# Freenet : Un Système Distribué de Stockage et de Récupération d'Informations

Ian Clarke[1], Oskar Sandberg[2], Brandon Wiley[3], and Theodore W. Hong[4] *

[1] Uprizer, Inc., 1007 Montana Avenue #323, Santa Monica, CA 90403, USA
ian@octayne.com
[2] Department of Numerical Analysis and Computer Science, Royal Institute of Technology, SE-100 44 Stockholm, Sweden
md98-osa@nada.kth.se
[3] College of Communication, University of Texas at Austin, Austin, TX 78712, USA
blanu@uts.cc.utexas.edu
[4] Department of Computing, Imperial College of Science, Technology and Medicine, 180 Queen's Gate, London SW7 2BZ, United Kingdom
t.hong@doc.ic.ac.uk

**Résumé** Nous décrivons Freenet, une application de réseau adaptatif d'égal à égal qui permet la publication, la duplication et la récupération des données tout en préservant l'anonymat à la fois des auteurs et des lecteurs. Freenet opère comme un réseau de nodes identiques qui partagent leur espace de mémoire pour stocker des fichiers de données et coopèrent pour router les requêtes vers la zone la plus probable du stockage physique de l'information. Freenet n'utilise ni de recherche par émission ni d'index centralisé localisé. Les fichiers sont référencés d'une manière indépendante de leur localisation. Ils sont dynamiquement dupliqués dans des zones plus proches des demandeurs et ils sont effacés des zones où ils ne présentent pas d'intérêt. Il est impossible de découvrir la vraie origine ou la destination d'un fichier traversant le réseau et difficile pour un opérateur de node de déterminer ou d'être responsable du contenu physique de son propre node.

## 1 Introduction

Les réseaux d'ordinateurs prennent de plus en plus d'importance comme moyen de stockage et d'échange d'informations. Néanmoins, les systèmes actuels n'offrent que peu d'intimité à leurs utilisateurs et stockent habituellement toutes les données sur un seul ou peu d'endroits fixés, créant un point central de panne. À cause du désir permanent des individus de protéger leur intimité comme auteur ou lecteur de types variés d'informations[28] et l'indésirabilité de points centraux de défaillance qui peuvent être attaqués par la partie adverse voulant enlever les données du système [11,27] ou tout simplement surchargés par un surcroit d'intérêt[1], les systèmes offrants une plus grande sécurité et fiabilité sont nécessaires.

* Le travail de Theodore W. Hong a été aidé par des dons provenant du Marshall Aid Commemoration Commission et du National Science Foundation.

Nous sommes en train de développer Freenet, un système de stockage et de récupération distribué de l'information destiné à tenir compte de ces propos d'intimité et de disponibilité. Le système opère comme un système de fichiers distribués indépendamment de la localisation des multiples ordinateurs individuels qui permettent aux fichiers d'être insérés et demandés anonymement. Il existe cinq buts lors de la conception :

- Anonymat à la fois pour les producteurs et les consommateurs d'informations
- Déresponsabilisation pour les stockeurs d'informations
- Résistance aux attaques des tierces personnes visant à refuser l'accès aux informations
- Stockage et routage dynamique des informations
- Décentralisation de toutes les fonctions réseau

Le système est conçu pour répondre de manière adaptative afin de permettre la transparence des déplacements , duplications et effacements des fichiers selon ce qui est nécessaire pour fournir un service efficace sans avoir recours à des recherches par émissions ou des index centralisés. Il n'est pas prévu pour garantir un stockage permanent de fichiers, même si on espère que suffisamment de nodes vont se connecter pour offrir de la capacité de stockage, permettant aux fichiers de rester indéfiniment. De plus, le système opère au niveau de la couche d'application et assure l'existence d'une couche de transport sécurisée, indépendante du type de transport. Il ne cherche pas à fournir l'anonymat pour l'utilisation générale du réseau mais seulement aux transactions des fichiers Freenet.

Freenet est actuellement développé comme un projet logiciel libre sur Sourceforge, et les implémentations préléminaires peuvent être téléchargées sur `http://www.freenetproject.org/`. Il a été démarré à partir du travail originellement fait par le premier auteur à l'Université de Edinburgh[12].

## 2 Travail apparenté

Plusieurs travaux apparentés non aboutis peuvent être distingués dans le même domaine. Des canaux anonymes d'égal à égal basés sur le schéma de réseau mixé de Chaum[8] ont été implémentés pour les emails par le Mixmaster remailer[13] et pour le traffic général sous TCP/IP par le routage en pelure d'oignon[19] et Freedom[32]. De tels canaux, néanmoins, ne sont pas en eux-même facilement adaptés pour les publications à large diffusion et ils sont plus vus comme des compléments à Freenet car ils ne fournissent ni stockage ni accès à des fichiers.

L'anonymat pour les consommateurs d'informations dans le contexte du web est fourni par les services du proxy du navigateur tel que Anonymizer[6], bien qu'ils ne fournissent pas de protection pour les producteurs d'informations et ne protègent pas les consommateurs contre les logs gardés par les services eux-mêmes. Le schéma de récupération des informations[10] fournit de plus fortes garanties pour les consommateurs d'informations mais seulement par la dissimulation de l'adresse du serveur de stockage de telle information trouvée. Dans la plupart des cas, le fait de contacter un serveur particulier, en lui-même, révèle beaucoup sur l'information retrouvée, ce qui peut-être contré en permettant à chaque serveur de détenir toutes les informations (c'est naturellement peu efficace). Le travail qui se rapproche le plus du nôtre est le système Crowds de

Reiter et Rubin[25], qui utilise une méthode similaire de requête par proxies pour les consommateurs bien que Crowds ne stocke pas lui-même les informations et ne protège pas les producteurs d'informations. Berthold *et al.* propose des Web Mixes[7], un système plus robuste qui transforme les messages en les gonflant, en les réorganisant, en leur adjoingnant des idioties pour augmenter la sécurité mais, encore une fois, cela ne protège pas les producteurs d'informations.

Le Rewebber[26] fournit une mesure de l'anonymat de producteurs d'information sur le web par le moyen d'un service URL crypté qui est essentiellement l'inverse d'un proxy anonymiseur de navigateur mais il a les mêmes difficultés pour fournir des protections contre l'opérateur du service. TAZ[18] a étendu cette idée en utilisant des chaînes imbriquées d'URLs cryptés qui pointent successivement sur des serveurs de redirection, bien qu'elles soient vulnérables à l'analyse répétée du trafic. Les deux reposent sur un simple serveur comme ressource ultime d'information. Publius[30] améliore la disponibilité en distribuant des parties de fichiers de manière à augmenter la redondance parmi $n$ serveurs de web, seulement $k$ d'entre eux sont nécessaires pour reconstruire le fichier; néanmoins, comme l'identité des serveurs eux-même n'est pas anonyme, une attaque peut enlever les informations en forçant la fermeture de $n$–$k$+1 serveurs. La proposition Eternity[5] a pour but d'archiver les informations de manière permanente et anonyme bien qu'il ne spécifie pas comment localiser les fichiers stockés, rendant plus difficile le service de mise à jour. Free Haven[14] est un système de publication anonyme intéressant qui utilise un réseau de confiance et un mécanisme d'échange de fichiers pour fournir un gros compte de serveurs tout en préservant l'anonymat.

