a pris le relais de Napster. C'est le premier système décentralisé non structuré. Il s'agit d'un système de partage de fichiers créé en mars 2000 par Frankel et Pepper. Il permet les recherches de fichiers (l'application grand public qui a fait connaître Gnutella) mais aussi tout type d'objets. Freenet[17] est un protocole de partage de données qui veut procurer plus de sécurité pour l'utilisateur. Il fut présenté en mars 2000 par Clarke, bientôt rejoint par Sandberg, Wiley, Hong et aidé de plusieurs développeurs. Son objectif affiché est d'assurer l'anonymat des nœuds du réseau, qu'ils effectuent des recherches ou mettent à disposition des objets. Nous détaillerons ces deux systèmes ci-dessous.

2.4.1 Gnutella

Gnutella (dans sa première version, la version 0.4), a connu quasiment autant de variantes que de logiciels client-serveur (Gnucleus, BearShare, Clip2 DSS, FirstPeer, giFT, Gnotella, Gnucleus, GPulp, gtk-gnutella, iMesh, LimeWire, Mactella, Morpheus, Newtella, Shareaza, ToadNode, etc.). En effet, aucun protocole n'ayant été diffusé, les logiciels client-serveur ont utilisé ce qu'ils ont pu déduire du protocole, menant ainsi à des versions de Gnutella différentes et parfois même incompatibles [99]. Le nombre de voisins n'est pas fixé par le protocole, et aucune méthode n'est imposée pour les choisir. Assez rapidement cependant, des entreprises comme Clip2 DSS [42], Lime Wire LLC (LimeWire) et First Peer (Bearshare) ont tenté d'uniformiser les différentes versions de protocoles utilisés [39] en identifiant les contraintes minimales permettant la compatibilité entre les différents logiciels client-serveur Gnutella. Grâce à ces études poussées et cette standardisation, ce système pair-à-pair fut donc le premier à faire l'objet de travaux pour mieux comprendre son fonctionnement et l'améliorer, comme l'ont proposé LimeWire avec ses passerelles Gwebcache [51], les différentes amélioration proposées par les développeurs [43], ou l'entreprise Clip2 DSS [45, 46].

Pour la recherche, Gnutella utilise une méthode d'inondation des requêtes plutôt qu'un routage vers un nœud précis. L'inondation est bornée par un nombre de sauts, fixé typiquement à 7 par défaut. Cela signifie qu'une requête n'effectue qu'un nombre limité de sauts. À chaque nœud recevant la requête, le nombre de sauts restant est décrémenté puis la requête est envoyée à tous les voisins. Pour renvoyer une réponse à une requête de recherche, Gnutella permet un retour direct de la réponse vers le nœud demandeur. Cette méthode est la plus simple et la plus rapide si cet envoi est autorisé (un pare-feu pourrait empêcher un tel envoi). Dans le cas contraire, la réponse peut revenir le long du chemin parcouru par la requête à l'aller (cela permet de résoudre un problème de pare-feu). Dans ce dernier cas, la longueur du chemin de retour parcouru sera évidemment plus long, mais néanmoins limité par la distance d'inondation.

Un grand nombre de travaux ont tenté d'améliorer la recherche de Gnutella, par exemple en permettant une inondation incrémentale [65], qui commence à inonder les voisins jusqu'à distance 1, puis 2, puis 3 etc. jusqu'à avoir trouvé une réponse. Cette méthode fonctionne bien lorsque les objets populaires sont plus répliqués que les objets peu recherchés.

[65] montre que des stratégies basées sur des marches aléatoires peuvent avanta-

geusement remplacer l'inondation. Un marcheur aléatoire vérifie alors régulièrement si l'origine de la requête veut continuer la recherche afin d'arrêter la marche une fois la requête ayant trouvé une réponse.

2.4.2 Freenet

À la suite de Gnutella mais dans une optique résolument plus paranoïaque arrive Freenet [16, 17, 34]. Ce protocole de partage de fichiers utilise des outils cryptographiques pour revendiquer l'anonymat de chaque utilisateur. En effet, avec les premiers procès à l'encontre des utilisateurs de systèmes pair-à-pair (serveurs ou utilisateurs) croît la menace qui pèse au dessus des utilisateurs de ces systèmes. Freenet maintient un réseau logique sans structure, mais les identifiants se choisissent au fur et à mesure des voisins proches de leurs identités, on voit arriver les principes qui donneront naissance aux systèmes décentralisés structurés (voir chapitre 2.5). Dans ce protocole, chaque nœud se voit attribuer un identifiant et est initialement connecté à des voisins tirés au hasard. Toutefois, les nœuds s'organisent progressivement en se connectant aux nœuds d'identifiants proches, sans pour autant maintenir une structure permettant un routage efficace, ce qui explique le nom parfois utilisé de réseaux faiblement structurés.

Freenet pousse plus loin le concept d'anonymiseur et de routage en oignon [49, 75]. Dans le routage dit «en oignon», un message envoyé par une source u_d vers un destinataire u_0 par l'intermédiaire de nœuds $u_{d-1} \dots u_1$. Pour cela, le message destiné au $i^{ème}$ intermédiaire u_i est chiffré avec la clé publique de i puis associé à l'identité de i. Ce message dit «en oignon» est donc créé récursivement par l'envoyeur et chiffré récursivement avec la clé privée de chaque nœud intermédiaire, en partant du dernier u_0 et en revenant jusqu'au premier u_{d-1}. Ainsi, aucun nœud intermédiaire ne connaît le destinataire ni même d'autre intermédiaire que le prochain routeur. Dans Freenet, pour chaque donnée hébergée par le système, une clé est attribuée dans le même espace que les identifiants des nœuds, par hachage cryptographique [17], afin qu'il soit particulièrement difficile de retrouver la nature de l'objet à partir de la clé. Cette clé est créée afin qu'une clé soit associée à un unique objet, et *vice versa*. L'adresse physique du nœud d'où provient la donnée est également associée à cette clé. Chaque donnée est chiffrée avant d'être partagée, la clé de la donnée permettant de la déchiffrer. Une donnée hébergée par un nœud n'étant pas accompagnée de sa clé, le nœud qui héberge cette donnée ne peut savoir quelles sont les données qu'il héberge. Les auteurs de Freenet affirment ainsi qu'un nœud ne peut donc être tenu responsable des données qu'il héberge.

Chaque nœud recevant une requête de recherche d'une clé vérifie s'il a la donnée associée. S'il l'a, celle-ci est renvoyée suivant le même chemin qu'à l'aller. S'il ne l'a pas, mais qu'il a la clé associée à une adresse physique, la requête est envoyée à cette adresse. S'il n'a ni la donnée, ni la clé, la requête est renvoyée au voisin non visité dont l'identifiant est le plus proche de la clé, c'est à dire dont le *ou exclusif* entre l'identifiant du voisin et la clé recherchée donne le plus petit résultat. Si tous les voisins ont déjà été visités, la requête est renvoyée au nœud qui l'a envoyée. Ainsi, la requête visite le réseau suivant un parcours en profondeur d'abord, guidé par l'identifiant le plus proche de la clé recherchée. Il est possible de limiter le nombre de sauts dans le réseau afin de limiter

le nombre de messages transitant dans le réseau, cela peut alors rendre la recherche non exhaustive.

Lorsqu'un objet correspondant à la requête est trouvé, il est renvoyé le long du chemin pris par la requête (sans les boucles). Cet objet est alors copié sur certains nœuds, de manière probabiliste. Dans ce cas, l'origine indiquée dans la requête est changée en l'identifiant de ce nœud. De plus, lorsqu'un nœud reçoit une recherche lancée par un nœud ou une réponse renvoyée par un nœud, il fait de l'origine du message un nouveau voisin. Le nombre de voisins est borné, et les liens sont supprimés selon une politique de cache du moins récemment utilisé (LRU). De même, les données les moins récemment utilisées sont effacées, tout en gardant leur clé associée à l'adresse physique de leur source.

Les auteurs proposent donc d'utiliser ce modèle pour que chaque nœud acquière au fur et à mesure des objets relatifs aux clés proches de son identifiant, arguant que les chemins de recherche (et donc de retour) des objets correspondant à ces clés ont une probabilité de passer par ce nœud supérieure à la probabilité de passer par un nœud tiré aléatoirement. Il en est de même pour les voisins qu'un nœud va découvrir au fur et à mesure qu'il enverra, recevra et retransmettra des requêtes dans le réseau.

Nous verrons au chapitre 5 que des adaptations de ces systèmes ont été proposées, utilisant les propriétés spécifiques de ces systèmes pour créer des algorithmes de recherche efficaces.

2.4.3 Comportement des systèmes décentralisés non-structurés

Gérer de nombreux utilisateurs

Les méthodes de recherche de Gnutella et Freenet ont l'avantage de décentraliser la recherche en plus de l'échange des objets, rendant ainsi plus difficiles les attaques sur un point faible des systèmes.

Cependant, concernant Gnutella, si la charge par nœud est diminuée par rapport à celle d'un serveur, l'inondation alourdit la charge imposée à chaque lien. En conséquence, la charge des nœuds de fort degré, et celle des liens attenants, est plus importante que celle des autres nœuds, car ils sont plus susceptibles de recevoir des requêtes. Les mesures effectuées sur le réseau Gnutella par Clip2 [41] durant une heure annoncent un minimum de 1.000 nœuds en juillet 2000, et 10.000 après août. La distribution des degrés des nœuds du réseau Gnutella a été observée comme une loi de puissance. La charge peut donc varier d'un nœud à l'autre, car quelques nœuds ont un degré très grand. La charge potentielle des nœuds de grand degré est plus importante que celle des autres nœuds puisque la probabilité de recevoir une requête est d'autant plus grande que le nombre de voisins est élevé. Il faut toutefois rappeler que le degré maximal est paramétrable et qu'il influe sur la quantité de résultats obtenue lors d'une recherche.

La charge (en nombre de messages) engendrée dans Gnutella par le trafic de contrôle est d'environ 50% du trafic total [86], ce qui représente un grand nombre de messages. De plus, même dans le trafic de requêtes, les messages sont envoyés aveuglément à tous les voisins, alors que le nombre de voisins qui peuvent effectivement répondre à ce message

est faible. C'est pourquoi différents travaux ont analysé la possibilité de réduire le trafic engendré par les requêtes [29, 53, 95]. L'utilisation de liens sémantiques permet de lier chaque nœud du système avec des nœuds ayant des intérêts similaires. Dans [95], un nœud a des liens vers des nœuds qui ont renvoyé des réponses à des recherches menées antérieurement dans le système. Ils contiennent donc des objets qui intéressent le nœud. Afin de limiter le nombre de voisins sémantiques, une mesure de proximité permet alors de les classer. Lorsque le nombre de voisins sémantique d'un nœud atteint son maximum, il est possible de choisir les voisins les moins intéressants pour les remplacer par de nouveaux voisins potentiellement plus utiles au nœud. Ces liens, appelés raccourcis, sont empruntés en priorité, et l'inondation n'a lieu que lorsque l'utilisation de ces raccourcis se révèle inefficace. Les évaluations effectuées sur des traces de proxy (trafic HTTP, pair-à-pair, et autres) estiment que la mise en place de ce type de mécanisme permet de donner des réponses aux requêtes en passant par les raccourcis pour 45% à 90% des recherches. Ce travail étudie aussi l'intérêt d'ajouter plusieurs raccourcis à la fois, autant de raccourcis que possible, et d'envoyer les requêtes aux raccourcis de raccourcis. Les travaux [29, 53] étudient l'impact de voisins dont les intérêts sont réputés proches dans les systèmes pair-à-pair. Ils ne sont pas exclusivement liés à Gnutella, ils seront détaillés dans le chapitre 6.

Concernant Freenet et Gnutella, le degré est aussi paramétrable, mais les nœuds ayant un identifiant proche d'une clé populaire sont susceptibles de recevoir beaucoup de messages la concernant, et leur charge sera alors plus importante. Le choix du nombre de voisins gagnerait à se faire selon le nombre de nœuds présents dans le réseau, afin de s'adapter à l'évolution du réseau.

Tolérance aux pannes et charge du réseau

Un système décentralisé non-structuré et son réseau étant totalement décentralisés, la panne d'un nœud ne remet pas en cause le fonctionnement de l'ensemble. Cependant, il peut rallonger considérablement le diamètre du réseau, voire même le déconnecter en deux parties. Ainsi, lorsque les connexions à haut débit asymétrique cohabitaient encore avec beaucoup de connexions bas débit, il fut constaté que les nœuds connectés à ce système par liaisons à bas débit devenaient des goulots d'étranglement pour les messages du réseau. En effet, leur faible débit et leur faible réactivité ne leur permettaient pas de router tous les messages, contrairement aux connexions hauts débit qui transmettaient tous les messages. L'apparition de ce phénomène correspond au moment où le nombre de requêtes par secondes et par nœud a augmenté de manière importante (jusqu'à 10 messages de recherche par seconde et par nœud) [40]. Cette augmentation du nombre de messages par nœud coïncide avec la fin du procès contre Napster. Le faible nombre de messages que pouvaient retransmettre les nœuds à bas débit déconnecte en août 2000 le réseau Gnutella en plusieurs réseaux distincts [40, 99]. Cette scission eut pour effet de diminuer le nombre de résultats obtenus lors des requêtes envoyées par les nœuds au réseau Gnutella.

En octobre 2000, la société Clip2 introduisit Reflector [47], un système permettant de connecter les nœuds à bas débit à un unique serveur ayant une connexion rapide. Cela

permet ainsi de maintenir les nœuds bas débit connectés à proximité du cœur du réseau, tout en diminuant le nombre de messages transitant dans le réseau puisque ces nœuds bas débit sont des feuilles dans le graphe. Simultanément, plusieurs propositions [44] ont été faites pour tenter de rendre son efficacité au réseau Gnutella. C'est le début de l'arrivée des systèmes hybrides, c'est-à-dire de l'utilisation de nœuds exerçant différentes responsabilités. Entre novembre et décembre 2000, les logiciels client-serveur Gnutella LimeWire et Bearshare commencèrent à sélectionner leurs voisins selon leur réactivité et leur débit [99]. Les nœuds connectés à bas débit ont alors vu leur degré devenir très faible et se sont donc retrouvés à l'extrémité du réseau, tandis que les connexions à hauts débits avaient des degrés plus importants, et formaient le cœur du réseau Gnutella. En effet, peu de nœuds utilisant LimeWire et Bearshare acceptaient de garder comme voisins des nœuds jugés peu efficaces. Cela explique le faible degré [40] de ces nœuds aux capacités réseaux limitées. Cette différence de degrés a fait apparaître des nœuds mieux connectés, devenus plus importants pour le fonctionnement du réseau. Ces nœuds ont par conséquence une charge supérieure à la moyenne.

Autonomie et envoi à des sous-réseaux

Ni Gnutella ni Freenet ne permettent de limiter l'envoi à un sous-réseau. Cependant, assurer qu'un message ne passe pas par certains nœuds ou liens pourrait se faire au moyen d'un enregistrement préalable de chaque nœud sur les nœuds du système qui pourraient être utilisés pour des recherches. Cet enregistrement préciserait pour chaque nœud les nœuds qu'il accepte d'emprunter ou pas pour une recherche. Cependant, on a vu que Gnutella utilisait une inondation qui touche potentiellement beaucoup de nœuds, et Freenet utilise une exploration potentiellement totale du réseau. Cet enregistrement pourrait donc se révéler coûteux en mémoire, si chaque nœud du réseau enregistre quels voisins utiliser ou pas. Une alternative pourrait consister à insérer dans les messages les nœuds ou liens à utiliser ou à interdire. La taille du message serait cependant alors augmentée, et chaque lien devrait donc supporter des messages bien plus grands.

Accès au réseau

Les premières versions de Gnutella reposaient sur l'entrée manuelle d'adresses physiques. La récupération de ces adresses se faisait par l'utilisateur via des applications telles qu'IRC ou une page HTTP. Puis, lors des connexions suivantes, le logiciel client-serveur tentait de se reconnecter aux voisins des précédentes connexions. L'utilisation de caches inter-connectés enregistrant les adresses IP des nœuds du réseau Gnutella passant par eux (protocole nommé gnuCache, et renommé ensuite Gwebcache [51], proposé par Gnucleus) a ensuite été proposé. Ainsi, lorsqu'un nouveau nœud se connecte à un cache, il récupère l'adresse physique d'un ou plusieurs nœuds du réseau auxquels il peut se connecter. Cette fonctionnalité permit l'automatisation de la connexion au réseau par les logiciels client-serveur à leur lancement. Freenet, pour assurer l'anonymat (et donc interdire de choisir un voisin), rend arbitraire le point d'entrée en utilisant un mécanisme pseudo-aléatoire pour choisir les voisins d'un nouveau nœud.

Protection des utilisateurs

N'utilisant aucun serveur, les systèmes comme Gnutella rendent la surveillance des comportements des utilisateurs plus difficile. La protection des utilisateurs est donc légèrement supérieure aux systèmes centralisés. Toutefois, les requêtes sont envoyées de façon non chiffrée dans le réseau. N'importe quel nœud placé sur le chemin d'une requête peut donc savoir quelle donnée est cherchée, et par qui.

Freenet, en revanche, est un protocole entièrement chiffré. Les objets, tout comme les clés, sont chiffrés, rendant difficile la découverte des objets demandés, hébergés, ou transférés. Une possibilité accessible à un adversaire « tout-puissant »serait de surveiller les messages arrivant et repartant d'un nœud en se basant sur le fait que si un message part sans qu'un message ne soit arrivé, c'est une requête pour une clé qui vient d'être envoyée par ce nœud. Cependant, le chiffrement rend difficile de savoir à quel objet correspond cette clé.

Dynamisme des nœuds

Utilisant un réseau logique sous-jacent non structuré pour connecter les nœuds, Gnutella permet le même dynamisme que les systèmes centralisés. Joindre et quitter le réseau est très simple grâce à l'absence de structure. Des pings permettent de vérifier le fonctionnement de ses voisins tout en découvrant, grâce aux réponses de ces derniers, de nouveaux nœuds, et donc autant de voisins potentiels.

Réactivité des nœuds

Du fait du faible degré moyen et de la distribution des degrés en loi de puissance, seuls quelques nœuds de grand degré peuvent mettre du temps à répondre aux requêtes à cause du nombre important de requêtes reçues. Cependant, ce degré est configurable et il est raisonnable de penser qu'un nœud qui se voit incapable d'agir et de répondre va diminuer son degré.

Rapidité de traitement des requêtes

Le système Gnutella configuré par défaut est relativement rapide puisque seuls 7 sauts (14 sauts si l'on compte les aller-retours) sont nécessaires pour obtenir une réponse à sa recherche. Ce nombre de sauts est configurable. Concernant Freenet, il est possible de limiter le nombre de sauts, afin d'éviter des messages qui parcourent tout le réseau.

Expressivité des requêtes

Gnutella se limite à des recherches exactes ou par sous-chaîne, c'est-à-dire aux chaînes de caractères comprenant les mots-clés recherchés. Cependant, ce type de système permet toutes les recherches élaborées comme les recherches approximatives, par intervalles, ou par expressions rationnelles.

Exhaustivité des réponses

Gnutella limite la distance de recherche à 7 sauts par défaut. La recherche n'y est donc pas exhaustive. Pour atteindre l'exhaustivité dans Gnutella, il faudrait explorer tout le réseau, ce qui signifierait l'exploration de tous liens. La limite du nombre de sauts permet de ne pas inonder le réseau de messages. Toutefois, le nombre de sauts nécessaire n'est pas le même pour des requêtes vers des objets rares ou des objets populaires, et il varie avec le nombre de nœuds dans le réseau. De plus, la charge en nombre de messages est trop forte pour les nœuds reliés au réseau par une faible bande passante (modem), ce qui a d'ailleurs mené au fractionnement du réseau Gnutella en août 2000, et donc à une diminution du nombre d'objets accessibles.

Concernant Freenet, la recherche est exhaustive si on ne limite pas le nombre de sauts, puisque la requête effectue une exploration en profondeur d'abord. Il est possible de limiter ce nombre de sauts au prix de l'exhaustivité.

Authentification des objets

Gnutella ne permet pas d'assurer qu'une donnée est ce qu'elle prétend être (par le nom de son fichier), ou même de savoir si elle est plébiscitée par les nœuds du réseau. La difficulté habituelle d'évaluation d'un objet ou d'un nœud est renforcée par le fait que, dans un environnement décentralisé et non structuré, aucune architecture ne permet actuellement de répartir des responsabilités concernant l'évaluation de la qualité des objets. Plusieurs nœuds pourraient prendre le rôle d'évaluateur pour un même objet, et il faudrait alors coordonner ces nœuds. Un nœud pourrait aussi tenter de tromper les nœuds sur la qualité d'un objet ou d'un nœud. L'utilisation d'algorithmes de quorum répartis pourrait permettre de résoudre ce problème.

Accessibilité des objets et «consommateurs égoïstes»

[5] a montré que de nombreux utilisateurs ne mettaient à disposition aucune donnée. Si des mécanismes ont permis d'interdire les recherches effectuées à partir de pages HTTP, aucun mécanisme évolué n'a été mis en place concernant les nœuds ne partageant rien. Il est cependant à noter qu'une fonctionnalité de Gnutella permet de connaître le nombre (et la taille) de données mises à disposition par un nœud, bien que cette information puisse être falsifiée. Un mécanisme pourrait se baser sur ces critères pour favoriser les nœuds qui partagent beaucoup de données, et interdire les nœuds qui ne partagent rien. Il suffirait néanmoins à ces derniers de diffuser de fausses informations pour tromper le contrôle. Un mécanisme plus évolué pourrait se baser sur les refus d'envoi de données pour renforcer le contrôle.

Réplication des objets

La recherche n'étant basée sur aucune structure, la réplication passive des données est la méthode la plus souvent utilisée dans les systèmes décentralisés non structurés. Dans Freenet, les fichiers transmis d'un fournisseur à un demandeur sont mis en cache

sur le chemin entre les deux. Dans Gnutella, la récupération de la donnée se fait directement du fournisseur au demandeur en utilisant le protocole HTTP, la donnée ne transite donc pas par le réseau Gnutella. C'est probablement parce que la fonctionnalité de téléchargement ne fait pas partie de la base du protocole que le téléchargement multi-source n'est pas proposé par la version originelle de Gnutella. Les logiciels client-serveur Gnutella envoyant des fichiers intègrent donc un serveur HTTP.

[65] propose de changer la réplication des données en un mode de réplication aléatoire ou le long du chemin de retour des requêtes, afin de limiter le coût de la réplication tout en permettant de satisfaire rapidement un grand nombre de requêtes.

Pare-feux et détection d'utilisation de systèmes pair-à-pair

Dans le cas où le téléchargement de la donnée est impossible, lorsque le nœud qui héberge l'objet est derrière un pare-feu, le demandeur peut router sa demande d'envoi de donnée par le même chemin pris par la requête vers le nœud qui l'héberge. Ce dernier tente alors d'envoyer la donnée vers le demandeur. Dans le cas où les deux nœuds sont derrières des pare-feux, l'échange est impossible.

2.5 Les systèmes décentralisés structurés, une organisation sans chef

Les systèmes décentralisés structurés sont les derniers arrivés des systèmes pair-à-pair. D'origine académique, ils sont apparus en 2001 avec CAN [83] et se proposent d'utiliser la théorie des graphes pour créer des systèmes totalement décentralisés dont le fonctionnement peut être prouvé. Leurs prédécesseurs les plus proches sont les systèmes décentralisés non-structurés dont ils reprennent la philosophie de décentralisation totale. Les travaux effectués sur les systèmes décentralisés structurés ont pour but principal une recherche efficace tout en équilibrant la charge supportée par les nœuds et les liens. Ces systèmes reposent sur le principe que, si d'habitude les services de recherche centralisés sont basés sur des associations *(clé,objet)*, il est possible de décentraliser cette indexation en donnant la responsabilité de chaque clé à un identifiant de nœud du réseau. Cette indexation nécessite alors que chaque nœud du réseau puisse trouver une clé, et donc le nœud qui en est responsable. Pour cela, un système décentralisé structuré impose une organisation des nœuds et de leur connexion à des voisins spécifiques.

Structure des nœuds : les seuls systèmes décentralisés structurés actuellement existants sont des réseaux à contenu adressable, basés sur des tables de hachage réparties (ou DHT pour Dynamic Hash Tables). Les réseaux à contenu adressable sont inspirés de graphes statiques aux caractéristiques bien connues : les hypercubes [88, 96, 105], les papillons [68], les tores [83], les de Bruijn [32, 72, 101], etc. En particulier le routage dans ces systèmes tire souvent directement profit d'un routage qui a été prouvé efficace dans le *graphe statique*, permettant donc l'envoi d'un faible nombre de messages. Ces systèmes se basent donc sur des *graphes orientés* ou les liens entre les nœuds sont sou-

vent des *arêtes*. Pour leur modélisation, on parle de *graphes dynamiques*. Ils utilisent une table de hachage commune à tous les nœuds pour associer à chaque objet et chaque nœud un identifiant, ou *clé*. Ces clés sont équitablement réparties sur un même espace de nommage. Chaque clé est gérée par au moins un nœud, qui gère aussi les objets associés à cette clé. L'envoi d'un message vers une clé permet donc d'atteindre indifféremment un nœud ou un objet.

Acheminement des messages : lorsqu'un nœud envoie un message vers une clé, ce message est alors transmis de nœud en nœud selon un routage ou une méthode proche inspirée du routage dans le graphe statique. Le message arrive ainsi à un nœud responsable de cette clé.

Dans de tels systèmes, chaque nœud qui souhaite partager un objet doit l'annoncer au système. Pour cela, il calcule la clé associée à cet objet $\mathcal{O}$ et envoie un message vers cette clé, contenant la clé et un moyen @ d'y accéder, par exemple son adresse physique. Le nœud responsable reçoit ce message et enregistre dans une table de clés l'association *clé*=@ [1]. Lors d'une recherche de l'objet $\mathcal{O}$, le nœud demandeur doit d'abord calculer la clé associée à $\mathcal{O}$, puis envoyer une requête vers cette clé. Une fois le nœud responsable de la clé atteint, celui-ci renvoie @, qui permet de contacter le nœud qui partage l'objet associé.

La publication est répétée à intervalle réguliers afin que l'objet soit accessible même en cas de départ du nœud responsable. Cette opération de republication doit être suffisamment fréquente pour ne pas rendre l'objet inatteignable en cas de départ du nœud responsable, mais pas trop fréquente pour ne pas surcharger le réseau par le trafic engendré par les publications.

Les systèmes décentralisés structurés tentent donc d'adapter des graphes statiques à l'environnement dynamique et à grande échelle des systèmes pair-à-pair. La structure de ces graphes permet de donner des preuves de leur comportement, en particulier sur leur degré, leur diamètre, la charge de chaque nœud et de chaque arête. On trouve pour ces systèmes trois types de résultats, avec dans un ordre décroissant de force : valeur exacte, avec forte probabilité, ou en moyenne. Ce type de résultats permettent de passer d'affirmations empiriques (utilisées pour les systèmes hybrides et décentralisés) à des garanties probabilistes sur le fonctionnement du système pair-à-pair.

Répartition de la charge : afin de répartir équitablement la charge parmi tous les nœuds dans le réseau, la clé associée à un objet est obtenue par une fonction de hachage donc l'intervalle de sortie correspond à l'espace de nommage. Cet espace de nommage est de taille suffisante pour éviter les collisions, c'est-à-dire que la probabilité que deux objets aient la même clé est négligeable. L'entrée de cette fonction, dans le cas de fichiers, peut être le titre du fichier par exemple. Les fonctions de hachage utilisées permettent de répartir de manière uniforme les objets dans l'espace de nommage. Cela signifie en

[1] Il n'est pas toujours possible de déplacer, d'installer ou de copier l'objet sur le nœud responsable, pour des raisons de taille (gros fichiers) ou de temps (processeurs). C'est pourquoi on considérera dans la suite que lors d'un routage sur une clé associée à un objet, seule la clé associée et un moyen d'accéder à l'objet associé sera renvoyée par le nœud responsable de la clé.

particulier que deux entrées très proches de la fonction de hachage correspondent à des résultats très différents. Dans le cas de données, deux fichiers de noms proches auront des clés très différentes, et auront pour responsables des nœuds très éloignés. Cette propriété permet indirectement de répartir la charge due aux objets populaires sur des nœuds éloignés. Ainsi, en évitant de créer des régions de nœuds responsables d'un sujet ou d'un thème, on évite de surcharger ces régions si le thème devient populaire.

Mise en place : seul le protocole Kademlia est actuellement utilisé à grande échelle. Il a été intégré dans Overnet [78] en juin 2002, puis Overnet a fusionné avec eDonkey [26] pour créer eDonkey2000 [26]. Une autre version de Kademlia a été intégrée dans les logiciels client-serveur eMule [61] sous le nom de «Kad !». Ces deux systèmes supportent jusqu'à un million d'utilisateurs. Enfin, une version de Kademlia a été intégrée dans certains clients le système de téléchargement BitTorrent [18], afin de décentraliser la recherche. Les travaux sur les systèmes décentralisés structurés seront plus particulièrement étudiés dans la partie qui leur est consacrée (chapitre 3.1).

2.5.1 Comportement des systèmes décentralisés structurés

Gérer de nombreux utilisateurs

Les systèmes décentralisés structurés ont pour but d'équilibrer le mieux possible la responsabilité des objets entre tous les nœuds. Ainsi, tous ont la même importance dans le réseau et, en cas de déconnexion brutale d'un nœud, le réseau ne perd que les objets hébergés par ce nœud. Si les clés et les identifiants des nœuds sont répartis correctement, les nœuds seront responsables d'un espace de clé à peu près de même taille, qui contiendra ainsi autant de clés publiées. Tous les nœuds seront donc responsables du même nombre d'objets. C'est pourquoi sont utilisées des fonctions de hachage, qui ont pour but de répartir les sorties de manière pseudo-aléatoires, c'est-à-dire qu'il est très difficile de déduire de la sortie quelle était l'entrée.

De même, les topologies utilisées permettent en général de limiter le nombre de messages nécessaires pour localiser un objet, et de répartir les trajets pris par les messages entre toutes les arêtes. Ainsi, le routage est équilibré entre tous les nœuds. On remarque toutefois qu'il est difficile aux réseaux à contenu adressable de répartir les objets selon leur popularité. Ainsi, certains objets populaires chargeront des nœuds plus que d'autres, car les demandes pour celles-ci seront plus nombreuses. Une méthode proposée dans [36] est décrite au chapitre 3.4.8.

Tolérance aux pannes et charge du réseau

Dans les réseaux à contenu adressable, les nœuds sont connectés à un certain nombre de voisins. Lorsque les pannes de nœuds sont aléatoires, un système avec un degré important peut résister plus facilement aux déconnexions. Le degré peut varier selon l'arrivée et le départ des nœuds :

- rester constant pour le degré entrant [68] ;
- varier mais rester constant en moyenne [32] ;

- être logarithmique en moyenne [32] ;
- ou logarithmique avec forte probabilité [88, 96, 105].

Dans certains systèmes, le degré dépend d'un paramètre, comme le nombre de dimensions dans CAN [83]. Ce degré peut aussi varier avec le logarithme du nombre de nœuds dans le réseau, comme pour Chord [96] (en optimisant pour ne pas garder de doublons parmi les voisins d'un nœud), D2B [32] ou Broose [101]. Les systèmes à degré logarithmique résistent par nature plus facilement à des pannes aléatoires que des systèmes à degré constant. Cela permet à Chord, D2B, et Broose de rester suffisamment connectés sans pour autant avoir trop de voisins, ce qui augmenterait inutilement le trafic de contrôle. Dans CAN, le nombre de dimensions peut donc améliorer la tolérance aux pannes s'il induit un degré approximativement $\log n$, où n est le nombre de nœuds maximal dans le réseau. Toutefois, l'estimation de $\log n$ n'est pas triviale à effectuer dans un réseau dynamique.

Enfin, certains systèmes (comme [31]) ont été proposés pour résister à la censure, et résister à l'élimination d'une fraction linéaire de nœuds du réseau. [31] nécessite en particulier un degré polylogarithmique.

La plupart des réseaux à contenu adressable ne traitent que les pannes d'arrêt, comme pour les autres systèmes pair-à-pair. La panne d'un nœud dans un réseau à contenu adressable a pour effet la disparition des objets que le nœud hébergeait (ces objets sont parfois répliqués comme nous le verrons plus loin). Concernant les objets dont était responsable le nœud partant, les informations de routage peuvent être remplacées par republication régulière. C'est alors un voisin du nœud défaillant qui le remplacera. Idéalement, en cas de pannes aléatoires de nœuds, les voisins restent connectés et il ne leur reste qu'à trouver un nœud qui va remplacer leur voisin défaillant. Ce nœud est dépendant de la topologie, et son identifiant peut être trouvé de manière déterministe.

Notons le travail [67] qui s'est attaqué au problème posé par le comportement des réseaux à contenu adressable en cas d'événements simultanés : arrivées de nœuds, départs de nœuds, et arrivées de messages.

Autonomie et envoi à des sous-réseaux

Parmi les systèmes décentralisés structurés étudiés dans cette thèse, aucun ne permet à un nœud d'interdire l'envoi de message à un nœud ou de ne pas utiliser des arêtes données, ni d'obliger les messages à passer par certains nœuds ou certaines arêtes.

Toutefois, des travaux [8, 54] capables d'effectuer des requêtes par intervalles et ordonnant les nœuds par noms de domaines permettent une localité des chemins, c'est-à-dire que lorsque deux nœuds sont dans un même domaine, un message envoyé de l'un à l'autre ne sortira pas de ce domaine.

Accès au réseau

Le problème du premier accès d'un nœud au réseau n'est pas résolu par les systèmes décentralisés structurés, qui se focalisent, comme on l'a vu, sur le routage. On suppose donc qu'un nœud cherchant à se connecter au système connaît un nœud déjà connecté

au réseau.

Protection des utilisateurs

Les systèmes décentralisés structurés ne tentent généralement pas d'assurer l'anonymat. Cependant, certains systèmes comme [31] permettent d'assurer la continuité du service lorsque, par exemple, un adversaire peut éliminer une fraction linéaire de nœuds du réseau.

Dynamisme des nœuds

Lorsqu'un nœud s'insère dans un réseau à contenu adressable, il trouve ses voisins, et il informe les nœuds qui doivent le prendre comme voisin. Il faut noter que le nombre de nœuds à contacter et de messages à envoyer pour se connecter peut donc être supérieur au degré du nœud à la fin de sa connexion. En effet, les réseaux à contenu adressable utilisent parfois des graphes orientés, donc un nœud X voisin d'un nœud Y ne signifie pas forcément que Y est voisin de X. Ainsi, Chord [96] a un degré $O(\log n)$ mais nécessite un nombre de messages $O(\log^2 n)$ pour gérer l'arrivée et le départ d'un nœud.

Réactivité des nœuds

La vivacité est directement liée au degré de chaque nœud. Puisque les nœuds ont en moyenne le même degré, la vivacité est la même en moyenne pour tous les nœuds.

Rapidité de traitement des requêtes

Dans les réseaux à contenu adressable, il convient de minimiser le diamètre du réseau puisque le routage est le principal objectif de ceux-ci. Le diamètre est polynomial pour CAN, tandis que des systèmes comme D2B, Chord, Tapestry et Pastry permettent un diamètre logarithmique, avec des différences qui seront décrites dans le chapitre suivant.

Expressivité des requêtes

Les réseaux à contenu adressable utilisent le hachage, ils ne permettent donc *a priori* que les recherches par mots-clés, ou exacts. Des systèmes comme [8, 54] proposent des recherches par intervalles.

Exhaustivité des réponses

Un système structuré décentralisé donne la responsabilité d'un objet à un seul nœud. Il suffit donc de trouver ce nœud pour trouver toutes les occurrences d'un même objet dans le système. Cependant, il faut noter que selon la façon dont la clé est attribuée à un objet, plusieurs clés peuvent être associées à un même contenu. Dans ce cas, une seule requête ne suffit plus à trouver tous les objets correspondants dans le réseau. Par exemple, si un fichier est nommée d'après son titre, deux fichiers peuvent contenir le même texte et avoir un titre différent. Toutefois, l'utilisation des réseaux à contenu

adressable s'est avérée particulièrement efficace lorsque les clés sont attribuées par hachage du contenu des fichiers puisque cela permet de trouver toutes les occurrences d'un même contenu, quelque soit le nom du fichier. Cette clé peut alors être trouvée en cherchant par exemple les clés associées à des mots-clés.

Authentification des objets

Ce problème n'est pas abordé dans les réseaux à contenu adressable puisque la priorité de ces systèmes est un routage efficace et l'équilibre de la charge.

Accessibilité des objets et «consommateurs égoïstes»

Chaque nœud est responsable d'objets dans les systèmes décentralisés structurés, donc même si un nœud ne propose pas d'objets au système, il participe au service en fournissant aux nœuds recherchant un objet l'adresse physique du nœud qui héberge cet objet.

Réplication des objets

Selon les systèmes décentralisés structurés, une réplication active est parfois effectuée. Ainsi, dans Pastry [88], chaque association *clé,nœuds* faisant le lien entre un objet et un nœud proposant cet objet est copiée sur plusieurs nœuds proches du responsable. Cela permet que l'objet reste accessible dans le cas où le responsable de sa clé quitte le réseau. Selon un principe similaire, les systèmes décentralisés structurés souples comme Kademlia et Broose maintiennent des voisins redondants afin de palier à des disparitions soudaines.

Pare-feux et détection d'utilisation de systèmes pair-à-pair

Les réseaux à contenu adressable étant totalement répartis, ils rendent difficile le blocage du trafic basé sur l'interdiction d'adresses physiques. En effet, aucun serveur n'est nécessaire au fonctionnement du service et un nœud du réseau peut avoir n'importe quelle adresse physique. Afin d'éviter l'analyse de trafic, il est aussi possible d'encoder les messages et l'envoi de données en HTTP comme le font d'autre protocoles (Gnutella, etc.) pour tromper un pare-feu. Ces fonctionnalités ne dépendent cependant pas du routage lui-même et ne sont donc pas considérées dans les travaux sur les réseaux à contenu adressable.

Deuxième partie

Les systèmes décentralisés structurés

Chapitre 3

État de l'art des systèmes décentralisés structurés

Dans le chapitre précédent, nous avons brièvement abordé le sujet des tables de hachage réparties et de leur utilisation pour la conception de système pair-à-pair. Nous allons ici détailler plus avant les travaux qui ont été effectués sur ce sujet ainsi que leurs spécificités.

Les systèmes décentralisés structurés ont deux avantages principaux sur leurs concurrents :

- ils permettent la recherche d'objets dans un réseau avec un faible nombre de messages, souvent au moyen d'un routage efficace ;
- des preuves formelles de leur efficacité et de leur comportement statistique peuvent être données (en moyenne ou avec forte probabilité[1]). En effet, aucun modèle déterministe ne permet actuellement de décrire convenablement le comportement des nœuds dans un système pair-à-pair général. Il n'a donc pour l'instant pas été possible de prouver ces systèmes de manière déterministe.

Les différentes propriétés telles que le degré ou le diamètre seront prouvées en fonction du nombre de nœuds présents dans le réseau n ou du nombre de nœuds maximum dans le réseau N.

Les systèmes décentralisés structurés sont aussi appelés réseaux à contenu adressable, nous allons maintenant détailler leur fonctionnement.

3.1 Fonctionnement d'un réseau à contenu adressable

Un réseau à contenu adressable permet d'envoyer des messages à travers un réseau, à destination d'un contenu, et ce sans avoir à connaître le nœud qui héberge ce contenu. Les fondements de ces systèmes ont été posés par CAN [83], qui les a baptisé «réseaux à contenu adressable» («Content-Adressable Network»). Les systèmes décentralisés structurés supportent la variation du nombre de nœuds de manière transparente, que leurs

[1] Dans la suite, nous dirons qu'un événement a lieu avec forte probabilité lorsqu'il a lieu avec une probabilité $1 - o(1)$, avec n le nombre de nœuds dans le système

propriétés dépendent du nombre de nœuds dans le réseau ou du nombre de nœuds maximal dans le réseau. On considère dans ce chapitre qu'un utilisateur a le moyen de trouver une (ou plusieurs) clé(s) associée(s) à un objet recherché. Ce sera cette clé associée à un contenu qui sera adressable dans les systèmes que nous allons décrire ici.

Les systèmes pair-à-pair décentralisés structurés sont composés pour cela de deux éléments :

- un réseau logique permettant l'arrivée et le départ des nœuds, et le routage d'un nœud à un autre ;
- une table de hachage répartie, permettant la publication et la recherche de clés.