`distributed.net`[15] a démontré le concept de partage de ressources d'ordinateurs parmi de multiples ordinateurs sur une grande échelle de cycles CPU; Napster[24] et Gnutella[17] ont fait la même chose pour l'espace disque, bien que le premier ne soit relié à un serveur central que pour localiser les fichiers et que le second utilise un système de recherche d'adresses inefficace. Aucun des deux ne duplique les fichiers. Intermemory[9] et India[16] sont des systèmes de serveurs de fichiers distribués coopératifs conçus pour de l'archivage à long terme à la mode Eternity, dans lequel les fichiers sont découpés en parts redondantes et distribuées parmi tous les participants. Akamai[2] fournit un service qui réplique les fichiers proche de la zone de consommation mais n'est pas approprié pour les producteurs qui sont des individus (à l'opposé des corporations). Aucun de ces systèmes ne tente de fournir l'anonymat.

## 3 Architecture

Freenet est implémenté comme un réseau d'égal à égal de nodes qui se contactent les uns les autres pour stocker et délivrer des fichiers de données, représentés par des clés indépendantes de la localisation. Chaque node gère son propre stock local de données, qu'il met à la disposition du réseau tant en lecture qu'en écriture, ainsi qu'une table de routage dynamique contenant les adresses des autres nodes et des clés qu'ils sont susceptibles de détenir. Cela signifie que la plupart des utilisateurs du système font tourner des nodes, à la fois pour donner des garanties de sécurité contre l'utilisation par mégarde d'un node étranger et pour augmenter la capacité globale disponible sur le réseau.

Le système peut être considéré comme un système de fichiers distribués incorporant une indépendance à la localisation et des copies faciles transparentes. De la même manière que les systèmes comme `distributed.net`[15] permettent aux utilisateurs ordinaires de partager les cycles CPU inutilisés sur leur machine, Freenet permet aux utilisateurs de partager l'espace disque inutilisé. Néanmoins, là où `distributed.net` utilise les cycles CPU pour son propre intérêt, Freenet est directement utilisable par les utilisateurs eux-même, agissant comme une extension de leur propre disque dur.

Le modèle de base implique que les demandes passent de nodes en nodes dans une chaîne de requêtes de proxies, avec chaque node prenant la décision du routage local, à la manière du routage IP (Internet Protocol). Selon les clés demandées, les routes seront différentes. Les algorithmes de routage pour le stockage et la restitution des données décrits dans les section suivantes sont conçus pour adapter les routes en permanence pour fournir une performance suffisante en utilisant une connaissance locale plutôt que globale. Les nodes ne connaissent dans la chaîne que leurs voisins immédiats pour les flux entrants et sortants. Chaque requête est accompagnée d'un compteur (appellé "hops to live" par analogie avec les time-to-live des IP) décrémenté à chaque passage dans un node afin d'éviter les chaînes infinies. Chaque requête se voit assigné un identificateur pseudo-aléatoire unique, de façon à ce que les nodes puissent prévenir les boucles en rejettant les demandes qu'ils ont déjà vues. Lorsque cela se produit, le node immédiatement précédent choisit simplement un node différent pour la redirection. Ce processus continue jusqu'à ce que la requête soit satisfaite ou dépasse la limite du hops-to-live. Alors le succès ou l'échec résultant est renvoyé dans la chaîne jusqu'au node émetteur.

Aucun node n'est privilégié parmi les autres. Il n'y a donc ni hiérarchie ni point central où la panne peut exister. Joindre le réseau est simplement une question de découverte de l'adresse de l'un ou l'autre des nodes existants par des moyens hors de Freenet, et alors on peut envoyer des messages.

### 3.1 Clés et recherche

Dans Freenet, les fichiers sont identifiés par des clés binaires de fichiers obtenues en appliquant une fonction de hachage. Nous utilisons actuellement la fonction SHA-1[4] 160 bits. Trois types de clés différentes sont utilisées, dont les buts et les spécifications sont différents.

Le type de clé de fichier le plus simple est la *clé à signature de mot clé* (KSK), qui est dérivée d'un court texte de description choisi par l'utilisateur lorsqu'il stocke le fichier dans le réseau. Par exemple, un utilisateur insérant un traité sur la stratégie pourrait donner comme description, `text/philosophy/sun-tzu/art-of-war`. Cette chaîne est utilisée comme entrée pour générer de manière déterministe une paire de clés public/privée. La moitié publique est alors hachée pour produite la clé du fichier.

La moitié privée de la paire de clés asymétrique est utilisée pour signer le fichier, fournissant un contrôle minimum d'intégrité pour qu'un fichier retrouvé corresponde à sa clé de fichier. Notez, néanmoins, qu'un attaquant peu utiliser une attaque de type dictionnaire contre la signature en compilant une liste de chaînes descriptives. Le fichier est alors crypté en utilisant la chaîne descriptive elle-même comme clé, pour les raisons qui sont expliquées dans la section 3.4.

Pour permettre aux autres de retrouver le fichier, l'utilisateur a seulement besoin de publier la chaîne descriptive. Cela rend les clés à signature par mots clés faciles à se rappeler et à communiquer aux autres. Néamoins, ils forment un identifiant très banal ce qui est problématique. Rien n'empêche, par exemple, deux utilsateurs de choisir la même chaîne descriptive pour des fichiers différents ou de s'engager dans le "key-squatting"—insérer des fichiers pourris sous des descriptions populaires.

Ces problèmes sont évités par les *clés "Signed Subspace Key"* (SSK) qui permettent l'utilisation des sous-identifiants personnels. Une utilisatrice crée un espace de nommage en générant aléatoirement une paire de clés publique/privée qui servira à reconnaître son identifiant. Pour insérer un fichier, elle choisit une chaîne de texte descriptive comme précédement. La clé publique de l'identifiant et la chaîne descriptive sont hachées indépendamment, associées avec un XOR, et re-hachées pour produire la clé de fichier.

Comme avec les clés à signature par mot clé, la moitié privée de la paire asymétrique est utilisée pour signer la le fichier. Cette signature, générée à partir d'une paire de clés aléatoire, est plus sécurisée que les signatures utilisées par les clés à signatures par mots clés. Le fichier est aussi crypté par la chaîne descriptive comme avant.