La localisation de clés s'effectue de manière répartie dans le réseau des nœuds connectés, chaque clé étant associée à un nœud. Nous ne détaillerons pas ici la façon d'obtenir la clé. Notons simplement qu'il peut s'agir par exemple du hachage du nom de l'objet ou, dans le cas d'un système de partage de fichiers, du hachage de son contenu (le hachage permet de répartir uniformément les clés sur les identifiants des nœuds). Remarquons que dans le cas où un objet n'a pas de nom ou de référence unique connu de manière universelle, il est possible que plusieurs clés soient attribuées au même objet. Toutefois, cette solution n'est pas souhaitable car la charge du nombre de clés par nœud doit être limitée, et il faut éviter les collisions de clés (même clé associée à plusieurs objets).

Chaque nœud est responsable d'une partie des clés présentes dans le système et connaît les adresses de quelques nœuds adjacents nécessaires au routage des messages dans le réseau. Il est alors possible, à partir de n'importe quel nœud effectuant une requête vers une clé dans le système, de router une requête jusqu'à un nœud associé à cette clé, qu'il s'agisse d'une requête d'insertion, de suppression, ou de recherche. Cela impose au système de maintenir une structure permettant l'envoi de messages, c'est-à-dire que chaque nœud tienne à jour un certain nombre de voisins. Lors de l'envoi d'un message dans le réseau, cette structure permet de se rapprocher du destinataire (dans le réseau logique) à chaque saut. Pour cela, chaque nœud peut être amené à jouer tour à tour le rôle de :

- client d'un message, lorsque ce nœud recherche une clé ;
- routeur lorsqu'il reçoit un message à transmettre ;
- serveur lorsqu'il reçoit un message de recherche pour une clé dont il responsable.

3.1.1 Structure du réseau

La structure imposée à ces systèmes s'inspire de graphes bien connus, comme les tores, les hypercubes, les papillons, les de Bruijn, etc. Ces graphes proposent en effet des propriétés intéressantes comme un diamètre faible, un degré constant ou faible, et un routage utilisant tous les nœuds de manière équitable. La topologie impose donc une orientation des liens lorsque les graphes dont ils s'inspirent sont orientés. Le maintien de la structure permet de bénéficier d'un routage décentralisé efficace, la contrepartie étant la contrainte pour les nœuds de devoir choisir leurs voisins selon leur identifiant, pour maintenir cette structure. Dans les systèmes non orientés, chaque nœud peut inciter ses voisins à faire suivre les requêtes qu'il lui envoie de manière simple. En effet, si un nœud u ne transmet pas les messages d'un voisin v, il est possible à v de rétorquer

en arrêtant de transmettre les messages qu'il reçoit de u. Les nœuds égoïstes reçoivent ainsi de moins en moins de messages de leurs voisins et sont donc progressivement déconnectés du réseau.

Pour formaliser, dans ces réseaux à contenu adressable, on associé l'espace de clés $\mathcal{C}$ à un graphe $G = (V, E)$. Chaque nœud $v \in V$ est responsable d'un espace de clés C_v selon son identifiant, tel que $\cup_{v \in V} C_v = \mathcal{C}$. Cette attribution de responsabilité peut se faire par la technique de hachage cohérent [58] qui permet d'équilibrer la charge du nombre de clés par nœud. Cette technique propose d'associer chaque clé à un nœud responsable. Pour chaque clé, un arbre aléatoire k-aire couvrant dans le réseau logique est enraciné au nœud responsable de la clé. Cet arbre permet le routage à partir d'un nœud quelconque dans l'arbre vers la clé, donc vers le nœud qui en est responsable. Les arbres aléatoires sont obtenus en utilisant $O(\log n)$ fonctions de hachage afin de répartir la charge du routage équitablement parmi les nœuds participants. Dans le cas d'un arbre unique, la racine et ses descendants proches supporteraient en effet une charge trop importante. Dans la suite, nous verrons que pour chaque système, ces arbres aléatoires seront obtenus de manière similaire mais avec une seule fonction de hachage : il s'agira d'assigner aux nœuds des identifiants unique dans le réseau puis de former l'arbre en combinant les chemins allant de chaque nœud du réseau au nœud racine, ces chemins étant obtenus par le routage du graphe utilisé par le système.

3.1.2 Routage

L'espace de clés est le même que l'espace des identifiants des nœuds. Router vers une clé revient donc à router vers le nœud responsable de cette clé (nous verrons que plusieurs nœuds peuvent être responsables de la même clé dans certains systèmes). C'est pourquoi dans ces réseaux, router vers une clé ou un nœud est équivalent. Nous avons expliqué que les réseaux à contenu adressable maintiennent une structure inspirée d'une topologie de graphe. Le routage de ces systèmes est ainsi une adaptation du routage dans la topologie de graphe utilisée. Ce routage permet ainsi à chaque nœud recevant un message de décider localement à quel voisin faire suivre le message. Le routage R est donc une application de $V \times \mathcal{C}$ dans V, avec la contrainte pour un nœud v_o qui envoie un message que le nœud destinataire $R(v_o, \gamma) = v_i$ est :

- soit le nœud origine v_o ;
- soit un voisin v_v du nœud v_o, c'est-à-dire que le message doit être transmis le long de l'arête (v_o, v_v).

Des topologies simples comme l'anneau sont écartées malgré la simplicité de leur mise à jour à cause du trop important nombre de sauts nécessaire pour le routage. La clique $\mathcal{K}_n$ impose un degré trop important pour chaque nœud : le trafic de contrôle résultant n'est pas réaliste pour un système pair-à-pair réactif. De la même façon, l'arbre binaire demande à la racine et à ses proches descendants de supporter une charge trop importante pour que cette topologie soit intéressante pour un système pair-à-pair qui équilibre la charge entre tous les nœuds. Dans la suite, on dira que deux nœuds sont proches s'ils sont proches dans le réseau logique, puisque le routage se fait selon les identifiants des nœuds.

3.1.3 Publication

Dans un réseau à contenu adressable, chaque nœud v maintient une table de clés contenant pour chaque entrée une clé $\gamma \in C_v$ associée à des informations concernant l'objet correspondant. Ces informations permettent entre autres d'accéder à cet objet, soit en donnant accès directement à l'objet, soit en précisant l'adresse physique du nœud qui partage cet objet. La publication d'une clé permettant la récupération de l'objet associé, elle se révèle à chaque requête plus rentable que l'inondation par exemple, en terme de nombre de messages reçus par nœud. Lorsqu'un nœud met à disposition un objet, il doit d'abord l'annoncer au système : on parle de *publication*. Alors seulement l'objet est disponible aux utilisateurs du système. La publication consiste, pour un nœud source qui annonce un objet $\mathcal{O}$, à calculer la clé γ associée à l'objet $\mathcal{O}$, puis à envoyer un message au nœud u responsable de la clé γ *via* le réseau logique. Ce message contient le nécessaire pour que le nœud u puisse accéder à l'objet : soit l'objet lui-même soit l'adresse physique du nœud et un moyen d'identifier l'objet. Lorsque le nœud u responsable de la clé γ reçoit le message de publication, il associe γ aux informations permettant d'accéder à l'objet $\mathcal{O}$. Ainsi, dans un système basé sur une table de hachage répartie, lorsqu'un nœud cherche un objet auquel est associé une clé $\gamma \in \mathcal{C}$, il cherche donc directement la clé γ, il n'est donc pas nécessaire de connaître l'identifiant du nœud responsable de l'objet pour chercher cet objet. Il serait en effet contraignant de devoir vérifier, avant d'envoyer un message, quel est le nœud le plus proche d'une clé qui est connecté au système. Une fois trouvé le nœud responsable, ce dernier répond au nœud demandeur et envoie les éventuelles informations y donnant accès.

3.1.4 Équilibrage de la charge des requêtes parmi les nœuds

L'un des buts des systèmes décentralisés étant de répartir parmi tous les nœuds la charge induite par les requêtes, il est impossible d'utiliser des structures comme les arbres k-aires, du fait de la lourde charge supportée par la racine.

Nous avons vu que la technique des *arbres aléatoires* [58] utilise des arbres différents dans les réseaux à contenu adressable pour le routage et l'équilibrage de charge. Elle consiste à associer un arbre couvrant à chaque chaque clé du système. Chaque nœud de cet arbre fonctionne alors comme un cache pour la clé. Chaque arbre est donc enraciné sur le responsable de la clé correspondante, et les identifiants des nœuds le formant sont alors obtenues aléatoirement par hachage, à partir d'informations propres à chaque nœud.

La contribution [58] propose aussi l'utilisation d'une seconde technique : *le hachage cohérent*. Là où un hachage habituel permettrait de répartir la charge parmi un nombre fixe de nœuds, le hachage cohérent permet une répartition de la charge malgré l'arrivée et le départ d'un certain nombre de nœuds dans le réseau, et ce quelque soit l'ordre d'arrivée et de départ des nœuds. En effet, concernant l'attribution des clés aux nœuds, ce hachage présente la propriété d'engendrer peu de changements lorsque la taille de l'ensemble de sortie de la fonction de hachage augmente.

Les arbres aléatoires proposés dans [58], et donc le trajet jusqu'à la racine, peuvent

avantageusement être spécifiés par un graphe plutôt que d'être attaché à chaque message. Le choix de ce graphe dont s'inspire un réseau à contenu adressable est donc important pour une bonne répartition du routage sur les nœuds. Dans la suite, nous calculerons la charge d'un nœud due aux requêtes qu'il devra retransmettre en calculant le nombre de requêtes pouvant passer par ce nœud divisé par le nombre total de paires (consommateur,clé) possibles $n|\mathcal{C}|$ (rappelons que n est le nombre de nœuds dans le réseau et $\mathcal{C}$ l'espace de clés du système. Nous nommerons cette charge *engorgement* d'un nœud.

3.1.5 Équilibrage de la charge des clés et des objets

Dans les systèmes pair-à-pair, il est nécessaire d'éviter que des nœuds ne soient surchargés par les requêtes à destination de leurs clés, pour qu'ils ne deviennent pas des points faibles du système. Afin de répartir la gestion des clés parmi tous les nœuds, le nombre de clés qui sont attribués à chaque nœud peut varier selon des paramètres tels que l'identifiant des voisins, le nombre de voisins, ou une estimation du nombre de nœuds dans le réseau. La répartition des clés peut se faire de manière équilibrée entre tous les nœuds grâce par une table de hachage répartie, sans recourir à des serveurs [82]. Afin d'éviter que trop de clés ne soient associées au même nœud, il est nécessaire d'avoir un espace de clés et d'identifiants de taille suffisante pour que peu de clés différentes soient associées à un même nœud (collision de clés). Comme vu précédemment, la technique du hachage cohérent permet l'attribution dynamique des clés aux nœuds malgré l'arrivée et le départ des nœuds dans le réseau.

Il existe des systèmes où les objets ne sont accessibles que sur les nœuds qui les proposent au réseau. Ces objets ne sont donc pas déplacés sur le nœud responsable de la clé associée à l'objet. Dans ce cas, il est possible d'équilibrer l'hébergement des clés associées à ces objets (par la technique de hachage cohérent associée aux arbres aléatoires), mais pas l'hébergement des objets lui-même. De plus, dans le cas d'un objet qui est plus populaire que les autres, la clé qui lui est associée peut être recherchée par beaucoup de nœuds. Les nœuds responsables de cette clé peuvent être les destinations d'un grand nombre de messages. Une solution à ce problème et proposée par des systèmes qui proposent une réplication active des données ou une mise en cache (voir chapitre 3.4.8). Cela permet de multiplier les sources potentielles, de limiter l'engorgement et de diminuer la charge de chaque nœud. Cependant, les systèmes pair-à-pairs étant à grande échelle, ces solutions ne peuvent être réellement efficaces que si elles s'adaptent au nombre de nœuds présents dans le réseau.

3.1.6 Arrivée et départ du réseau

Lorsqu'un nœud se connecte au système, il doit obtenir un identifiant tiré aléatoirement, choisi, ou donné (par le hachage d'informations personnelles uniques, comme l'adresse physique par exemple). Cet identifiant lui permet alors de contacter le nœud u responsable des clés qu'il va devoir gérer, ainsi que les nœuds auxquels il va devoir se connecter. Une fois contacté, ce nœud responsable u divise alors sa zone de responsa-

bilité en deux parties, idéalement équilibrées. Il laisse alors la responsabilité d'une des deux zones obtenues au nouveau nœud. Les deux nœuds protagonistes changent alors éventuellement leurs identifiants, par exemple en en augmentant la taille ou en les modifiant de façon a ce que leurs identifiants respectifs soient toujours cohérents avec leur zone de responsabilité. Notons que les voisins du nouveau nœud peuvent être des voisins de nœud u, ce qui diminue alors le nombre de messages nécessaires à la connexion du nouveau nœud.

Afin d'assurer le dynamisme du système, les nœuds doivent pouvoir s'insérer et quitter le réseau rapidement. Un faible degré est donc essentiel pour limiter le nombre de messages nécessaires à la découverte des nouveaux voisins lors de l'arrivée dans le système. Le degré d'un nœud détermine aussi le nombre de messages qui devront être envoyés pour vérifier que ce nœud est toujours connecté à ses voisins. Le degré d'un réseau à contenu adressable indique donc l'importance du trafic de contrôle qui va circuler dans le réseau.

De manière similaire à l'arrivée d'un nœud, lorsqu'un nœud quitte le système, soit il doit prévenir ses voisins et transférer les clés dont il est responsable, soit chaque nœud doit vérifier fréquemment que ses voisins sont toujours connectés. Souvent, les deux méthodes sont combinées. Afin de diminuer le nombre de messages nécessaires au départ d'un nœud, le nœud qui reprend la responsabilité de ces clés est souvent un des voisins du nœud partant.

3.2 État de l'art des réseaux à contenu adressable

Nous allons maintenant décrire plusieurs travaux qui ont permis la naissance puis l'amélioration des systèmes décentralisés structurés. Nous verrons pour chacun la structure utilisée comme base, et la façon dont est géré le réseau, en particulier :

- le routage ;
- l'insertion de nœuds ;
- la publication de clés ;
- la recherche de clés.

Cette description sera accompagnée des résultats obtenus par chaque protocole concernant ses propriétés. Tout au long de ce chapitre, n désignera le nombre courant de nœuds dans le réseau tandis que N sera le nombre maximal de nœuds dans le réseau. Nous constaterons en particulier que les graphes utilisés pour l'interconnexion des systèmes décentralisés structurés ont suivi une évolution similaire aux architectures parallèles il y a quelques dizaines d'années, une partie de la problématique actuelle étant en effet la même. L'expérience des architectures parallèles a donc été réutilisée avec succès pour les réseaux à contenu adressable. Enfin, bien qu'un grand nombre de ces protocoles utilisent des optimisations et les intègrent à leur évaluation, celles-ci sont souvent utilisables pour tous les réseaux à contenu adressable, ce qui rend difficile l'évaluation des qualités intrinsèques du protocole. Nous terminerons donc ce chapitre en détaillant les optimisations dont l'apport peut servir à tous les réseaux à contenu adressable.

3.2.1 CAN

CAN [83] est la contribution qui baptisa les réseaux à contenu adressable, elle énoncé aussi les grands principes de ces systèmes. Publié dans le cadre d'un travail à AT&T en 2001, cet article pose les bases nécessaires à la création de réseau à contenu adressable et propose un exemple de réseau à contenu adressable afin d'en montrer le fonctionnement. Il fut publié la même année que Chord [96], qui s'est lui plus attaché à l'obtention de bornes efficaces et moins aux fondements et aux caractéristiques générales de ces systèmes. La proposition CAN se termine par une liste d'optimisations possibles pour les réseaux à contenu adressable. Des simulations sur un nombre de nœuds allant jusqu'à 260.000 confirment les bornes moyennes présentées pour l'exemple de topologie qui est présenté. Nous présentons ici l'exemple de la topologie. Tous les calculs de bornes sont effectués en moyenne, en considérant des nœuds répartis de manière équitable parmi l'espace des clés.

Structure logique

CAN propose un exemple basé sur une topologie de *tore à β dimensions* où les connexions sont non orientées. Cette topologie permet une représentation simple et intuitive de la structure du réseau tout en utilisant les distances euclidiennes pour estimer la distance entre deux nœuds. Tout nœud a un identifiant composé d'un intervalle pour chacune des β dimensions, chaque intervalle étant inclus dans un intervalle $[0, 1]$. On appelle zone de responsabilité C_v d'un nœud v le produit cartésien des intervalles de chaque dimension $C_v = I_1 \times \ldots \times I_\beta$, $I_i \subseteq [0, 1]$.

Le routage de CAN s'effectue de proche en proche : à chaque saut, on ne peut changer de coordonnées que sur une dimension. Si plusieurs voisins existent, alors on choisit celui dont l'intervalle est le plus proche de la destination.

Insertion des nœuds dans le réseau

Lorsqu'un nœud veut entrer dans le système, il tire aléatoirement uniformément un identifiant sur $[0, 1]^\beta$. Il utilise alors une passerelle pour envoyer un message au nœud contenant ces coordonnées. Ce dernier divise alors sa zone de responsabilité en deux parties égales, et en attribue une au nouveau nœud, puis les voisins des deux nœuds sont mis à jour.

Dans la figure 3.1, on voit un exemple d'insertion dans un réseau CAN pour $d = 2$:

- après le tirage de coordonnées $(2/3, 2/3)$ par un nouveau nœud u,
- il envoie à sa passerelle $[0, 1/2[\times[0, 1/2[$ une demande d'insertion aux coordonnées $(2/3, 2/3)$ (a). Cette passerelle route alors le message vers le nœud responsable de l'identité $(2/3, 2/3)$, qui est $[1/2, 3/4[\times[1/2, 1]$;
- ce nœud partage alors sa zone de responsabilité en deux zones de taille égale (b), et le nouveau nœud va s'insérer à cet endroit. $[1/2, 3/4[\times[1/2, 1]$ prend l'identité $[1/2, 3/4[\times[3/4, 1]$ tandis que u prend l'identifiant $[1/2, 3/4[\times[1/2, 3/4[$ et récupère les clés correspondant aux intervalles de son identifiant.

Connexion entre les nœuds

Les voisins d'un nœud $u = I_1 \times \ldots \times I_\beta$ sont les nœuds $v = J_1 \times \ldots \times J_\beta \Leftrightarrow \exists j \in [0, \beta] I_j \cap J_j = \emptyset, \forall k \neq j I_k \cap J_k \neq \emptyset$. Ce sont les nœuds qui partagent une frontière commune avec u dans le tore. Si les nœuds sont répartis équitablement sur l'espace des clés, un nœud u est connecté en moyenne à deux voisins dans chacune des β dimensions du système : un dont l'intervalle précède l'intervalle de u et un dont l'intervalle le suit. Le *degré moyen* est donc constant et égal à 2β. C'est parce qu'en pratique, les nœuds ne sont pas toujours équitablement répartis qu'il est possible qu'une zone adjacente à un nœud soit gérée par plusieurs voisins qui s'en sont partagé la responsabilité.

Publication des clés

Toute clé a un identifiant défini sur $[0, 1]^\beta$, comme les identités des nœuds. Lorsqu'une clé est publiée par un nœud, un message est envoyé au nœud dont les d intervalles (pour chaque dimension) contiennent les d coordonnées de la clé.

Routage

Lorsqu'un nœud doit envoyer un message à un autre nœud, il le transmet au nœud dont l'identifiant est le plus proche de la destination selon la distance de Manhattan. Il a pour cela le choix entre au moins 2 voisins dans chacune des β dimensions. Dans un réseau à n nœuds et β dimensions, chaque dimension contient en moyenne de $n^{1/\beta}$ nœuds. Un message routé à travers le réseau sera donc *au plus* à distance $n^{1/\beta}/2$ dans chacune des β dimensions, tandis qu'en moyenne, il sera à une distance de $n^{1/beta}/4$. Le *nombre de sauts moyen* nécessaires à l'acheminement d'un message vers une destination quelconque est donc $(\beta/4)(n^{1/\beta}) = O(\beta n^{1/\beta})$. La figure 3.1(a) montre le chemin parcouru lors du routage depuis le nœud $[0, 1/2[\times[0, 1/2[$ vers une clé $(2/3, 2/3)$ gérée par le nœud $[1/2, 3/4[\times[1/2, 1]$, *via* le nœud $[1/2, 1] \times [0, 1/2[$.

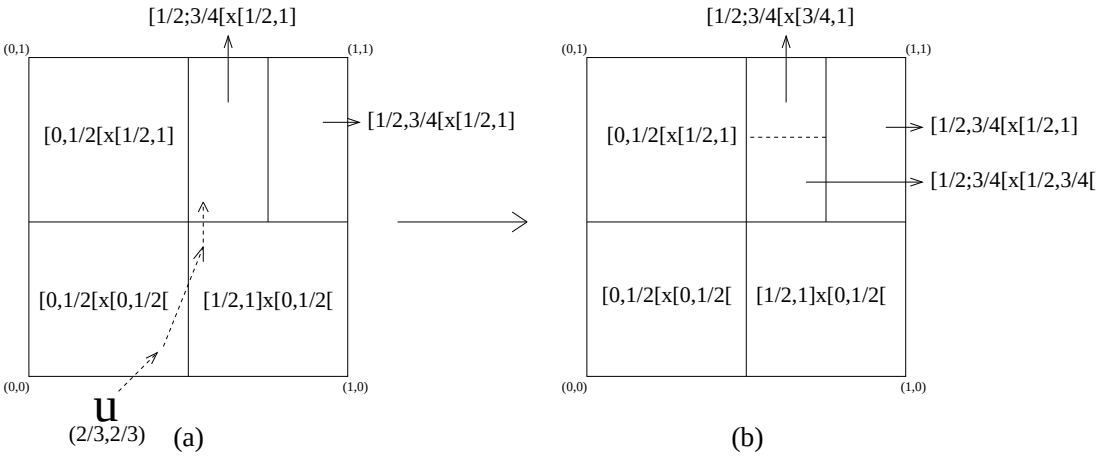

FIG. 3.1 – Routage (a) et insertion (b) dans CAN.

Départ de nœuds du réseau

Aucune opération n'est prévue pour le départ des nœuds. La republication régulière des objets doit suffire à maintenir les clés associées accessibles. Afin de remplacer plus rapidement les nœuds disparus, chaque nœud envoie régulièrement les coordonnées de son espace de clés, accompagnées de la liste de ses voisins accompagnés de leurs coordonnées. Ainsi, lorsqu'un nœud disparaît, chacun de ses voisins démarre un compte-à-rebours initialisé à un temps dépendant de sa taille. Une fois son compte-à-rebours arrivé à 0, un nœud envoie un message de récupération à tous les voisins du nœud disparu, accompagné de la taille de sa zone. Lorsqu'un nœud reçoit un message de récupération d'un nœud dont l'espace de clés est plus petit que le sien, il annule son compte-à-rebours. Sinon, il renvoie à ce nœud un message de récupération avec la taille de son propre espace de clés. Le nœud responsable du plus petit espace de clés récupère ainsi la responsabilité de l'espace de clés qui était géré par le nœud disparu.

Afin d'éviter de fragmenter l'espace de clés, un algorithme permet de choisir un remplaçant pour un nœud disparu. Il est basé sur la représentation de l'espace de clés par un arbre, où les feuilles sont des espaces de clés gérés par des nœuds du réseau, tandis que les nœuds internes sont des espaces de clés qui ont été divisés (une ou plusieurs fois) pour être géré par plusieurs nœuds du réseau. Lorsqu'un nœud u quitte le réseau, un remplaçant temporaire v est trouvé comme expliqué précédemment. v effectue alors une recherche en profondeur dans le sous-arbre de l'espace de clés enraciné au nœud (feuille ou nœud interne) voisin du nœud disparu. Cette recherche s'arrête lorsque deux feuilles x et y ayant le même père sont trouvées. x prend alors la responsabilité de l'espace de clés de y tandis que y remplace le nœud disparu x. Cette recherche peut s'effectuer de manière locale. Il suffit pour cela de suivre le voisin qui a été promu responsable de l'espace de clés voisin lors du dernier partage. Si ce voisin est unique, alors un couple de feuille a été trouvé. Sinon, l'exploration continue récursivement avec les nœuds responsables de cette espace de clés.

Divers

CAN propose un grand nombre d'optimisations, dont l'effet est évalué et comparé entre elles :

- augmenter le nombre de dimensions (voir chapitre 3.4.1) ;
- maintenir plusieurs réseaux logiques (voir chapitre 3.4.2) ;
- tenir compte de la distance physique à chaque saut pour envoyer le message au voisin le plus proche physiquement (voir chapitre 3.4.3) ;
- mutualiser la responsabilité d'une même zone entre plusieurs nœuds (voir chapitre 3.4.4) ;
- utiliser plusieurs fonctions de hachage pour publier et localiser les clés (voir chapitre 3.4.5) ;
- influencer la topologie logique de la topologie physique (voir chapitre 3.4.7) ;
- équilibrer la charge des nœuds à leur arrivée en choisissant un voisin du point d'insertion si cela peut améliorer la charge 3.4.6 ;
- mettre en cache et répliquer les clés (voir chapitre 3.4.8).

Dans la table suivante, β est le nombre de dimensions du tore utilisé par CAN.

CAN [83]	en moyenne
diamètre	$O(\beta n^{1/\beta})$
degré	$O(\beta)$
insertion(nombre de messages)	$O(\beta n^{1/\beta})$
engorgement	$O(\beta n^{1/\beta-1})$

3.2.2 Chord

Proposé en 2001 comme CAN [83], Chord [96, 97] aborde le problème des réseaux à contenu adressable de manière moins généraliste. Il s'appesantit sur l'obtention de bornes efficaces obtenues avec forte probabilité pour le degré, le routage, et l'insertion dans le réseau. Ce protocole utilise le hachage cohérent [58] mais propose l'utilisation d'une seule fonction de hachage (SHA-1), afin rendre le comportement du système déterministe. L'évaluation de ce protocole est effectuée au moyen de résultats avec forte probabilité, de simulations sur 10^5 à 10^9 nœuds et d'un déploiement sur une plate-forme d'expérimentation de 180 nœuds.

Structure logique

Chord construit un réseau qui est en fait très proche de la topologie de l'hypercube où les connexions sont orientées. Il utilise le hachage cohérent afin d'éviter de déplacer trop de clés à l'arrivée d'un nouveau nœud. La topologie de Chord est représentée sous forme d'un anneau représentant l'espace de nommage $[0, 2^\ell[$ bouclant en ses extrémités, avec $\ell = \log N$ le nombre de bits d'une adresse logique. C'est pourquoi nous donnerons des résultats de calculs modulo 2^ℓ pour ce protocole. Le nœud dont l'identifiant est immédiatement supérieur (respectivement inférieur) à x sera nommé $succ(x)$ (respectivement $pred(x)$). Chaque nœud est connecté aux nœuds $succ(x)$ et $pred(x)$, formant ainsi un anneau. En plus de ces voisins sur l'anneau, un nœud x est connecté à jusqu'à $\ell = \log_2 N$ nœuds différents. La table de routage est donc de taille $\lceil \log N \rceil + 2$. Dans ce système, chaque clé c est associée au nœud d'identifiant immédiatement supérieur à c.

Connexion entre les nœuds

Chaque nœud x de Chord a au plus $\ell = \log_2 N$ voisins auxquels s'ajoute $pred(x)$. Le $i^{ème}$ voisin de x est alors $succ(x+2^i)$, $i \in [0, \ell]$. Si les nœuds sont répartis uniformément, alors des voisins successifs dans la table de routage d'un nœud peuvent être les mêmes. Le nombre de voisins différents est alors $\Delta = O(\log n)$. Pour des raisons de redondance, chaque nœud va aussi retenir les α plus proches successeurs de chacun de ses voisins, α étant une constante pouvant être définie selon le risque de déconnexion d'un nœud. Le *degré sortant* de Chord est donc $\Delta = O(\alpha \log n)$ a.f.p., et il est au maximum $O(\alpha \log N)$ (rappelons que $\ell = \log N$). Cela permet à Chord d'effectuer un compromis entre le risque de connexion et le degré des nœuds. Ainsi, en fixant $\alpha = O(\log n)$, Chord prouve que le routage reste efficace même lorsque chaque nœud a une probabilité de déconnexion

de 1/2. Le degré entrant de Chord est $O(\log n)$ en moyenne et $O(\log^2 n)$ a.f.p. le *degré* est donc $O(\alpha \log n)$ en moyenne et $O(\log^2 n + \alpha \log n)$ a.f.p.

Publication des clés

Puisque le routage garantit de trouver le nœud responsable d'une clé destination, il suffit d'envoyer un message de publication à destination de la clé c pour que le nœud responsable le reçoive, et mette à jour sa table de clés.

Routage

Lorsqu'un nœud souhaite envoyer un message à un nœud v ou trouver une clé v, il utilise l'algorithme suivant : il cherche parmi ses voisins le nœud dont l'identifiant est le plus grand tout en étant inférieur à v, et lui transmet le message. Le nœud qui reçoit le message exécute alors à son tour cet algorithme. La recherche divise la distance (dans le réseau logique) séparant le nœud courant du nœud destination v par un facteur au moins 2 à chaque étape, assurant de trouver la destination en un *nombre de sauts* a.f.p. de $\lceil \log_2 n \rceil$ et un nombre de sauts maximum de $O(\log N)$. En effet, le pire cas consiste pour un nœud à envoyer un message qui doit être routé par tous les nœuds du réseau. Par exemple, lorsqu'un nœud x envoie un message à destination de la clé $x-1$ dans un réseau à $n = \log_2 N$ nœuds, cette borne est atteinte si pour tout $i \in [1, l-1]$, le $i^{ème}$ nœud sur la route du message $x + \sum_i^{l-1} 2^i$.

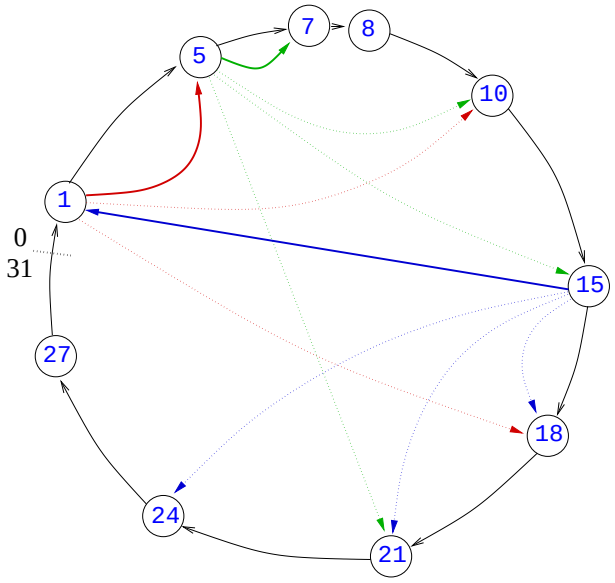

FIG. 3.2 – Routage dans Chord.

Un exemple de routage est donné dans la figure 3.2. Cette figure représente des nœuds dans un espace de nommage $[0, 32[$. Le nœud d'identifiant 15 envoie un message à destination de la clé 6. Des arcs pleins montrent le chemin suivi par le message de son

origine à sa destination. Le voisinage des nœuds intervenant dans le routage (15, puis 1, puis 5, puis 7) de cette requête est représenté avec des arc pointillés.

La requête est d'abord envoyée par le nœud 15 à son voisin d'identifiant 1 (le voisin qui a le plus grand identifiant modulo 32 inférieur à 6). Selon la même méthode, le message passe ensuite par 5 puis 7. Le message est alors arrivé à destination puisque la clé 6 est hébergée par le nœud 7 (c'est le nœud de plus petit identifiant supérieur à 6).

Insertion des nœuds dans le réseau

Lorsqu'un nouveau nœud x rejoint le réseau, il doit créer sa table de routage et la remplir, mettre à jour les tables de routage des nœuds présents dans le réseau, et récupérer les clés dont il est désormais responsable. Par le biais d'une passerelle, x peut remplir sa table de routage en effectuant une recherche avec l'algorithme de routage. Il faut donc $O(\log n)$ messages pour chacun de ses $\log N$ voisins d'identifiants $x + 2^i$, $i \in [0, \ell[$ ($\ell = \log N$). Cette étape prend $O(\log N \log n)$ messages a.f.p. et peut être réduite à $O(\log^2 n)$ messages a.f.p. en ne tenant compte dans la table de routage que des voisins différents. Il suffit pour cela de vérifier avant l'envoi d'un message au $i + 1^{ème}$ voisin si le $i^{ème}$ voisin n'est pas aussi le $i + 1^{ème}$ voisin). Cet algorithme d'insertion du nœud x peut partir de la table de routage de $pred(x)$, et nécessite alors $O(\log n)$ messages a.f.p.

Quoiqu'il en soit, il sera nécessaire de prévenir les nœuds du réseau qui doivent avoir ce nouveau nœud pour voisin. Pour tout $i \in [0, \ell[$, chaque nœud x contacte pour cela $pred(x - 2^i)$ dans l'ordre décroissant des i en utilisant l'algorithme de routage. Chacun de ces nœuds avertit ses prédécesseurs sur l'anneau jusqu'au premier prédécesseur y pour lequel $succ(y + 2^i) \neq x$. Un nœud x a $O(\log n)$ prédécesseurs en moyenne, et $O(\log^2 n)$ prédécesseurs a.f.p. ($O(\log n)$ nœuds ont x pour $i^{ème}$ successeur). Il faut donc $O(\log n)$ messages pour atteindre le $i^{ème}$ prédécesseur, . En moyenne, trouver les $O(\log n)$ prédécesseurs d'un nouveau nœud demande donc $O(\log^2 n)$ messages. Avec forte probabilité, $O(\log n)$ messages sont nécessaires à un nœud x pour trouver le plus grand nœud qui l'a pour $i^{ème}$ successeur. Il faut ensuite $O(\log n)$ messages pour trouver les autres prédécesseurs de x de niveau i *via* l'anneau. Il faut donc $O(\log^2 n)$ messages a.f.p. pour trouver tous les prédécesseurs d'un nouveau nœud. Afin de réduire ce coût, les auteurs préfèrent l'utilisation de la méthode dite «auto-stabilisante» décrite au chapitre *Divers* pour maintenir à jour le voisinage en dehors de l'anneau.

Enfin, les clés c de $succ(x)$ dont l'identifiant vérifie $c \leq x$ doivent être déplacées vers x. Cette étape ne nécessite donc qu'un message.

L'*insertion* d'un nœud dans le réseau nécessite donc $O(\log^2 n)$ messages en moyenne et a.f.p.

Départ de nœuds du réseau

Lorsqu'un nœud quitte le réseau en prévenant ses voisins, il transfère les clés dont il est responsable à son successeur. Ce successeur met alors à jour ses connexions, et avertit son nouveau prédécesseur pour qu'il mette à jour ses connexions.

Divers

Par ailleurs, chaque nœud x met à jour régulièrement son voisinage en exécutant un algorithme qui vérifie que $pred(x)$ et $succ(x)$ sont corrects. x tire ensuite aléatoirement un voisin y, et vérifie qu'il s'agit bien du bon nœud successeur. Si tel n'est pas le cas, le successeur correct est recherché par l'algorithme de routage. Cet algorithme de vérification du voisinage des nœuds permet de gérer une arrivée parallèle de plusieurs nœuds à la fois plus simple que la mise-à-jour nécessitant $O(\log^2 n)$ messages.

Chord [96]	en moyenne	a.f.p.
diamètre	$O(\log n)$	$O(\log n)$
degré	$O(\log n)$	$O(\log^2 n)$
insertion (nombre de messages)	$O(\log^2 n)$	$O(\log^2 n)$
engorgement	$O(\log n/n)$	

3.2.3 Tapestry

Le projet Tapestry [105] a été initié pour servir de base au système d'entrepôt de données persistant Oceanstore. Ce protocole a été présenté en 2001 [105], la même année que CAN [83] et Chord [96]. Il se base sur un algorithme de Plaxton, Rajaraman et Richa, qui a été adapté au dynamisme d'un système pair-à-pair. Ce protocole a été mis en œuvre dans le système de partage de fichiers OceanStore. Certaines bornes présentées dans cette contribution sont données, et une évaluation de ce protocole est effectuée par des simulations.

Structure logique

Dans Tapestry, lorsqu'un message est routé vers un destinataire, le préfixe commun entre les nœuds intermédiaires et le destinataire augmente à chaque saut. Pour cela, les nœuds maintiennent une topologie qui ressemble à un hypercube, où les connexions sont orientées. Les identifiants sont composés de $\ell = \log N / \log \beta$ chiffres, où N est le nombre maximum de nœuds dans le réseau. Dans l'article [105], un alphabet hexadécimal est utilisé, et donc $\beta = 16$.

Connexion entre les nœuds

Afin de pouvoir assurer le routage, chaque nœud x doit maintenir une table de routage contenant $\beta\ell$ voisins. La table de routage d'un nœud $x = x_1 \dots x_\ell$ contient des voisins de la forme $y_0 \dots y_{i-1} z x_{i+1} \dots x_{\ell-1}$, avec $0 \leq i \leq \ell$, $0 \leq z \leq \beta$ et y_k quelconque pour $0 \leq k < i$ (lorsque $i = \ell - 1$, l'identifiant est $y_0 \dots y_{\ell-2} z$). Pour des raisons de redondance, chaque contact est doté de deux remplaçants au cas où il serait déconnecté. Un nœud connaît aussi les adresses des nœuds qui l'ont pour voisin afin de les aider à maintenir leur table de routage, ce qui double la taille moyenne de la table de routage. Le *degré* de Tapestry est donc $O(\beta \log N / \log \beta)$.

Lorsque plusieurs choix sont possibles, le nœud retenu comme voisin principal est celui dont la latence est la plus faible, afin de diminuer le temps des sauts entre voisins (voir chapitre 3.4.3).

Publication des clés

La publication d'une clé par un nœud source se fait sur chaque nœud rencontré sur le chemin du possesseur au responsable de la clé. Tapestry utilise aussi la multi-publication de clé en concaténant des chiffres (déterminés par le système) à l'objet avant hachage, afin d'obtenir plusieurs nœuds responsables pour un objet (voir chapitre 3.4.5). Tapestry utilise aussi une mise en cache (voir chapitre 3.4.8) des associations (clé,nœud) pour accélérer les recherches.

La réflexion menée par Tapestry sur l'influence et la faisabilité des republications régulières de clés (en terme de trafic entre autres) mène Tapestry à proposer l'ajout aux associations (clé,nœud) d'un numéro de version de publication et de l'adresse physique du nœud qui a envoyé le message de publication. Dans le cas de changement de nœud responsable d'un objet, mais aussi d'un changement de version d'un objet proposé par une source, cela permet de mettre à jour les copie effectuées le long de la publication seulement sur les nœuds entre la source de l'objet et le responsable (éventuellement nouveau) de la clé.

Routage

Lorsqu'un message est envoyé vers une clé, il suit un routage préfixe. Le message se rapproche ainsi du nœud responsable de la clé à chaque saut. Lors de l'envoi d'un message vers la clé $x_1 \dots x_\ell - 1$, il est possible qu'un nœud $y_0 \dots y_i x_{i+1} \dots x_{\ell-1}$ sur le chemin ne trouve pas de voisin $y_0 \dots y_{i-1} x_i \dots x_{\ell-1}$ auquel faire suivre le message. Le message est alors transféré par une méthode dite de routage «alternatif». $y_0 \dots y_i x_{i+1} \dots x_{\ell-1}$ remplace alors ce voisin absent par un de ses voisins ayant un préfixe commun avec la clé d'au moins $\ell - i - 1$ chiffres (cela permet de s'assurer que la distance entre le message et son destinataire dans le réseau logique n'augmente jamais). Le voisin remplaçant est choisi de manière déterministe, par exemple en prenant le voisin $z_0 \dots z_{i-1}((x_i + 1) \ mod \ \beta) \dots x_{\ell-1}$. Si ce nœud remplaçant est lui-même absent de la table de routage, alors son remplacement est effectué de la même manière par un nœud $z_0 \dots z_{i-1}((x_i + 2) \ mod \ \beta) \dots x_{\ell-1}$. Lorsqu'un nœud ne trouve aucun remplaçant dans sa table de routage (c'est-à-dire si la colonne est vide), l'algorithme décide de manière déterministe d'un responsable pour la clé. Cet algorithme impose que, si il manque des voisins à un nœud, tous les nœuds du réseau ayant le même suffixe ont les mêmes voisins manquants. Un exemple de routage alternatif est donné ci-dessous.

Le nombre de sauts maximal peut donc dépasser $O(\log N / \log \beta)$ si le transfert du message doit faire appel au routage alternatif. Le *nombre de sauts* moyen pour transférer un message est $O(\log N / \log \beta)$.

La figure 3.3 donne un exemple de routage d'un nœud 0123 à un nœud d'identifiant 4642. Le message est routé de 1234 à $xyz2$, puis est envoyé à $x'y'42$. Ce nœud

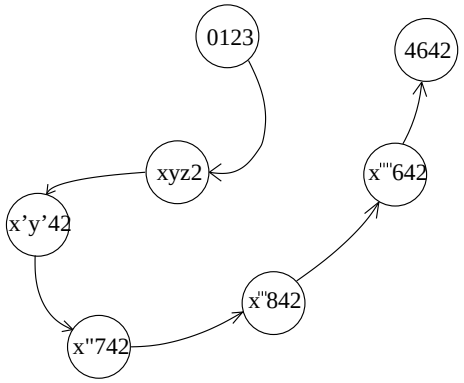

FIG. 3.3 – Routage dans Tapestry.

n'ayant pas de voisin dont le suffixe soit 642, le message est transmis au remplaçant de ce nœud, x"742. Ce nœud n'ayant lui-même pas de voisin de suffixe 642 ni de voisin de suffixe 742, le message est transmis à $x"'842$, qui lui a un voisin de suffixe 642 et lui envoie le message. Le message est donc envoyé à $x""642$, qui connaît le responsable de la clé 6462. Notons qu'il n'est pas nécessaire d'envoyer le numéro de l'étape du routage en plus du destinataire avec le message, ce numéro d'étape pouvant se déduire du plus long suffixe commun entre le nœud courant et le destinataire.

Insertion des nœuds dans le réseau

Lors de l'insertion d'un nouveau nœud x *via* une passerelle y, x se choisit un identifiant et envoie un message vers cet identifiant. Ce message lui permet de remplir sa table de routage comme suit : le $i^{ème}$ nœud sur la route du message ($i \in [1, \ell]$), en partant de la passerelle y, fournit ses voisins ayant pour préfixes $y_0 \dots y_{\ell-i-1}z$, $z \in [0, \beta - 1]$. À chaque fois qu'un nœud lui communique ses voisins pour un i donné, x réordonne les nœuds associés à chacun de ses voisins selon leur proximité dans le réseau physique. x vérifie ensuite la proximité des voisins de ses voisins et les réordonne de même. Il continue jusqu'à ce que les améliorations apportées soit inférieures à un seuil fixé par le système. Les nœuds qui doivent prendre x pour voisin sont alors avertis de son arrivée pour mettre à jour leur table de routages. Ils sont trouvés par un envoi de messages vers les nœuds remplaçants (en cas d'absence de nœuds dans la table de routage) selon la méthode de routage.

Départ de nœuds du réseau

Lors du départ d'un nœud, celui-ci prévient les nœuds dont il est voisin afin qu'ils mettent leur table de routage à jour. Pour chaque association (clé,x) qu'il a en copie dans sa table de clés le nœud partant prévient celui qui partage l'objet associé à la clé afin qu'il publie à nouveau.

Divers

Tapestry vérifie régulièrement que ses voisins sont vivants et leur proximité physique, afin de choisir comme voisin principal le plus proche parmi ses voisins possibles (voir chapitre 3.4.3). Dans le cas où un voisin ne répond plus, il est remplacé comme voisin principal mais reste dans les doublons de la table de routage durant une période fixée par le système, au cas où cette indisponibilité ne serait que temporaire.

Des simulations montrent que Tapestry bénéfice d'une bonne proximité physique par le choix de voisins proches dans le réseau physique. En effet, le nombre de voisins parmi lesquels un nœud choisit le prochain intermédiaire d'un message décrémente exponentiellement à chaque saut, donc les intermédiaires pour router le message est de plus en plus faible. La probabilité que ces intermédiaires soient proches dans le réseau physique d'autant moins importante à chaque saut. Le temps mis à effectuer les premiers sauts est donc faible, et augmente à chaque saut.

Ce protocole identifie les nœuds en utilisant un alphabet sur β chiffres (voir chapitre 3.4.1), il propose aussi une technique de réplication décrite en chapitre 3.4.8.

Dans la table suivante, β est le nombre de chiffres de l'alphabet utilisé.

Tapestry [105]	en moyenne
diamètre	$O(\log n/\log \beta)$
degré	$O(\beta \log N/\log \beta)$
insertion (nombre de messages)	$O(\log n/\log \beta)$
engorgement	$O(\log n/(n \log \beta))$

3.2.4 Pastry

Le protocole Pastry [88] est la base d'un système d'entrepôt de donnée anonyme du département Recherche de Microsoft, nommé PAST. Il a de grandes similitudes avec Tapestry, c'est pourquoi nous décrirons son fonctionnement par rapport à Tapestry. Des simulations ont permis de vérifier le bon fonctionnement du protocole.

Structure logique

La structure logique et le routage sont inspirés du même algorithme que Tapestry : celui proposé par Plaxton, Rajaraman et Rachi. Pastry utilise des identifiants codés sur un alphabet de β chiffres, de longueur $\log N/\log \beta$.

Connexion entre les nœuds

Pastry utilise une table de routage remplie de manière similaire à Tapestry, mais il utilise aussi un ensemble M de voisins proches dans le réseau physique ($|M|$ fixé par le système, typiquement $|M| = \beta$), et un ensemble L de feuilles ($|L|$ fixé par le système, typiquement $|L| = \beta$) qui sont les nœuds dont les identifiants sont les plus proches de l'identifiant du nœud.

Publication des clés

Pastry n'est utilisé que pour faire du routage dans le réseau logique, mais dans l'application PAST basée dessus, une clé est enregistrée sur α nœuds d'identifiants les plus proches, où α est un paramètre du système assurant qu'un message routé vers une clé trouve l'un des α plus proche nœud de cette clé. À l'inverse de Tapestry, une association (clé,nœud) clé n'est pas enregistrée le long du chemin entre la source et le responsable de la clé. Toutefois la réplication des associations (clé,nœud) se fait par mise en cache et non de manière contrôlée comme c'est le cas pour Tapestry.

Routage

Pour le routage, un nœud $x_0 \dots x_i u_{i+1} \dots u_{\ell-1}$ devant router un message vers la clé $u_0 \dots u_{\ell-1}$ cherche d'abord si le responsable de cette clé est une de ses $|L|$ feuilles (cela revient à vérifier si la clé est dans l'intervalle couvert par ses $|L|$ feuilles). Si c'est le cas, le message est envoyée à ce nœud responsable. Sinon, le nœud $x_0 \dots x_i u_{i+1} \dots u_{\ell-1}$ cherche si un destinataire peut être trouvé dans sa table de routage, par routage préfixe (comme dans Tapestry). Enfin, si un tel nœud n'est pas trouvé, $x_0 \dots x_i u_{i+1} \dots u_{\ell-1}$ cherche parmi ses voisins le nœud dont :

- le préfixe commun avec la clé $u_0 \dots u_{\ell-1}$ est au moins $\ell - i - 1$;
- l'identifiant rapproche le plus de la clé $u_0 \dots u_{\ell-1}$.

En effet, un chiffre de l'alphabet $[0, \beta]$ est constitué de plusieurs bits, plusieurs voisins peuvent donc rapprocher de la clé sans pour autant avoir pour suffixe $u_i \dots u_{\ell-1}$. Contrairement à Tapestry, le routage de Pastry a pour but d'atteindre l'un des α nœuds les plus proches de la clé (où α est un paramètre fixé par le système) et non l'unique responsable de la clé.

Insertion et départ des nœuds du réseau

La méthode d'insertion de nœuds dans le réseau est la mêmes que Tapestry, et un nœud choisit aussi des voisins proches dans le réseau physique lorsque c'est possible. Lors d'un départ, un remplaçant au voisin est trouvé, de manière différente selon qu'il était une feuille, un voisin proche dans le réseau physique, ou un voisin de la table de routage.

Divers

Les simulations menées par Pastry montre qu'il bénéficie de la proximité physiques de voisins de la même façon que Tapestry.

Dans la table suivante, β est le nombre de chiffres de l'alphabet utilisé.

Pastry [88]	en moyenne
diamètre	$O(\log n / \log \beta)$
degré	$O(\beta \log N / \log \beta)$
insertion (nombre de messages)	$O(\log n / \log \beta)$
engorgement	$O(\log n / (n \log \beta))$

3.2.5 Viceroy

Le protocole Viceroy [68] proposé en 2002 permet un degré sortant contant, au prix toutefois d'un protocole complexe. Il se montre ainsi plus adapté à des environnements où les pannes sont rares. L'évaluation de ce protocole est effectuée par des résultats avec forte probabilité.

Structure logique

Viceroy utilise comme base un graphe papillon. Chaque nœud a un identifiant tiré aléatoirement uniformément sur $[0, 2^\ell[$, (une valeur $\ell = 128$ est proposée) et un niveau tiré aléatoirement sur $[1, \log_2 n]$ après estimation du nombre de nœuds n dans le réseau. Tous les calculs sur les identifiants se font donc modulo 2^ℓ. Les nœuds sont connectés de deux façons différentes :

- un anneau bidirectionnel relie les nœuds selon leur identifiant, quelque soit leur niveau ;
- des anneaux de niveaux relient les nœuds de chaque niveau ;
- un graphe papillon relie les nœuds, au moyen de liens vers des nœuds des niveaux supérieur et inférieur.

Pour n nœuds dans le réseau, Viceroy comporte $\log_2 n$ niveaux de $n/\log n$ nœuds chacun.

Connexion entre les nœuds

Un nœud $u = u_0 \dots u_\ell$ de niveau p fait partie de trois structures distinctes :

- un anneau global reliant les nœuds selon leur identifiant, quelque soit leur niveau. Le nœud u est alors relié aux nœuds $succ(u)$ dont l'identifiant est le plus proche supérieur à u et $pred(u)$ le plus proche identifiant inférieur à u ;
- un anneau de niveau p relie les nœuds de niveau p selon leur identifiant. Le nœud u est alors relié aux nœuds $succ_p(u)$ et $pred_p(u)$ de niveau p dont l'identifiant est respectivement le plus proche supérieur à u et le plus proche identifiant inférieur à u ;
- des liens inspirés du graphe papillon connectent $u = u_1 \dots u_\ell$ aux niveaux supérieur et inférieur : les liens mènent aux nœuds de niveau $p + 1$:
  - $succ_{p+1}(u)$ d'identifiant immédiatement supérieur à $u_1 \dots u_\ell$;
  - $succLong_{p+1}(u)$ d'identifiant immédiatement supérieur à $u_1 \dots u_{p-1}\overline{u_p}u_{p+1} \dots u_\ell$.

  Si $p \neq 1$, un lien connecte aussi u au nœud de niveau $p - 1$ dont l'identifiant est immédiatement supérieur à $u_1 \dots u_\ell$.

Le degré sortant d'un nœud est donc 7 auquel s'ajoute en moyenne $O(1)$ voisins entrants.

La figure 3.4 représente un exemple de réseau sur un espace d'identifiants $[0, 2^4[$ ($\ell = 4$). Pour plus de clarté, on n'y voit pour chaque nœud de niveau p que les liens vers ses voisins de niveau $p + 1$. Chaque nœud a donc en plus ses connexions (successeur et prédécesseur) dans l'anneau global et la connexion vers le nœud de niveau inférieur.

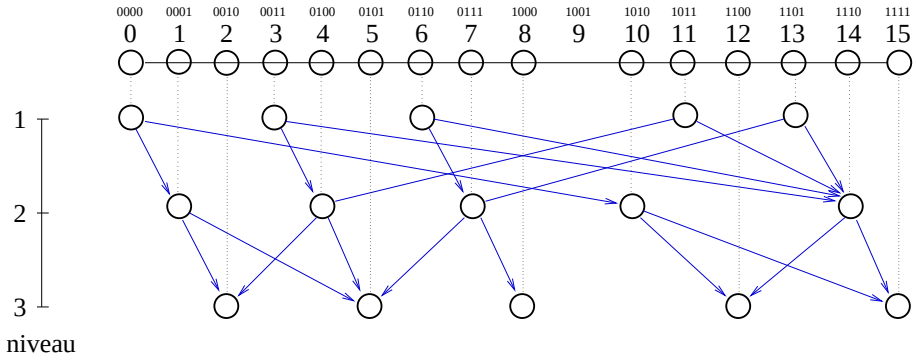

FIG. 3.4 – Connexions dans Viceroy.

Publication des clés

Le responsable d'une clé est le nœud dont l'identifiant est immédiatement supérieur à l'identifiant de la clé. La publication se fait sur le nœud responsable trouvé par le routage.

Routage

Le routage d'un message vers une clé u s'effectue alors en trois étapes :

- le message est récursivement «redescendu» vers un nœud de niveau 1 *via* les liens vers le niveau supérieur ;
- une fois arrivé au niveau 1, le message est ensuite récursivement dirigé vers la clé en empruntant successivement les liens qui le rapproche le plus de cette clé. Sur un nœud u de niveau p sur le chemin vers la clé x : si $|x-u| < 2^p$, le message est dirigé vers le $succ_p(u)$. Sinon, le message est dirigé vers $succLong_p(u)$. Si aucun lien n'existe ou que le lien à suivre mène à un nœud d'identifiant supérieur à la clé, on passe à la phase suivante ;
- tant que u n'est pas le nœud courant n'est pas responsable de la clé x, cette phase est répétée récursivement. Sinon, si $succ_p(u) \in [u, x]$, le message est dirigé vers $succ_p(u)$. Si $pred_p(u) \in [u, x]$, le message est dirigé vers $pred_p(u)$. Dans les autres cas, le message est dirigé vers $succ(u)$ ou $pred(u)$ selon celui qui rapproche le plus de la clé x.

Cet algorithme s'arrête une fois trouvé le nœud responsable de la clé x.

Insertion des nœuds dans le réseau

Pour chaque nœud u, on nommera $succ(u)$ le nœud dont l'identifiant est immédiatement supérieur à celui du nœud u sur l'anneau global (quelque soit son niveau). $pred(u)$ sera le nœud dont l'identifiant est immédiatement inférieur à celui du nœud u. $succ_k(u)$ sera le premier successeur de u de niveau k dans l'anneau global.

Lorsqu'un nouveau nœud arrive, il tire aléatoirement un identifiant u et contacte une passerelle x dans le réseau pour qu'elle envoie un message d'insertion vers la clé u.

Notons $w = succ(u)$ l'identifiant du nœud trouvé par ce message, u s'insère alors entre w et $pred(w)$, dans l'anneau global. Les clés dont u est responsable, entre $pred(u)$ et u, lui sont alors transférées par leur ancien responsable w.

u se choisit ensuite aléatoirement un niveau p entre 1 et $\log_2 n_0$ où $n_0 =\backsimeq 1/|u - succ(u)|$ est l'approximation du nombre courant de nœuds n. u cherche alors cherche son successeur de même niveau p (respectivement son prédécesseur de même niveau p) par un parcours récursif de l'anneau global *via* son successeur (respectivement son prédécesseur). u recherche ensuite $succ_{p+1}(u)$ par un parcours récursif de l'anneau global *via* son successeur. Enfin, il recherche par la méthode de routage normale le nœud $succ(u_1 \dots u_{p-1}\overline{u_p}u_{p+1} \dots u_\ell)$ de niveau supérieur $p+1$.

À un niveau inférieur à $(\log_2 n)/2$, la recherche de voisins par parcours récursif *via* les successeurs sur l'anneau global peut demander un grand nombre de sauts. C'est pourquoi si le nombre de sauts nécessaire est supérieur à $O(\log_2{}^2 n)$, le lien vers le voisin est laissé vide.

Départ de nœuds du réseau

Quand un nœud quitte le réseau, il prévient ses voisins sortants et ses voisins entrants. Il trouve ces derniers par recherche normale. Enfin, il transfère ses clés à son successeur.

Viceroy [68]	en moyenne	a.f.p.
diamètre	$O(\log n)$	$O(\log n)$
degré	$O(1)$	$O(\log n)$
insertion (nombre de messages)	$O(\log n)$	$O(\log^2 n)$
engorgement	$O(\log n/n)$	$O(\log^2 n/n)$

3.3 Systèmes décentralisés structurés souples

L'une des difficultés à laquelle doivent faire face les réseaux à contenu adressable que nous avons décrit jusqu'alors est le routage dans un environnement dynamique. Afin d'assurer la disponibilité des clés, des systèmes ont proposés de recopier les clés sur des nœuds proches de leur responsable. Pastry [88] a proposé un routage qui trouve l'un des α plus proches nœud de la clé recherchée (α étant un paramètre fixé par le système). Couplée à la redondance des clés, Pastry autorise donc plus aisément les pannes de nœuds responsables des clés.

En 2002 sont apparus des systèmes qui poussent le principe cette décentralisation plus loin en ajoutant à la redondance des clés une nouvelle méthode d'envoi des messages. Ces systèmes changent le routage utilisé habituellement dans les systèmes structurés en une recherche itérative des α nœuds les plus proches de la clé, α étant là encore une constante fixée par le système. Cet algorithme se termine lorsqu'un nombre suffisant de nœuds proches de la clé sont trouvés. Cette méthode permet au système de résister à la disparition de nœuds serveurs *et* routeurs de manière simple. Nous allons décrire dans la suite deux systèmes basés sur ce principe.

3.3.1 Kademlia, ou l'introduction d'un peu de souplesse

Kademlia [70] est un protocole créé pour être moins sensibles aux pannes. Il se base lui aussi sur un graphe hypercube. C'est le seul protocole déployé réellement à grande échelle, avec le logiciel eMule, *via* l'implantation *Kad !* et anciennement le logiciel eDonkey (voir chapitre 2.2.1). Tous deux permettent jusqu'à un million d'utilisateurs, chaque nœud utilisant selon ses préférences un serveur ou ce protocole décentralisé pour chacune de ses recherches. Il est donc dans ce cas une solution alternative permettant de décharger les serveurs utilisés par ce système semi-décentralisé. Comme Tapestry et Pastry, Kademlia utilise la métrique *ou exclusif* (*XOR*) pour calculer la distance séparant (dans le réseau logique) deux identifiants de nœuds. Il donne une esquisse de preuve de son comportement, qui est complétée dans Broose [100].

Structure logique

Les nœuds du réseau sont identifiés sur l'intervalle $[0, 2^\ell[$ où ℓ est un paramètre du système. Ces nœuds peuvent être représentés sous la forme d'un arbre binaire préfixe. Chaque nœud u a alors une vue partielle de cet arbre sous forme d'un peigne, ses voisins à la profondeur i sont au plus au nombre de γ, qui est un paramètre fixé par le système.

Connexion entre les nœuds

Pour savoir quels voisins attribuer à un nœud u, on identifie sur le peigne menant de la racine au nœud tous les sous-arbres existants. Un nœud u se choisira au plus γ nœuds voisins pour chaque sous-arbre identifié. Le *degré* moyen d'un nœud inséré dans un réseau de n nœuds identifiés sur un alphabet de β chiffres est donc $O(\gamma\beta \log n / \log \beta)$. Dans un sous-arbre, chacun des γ voisins sera classé selon son ancienneté dans le système, du point de vue du nœud u (c'est-à-dire selon la date du dernier message qu'il a reçu de ce voisin). Lorsqu'un nœud u découvre un nouveau nœud v, plusieurs cas sont possibles. Considérons le sous-arbre de voisins de u qui pourrait contenir v, c'est-à-dire le sous-arbre regroupant les voisins dont la préfixe commun avec l'identifiant de u est de même longueur que le préfixe commun entre l'identifiant de u et celui de v :

- soit il contient moins de γ voisins, v devient alors voisin de u dans ce sous-arbre ;
- soit le sous-arbre contient déjà γ voisins, u vérifie alors que chacun de ces voisins de ce sous-arbre est encore connecté au système.
  - si au moins un voisin est déconnecté, alors il est supprimé et le nouveau nœud v devient voisin de u dans ce sous-arbre.
  - sinon, si ce sous-arbre contient u, posons l la longueur du préfixe commun à tous les nœuds de ce sous-arbre. Ce sous-arbre est alors remplacé par deux sous-arbres dont les nœuds partagent un préfixe commun de $l+1$ bits, et dont le $l+1^{ème}$ bit est respectivement 0 et 1. Les voisins de u sont alors répartis entre ces deux nouveaux sous-arbres.

La métrique du *ou exclusif* étant symétrique, les voisins d'un nœud u sont aussi les nœuds dont u est voisin, les voisins auxquels un nœud envoie des messages sont donc

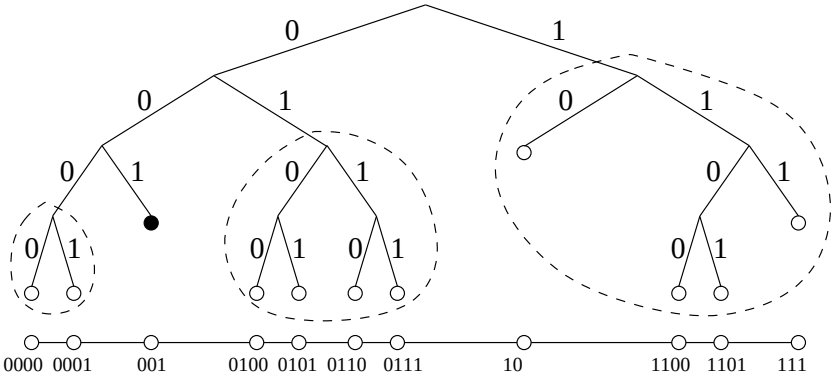

FIG. 3.5 – Voisinage d'un nœud dans Kademlia (γ voisins par ensemble).

aussi les nœuds susceptibles de lui envoyer des messages. Cela permet d'encourager la participation des voisins comme nous l'avons dit précédemment.

Publication des clés

Lorsqu'une clé v est publiée par un nœud, il recherche les γ nœuds dont les identifiants sont les plus proches de cette clé. Il enregistre alors cette clé sur ces nœuds. Pour rechercher une clé, l'algorithme de recherche s'arrête dès qu'une clé est trouvée. Les clés sont republiées toutes les heures pour compenser le départ de nœuds.

Kademlia permet de limiter la charge des nœuds responsables de clés populaires par un mécanisme de cache (voir chapitre 3.4.8). Le temps d'expiration de la clé ainsi recopiée dépendra de l'estimation du nombre de nœuds entre l'identifiant du nœud où est recopiée la clé et l'identifiant de la clé.

Acheminement des messages

Lorsqu'un nœud u doit envoyer un message dans le système à un destinataire, il initialise un ensemble U de nœuds avec les γ nœuds du sous-arbre de voisins le plus proche de la clé v. Dans le cas où ce sous-arbre comporte moins de γ nœuds, u initialise l'ensemble U avec ses γ voisins les plus proches de la clé v. Pour chaque nœud de cet ensemble, u distingue les nœuds qu'il a déjà contacté, le tour auquel il les a contacté, et s'ils ont répondu. Soit $\alpha < \gamma$ une constante fixée par le système pour représenter le nombre de nœuds auxquels il envoie des requêtes. u exécute alors un algorithme itératif sur l'ensemble U :

- u envoie aux α nœuds de U les plus proches de v un message pour obtenir leur γ voisins les plus proches de la clé v.
- lorsqu'un nœud répond en envoyant ses γ plus proches voisins de la clé v, ces voisins sont ajoutés à l'ensemble U s'ils n'y sont pas encore ;
- si aucun des nœuds qui ont été contactés à un tour donné n'a répondu au bout d'un certain temps, alors u sélectionne dans U les γ nœuds les plus proches de la clé v et qui n'ont pas encore été contactés.

Lorsqu'un réponse arrive à u, il peut contacter le nœud de u le plus proche de la clé v sans attendre toutes les réponses en cours. L'algorithme s'arrête lorsque les α plus proches nœuds de U ont répondu ou que, dans le cas d'un recherche d'une seule clé v, une clé a été trouvée.

Cet algorithme a pour but d'augmenter le préfixe commun entre l'identifiant de la clé v et celui des nœuds de U les plus proches de v à chaque saut. Le *nombre de sauts* moyen nécessaires pour une recherche de clé identifié sur un alphabet de β chiffres est $O(\log n/\log \beta)$. Notons que le nombre de messages moyen pour effectuer cette recherche est lui de $O(\alpha \log n/\log \beta)$.

Lorsque plusieurs nœuds de U peuvent être les destinataires d'un message, il est possible de choisir le destinataire selon sa proximité physique, comme vu au chapitre 3.4.3.

Insertion des nœuds dans le réseau

Pour s'insérer dans le système, un nœud u tire un identifiant uniformément aléatoirement sur $[0, 2^\ell[$. u utilise alors pour s'insérer une passerelle qu'il ajoute à son propre sous-arbre (ce sous-arbre ne contient alors que deux nœuds). u effectue ensuite une recherche sur la clé u, ce qui lui permet de découvrir des nœuds dont l'identifiant est de plus en plus proche de son identité. u ajoute ces nœuds à ses voisins dans les sous-arbres correspondant, selon leur préfixe. Cette méthode avertit en même temps de ses nouveaux voisins de son existence. L'*insertion* d'un nouveau nœud dans un réseau de n nœuds identifié sur un alphabet de β chiffres demande donc en moyenne $O(\alpha \log n/\log \beta)$ messages, comme la recherche d'une clé.

Départ de nœuds du réseau

Aucune opération spécifique n'est prévue pour le départ des nœuds. Les nœuds découvrent de nouveaux voisins qui remplacent leur voisins déconnectés, tandis que la republication régulière des objets doit suffire à garder les clés associées accessibles.

Divers

Le paramètre γ est choisi tel que la probabilité que les γ nœuds d'un sous-arbre se déconnectent en une heure soit négligeable. La valeur de ce paramètre se base sur l'étude des demi-vies des nœuds d'un système Gnutella [7] qui a observé qu'un nœud a une probabilité de rester connecté au réseau d'autant plus grande qu'il y est connecté depuis longtemps.

Dans la pratique, Kademlia utilise un arbre β-aire et non binaire, afin de diminuer le nombre de sauts nécessaire à une recherche de clé, comme la plupart des systèmes (voir chapitre 3.4).

Ce protocole propose aussi une optimisation du trafic de contrôle servant à la vérification que les voisins sont vivants.

Dans la table suivante :

- α le nombre de messages envoyés à chaque étape d'une recherche ;
- β est le nombre de chiffres de l'alphabet utilisé ;

– γ le nombre de voisins dans chaque sous-arbre d'un nœud.

Notons que le diamètre n'est pas le même que le nombre de messages nécessaire à une recherche de clé puisque α messages sont envoyés à chaque étape d'une recherche. Toutefois, un nœud déjà sollicité ne sera pas sollicité à nouveau.

Kademlia [70]	en moyenne
diamètre	$O(\log n/\log \beta)$
recherche (nombre de messages)	$O(\alpha \log n/\log \beta)$
degré	$O(\gamma \log n/\log \beta)$
insertion (nombre de messages)	$O(\alpha \log n/\log \beta)$

3.3.2 Broose

Broose [100, 101] a proposé en 2004 d'utiliser un graphe de de Bruijn non orienté, c'est-à-dire en permettant de décaler les identifiants vers la gauche ou vers la droite. Nous allons décrire ici la version binaire de Broose. Il s'inspire pour cela de la méthode de recherche de Kademlia, c'est-à-dire qu'il utilise la même métrique de distance basée sur le *ou exclusif* et qu'il s'agit d'un système décentralisé structuré souple. Son but est, comme Kademlia, d'augmenter la tolérance aux pannes des systèmes décentralisés structurés tout en tirant parti des avantages du graphe de de Bruijn pour diminuer le degré nécessaire. Il prouve aussi pour ce type de systèmes quel est le nombre de voisins proches nécessaires pour un nœud u (dans le sous-arbre contenant u pour Kademlia, ou dans l'ensemble de frères pour Broose) afin que l'acheminement des messages fonctionne correctement. L'évaluation de Broose est faite au moyen de preuves avec forte probabilité et de simulations.

Connexion entre les nœuds

Les nœuds du réseau sont identifiés sur l'intervalle $[0, 2^\ell[$ où ℓ est un paramètre du système. Chaque nœud $u = u_1...u_\ell$ sépare ses voisins en un ensemble de voisins droits, nommons-le X et un ensemble voisins gauches, nommons-le Y. X est composé de λ voisins (λ est un paramètre fixé par le système) et est divisé en deux ensembles complémentaires : X_0 est constitué des λ' plus proches nœuds d'identifiants $0u_1 \dots u_{\ell-1}$ et X_1 est constitué des $\lambda - \lambda'$ plus proches nœuds d'identifiants $1u_1 \dots u_{\ell-1}$, avec $\lambda/2 \leq \lambda' \leq \lambda$. Un nœud a aussi pour voisins un ensemble Z de $\delta = 7\lambda$ frères qui sont les nœuds d'identifiants les plus proches de u. Enfin, l'ensemble de voisins gauches Y est composés des nœuds d'identifiants $u_1 v_1 \dots v_{\ell-1}$ tels que u fasse partie des λ' plus proches voisins de v. u a donc v pour voisins dans Y si v a u pour voisin dans X. Le *degré* des nœuds dans un système utilisant un alphabet de β chiffres est donc $O(\lambda\beta)$.

Quant un nœud u reçoit un message d'un nœud v du réseau qui pourrait avantageusement entrer dans sa table de routage, v devient voisin de u. Lorsqu'un choix existe, un nœud garde comme voisins les nœuds les plus proches physiquement (contrairement à Kademlia qui garde les plus anciens dans le réseau). Cela peut augmenter le nombre de mise à jour nécessaire mais assure des sauts plus rapides dans le réseau logique.

Publication des clés

Comme Kademlia, lorsqu'une clé x est publiée par un nœud, il recherche par la méthode de routage les λ nœuds dont les identifiants sont les plus proches de cette clé. Il enregistre alors cette clé sur ces nœuds. Les clés sont aussi republiées toutes les heures pour compenser le départ de nœuds. Cette republication est effectuée par l'un des responsables de la clé et non par le nœud qui partage l'objet associé. Un nœud u ne republie une clé que si aucun autre nœud n'a republié cette clé (lié au même objet partagé) au cours de la dernière heure, et si le nœud est toujours parmi les λ plus proches nœuds de la clé, c'est-à-dire s'il en est toujours responsable. Une telle republication n'a lieu que durant 24 heures après la première publication afin de permettre l'expiration des clés publiées, et le nœud qui partage l'objet associé doit republier la clé toutes les 24 heures (soit directement, soit après être averti par l'un des responsables de la clé associée).

Une méthode de réplication est proposée afin de limiter la charge des nœuds responsables de clés populaires (voir chapitre 3.4.8).

Acheminement des messages

Lorsqu'un nœud u envoie un message à destination d'une clé x, il utilise une *recherche droite* qui s'effectue en plusieurs tours. Cette recherche droite utilise un routage par décalage des identifiants vers la droite. Comme dans Kademlia, u crée un ensemble de nœuds U. Il y place son identité puis estime la distance d_u qui le sépare de x dans le réseau logique (par la métrique du *ou exclusif*). La distance séparant la clé x des nœuds de U les plus proches de cette clé est nommée d. Pour chaque nœud de cet ensemble U, u distingue les nœuds qu'il a déjà contacté, la distance qui les sépare de x (c'est-à-dire le tour auquel u les a contacté), et s'ils ont répondu. Soit α une constante fixée par le système pour représenter le nombre maximal de nœuds auxquels u envoie des requêtes. Comme Kademlia, u va alors envoyer itérativement des messages aux nœuds de U choisis tels que au $i^{ème}$ tour, $i \in [0, d_u]$, les messages sont envoyés à des nœuds à distance $d = d_u - i$ de x.

- u envoie un message à un nombre de nœuds variant de 1 à α pour obtenir leur voisinage à distance $d-1$ de x. Il s'agit des λ' ou λ voisins qui sont dans leur ensemble X_j pour $j = x_d$ le $d^{ème}$ chiffre de la clé x ;
- si u reçoit une réponse pour une demande envoyée à un nœud à distance d, d est décrémentée et U est remplacé par l'ensemble des nœuds contenus dans la réponse ;
- si u reçoit une réponse d'un nœud à distance $d+1$, il ajoute à U les nœuds contenus dans la réponse ;
- si u reçoit une réponse d'un nœud à distance supérieure à $d+1$, il ignore cette réponse.

L'algorithme s'arrête lorsque la distance atteinte est $d = 0$. Le *nombre de sauts* moyen nécessaires pour trouver une clé identifiée sur un alphabet de β chiffres est $O(\log n / \log \beta)$. Notons que le *nombre de messages* moyen pour trouver cette clé est lui de $O(\alpha \log n / \log \beta)$.

Si pour une distance d, aucun des nœuds ne répond, ces nœuds sont retirés de U tandis que la requête est envoyée à α nœuds de U (pas encore contactés). Cet algorithme

permet de trouver un des λ' plus proches voisins et est donc efficace pour trouver une clé.

Toutefois, si aucune clé n'est trouvée par cette méthode en arrivant à distance 0, alors il est nécessaire d'effectuer une recherche complète pour trouver les λ plus proches nœuds de v. Dans ce cas, u demande aux λ nœuds les proches de v qu'il connaît (parmi son voisinage *et* les nœuds de U) leurs λ frères les plus proches de v. Si un de ces nœuds ne répond pas, alors il envoie cette demande au $\lambda + 1^{ème}$ nœud le plus proche de v qu'il connaisse. Cette opération est réitérée jusqu'à ce que les λ plus proches nœuds de v aient répondu. Cette recherche aussi être effectuée par le biais d'une recherche gauche si désiré. Contrairement à Kademlia pour lequel cette dernière phase peut demander plusieurs sauts, Broose ne nécessite qu'un seul saut pour finir cette phase de la recherche.

Une recherche gauche d'une clé x par un nœud u suivant le même principe que la recherche droite peut aussi être effectuée par tout nœud u.

Insertion des nœuds dans le réseau

Lorsqu'un nœud veut s'insérer dans le réseau, il tire aléatoirement uniformément un identifiant $u = u_1 \dots u_n$. Il passe alors par une passerelle qui va effectuer pour lui une recherche complète (donc avec recherche de frères une fois arrivé à distance 0) sur les clés $0u_1 \dots u_{\ell-1}$ et $1u_1 \dots u_{\ell-1}$. Une fois λ' réponses obtenues pour chacune des requêtes, ces nœuds forment l'ensemble X composé des ensembles X_0 et X_1. L'ensemble des frères Z de u peut alors être rempli par en demandant à tous les voisins droits de u (les nœuds de l'ensemble X) leurs ensembles de voisins gauches Y. Les $\delta = 7\lambda$ plus proches nœuds trouvés sont alors placés dans l'ensemble des frères Z. Enfin, u peut créer son ensemble de voisins gauches Y en demandant à ses frères (les nœuds de l'ensemble Z) leurs ensembles de voisins gauches Y. Une méthode alternative est proposée pour trouver les frères d'un nouveau nœud u, elle permet de partir de l'ensemble des λ plus proches nœuds de u. L'*insertion* d'un nouveau nœud dans un réseau de n nœuds identifié sur un alphabet de β chiffres demande donc en moyenne $O(\alpha \log n / \log \beta) + O(\lambda)$ messages. Il faut toutefois noter que le facteur $O(\lambda)$ permet d'assurer que l'acheminement d'un message permet de trouver les λ plus proches nœuds d'une clé (il s'agit de γ pour Kademlia).

Afin de mettre à jour les voisins, les requêtes vers les clés $0u_1 \dots u_{\ell-1}$ et $1u_1 \dots u_{\ell-1}$ sont effectuées toutes les heures, suivies d'une mise-à-jour des ensembles de frères et de voisins gauches et droits. Notons que comme Kademlia, la mise-à-jour des voisins de u se fait naturellement grâce à tous les messages que reçoit u. Afin d'aider à la mise-à-jour des ensembles de voisins gauches Y, il est aussi proposé pour les recherches d'associer au message envoyé à un nœud $u \in U$ à distance d la liste des nœuds de U à distance $d+1$.

Départ de nœuds du réseau

Aucune opération spécifique n'est prévue pour le départ des nœuds. Les nœuds découvrent de nouveaux voisins qui remplacent leur voisins déconnectés, tandis que la republication régulière des objets doit suffire à garder les clés associées accessibles.

Divers

De manière générale viennent s'ajouter à Broose des optimisations comme :
- l'utilisation pour les identifiants de nœuds d'un alphabet β-aire, où β est une estimation de $\log_2 n$ avec n le nombre de nœuds présents dans le réseau (voir chapitre 3.4.1) ;
- l'envoi des messages de préférence à des nœuds physiquement proches lorsque le choix existe (voir chapitre 3.4.3) ;
- la mise en cache des clés sur plusieurs nœuds de préfixes différents (voir chapitre 3.4.8), par une méthode adaptée au réseau construit par Broose. Lorsqu'une clé est renvoyée par un nœud u à distance estimée i de l'identifiant de la clé, la clé est mise en cache sur le nœud qui a renvoyé le nœud u.
- la réplication des clés sur des nœuds du réseau (voir chapitre 3.4.8). La méthode utilisée pour trouver les nœuds sur lesquels sont copiées les associations (clé,nœud) est la méthode du hachage mixte (présentée au chapitre 3.4.8).

Dans la table suivante :
- α le nombre maximal de messages envoyés à chaque étape d'une recherche ;
- β est le nombre de chiffres de l'alphabet utilisé ;
- λ le nombre de voisins des ensembles gauches et droits pour chaque nœud.

Comme pour Kademlia, notons que le diamètre n'est pas le même que le nombre de messages nécessaire à une recherche de clé puisque α messages sont envoyés à chaque étape d'une recherche. Toutefois, un nœud déjà sollicité ne sera pas sollicité à nouveau.

Broose [101]	en moyenne	a.f.p.
diamètre	$O(\log(n/\lambda)/\log\beta)$	$O(\log(n/\gamma)/\log\beta)$
recherche (nombre de messages)	$O(\alpha\log n/\log\beta)$	
degré	$O(\lambda\beta)$	$O(\lambda\beta)$
insertion (nombre de messages)	$O(\lambda+\alpha\beta\log n/\log\beta)$	

3.3.3 Quelques autres propositions

Après ces premiers travaux qui ont servi de base aux réseaux à contenu adressable, bien des travaux ont permis l'amélioration de ces derniers.

Symphony [69] est une proposition de réseau à contenu adressable basé sur un graphe petit monde, qui ont la propriété d'avoir des distance faibles pour un degré faible. Il utilise pour cela une topologie en anneau assortie de k liens longs sur chaque nœuds afin de permettre un routage vers le voisin le plus proche du destinataire dans le réseau logique. Ce routage s'effectue en un nombre de sauts moyen $O((\log^2 N)/k)$ pour un degré constant $O(k)$.

«P2P against Censorship» [31], propose de maintenir une structure dans laquelle l'effacement d'une fraction constante de nœuds par rapport au nombre de nœuds total du réseau laisse le système fonctionnel. En effet, les protocoles vus plus haut se basaient sur l'hypothèse de pannes équiprobables, indépendantes les unes des autres et en nombre limité. Il s'agissait là de pannes d'arrêt. Toutefois, les auteurs de cette

proposition prenaient pour exemple le cas de régimes autoritaires où le gouvernement peut supprimer une quantité importante et ciblée de nœuds d'un réseau. Les réseaux à contenu adressable cités précédemment ne peuvent résister à ce type d'attaque. La proposition [31] nécessite toutefois un degré $O(\log N)$ et demande $O(\log^2 N)$ messages par recherche. Une version dynamique a été proposée dans [20]. Ce dernier utilise une topologie multi-papillon permettant d'atteindre un diamètre $O(\log N)$ tout en permettant de résister à la suppression d'une fraction constante de nœuds.

Plusieurs travaux se sont attaqués au problème de la proximité physique du réseau logique (voir chapitre 1.2.9). TOPLUS a poussé le travail d'optimisation commencé par Tapestry et Pastry concernant le rapprochement de la topologie physique par la topologie logique. Cette contribution propose d'organiser la topologie logique en fonction de la topologie physique en utilisant comme indication de mesure de proximité physique la longueur du préfixe commun de deux adresses physiques IP. Les nœuds logique du réseau sont regroupées en ensemble de nœuds proches physiquement, c'est-à-dire ayant un préfixe IP commun. Les nœuds de l'arbre sont regroupés récursivement en nœuds internes parents de l'arbre de plus de moins en moins proches physiquement et regroupant de plus en plus de nœuds du réseau. Un nœud fait donc partie d'une feuille et de tous les nœuds internes qui sont sur le chemin de cette feuille à la racine de l'arbre. Chaque nœud interne de l'arbre peut en plus décider de former un cache local afin d'accélérer les temps de réponse pour les requêtes populaires, tout en profitant de la rapidité due à la proximité physique. Tous les nœuds du réseau composant une feuille de l'arbre sont voisins les uns des autres. De plus, pour chaque nœud interne dont il fait partie, il a au moins un voisin dans l'ensemble des nœuds frères de ce nœud interne. Cela présente l'inconvénient de devoir limiter artificiellement la profondeur de l'arbre afin de limiter le degré des nœuds, qui a pour effet de rassembler des nœuds qui sont moins proches. TOPLUS est accompagné d'une étude de l'impact de la limitation du degré sur l'efficacité du système.