Pour permettre aux autres de retrouver le fichier, l'utilisatrice publie la chaîne descriptive avec sa clé publique personalisée accompagnée du sous- identifiant personel. Le stockage des données requiert néanmoins la clé privée de telle manière que seule la propriétaire de de l'identifiant personalisé puisse y ajouter des fichiers.

La propriétaire a maintenant la possibilité de gérer son propre espace de nommage. Par exemple, elle peut simuler une structure hiérarchique en créant des répertoires de fichiers contenant des pointeurs hypertexte pour les autres fichiers. Un répertoire sous la clé `text/philosophy` pourrait contenir une liste de clés tels que `text/philosophy/sun-tzu/art-of-war`, `text/philosophy/confucius/analects`, et `text/philosophy/nozick/anarchy-state-utopia`, utilisant une syntaxe appropriée interprétable par un client. Les répertoires peuvent à leur tour pointer vers les autres répertoires.

Le troisième type de clé est la *clé à hachage de contenu* (CHK), qui est utile pour implémenter la mise à jour et le découpage de fichiers. Une clé à contenu de hachage est simplement dérivée par un hachage direct du contenu du fichier correspondant. Ceci donne à chaque fichier une clé pseudo-unique. Les fichiers sont aussi cryptés par une clé générée aléatoirement. Pour permettre aux autres de retrouver le fichier, l'utilisatrice publie la clé à hachage de contenu elle-même avec la clé de décryptage. Notez que la clé de décryptage n'est jamais stockée avec les fichiers mais est seulement publiée avec la clé du fichier, pour des raisons qui sont expliquées dans la section 3.4.

Les clés à contenu de hachage sont plus utiles en conjonction avec les clés à signature de l'identifiant utilisant des mécanismes de redirection. Pour stocker un fichier pouvant être mis à jour, une utilisatrice l'insère d'abord sous clé à hachage de contenu. Elle insère alors un fichier indirect sous une clé à signature de l'identifiant dont le contenu est celui de la clé de hachage. Ceci permet aux autres de retrouver le fichier en deux étapes, selon la clé à signature de l'identifiant.

Pour mettre à jour un fichier, la propriétaire insère d'abord une nouvelle version sous sa clé de hachage de contenu, qui doit être différente de celle de

la version précédente. Elle insère ensuite un nouveau fichier d'indirection sous la clé originale à signature de l'identifiant pointant vers la version mise à jour. Lorsqu'une insertion atteint un node qui possède l'ancienne version, une collision de clé apparaîtra. Le node contrôlera la signature de la nouvelle version, vérifiant qu'elle est valide et plus récente et remplacera l'ancienne version. De cette façon, les clés à signature de l'identifiant mèneront à la version la plus récente du fichier alors que les anciennes version pourront toujours être trouvées directement par la clé de hachage de contenu si désiré. (Si elles ne sont pas demandées, néanmoins, ces anciennes versions vont éventuellement être enlevées du réseau —voir la section 3.4.) Ce mécanisme peut être utilisé pour gérer les répertoires de même que les fichiers réguliers.

Les clés à hachage de contenu peuvent être aussi être utilisées pour découper les fichiers en plusieurs parties. Pour de grands fichiers, le découpage peut être nécessaire à cause du stockage ou des limitations de bande passante. De même, le découpage de fichiers de taille moyenne en parties à taille standard (e.g. $2^n$ kilo-octets) ont aussi des avantages en luttant contre l'analyse du traffic. C'est facilement accompli en insérant chaque partie séparément sous une clé à hachage de contenu et en créant un fichier indirect (ou des fichiers indirects à multiples niveaux) pour pointer les parties individuelles.

Tout ceci laisse le problème de retrouver les clés au premier endroit. La manière la plus directe pour ajouter des capacités de recherche à Freenet et de lancer une araignée hypertexte comme celles utilisées pour les recherches sur le web. Alors que cela semble être une solution attractive, ceci entre en conflit avec un des buts de la conception qui est d'éviter la centralisation. Une alternative possible est de créer une classe spéciale de fichiers indirects allégés. Lorsqu'un fichier est inséré, l'auteur peut aussi insérer des fichiers indirects, chacun contenant un pointeur vers le fichier réel, nommé selon les mots de recherche qu'elle a choisi. Ces fichiers indirects sont différents des fichiers normaux dans le sens où des fichiers multiples avec la même clé (i.e. recherche par mots clé) auront la permission d'exister et les requêtes pour de telles clés ne seront pas arrêtées dès le premier fichier trouvé mais continueront à chercher jusqu'à ce qu'un nombre spécifique de résultats soit accumulé au lieu de s'arrêter au premier fichier trouvé. La gestion d'un tel volume de fichiers directs est un problème qui reste ouvert.

Un mécanisme alternatif est d'encourager les individus à créer leurs propres compilations de clés favorites et de les rendre publique. C'est une approche qui est d'un usage courant dans le world-wide web.

### 3.2 Retrouver les données

Pour retrouver un fichier, une utilisatrice doit d'abord obtenir ou calculer sa clé de fichier binaire. Alors elle envoie une requête à son propre node en spécifiant cette clé et une valeur hops-to-live. Lorsqu'un node reçoit une requête, il contrôle d'abord s'il possède les données dans son propre stock et les renvoie, s'il les trouve, en ajoutant une note informant qu'il est la source de la donnée. S'il ne trouve rien, il cherche une similitude dans sa table de routage avec la clé de la requête et la fait suivre au node correspondant. Si la requête est enfin couronnée de succès, la réponse se fait avec les données par le node qui les renvoie vers son prédécesseur, en mettant le fichier dans son propre cache de données. Il crée aussi une nouvelle entrée dans sa table de routage associant la source de la donnée avec la clé de requête. Une requête suivante pour la même

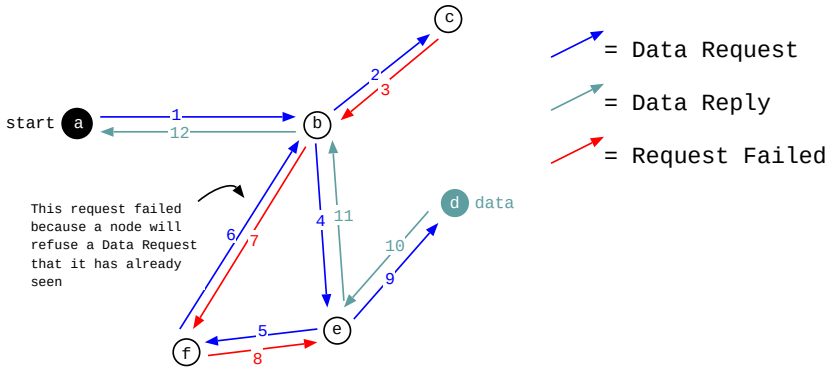


**Fig. 1 –.** *A typical request sequence.*

clé sera immédiatement satisfaite à partir du cache local; une requête pour une clé " similaire " (déterminée par la distance lexicographique) sera transmise à la précédente source de donnée positive. Le maintien de la table des sources de données est potentiellement un risque de sécurité, n'importe quel node sur le chemin peut décider unilatéralement de changer le message de réponse pour se déclarer lui-même, ou tout autre node arbitrairement choisi, comme la source de donnée.