Un autre travail, LAND [3], permet d'assurer que le rapport entre le nombre de sauts logiques pour relier deux nœuds et le nombre de sauts nécessaires pour les relier dans le réseau physique peut être amené aussi proche de 1 que désiré. Toutefois, LAND nécessite pour cela de supposer des contraintes fortes sur la répartition physique des nœud connecté au réseau logique. LAND se base, comme Tapestry [105] et Pastry [88], sur des nœuds maintenant des voisins à $\log N$ niveaux différents, permettant d'augmenter à chaque saut la taille du préfixe commun au nœud courant et à la destination. Cela lui permet d'assurer un degré moyen logarithmique, pour un nombre de sauts logarithmique. La distance parcourue dans le réseau logique n'excède pas la distance parcourue dans le réseau physique d'un rapport supérieur à 1ϵ. L'arrivée d'un nœud dans LAND demande $O(\log^2 n)$ messages. LAND propose une extension avec super-nœuds qui permet d'assurer un rapport $2 + \epsilon$ et un degré moyen constant (degré entrant moyen logarithmique pour les super-nœuds).

3.4 Optimisations

Parmi les systèmes que nous venons de lister, un certain nombre d'entre eux proposent des optimisations qui ne sont pas uniquement valables spécifiquement pour le système. Nous allons ici détailler dans cette partie les optimisations dont peuvent bénéficier tous les systèmes décentralisés structurés, et les éventuelles contraintes que ces optimisations engendrent. La plupart de ces optimisations sont inspirées de l'algorithmique répartie et ne sont donc pas complètement nouvelles.

3.4.1 Les multiples dimensions ou utiliser un alphabet β-aire

La plupart de systèmes décentralisés structurés sont décrits en identifiant les nœuds sur un alphabet binaire. CAN [83] a proposé d'améliorer les performances des réseaux à contenu adressable, concernant le nombre de sauts à effectuer pour trouver un nœud. L'augmentation du nombre de dimensions de 2 à β est équivalent à passer d'un alphabet binaire à un alphabet β-aire. Une autre manière de voir cette optimisation est de considérer que le degré reste binaire mais que les bits sont traités par groupe de $\log_2 \beta$. Augmenter la taille de l'alphabet a pour effet d'augmenter le *degré* de 2 à β pour un *nombre de sauts* diminuant d'un rapport $\log_2 \beta$.

3.4.2 Maintien de plusieurs réseaux logiques, ou comment utiliser plusieurs réalités d'un même monde

CAN [83] propose de maintenir plusieurs réseaux logique à la fois en attribuant à chaque nœud plusieurs identifiants, au moyen de plusieurs tables de hachage réparties. Ces réseaux logiques correspondent à autant de structures différentes liant les mêmes nœuds, ils ont été baptisés «réalités» par CAN. Chaque nœud multiplie alors son *degré* par r, qui est le nombre de table de hachages réparties utilisées. À chaque réalité correspond en effet un réseau logique, et pour chaque nœud des voisins spécifiques.

Cela permet ainsi, au moment de renvoyer un message, d'augmenter le nombre de choix possibles pour le prochain nœud, puisqu'un nœud connaît les coordonnées de chacun de ses voisins dans chaque réseau logique. Cette optimisation diminue donc le *nombre de sauts*.

Le *nombre de clés* attribuées à chaque nœud augmente aussi d'un facteur r puisque chaque objet est publié une fois dans chaque réseau logique. Cela permet de rechercher la clé sur un réseau logique choisi aléatoirement et, en cas d'impossibilité, de rechercher cette clé en changeant de réseau logique. Cette redondance augmente donc la *disponibilité des clés*.

Enfin, l'augmentation du nombre de voisins augmente aussi la *tolérance aux pannes du routage* : un nœud ayant perdu tous ses voisins dans un réseau logique peut utiliser un autre réseau logique pour retrouver de nouveaux voisins dans son réseau logique. Le nombre de chemins permettant d'atteindre un destinataire augmente aussi avec le nombre de réseau logique.

3.4.3 Routage vers le voisin le plus rapide

Lorsque plusieurs plus courts chemins vers un nœud existent, CAN [83] propose d'améliorer l'efficacité du routage en choisissant, parmi ces chemins, celui dont le prochain nœud est le plus proche (voir chapitre 1.2.9) : cette optimisation a pour but de diminuer le *délai par saut*. Comme nous l'avons déjà vu (voir chapitre 1.2.9), trouver une bonne mesure de proximité n'est pas trivial. Une mesure simple actuellement utilisée est la mesure du temps d'aller-retour d'une requête vers le nœud (RTT). Si l'on accepte l'hypothèse que des nœuds dont les adresses physiques sont proches seront proches dans le réseau physique (comme c'est souvent le cas dans le réseau IP), il est aussi possible d'évaluer la proximité deux nœuds grâce à la longueur du préfixe commun à leur adresse physique respective.

De manière générale, en diminuant le délai à chaque saut d'une requête, on diminue aussi le *délai de cette requête*. Notons que cette optimisation peut fausser l'utilisation de protocoles qui tiendraient compte de la topologie physique de manière plus fine (et donc pas seulement à un saut).

3.4.4 Redondance dans la responsabilité de clés

CAN [83] propose d'augmenter le nombre p de nœuds responsables d'une zone de clés. C'est une autre manière de parler de nœuds virtuels. Le *nombre de clés* attribuées à chaque nœud augmente donc d'un facteur p tandis que la *disponibilité des clés* augmente. Pour cela, il suffit que chaque nœud maintienne une liste des nœuds frères responsables de la même zone de clés. Lorsque ce nombre de nœuds frères dépasse un certain nombre fixé, la zone est divisée équitablement en deux parties, et les nœuds frères se répartissent équitablement la gestion des deux zones.

3.4.5 Multi-publication de clés

Afin d'augmenter la *résistance aux pannes* des nœuds, CAN [83] propose d'associer à un réseau à contenu adressable t tables de hachages réparties au lieu d'une seule, afin de publier t fois chaque objet dans le réseau logique. Le même effet peut être obtenu par la concaténation de quelques chiffres différents à l'objet avant hachage, afin d'obtenir plusieurs identifiants de nœuds. Cela permet d'associer chaque objet à plusieurs clés. À chaque publication, une clé est enregistrée t fois plus que pour l'utilisation d'une seule table de hachage répartie. À chaque recherche d'un objet, un nœud a le choix parmi t destinations associées chacune à une clé différente. Le *nombre de clés* attribuées à chaque nœud augmente donc aussi d'un facteur t en moyenne. Cette optimisation permet une *tolérance aux pannes* moins lourde que celle proposée par plusieurs réseaux logiques. Elle n'augmente toutefois que la *disponibilité des clés* et non le nombre de chemins pour les atteindre ni le degré de chaque nœud.

3.4.6 Équilibrage de charge à l'arrivée et au départ du réseau

À l'arrivée d'un nouveau nœud dans le système, il est possible de *rééquilibrer la charge des zones de clés* parmi les nœud. En effet, les nouveaux nœuds ont des identifiants répartis uniformément sur l'espace de clés *en moyenne* et les zones de responsabilités de clés sont de même taille *en moyenne*. Dans des systèmes comme Chord [96], où le degré d'un nœud est lié au nombre de clés dont il est responsable, cela permet de répartir la charge du trafic de contrôle *en moyenne*. Toutefois, la probabilité que des zones de clés n'aient pas la même taille n'est pas négligeable, comme les travaux présentés dans l'état de l'art le montrent souvent.

Il est possible de mieux répartir la charge due au nombre de clés et/ou au degré de plusieurs manières :

- modifier l'endroit où s'insère un nouveau nœud, comme le proposent CAN [83] et D2B [33] (voir chapitre 4) ;
- choisir un nœud spécifique pour remplacer un nœud qui part, comme le proposent CAN [83] et D2B [33] (voir chapitre 4) ;
- utiliser un algorithme qui vérifie régulièrement la bonne répartition de la charge sur chaque nœud.

Nous allons voir succinctement comment la première et la seconde méthode peuvent être mise en œuvre, tandis que la troisième peut s'inspirer des deux premières. Ces méthodes peuvent être combinées pour en améliorer l'efficacité. Toutefois, la troisième méthode nécessite un trafic de contrôle régulier.

Équilibrage de charge à l'arrivée dans le réseau

CAN [83] a d'abord relevé qu'il est facile pour un nouveau nœud de vérifier auprès du responsable de son identifiant initial si un de ses voisins est plus chargé. Le nouveau nœud choisit alors un identifiant géré par ce voisin.

Plusieurs systèmes ont ensuite proposé la simple sélection de plusieurs identifiants logiques à la connexion afin d'effectuer plusieurs tentatives d'insertion et de choisir l'identifiant qui gérera le plus de clés. Nous présentons au chapitre 4.3.2 une analyse pour choisir au mieux l'identifiant d'un nouvel arrivant, ainsi qu'une proposition pour augmenter à coût nul le choix d'identifiants pour un tel arrivant.

Ces différentes techniques peuvent être combinées pour permettre aux nouveaux nœuds de choisir un identifiant qui donne les meilleurs résultats en terme de rééquilibrage de charge.

Équilibrage de charge au départ du réseau

De manière similaire, lorsqu'un nœud quitte le système en prévenant le réseau, il est possible de lui chercher un remplaçant qui est peu chargé. Ce remplaçant est recherché parmi l'ensemble de nœuds qui peuvent être responsables de ses clés, les voisins du nœud partant souvent). Parmi ces nœuds, celui qui a la plus petite zone de clés est choisi pour être responsable des clés du nœud partant. CAN [83] a proposé le maintien d'une structure d'arbre afin de pouvoir rééquilibrer à chaque départ le nombre de clés dont

est responsable chaque nœud. D2B [33] adapte cette méthode au graphe de de Bruijn (voir chapitre 4.1.4).

De cette manière, il est possible d'effectuer un rééquilibrage à chaque fois qu'un nœud quitte le réseau prévenant ses voisins. Toutefois, cette méthode ne permet pas de gérer le cas des nœuds tombant en panne ou ceux qui quittent le système sans prévenir le réseau.

3.4.7 Inspirer la topologie logique de la topologie physique

Lors de la création du réseau avec un certain nombre de nœuds, CAN [83] relève qu'il est possible d'organiser ces nœuds de façon à ce que les voisins dans le réseau logique soient proches dans le réseau physique (voir chapitre 1.2.9), de façon à diminuer le *délai par saut*, et donc le *délai par requête*. Toutefois, le maintien de cette proximité au cours de l'arrivée et du départ des nœuds dans le réseau est loin d'être trivial. Des systèmes comme Tapestry [105] et Pastry [88] font choisir aux nœuds les voisins estimés les plus proches dans le réseau physique. Nous verrons au chapitre 4.3.3 une méthode pour augmenter le choix d'identifiants dans ce but.

Notons que l'utilisation de ces optimisations peut faire l'objet d'un compromis avec l'équilibrage de la charge à l'arrivée et au départ des nœuds. Rappelons toutefois que le rapprochement des réseaux logique et physique souffre des faiblesses que nous avons exposées au chapitre 1.2.9.

3.4.8 Mise en cache et réplication des clés

De manière standard, les clés sont publiées sur un seul nœud, ou sur un groupe de nœuds proches dans le réseau logique. Cela peut engendrer des problèmes de charge pour des nœuds responsables d'une clé très populaire et leurs voisins. Deux solutions sont envisagées par CAN [83] :

- la mise en cache des clé ;
- la réplication des clés.

De manière générale, lorsque la copie ou la réplication est utilisée, une étape de vérification au routage est ajoutée afin de vérifier, avant l'arrivée du message au destinataire, si la clé recherchée n'est pas enregistrée sur le nœud courant. Dans ces deux cas, il est judicieux d'adapter le nombre de copie d'une association *clé=nœud* à la popularité de la clé.

Mise en cache des clés

La mise en cache permet à un nœud de garder en mémoire les résultats des requêtes qui sont passées par lui. L'avantage de la mise en cache est que le nombre de nœuds pouvant fournir une clé et les nœuds associés augmente naturellement (sans intervention active) avec sa popularité. Toutefois, le coût supplémentaire est pour chaque nœud une quantité de mémoire bornée qui s'ajoute à la mémoire nécessaire pour gérer ses clés. De plus, cette technique impose que les réponses aux recherches passent par le réseau logique, et les clés sont donc recopiées sur un ensemble connexe de nœuds du réseau. La

charge due à ces clés est donc potentiellement répartie sur la périphérie de cet ensemble de nœuds, rendant inutiles les clés gérées par les nœuds au centre de cet ensemble. La politique de cache peut être renforcée en fixant une date d'expiration pour les copie de clés selon la distance qui les sépare de leur responsable.

Réplication des clés

La réplication de clés permet à un nœud de définir un certain nombre de responsables pour une clé. Cela répartit ainsi la charge due aux clés populaires. Le coût supplémentaire est alors pour chaque nœud une quantité de mémoire bornée qui s'ajoute à la mémoire nécessaire pour gérer ses clés. Choisir les nœuds responsable d'une association *clé=nœud* en fonction de la clé permet de profiter des propriétés naturelles d'équilibre de charge concernant le nombre de clés par nœud. Une réplication judicieuse permet de plus de diminuer le nombre de sauts nécessaire à un message pour trouver une association *clé=nœud*.

Dans les systèmes où l'acheminement des messages trouve l'un des α nœud le plus proche de la clé (α étant un paramètre du système), les clés doivent alors être répliquées sur un nombre $\beta \geq \alpha$ de nœuds les plus proches. C'est le cas de Pastry [88] ou Kademlia [70], où β est constant. Dans ces systèmes, selon le paramètre β choisi, des nœuds responsables de clés peuvent avoir des voisins entrants qui sont eux-mêmes tous responsables pour cette clé. La copie de l'association *clé=nœud* sur de tels nœuds ne permet donc ni de diminuer la charge des nœuds responsables de l'association, ni de diminuer le nombre de sauts nécessaires à l'obtention d'une réponse.

Une méthode proposée par Tapestry [105] permet d'adapter le nombre de réplication d'une association *clé=nœud* à la popularité de cette clé. Il s'agit alors pour chaque nœud de trouver la source d'un éventuel trafic important, dû à une clé particulière. Chaque nœud u responsable d'une association *clé=nœud* vérifie alors si le nombre de messages provenant d'un voisin v donné pour une clé x dépasse un borne γ. Dans l'affirmative, un message est envoyé à v afin qu'il exécute lui-même cet algorithme pour la clé x. Cet algorithme s'arrête au nœud source de ce trafic, c'est-à-dire le nœud w pour lequel aucun voisin n'a envoyé plus de γ messages pour la clé x. Un message est alors envoyé au nœud responsable u afin qu'il envoie une copie de l'association x=*nœud* à w.

Une autre technique de réplication, prenant en compte la collision de clés, est proposée par Broose [36, 101] sous le nom de hachage mixte. Elle consiste à répliquer les associations liées à une clé sur des nœuds différents, tous descendants d'un même arbre. Dans cet arbre enraciné au responsable de la clé, les associations sont recopiées sur des nœuds dont l'identifiant dépend de l'identifiant de la clé, de l'objet associé à cette clé, et du niveau de réplication (nombre d'objets associés à cette clé). Le nombre d'associations *(clé,objet)=nœud* enregistrées sur un nœud est borné (quelque soit le clé et l'objet), limitant ainsi la charge due aux clés gérées par un nœud. Lorsque cette borne est dépassée, les associations sont répliquées sur les fils appropriés dans l'arbre, et le niveau de réplication augmente.

L'avantage de la réplication par rapport à la mise en cache est donc que le nœud responsable d'une clé maîtrise le nombre et l'emplacement des copies de ses clés. De

plus, elle ne nécessite pas de retour des réponses *via* le même chemin que la recherche.

3.5 Comparatif des différents systèmes

Nous avons présenté dans ce chapitre un aperçu des contributions qui ont donné naissance aux réseaux à contenu adressable. En particulier, si la plupart de ces systèmes permettent un degré logarithmique, chacun a participé à l'amélioration de différentes bornes, principalement :

- le degré ;
- le nombre de messages nécessaire à l'insertion ;
- l'engorgement, c'est-à-dire le nombre de requêtes qui peuvent passer par un nœud.

Nous présentons ici un tableau récapitulatif des performances *moyennes* des systèmes vu dans de chapitre. Dans ce tableau :

- n est le nombre courant de nœuds présents dans le réseau ;
- N est le nombre de nœuds maximal dans le réseau ;
- α le nombre de messages envoyés à chaque étape d'une recherche ;
- β est le nombre de chiffres de l'alphabet utilisé ;
- γ (pour Kademlia) le nombre de voisins dans chaque sous-arbre d'un nœud.
- λ (pour Broose) le nombre maximal de voisins des ensembles gauches et droits pour chaque nœud.

Rappelons que le diamètre de Kademlia et Broose n'est pas le même que le nombre de messages nécessaire à une recherche de clé puisque α messages sont envoyés à chaque saut lors d'une recherche (toutefois, un nœud déjà sollicité ne sera pas sollicité à nouveau).

	diamètre	degré	insertion [1]	engorgement
CAN [83]	$O(\beta n^{1/\beta})$	$O(\beta)$	$O(\beta n^{1/\beta})$	$O(\beta n^{1/\beta-1})$
Chord [96]	$O(\log n)$	$O(\log n)$	$O(\log^2 n)$	$O(\log n/n)$
Tapestry [105]	$O(\log n/\log \beta)$	$O(\beta \log N/\log \beta)$	$O(\log n/\log \beta)$	$O(\log n/(n \log \beta))$
Pastry [88]	$O(\log n/\log \beta)$	$O(\beta \log N/\log \beta)$	$O(\log n/\log \beta)$	$O(\log n/(n \log \beta))$
Viceroy [68]	$O(\log n)$	$O(1)$	$O(\log n)$	$O(\log n/n)$
Kademlia [70]	$O(\log n/\log \beta)$	$O(\gamma \log n/\log \beta)$	$O(\alpha \log n/\log \beta)$	
Broose [101]	$O(\log(n/\lambda)/\log \beta)$	$O(\lambda\beta)$	$O(\lambda + \alpha \log n/\log \beta)$	

Pour les systèmes qui livrent une analyse *a.f.p.*, voici un tableau récapitulatif de leur caractéristiques :

	diamètre	degré	insertion [1]	engorgement
Chord [96]	$O(\log n)$	$O(\log^2 N)$	$O(\log^2 n)$	
Viceroy [68]	$O(\log n)$	$O(\log n)$	$O(\log^2 n)$	$O(\log^2 n/n)$
Broose [100]	$O(\log(n/\gamma)/\log \beta)$	$O(\lambda\beta)$		

[1] nombre de messages nécessaires

Pour conclure, les propriétés des systèmes pair-à-pair décentralisés se révèlent intéressantes lorsque l'on peut facilement associer une clé à un objet, comme pour les DNS [19]. L'utilisation du hachage permet la répartition de la charge parmi les nœuds, par exemple en répartissant les identifiants dans le réseau, ou la gestion des clés parmi les nœud. Cette répartition ajoutée à l'efficacité de l'acheminement des messages est l'un des apports principaux des réseaux à contenu adressable. De manière générale, ils se révèlent particulièrement efficaces lorsqu'un nom unique peut être associé aux objets mis à disposition par le réseau : nom de domaine, côte (ISSN d'un livre), etc.

Chapitre 4

Le protocole D2B

Dans ce chapitre, nous allons présenter D2B [33], une proposition de réseau à contenu adressable basé sur un graphe de de Bruijn et qui permet de router tout message en un *nombre de sauts logarithmique* avec un *degré moyen constant*. Parmi les topologies proposées pour créer des réseaux à contenu adressable, nous avons vu les tores, les hypercubes, les papillons, etc. Le graphe de de Bruijn est connu pour permettre la construction de réseaux de degré constant et de faible diamètre [11]. Ces propriétés intéressantes donnent au graphe de de Bruijn des atouts certains pour la création d'un réseau structuré dynamique.

Plusieurs travaux ont découvert au même moment que D2B l'intérêt de la topologie de Bruijn pour la création de réseaux à contenu adressable [1, 33, 57, 72]. Koorde [57] propose un espace de clé $[0, 2^m - 1]$ et une connexion de chaque nœud d'identifiant x aux nœuds d'identifiant $2x \mod 2^m$ et $(2x+1) \mod 2^m$ comme décrit sur la figure 4.1(a). [72] utilise un espace de nommage $[0, 1[$, où chaque nœud x a pour voisins les nœuds $x/2$ et $x/2 + 1/2$, comme le montre la figure 4.1(b). L'intuition derrière ces deux travaux est la même : un décalage vers la droite d'un nombre binaire correspond à une multiplication par 2 dans [57] tandis qu'un décalage vers la gauche correspond à une division par 2 dans [72]. [1] décrit la création d'un réseau logique dynamique à partir d'une topologie statique. Un exemple est donné pour le graphe de de Bruijn exhibant ses bonne propriétés en terme de degré et de diamètre.

Pour finir, nous avons déjà parlé au chapitre précédent du protocole Broose [101], proposé à la suite de D2B, qui s'est inspiré de la topologie de de Bruijn pour proposer un système basé sur une recherche itérative des nœuds les plus proches de la destination, comme Kademlia [70].

C'est pourquoi D2B, basé sur le graphe de de Bruijn vérifie plusieurs propriétés résumées ci-dessous :

- le degré moyen du réseau logique maintenu par D2B est constant ;
- une recherche sur une clé débutée sur un nœud quelconque atteint le nœud responsable de la clé après au plus $O(\log n)$ a.f.p., où n est le nombre de nœuds insérés dans le réseau ;
- le nombre de messages nécessaires à l'arrivée et au départ d'un nœud dans le

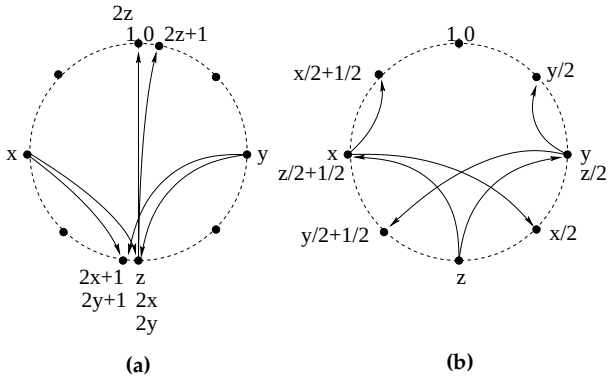

FIG. 4.1 – (a) connexions aux nœuds d'identifiants multipliés par 2 par rapport aux (b) connexions par division par deux (comme dans [72])

réseau est constant en moyenne. Le temps de connexion et de déconnexion d'un nœud au réseau est donc constant en moyenne. La mise à jour nécessaire à l'arrivée ou au départ d'un nouveau nœud dans le réseau prend un temps au plus $O(\log n)$ a.f.p. ;

- chaque nœud est en moyenne responsable de $|K|/n$ clés, et $|K| \log_2 n/n$ clés a.f.p., où K est le nombre de clés publiées dans le réseau ;
- l'engorgement moyen d'un nœud, c'est-à-dire le nombre de chemins passant par un nœud divisé par le nombre de chemins d'un nœud quelconque à une clé quelconque, est $O(\log n/n)$, et il ne dépasse pas $O(\log^2 n/n)$ a.f.p.

Nous proposons aussi de compléter la version binaire de D2B par une version d-aire (voir section 3.4.1), avec $d > 2$, où chaque nœud et chaque clé a un identifiant en base d, et où le réseau utilise comme base un graphe de de Bruijn de dimension d. Le degré moyen de cette version d-aire est alors $O(d)$, pour un diamètre moyen $O(\log n/\log d)$. Cela permet d'effectuer un compromis entre temps de connexion et de déconnexion d'une part, et temps de publication et de recherche d'autre part. Augmenter le degré permet accessoirement d'améliorer la tolérance aux pannes. Notons que si $d = O(\log n)$, alors le diamètre moyen est $O(\log n/\log\log n)$. Si l'on connaît la probabilité de déconnexion des nœuds ou que l'on a un nombre de sauts moyen à assurer, alors il est donc possible de calibrer le degré grâce à cette propriété.

4.1 Le réseau à contenu adressable D2B

Nous allons décrire ici D2B, les méthodes d'arrivée et de départ dans le système et les méthodes de recherche et de publication. D2B dépend d'un paramètre $d \geq 2$ qui est le degré du graphe. Nous allons décrire une version binaire de D2B ($d = 2$) pour plus de clarté, une version à d dimensions sera décrite dans la section 4.3.

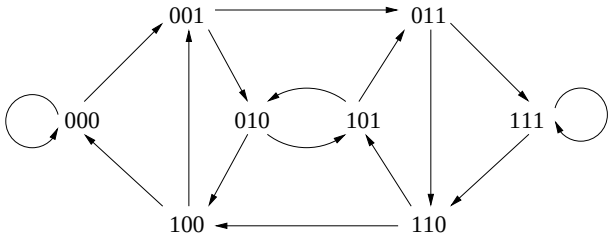

FIG. 4.2 – Le graphe de de Bruijn $B(2,3)$

4.1.1 Le graphe de de Bruijn

Le graphe statique sur lequel est basée la version à d dimensions de D2B ($d \geq 2$) est le graphe de de Bruijn $B(d,k)$ pour $k \geq 1$. Ce graphe a été défini dans [22]. Il s'agit d'un graphe orienté dont les nœuds sont des chaînes de longueur k sur l'alphabet $\{0,\ldots,d-1\}$. Des arcs lient le nœud $x_1x_2\ldots x_k$ aux k nœuds $x_2\ldots x_k\alpha$ pour $\alpha \in \{0,\ldots,d-1\}$. La figure 4.2 montre le graphe $B(2,3)$. Notons qu'il existe des boucles sur tous les nœuds $\alpha\ldots\alpha$, $\alpha \in \{0,\ldots,d-1\}$ et que $B(d,k)$ n'est pas sommet transitif. Toutefois, nous verrons que cela n'a aucun impact sur les performances de D2B, qui répartit uniformément la charge sur les nœuds. $B(d,1)$ est le graphe complet de d nœuds avec une boucle sur chaque nœud.

$B(d,k)$ comporte d^k nœuds, un degré entrant et sortant d, et un diamètre k. Un routage de $x_1\ldots x_k$ vers $y_1\ldots y_k$ peut se faire en empruntant la route

$$x_1\ldots x_k \rightarrow x_2\ldots x_ky_1 \rightarrow \ldots \rightarrow x_ky_1\ldots y_{k-1} \rightarrow y_1\ldots y_k.$$

Une route plus courte peut être obtenue en cherchant la plus longue suite de chiffres qui soit à la fois suffixe de $x_1\ldots x_k$ et préfixe de $y_1\ldots y_k$. Soit $x_i\ldots x_k = y_1\ldots y_{k-i+1}$ cette suite, alors le plus court chemin de $x_1\ldots x_k$ à $y_1\ldots y_k$ est :

$$x_1\ldots x_k \rightarrow x_2\ldots x_ky_{k-i+2} \rightarrow \ldots \rightarrow x_ky_{k-i+2}\ldots y_{k-1} \rightarrow y_1\ldots y_k.$$

4.1.2 Description générale de D2B

La version de D2B à 2 dimensions utilise un espace de clés $\kappa = \{0,\ldots,2^\ell-1\}$, qui est l'ensemble des chaînes binaires de longueur ℓ. Tous les participants de D2B connaissent une même fonction de hachage h qui leur permet de hacher les identifiants d'objets dans l'espace de nommage K. Chaque nœud de D2B a un identifiant constitué d'au plus ℓ bits. La fonction h' qui assigne les identifiants aux nœuds sera définie plus tard. D2B autorise donc un maximum de 2^ℓ nœuds. Remarquons que ce n'est pas une réelle limite dès lors qu'on choisit $\ell = 128$ ou 256, puisque cela assure que le nombre de clés est plus grand que le nombre d'adresses autorisées par IPv6 par exemple. Nous définirons la valeur d'un nœud u d'identifiant $x_1\ldots x_k, x_i \in \{0,1\}$ comme étant $val(u) = 2^{\ell-k}\sum_{i=1}^{k} x_i2^{k-i}$.

Définition 4.1 *Un ensemble universel de préfixes est un ensemble S de mots binaires tel que pour tout mot de longueur infinie $\omega \in \{0,1\}^*$, il existe un unique mot dans S qui soit préfixe de ω. L'ensemble vide est aussi un ensemble universel de préfixes. L'ordre lexicographique permet d'ordonner les nœuds entre eux.*

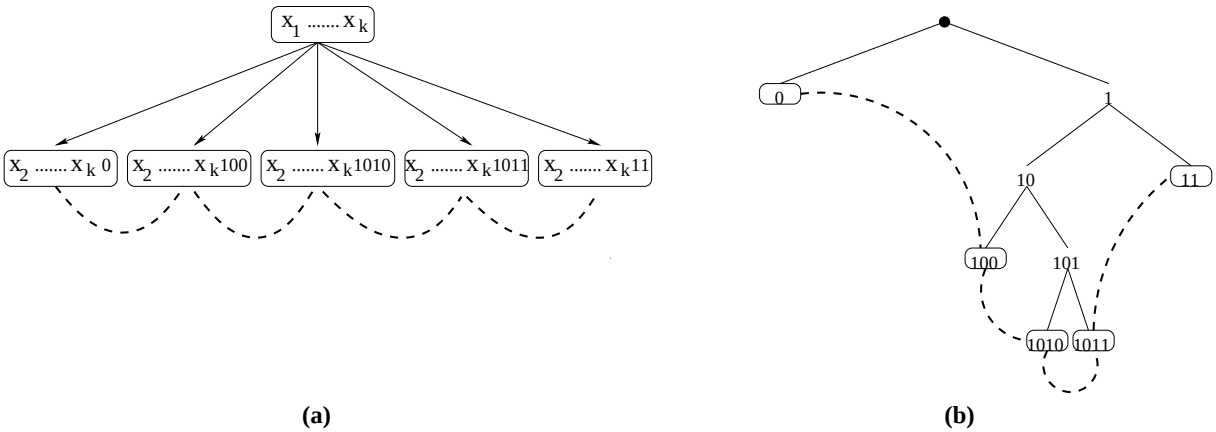

FIG. 4.3 – Connexions vers les fils (traits pleins) et jumeaux (traits pointillés)

Par exemple, $\{0, 100, 1010, 1011, 11\}$ est un ensemble universel de préfixes. D2B assure qu'à tout moment, l'ensemble des identifiants de tous les nœuds alors dans le réseau est un ensemble universel de préfixes.

Répartition des clés

Un nœud u d'identifiant $x_1 \dots x_k$ est responsable de toutes les clés entre $val(u)$ et $2^{m-k}(1 + \sum_{i=1}^{k} x_i 2^{k-i}) - 1$. Plus intuitivement, les clés dont l'identifiant est $\kappa_1 \dots \kappa_\ell$ sont gérées par le nœud $x_1 \dots x_k$ si et seulement si $x_1 \dots x_k$ est préfixe de $\kappa_1 \dots \kappa_\ell$. Ainsi, un nœud d'identifiant $x_1 \dots x_k$ est responsable de $2^{\ell-k}$ clés. Inversement, un nœud responsable de 2^q a un identifiant sur $\ell - q$ bits. Toutes les clés sont assignées puisque, par construction, les identifiants des nœuds forment un ensemble universel de préfixes.

Connexions de routage

À tout instant, le nœud d'identifiant $x_1 \dots x_k$ a soit un unique voisin sortant d'identifiant $x_2 \dots x_j$ avec $j \leq k$, soit plusieurs voisins sortants dont les identifiants sont de la forme $x_2 \dots x_k y_1 \dots y_m$ avec $1 \leq m \leq \ell - k + 1$. Dans ce second cas, l'ensemble des suites $y_1 \dots y_m$ forme un ensemble universel de préfixes. En particulier, si $x_2 \dots x_k y_1 \dots y_m$ est un voisin sortant de $x_1 \dots x_k$, aucun des identifiants $x_2 \dots x_k y_1 \dots y_i$, $i < m$ n'est au même instant présent dans le réseau. Dans la suite, un voisin sortant d'un nœud u est simplement nommé *fils* de u. Les fils d'un nœud u nommé $x_1 \dots x_k$ sont détaillés dans la figure 4.3(a). Dans cet exemple, le nœud $x_1 \dots x_k$ a cinq fils nommés $x_2 \dots x_k 0$, $x_2 \dots x_k 100$, $x_2 \dots x_k 1010$, $x_2 \dots x_k 1011$ et $x_2 \dots x_k 11$. Dans le réseau, aucun nœud n'a pour identifiant $x_2 \dots x_k 1$, $x_2 \dots x_k 10$ ou $x_2 \dots x_k 101$.

À l'inverse, à tout moment, le nœud d'identifiant $x_1 \dots x_k$ a des voisins entrants (appelés *pères*) soit d'identifiants de la forme $\alpha x_1 \dots x_j$ avec $\alpha \in \{0, 1\}$ et $j \leq k$, soit d'identifiants de la forme $\beta x_1 \dots x_k y_1 \dots y_m$, pour $\beta \in \{0, 1\}$ et $1 \leq m \leq \ell - k - 1$.

Dans le second cas, l'ensemble de suites $y_1 \dots y_l$ forme un ensemble universel de préfixes. Remarquons que les deux formes de pères peuvent coexister simultanément, mais pour $\alpha \neq \beta$.

Remarque 4.1 À cause des boucles sur les nœuds $0 \dots 0$ et $1 \dots 1$, les connexions avec les fils et les pères sont légèrement différentes pour ces deux nœuds. Un nœud u d'identifiant $\alpha\alpha \dots \alpha, \alpha \in \{0,1\}$ a pour fils les nœuds $\alpha \dots \alpha y_1 \dots y_m, m \geq 1$, où $y_m = \overline{\alpha}$. L'ensemble d'identifiants $y_2 \dots y_m$ est un ensemble universel de préfixes. Les pères du nœud u ont pour identifiant soit $\overline{\alpha}\alpha \dots \alpha$ avec $j \leq k$ chiffres α, ou $\overline{\alpha}\alpha\alpha \dots \alpha y_1 \dots y_m$, avec k chiffres α et $m \geq 1$. Dans le second cas, l'ensemble de suites $y_1 \dots y_m$ forme un ensemble universel de préfixes.

Connexions jumelles

En plus des connexions avec les fils et des pères, chaque nœud v a pour voisins deux jumeaux (voire figure 4.3(b)) : le jumeau inférieur est le nœud x ayant la plus grande valeur $val(x) < val(v)$ tandis que le jumeau supérieur est le nœud y ayant la plus petite valeur $val(y) > val(v)$. Remarquons que tous les fils d'un nœud u sont liés par des *connexions jumelles* comme indiqué dans la figure 4.3(a) et que les nœuds sont connectés en anneau par leur connexion jumelle (les nœuds de plus grand et de plus petit identifiant sont respectivement le nœud jumeau inférieur du plus petit et le jumeau supérieur du plus grand).

Protocole de routage

Le routage dans D2B s'effectue de manière similaire au graphe de de Bruijn. Plus précisément, soit $x_1 \dots x_k$ l'identifiant d'un nœud u dans D2B, et soit $\kappa = \kappa_1 \dots \kappa_\ell$ une clé quelconque. Soit S la plus longue chaîne binaire qui soit suffixe de $x_1 \dots x_k$ et préfixe de $\kappa_1 \dots \kappa_\ell$, S pouvant être $\emptyset$. Si $S = x_1 \dots x_k$, u est responsable de κ. Sinon, si u a un fils unique v d'identifiant $x_2 \dots x_j$ pour $j \leq k$, alors la requête pour la clé κ est envoyée à ce nœud. Si u a plusieurs fils, alors la requête pour κ est envoyée au fils d'identifiant $x_2 \dots x_k y_1 \dots y_l$ tel que $Sy_1 \dots y_l$ est un préfixe de $\kappa_1 \dots \kappa_\ell$. D'après la propriété d'ensemble universel de préfixes, il existe un unique fils qui corresponde.

Publication de clés

Un nœud u du système qui désire publier un objet calcule la clé $\kappa \in \mathcal{K}$ correspondante en utilisant la fonction de hachage h. Il envoie ensuite un message de publication à travers le réseau. Le format de ce message est $\langle publication, @_u, \kappa, \mathcal{O} \rangle$, où $@_u$ est l'adresse physique de u (par exemple l'adresse IP) et $\mathcal{O}$ l'objet recherché. Le message est routé comme un message de recherche, selon la représentation binaire de κ. Quand le nœud responsable de κ reçoit un message de publication $\langle publication, @_u, \kappa, \mathcal{O} \rangle$, il place $@_u$ dans l'entrée κ de sa table de clés, c'est-à-dire la table comportant les correspondances entre les adresses physiques des objets et les clés associées à ces objets.

4.1.3 Insertion dans le réseau

Comme pour la plupart des réseaux à contenu adressable (voir chapitre 1.2.5), nous considérons que des adresses physiques de nœuds présents dans le réseau sont publiquement disponibles (par exemple par le biais d'un site HTTP). Ainsi, on considère qu'un nœud désirant rejoindre le réseau connaît au moins un contact déjà inséré dans le réseau, nommé passerelle. L'opération d'insertion dans le réseau se décompose en trois parties :

- obtention d'un identifiant D2B ;
- nouvelle répartition des clés ;
- mise à jour des connexions (pour le routage comme pour les jumeaux).

Soit u un nœud s'insérant dans le réseau, et soit v une passerelle pour D2B, c'est-à-dire que u connaît l'adresse physique $@_v$ de v.

Obtention d'un identifiant D2B

Pour s'insérer dans le réseau, un nouveau nœud u choisit uniformément aléatoirement un identifiant temporaire qui est une chaîne de ℓ bits $u_1 \dots u_\ell$. u contacte alors sa passerelle v et un message d'insertion est envoyé à partir de v dans le réseau. Le format de ce message est $\langle \mathit{insertion}, @_u, u_1 \dots u_\ell \rangle$, où $@_u$ est l'adresse physique de u. Ce message est routé comme un message de recherche, où $u_1 \dots u_\ell$ joue le rôle de la clé. Le message d'insertion arrive donc ainsi à un nœud w d'identifiant $x_1 \dots x_k$ qui est responsable de la clé $u_1 \dots u_\ell$, c'est-à-dire que $x_1 \dots x_k$ est préfixe de $u_1 \dots u_\ell$. Si $k = \ell$, c'est-à-dire si $x_1 \dots x_k = u_1 \dots u_m$, l'insertion échoue et u doit choisir un autre identifiant temporaire. Dans un réseau de n nœuds, un tel échec arrive avec une probabilité au plus $n/2^\ell$, qui est virtuellement nulle même pour un milliard de nœud, pour $\ell = 128$ ou 256. Ainsi, nous considérerons que $k \leq \ell$ (en pratique, $k \ll \ell$). Le nœud u prend alors l'identifiant $x_1 \dots x_k 1$ tandis que w transforme son identifiant en $x_1 \dots x_k 0$. Cette opération est nommée *extension d'identifiant*.

À cet instant, seul le nœud w connaît u. Afin de préserver la cohérence, w continue de jouer le rôle de $x_1 \dots x_k$ jusqu'à la fin de l'opération d'insertion.

Nouvelle répartition des clés

Dans la table de clés de w, toutes les clés qui ont $x_1 \dots x_k 1$ pour préfixe sont transférées de w au nouveau nœud u. Seules les clés correspondant à des objets proposés sur le réseau sont transférées. Ainsi, le nombre de clés transférées est bien inférieur à $2^{\ell-k-1}$, qui est la taille de l'intervalle de clés géré par w. Toujours afin de préserver la cohérence, le nœud w garde une copie de la table de clés correspondant aux clés transférées jusqu'à la fin de l'opération d'insertion.