Si un node ne peut pas transmettre une requête au node le plus approprié selon lui parce que cette cible est non-fonctionnelle ou qu'une boucle a été créée, le node ayant la seconde clé la plus proche sera tenté, puis le troisième le plus proche et ainsi de suite. Si un node n'a plus de candidats à essayer, il rendra compte de l'échec au node précédent qui tentera ensuite *son* second choix, etc. De cette manière, une requête opère comme une ascension de colline escarpée avec recherche en arrière d'un chemin possible. Si le compteur du hops-to-live est dépassé, un résultat d'échec est renvoyé en retour au node original sans qu'il y ait d'autres tentatives sur les nodes suivants. Les nodes peuvent unilatéralement raccourcir la valeur hops-to-live pour réduire la charge du réseau. Après un certain temps, ils peuvent aussi oublier les requêtes en cours afin de laisser la mémoire destinée aux messages libre.

La figure 1 décrit une séquence typique d'un message d'une requête.

L'utilisateur initie une requête au node *a*. Le node *a* fait suivre la requête au node *b*, qui le fait suivre au node *c*. Le node *c* ne peut pas contacter d'autres nodes et envoie un message en retour " échec de la requête " à *b*. Le node *b* tente alors son second choix, *e*, qui fait suivre la requête vers *f*. Le node *f* fait suivre la requête à *b*, qui détecte la boucle et renvoie par le même chemin un message d'erreur. Le node *f* ne peut contacter d'autres nodes et renvoie le message au node précédent *e*. Le node *e* fait suivre la requête à son second choix, *d*, qui possède les données. Cette donnée est renvoyée à partir de *d* via *e* et *b* jusqu'à *a*, qui le renvoie à l'utilisateur. La donnée est aussi dans le cache de *e*, *b* et *a*.

Ce mécanisme implique un certain nombre d'effets. Le plus important, est que la qualité du routage s'améliore avec le temps pour deux raisons. La première est que les nodes vont se spécialiser dans la localisation de jeu de clés similaires.

Si un node est listé dans la table de routage sous une clé particulière, il aura tendance à recevoir des requêtes pour des clé similaires. Il est donc susceptible d'acquérir plus " d'expérience " dans la réponse à ces requêtes et devient mieux informé par ses tables de routage sur les autres nodes possédant ces clés. la seconde est que les nodes vont devenir identiquement spécialisés en des clusters de fichiers ayant des clés similaires. Le fait de faire suivre une requête avec succès permettra au node d'acquérir une copie du fichier demandé et les requêtes seront pour des clé similaires. Le node aura donc des fichiers avec ces clés de même type. Pris ensemble, ces deux effets devraient améliorer l'efficacité des futures requêtes dans un cycle d'auto-renforcement, les nodes élaborant les tables et les stocks de données spécialisés dans un certain type de clés qui seront justement celles demandées.

En plus, le mécanisme de recherche dupliquera de manière transparente les données populaires dans le système et le miroir sera plus proche des demandeurs. Par exemple, si un fichier qui est originellement localisé à Londres est demandé à Berkeley, il sera dans le cache local et fournira des réponses plus rapides aux requêtes suivantes de Berkeley. Il est aussi copié sur chaque ordinateur le long du chemin, fournissant une redondance si le node de Londres tombe en panne ou s'il est arrêté. (Notez que " le long du chemin " est déterminé par la proximité de la clé et ne correspond pas nécessairement à une correspondance géographique).

Finalement, lorsque les nodes traitent des requêtes, ils créent des entrées dans la table de routage pour les nodes précédement inconnus qui fournissent des fichiers, augmentant la connectivité. Ceci aide les nouveaux nodes à découvrir plus du réseau (bien qu'il n'aide pas le reste du réseau à *les* découvrir; pour ceci, le mécanisme d'annonce, décrit dans la fsection 3.5 est nécessaire). Notez que des liens directs sont créés, court-circuitants les nodes intermédiaires utilisés. Alors, les nodes qui ont fourni des données avec succès gagneront des entrées dans la table et seront contactés plus souvent que les autres nodes.

Les clé sont dérivées du hachage et donc la proximité lexicographique n'implique pas la même chose pour la chaîne descriptive originale et il n'y a donc pas de similitude pour les sujets des fichiers correspondants. (Néanmoins, ce manque de proximité n'est pas important car l'algorithme de routage est basé sur le fait de savoir où les clés sont situées, non pas où les sujets sont situés. Ceci étant, supposons que `text/philosophy/sun-tzu/art-of-war` est haché en `AH5JK2`, les requêtes pour ce fichier peuvent être routées plus efficacement en créant des clusters contenant `AH5JK1`, `AH5JK2`, et `AH5JK3`, et non pas en créant des clusters de travaux sur la philosophie. Justement, l'utilisation du hachage est souhaitable précisement parce que les travaux de philosophie seront dispersés sur le réseau, diminuant l'impact de la défaillance d'un seul node qui aurait pu rendre toute la philosophie indisponible. La même chose est vraie pour les sous-espaces personnels –les fichiers appartenant au même sous espace seront éparpillés dans les différents nodes.

### 3.3 Stockage des données

L'insertion suit une stratégie parallèle aux requêtes. Pour insérer un fichier, une utilisatrice lui calcule d'abord une clé de fichier binaire, utilisant une des procédures décrites dans la section 3.1. Elle envoie le message à son propre node spécifiant la clé proposée et la valeur hops-to-live (ceci déterminera le nombre de nodes sur lesquels on doit la stocker). Lorsqu'un node recoit une proposition

d'insertion, il contrôle alors son propre stock de données pour voir s'il a déjà pris la clé. Si la clé est trouvée, le node retourne le fichier pré-existant comme si la requête avait été faite pour lui. L'utilisateur saura alors qu'une collision s'est produite et peut refaire une tentative en utilisant une clé différente. Si la clé n'est pas trouvée, le node cherche la clé la plus proche sur sa table de routage par rapport à celle proposée et fait suivre l'insertion au node correspondant. Si l'insertion crée une collision et retourne les données, le node reverra les données vers le node précédent et recommencera comme si la requête avait été faite (i.e. le fichier sera dans le cache local et il créera une entrée dans la table de routage pour la source de la donnée).

Si la valeur limite de hops-to-live est atteinte sans avoir détecté de collision de clé, un résultat " tout est clair " sera renvoyé vers le node original. Notez que pour les insertions le résultat est positif contrairement aux cas des requêtes. L'utilisateur insère alors les données qui vont se propager le long du chemin établi par la requête initiale et stockés dans chaque node du chemin. Chaque node créera alors une entrée dans sa table de routage associant le node inséreur (comme source des données) avec la nouvelle clé. Pour éviter les problèmes évidents de sécurité, tous les nodes le long du chemin peuvent unilatéralement décider de changer le message inséré pour se déclarer lui-même ou tout autre node comme la source de donnée.

Si un node ne peut pas transmettre une requête au node le plus approprié selon lui parceque cette cible est hors activité ou que cela créerai une boucle, il essayera le node ayant la deuxième meilleure clef la plus proche, puis la troisième et ainsi de suite de la même manière que pour les requêtes. Si tous les retours vers le node sont négatifs, cela indique que moins de nodes que ceux demandés peuvent être contactés. Comme pour les requêtes, les nodes peuvent réduire les valeurs hops-to-live et/ou oublier les insertions en cours après une certaine période.

Ce mécanisme a trois effets. Premièrement, les fichiers nouvellements insérés sont placés sélectivement sur des nodes possédant déjà des fichiers avec des clé similaires. Ceci renforce le clustering des clés établi par le mécanisme de requête. Deuxièmement, de nouveaux nodes peuvent indiquer au reste du réseau leur existence par l'insertion des données. Troisièmement, une tentative d'attaque en supplantant un fichier existant en faisant délibérément des collisions (e.g., en insérant un fichier corrompu ou vide sous la même clé) aura plus de chances de disséminer le vrai fichier plus avant car lors de collisions, il est propagé. (Note, ceci n'est vrai que pour les clés à signatures de mots car les autres types de clés sont plus facilement vérifiable.)

### 3.4 Gestion des données

Tout système de stockage d'informations doit gérer le problème de la capacité limitée de stockage. Les opérateurs de nodes individuels peuvent configurer le volume de stockage dédié à Freenet. Le stockage des nodes est géré comme un cache LRU[29] (Least Recently Used), dans lequel les données sont triées en ordre décroissant de temps des requêtes récentes (ou la date d'une insertion si la donnée n'a jamais été demandée). Lorsqu'un nouveau fichier arrive (soit pour une nouvelle insertion, soit pour une requête réussie) qui fera dépasser la taille du stockage, les fichiers les moins récemment utilisés sont évincés dans l'ordre de manière à générer de la place. L'impact résultant sur la disponibilité est par

là même mitigé par le fait que l'entrée de la table de routage créée va rester pendant un certain temps, permettant potentiellement au node d'avoir plus tard de nouvelles copies par la source de donnée originale. (Les entrées dans la table de routage sont aussi finalement effacées d'une même manière que le remplissage de la table, bien qu'ils soient gardés plus longtemps car prenant moins de place.)

Strictement, le stock des données n'est pas un cache, car c'est l'ensemble des stocks de données qui le forme. Ceci étant, il n'y a pas de copie " permanente " qui soit dupliquée dans un cache. Une fois que tous les nodes ont décidé, en parlant collectivement, de laisser tomber un fichier particulier, il ne sera plus disponible sur le réseau. De cette manière, Freenet diffère de services tels que Eternity et Free Haven, qui cherchent à fournir des garanties sur la survie des fichiers.

Ce mécanisme d'expiration a néanmoins un coté avantageux car il permet aux documents obsolètes d'être effacés une fois que leur mise à jour existe. Si un document obsolète est encore utilisé et considéré comme valable pour des raisons historiques, il restera stocké exactement aussi longtemps qu'il sera demandé.

Pour des raisons politiques ou légales, il peut être désirable, pour des opérateurs de node de ne pas savoir explicitement quel est le contenu de leur stock de données. C'est pourquoi il est donc recommandé que tous les fichiers insérés soient cryptés. Les procédures de cryptage utilisées ne sont pas destinées à sécuriser le fichier — c'est impossible car le requêteur (potentiellement quiconque) doit être capable de décrire le fichier une fois retrouvé. À la place, l'objectif est que l'opérateur de node puisse ne rien connaître du contenu de son stock de données car ce qu'il peut en connaître *à priori* est la clé de fichier, pas la clé de cryptage. Les clés de cryptage pour les données à mots clés signés et les sous-identifiants personnels peuvent être seulement obtenus en renversant un hachage et les clés de cryptage pour les données au contenu haché sont sans lien. Avec efforts bien sûr, une attaque par dictionnaire révèlera quelles clés sont présentes — comme cela doit être le cas pour que les requêtes puissent fonctionner –mais le le surcroît de travail qu'un tel effort demanderait est destiné à fournir une mesure de couverture pour les opérateurs de nodes.

### 3.5 Ajouter des nodes

Un nouveau node peut joindre le réseau en découvrant l'adresse d'un ou plusieurs nodes existant par des moyens extérieurs au réseau et ensuite démarrer l'envoi de messages. Comme mentionné précédement, le mécanisme de recherche permet naturellement aux nouveaux nodes d'apprendre plus du réseau avec le temps. Néanmoins, pour permettre aux nodes existants de *les* découvrir, les nouveaux nodes doivent d'abord annoncer leur présence. Ce processus est compliqué par deux exigences contradictoires. D'un coté, pour promouvoir un routage efficace, nous voudrions que tous les nodes existants soient cohérents dans la décision des clé à envoyer au nouveau node (i.e. quelle clé assigner à leur table de routage). D'un autre coté, il y aura un problème de sécurité si les nodes peuvent choisir leurs clés de routage, ce qui empêche les chemins les plus directs de réalisation de la cohérence.

Nous utilisons un protocole crypté pour satisfaire ces deux exigences. Un nouveau node qui rejoint le réseau choisit des (((((((seed))))))) aléatoires et envoie un message d'annonce contenant cette adresse et le hachage de ces

(((((((seed))))))))) à quelques nodes existants. Lorsqu'un node reçoit une annonce d'un nouveau node, il génère un ((((((((((seed)))))))))) aléatoire, effectue un XOR avec le hachage reçu et re-hache le résultat pour créer un engagement. Il fait alors suivre le nouveau hachage vers quelques nodes choisis aléatoirement dans sa table de routage. Ce processus continue jusqu'à ce que les hops-to-live de l'annonce soient épuisés. Le dernier node recevant l'annonce ne fait que générer un ((((((seed)))))). Maintenant, tous les nodes de la chaîne révèlent leur (((((((seed))))))) et la clé pour le nouveau node est assignée avec le XOR de tous les (((((((seeds))))))). Contrôler l'engagement permet à chaque node de confirmer que tout le monde a réellement révélé ses ((((((((((((seeds)))))))))))). Ceci produit une clé aléatoire consistante qui ne peut pas être influencée par un participant malfaisant. Chaque node ajoute alors une entrée pour le nouveau node dans sa table de routage sous cette clé.