Mise à jour des connexions

a) Connexions avec les fils : le nœud u récupère de w les adresses physiques de tous les fils de w. Considérons ici les deux cas, selon que w a une boucle sur lui, c'est-à-dire

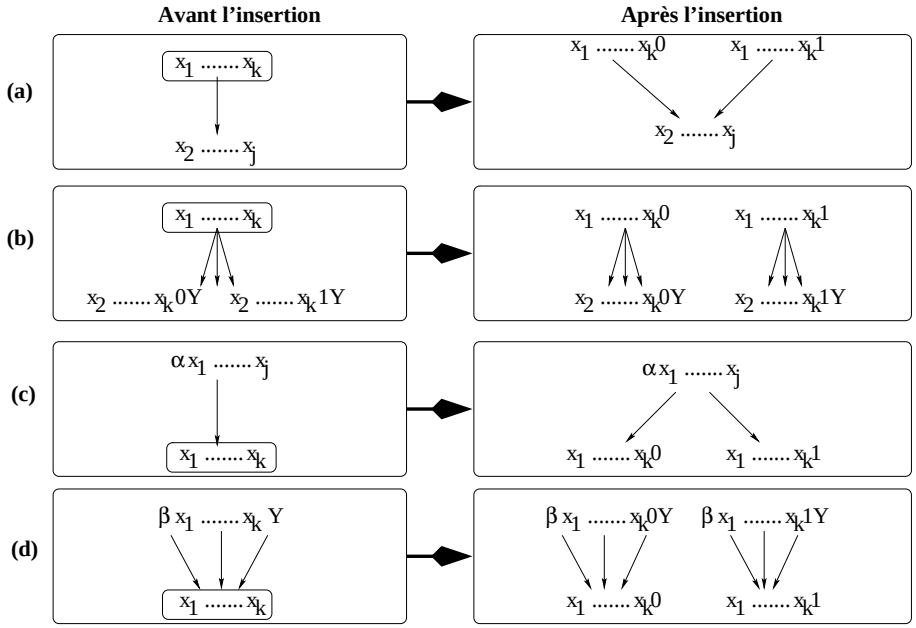

FIG. 4.4 – Mise à jour des connexions après une arrivée

selon que w a pour identifiant soit $00\ldots0$ soit $11\ldots1$ ou pas.

a.1) Cas général : il existe deux indices $i \neq j$ tel que $x_i \neq x_j$. Considérons les deux cas exclusifs suivants : (a) si w a un fils unique nommé $x_2 \ldots x_j$, $j \leq k$, ce fils devient le fils unique de u et reste le fils de w (voir figure 4.4(a)) ; (b) si w a plusieurs fils, d'identifiants de la forme $x_2 \ldots x_k y_1 \ldots y_l$, $l \geq 1$ (voir figure 4.4(b)), ceux qui satisfont $y_1 = 1$ deviennent les fils de u. Ils sont informés par w qu'il n'est plus leur père et qu'il doit être remplacé par u. Les fils de w tels que $y_1 = 0$ restent fils de w.

a.2) Cas spécifique : l'identifiant de w est $\alpha\alpha\ldots\alpha$, $\alpha \in \{0,1\}$. Il a alors un fils de la forme $\alpha\ldots\alpha y_1\ldots y_l$, $l \geq 1$ avec $y_1 = \overline{\alpha}$. Par extension d'identifiant, soit w soit u prend l'identifiant $\alpha\alpha\ldots\alpha\alpha$, tandis que l'autre prend l'identifiant $\alpha\alpha\ldots\alpha\overline{\alpha}$. Le nœud d'identifiant $\alpha\alpha\ldots\alpha\alpha$ prend $\alpha\alpha\ldots\alpha\overline{\alpha}$ comme fils unique, tandis que $\alpha\alpha\ldots\alpha\overline{\alpha}$ prend tous les nœuds $\alpha\ldots\alpha y_1\ldots y_l = \alpha\ldots\alpha\overline{\alpha}y_2\ldots y_l$ comme fils.

Remarque 4.2 il peut arriver qu'un nouveau nœud u doive se connecter aux fils de w et que certains des fils de w changent d'identifiant (par extension, comme on vient de le voir, ou par contraction, comme on le verra dans le chapitre 4.1.4) entre le début et la fin de l'insertion de u dans le réseau. u en est alors informé grâce à la connexion qu'il maintient avec w jusqu'à la fin de son insertion Il peut alors se connecter correctement à ses fils.

b) Connexions avec les pères : chaque père w' de w est informé de l'existence d'un nouveau nœud u d'identifiant $x_1 \dots x_k 1$, de l'adresse physique $@_u$ de ce nœud u, et de le nouvel identifiant $x_1 \dots x_k 0$ de w. Pour tout père w', D2B procède comme suit.

b.1) Cas général : il existe deux indices $i \neq j$ tel que $x_i \neq x_j$. Considérons les deux cas sous-suivants :
(a) si w' a pour identifiant $\alpha x_1 \dots x_j$, avec $j \leq k$ (voir figure 4.4(c)), w' prend u comme fils, et modifie l'identifiant de w dans sa table de routage. Ainsi, w' a un nouveau fils, et son degré augmente de 1 ;
(b) si w' a pour identifiant $\beta x_1 \dots x_k y_1 \dots y_l$ avec $l \geq 1$ (voir figure 4.4(d)), w' garde w comme fils si $y_1 = 0$, ou remplace w par u si $y_1 = 1$.

b.2) Cas spécifique : w a pour identifiant $\alpha \dots \alpha$, avec k chiffres $\alpha \in \{0, 1\}$. Encore une fois, par extension d'identifiant, un nœud prend comme identifiant $\alpha \dots \alpha\alpha$, tandis que l'autre prend pour identifiant $\alpha \dots \alpha\overline{\alpha}$. Deux sous-cas existent : (a) w a un père de la forme $\overline{\alpha} \dots \alpha$ avec $j \leq k$ chiffres α. Le nœud d'identifiant $\alpha \dots \alpha\alpha$ prend ce nœud comme père, tandis que le nœud d'identifiant $\alpha \dots \alpha\overline{\alpha}$ prend les deux nœuds $\alpha \dots \alpha\alpha$ et $\alpha \dots \alpha\overline{\alpha}$ pour pères. (b) w a des pères de la forme $\overline{\alpha}\alpha \dots \alpha y_1 \dots y_l$, avec k chiffres α et $l \geq 1$. Le nœud d'identifiant $\alpha \dots \alpha\alpha$ prend alors pour pères les nœuds tels que $y_1 = \alpha$, tandis que $\alpha \dots \alpha\overline{\alpha}$ prend pour pères les nœuds pour lesquels $y_1 = \overline{\alpha}$ ainsi que $\alpha \dots \alpha\alpha$.

c) connexions jumelles : Le nœud u récupère de w l'adresse physique de son jumeau supérieur, qui est l'ancien jumeau supérieur de w. Ce nœud est informé que son jumeau inférieur n'est plus w mais u. Le nouveau jumeau supérieur de w est simplement u et le jumeau inférieur de u est w.

4.1.4 Quitter le réseau

L'opération de départ s'effectue en trois étapes :
- remplacement du nœud partant ;
- nouvelle répartition des clés ;
- mise à jour des connexions.

Évidemment, si un nœud tombe en panne, l'opération de départ ne peut pas être effectuée. Le cas d'une panne est en fait très différent du cas du départ d'un nœud qui prévient ses voisins (nœud poli), et sera considéré plus loin (voir section 4.3.2). Nous allons donc considérer ici que le nœud d'identifiant $x_1 \dots x_k$ quitte poliment le système.

Remplacement d'un nœud partant

Lorsqu'un nœud u d'identifiant $x_1 \dots x_k$ quitte le réseau, plusieurs cas existent. Si un nœud v d'identifiant $x_1 \dots x_{k-1}\overline{x_k}$ est présent dans le système (comme jumeau inférieur ou supérieur de u), alors les tables de routage de u et de v sont fusionnées et gérées complètement par v, qui est alors renommé $x_1 \dots x_{k-1}$. Si aucun nœud d'identifiant

$x_1 \dots x_{k-1}\overline{x_k}$ n'est présent dans le système, alors la recherche d'un remplaçant est un peu plus complexe. Par exemple, le nœud $x_1 \dots x_{k-1}\overline{x_k}$ peut avoir étendu son identifiant en $x_1 \dots x_{k-1}\overline{x_k}0$ et $x_1 \dots x_{k-1}\overline{x_k}1$. Éventuellement, un de ces deux nœuds (voire même les deux) a lui-même étendu son identifiant, etc. De telles extensions d'identifiants créent un arbre binaire virtuel enraciné en $x_1 \dots x_{k-1}\overline{x_k}$ et dont les feuilles sont les nœuds alors présents dans le système (voir figure 4.3(b)). Dans cet arbre, les fils d'un nœud interne d'identifiant $x_1 \dots x_{k-1}\overline{x_k}y_1 \dots y_p$ sont les nœuds d'identifiant $x_1 \dots x_{k-1}\overline{x_k}y_1 \dots y_p0$ et $x_1 \dots x_{k-1}\overline{x_k}y_1 \dots y_p1$. Puisque la profondeur de cet arbre binaire virtuel est bornée (elle égale au plus $m-k$), il existe au moins une paire de feuilles dont les identifiants diffèrent seulement par leur bit le plus à droite. Nommons *paire critique* une telle paire de feuilles. Dans la figure 4.3(b), une paire critique est $\{x_2 \dots x_{k-1}\overline{x_k}1010, x_2 \dots x_{k-1}\overline{x_k}1011\}$. Les connexions jumelles permettent de trouver une paire critique pour tout nœud u nommé $x_1 \dots x_k$ quittant le système comme nous allons le montrer. Si $x_k = 0$, un message de recherche de paire critique est envoyé au jumeau supérieur u' de u. Ce message a pour format $\langle depart, @_u \rangle$. Si u' a pour identifiant $x_1 \dots x_{k-1}1$, $\{u, u'\}$ est une paire critique. Sinon, u' renvoie le message à son jumeau supérieur u''. Si les identifiants de u' et u'' ne diffèrent que par leur bit le plus à droite, $\{u', u''\}$ est une paire critique, etc. Puisque la suite de jumeaux est bornée, une paire critique sera toujours trouvée à la fin. Dans le cas $x_k = 1$, la méthode est la même en utilisant les jumeaux inférieurs à la place des jumeaux supérieurs.

Intuitivement, une fois trouvé un remplaçant, un des nœuds de la paire critique sera remplaçant pour u tandis que l'autre sera le remplaçant pour les deux nœuds de cette paire critique. Une explication détaillée suit dans la section suivante.

Nouvelle répartition des clés

Une fois un remplaçant $v = y_1 \dots y_{l-1}y_l$ trouvé pour le nœud $u = x_1 \dots x_k$, et la paire critique $\{v, w\}$ associée, ce remplaçant change son identifiant en $x_1 \dots x_k$ et récupère toutes les clés gérées jusqu'alors par le nœud partant u, c'est-à-dire les clés ayant pour préfixe $x_1 \dots x_k$. L'autre nœud $w = y_1 \dots y_{l-1}\overline{y_l}$ de la paire critique remplace v, récupère les clés dont s'occupait v avant et prend l'identifiant $y_1 \dots y_{l-1}$.

Mise à jour des connexions

Il existe deux cas pour cette opération, selon que le nœud partant u appartient ou non à la paire critique $\{v, v'\}$ trouvée plus haut.

Si $u \in \{v, v'\}$, le nœud u' d'identifiant $x_1 \dots x_{k-1}\overline{x_k}$ appartient à $\{v, v'\}$ aussi et devient donc le remplaçant pour u et u'. Ainsi, il reçoit de u toutes les clés qu'il gérait jusqu'alors. Il reçoit aussi de u toutes les informations concernant ses connexions avec ses jumeaux, ses pères et ses fils. Le nœud u' est renommé $x_1 \dots x_{k-1}$: cette opération est nommée *contraction d'identifiant*. u' informe alors ses pères de sa contraction d'identifiant et u informe ses pères qu'il quitte le réseau, u peut alors quitter le réseau. Le nœud u' met à jour sa table de routage avec les adresses physiques et les identifiants des anciens voisins de u, qui deviennent alors les fils de u'. Enfin, u' informe les anciens

pères de u qu'il est, à partir de cet instant, leur fils, d'identifiant $x_1 \dots x_{k-1}$.

Si $u \notin \{v, v'\}$, on peut noter sans perte de généralité que v a pour identifiant $x_1 \dots x_{k-1}\overline{x_k}y_1 \dots y_p 0$ et que v' a pour identifiant $x_1 \dots x_{k-1}\overline{x_k}y_1 \dots y_p 1$. Le nœud v' est le remplaçant de v et v', tandis que le nœud v est le remplaçant de u. v' effectue donc la même opération pour v et v' que u et u' dans le cas précédent. En particulier, l'identifiant de v' est contracté $x_1 \dots x_{k-1}\overline{x_k}y_1 \dots y_p$. Le nœud v prend l'identifiant de u, et récupères de u sa table de routage et sa table de clés. Dès que v aura récupéré toutes les informations de u, le nœud u quitte le système.

Remarque 4.3 En cas de panne, des nœuds peuvent perdre des fils sans être prévenus. Il est possible pour un nœud u de trouver des remplaçants à un fils défaillant $x_1 \dots x_k$ lorsqu'il se rend compte de leur défaillance. Si un nœud $x_1 \dots x_{k-1}$ existe parmi les fils, c'est qu'une contraction d'identifiant a eu lieu et que le remplacement est en cours. Sinon, u prévient le jumeau supérieur du nœud en panne c'est-à-dire le responsable de la clé $x_1 \dots \overline{x_k}0 \dots 0$. Ce nœud recherche alors une paire critique recherchée en suivant le chaîne des jumeaux de la même manière que vu en section 4.1.4. L'opération de contraction est alors lancée avec le remplaçant v de la paire critique $\{v, w\}$ pour remplacer le nœud défaillant.

Lorsqu'un nœud n voit un de ses pères $x_1 \dots x_k$ disparaître, il attend un message du père remplaçant. Toutefois, si l'attente dépasse un délai fixé par le système, il prévient le jumeau supérieur du père en panne, c'est-à-dire le nœud responsable de la clé $x_1 \dots \overline{x_k}0 \dots 0$.

Notons que dans le cas où la route utilisée normalement pour acheminer un message est coupée à cause d'une panne, il est possible de trouver d'autres routes en n'utilisant pas le routage optimisé. De manière plus générale, il est possible de faire passer ce message par un voisin quelconque qui l'enverra au réseau (*via* un autre chemin).

4.1.5 Exemple de comportement de D2B

La figure 4.5 montre un exemple d'évolution de D2B au cours du temps. Le premier nœud arrive dans le système (voir (a)) et prend pour identifiant vide $\varnothing$.

Quand un second nœud arrive dans le système (voir (b)), cet identifiant est étendu en 0 tandis que le nouveau nœud prend l'identifiant 1.

Un nœud arrive ensuite dans le réseau (voir (c)). Considérons qu'il choisit l'identifiant temporaire $1\dots$, le nœud 1 étend son identifiant en 10 alors que le nouveau nœud prend l'identifiant 11.

Un quatrième nœud arrive (voir (d)). Considérons qu'il choisit l'identifiant $0\dots$, le nœud 0 étend alors son identifiant en 00 tandis que ce nouveau nœud prend l'identifiant 01. Le réseau résultant est alors le graphe $B(2, 2)$.

En (e), un nouveau nœud arrive et choisit l'identifiant temporaire $01\dots$

En (f), un nouveau nœud arrive avec l'identifiant temporaire $011\dots$

En (g), un nouveau nœud arrive avec pour identifiant temporaire $11\dots$

En (h), un nouveau nœud arrive avec pour identifiant temporaire $00\dots$ Remarquons que le nœud 10 a un degré sortant de 5. en (h).

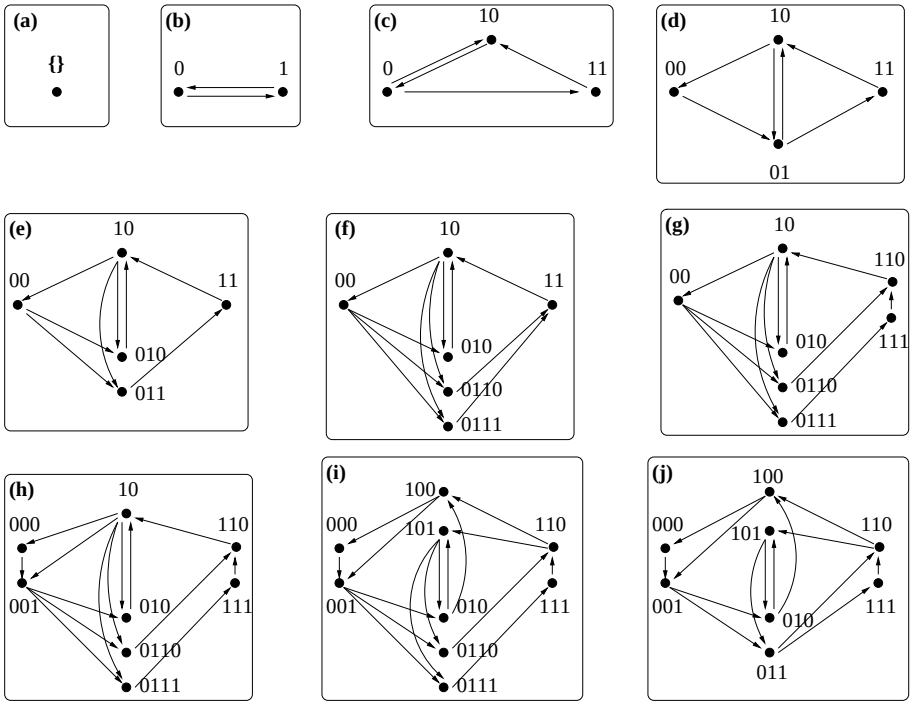

FIG. 4.5 – un exemple de comportement de D2B

En (i), un nouveau nœud arrive avec pour identifiant temporaire 10 . . ., Les nœuds 100 et 101 ont un degré approximativement égal à la moitié du degré de 10.

Enfin, en (j), le nœud d'identifiant 0110 quitte le système, 0111 contracte donc son identifiant en 011. Le réseau résultant est alors le graphe $B(2,3)$ décrit à la figure 4.2.

Le fait que les étapes (d) et (j) débouchent sur un graphe de de Bruijn est une coïncidence, et de manière générale, le graphe maintenu par D2B n'est pas isomorphe au graphe de de Bruijn. Toutefois, la «topologie moyenne» du réseau D2B est proche de celle maintenue par un graphe de de Bruijn pour des valeurs de n proches de puissances de 2.

4.2 Propriétés générales de D2B

Cette section est entièrement dédiée à la preuve du théorème 4.1 qui suit. La preuve est faite sous la contrainte que les arrivées et les départs ne se recouvrent pas. Toutefois, il est statistiquement possible de relâcher cette contrainte forte en utilisant les techniques présentées dans [66, 67].

Théorème 4.1 *Le réseau D2B d'ensemble de clés $\mathcal{K} = \{$chaîne de m bits$\}$ est correct et satisfait les propriétés suivantes :*

- *le nombre moyen de clés gérées par un nœud dans un réseau de n nœuds est $|\mathcal{K}|/n$ et a.f.p. au plus $O(|\mathcal{K}|\log n/n)$;*

- *une recherche/publication pour une clé κ lancée par un nœud d'identifiant $x_1 \dots x_k$ est routée correctement et, a.f.p., atteint le nœud responsable de la clé κ en au plus $O(\log n)$ sauts. La plus longue route suivie par un message de recherche est au plus $O(\log n)$ avec une probabilité $1 - o(1)$;*
- *sur tout nœud intermédiaire, la décision du routage nécessite $O(\log \log n)$ comparaisons de mots sur $O(\log n)$ bits. L'engorgement moyen d'un nœud quelconque est $O(\log n/n)$, ce qui est optimal pour tout réseau de degré constant. A.f.p., l'engorgement d'un nœud est au plus $O(\log^2 n/n)$;*
- *durant une insertion ou un départ, la répartition des clés implique seulement deux nœuds pour une insertion et au plus trois nœuds pour un départ. Le nombre moyen de changement de liens dûs à une insertion ou un départ est $O(1)$ et, a.f.p., ne dépasse pas $O(\log n)$.*

Ce sont les méthodes d'insertion et de départ qui ont pour conséquence que la répartition des clés n'implique que deux nœuds dans le cas d'une insertion et au plus trois nœuds dans le cas d'un départ. Les autres propriétés sont des conséquences des lemmes suivants.

Lemme 4.1 *À tout instant :*

1. *pour toute clé $\kappa \in \{0,1\}^\ell$, il existe un unique nœud dans le réseau D2B dont l'identifiant est préfixe de κ ;*
2. *soit u un nœud de D2B d'identifiant $x_1 \dots x_k$ qui a au moins deux fils. S'il existe i, j tels que $x_i \neq x_j$, les fils de u ont un identifiant de la forme $x_2 \dots x_k y_1 \dots y_p$, $p \geq 1$, et l'ensemble de suites $y_1 \dots y_p$ est un ensemble universel de préfixes. Si $x_1 = \dots = x_k = \alpha$, les fils de u ont des identifiants de la forme $x_2 \dots x_k y_1 \dots y_p$, $p \geq 2$, $y_1 = \overline{\alpha}$, et l'ensemble de suites $y_2 \dots y_p$ de tous les fils de u est un ensemble universel de préfixes.*

Preuve Initialement, il y a un unique nœud d'identifiant $\varnothing$ dans le réseau. Cette identifiant est préfixe de toute chaîne dans $\{0,1\}^*$. Ainsi, (1) est vérifiée initialement. Le nœud d'identifiant $\varnothing$ n'a ni père, ni fils. (2) est donc vérifié aussi. Nous allons montrer que ces deux propriétés sont préservées après une insertion ou un départ.

– **Le cas de l'insertion** Supposons que le réseau satisfasse (1) et (2), et qu'un nouveau nœud u arrive dans le réseau. Soit $s_1 \dots s_\ell$ l'identifiant temporaire de u, et soit $x_1 \dots x_k$ l'identifiant du nœud v responsable à cet instant de la clé $s_1 \dots s_\ell$. Le nouveau nœud u prend l'identifiant $x_1 \dots x_k 1$, alors que v étend son identifiant en $x_1 \dots x_k 0$. La méthode de répartition des clés décrite au chapitre 4.1.3 assure que la propriété 1 est satisfaite après une insertion puisque toutes les clés de préfixe $x_1 \dots x_k 1$ sont déplacées de v à u. Pour la propriété 2, nous allons examiner séparément le cas des connexions des pères et celui des fils.

Si le nœud v a au moins deux fils avant l'insertion, alors la propriété 2 assure que ces fils ont des identifiants de la forme $x_2 \dots x_k y_1 \dots y_p$. Par construction (voir la mise à jour des connexions dans le chapitre 4.1.4), s'il existe deux indices $i \neq j$ tels que

$x_i \neq x_j$, les fils d'identifiants $x_2 \dots x_k 1 y_2 \dots y_p$ deviennent les fils de u, tandis que les fils d'identifiants de la forme $x_2 \dots x_k 0 y_2 \dots y_p$ restent les fils de v. Puisque l'ensemble initial de suites $y_1 \dots y_p$ est un ensemble universel de préfixes, le même raisonnement s'applique aux deux ensembles de suites $y_2 \dots y_p$ correspondant à u et v. Remarquons que ces suites peuvent être vides, mais une chaîne vide est un ensemble universel de préfixes. Notons aussi que l'on ne considère ici que les nœuds avec au moins deux fils. La propriété 2 reste donc satisfaite pour u et v. Si $x_1 = \dots = x_k = \alpha$, le nœud $x_1 \dots x_k \alpha$ a un unique fils, et le fils $x_1 \dots x_k \overline{\alpha}$ a pour fils tous les nœuds initialement fils de v. Par la propriété 2, ces fils ont pour identifiant $\alpha \dots \alpha y_1 \dots y_p$ avec $y_1 = \overline{\alpha}$, $p \geq 2$, et l'ensemble des suites $y_2 \dots y_p$ est un ensemble universel de préfixes. La propriété 2 reste donc satisfaite pour u et v.

Si le nœud v a un père d'identifiant de la forme $\alpha x_1 \dots x_j$ avant l'insertion, alors après l'insertion ce père a remplacé son fils $x_1 \dots x_k$ par deux fils d'identifiants $x_1 \dots x_k 0$ et $x_1 \dots x_k 1$, et la propriété 2 est donc vérifiée. Si le nœud v a des pères d'identifiant de la forme $\beta x_1 \dots x_k y_1 \dots y_p$ avant l'insertion, alors après l'insertion, les pères de la forme $\beta x_1 \dots x_k 0 y_2 \dots y_p$ ont $x_1 \dots x_k 0$ comme unique fils, et les pères de la forme $\beta x_1 \dots x_k 1 y_2 \dots y_p$ ont $x_1 \dots x_k 1$ comme unique fils. La propriété 2 est donc maintenue après une insertion.

– **Le cas du départ :** supposons que le réseau satisfasse les propriétés 1 et 2, et que le nœud u d'identifiant $x_1 \dots x_k$ quitte le réseau. D'après la description de l'opération dans le chapitre 4.1.4, nous considérons d'abord par simplicité que u appartient à une paire critique, c'est-à-dire qu'il existe un nœud v d'identifiant $x_1 \dots x_{k-1}\overline{x_k}$ actuellement dans le réseau. Par construction, le nœud v se renomme en $x_1 \dots x_{k-1}$, et récupères toutes les clés auparavant gérées par u. La propriété 1 reste ainsi satisfaite après un départ.

Si $x_1 \dots x_k$ a un fils unique $x_2 \dots x_j$, $j < k$ avant le départ, alors $x_1 \dots x_{k-1}\overline{x_k}$ a aussi $x_2 \dots x_j$ comme fils unique. Après le départ, le nœud $x_2 \dots x_j$ devient le fils unique de $x_1 \dots x_{k-1}$. La propriété 2 reste donc satisfaite après un départ. Si $x_1 \dots x_k$ a un fils unique $x_2 \dots x_k$ avant le départ, et que $x_1 \dots x_{k-1}\overline{x_k}$ a des fils ayant des identifiant de la forme $x_2 \dots x_{k-1}\overline{x_k} y_1 \dots y_p$ avant le départ, où les suites $y_1 \dots y_p$ forment un ensemble universel de préfixes, alors après le départ, $x_1 \dots x_{k-1}$ a pour fils $x_2 \dots x_k$ et tous les nœud d'identifiant de la forme $x_2 \dots x_{k-1}\overline{x_k} y_1 \dots y_p$. La propriété 2 reste donc vérifiée après le départ puisque $\{x_k\} \cup \{\overline{x_k} y_1 \dots y_p\}$ est une ensemble universel de préfixes. Enfin, si le nœud $x_1 \dots x_k$ a des fils d'identifiants $x_2 \dots x_k y_1 \dots y_p$ avant le départ, tandis que $x_1 \dots x_{k-1}\overline{x_k}$ a des fils d'identifiants $x_2 \dots x_{k-1}\overline{x_k} z_1 \dots z_q$, alors après le départ, le nœud d'identifiant $x_1 \dots x_{k-1}$ a pour fils les nœuds d'identifiant $x_2 \dots x_k y_1 \dots y_p$ et $x_2 \dots x_{k-1}\overline{x_k} z_1 \dots z_q$. La propriété 2 reste donc satisfaite après le départ puisque les suites $y_1 \dots y_p$ et $z_1 \dots z_q$ sont des ensembles universels de préfixes.

Les pères de $x_1 \dots x_k$ et $x_1 \dots x_{k-1}\overline{x_k}$ respectivement de la forme $\alpha x_1 \dots x_k y_1 \dots y_p$ et $\beta x_1 \dots x_{k-1}\overline{x_k} z_1 \dots z_q$, ont $x_1 \dots x_{k-1}$ comme fils unique après le départ. Le propriété 2 est donc satisfaite. Un père de $x_1 \dots x_k$ et $x_1 \dots x_{k-1}\overline{x_k}$ d'identifiant de la forme $\alpha x_1 \dots x_j$, $j < k$, a pour fils $x_1 \dots x_{k-1}$ après le départ. Donc, si $\alpha x_1 \dots x_j$ satisfait la propriété 2 avant le départ, alors il la satisfait aussi après le départ. ■

Lemme 4.2 *Une recherche débutée par un nœud d'identifiant $x_1 \dots x_k$ atteint sa destination en au plus k sauts. Il en est de même pour une publication.*

Preuve Considérons un nœud u d'identifiant $x_1 \dots x_k$ recherchant une clé $\kappa = \kappa_1 \dots \kappa_\ell$. La recherche est effectuée en routant $\kappa_1 \dots \kappa_\ell$ vers le nœud v responsable de la clé.

Assertion : l'identifiant d'un nœud $w \neq v$ rencontré sur le chemin de u à v est de la forme $x_i \dots x_j S$ où $j \geq i$, S est une chaîne binaire qui peut être vide, et la longueur de la plus longue chaîne binaire qui soit suffixe de $x_i \dots x_j S$ et préfixe de κ est de longueur au moins $|S|$.

En u, $i = 1$, $j = k$ et $S = \varnothing$, donc cette assertion est vérifiée pour le premier nœud du chemin. Soit $x_i \dots x_j s_1 \dots s_\sigma$ l'identifiant du nœud courant $w \neq v$. Supposons que la longueur de la plus longue chaîne binaire T suffixe de $x_i \dots x_j s_1 \dots s_\sigma$ et préfixe de κ soit de longueur au moins σ. Supposons d'abord que w n'ait pas pour identifiant $\alpha\alpha \dots \alpha$. Si w a plus d'un fils, alors le lemme 4.1 implique qu'il existe un fils w' d'identifiant $L = x_{i+1} \dots x_j s_1 \dots s_\sigma y_1 \dots y_p$ telle que $T y_1 \dots y_p$ soit une chaîne binaire suffixe de L et préfixe de κ. D'après le choix de $y_1 \dots y_p$ dans la méthode de routage, le nœud suivant sur le chemin de u à v est le nœud w' d'identifiant L. Cette identifiant est de la forme $x_{i'} \dots x_j S'$ et vérifie l'assertion. Si w a un fils unique, alors il est de la forme $x_{i+1} \dots x_{j'}$ avec $i+1 \leq j' \leq j$, ou de la forme $x_i \dots x_j s_1 \dots s_{\sigma'}$ avec $1 \leq \sigma' \leq \sigma$. Dans ces deux cas, l'identifiant des fils vérifie l'assertion. Le cas où w est d'identifiant $\alpha\alpha \dots \alpha$ (ce qui ne peut arriver que pour $w = u$) est traité de la même manière, en appliquant à nouveau le lemme 4.1. L'assertion est donc toujours vérifiée.

D'après cette assertion, si $x_i \dots x_j S$ est l'identifiant du nœud courant sur le chemin de u à v, alors l'identifiant du nœud suivant est de la forme $x_{i+1} \dots x_{j'} S'$, où S et S' vérifient les hypothèses de l'assertion. Donc après $i-1$ sauts à partir d'un nœud d'identifiant $x_1 \dots x_k$: soit un nœud d'identifiant $x_i \dots x_j S$, est atteint, avec $x_i \dots x_j S$ est un préfixe de κ, atteint, soit un nœud d'identifiant $x_i S$ est atteint avec S un préfixe de κ. Dans le premier cas, le routage est terminé. Dans le second cas, le nœud suivant sur le chemin est la destination. Dans les deux cas, le nœud v responsable de la clé κ est atteint, et le nombre de sauts le long du chemin de u à v est au plus $(i-1)+1 \leq k$. ■

Lemme 4.3 *Supposons que les nœuds arrivent et repartent aléatoirement. Alors a.f.p. l'identifiant $x_1 \dots x_k$ d'un nœud dans un réseau de n nœuds vérifie $\log_2 n - \log_2 \log_2 n - O(1) \leq k \leq O(\log n)$. De plus, avec probabilité $1 - o(1)$, le plus long identifiant $x_1 \dots x_k$ vérifie $k = O(\log n)$.*

Preuve Considérons un nœud u d'identifiant $x_1 \dots x_k$ dans D2B. Puisque les nœuds arrivent et repartent aléatoirement, l'ensemble des identifiants dans un réseau D2B de n nœuds sont ceux qui seraient obtenus en choisissant n entiers indépendamment uniformément aléatoirement au hasard dans $[0, 2^\ell[$. Soit I un intervalle de $[0, 2^\ell[$ débutant en $val(x)$ et contenant $c 2^\ell \log_2 n / n$ entiers, pour toute constante $c > 3$. La probabilité qu'un entier soit choisi dans I est $c \log_2 n / n$. Soit X la variable aléatoire dénombrant les entiers choisis dans I. La borne de Chernoff [1], assure que $\text{Prob}(|X - c \log_2 n| >$

[1] Rappelons que la borne dite de Chernoff assure que pour N variables de Bernoulli indépendantes deux à deux $X_1, \dots, X_N$ de même paramètre $p > 0$, $\text{Prob}(|\sum_i X_i - Np| > k) < 2e^{-k^2/3Np}$, pour tout

$\sqrt{3c}\log_2 n) < 2/n$. Ainsi, a.f.p., au moins un entier est choisi dans I, et donc u est responsable de moins de $c2^\ell \log_2 n/n$ clés. De plus, puisqu'un nœud responsable d'au plus 2^q clés a un identifiant sur au moins $\ell - q$ bits, $k \geq \log_2 n - \log_2\log_2 n - \log_2 c$. D'autre part, a.f.p., moins de $O(\log n)$ entiers sont choisis dans I, et donc u est responsable d'au moins $\frac{|I|}{2^{O(\log n)}} = \frac{c2^\ell \log n}{n2^{O(\log n)}}$ clés. Il en résulte que $k \leq O(\log n)$.

Divisons $[0, 2^\ell[$ en $\Theta(n/\log n)$ intervalles de taille $2^\ell \log_2 n/n$, et considérons n entiers choisis uniformément indépendamment aléatoirement dans $[0, 2^\ell[$. On peut appliquer le résultat suivant de Raab et Steger [82], sur le jeu des balles dans les paniers. Supposons que nous lancions n balles indépendamment et uniformément aléatoirement dans b paniers, avec $n = c\, b\log_2 b$ pour une constante donnée c. Soit X la variable aléatoire dénombrant les balles dans les paniers. Alors $\text{Prob}(X > d\log_2 n) = o(1)$ avec d une constante dépendant de c. Appliquer directement ce résultat à notre situation assure que la probabilité que le nombre maximal d'entiers choisis dans n'importe quel intervalle dépasse $O(\log n)$ est $o(1)$. C'est pourquoi, avec probabilité $1 - o(1)$, le nombre minimal de clés gérées par un nœud d'un réseau D2B est au moins $2^\ell \log_2 n/n2^{O(\log n)}$, et donc que la longueur maximale de tous les identifiants est au plus $O(\log n)$. ■

Ce qui suit est une conséquence directe du lemme 4.3.

Corollaire 4.1 *Le nombre de clés gérées par un nœud d'un réseau D2B de n nœuds est, a.f.p. au plus $O(2^\ell \log n/n)$.*

Ce qui suit est une conséquence directe des lemmes 4.2 et 4.3.

Corollaire 4.2 *Le nombre de sauts effectués par une recherche afin d'atteindre sa destination dans un réseau D2B de n nœuds est a.f.p. au plus $O(\log n)$.*

Lemme 4.4 *Le nombre moyen de modification de liens dues à une insertion ou un départ est constant, et est au plus $O(\log n)$ a.f.p.*

Preuve Soit un unique fils, alors ses propres fils ont pour identifiants des chaînes de la forme $x_2 \ldots x_k y_1 \ldots y_p$ où $p \geq 1$. L'intervalle de clés couvert par les fils de u s'étend de $x_2 \ldots x_k 0 \ldots 0$ avec $\ell - k + 1$ zéros à $x_2 \ldots x_k 1 \ldots 1$ avec $\ell - k + 1$ un. Le lemme 4.3 implique que, a.f.p., $k \geq \log_2 n - \log_2\log_2 n - O(1)$. Le nombre de clés gérées par l'ensemble des fils de u est au plus $2^{\ell - \log_2 n + \log_2\log_2 n + O(1)} = O(2^\ell \log n/n)$. La borne de Chernoff permet d'affirmer que cette intervalle de clés est, a.f.p., couvert par au plus $O(\log n)$ nœuds. Le degré sortant de u est donc a.f.p., $O(\log n)$. Le même raisonnement s'applique pour le degré entrant du nœud u en considérant séparément des pères de la forme $0x_1 \ldots x_k y_1 \ldots y_p$, et ceux de la forme $1x_1 \ldots x_k y_1 \ldots y_p$. Le degré de u est donc a.f.p. $O(\log n)$. ■

Lemme 4.5 *L'engorgement moyen d'un nœud est $O(\log n/n)$, et a.f.p $O(\log^2 n/n)$.*

$k \leq Np$ positif.

Preuve Soit u un nœud quelconque dans D2B, et soit $x_1 \dots x_k$ son identifiant. Calculons une borne supérieure de la charge sur u, c'est-à-dire sur le nombre de recherche passant par u ou à destination de u. D'une part, la taille moyenne d'une table de clés gérée par u est $O(2^\ell/n)$. Puisqu'il existe $n-1$ sources possibles, la charge moyenne engendrée par la recherche de clés gérées par u est $O(2^\ell)$.

D'autre part, les messages de recherche traversant u ont tous la même propriété : pour une source d'identifiant $y_1 \dots y_p x_1 \dots x_i$, $i \geq 1$, les clés recherchées doivent être de la forme $x_j \dots x_k \kappa_1 \dots \kappa_{\ell-k+j-1}$ où $j \leq i+1$. Le nombre moyen de nœuds ayant un identifiant finissant par la suite $x_1 \dots x_i$ est $n/2^i$. Le nombre de clés de la forme $x_j \dots x_k \kappa_1 \dots \kappa_{\ell-k+j-1}$ avec $j \leq i+1$ est au plus $2^{\ell-k+i}$, et donc en moyenne au plus $2^{\ell+i}/n$. Donc, pour i fixé, l'augmentation de charge est au plus 2^ℓ Puisqu'il y a $O(\log n)$ valeurs possibles pour i, la charge moyenne totale est $O(2^\ell \log n)$, et donc l'engorgement est au plus $O(\log n/n)$.

Le lemme 4.3 assure a.f.p. que $k \geq \log_2 n - \log_2 \log_2 n - O(1)$ C'est pourquoi la taille de la table de clés gérées par u est a.f.p. au plus $O(2^\ell \log n/n)$. Soit $i_0 = \log_2 n - \log_2 \log_2 n$ et soit $i \leq i_0$. L'application de la borne de Chernoff permet d'affirmer que le nombre de nœuds dont l'identifiant finit par la suite $x_1 \dots x_i$ est au plus $O(n/2^i)$ avec une probabilité au moins $1 - O(\frac{1}{n \log n})$. Donc la contribution de ces nœuds à la charge de u est au plus $O(2^{\ell-k+i} n/2^i) \leq O(2^\ell \log n)$ avec une probabilité au moins $1 - O(\frac{1}{n \log n})$. Les nœuds dont l'identifiant contient la suite $x_1 \dots x_i$ comme suffixe pour un $i \leq i_0$ contribue donc pour $O(2^\ell \log^2 n)$ à la charge de u a.f.p.

Soit $i > i_0$ cette fois, la contribution à la charge de u par les nœuds dont l'identifiant contient la suite $x_1 \dots x_i$ comme suffixe. Par la borne de Chernoff, nous pouvons affirmer a.f.p. qu'au plus $O(\log n)$ nœuds d'identifiant se finissent par la suite $x_{i-i_0} \dots x_i$, et donc au plus $O(\log n)$ nœuds d'identifiant finissent par la suite $x_1 \dots x_i$. La contribution de ces nœuds à la charge de u est donc au plus $O((\log n) 2^{\ell-k+i})$. En additionnant les contributions de ces nœuds pour tout i, $i_0 < i \leq k$, la contribution totale à la charge est $O(2^\ell \log n)$.

La charge totale supportée par u est donc a.f.p $O(2^\ell \log^2 n)$. L'engorgement est a.f.p. $O(\log^2 n/n)$. ■

Remarque 4.4 L'engorgement moyen de $O(\log n/n)$ est optimal pour un réseau de n nœud de degré constant avec $|\mathcal{K}|/n$ clés par nœud. De plus, en considérant un graphe orienté de degré entrant et sortant maximal Δ. Le nombre de nœuds à distance $\leq d$ à partir d'un nœud u est au plus $\sum_{i=0}^{d} \Delta^i$. Il y a donc au plus $O(\sqrt{n}/\Delta)$ nœuds à distance $\leq \frac{1}{2} \log_\Delta n$, et donc $\Theta(n)$ nœuds à distance $\Omega(\log n)$. C'est pourquoi chaque nœud contribue pour $\Omega(|\mathcal{K}| \log n)$ à la charge, ce qui entraîne une charge globale de $\Omega(n|\mathcal{K}| \log n)$. Pour obtenir $n - o(n)$ nœuds ayant pour charge $O(|\mathcal{K}| \log n)$, la charge globale doit être équilibrée sur tous les nœuds. Ainsi, $n - o(n)$ nœuds ont une charge $\Omega(|\mathcal{K}| \log n)$, et donc un engorgement de $\Omega((\log n)/n)$.

4.3 Variantes et optimisations

Afin d'améliorer l'efficacité de D2B, plusieurs optimisations peuvent être utilisées, dont toutes les optimisations présentées au chapitre 3.4. Les variantes de D2B que nous allons détailler plus particulièrement ici sont la version à d dimensions du réseau (base d-aire du chapitre 3.4.1), une discussion sur la robustesse de D2B, une stratégie simple pour diminuer le degré des nœuds (voir chapitre 3.4.6), un moyen de rapprocher le réseau logique du réseau physique 3.4.7).

4.3.1 Le réseau D2B à d dimensions

D2B à d dimensions, $d \geq 2$, utilise un ensemble de clés $\mathcal{K} = \{0, \ldots, d^\ell\}$, c'est-à-dire l'ensemble de mots de longueur ℓ sur un alphabet de d lettres $0, 1, \ldots, d-1$. La topologie sous-jacente de D2B est $B(d, k)$. Plus précisément, un nœud de D2B à d dimensions est identifié par un couple $\langle x_1 \ldots x_k, [a, b]\rangle$ où $x_i \in \{0, \ldots, d-1\}$, et $0 \leq a \leq b \leq d-1$.

Un nœud $\langle x_1 \ldots x_k, [a, b]\rangle$ est responsable des clés $\kappa \in \{0, \ldots, d-1\}^\ell$ si et seulement si $x_1 \ldots x_k\alpha$ est un préfixe de κ pour tout $\alpha \in [a, b]$. Un ensemble universel de préfixes défini de manière similaire que pour $d = 2$ assure que toutes les clés sont gérées. Pendant une insertion, si l'identifiant temporaire du nœud u est géré par le nœud w d'identifiant $\langle x_1 \ldots x_k, [a, b]\rangle$, alors v étend son identifiant de la manière suivante. Si $a < b$, alors v change son identifiant en $\langle x_1 \ldots x_k, [a, a + \lfloor \frac{b-a}{2} \rfloor]\rangle$ alors que u prend l'identifiant $\langle x_1 \ldots x_k, [a + \lfloor \frac{b-a}{2} \rfloor + 1, b]\rangle$. Si $a = b$, alors v change son identifiant en $\langle x_1 \ldots x_k a, [0, \lfloor \frac{d-1}{2} \rfloor]\rangle$ tandis que u prend l'identifiant $\langle x_1 \ldots x_k a, [\lfloor \frac{d-1}{2} \rfloor + 1, d-1]\rangle$.

Les fils du nœud $\langle x_1 \ldots x_k, [a, b]\rangle$ sont soit de la forme $\langle x_2 \ldots x_j, [\alpha, \beta]\rangle$, $j \leq k$, soit de la forme $\langle x_2 \ldots x_k y_1 \ldots y_p, [\alpha, \beta]\rangle$, $p \geq 1$. Le routage s'effectue comme dans le cas à deux dimensions, en cherchant le fils d'identifiant ayant le plus long préfixe commun avec la clé recherchée. Les connexions avec les jumeaux sont définis comme pour le cas à deux dimensions, et l'opération de départ s'effectue de la même manière en cherchant une paire critique par le biais des jumeaux.

Il est facilement vérifiable que la longueur moyenne k d'un identifiant $\langle x_1 \ldots x_k, [a, b]\rangle$ est $O(\log N / \log d)$, et que le nœud a alors un degré de $O(d)$ et un diamètre $O(\log N / \log d)$. Ainsi, la version D2B à d dimensions permet un compromis entre le temps demandé par une recherche et le temps nécessaires à la mise à jour après une insertion ou un départ. Cela permet aussi d'accroître la résistance aux pannes, comme dit dans notre dernière section

4.3.2 Accroître les performances de D2B

Robustesse

Comme on l'a vu dans le chapitre 1.2.2, un système pair-à-pair doit pouvoir tolérer un certain nombre de pannes et de déconnexions soudaines qui empêchent les nœuds d'exécuter l'opération de départ. C'est pourquoi comme dans beaucoup de réseaux à contenu adressable, les nœuds de D2B doivent contrôler que chacun de leurs voisins est vivant par un échange périodique de messages de contrôle. Lorsqu'un voisin est reporté

défaillant, ses voisins le considèrent comme ayant quitté le réseau. Toutefois, cela fait perdre la table de clés de ce nœud. Les nœuds republient donc leur clés périodiquement afin de corriger et mettre à jour les tables de clés (voir [56] pour plus de détails). Si un nœud perd tous ses voisins, c'est-à-dire si tous les voisins partent soudainement du réseau (donc sans prévenir par l'opération de départ), alors ce nœud doit se réinsérer en contactant une passerelle (une des nœuds dont l'adresse est publique, ou un contact dont on connaît l'adresse physique), et simule le départ de ses voisins.

Comme on l'a vu au chapitre 1.2.2, il est préférable que le degré des nœuds soit suffisamment important pour que la probabilité de déconnexion totale d'un nœud soit négligeable. La version à d dimensions de D2B a un degré moyen de $\Theta(d)$. On peut ainsi prendre d aussi grand que nécessaire pour qu'un nœud ait une probabilité négligeable de perdre tous ses voisins simultanément. Si l'utilisation de D2B à deux dimensions est préférée (par exemple pour des raisons de simplicité), une autre solution consiste à connecter chaque nœud $x_1 \dots x_k$ à au moins $\log_2 n$ descendants de la forme $x_i \dots x_k y_1 \dots y_p$, pour $i \geq 1$. Cette solution présente l'avantage de diminuer le nombre de sauts nécessaires à une recherche de $O(\log n)$ à $O(\log n / \log\log n)$.

Choix optimisé de l'identifiant

Le degré maximum de D2B est donné par le jeu des «balles dans un panier». Si I est un intervalle de $[0, 2^\ell[$ de longueur $2^\ell \log_2 n/n$, nous avons vu que la borne de Chernoff assure qu'au plus $O(\log_2 n)$ nœuds ont des valeurs dans I a.f.p., et donc que le degré d'un nœud donné est $O(\log_2 n)$ a.f.p. Le résultat de [82] est que le maximum, sur tous les intervalles I, du nombre de nœuds qui ont des valeurs dans I est $O(\log_2 n)$, avec une probabilité $1 - o(1)$. Cela découle du fait que le lancer de n balles aléatoire dans b paniers avec $n \simeq b \log_2 b$ engendre un nombre maximum de balles dans un panier de $O(\log_2 n)$ avec probabilité $1 - o(1)$.

Azar *et autres* se sont attaqués dans une contribution prolifique [9] à un problème non sans rapport avec le problème de répartition des nœuds sur l'espace de clés : m balles sont lancées une à une dans n paniers et, à chaque lancer de balle, d paniers sont sélectionnées au hasard. Chaque balle choisit alors le panier contenant le moins de balles parmi les d paniers sélectionnés. [9] prouve que, lorsque n tend vers l'infini, le nombre de balles du panier le plus rempli est$(1 + o(1)) \ln\ln n / \ln d + \Theta(b/n)$ a.f.p. L'écart type est donc exponentiellement inférieur par rapport au cas où aucun choix n'est proposé aux balles, même pour $d = 2$. Cela suggère de donner le choix entre $d \geq 2$ différents identifiants à chaque nœud s'insérant dans le réseau. Pour chaque identifiant temporaire v, u calcule alors le nombre de clés qui lui seront assignées s'il choisit v comme identifiant. Le nœud u choisit ensuite l'identifiant qui maximise le nombre de clés qui seront sous sa responsabilité, et diminue donc d'autant le nombre de clés d'un nœud chargé du réseau. De cette manière, la responsabilité des clés et le trafic de contrôle devrait être plus équilibrée.

Dans des réseaux à contenu adressable où l'identifiant est choisi indépendamment de la passerelle, nous proposons une méthode laissant le choix parmi un nombre de nœuds qui dépend du nombre de sauts effectué par un message, et ce à faible coût puisqu'elle

ne nécessite qu'un seul message d'insertion. Pour cela, après avoir choisi un identifiant initial, un nouveau nœud choisit son identifiant parmi les d nœuds qui ont transmis son message d'insertion vers son identifiant initial. Dans les réseaux à contenu adressable que nous avons vu, qui envoient un message en $O(\log n)$ sauts, cette méthode laisse le choix parmi un nombre suffisant de nœuds. Dans les systèmes où les nœuds connaissent le nombre de clés gérés par leurs voisins, le nombre de nœuds parmi lesquels est choisi le nouvel identifiant peut être encore augmenté à faible coût. Il est en effet possible de choisir un identifiant parmi ceux gérés par les nœuds qui ont transmis le message d'insertion *et* leurs voisins.

4.3.3 Rapprocher la topologie logique de la topologie physique

Afin de trouver à faible coût un nombre important de nœuds pour permettre la diminution des délais des sauts dans le réseau logique, nous suggérons une adaptation des différentes méthodes de rééquilibrage de charge à l'arrivée d'un nœud dans le réseau (vues ci-dessus). Au lieu de choisir l'identifiant qui lui permet de récupérer un nombre maximal de clés, un nœud choisit alors un identifiant géré par le nœud le plus proche. Dans le cas de réseaux où l'inégalité triangulaire n'est pas prouvée, comme Internet, il est aussi nécessaire de vérifier que tous les futurs voisins sont eux aussi proches du nouveau nœud dans le réseau physique.

Rapprocher le réseau logique du réseau physique

Un réseau à contenu adressable est un réseau logique. Comme on l'a vu dans le chapitre 1.2.9, les connexions peuvent se faire entre deux nœuds très éloignés dans le réseau physique (en terme de coordonnées géographiques, de délais d'envoi de messages, etc.). Nous avons vu que Tapestry et d'autres réseau à contenu adressable permettent d'offrir des chemins alternatifs entre deux nœuds et tentent de choisir le meilleur (à travers les estimations que nous avons vu). Toutefois, le réseau lui-même n'est pas optimisé pour une meilleure utilisation du réseau physique et les chemins sont calculés *a posteriori*. Nous proposons la technique 3.4.7 présentée dans la section précédente.

4.4 Comparatif des performances des systèmes décentralisés structurés

Nous présentons ici un tableau récapitulatif des performances *moyennes* des systèmes vu dans ce chapitre. Dans ce tableau :

- n est le nombre courant de nœuds présents dans le réseau ;
- N est le nombre de nœuds maximal dans le réseau ;
- α le nombre de messages envoyés à chaque étape d'une recherche ;
- β est le nombre de chiffres de l'alphabet utilisé ;
- γ (pour Kademlia) le nombre de voisins dans chaque sous-arbre d'un nœud.
- λ (pour Broose) le nombre maximal de voisins des ensembles gauches et droits pour chaque nœud.

Rappelons que le diamètre de Kademlia et Broose n'est pas le même que le nombre de messages nécessaire à une recherche de clé puisque α messages sont envoyés à chaque saut lors d'une recherche (toutefois, un nœud déjà sollicité ne sera pas sollicité à nouveau).

	diamètre	degré	insertion [1]	engorgement
CAN [83]	$O(dn^{1/d})$	$O(d)$	$O(dn^{1/d})$	$O(dn^{1/d-1})$
Chord [96]	$O(\log n)$	$O(\log n)$	$O(\log^2 n)$	$O(\log n/n)$
Tapestry [105]	$O(\log n/\log \beta)$	$O(\beta \log N/\log \beta)$	$O(\log n/\log \beta)$	$O(\log n/(n \log \beta))$
Pastry [88]	$O(\log n/\log \beta)$	$O(\beta \log N/\log \beta)$	$O(\log n/\log \beta)$	$O(\log n/(n \log \beta))$
Viceroy [68]	$O(\log n)$	$O(1)$	$O(\log n)$	$O(\log n/n)$
Kademlia [70]	$O(\log n/\log \beta)$	$O(\gamma \log n/\log \beta)$	$O(\alpha \log n/\log \beta)$	
Broose [101]	$O(\log(n/\lambda)/\log \beta)$	$O(\lambda\beta)$	$O(\lambda + \alpha \log n/\log \beta)$	
D2B [33]	$O(\log n/\log \beta)$	$O(\beta)$	$O(\log n/\log \beta)$	$O(\log n/(n \log \beta))$

Pour les systèmes qui livrent une analyse *a.f.p.*, voici un tableau récapitulatif de leur caractéristiques :

	diamètre	degré	insertion [1]	engorgement
Chord [96]	$O(\log n)$	$O(\log N)$	$O(\log^2 n)$	
Viceroy [68]	$O(\log n)$	$O(\log n)$	$O(\log^2 n)$	$O(\log^2 n/n)$
Broose [101]	$O(\log(n/\gamma)/\log \beta)$	$O(\lambda\beta)$		
D2B [33]	$O(\log n/\log \beta)$	$O(\beta \log n/\log \beta)$	$O(\log n/\log \beta)$	$O(\log^2 n/(n \log \beta))$

4.5 Évaluation de D2B

À la suite de la publication de D2B, Elyès Ben Hamida [52] a étudié le comportement d'une version légèrement modifiée du protocole au moyen du logiciel FreePastry [35] (qui comporte déjà l'implantation de Pastry). L'étude du comportement de D2B a été effectué sur 50 machines d'une grappe du réseau Grid5000, au moyen d'une trentaine d'expériences afin d'obtenir une moyenne significative. Des expériences ont aussi été menée avec émulation de 400 nœuds virtuels sur 40 machines.

Les premiers résultats montrent un nombre de sauts augmentant de manière logarithmique, et un degré moyen entrant et sortant faibles $(2, 18)$ qui vont dans le sens des preuves que nous avons apportées dans cette thèse concernant le comportement de D2B. Le degré moyen sortant comporte toutefois une valeur médiane plus faible que celle à laquelle on pourrait s'attendre (1) qui peut être dûe aux différences entre le protocole et l'adaptation qui a servi à l'évaluation.

[1] nombre de messages nécessaires

4.6 Conclusion

Nous venons de présenter les systèmes décentralisés structurés, ces systèmes se révèlent intéressants en terme de performances lorsque l'on peut facilement associer une clé à un objet (DNS [19] par exemple), c'est-à-dire par exemple lorsque les objets partagés peuvent être nommées de manière unique, avec un format spécifique : nom de domaine, côte (ISSN d'un livre), etc. Les réseaux à contenu adressable permettent alors un routage efficace pour la recherche d'objets, au prix du maintien d'une structure d'interconnexion entre les nœuds.

Après avoir décrit les principaux systèmes existants, nous avons décrit plusieurs techniques permettant l'amélioration des performances des réseaux à contenu adressable. Ces techniques vont de l'accélération du routage à l'équilibrage de la charge en passant par l'augmentation de l'accessibilité des objets partagés.

Nous avons enfin vu le réseau à contenu adressable D2B, qui assure un routage en un nombre de sauts logarithmique et un degré constant en moyenne et logarithmique a.f.p. Cela permet de limiter le nombre de messages nécessaires lors de l'arrivée d'un nœud dans le système à $O(\log n)$ a.f.p. ce qui est une amélioration non négligeable par rapport aux réseaux à contenu adressable précédents comme Chord [96] et Viceroy [68] ($O(log^2 n)$ a.f.p. tous les deux). De plus, le degré de ce système peut être adapté selon les contraintes, par exemple selon la probabilité qu'un nœud soit déconnecté. La charge du routage est équilibrée parmi les nœuds, ce qui permet de limiter l'engorgement dû au routage et le nombre de clés gérées par chaque nœud. Un mécanisme général simple est proposé afin de rééquilibrer la charge des clés parmi les nœuds lors du départ d'un nœud du réseau. Enfin, ce système peut aussi bénéficier des diverses techniques d'améliorations que nous avons vu précédemment.

Troisième partie

Les systèmes décentralisés avec proximité d'intérêts

Nous avons vu qu'il est possible que les nœuds d'un système pair-à-pair s'organisent localement pour maintenir une structure routable, et permettre des recherches efficaces. Toutefois, dans le cas d'objets dont le nom n'a pas de format ou de nom universel, l'obtention d'une clé associée à cet objet n'est pas simple.

D'autres méthodes de connexion locales entre les nœuds peuvent toutefois exister sans pour autant nécessiter le maintien d'une structure. En particulier, nous avons vu au chapitre 1.2.9 que rapprocher dans le réseau logique les nœuds qui ont des intérêts proches pourrait accélérer les recherches dans les systèmes pair-à-pair. Cependant, la comparaison des intérêts des nœuds et l'interconnexion à maintenir entre eux afin de permettre une recherche efficace ne sont pas triviales. Nous allons voir dans cette partie comment les diverses propriétés statistiques qui ont été observées dans les échanges des systèmes pair-à-pair peuvent être exploitées pour connecter les nœuds dans un système pair-à-pair décentralisé, et quelles méthodes de recherche ont été proposées pour en tirer parti.

Chapitre 5

Utiliser les propriétés des graphes d'intérêts

Durant leur premières années, les systèmes pair-à-pair décentralisés avaient deux choix pour transmettre les messages de recherche : soit l'inondation à distance bornée (voir chapitre 2.4), soit des méthodes basées sur des tables de hachages réparties (voir les sections 2.5 et 3). Ces deux méthodes ont chacune leur inconvénient.

- L'inondation utilise un grand nombre de messages et consomme donc une grande quantité de bande passante, ressource rare et chère. L'inondation bornée interdit quant à elle la recherche exhaustive ;
- Les tables de hachages réparties imposent pour chaque nœud des voisins donnés ce qui rend plus difficile la gestion des connexions et déconnexions des utilisateurs. De plus, la recherche dans ces systèmes nécessite que chaque objet ait un nom unique ou en nombre limité, ce qui n'est pas toujours possible.

C'est pourquoi la création de systèmes décentralisés permettant des réponses rapides tout en demandant un faible trafic de contrôle est encore un problème ouvert. Actuellement, les systèmes laissent le choix entre connecter les nœuds sans maintenir de structure –ce qui est simple mais nécessite d'inonder– ou connecter les nœuds de manière structurée –ce qui permet de router les recherches mais est complexe et rend difficile la recherche de certains objets (ceux qui n'ont pas un nom unique).

Toutefois, des observations récentes [29, 50, 62] ont montré que les réseaux pair-à-pair présentent des propriétés d'*agrégation* (clustering). [65] a montré que cette agrégation peut être utilisée afin de diminuer le nombre de messages de recherche dans des systèmes comme Gnutella. Par ailleurs, ces systèmes pair-à-pair présentent aussi une structure en loi de puissance [29, 63] utilisable pour créer des méthodes de routage. L'avantage de telles méthodes serait de ne pas nécessiter le maintien d'une structure spécifique comme les réseaux à contenu adressable, et réglerait le problème du calcul de clé pour des objets n'ayant pas un nom unique. Nous allons présenter dans ce chapitre les observations effectuées sur les propriétés statistiques des réseaux pair-à-pair.

Afin de limiter le temps de recherche dans un système pair-à-pair, nous avons vu au chapitre 1.2.9 qu'il est possible de rapprocher le réseau logique du réseau physique,

en abordant les intérêts et les inconvénients de cette optimisation. En particulier, le système pair-à-pair n'a aucune influence sur le réseau physique et ne peut donc que s'inspirer de ce dernier. En revanche, le système contrôle totalement les connexions logiques entre les nœuds. En particulier, s'il ne peut diminuer les délais physiques entre deux nœuds dans le réseau logique, il peut connecter les nœuds afin que les recherches ne nécessitent qu'un faible nombre de sauts dans le réseau logique. En effet, si deux nœuds ont des intérêts communs, alors ils vont probablement échanger des objets. En pratique, lorsqu'ils échangent, des nœuds ne le font pas avec des nœuds au hasard, mais ils ont au contraire plutôt tendance à se regrouper en communautés. La grande majorité des échanges se font à l'intérieur d'une même communauté, et très peu ont lieu vers d'autres communautés. Remarquons que ces échanges ne sont pas forcément symétriques. Plusieurs travaux [29, 53, 62, 95, 102] ont proposé des améliorations pour les systèmes pair-à-pair basées sur cette propriété.

5.1 Représenter les intérêts des pairs comme un graphe

Nous allons voir que différentes observations ont permis de constater la présence de communautés dans les échanges des utilisateurs. On peut représenter les intérêts communs à des nœuds par des arcs dans un graphe orienté, où les nœuds sont les utilisateurs. On nommera cette représentation «graphe des intérêts» des utilisateurs, ou plus simplement «graphe des intérêts».

Toutefois, si un système pair-à-pair souhaite utiliser cette proximité d'intérêts pour son fonctionnement, il faut que chaque nœud puisse détecter les nœuds qui ont des intérêts communs aux siens *localement* et *à la volée*. C'est pourquoi un système pair-à-pair n'a accès qu'à un nombre limité d'informations sur les nœuds. Dans le cas de fichiers, un intérêt commun peut être déduit si des nœuds :

- proposent ou téléchargent des fichiers identiques ;
- proposent ou téléchargent des fichiers ayant le même contenu, mais dans un format potentiellement différent ;
- se déclarent intéressés par les mêmes mots clés ;
- etc.

Toutefois, dans un environnement pair-à-pair, des nœuds sont susceptibles d'être intéressés par des objets très populaires, qui sont moins représentatifs des intérêts des nœuds, ce type de mesure peut donc être raffiné en prenant en compte le nombre d'objet partagés, comme le fait [62] de manière globale. [14] propose une mesure de proximité entre deux nœuds u et v du point de vue de u (donc de manière non symétrique) qui tienne compte de la générosité des nœuds, car ils peuvent fournir potentiellement plus d'objets à un nœud, en plus de la popularité des fichiers partagés (ou de leur estimation). L'avantage de cette contribution est qu'elle permet de calculer localement une proximité des intérêts à la volée.

Par ailleurs, le dynamisme des systèmes pair-à-pair se ressent aussi dans les intérêts des utilisateurs, qui changent au cours du temps. C'est pourquoi permettre l'expiration d'un intérêt, au moyen d'une politique de cache ou en fixant cette durée pour chaque in-

térêt peut se révéler intéressant. Ces changements d'intérêts peuvent être plus brusques, lorsque la personne qui utilise un nœud pour se connecter au système change, ses intérêts étant *a priori* sans rapport. La gestion de ces événements peut alors se faire de manière simple en définissant des profils d'utilisateurs, maintenant une connexion au système tout en permettant le changement des intérêts du nœuds, des objets partagés et des voisins.

La plupart des travaux qui s'intéressent à l'exploitation des propriétés communes aux systèmes pair-à-pair(nous les détaillons plus bas) se basent sur des traces d'exécution. Celles-ci permettent de représenter le comportement des utilisateurs dans des systèmes pair-à-pair ou des systèmes où les utilisateurs sont supposés avoir les mêmes comportements : on y retrouve les lois de puissance et l'agrégation en communautés. Plusieurs travaux [29, 50, 62] ont présenté les propriétés statistiques des graphes d'intérêt trouvés dans eDonkey [26], un système pair-à-pair décentralisé non-structuré parmi les plus utilisés au monde. Ces propriétés sont communes aux réseaux sociaux et à beaucoup de réseaux d'interaction. Les graphes d'intérêts ont en particulier :

- une faible densité, c'est-à-dire que le degré moyen est très faible en comparaison du nombre de nœuds dans le réseau ;
- une faible distance moyenne entre deux nœuds, c'est-à-dire que le nombre moyen de sauts nécessaires pour relier deux nœuds du réseau croit logarithmiquement avec la taille du réseau ;
- une structure d'agrégats, c'est-à-dire que malgré une faible densité globale, la densité locale est importante. Cette propriété est intuitivement représentée par la phrase : «les amis de mes amis sont mes amis» ;
- une distribution des degrés en loi de puissance, où la proportion de nœuds ayant un degré d_i est de l'ordre de d_i^{α}. Les degrés des nœuds y sont très différents et le degré moyen n'est pas significatif car si beaucoup de nœuds ont un degré faible, des nœuds de grand degré existent et faussent la moyenne.

5.2 Utiliser la propriété de loi de puissance

L'utilisation de la distribution des degrés en loi de puissance des réseaux réels pour effectuer des recherches efficaces a été proposée dans [4, 60, 90] Dans ces articles, les auteurs approximent les distributions hétérogènes des degrés par des lois de puissance et étudient les propriétés de marche aléatoires ou déterministes dans des graphes aléatoires pour des distributions de degrés données (voir aussi [65]).

Dans [4] est proposée une étude analytique d'un algorithme basé sur une recherche en profondeur privilégiant le voisin de plus grand degré : à chaque saut de la recherche, le nœud courant vérifie ses voisins (voire les voisins de ses voisins) afin de savoir s'il s'agit du nœud recherché. Dans l'affirmative, la recherche s'arrête. Sinon, la requête est envoyée au voisin de plus grand degré. Une analyse en moyenne de cette méthode est confirmée par simulation. Elle montre que le nombre de sauts moyen nécessaire pour trouver un objet dans un réseau aléatoire en loi de puissance de n nœuds et de coefficient α croit sous-linéairement en $n^{3(1-2/\alpha)}$ pour $2 < \alpha < 3$. Quelques simulations

montrent une diminution du nombre de sauts nécessaire, bien qu'elles n'étudient pas l'évolution de ce nombre au cours du temps.

Dans [60], les auteurs effectuent des simulations basées sur un autre modèle de loi de puissance [10] et comparent la recherche d'un nœud par plus court chemin à des algorithmes locaux choisissant à chaque saut :

- un voisin aléatoire (marche aléatoire) ;
- un voisin avec une probabilité d'autant plus grande que son degré est grand (stratégie préférentielle) ;
- le voisin de plus grand degré (parcours en profondeur guidée par plus grand degré).

Comme dans [4], chacun de ces algorithmes s'arrête lorsqu'un nœud est voisin du nœud recherché. Ils observent alors que cette dernière stratégie est la plus efficace des trois stratégies locales au regard de la distance séparant deux nœuds via chacune des stratégies. Il faut toutefois noter que, dans leur calcul de distance en nombre de sauts nécessaire pour aller d'un nœud à l'autre, les sauts effectués dans une boucle (un ensemble de nœuds parcourus sans trouver de réponse) ne sont pas comptés. C'est pourquoi si la distance annoncée dans [60] pour le parcours en profondeur guidé par le plus grand degré est logarithmique, le nombre de sauts nécessaire pour trouver un nœud de manière décentralisée est en fait polynomial et non logarithmique. Afin de vérifier si cet algorithme est particulièrement adapté aux graphes en loi de puissance, les auteurs de [60] étudient aussi ces stratégies dans le contexte de graphes petit-mondes, en utilisant le modèle proposé par Watts et Strogatz [103]. La stratégie de recherche en profondeur en privilégiant le voisin de plus grand degré se révèle moins efficace dans ces réseaux. En effet, le choix du voisin qui aura la plus grande probabilité de trouver le nœud recherché n'est plus faisable car tous les nœuds ont un degré similaire dans ces graphes. L'efficacité de la stratégie de recherche en profondeur privilégiant le voisin de plus grand degré semble donc propre à certains graphes comme les graphes en loi de puissance. Les auteurs étudient aussi, pour chaque stratégie, la résistance aux pannes aléatoires de nœuds et aux attaques ciblées sur les nœuds de plus grand degré. Ils montrent alors que le nombre de communications entre les couples de nœuds qui sont rendues impossibles par des pannes aléatoires (en cas de déconnexion du réseau) ne varie pas selon la stratégie de recherche utilisée. Enfin, pour chaque stratégie est étudiée l'évolution du diamètre de réseau en fonction du nombre de nœuds supprimés, par panne et par attaque.

[65] compare une stratégie basée sur k marcheurs qui effectuent l pas à l'inondation bornée et l'inondation incrémentale dans différents réseaux : un réseau Gnutella de 4.736 nœuds, un réseau en loi de puissance de 9.230 nœuds, un réseau aléatoire de 9.836 nœuds, et une grille à deux dimensions de 10.000 nœuds. Une méthode est proposée pour arrêter la recherche par cette stratégie si un marcheur a déjà trouvé une réponse : chaque marcheur vérifie régulièrement si l'origine de la requête veut continuer la recherche. Pour cela, ils distinguent le cas où la répartition des requêtes vers les objets est uniforme et en loi de Zipf. De même, ils distinguent le cas des objets disponibles en nombre uniforme, proportionnel au nombre de requêtes effectuées dessus, et proportionnel à la racine du nombre de requêtes effectuées dessus. Dans le réseau Gnutella, le nombre moyen de messages reçus par nœud par rapport à l'inondation diminue d'un facteur 35 à 74 pour la stratégie d'inondation incrémentale, et d'un fac-

teur 84 à 104 pour la stratégie utilisant 32 marcheurs aléatoires. Ces amélioration se font au prix d'une augmentation du nombre de sauts nécessaires pour trouver l'objet cherché d'un facteur inférieur à 1.7 pour l'inondation bornée et d'un facteur inférieur à 2.6 pour la stratégie des 32 marcheurs. Par ailleurs, cet article étudie les stratégies de réplication listées plus haut pour déterminer les plus efficaces pour une stratégie de multiples marcheurs aléatoires. Il conclut que la meilleure technique est la réplication de chaque objet en $O(\sqrt{q_i})$ nœuds, où q_i est le nombre de requêtes effectuées sur cet objet, et où la politique de cache des nœuds efface les objets de manière indépendante de leur utilisation. Si la réplication sur des nœuds aléatoire se révèle plus efficace, une réplication sur le chemin d'une marche aléatoire nécessite presque autant de sauts en moyenne pour obtenir une réponse à une requête. Toutefois, [90] relève que la quantité de mémoire nécessaire par nœud peut être problématique et qu'un objet partagé par un nœud de faible degré nécessitera un grand nombre de sauts. De plus, basés sur un modèle probabiliste, ces résultats nécessiteraient d'être confirmés dans le cadre de requêtes réelles.

[90] propose une approche originale basée sur une publication par chaque nœud de la liste des objets qu'il partage, le long d'une marche aléatoire de longueur l. Une requête de recherche parcourt ensuite une marche aléatoire de même longueur l, et une inondation probabiliste est effectuée par chaque nœud rencontré sur la marche aléatoire. Cette inondation est effectuée vers chaque arête avec une probabilité supérieure au seuil de percolation du réseau. Les auteurs calculent la longueur de la marche aléatoire nécessaire pour trouver un objet dans un graphe en loi de puissance où les degrés des nœuds sont indépendants. Dans le cas de graphe de paramètre $2 < \alpha < 3$, une recherche peut trouver un objet en définissant $l \sim n^{1-2/\alpha}$. Le nombre de sauts effectués par la percolation étant logarithmique, $O(\log n)$ sauts sont nécessaires pour trouver un objet.

5.3 Utiliser les agrégats

Si la nature hétérogène des nœuds est montrée (entre autres) par la distribution des degrés, les facteurs sociaux et culturels donnent au graphe des intérêts une structure d'agrégats. Par exemple, si un nœud u_1 est intéressé par un objet $\mathcal{O}$ proposé par un autre nœud u_2, il sera probablement aussi intéressé par d'autres objets partagées par u_2. De plus, u_1 sera probablement aussi intéressé par les objets partagés par des nœuds intéressés par $\mathcal{O}$, voire par d'autres objets partagés par u_1. Nous pouvons résumer cela par deux assertions :

- les pairs s'organisent en communautés, qui sont des sous-graphes denses ;
- deux nœuds qui échangent un objet vont probablement échanger à nouveau des objets à l'avenir.

Pour ces nœuds dont les intérêts sont proches, on trouvera souvent le terme de voisinage sémantique, par analogie avec la similarité que l'on retrouve dans les mots-clés qui les intéresse. On parlera aussi de communautés pour des ensembles de nœuds ayant les mêmes intérêts. Notons que cet ensemble peut dépendre du point de vue de chaque nœud.

Dans [29] est effectuée une analyse débutée dans [53] sur la répartition des fichiers par grandes classes : audio et vidéo en particulier. Les auteurs utilisent la même trace que dans [53]. Après avoir noté un pourcentage de nœuds égoïstes (voir chapitre 1.2.13) élevé (68%), ce travail établit la répartition des fichiers dans les deux grandes classes audio et vidéo : 48% des fichiers sont audio et 16% sont des vidéos, tandis que l'espace pris sur le réseau change puisque 67% de l'espace est pris par des vidéos pour 16% de fichiers audio, bien plus petits. Ces types de fichiers représentent la majorité des fichiers du réseau, en nombre et en espace.

[29] montre ensuite que la proximité sémantique s'accompagne d'une proximité géographique pour les vidéos. En effet, pour 65% des fichiers, la moitié des sources au moins se trouvent dans le même pays. La récupération de vidéos dans un système pair-à-pair bénéficie donc d'une proximité géographique des sources, ce qui s'explique par le fait qu'un nœud tentera de récupérer une vidéo dans la langue de son pays. D'autre part, cet article avance que les recherches sur fichiers audio seraient les premières à gagner si les délais étaient diminués, puisque ces requêtes représentent 48% des requêtes effectuées dans leur trace d'exécution.

Cet article calcule pour tous les couples de nœuds partageant au moins un nombre donné de fichiers, la probabilité qu'ils partagent aussi au moins un fichier supplémentaire. Les expériences menées dans cette contribution montrent alors que dès un faible nombre de fichiers communs à deux nœuds, 10 par exemple, la probabilité qu'ils partagent un fichier de plus est élevée : la probabilité de partager 11 fichiers lorsque deux nœuds partagent 10 fichiers est de 80%. Cette probabilité augmente jusqu'à être très proche de 100% dès que 50 fichiers communs sont partagés par deux nœuds, elle chute brutalement autour de 325 fichiers partagés. De manière similaire, une étude de la corrélation entre les fichiers vient compléter cette corrélation entre les nœuds, montrant qu'elle est elle aussi élevée . Les auteurs expliquent cette corrélation par le fait que lorsque deux nœuds ont le même morceau d'un album, ou les mêmes parties d'un logiciel par exemple, il est probable qu'ils aient aussi les autres morceaux du même album ou les autres fichiers composant la logiciel.

D'autres contributions [62, 102] ont relevé que la structure d'agrégats du graphe des intérêts peut être utilisée pour créer des systèmes pair-à-pair efficaces, bien qu'aucune méthode n'ait alors été proposée pour concevoir un système pair-à-pair qui n'utilise *que* ces propriétés.

5.3.1 Choisir efficacement ses voisins

L'utilisation de la proximité d'intérêts des nœuds dans les systèmes pair-à-pair repose sur un choix essentiel : celui des voisins. Afin de ne pas nécessiter une quantité de mémoire trop importante, des politiques de remplacement des voisins sont souvent appliquées, comme nous allons le voir. Ces politiques sont assez naturellement inspirées de la gestion de cache.

Défavoriser les voisins les moins récemment utilisé (LRU)

La politique LRU (Least Recently Used) est l'une des politiques de cache les plus utilisées. Rappelons tout de même qu'elle consiste à garder une liste des derniers nœuds qui ont servi pour les k dernières requêtes, où k est la taille de la liste de voisins. C'est la politique qui a été le plus étudiée et qui sert de base de comparaison pour la plupart des systèmes. Elle a l'avantage de ne nécessiter qu'une quantité de mémoire constante égale au nombre de voisins k. Elle souffre du fait qu'une fois la période de lancement du réseau passée, la méthode change la liste des nœuds à chaque nouvelle requête, oubliant ainsi des voisins qui n'ont pas été utilisés depuis plusieurs requêtes. [102] donne l'exemple du remplacement par un nœud d'un voisin de sa communauté d'intérêt par un nœud qui répond à une requête concernant un objet populaire, tandis que [15] souligne que des nœuds ayant un grand nombre d'intérêts et obtenant des réponses de plusieurs nœuds à la fois remplaceront des voisins intéressants par des voisins potentiellement moins intéressants.

Maintenir un historique, ou avoir la mémoire longue

Cette politique de remplacement consiste à tenir à jour une liste des nœuds qui ont répondu à au moins une requête du nœud, et le nombre de fois qu'il a fourni une réponse à une requête depuis le lancement du réseau. Dans cette liste classée par ordre décroissant du nombre de réponses fournies, les voisins sont les x premiers nœuds. Elle ne souffre pas de l'inconvénient de la politique LRU, qui supprime des nœuds efficaces pour les remplacer par des nœuds plus récents mais pas forcément efficaces, car elle garde en mémoire toutes les requêtes effectuées depuis le début. Toutefois, son coût en mémoire augmente avec le nombre de nœuds qui ont fourni des réponses depuis le lancement du réseau. [102] relève aussi que l'adaptation des listes de voisins, dans le cas de changements d'intérêts, est plus longue lorsqu'un grand nombre de requêtes ont été effectuées. [102] utilise cette méthode comme base de comparaison avec la politique LRU.

Garder des voisins à fort taux de succès

La méthode de choix des voisins basée sur leur taux de succès est utilisée et évaluée par [95] pour une méthode de recherche où les nœuds transmettent une requête à leurs raccourcis dans leur ordre de classement tant qu'ils n'obtiennent pas de réponse. Elle consiste à classer les raccourcis selon leur taux de succès, c'est-à-dire le rapport entre le nombre de réponses qu'ils ont fourni et le nombre total de requêtes qui leur ont été transmises. Elle utilise une mémoire bornée et permet l'entrée rapide de nouveaux nœuds : l'efficacité moyenne initiale d'un raccourci est 100%, puis elle évolue jusqu'à se stabiliser (dans les expériences menée par [95]) à un taux d'efficacité moyen.

Défavoriser les voisins qui ont des objets populaires

La sélection des voisins selon leur popularité est proposée et évaluée par [102]. Elle a pour but de retenir les nœuds ayant des intérêts similaires tout en étant aussi simple et peu coûteuse que la politique LRU. Elle utilise pour cela une liste de nœuds, associée pour chacun à la date de la dernière réponse qu'il a fourni et au nombre de réponses obtenues du réseau lorsque ce nœud a fait son entrée dans la liste. Lorsqu'une requête est effectué par un nœud u et obtient k réponses, dont une fournie par un nouveau nœud v, v entre dans la liste de u dans l'un des cas suivants :

1. la liste de u n'est pas pleine ;
2. dans la liste de u, la différence entre la date de dernière participation d'un nœud w et la date courante est supérieure à une constante β fixée par le système. Le nœud w est alors supprimé et le nouveau nœud v est entré avec la date courante et k ;
3. dans le cas où aucune des deux premières conditions n'est satisfaite mais qu'au moins un nœud a fourni un nombre de réponses supérieur à k, le nœud qui parmi ceux-ci a la plus ancienne date de réponse est supprimé. v est alors entré dans la liste de u associé à la date courante et à k.

Remarquons que la date de dernière réponse permet alors de représenter l'activité du nœud tandis que le nombre de réponses obtenue est une mesure de la rareté des objets qu'il partage, inversement proportionnel à l'intérêt qu'il représente. Afin de ne pas juger l'intérêt des nœuds sur leur première réponse mais sur toutes leurs réponses qu'ils ont fourni, on pourrait modifier cette technique pour que le nombre de réponses associé à un nœud soit le nombre minimum de réponses obtenues parmi toutes les requêtes où il a été impliqué. Cela permettrait de mesurer le plus rare objet partagé par chaque nœud.

À la suite de [95], [102] effectue l'évaluation des politiques de choix basées sur un historique par rapport à la politique LRU, avec un modèle théorique de répartition de 1.000 objets parmi 2.000 nœuds. Pour cela, ils vérifient le taux de succès des raccourcis lorsqu'un nœud n'utilise que de tels voisins pour localiser un objet. Les auteurs montrent que la politique de popularité permet d'augmenter le taux de succès 10% au dessus de la politique LRU, et bien que 15% moins efficace que l'historique, elle semble un bon compromis entre le coût en mémoire de la première et l'efficacité de la seconde.

La même année, [53] vérifie cette évaluation à partir d'une trace d'exécution tirée du réseau eDonkey précisant quelles données sont partagées par quels nœuds, parmi 12.000 nœuds et 923.000 fichiers. La comparaison des deux méthodes est alors effectuée avec la politique LRU et une politique de choix de voisin aléatoire. C'est cette étude basée sur des données réelles que nous allons détailler ici.

Dans le but de vérifier l'impact de la disparition des nœuds de plus grand degré, cet article présente une étude décrivant l'effet de la suppression des 5%, 10% et 15% des nœuds fournissant le plus d'objets au réseau. Cette comparaison est effectuée sur le taux du succès en fonction de la taille de la liste de voisins d'intérêts proches. Les auteurs montrent alors que pour des listes de voisins de faible taille, la diminution du taux de succès est la même quelque soit le nombre de pairs supprimés, tandis que pour des listes

de voisins de taille importante, le taux de succès ne varie que de 6%. De plus, on relève que si l'impact de la suppression des 5% de nœuds les plus généreux baisse le taux de succès d'approximativement 10%, la diminution maximale du taux de succès pour 15% est de l'ordre de 15%. Cela pourrait signifier que des attaques sur plus de 15% des nœuds généreux diminueraient le taux de réponse d'au plus le pourcentage de nœuds généreux supprimés. Ces résultats sont donc *a priori* valable aussi pour des pannes de nœuds aléatoires. Des simulations montrent de manière intéressante que l'efficacité de la proximité d'intérêts n'est pas due à la générosité de quelques nœuds fournissant beaucoup d'objets au réseau. C'est donc bel et bien le regroupement des nœuds selon leurs intérêts qui permet l'efficacité des stratégies utilisant cette proximité sémantique.

Afin de vérifier si la proximité sémantique des nœuds est transitive, ils explorent les voisins sémantiques des voisins sémantiques lors de la recherche sémantique. Ils vérifient la façon dont croît le nombre de réponses obtenues à deux sauts par rapport au nombre de réponses obtenues en contactant le même nombre de voisins sémantiques à un saut. Ce nombre de réponse croissant de la même manière, cela montre que la proximité d'intérêt est transitive. Cela signifie en effet que pour un nœud, les voisins sémantiques de mes voisins sémantiques sont mes voisins sémantiques.