## 4 Détails du protocole

Le protocole Freenet est orienté par paquets et utilise les messages autocontenant. Chaque message inclue un ID de transaction pour que les nodes puissent chercher l'état des insertions et des requêtes. Cette conception est prévue pour permettre la flexibilité dans le choix des mécanismes de transports pour les messages, qu'ils soient TCP, UDP, ou d'autres technologies comme la radio par paquets. Pour l'efficacité, les nodes utilisant un canal persistant comme une connexion TCP peuvent aussi envoyer de multiples messages sur la même connexion. L'adresse des nodes consiste en une méthode de transport plus un identificateur spécifique au transport (comme une adresse IP et un numéro de port), e.g. `tcp/192.168.1.1:19114`. Les nodes qui changent d'adresse fréquement peuvent aussi utiliser des adresses virtuelles stockées sous *address-resolution keys* (ARK's), qui sont des clés à sous-identifiants personnels mises à jour pour contenir les adresses courantes réelles.

Une transaction Freenet démarre avec un message Request.Handshake à partir d'un node avec un autre, spécifiant l'adresse de retour désirée du node émetteur[1]. (L'adresse de retour de l'émetteur peut être impossible à déterminer automatiquement par la seule couche de transport ou l'émetteur peut souhaiter utiliser un système de transport différent de celui utilisé pour envoyer le message.) Si le node éloigné est actif et répond aux requêtes, il renverra un Reply.Handshake spécifiant le numéro de version de protocole qu'il comprend. Les reconnaissances mutuelles sont mémorisées pour quelques heures et les transactions suivantes entre les mêmes nodes pendant ce temps peuvent omettre cette étape.

Tous les messages contiennent un ID de transaction de 64 bits généré aléatoirement, une limite de hops-to-live et un compteur de profondeur (Depth). Bien que l'ID ne puisse pas être garanti comme unique, la possibilité d'une collision se produisant pendant la durée de vie de la transaction parmi un nombre limité de nodes traversés est extrêmement faible. Hops-to-live est fixé par l'initiateur du message et est décrémenté à chaque saut pour éviter qu'il ne soit transmis à l'infini. Pour réduire l'information que la partie adverse pourrait obtenir à partir de la valeur hops-to-live, les messages ne se terminent pas automatiquement après que le hops-to-live ait atteint 1 mais poursuivent leur chemin avec une probabilité finie (avec un hops-to-live encore à 1). Depth est incrémenté à chaque saut et est

1. Rappelez vous que le node émetteur peut ne pas être le requêteur original.

utilisé par un node faisant une réponse pour initialiser hops-to-live à une valeur suffisamment haute pour atteindre le demandeur. Les requêteurs doivent l'initialiser à une petite valeur aléatoire pour rendre leur localisation floue. Comme pour hops-to-live, une profondeur de 1 n'est pas automatiquement décrémentée mais est renvoyée avec une probabilité finie.

Pour demander des données, le node émetteur envoie un message Request.Data spécifiant l'ID de la transaction, le hops-to-live initial et une clé de recherche. Le node récepteur contrôlera son stock de données pour la clé et s'il ne la trouve pas, fera suivre la demande à un autre node comme décrit dans la section 3.2. En utilisant le compteur hops-to-live, le node émetteur démarre un compteur pour le temps supposé qu'il prendra pour contacter les autres nodes, après quoi il supposera un échec. Alors que la requête est traitée, le node éloigné peut renvoyer périodiquement un message Reply.Restart indiquant que les messages sont bloqués sur les timeouts du réseau, afin que les nodes émetteurs puissent savoir qu'il faut étendre le timer.

Si la requête est finalement menée à bien, le node éloigné répondra avec un message Send.Data contenant la donnée demandée et l'adresse du node qui l'a fournie (avec une possibilité de trucage). Si la requête est finalement un échec et le hops to live épuisé sans réponse, le node éloigné répondra avec un Reply.NotFound. Le node émetteur décrémentera alors le hops-to-live du Send.Data (ou du Reply.NotFound) et le renverra sur le chemin de retour, à moins qu'il soit le destinataire de la requête. Ces deux messages terminent la transaction et libèrent les ressources occupées. Néanmoins, s'il reste des hops-to-live, habituellement parce la requête s'est retrouvée dans une impasse où il n'y avait pas de chemins non-bouclés viables, le node éloigné répondra avec un Request.Continue, en donnant le nombre de hops-to-live restant. Le node émetteur tentera alors de contacter le node suivant le plus proche dans sa table de routage. Il renverra aussi un Reply.Restart.

Pour insérer des données, le node émetteur envoie un message Request.Insert spécifiant un ID de transaction généré aléatoirement, un hops-to-live initial, un depth, et une proposition de clé. Le node éloigné contrôlera son stock de données pour la clé et s'il ne la trouve pas, fera suivre l'insertion vers un autre node comme il est décrit dans la section 3.3. Les messages Timers et Reply.Restart sont aussi utilisés de la même manière que pour les requêtes.

Si l'insertion, en fin de compte, donne une collision de clés, le node éloigné répondra soit par un message Send.Data contenant la donnée existante, soit par un Reply.NotFound (si la donnée n'a pas été réellement trouvée mais que la table de donnée la référence). Si l'insertion ne rencontre pas de collision et continue de circuler sur les nodes avec une valeur résiduelle de hops-to-live supérieure à zéro, le node éloigné répondra avec un Request.Continue. Dans ce cas, Request.Continue est un résultat d'échec indiquant que moins de nodes que ce qui était demandé ont été contactés. Ces messages seront passés dans le flux montant comme pour le cas des requêtes. Les deux messages terminent la transaction et libèrent toutes les ressources utilisées. Néanmoins, si l'insertion se termine sans avoir rencontré de collision, le node éloigné répondra avec un Reply.Insert, indiquant que l'insertion peut continuer. Le node émetteur renverra sur le flux montant un Reply.Insert et attendra de son prédécesseur un Send.Insert contenant les données. Lorsqu'il reçoit la donnée, il la stockera localement et fera suivre le Send.Insert vers le flux descendant, concluant la transaction.

# 5 Analyse de performance

Des simulations ont été menées sur une des premières versions pour donner quelques indications sur les performances. Nous en donnons ici les résultats les plus importants; pour plus de détails, voyez[21].