Enfin, [53] relève qu'il est possible d'annoter les voisins selon les catégories de fichiers pour lesquels ils sont intéressants (audio ou vidéo par exemple, la nature d'un fichier est identifiée par son extension). L'expérience présentée dans cet article montre qu'une recherche de fichiers audio permet d'améliorer le nombre de fichiers trouvés de 3% à 7% lorsque chaque nœud distingue parmi ses voisins ceux qui sont intéressants pour des fichiers audio en particulier. Une des difficulté est alors que la politique de cache ne fasse pas perdre de voisins d'une catégorie parce que les recherches courantes concernent une autre catégorie. [53] propose donc de séparer les listes de voisins selon les catégories de fichiers. Toutefois, la gestion dynamique de ces types de fichiers permettrait l'adaptation du réseau au cours du temps, par exemple en cas d'apparition d'un nouveau type fichier. Une solution simple serait de tenir compte des répertoires où sont rangés les fichiers partagés, les recherches en apprentissage permettront des méthodes plus évoluées si nécessaire.

Favoriser les voisins les plus souvent utilisé (MOU)

La politique MOU (Most Often Used) est proposée et comparée à la politique LRU dans [15]. Elle consiste pour chaque nœud à maintenir une liste des nœuds candidats pour devenir raccourcis, elle attribue pour cela une valeur à chaque candidat, qui sera d'autant plus grande que ce nœud est intéressant pour le nœud. À chaque requête :

1. l'intérêt de chaque nœud est multiplié par une constante α fixée par le système ($\alpha \in]0,1[$ si l'on veut diminuer l'intérêt des nœuds avec le temps) ;
2. pour toute réponse, tous les nœuds sur le chemin de la requête voient leur valeur augmentée d'une valeur qui décroît exponentiellement avec la distance qui les sépare de la source.

Cette méthode nécessite que chaque nœud retienne pour chaque requête une liste des nœuds qui l'ont transmise.

La politique MOU permet d 'atteindre un degré entrant moyen stable assez rapidement, ce qui signifie qu'en moyenne le nombre moyen de nœuds susceptibles d'envoyer un message à un nœud se stabilise. En comparaison, la politique LRU ne se stabilise à peu près que pour 1 raccourci. Le nombre de mises-à-jour moyenne nécessaires pour un raccourci MOU par nœud diminue très rapidement puisqu'il est inférieur à 50 après 2.500 requêtes, tandis qu'il est encore supérieur à 50 pour la politique LRU après 2.000.000 requêtes.

5.3.2 Diminuer le trafic

Le trafic chargeant les nœuds du réseau étant un des problèmes principaux des réseaux décentralisés non-structurés, plusieurs solutions ont été proposées afin d'utiliser les propriétés de loi de puissance et d'agrégats qu'on retrouve dans ces réseaux.

Utiliser en priorité des raccourcis

[95] propose d'améliorer Gnutella en y ajoutant pour chaque nœuds des «raccourcis» vers des voisins ayant les mêmes intérêts. Pour cela, lorsqu'un nœud reçoit une série de réponse pour une recherche qu'il a effectué, il ajoute aléatoirement un des nœuds qui ont répondu à ses raccourcis. La politique de classement de ces raccourcis utilisée par [95] est le *taux de succès* de ces raccourcis (décrite ci-dessus au chapitre 5.3.1).

Les auteurs évaluent l'efficacité de l'utilisation des raccourcis au moyen de simulations basées sur des traces d'exécutions, obtenues à partir de caches HTTP (comprenant de 868 à 32.361 nœuds) et de réseaux Gnutella (comprenant jusqu'à 542 nœuds) et KaZaa (comprenant jusqu'à 12.558 nœuds). Ils injectent ces traces dans des réseaux ramenés à une taille équivalente, et obtenus à partir de réseaux réels entre 8.000 et 40.000 nœuds. Les auteurs comparent le comportement de la méthode d'inondation (de Gnutella) à l'utilisation des raccourcis puis, dans le cas où aucune réponse n'est trouvée, en utilisant la seule méthode d'inondation.

Nombre de sauts : les raccourcis permettent dans le pire des cas une diminution du nombre moyen de sauts nécessaire pour trouver un objet de 4 à $1,5$. Les raccourcis permettent donc de tirer parti de la proximité d'intérêt d'autres nœuds dans le réseau pour réduire significativement le temps de recherche d'un objet. Si la méthode de recherche alternative est celle du système structuré Chord, le nombre de sauts nécessaire passe de 7 à $1,5$.

Charge des nœuds : les raccourcis réduisent la charge du nombre moyen de messages reçu par nœud d'un facteur 3 pour les systèmes pair-à-pair Gnutella et KaZaA, à 7 pour les requêtes vers des sites HTTP. En utilisant une méthode de recherche basée sur un système structuré comme Chord, cette charge diminue d'un facteur 2 à 4.

Taux de succès : les auteurs montrent que l'utilisation de raccourcis dans les réseaux pair-à-pair comme Gnutella ou KaZaA permet de trouver un objet dans 53% à 58% des

cas, et ce taux de réussite monte entre 82% et 90% pour les requêtes effectués pour des sites http.

Modification de l'interconnexion

[15] propose aussi l'utilisation de raccourcis, mais ceux-ci servent à influencer l'interconnexion dans le réseau, sans changer la méthode d'acheminement des messages, ni à y ajouter d'étape. Ces raccourcis sont donc utilisés comme les autres voisins, tirés aléatoirement, en utilisant une méthode de recherche comme l'inondation bornée de Gnutella. Pour le choix de ces raccourcis sont comparées deux politiques : LRU et MOU. L'utilisation de la politique MOU permet de constater l'émergence d'une arborescence connectant les raccourcis entre eux, en plus de l'interconnexion aléatoire du réseau. Ces deux stratégies sont évaluées par des simulations effectuées sur 6 voisins par nœud, dont 0 à 5 raccourcis, en créant un réseau de 20.000 nœuds, 200.000 objets partagés pour lesquels la répartition et le nombre de requêtes associés sont créées à partir des observations de réseaux réels.

Afin de diminuer le trafic de requêtes, ils proposent qu'un nœud divise le nombre de sauts restants d'une requête par deux lorsqu'elle concerne les intérêts du nœud. Cela permet de représenter l'augmentation de probabilité de trouver une réponse lorsque l'on envoie une requête à des nœuds qui sont intéressés par le sujet de la requête.

Nombre de sauts : la modification de l'interconnexion permet de diminuer le nombre de sauts nécessaire de 30% pour 1 raccourci jusqu'à 80% pour 5 raccourcis, les politiques LRU et MOU donnant des résultats très proches et légèrement à l'avantage de LRU ($0,1$ sauts de moins au pire).

Charge des nœuds : l'étude de la variation de la charge supportée par nœud, et de la charge maximale supportée par les nœuds montre que la politique MOU est au moins aussi efficace que la politique LRU. La charge maximale d'un nœud augmente d'un facteur légèrement inférieur à $2,51$ pour les deux politiques, avec un léger avantage pour le MOU. La répartition du trafic semble assez peu déséquilibré puisque la variation de cette charge passe de $2,3$, si aucun raccourci n'est utilisé, à $3,8$ lorsque 5 raccourcis MOU et $4,1$ lorsque 5 raccourcis LRU.

Puisque le nombre de sauts nécessaires pour trouver un objet semble diminuer, les auteurs vérifient l'effet sur la charge de la diminution du nombre de sauts lorsqu'une requête concerne les intérêts d'un nœud. Ils comparent alors un nombre constant de 6 sauts à leur méthode, diminuant de moitié le nombre de sauts en cas de requête sur les centres d'intérêt d'un nœud. Le trafic diminue d'un facteur $3,9$ avec 5 raccourcis jusqu'à $8,3$ en n'utilisant aucun raccourci, mais augmente le taux d'échec des requêtes. Toutefois, l'augmentation du nombre de sauts initial d'une requête à 7 permet de diminuer le taux d'échec par rapport à un nombre de sauts constant tout en divisant le trafic d'un facteur d'un facteur de $1,5$ (pour 5 raccourcis), à $1,8$ (sans utiliser de raccourci).

Taux de succès : le taux de succès de cette méthode pour un seul saut passe de moins de 10% si aucun raccourci n'est utilisé à plus de 30% pour l'utilisation d'un seul raccourci et plus de 60% pour 5 raccourcis. Pour 3 sauts, ce taux passe de 55% si aucun raccourci n'est utilisé à plus de 80% pour un seul raccourci, et plus de 90% pour 5 sauts. Toutefois, l'observation du nombre de réponses obtenues pour une requête en fonction du nombre de sauts révèle qu'en nombre important, les raccourcis diminuent le nombre de réponses obtenues par requêtes. Une raison avancée par les auteurs est qu'un nombre de voisins aléatoires trop faible empêche de trouver des réponses en dehors des nœuds ayant les mêmes intérêts (les raccourcis).

Chapitre 6

La méthode QRE

Nous allons décrire dans ce chapitre notre méthode *QRE* (prononcer à l'anglaise *query*), qui permet de rechercher des objets aussi efficacement que des réseaux à contenu adressable sans avoir à maintenir de structure. Afin de mettre en avant ses principales propriétés, cette méthode est délibérément maintenue aussi simple que possible. Pour toute implantation de *QRE*, il est donc évident qu'il serait nécessaire d'utiliser des heuristiques classiques utilisées habituellement pour en améliorer le comportement. De plus, la simplicité de *QRE* permet d'évaluer directement l'impact de cette contribution. En effet, le biais dans lesquels étaient tombés plusieurs évaluations de réseaux à contenu adressable vus au chapitre 3.4 était de proposer des optimisation qui étaient évaluées en même temps que le protocole lui-même. Cela rendait difficile l'évaluation des qualités intrinsèques du protocole par rapport à l'apport des optimisations utilisables par tous les systèmes structurés.

6.1 Proposition d'une méthode de recherche efficace

Afin de représenter le graphe des intérêts des utilisateurs, nous proposons de lier deux nœuds par un arc lorsqu'ils ont déjà échangé par le passé. Notons que cela ne signifie pas forcément que deux nœuds liés ont les mêmes intérêts à un instant donnée, mais il assure qu'ils ont été lié par un échange à un moment. Cette méthode simple présente l'avantage d'être utilisable localement et à la volée. Cette méthode est locale et fonctionne à la volée, ce qui permet une utilisation par chaque nœud du réseau. Ce modèle a pour but de valider le plus indépendamment notre méthode de routage, une utilisation de cette méthode dans un réseau réel devrait adapter les améliorations proposées par les travaux cités au chapitre précédent, entre autres :

- la gestion de plusieurs catégories de fichiers [53] ;
- l'utilisation de politiques de cache pour la gestion des voisins, afin de permettre le remplacement des voisins qui ne sont plus intéressants [53], et qui maintienne l'efficacité du routage vers le voisin de plus grand degré.

Comme nous venons de le voir, plusieurs travaux ont montré la nature en loi de puissance et la structure d'agrégats des graphes d'intérêts. La plupart d'entre eux ont

insisté sur les améliorations possibles des protocoles existants en utilisant *l'une* de ces propriétés. Nous allons montrer dans la suite que plusieurs de ces propriétés peuvent être utilisées simultanément pour la création de méthodes de recherche simples mais efficaces.

Pour cela, nous allons utiliser une méthode de recherche où *le réseau logique n'est autre que le graphe des intérêts*, c'est-à-dire le graphe défini par les nœuds du système reliés par des arêtes si un échange a déjà eu lieu entre ces nœuds.

Bien que le réseau logique ne soit pas une structure logique, nous présentons une stratégie de recherche simple non basée sur l'inondation et qui ne requiert aucune information sur la topologie globale du réseau logique. Cette stratégie est simple en ce sens qu'elle ne nécessite pas d'opération de publication complexe. De plus, la nature gloutonne de cette stratégie rend les opérations d'insertion et de départ du réseau très simples. Cela permet à cette méthode de supporter sans difficulté les pannes de nœuds, puisqu'aucune structure n'est à maintenir entre ceux-ci.

Afin d'évaluer les performances de notre méthode, nous avons effectué des simulations basées sur des *traces d'exécution réelles*. Ces simulations montrent que notre méthode permet une recherche en un nombre de sauts moyen *logarithmique*, ce qui est bien plus efficace que les systèmes proposant l'utilisation d'une des deux propriétés statistiques des graphes des intérêts (nature en loi de puissance ou structure d'agrégat.

6.2 Les principes utilisés dans QRE

Dans le réseau logique, les connexions entre les nœuds dépendent des requêtes effectuées dans le système : un nœud est connecté aux nœuds dont il a recherché un objet ou qui ont recherché un de ses objets. Les requêtes sont routées par une procédure de recherche décrite plus loin, et sont de la forme $\langle @, \mathcal{O}, k \rangle$ où $@$ est l'adresse du nœud à l'origine de la requête (son adresse IP par exemple), $\mathcal{O}$ est la description d'un objet, et $k \geq 1$ est le nombre de sources différentes recherchées pour $\mathcal{O}$ par le nœud à l'origine de la requête.

Nous considérons que tous les nœuds du système maintiennent, en plus de l'accès aux objets qu'ils partagent, une courte description de ces objets dans ce que nous nommerons un index local. Un nœud maintiendra aussi une copie de l'index local de chacun de ses voisins [1]. Enfin, on considère que chaque nœud connaît le degré de chacun de ses voisins. Des communications régulières (mais pas nécessairement fréquentes) permettront à chaque nœud de mettre à jour ses informations de voisinage.

6.2.1 La stratégie de recherche

Lors de la réception d'une requête $R = \langle @, \mathcal{O}, k \rangle$, le système effectue une recherche en profondeur d'abord, en envoyant d'abord la requête à son voisin de plus grand degré : pour chaque nœud u qui reçoit la requête R, si $\mathcal{O}$ ne peut être trouvée ni sur u ni sur

[1] Une méthode similaire a été utilisée dans les systèmes pair-à-pair hybrides comme Gnutella ou KaZaA

un de ses voisins, alors u renvoie R à son voisin de plus grand degré parmi ceux à qui il n'a pas encore renvoyé R. S'il n'en existe pas, u renvoie alors R au nœud qui lui a envoyée.

La figure 6.1 résume cette stratégie simple de recherche.

(1) **si** u partage $\mathcal{O}$, **alors** u envoie $\mathcal{O}$ à @ ;
(2) **sinon**
  (2.1) **si** u a un voisin u' qui partage $\mathcal{O}$
    alors u renvoie R à u' ;
  (2.2) **sinon si** u a déjà envoyé R à tous ses voisins
    alors u renvoie R à son voisin duquel il a reçu R ;
  (2.3) **sinon** u renvoie R au voisin de degré maximum à qui il n'a pas encore renvoyé R.

FIG. 6.1 – Stratégie de recherche dans *QRE* pour la requête $R = \langle @, \mathcal{O}, k = 1 \rangle$.

La stratégie ci-dessus décrit le cas $k = 1$: la recherche s'arrête dès la première copie de $\mathcal{O}$ trouvée. Si $k > 1$, alors u décrémente k du nombre de sources pour $\mathcal{O}$ qui ont été trouvés (parmi lui et ses voisins) et renvoie la requête comme précédemment, en ayant mis à jour le nombre de copie demandées.

De plus, afin d'éviter les boucles dans la recherche, les nœuds doivent retenir les requêtes R qu'ils ont traité jusqu'alors, ainsi que les identités des voisins auxquels ils ont déjà renvoyé R. Cette méthode est déjà utilisée par Gnutella, et les méthodes utilisées par ce système peuvent être réutilisées ici.

Enfin, notons que *QRE* n'utilise pas de hachage pour la recherche. Il permet ainsi d'effectuer des requêtes complexes (voir chapitre 1.2.10). comme des requêtes approximatives, par expressions rationnelles ou encore par intervalle. En effet, le format dans lequel sera exprimée la recherche par le nœud demandeur peut être utilisée tel quel dans la requête dès lors que le protocole permet ce format de recherche. Notons par ailleurs que dès lors que l'objet recherché est proposé sur un nœud du système, la stratégie de recherche le trouvera. Si l'objet cherché n'existe pas, la requête parcourra tout le graphe. Toutefois, elle ne sera pas dupliquée en plusieurs message, maintenant le coût de ce parcours minimal.

6.2.2 Dynamique du système

Dans *QRE*, une requête réussie, c'est-à-dire une requête qui a trouvé une réponse, engendre une ou plusieurs modifications dans les connexions entre les nœuds du réseau logique : si u_1 reçoit une réponse positive pour une requête R de la part d'un nœud u_2, alors une arête est créée entre u_1 et u_2. Cela signifie que u_1 et u_2 échangent leur adresse physique et leur index local. De plus, leurs voisins sont informés de leur augmentation de degré. De cette manière, le système maintient un réseau logique qui n'est autre que le graphe des intérêts des nœuds qui le composent.

Comme dans la plupart des systèmes pair-à-pair proposés auparavant, nous supposons que tout nœud tentant de se connecter connaît une passerelle (voir chapitre 1.2.5),

c'est-à-dire un nœud connecté au système et dont l'adresse physique lui est connu avant de s'insérer. Cette passerelle peut être par exemple la machine d'une connaissance qui lui a proposé d'utiliser *QRE*. Nous supposons qu'un nœud qui s'insère veut partager ou rechercher un objet (sinon, il n'a pas d'intérêt à s'insérer dans le système). Un nouveau nœud est donc toujours associé à un objet, qu'il fournit ou qu'il recherche. L'opération d'insertion se base sur cet objet, nommons-le $\mathcal{O}$. Le nouveau nœud envoie une requête de recherche pour $\mathcal{O}$ et se connecte, comme précisé sur la figure 6.1, aux nœuds qui répondent à la requête. Si aucun nœud ne répond, le nouveau nœud reste connecté à la passerelle. Ainsi, dans le meilleur des cas, le nouveau nœud est connecté à un nœud qui a un intérêt commun avec lui. Dans le cas où il ne trouve personne, il est connecté à la passerelle, qui est un de ses contacts extérieur au réseau. Il n'est donc pas déraisonnable d'imaginer que le propriétaire de la machine passerelle a des goûts en commun avec le nouveau nœud. Ce procédé peut être aisément modifié afin d'augmenter le nombre de connexions à l'insertion en recherchant plusieurs objets, par exemple tous les objets que le nœud veut proposer au réseau.

Lorsqu'un nœud souhaite quitter le système, il en informe ses voisins au moyen d'un message et se déconnecte simplement du système. Chaque nœud recevant ce message ôte le nœud parti de la liste de ses voisins ainsi que l'index local qui lui est associée. Il informe ensuite ses voisins de son nouveau degré, décrémenté. Remarquons qu'il suffit simplement à *QRE* de vérifier régulièrement la présence de ses voisins pour pouvoir gérer les pannes de nœuds. En effet, aucune structure n'étant maintenue, la perte d'un voisin n'a aucune incidence sur la possibilité de router un message.

Enfin, insistons sur le fait que *QRE* ne nécessite *aucun* système pair-à-pair sous-jacent pour fonctionner, contrairement aux méthodes proposées précédemment (décrites au chapitre 5). La procédure d'insertion se suffit à elle-même, et le réseau logique grandit grâce aux passerelles à partir des requêtes qui y sont effectuées et des réponses qui sont trouvées. Typiquement, les premiers nœuds se connectent directement à la passerelle, car la probabilité que les données qu'ils recherchent soient dans le réseau est faible : peu de nœuds sont connectés, donc peu de nœuds partagent des objets). Toutefois, après une période de montée en charge, les nœuds rechercheront des objets qui seront partagés sur le système, et le réseau logique grandira alors, de manière non triviale.

6.3 Performances de QRE

Il n'existe actuellement aucun modèle satisfaisant pour représenter le comportement précis des utilisateurs d'un système pair-à-pair. L'évaluation de notre méthode aurait en effet besoin d'utiliser une modélisation à la fois de la structure d'agrégats des graphes d'échanges, de la distribution des degrés en loi de puissance et du fait que deux nœuds qui ont échangé échangeront probablement à nouveau. C'est pourquoi nous avons évalué notre méthode par simulation. Nous avons utilisé pour cela des traces *réelles* obtenues d'un serveur `eDonkey` [25] (de type Lugdunum). Les observations effectuées sur ces traces sont détaillées dans [50]. La trace sur laquelle nous avons basé notre évaluation dure 2h 53min et concerne 46.202 nœuds. Elle est représentative de toutes les traces de ce

type concernant les propriétés que nous utilisons dans *QRE*.

6.3.1 Protocole de simulation

À partir de notre trace d'échange, nous avons extrait une liste chronologique de k-uplet $R^{(i)} = (u_0^{(i)}, u_1^{(i)}, u_2^{(i)}, \ldots, u_{k_i}^{(i)})$, chacun étant associé à une requête :

- $u_0^{(i)}$ est la source de la $i^{ème}$ requête ;
- k_i est le nombre de copies de l'objet demandé ;
- $u_j^{(i)}$, $j = 1, 2, \ldots, k_i$, sont les fournisseurs pour cet objet.

Nous avons utilisé 342.204 requêtes qui concernaient un total de 46.202 nœuds. Remarquons que cette simulation permet de gérer l'une des difficulté liée aux réseaux pair-à-pair : les nœuds égoïstes. Notre trace d'exécution tient en effet compte de leur présence et de leurs requêtes dans le réseau, puisque leurs requêtes sont enregistrées par le serveur qui nous fournit la trace, et que ces nœuds ne répondent jamais à des requêtes effectuées par les autres nœuds.

Notre simulateur traite chaque k-uplet de la manière suivante. Au pas i, la requête $R^{(i)}$ est traitée et simule le comportement de *QRE* face à une requête R pour laquelle $u_1^{(i)}, u_2^{(i)}, \ldots, u_{k_i}^{(i)}$ est la liste des fournisseurs pour l'objet demandé par $u_0^{(i)}$.

En d'autre termes, nous simulons le comportement de *QRE* tel qu'il est décrit dans la section 6, pour une requête $\langle u_0^{(i)}, \mathcal{O}_i, k_i \rangle$ où $u_1^{(i)}, u_2^{(i)}, \ldots, u_{k_i}^{(i)}$ sont les nœuds qui peuvent fournir l'objet $\mathcal{O}_i$ à cet instant.

Il se peut que pour certaines requêtes, $u_0^{(i)}$ ne soit pas encore inséré dans le réseau au pas i, du fait que nos traces ne contiennent que les requêtes de recherche dans le réseau eDonkey et pas les connexions de nœuds qui n'ont encore effectué aucune recherche. Dans un tel cas, le simulateur effectue l'opération d'insertion du nœud $u_0^{(i)}$ avec pour passerelle un nœud choisi uniformément aléatoirement parmi les nœuds du réseau. Ensuite, $u_0^{(i)}$ se connecte la passerelle et effectue sa requête. Le lien entre la passerelle et $u_0^{(i)}$ est ensuite retiré lorsque ce dernier reçoit une réponse d'un nœud pour l'objet $\mathcal{O}_i$, auquel il est connecté.

6.3.2 Résultats de la simulation

La figure 6.2 représente la distribution des degrés des nœuds, c'est-à-dire pour $k \geq 1$, le nombre $\delta(k)$ de nœuds qui ont un degré k. C'est une distribution à queue lourde (il existe des nœuds avec de grands degrés) proche d'une loi de puissance. Nous verrons plus loin que cette distribution en queue lourde apparaît suffisante pour que notre stratégie de recherche soit efficace. Dans nos simulations, il est important de noter que si le degré maximal est de 690, seuls $0,25\%$ des nœuds ont un degré supérieur à 300. À l'inverse, $2/3$ des nœuds ont un degré inférieur à 20. Le degré moyen, bien que peu significatif sur ce type de distribution, est de $47,9$. Ces caractéristiques prouvent que le degré des nœuds de *QRE* évolue de manière satisfaisante avec le nombre de nœuds, ce qui est résumé par :

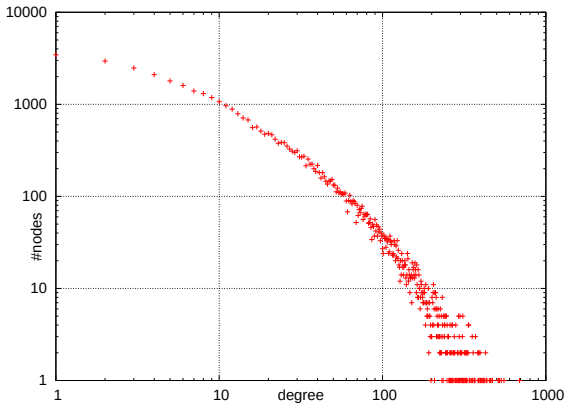

FIG. 6.2 – Distribution des degrés dans le réseau logique de *QRE*.

Assertion 1 : le graphe des intérêts, et donc le réseau logique créé par *QRE*, suit une distribution à queue lourde.

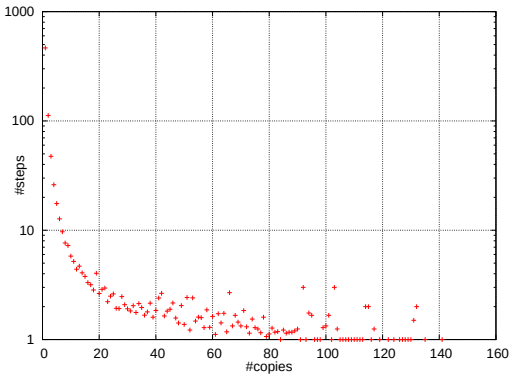

FIG. 6.3 – Impact du nombre de sources sur le temps de recherche.

La figure 6.3 montre le nombre de sauts moyen nécessaire pour trouver une copie d'un objet, en fonction du nombre de sources de cet objet. Ce nombre de sauts diminue rapidement avec le nombre de sources. En pratique, trouver un objet $\mathcal{O}$ dont au moins 7 copies se trouvent dans le réseau nécessite moins de 10 sauts en moyenne. De plus, un objet populaire $\mathcal{O}$, c'est-à-dire pour lequel au moins 25 copies sont présentes dans le réseau, a en moyenne une copie à au plus deux sauts du nœud qui le recherche. Soulignons que cela ne signifie pas qu'une copie de $\mathcal{O}$ est à distance au plus 2 de tout nœud, mais qu'une copie de $\mathcal{O}$ est à distance au plus 2 de tout nœud *intéressé* par $\mathcal{O}$.

Cela démontre l'existence de communautés dans le graphe des intérêts, représenté et utilisé par la méthode *QRE*, ce qui est résumé par :

Assertion 2 : le graphe des intérêts, et donc le réseau logique créé par *QRE*, a une structure d'agrégats.

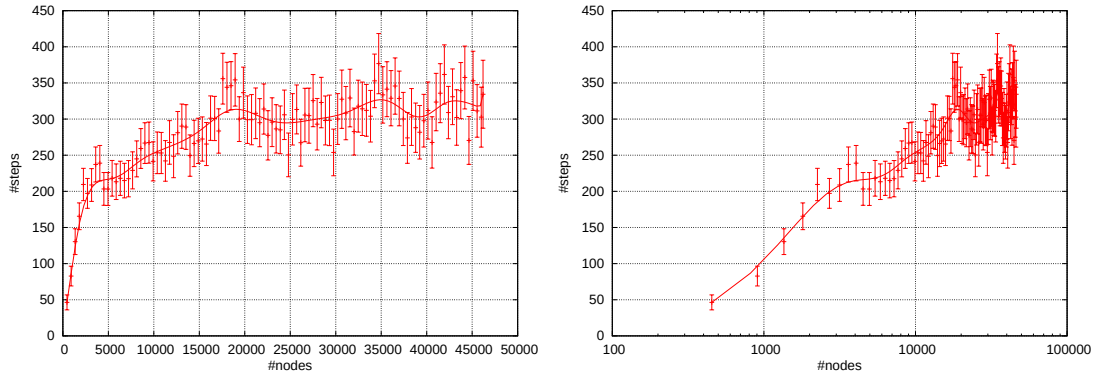

FIG. 6.4 – Nombre moyen de sauts nécessaire pour trouver un objet.

La figure 6.3 montre toutefois que les objets rares sont trouvés par la méthode *QRE* après un nombre de sauts importants. Toutefois, une fois que ce coût a été payé par un nœud, les recherches suivantes pour le même objet nécessiteront de moins en moins de sauts en même temps que le nombre de copies de cet objet dans le réseau sera de plus en plus important, et que les sources de ces copies seront de plus en plus connectées. La figure 6.4 (gauche) montre le nombre de sauts moyen $s(n)$ nécessaires pour trouver une copie d'un objet en fonction du nombre de nœuds n dans le système. Une régression linéaire indique que $s(n)$ évolue linéairement avec le logarithme du nombre de nœuds n du système (voir figure 6.4 (droite)).

C'est le résultat expérimental le plus important de notre contribution, qui est résumé par :

Résultat expérimental : la recherche dans *QRE* nécessite un nombre moyen de sauts logarithmique en fonction du nombre de nœuds présents dans le réseau.

Cela montre que la méthode de recherche de *QRE* est aussi efficace que les méthodes basées sur des tables de hachages réparties comme Chord [96], Viceroy [68], ou celle qui sont basées sur les graphe de *de Bruijn* binaire [1, 33, 57, 72].

De plus, *QRE* se révèle plus efficace que les résultats précédents décrits à la section 5. Cela montre que les performances de cette méthode ne sont pas seulement dus à la nature en loi de puissance ou à la structure d'agrégats du réseau logique, mais à des propriétés plus subtiles qui restent à découvrir et étudier. Ces propriétés ont trait entre autres autres :

- au regroupement des nœuds en communauté d'intérêts similaires ;
- à la façon dont les communautés se recoupent ;
- aux rôles et aux relations des nœuds de grand degré dans ces communautés.

La figure 6.5 montre le nombre moyen de sauts nécessaires pour trouver 20% des copies d'un objet présent dans le système. La localisation de copies d'un objet populaire demande au plus 10 sauts. *QRE* peut donc être utilisé efficacement en combinaison avec des protocoles permettant le téléchargement de gros fichiers en parallèle.

Pour finir, la figure 6.6 montre, pour $k \geq 1$, le nombre de requêtes qui nécessitent un nombre de sauts k. Cette distribution est à queue lourde : la plupart des requêtes

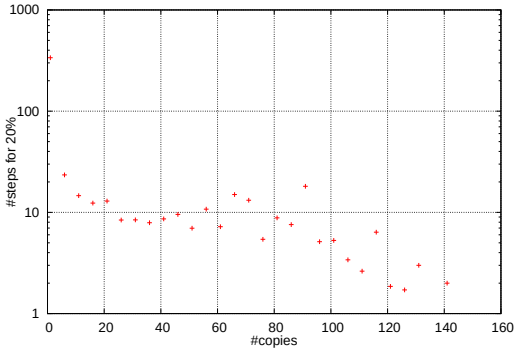

FIG. 6.5 – Temps de recherche pour trouver 20% des sources d'un objet.

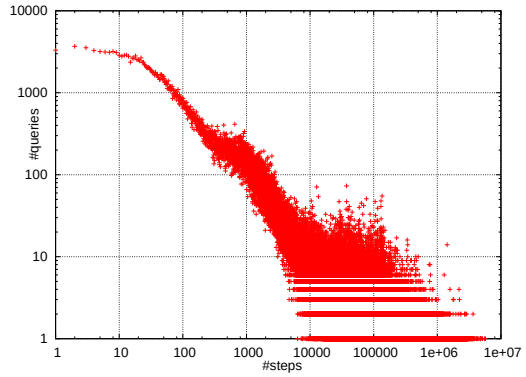

FIG. 6.6 – Distribution du nombre de sauts nécessaires à trouver un objet.

demandent peu de sauts, mais quelques requêtes demandent un grand nombre de sauts (ce qui correspond à des objets très rares pour lesquels il est nécessaire de parcourir tout le réseau ou presque). Typiquement, borner le nombre de sauts à 100 permettrait à la quasi-totalité des requêtes de trouver leurs réponses si elles existent.

6.4 Conclusion et perspectives

Dans ce chapitre, nous avons présenté une nouvelle méthode d'interconnexion et de routage, dont le principal objectif est de pousser plus avant l'utilisation des propriétés des intérêts des utilisateurs de systèmes pair-à-pair. Ces propriétés figurent en effet probablement parmi les facteurs clés qui permettront la création de systèmes pair-à-pair totalement décentralisés plus efficaces. Afin de confirmer cette hypothèse, nous avons proposé d'utiliser le graphe des intérêts comme réseau logique, et nous avons défini sur cette base des opérations simples de recherche, d'arrivée et de départ, créant ainsi un système auto-organisé. Ce sont les propriétés du graphe des intérêts, en particulier (1) sa structure d'agrégats et (2) l'hétérogénéité des degrés de ses nœuds, mais aussi des propriétés plus subtiles qui restent à découvrir, qui permettent à notre méthode de

localiser les objets en un nombre de sauts logarithmique, sans utiliser ni inondation, ni opérations complexes de publication ou de routage.

De manière générale, les difficultés qui pourraient être rencontrées lors de l'utilisation d'une telle méthode sont pour beaucoup liées au grand degré de certains nœuds.

Déconnexions : puisque les requêtes obtenues et utilisées dans les expériences menées sur *QRE* ne contiennent pas les déconnexions des nœuds, les simulations effectuées portent donc sur un réseau strictement croissant. L'étude de l'impact des déconnexions de nœuds sur des réseaux créés avec la méthode *QRE* permettrait donc d'évaluer l'effet de pannes et d'attaques sur les nœuds généreux.

Des travaux comme [53] indiquent que lors de l'utilisation en priorité de raccourcis qui ont des intérêts proches pour la recherche, le taux de succès ne souffre pas d'une suppression des nœuds généreux, donc *a fortiori* de déconnexions aléatoires. Toutefois, la méthode *QRE* connecte les nœuds selon leurs requêtes et utilise un routage et non une simple interrogation des voisins. L'efficacité de la recherche peut donc souffrir de la disparition de nœuds de grand degré, même si elle est peu probable (en cas de pannes aléatoires).

Dans le cas probable où ces déconnexions ont une influence, il est nécessaire de limiter le degré des nœuds, mais de manière à ne pas gêner le routage. La raison principale de cette limitation est que si la distribution des degrés suit une loi de puissance (et assure donc que peu de nœuds ont un grand degré), un routage vers les nœuds de degré élevé implique une charge plus importante sur les nœuds de grand degré en terme de nombre de requêtes à router.

Nœuds menteurs : la charge engendrée par un grand degré peut inciter les nœuds à mentir sur leur degré ou les fichiers qu'ils partagent (voir chapitre 1.2.2), faussant ainsi la recherche. L'évaluation de l'impact de tels comportements byzantins permettrait d'évaluer le risque qu'ils représentent pour l'efficacité des systèmes pair-à-pair. En particulier, les effet négatifs semblent moins importants lorsque les requêtes sont parallélisées, comme dans Gnutella. Une telle parallélisation des requêtes dans *QRE* pourrait alors représenter une solution pour limiter l'effet de nœuds menteurs (nous verrons plus bas des méthodes pour paralléliser efficacement les requêtes). L'étude du nombre de menteurs à partir duquel de tels mensonges ont un impact significatif sur l'efficacité de la recherche serait intéressant afin de déterminer la sensibilité de *QRE* à des comportements byzantins.

Limiter le degré des nœuds : la distinction du degré entrant et sortant des nœuds permettrait en plus d'inciter au partage (si un nœud favorise les recherches de ses voisins entrants qui sont aussi ses voisins sortants) de diminuer la quantité de mémoire nécessaire à chaque nœud (grâce à la diminution du degré sortant) et le nombre de requêtes reçues (grâce à la diminution du degré entrant). Toutefois, l'impact d'une telle distinction n'est pas évident, il faut en particulier s'assurer que cela ne déconnecte pas le réseau.

Afin de diminuer le degré de tous les nœuds du réseau, il est possible de diminuer artificiellement le degré maximal de la distribution à queue lourde, et avec le degré de tous les nœuds du réseau, tout en maintenant l'ordre de classement du degrés des nœuds afin de maintenir l'efficacité du routage.

L'utilisation du degré pour classer les voisins d'un nœud et ainsi choisir le destinataire d'une requête est similaire à l'attribution d'une valeur universelle pour chaque nœud, permettant un ordre total sur les nœuds. Le classement d'un nœud est donc indépendant des intérêts des nœuds, et est connu de ses voisins. Nous avons vu au chapitre 5.3.1 différentes politiques de classement des voisins qui permettraient à *QRE* d'adapter le voisinage aux intérêts des nœuds et donc de diminuer le degré sortant. En particulier, les méthodes [14, 15, 102] semblent efficaces pour choisir des voisins susceptibles d'augmenter la probabilité de trouver l'objet, donc de diminuer le nombre de sauts nécessaire et le nombre de messages nécessaires à la recherche dans *QRE*. Toutefois, comme nous l'avons expliqué plus haut, ces méthodes demandent à être adaptées à la méthode de routage, qui exploite la distribution du degrés des nœuds en queue lourde.

Par ailleurs, [15] relève que la présence de voisins aléatoires permet d'augmenter le nombre de réponses obtenues. L'introduction de voisins aléatoires pourrait donc permettre aux nœuds de connaître des voisins dont les intérêts sont différents mais qui pourraient avoir des objets intéressants, en cas de requêtes hors de ses intérêts habituels par exemple.

D'autre part, il serait possible de modifier la procédure de connexion entre les nœuds après une requête afin de ne connecter deux nœuds que si le nombre de sauts nécessaires pour aller du demandeur au destinataire est inférieur à une borne fixée par le système. Cela permet de diminuer le degré des nœuds au prix d'une augmentation du nombre de messages et du nombre moyen de sauts d'une requête. Ce procédé peut être développé en déportant des connexions d'un nœud vers un de ses voisins de degré plus faible, afin de ne pas trop augmenter son degré (le choix de quand déporter une connexion devrait alors se faire dynamiquement selon le degré du nœud qui refuse la connexion et le nœud qui l'accepte et non par rapport à une constante, toujours afin de permettre au routage de fonctionner).

Afin de mieux répartir de telles connexions, il est possible d'adopter un modèle de connexion inspiré de la méthode proposée par [15] : un nœud p_0 recevant une réponse pour une requête ayant parcouru le chemin $p_0, p_1, \ldots, p_{k-1}, p_k$ se connecte alors au nœud p_i, $0 < i < k$ avec une probabilité décroissant lorsque i se rapproche de 0. Une décroissance exponentielle semble indiquée dans ce cas.

Diminuer la charge : la charge supportée par les nœuds de grand degré est plus importante que pour les autres nœuds dans *QRE*. Une évaluation du nombre de messages reçus par ces nœuds en fonction du nombre de nœuds permettrait de connaître la répartition de cette charge et choisir le meilleur moyen de la répartir entre les nœuds.

Par ailleurs, l'évaluation du nombre de requêtes concernant des objets qui n'existent pas dans les réseaux réels est rarement effectuée, de même que le coût en nombre de message dans le réseau et par nœud. Elle permettrait pourtant d'estimer l'intérêt de

la limitation du nombre de sauts dans les différentes méthodes de recherche, y compris pour *QRE* (lorsque la probabilité de trouver un autre copie d'un objet est négligeable). À défaut d'assurer l'exhaustivité, la limitation du nombre de sauts permettrait de diminuer le nombre de messages circulant dans le réseau

La gestion dynamique du nombre de sauts effectués par une recherche est aussi possible, par exemple en fonction :

- du degré maximal rencontré et du nombre de sauts effectués ;
- du fait que la requête concerne des intérêts du nœud courant, comme proposé par [15] ;
- du nombre de sources trouvées pour l'objet recherché.

Ces trois méthodes permettent en fait l'estimation de la probabilité de trouver d'autres sources pour un objet recherché. Cela permet d'adapter la recherche au chemin parcouru par chaque requête.

Paralléliser la recherche : cela permet de toucher plus rapidement un même nombre de nœuds, et donc augmenter plus rapidement la probabilité de trouver des copies de l'objet recherché.

L'envoi parallèle d'une requête à plusieurs voisins est une solution permettant d'augmenter cette probabilité. Toutefois, elle nécessite le contrôle du nombre de sauts effectués afin de ne pas augmenter inutilement le nombre de messages. Ce contrôle peut se faire en fixant pour chaque requête un nombre de sauts incrémental, cette méthode a en effet permis à [65] d'observer pour l'inondation une diminution du trafic dans le réseau. Cet article suggère que chaque requête envoyée en parallèle vérifie régulièrement si le demandeur a trouvé le nombre de sources demandé, afin d'arrêter la recherche. La comparaison de ces méthodes avec des méthodes qui adaptant le nombre de sauts effectués à chaque requête, comme celles décrites ci-dessus, permettrait de vérifier les conditions dans lesquelles ces méthodes se révèlent efficaces.