## 5.1 Convergence du réseau

Pour tester l'adaptabilité du routage du réseau, nous avons créé un réseau de test de 1000 nodes. Chaque node avait un stock de données de 50 éléments et une table de routage de 250 adresses. Le stock de données était initialement vide et les tables de routage étaient initialisées pour se connecter au réseau avec une topologie en treillis d'anneaux dans lesquelles chaques nodes avait des entrées de routage pour ses deux voisins les plus proches dans sa table de routage pour chaque cotés. Les clés associées avec ces entrées de routage étaient conçues comme des hachages des adresses de nodes de destination. Utiliser le hachage a l'utile propriété que les clés résultantes sont à la fois aléatoires et cohérentes (ceci étant, toutes les références à un node donné utiliseront la même clé).

Les insertions de clés aléatoires ont été envoyées à des nodes aléatoirement dans le réseau, parsemés aléatoirement avec les requêtes pour des clés choisies aléatoirement, connues pour avoir été précédement insérée, en utilisant un hops-to-live de 20 pour chaque. Chaque 100 unités de saut, une capture du réseau a été prise et sa performance mesurée en utilisant un jeu de requêtes de sondes. Chaque sonde consiste en 300 requêtes aléatoires pour des clés précédement insérées, utilisant un hops-to-live 500. Nous enregistrons la distribution résultante de la *longueur du chemin de la requête*, le nombre de saut réellement faits avant de trouver la donnée. Si la requête n'a pas trouvé la donnée, la longueur du chemin a été prise à 500.

La figure 2 montre l'évolution des premiers, second, et troisièmes tiers de la longueur de chemin de la requête au cours du temps, la moyenne sur dix essais. Nous pouvons voir que la grande longueur initiale du chemin décroit rapidement avec le temps. Au début, peu de requêtes réussissent mais lorsque le réseau converge, la longueur médiane du chemin descend à six.

## 5.2 Augmentation de taille

Ensuite, nous avons examiné le comportement à l'augmentation de taille du réseau. En démarant avec un petit réseau de 20 nodes initialisés de la même manière que dans la section précédente, nous avons ajouté quelques nodes au cours du temps et nous avons mesuré les changements dans la longueur du chemin des requêtes.

Les requêtes et les insertions ont été simulés aléatoirement comme précédement. Pour toutes les cinq étapes de temps, un nouveau node a été créé et ajouté au réseau en lui faisant faire une simulation de message d'annonce avec un hops-to-live de 10 envoyé vers des nodes aléatoirement choisis. La clé assignée par cette annonce a été prise pour être le hachage de la nouvelle adresse du node. Notez que cette procédure n'implique pas nécessairement un taux linéaire de croisance du réseau mais plutôt une relation linéaire entre le taux de requêtes et le taux de croissance. Il semble que les deux taux soient proportionnels à la

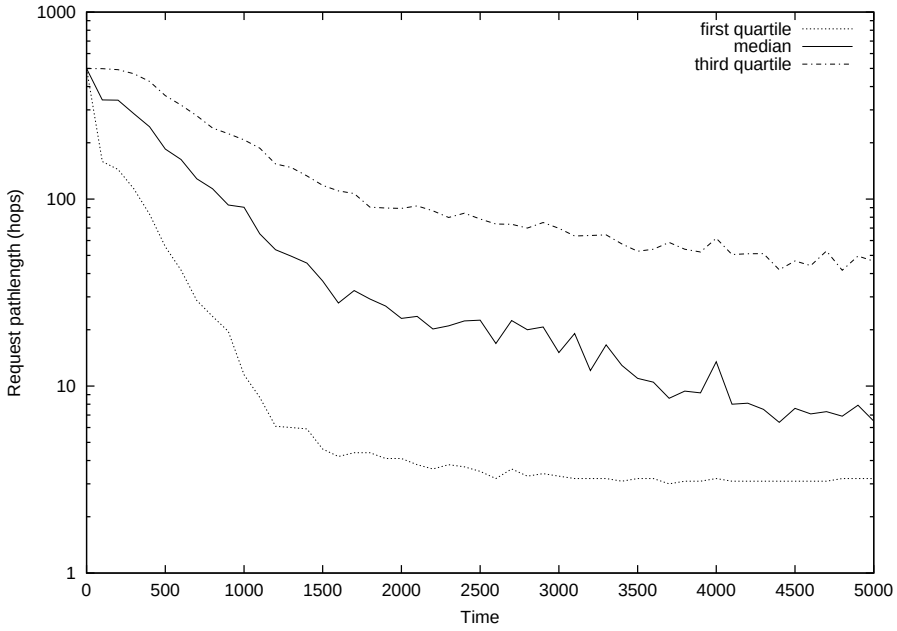


**Fig. 2 –.** *Evolution de la longueur du chemin de la requête au cours du temps.*

taille du réseau (produisant un taux de croissance exponentiel en réalité, opposé au temps de la simulation) et nous pensons que ce modèle est justifiable.

La figure 3 montre l'évolution des premiers, second, et troisièmes tiers de la longueur de chemin de la requête par rapport à la taille du réseau, moyenné sur dix tentatives. Nous pouvons voir que la taille du chemin est à peu près logarithmique, avec des changement de pente autour des 40,000 nodes. Nous supposons que le changement de pente est un résultat du remplissage des tables de routage et peut être amélioré en ajoutant un petit nombre de nodes avec des tables de routage un peu plus grandes. La section 5.4 parle de ce problème plus profondément. Là où les tables de routage étaient limitées à 250 entrées par les nécessités en mémoire de la simulation, les nodes réels de Freenet pourront avoir facilement des milliers d'entrées. Néanmoins, même ce réseau limité apparait capable de grossir à un million de nodes avec une longueur de chemin médiane de 30. Notez aussi que le réseau a grandi continuellement sans période de stailisation de convergence.

### 5.3 Tolérance à la panne

Finallement, nous considéront la tolérance à la panne du réseau. Démarrer avec un réseau monté à 1000 nodes par la méthode précédente, nous avons enlevé au réseau des nodes choisis aléatoirement au réseau pour simuler la défaillance des nodes. La figure 4 montre l'évolution résultante de la longueur du chemin, moyenné sur dix essais. Le réseau est étonnament résistant à des pannes plutôt importantes. La longueur médiane du chemin reste en-dessous de 20 même avec plus de 30% de pannes de nodes.