Méthode de recherche : nous avons vu que plusieurs méthode de routage avaient été proposées pour les réseaux en loi de puissance, par exemple les stratégies évaluées dans [4, 60, 65, 90]. La méthode du parcours en profondeur guidée par le plus grand degré est la plus efficace des méthodes connues pour des réseaux dont les degrés des nœuds sont répartis en loi de puissance. Toutefois, la propriété de communauté peut rendre une autre méthode plus appropriée. Des améliorations simples de la recherche comme router une requête vers le nœud ayant le plus grand nombre de voisins qui n'ont pas encore reçu cette requête aurait par exemple un coût en mémoire trop important (pour les nœuds de fort degré et leurs voisins). L'étude d'autres méthodes de routage qui tirent parti des propriétés d'agrégats en plus de la loi de puissance semble donc une direction intéressante pour améliorer les performances de ces réseaux. Elles peuvent permettre d'améliorer le coefficient logarithmique du nombre de sauts moyen d'une requête, mais elles peuvent aussi avoir pour but une meilleure tolérance aux pannes par exemple.

Évaluation : une phase d'émulation à grande échelle, par exemple au moyen de grilles de calcul et de traces réelles, permettrait enfin de vérifier le comportement général de *QRE* en milieu réel, en particulier face aux requêtes parcourant le réseau au même instant. Une comparaison par rapport aux autres systèmes pair-à-pair décentralisés permettrait alors éventuellement d'observer des différences encore non observées entre les méthodes d'interconnexion et de recherche, et leurs implications sur la robustesse et la sécurité du réseau.

Conclusion générale et perspectives

Conclusion générale

Nous avons présenté dans ce manuscrit une étude des systèmes pair-à-pair décentralisés en nous intéressant particulièrement à la problématique de la recherche et plus généralement au routage dans les systèmes pair-à-pair décentralisés, et à l'interconnexion nécessaire pour assurer ce routage. Nous avons utilisé pour cela deux approches différentes : d'une part celles des réseaux à contenu adressable et d'autre part l'exploitation des propriétés des échanges dans les systèmes pair-à-pair. Les systèmes pair-à-pair sont apparus il y a une dizaine d'années, et ont fait apparaître une problématique qui leur est propre, comprenant entre autres les exigences suivantes :

- une décentralisation totale, en particulier la charge (trafic, espace de clés, etc.) sur les nœuds doit être équitablement répartie ;
- un faible temps de réponse ;
- supporter le dynamisme dans l'arrivée et le départ des nœuds ;
- permettre une recherche exhaustive.

Basés sur l'expérience des architectures parallèles, [83, 96] ont proposé d'interconnecter les nœuds en une structure logique afin d'assurer de bonnes performances à ces systèmes, en particulier concernant le degré des nœuds et le diamètre du réseau. [83] baptisa les systèmes basés sur une telle structure les «réseaux à contenu adressable». Cette contribution était accompagnée d'un exemple basé sur un tore à d dimensions. De nombreuses propositions de structures suivirent ou apparurent en parallèle, allant de l'hypercube au papillon. Chord [96] ou Viceroy [68] ont tenté de donner des bornes efficaces pour les différentes contraintes imposées par les systèmes pair-à-pair au moyen de résultats avec forte probabilité.

Dans cette thèse, nous avons présenté *D2B* [33], un réseau à contenu adressable basé sur le graphe de de Bruijn, connu pour assurer un degré constant et un diamètre $\log_2 n$, où n est le nombre de nœuds du graphe. *D2B* assure un diamètre d'au plus $O(\log n)$ avec forte probabilité, où n est le nombre courant de nœuds du réseau. Le degré de *D2B* est constant en moyenne, et au plus $O(\log n)$ avec forte probabilité. Cela permet de limiter le nombre de messages envoyés à l'arrivée ou au départ d'un nœud dans le système à un nombre constant de mises à jour en moyenne. L'insertion est donc en particulier plus économe en messages qu'une grande partie des réseaux à contenu adressable, dont Chord [96], et au pire aussi économe. Le coût de l'insertion en messages est avec forte probabilité inférieur à tous les réseaux à contenu adressable, dont Viceroy [68]. Nous

avons limité dans *D2B* les différents facteurs qui peuvent charger les nœuds :

- l'engorgement de *D2B*, c'est-à-dire le nombre de requêtes que peut avoir à router chaque nœud par rapport au nombre total de requêtes possibles, est au plus $O(\log n/n)$ en moyenne, et $O(\log^2 n/n)$ avec forte probabilité ;
- le nombre de requêtes à destination d'un nœud, c'est-à-dire le nombre de clés gérées par un nœud, est borné par $O(|K| \log n/n)$ avec forte probabilité, où $|K|$ est le nombre de clés de l'espace de clés K et n est le nombre courant de nœuds dans le réseau.

Les premières simulations et émulations de *D2B* sont encourageantes et confirment l'efficacité de *D2B* concernant son degré et le diamètre de son réseau [52].

Un apport majeur de notre contribution par rapport aux systèmes déjà existants est donc une *charge moins importante* sur les nœuds en terme de nombre de messages tout en assurant un diamètre et un degré faibles. C'est là une des contraintes importante pour les systèmes pair-à-pair, dont le but est la décentralisation. Notons que tout comme Chord [96] ou Viceroy [68], mais contrairement à Tapestry [105] ou Pastry [88], les bornes sont exprimées en fonction du nombre courant de nœuds et non du nombre maximal de nœuds dans le réseau. Cela permet d'assurer de meilleurs performances durant la montée en charge du système. Nous avons enfin suggéré plusieurs optimisations pour améliorer l'efficacité de ces réseaux, entre autre pour la répartition de la charge.

Les réseaux à contenu adressable ont toutefois l'inconvénient de ne permettre que les recherches exactes, car la recherche est basée sur le hachage de nom (d'objet). Cette limite est contraignante pour des applications où un même objet est susceptible d'être nommé différemment (comme les fichiers) et où plusieurs hachages peuvent donc correspondre à un même objet. En effet, un nœud recherchant un objet n'a alors plus la garantie d'obtenir toutes les réponses disponibles dans le réseau. Dans de telles situations, une recherche par expression rationnelle ou une recherche approximative (aussi appelée *wildcard*) serait alors plus adaptée, mais les réseaux tels que CAN, Chord, ou *D2B* ne permettent pas ce type de recherche.

C'est pourquoi nos recherches nous ont mené aux graphes modélisant les échanges entre les nœuds. En effet, plusieurs contributions [29, 53, 62, 102] ont signalé la présence de propriétés statistiques de loi de puissance et petit-monde dans les échanges des systèmes pair-à-pair. Ces différentes études étaient accompagnées de propositions d'améliorations pour des systèmes pair-à-pair existants, principalement l'utilisation de raccourcis et la modification de l'interconnexion. Poussant plus loin l'utilisation des propriétés des échanges dans les systèmes pair-à-pair, nous avons proposé une nouvelle méthode pour la conception de systèmes pair-à-pair non structurés.

Notre méthode de connexion et de routage, *QRE*, tire parti des propriétés de loi de puissance et de regroupement des nœuds en communauté. Ces deux propriété avaient déjà été utilisées, mais séparément. La méthode *QRE* se base sur un parcours en profondeur d'abord privilégiant le voisin de plus grand degré, qui avait été proposée et évaluée dans [4, 60] pour les graphes à distribution des degrés en loi de puissance. Afin d'assurer l'interconnexion des nœuds, chaque nœud est connecté à tous les nœuds avec lesquels il a échangé dans le passé. Le résultat principal de cette contribution concerne l'efficacité de la recherche. Dans *QRE*, les requêtes trouvent une réponse en un nombre

de sauts moyen logarithmique en fonction du nombre de nœuds dans le réseau. Notre routage simple ne demande pas de structure spécifique, ni de publication complexe tout en se montrant donc aussi efficace que les réseaux à contenu adressable comme *D2B*. Dans *QRE*, la distribution des degrés des nœuds ainsi que la distribution du nombre de requêtes selon le nombre de sauts effectués suivent une loi de puissance. Cela signifie qu'une grande partie des requêtes demande un faible nombre de sauts. Il existe toutefois toujours des requêtes demandant un nombre de sauts importants. *QRE* regroupe les nœuds en communauté au sein desquelles sont présentes la plupart des réponses aux requêtes qui y sont initiées. Toutefois, il existe des requêtes pour des objets qui ne sont pas encore présents dans la communauté, et qui sont donc plus coûteuses. La méthode *QRE* a aussi pour caractéristique originale de tirer parti de la réplication des objets dans les systèmes pair-à-pair puisque les simulations montrent par exemple que, dès que sept sources d'un objet existent dans un réseau de 45.000 nœuds, moins de dix sauts suffisent en moyenne pour trouver une des sources de l'objet. L'originalité de notre contribution réside dans son efficacité malgré sa simplicité : c'est la première méthode à utiliser avec succès conjointement les propriétés de loi de puissance et de regroupement des nœuds en communauté dans les systèmes pair-à-pair.

Perspectives

Plusieurs types de systèmes pair-à-pair ont émergé, selon le type de contraintes qui étaient imposées, obligeant *grosso modo* à choisir entre garantie d'efficacité (réseaux à contenu adressable), adaptation à la capacité des nœuds (systèmes hybrides), et requêtes évoluées (systèmes décentralisés non structurés).

Nous avons décrit dans le chapitre 6 plusieurs directions pour l'amélioration de méthodes utilisant les propriété de lois de puissance et d'agrégation des nœuds en communautés d'intérêts. Malgré le nombre de résultats parus sur ce sujet, nous avons listé le long de ce manuscrit plusieurs problèmes ouverts dont la résolution permettrait la création de systèmes pair-à-pair plus efficaces. En particulier, si nous avons contribué à montrer que l'utilisation des propriétés des échanges dans les systèmes pair-à-pair se révèlent prometteuses pour la conception de systèmes pair-à-pair efficaces tout en autorisant des requêtes évoluées. Ces propriétés restent toutefois encore trop mal connues pour pouvoir les exploiter pleinement.

À moyen terme, l'étude des propriétés des graphes d'échanges permettra une meilleure compréhension des échanges dans les systèmes pair-à-pair. Plusieurs propriétés restent en effet encore à étudier. Une meilleure connaissance des relations entre les nœuds de grand degré dans le graphe des intérêts permettrait de savoir si ces nœuds fournissent ou demandent des objets uniquement dans leur communauté ou s'intéressent aussi à des objets extérieurs, que ces objets soient populaires ou pas. La connaissance des caractéristiques des communautés, comme leur recouvrement ou les moyens de les reconnaître selon le point de vue des nœuds et des requêtes permettrait d'affiner les algorithmes. Il serait ainsi possible par exemple de changer le comportement d'une requête selon qu'elle change de communauté ou pas. De plus, l'impact sur les communautés des nœuds de

grand degré reste à évaluer, ainsi que l'impact sur ces communautés de diverses propriétés précédemment étudiées indépendamment comme :

- la popularité des objets parmi les nœuds [29, 62] ;
- le rapport entre la proximité d'intérêt et la proximité physique des nœuds [29] ;
- le regroupement des objets partagés par les nœuds en communautés [62] ;
- la corrélation entre les fichiers sur un même nœud selon leur rareté [29].
- la proportion de voisins égoïstes.

La caractérisation du dynamisme d'un réseau pair-à-pair servirait à mieux comprendre la régularité des requêtes selon les nœuds. La découverte des propriétés caractéristiques des systèmes pair-à-pair et une meilleure connaissance de celles-ci permettrait à terme la conception d'un modèle des relations entre les nœuds des réseaux pair-à-pair, ce qui constituerait une avancée majeure dans ce domaine. Les quelques propositions de modèles [65, 102] qui ont été définis jusque là sont encourageantes et permettent dores et déjà une modélisation des propriétés élémentaires des systèmes pair-à-pair. Des améliorations autoriseraient par exemple des requêtes à être associées à plusieurs communautés, voire donneraient une répartition des communautés ciblées par les requêtes d'un nœud. L'amélioration de ces modèles permettrait à terme de contourner la difficulté à obtenir des traces de systèmes pair-à-pair, et ainsi de valider les propositions de nouveaux systèmes pair-à-pair et d'effectuer des comparaisons à moindre coût. Des études théoriques plus complexes basées sur ce modèle seraient alors possibles, comme par exemple l'analyse des compromis possibles pour une méthode de recherche entre d'une part la charge des nœuds et de l'autre le nombre de réponses et la pertinence de la recherche.

Si les systèmes pair-à-pair servent actuellement principalement à la recherche décentralisée de fichiers, nous avons aussi vu que leur champ d'application était bien plus large : la décentralisation à faible coût peut remplacer des services centralisés dans des réseaux à grande échelle, comme les DNS dans Internet [19], des services de téléphonie sur Internet, mais aussi améliorer les réseaux de calcul [104] voire les bases de données. En effet, si le géant des bases de données Oracle distribue depuis quelques temps une version $10g$ de son produit (g pour *grid*), c'est bien que la répartition de la charge correspond à une demande des grands acteurs de l'informatique.

Faisant suite à la création d'univers persistants (mondes virtuels) utilisés par exemple dans les jeux vidéos en ligne, des univers persistants pair-à-pair [94] ont déjà fait leur apparition. Ils présentent l'avantage de faire supporter la charge de leur fonctionnement aux utilisateurs, permettant ainsi l'indépendance et l'autonomie du monde virtuel.

D'autre part, si les systèmes de calcul à grande échelle ont longtemps préféré l'utilisation des grilles de calcul, des projets ont débuté il y a plusieurs années pour tirer profit de la puissance de calcul inutilisée des parcs informatiques de grands organismes. Des systèmes comme [104] tentent d'intégrer des outils d'algorithmique répartie pour assurer des garanties actuellement difficile à obtenir avec des systèmes pair-à-pair.

Dans le domaine des services pour les réseaux à grande échelle, comme les DNS, des propositions comme [19] ont montré tout l'intérêt des réseaux à contenu adressable. La téléphonie sur Internet proposée par skype [92] permet de tirer parti de la garantie de performances des réseaux à contenu adressable. Une partie de l'administration de ces

services semble toutefois encore limiter la décentralisation, dans le cas des DNS, il s'agit de l'enregistrement de noms de domaines, de la vérification et la rétribution éventuelle de ce service.

Concernant les bases de données, les systèmes pair-à-pair semblent la prochaine étape de la décentralisation entamée ces dernières années Toutefois, les bases de données aujourd'hui déployées sur de très grands réseaux imposent des contraintes actuellement difficiles à satisfaire. Il s'agit entre autres de la cohérence, la persistance, la gestion des droits pour les différents accès aux données, et la disponibilité. L'assurance de délais de réponses courts est aussi une contrainte essentielle en plus du support de requêtes évoluées (recherche d'expressions rationnelles et recherche par intervalles) pour des utilisateurs que le modèle client-serveur a habitué a des garanties fortes. Les réseaux à contenu adressable permettent aujourd'hui de garantir des temps de réponses faibles sans permettre les requêtes évoluées, tandis que les systèmes non structurés sont dans le cas inverse. Il faut toutefois noter une amélioration de l'expressivité des requêtes de réseaux à contenu adressable avec l'apparition de systèmes capables d'effectuer des requêtes par intervalles, comme [8, 54], basés sur les listes à enjambement [81] (aussi nommées skip lists). Quel que soit le système, la cohérence des données doit faire l'objet d'un compromis avec les temps de réponse du fait de la transmission des messages par un réseau physique asynchrone. La persistance des données dans un système pair-à-pair peut être assurée par des mécanismes de réplication tels qu'on les a vus dans ce manuscrit. La sécurité des données, la gestion des droits de création, d'ajout et de consultation des données peut s'inspirer des méthodes utilisées dans des systèmes comme Freenet [34]. De toutes ces contraintes, autoriser les recherches évoluées semblent la contrainte la plus difficile à adapter aux réseaux à contenu adressable C'est pourquoi les systèmes décentralisés non-structurés me semblent les plus prometteurs pour la conception de bases de données réparties.

Ces différents horizons ouverts aux systèmes pair-à-pair permettraient de remplacer des systèmes centralisés, et nécessiteront donc l'assurance de certaines garanties, même si certaines contraintes sont relâchées. Enfin, le développement d'algorithmes auto-stabilisants pourraient être une solution pour assurer le fonctionnement des services malgré le départ et l'arrivée régulières de nœuds, voire le comportement byzantins d'une partie des nœuds.

Bibliographie

[1] Ittai Abraham, Baruch Awerbuch, Yossi Azar, Yair Bartal, Dahlia Malkhi, and Elan Pavlov. A generic scheme for building overlay networks in adversial scenarios. In *17th Int. Parallel and Distributed Processing Symposium (IPDPS)*, 2003. `http://research.microsoft.com/~dalia/pubs/ipdps3.ps`.

[2] Ittai Abraham, Dahlia Malkhi, and Oren Dobzinski. LAND : Locality aware networks for distributed hash tables. Technical Report TR 2003-75, Leibnitz Center, The Hebrew University, June 2003. `http://www.cs.huji.ac.il/~ittaia/papers/LAND-TR.pdf`.

[3] Ittai Abraham, Dahlia Malkhi, and Oren Dobzinski. LAND : Stretch $(1 + \varepsilon)$ locality aware networks for DHTs. In *Proceedings of the ACM-SIAM Symposium on Discrete Algorithms (SODA04)*, 2004. `http://research.microsoft.com/~dalia/pubs/AMD04.pdf`.

[4] L. Adamic, R. Lukose, A. Puniyani, and B. Huberman. Search in power law networks. *Physical Review E*, 64 :046135, 2001. `http://arxiv.org/abs/cs/0103016`.

[5] Eytan Adar and Bernardo A. Huberman. Free riding on gnutella. *First Monday 5*, October 2000. `http://www.firstmonday.dk/issues/issue5_10/adar/`.

[6] Nazareno Andrade, Miranda Mowbray, Aliandro Lima, Gustavo Wagner, and Matei Ripeanu. Influences on cooperation in bittorrent communities. In *Proceedings of the 3rd Workshop on Economics of P2P Systems (P2P Econ)*, 2005. `http://www.acm.org/sigs/sigcomm/sigcomm2005/paper-AndMow.pdf`.

[7] Stefan Saroiu ans P. Krishna Gummadi and Steven D. Gribble. A measurement study of peer-to-peer file sharing systems. In *Proceedings of Multimedia Computing and Networking (MMCN)*, San Jose, CA, USA, January 2002. ACM Press. `http://www.mpi-sws.mpg.de/~gummadi/papers/mmcn.pdf`.

[8] James Aspnes and Gauri Shah. Skip graphs. In *Fourteenth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 384–393, January 2003.

[9] Yossi Azar, Andrei Z. Broder, Anna R. Karlin, and Eli Upfal. Balanced allocations. *SIAM Journal on Computing*, 29(1) :180–200, 2000. `http://www.cs.brown.edu/people/eli/papers/SICOMP29.pdf`.

[10] Albert-Laszlo Barabasi and Reka Albert. Emergence of scaling in random networks. *Science*, 286 :509, 1999.

[11] Jean-Claude Bermond and Claudine Peyrat. de Bruijn and Kautz networks : a competitor for the hypercube ? In François André and Jean-Pierre Verjus, editors, *Proceedings of the 1st European Workshop on Hypercubes and Distributed Computers, Rennes*, pages 279–293. North Holland, 1989.

[12] Burton H. Bloom. Space/time trade-offs in hash coding with allowable errors. *Commun. ACM*, 13(7) :422–426, 1970. `http://gnunet.org/papers/p422-bloom.pdf`.

[13] Nicolas Bonnel and Fabienne Moreau. Quel avenir pour les moteurs de recherche ? In Sylvie Saget Alexandre Vautier, editor, *Actes de MajecSTIC 2005 : Manifestation des Jeunes Chercheurs francophones dans les domaines des STIC*, pages 291–299. IRISA and IETR and LTSI, novembre 2005. `http://www.irisa.fr/texmex/people/bonnel/publis/Bonnel_MAJECSTIC05_web.pdf`.

[14] Yann Busnel and Anne-Marie Kermarrec. Integrating file popularity and peer generosity in proximity measure for semantic-based overlays. Technical Report 1756, ISISA, october 2005. `http://hal.inria.fr/inria-00000502/en/`.

[15] Vincent Cholvi, Pascal A. Felber, and Ernst W. Biersack. Efficient search in unstructured peer-to-peer networks. *European Transactions on Telecommunications, Special Issue on P2P Networking and P2P Services*, 15(6), December 2004.

[16] Ian Clarke, Scott G. Miller, Theodore W. Hong, Oskar Sandberg, and Brandon Wiley. Protecting free expression online with Freenet. *IEEE Internet Computing*, 2002. `http://freenet.sourceforge.net/papers/freenet-ieee.pdf`.

[17] Ian Clarke, Oskar Sandberg, Brandon Wiley, and Theodore W. Hong. Freenet : A distributed anonymous information storage and retrieval system. In Hannes Federrath, editor, *Proceedings of Designing Privacy Enhancing Technologies : Workshop on Design Issues in Anonymity and Unobservability*, volume 2009 of *Lecture Notes in Computer Science*, pages 46–66, Berkeley, CA, USA, July 2000. Springer. `http://www.cl.cam.ac.uk/~twh25/academic/papers/icsi-revised/`.

[18] Brahm Cohen. Incentives build robustness in bittorrent, May 2003. `http://www.bittorrent.com/bittorrentecon.pdf`.

[19] Russ Cox, Athicha Muthitacharoen, and Robert T. Morris. Serving dns using a peer-to-peer lookup service, March 2002. `http://pdos.csail.mit.edu/chord/papers/ddns.pdf`.

[20] Mayur Datar. Butterflies and peer-to-peer networks. In *European Symp. on Algorithms*, number 2461 in LNCS, page 310. Springer, 2002. `http://dbpubs.stanford.edu:8090/pub/2002-33`.

[21] Dc++. `http://www.dcpp.net/`.

[22] Nicolaas de Bruijn. A combinatorial problem. *Koninklijke Nederlandse Academie van Wetenschappen Proc.*, 49, 1946.

[23] Doubleclick. `http://emea.doubleclick.com/fr/`.

[24] Doubleclick piraté. `http://www.transfert.net/a4800`.

[25] *unofficial* edonkey protocol specification v0.6.2. `http://prdownloads.sourceforge.net/pdonkey/eDonkey-protocol-0.6.2?download`.

[26] edonkey2000. `http://www.edonkey2000.com`.

[27] Slashdot effect. `http://en.wikipedia.org/wiki/Slashdot_effect`.

[28] Fasttrack. `http://cvs.berlios.de/cgi-bin/viewcvs.cgi/*checkout*/gift-fasttrack/giFT-FastTrack/PROTOCOL?rev=1.19`.

[29] Fabrice Le Fessant, Sidath Handurukande, Anne-Marie Kermarrec, and Laurent Massoulié. Clustering in peer-to-peer file sharing workloads. In *Proceedings of the 3rd International Workshop on Peer-to-Peer Systems (IPTPS)*, pages 217–226, 2004. `http://iptps04.cs.ucsd.edu/papers/le-fessant-clustering.pdf`.

[30] Fleming file sharing system. `http://sourceforge.net/projects/ffss/`.

[31] Amos Fiat and Jared Saia. Censorship resistant peer-to-peer content-addressable networks. In *Proceedings of the thirteenth annual ACM-SIAM symposium on Discrete algorithms (SODA)*, pages 94–103, Philadelphia, PA, USA, 2002. Society for Industrial and Applied Mathematics. `http://www.cs.unm.edu/~saia/censor.pdf`.

[32] Pierre Fraigniaud and Philippe Gauron. An overview of the content addressable network D2B. In *Proceedings of the 22nd ACM Symposium on Principles of Distributed Computing (PODC'03)*, jul 2003. `http://www.lri.fr/~gauron/articles/D2B-PODC03.ps`.

[33] Pierre Fraigniaud and Philippe Gauron. D2B : a de bruijn based content-addressable network. *Theoretical Computer Science, Special Issue on Complex Systems*, 2006.

[34] Freenet. `http://freenet.sourceforge.net/`.

[35] Freepastry. `http://freepastry.org`.

[36] Anh-Tuan Gai. *Autour de l'accessibilité, de la distribution et de la pérennité de données dans les réseaux de pair-à-pair*. PhD thesis, Université Paris 6 - Pierre et Marie Curie, 2006.

[37] Luis Garces-Erice, Keith W. Ross, Ernst W. Biersack, Pascal A. Felber, and Guillaume Urvoy-Keller. Topology-centric look-up service. In *Proceedings of COST264 Fifth International Workshop on Networked Group Communications (NGC)*, Munich, Germany, 2003. http://members.unine.ch/pascal.felber/publications/NGC-03.pdf.

[38] interface gift-fasttrack. http://cvs.sourceforge.net/viewcvs.py/gift/giFT/README?rev=1.5.

[39] Gnutella v0,4, 2000. http://rfc-gnutella.sourceforge.net/developer/stable/index.html.

[40] Bandwidth barriers to gnutella network scalability, September 2000. http://web.archive.org/web/20001216071600/dss.clip2.com/dss_barrier.html.

[41] Gnutella : To the bandwidth barrier and beyond, November 2000. http://web.archive.org/web/20010202143400/dss.clip2.com/gnutella.html.

[42] Clip2 dss : Gnutella. http://web.archive.org/web/20010505085531/dss.clip2.com/, le site a disparu mais est disponible en passant par un archiveur d'Internet.

[43] Gnutella improvements, March 2001. http://www.cs.iit.edu/~yee/classes/cs595f03/gnutellaimprove.html.

[44] Gnutella proposals. http://rfc-gnutella.sourceforge.net/Proposals/.

[45] The gnutella protocol specification v0.4, 2000. Document revision 1.0 http://web.archive.org/web/20010607053501/http://dss.clip2.com/GnutellaProtocol04.pdf.

[46] The gnutella protocol specification v0.4, 2000. Document revision 1.2 http://www9.limewire.com/developer/gnutella_protocol_0.4.pdf.

[47] Clip2 reflector, 2000. http://web.archive.org/web/20010408224017/dss.clip2.com/reflector.html.

[48] Rfc : The gnutella protocol specification v0.6. http://rfc-gnutella.sourceforge.net/developer/testing/index.html.

[49] David Goldschlag, Michael Reed, and Paul Syverson. Onion routing for anonymous and private internet connections. *Communications of the ACM*, 42(2) :39–41, 1999. http://www.onion-router.net/Publications/CACM-1999.pdf.

[50] Jean-Loup Guillaume, Matthieu Latapy, and Stevens Le-Blond. Statistical analysis of a P2P query graph based on degrees and their time-evolution. In *Proceedings of the 6th International Workshop on Distributed Computing (IWDC 04)*, pages 126–137, 2004. http://hal.ccsd.cnrs.fr/ccsd-00016859/en/.

[51] Gwebcache, 2000. http://www.gnucleus.com/gwebcache/.

[52] Elyès Ben Hamida. Déploiement du protocole D2B pour les réseaux pair à pair, janvier 2004.

[53] Sidath Handurukande, Anne-Marie Kermarrec, Fabrice Le Fessant, and Laurent Massoulié. Exploiting semantic clustering in the edonkey p2p network. In *Proceedings of the 11th ACM SIGOPS European Workshop*, pages 109–114, September 2004.

[54] Nicholas J. A. Harvey, Michael B. Jones, Stefan Saroiu, Marvin Theimer, and Alec Wolman. Skipnet : A scalable overlay network with practical locality properties. In *USENIX Symposium on Internet Technologies and Systems*, pages 113–126, 2003.

[55] Oliver Heckmann and Axel Bock. The edonkey 2000 protocol. http://prdownloads.sourceforge.net/pdonkey/eDonkey-protocol-0.6.2?download.

[56] Kirsten Hildrum, John Kubiatowicz, Satish Rao, and Ben Y. Zhao. Distributed object location in a dynamic network. In *14th ACM Symp. on Parallel Algorithms and Architecture (SPAA)*, 2002. http://oceanstore.cs.berkeley.edu/publications/papers/pdf/SPAA02.pdf.

[57] M. Frans Kaashoek and David R. Karger. Koorde : A simple degree-optimal distributed hash table. In *Proceedings of the 2nd International Workshop on Peer-to-Peer Systems (IPTPS)*, volume 2735 of *Lecture Notes in Computer Science*, pages 98–107, Berkeley, CA, October 2003. Springer-Verlag. http://project-iris.net/irisbib/papers/koorde:iptps03/paper.pdf.

[58] David Karger, Eric Lehman, Tom Leighton, Mathhew Levine, Daniel Lewin, and Rina Panigrahy. Consistent hashing and random trees : Distributed caching protocols for relieving hot spots on the world wide web. In *ACM Symposium on Theory of Computing*, pages 654–663, may 1997. http://theory.lcs.mit.edu/~karger/Papers/web.ps.

[59] Kazaa. http://www.kazaa.com.

[60] Beom Jun Kim, Chang No Yoon, Seung Kee Han, and Hawoong Jeong. Path finding strategies in scale-free networks. *Physical Review E*, 65 :027103, 2002. http://arxiv.org/abs/cond-mat/0111232.

[61] Yoram Kulbak and Danny Bickson. The emule protocol speficiation. Technical Report TR-2005-03, Leibniz Center, School of Computer Science and Engineering, The Hebrew University, December 2001. http://www.cs.huji.ac.il/labs/danss/p2p/eMule/emule.pdf.

[62] Stevens Le-Blond, Jean-Loup Guillaume, and Matthieu Latapy. Clustering in P2P exchanges and consequences on performances. In *Proceedings of the 4th*

International Workshop on Peer-To-Peer Systems (IPTPS), pages 193–204, 2005. http://hal.ccsd.cnrs.fr/ccsd-00016820/en/.

[63] Nathaniel Leibowitz, Matei Ripeanu, and Adam Wierzbicki. Deconstructing the kazaa network. In *Proceedings of the 3rd WIAPP*, pages 112–120, Washington, DC, USA, 2003. IEEE Computer Society. http://people.cs.uchicago.edu/~matei/PAPERS/kazaa.pdf#search='Deconstructing.

[64] Jie Lu and Jamie Callan. Content-based retrieval in hybrid peer-to-peer networks. In *Proceedings of the twelfth international conference on Information and knowledge management (CIKM)*, pages 199–206, New York, NY, USA, 2003. ACM Press. http://www-2.cs.cmu.edu/~jielu/Papers/cikm03_jielu_irp2p.pdf.

[65] Qin Lv, Pei Cao, Edith Cohen, Kai Li, and Scott Shenker. Search and replication in unstructured peer-to-peer networks, 2002.

[66] Nancy Lynch, Dahlia Malkhi, and David Ratajczak. Atomic data access in content-addressable networks : A position paper. In *Proceedings of 1st Int. Workshop on Peer-to-Peer Systems (IPTPS)*, 2002. http://www.cs.rice.edu/Conferences/IPTPS02/137.pdf.

[67] Nancy Lynch, Dahlia Malkhi, and David Ratajczak. Atomic data access in distributed hash tables. In *Peer-to-Peer Systems*, volume 2429 of *LNCS Hot series*, page 295. Springer-Verlag, 2002.

[68] Dahlia Malkhi, Moni Naor, and David Ratajczak. Viceroy : a scalable and dynamic lookup network. In *Proceedings of the 21st ACM Symp. on Principles of Distributed Computing (PODC)*, pages 183–192, 2002. http://www.cs.huji.ac.il/~dalia/pubs/MNR02.ps.gz.

[69] Gurmeet Singh Manku, Mayank Bawa, and Prabhakar Raghavan. Symphony : Distributed hashing in a small world. In *Proc. 4th USENIX Symposium on Internet Technologies and Systems (USITS 2003)*, 2003. http://www-db.stanford.edu/~bawa/Pub/symphony.pdf.

[70] Petar Maymounkov and David Mazieres. Kademlia : A peer-to-peer information system based on the XOR metric. In *Proceedings of 1st Int. Workshop on Peer-to-Peer Systems (IPTPS)*, March 2002. http://scs.cs.nyu.edu/~petar/kpos.ps.

[71] Gordon Mohr. Hash/urn gnutella extensions (huge) v0.94, April 2002. http://rfc-gnutella.sourceforge.net/src/draft-gdf-huge-0_94.txt.

[72] Moni Naor and Udi Wieder. Novel architectures for P2P applications : the continuous-discrete approch. In *Proceedings of the 15th ACM Symp. on Parallelism in Algorithms and Architectures (SPAA)*, pages 50–59, New York, NY, USA, 2003. ACM Press. http://www.wisdom.weizmann.ac.il/~uwieder/papers/spaa03camera.ps.

[73] Napster. `http://www.napster.com/`, Napster ayant été racheté, c'est devenu une société vendant de la musique en ligne.

[74] Napster messages. `http://opennap.sourceforge.net/napster.txt`.

[75] Onion router. `http://www.onion-router.net/`.

[76] Opennap. `http://opennap.sourceforge.net`.

[77] Opennap-ng. `http://opennap-ng.sourceforge.net`.

[78] Overnet. `http://www.overnet.com`, il s'agit du même site qu'eDonkey/eDonkey2000 depuis la fusiondes deux systèmes.

[79] Overpeer. `http://www.overpeer.com/`.

[80] P2p overflow par cocky. `http://www.cocky.fr/`.

[81] William Pugh. Skip lists : a probabilistic alternative to balanced trees. In *Communications of the ACM*, volume 33(6), pages 668–676, 1990 June.

[82] Martin Raab and Angelika Steger. Balls into bins – a simple and tight analysis. In *2nd Workshop on Randomization and Approximation Techniques in Computer Science (RANDOM)*, volume 1518 of *LNCS*. Springer, 1998. `http://wwwmayr.informatik.tu-muenchen.de/personen/raab/publ/balls.pdf`.

[83] Sylvia Ratnasamy, Paul Francis, Mark Handley, Richard Karp, and Scott Schenker. A scalable content-addressable network. In *Proceedings of SIGCOMM*, pages 161–172, New York, NY, USA, August 2001. ACM Press. `http://www.icir.org/sylvia/cans.ps`.

[84] Razorback. `http://www.razorback2.com/fr/statistics`.

[85] Retspan. `http://www.retspan.info/`.

[86] Matei Ripeanu. Peer-to-peer architecture case study : Gnutella network. In *Proceddings of the First International Conference on Peer-to-Peer Computing (P2P)*, pages 99–100, Washington, DC, USA, 2001. IEEE Computer Society.

[87] Christopher Rohrs. Query routing for the gnutella network. Technical report, Lime Wire LLC, December 2001. crohrs@limewire.com `http://rfc-gnutella.sourceforge.net/Proposals/QRP/query_routing.htm`.

[88] Antony Rowstron and Peter Druschel. Pastry : Scalable, distributed object location and routing for large-scale peer-to-peer systems. In *Proceedings of the 18th IFIP/ACM International Conference on Distributed Systems Platforms (Middleware)*, November 2001. `http://www.cs.cornell.edu/People/egs/615/pastry.pdf`.

[89] Stefan Saroiu, Krishna P. Gummadi, and Steven D. Gribble. Measuring and analyzing the characteristics of napster and gnutella hosts. *Multimedia Syst.*, 9(2) :170–184, 2003. `http://www.cs.toronto.edu/~stefan/publications/mm_systems_journal/2002/mm.pdf`.

[90] Nima Sarshar, P. Oscar Boykin, and Vwani P. Roychowdhury. Percolation search in power law networks : Making unstructured peer-to-peer networks scalable. In *Proceedings of IEEE International Conference on Peer-to-Peer Computing (P2P)*, 2004.

[91] Anurag Singla and Christopher Rohrs. Ultrapeers : Another step towards gnutella scalability. Technical report, Lime Wire LLC, December 2001. {anuarg,crohrs}@limewire.com `http://rfc-gnutella.sourceforge.net/Proposals/Ultrapeer/Ultrapeers.htm`.

[92] Skype. `http://www.skype.com/`.

[93] Slyck. `http://www.slyck.com`.

[94] Solipsis. `http://solipsis.netofpeers.net`.

[95] Kunwadee Sripanidkulchai, Bruce Maggs, and Hui Zhang. Efficient content location using interest-based locality in peer-to-peer systems. In *Proceedings of the 22nd Annual Joint Conference of the IEEE Computer and Communications Societies (INFOCOM)*, April 2003.

[96] Ion Stoica, Robert Morris, David Karger, M. Frans Kaashoek, and Hari Balakrishnan. Chord : A scalable peer-to-peer lookup service for internet applications. In *Proceedings of SIGCOMM*, volume 31, pages 149–160, New York, NY, USA, September 2001. ACM Press. `http://nms.lcs.mit.edu/papers/chord.html`.

[97] Ion Stoica, Robert Morris, David R. Karger, M. Frans Kaashoeck, and Hari Balakrishnan. Chord : A scalable peer-to-peer lookup protocol for internet applications. Technical Report MIT-LCS-TR-819, MIT LCS, 2001. `http://pdos.csail.mit.edu/chord/papers/chord-tn.ps`.

[98] Stun. `http://fr.wikipedia.org/wiki/STUN`.

[99] Kelly Truelove. Gnutella : Alive, well and changing fast, January 2001. `http://www.openp2p.com/lpt/a/573`.

[100] Anh tuan Gai and Laurent Viennot. Broose : a practical distributed hashtable based on the de bruijn topology. Technical Report 5238, INRIA, mars 2004. `http://www.inria.fr/rrrt/rr-5238.html`.

[101] Anh tuan Gai and Laurent Viennot. Broose : A practical distributed hashtable based on the de-brujin topology. In *4th IEEE International Conference on Peer-to-Peer Computing (P2P)*, August 2004.

[102] Spyros Voulgaris, Anne-Marie Kermarrec, Laurent Massoulié, and Maarten van Steen. Exploiting semantic proximity in peer-to-peer content searching. In *Proceedings of the 10th IEEE Int. Workshop on Future Trends in Distributed Computing Systems (FTDCS)*, 2004.

[103] Duncan J. Watts and Steven H. Strogatz. Collective dynamics of "small-world" networks. *Nature*, 393 :440–442, 1998.

[104] Xtremweb. `http://www.xtremweb.net`.

[105] Ben Y. Zhao, John Kubiatowicz, and Anthony D. Joseph. Tapestry : An infrastructure for fault-tolerant wide-area location and routing. Technical Report UCB/CSD-01-1141, U. C. Berkeley, 2001. `http://www.cs.berkeley.edu/~ravenben/publications/CSD-01-1141.pdf`.

Table des figures

Résumé

Les systèmes pair-à-pair décentralisés mettent en relation un grand nombre d'utilisateurs pour mutualiser des ressources dans un environnement dynamique. Les applications de ces systèmes vont du partage de fichiers à la téléphonie par Internet, en passant par la décentralisation de services comme les DNS. Afin de limiter les ressources nécessaires, chaque recherche doit transmettre les messages rapidement et contrôler le nombre de messages dans le réseau. C'est pourquoi cette thèse s'intéresse au routage et à l'interconnexion nécessaire pour assurer ce routage.

Dans la première partie, je présente un état de l'art des différents systèmes existants. J'y détaille la problématique générale et les caractéristiques des différentes classes de systèmes pair-à-pair.

La seconde partie traite des réseaux à contenu adressable, qui permettent d'assurer un routage avec des bornes sur le nombre de sauts des requêtes et la charge par noeud. J'y expose d'abord les différents réseaux à contenu adressable existants. Je présente ensuite un nouveau protocole, D2B, dont je prouve en particulier qu'il améliore la charge par noeud tout en assurant un nombre de sauts et un degré faible. Je détaille enfin une liste d'optimisations applicables aux réseaux à contenu adressable en général, ou à D2B en particulier.

Dans la troisième partie, je récapitule les principaux travaux qui tirent parti des communautés d'utilisateurs d'une part, et ceux qui exploitent la structure de loi de puissance de l'autre. Je présente ensuite ma seconde contribution, QRE, qui exploite efficacement à la fois ces deux caractéristiques des échanges : agrégats en communautés et loi de puissance.

Abstract

Decentralized peer-to-peer systems allow a large number of users to interact in order to gather resources in a dynamic environment. These systems have applications in many areas : file sharing, telephony on Internet, DNS decentralization, etc. Each query has to quickly forward messages to bound resources and to control the number of messages in the network. Thus, this thesis focuses on routing and interconnection needed to handle such a routing.

In the first part, I present a state of the art of the different existing systems. I develop the general issue and the different classes of peer-to-peer systems.

The second part covers content-addressable networks which allow to assure bounds on the number of hops and the nodes' load. I first expose the existing content-addressable networks. I then present the new protocol D2B and prove that it enhances the nodes' load as well as it assures a low number of hops and a low degree. At last, I list optimizations which can be used in content-addressable networks in general, or in D2B in particular.

In the third part I summarize the main results which use clustering in networks on one hand, and exploit power law property on the other hand. I present then my second contribution, QRE. It efficiently exploits both properties of exchanges : clustering in communities and power law.