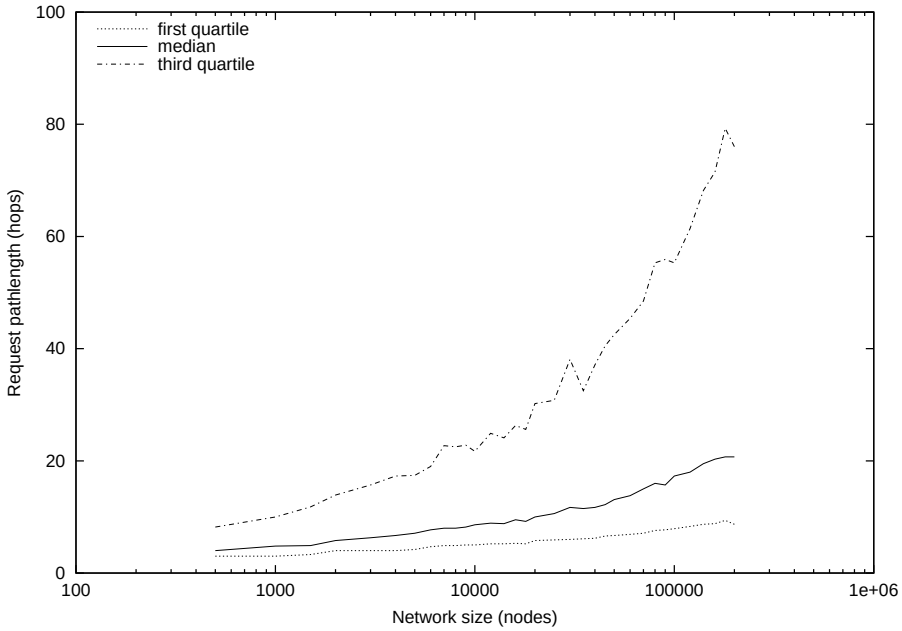


**FIG. 3 –.** *Longueur du chemin de la requête par rapport à la taille du réseau.*

### 5.4 Le modèle Small-world

Les caractéristiques de capacité de croissance et de résistance aux pannes de Freenet peuvent être expliquées en termes de modèle réseau *small-world* [23,31,22,3]. Dans un réseau small-world, la majorité des nodes ont relativement peu de connexions locales vers les autres nodes alors qu'un petit nombre de nodes ont de grands jeux de connexions. Les réseaux Small-world permettent des chemins courts entre des points arbitraires à cause des raccourcis fournis par les nodes biens connectés, comme mis en évidence par l'expérience de l'examen du passage de lettre par Milgram [23] et le jeu de nombre Erdös cité par Watts et Strogatz[31].

Freenet est-il un small world? Un facteur clé dans l'identification d'un réseau small-world est l'existence d'une distribution de liens à échelle libre de puissance dans un réseau, de même que la fin d'un telle distribution fournit aux nodes hautement connectés pour créer des chemins courts. La figure 5 montre la distribution moyenne des liens (i.e. les entrées de la table de routage) dans le réseau à 1000 nodes utilisé dans la section précédente. Nous voyons que la distribution approxime d'une manière très proche une loi de puissance, excepté pour les points anormaux représentants les nodes avec 250 entrées sur la table de routage. Lorsque nous avons utilisé des tables de routage de tailles différentes, ce point de rupture s'est déplacé mais le caractère de puissance de la distribution reste le même.

En plus de fournir des chemins courts, la distribution à loi de puissance donne aussi un réseau small-world avec un haut degré de tolérance. Les pannes aléatoires peuvent plus impliquer les nodes majoritaires possédant un faible nombre de connexions. La perte de nodes faiblement connectés ne va pas affecter le rou-

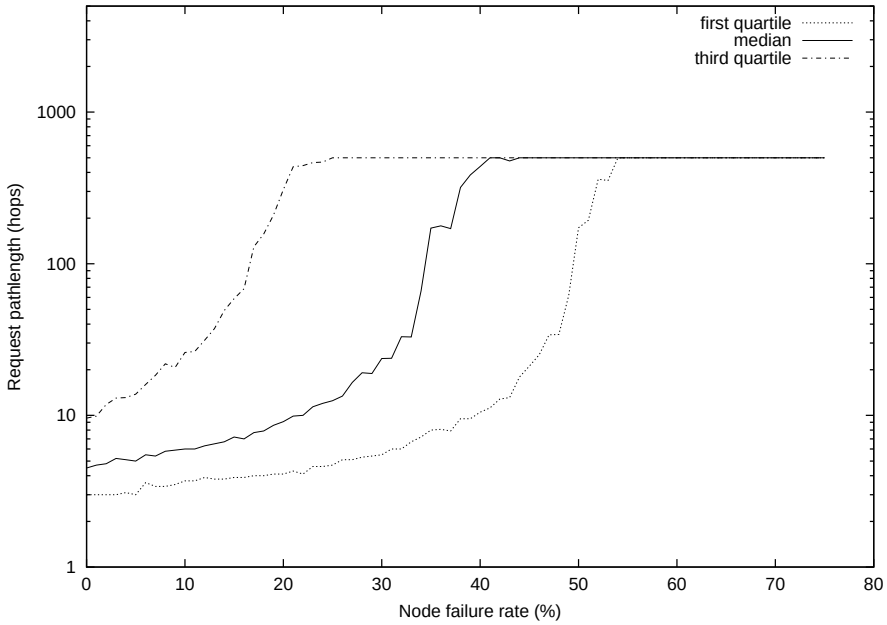


**Fig. 4 –.** *Changement dans la longueur du chemin avec des pannes réseaux.*

tage dans le réseau. C'est seulement lorsque le nombre de pannes devient suffisamment haut pour qu'il impacte un nombre significatif de nodes bien connectés que les performances du routage seront affectées d'une manière remarquable.

## 6 Sécurité

Le but primaire de la sécurité sous Freenet est de protéger l'anonymat des demandeurs et de ceux qui insèrent les fichiers. Il est aussi important de protéger l'identité des stockeurs de fichiers. Bien que, trivialement, n'importe qui puisse transformer un node en un stockeur en lui demandant un fichier, et donc " l'identifier " comme un stockeur, l'important c'est qu'il reste d'autres conteneurs du fichier, non-identifiés, de telle manière que la partie adverse ne puisse enlever le fichier en attaquant tous les nodes qui le détiennent. Les fichiers doivent être protégés contre les modifications malveillantes et finalement, le système doit être résistant aux attaques empéchant le service.

Reiter et Rubin[25] présentent une taxonomie utile des propriétés des communications anonymes selon trois axes. Le premier axe est celui du type d'anonymat : celui du receveur ou de l'émetteur, qui signifie respectivement que la partie adverse ne peut déterminer qui a envoyé ou qui a reçu. Le second axe est la partie adverse en question : une oreille indiscrète locale, un node malsain ou la collaboration de certain d'entre eux, ou un serveur de web (non appliquable à Freenet). Le troisième axe est le degré de l'anonymat, qui va de privé absolu (la présence de la communication ne peut pas être perçue) au soupçon (l'émetteur n'apparait pas plus qu'un autre comme étant à l'origine du message), innocence probable (l'émetteur est peu susceptible d'être l'auteur du message), innocence,

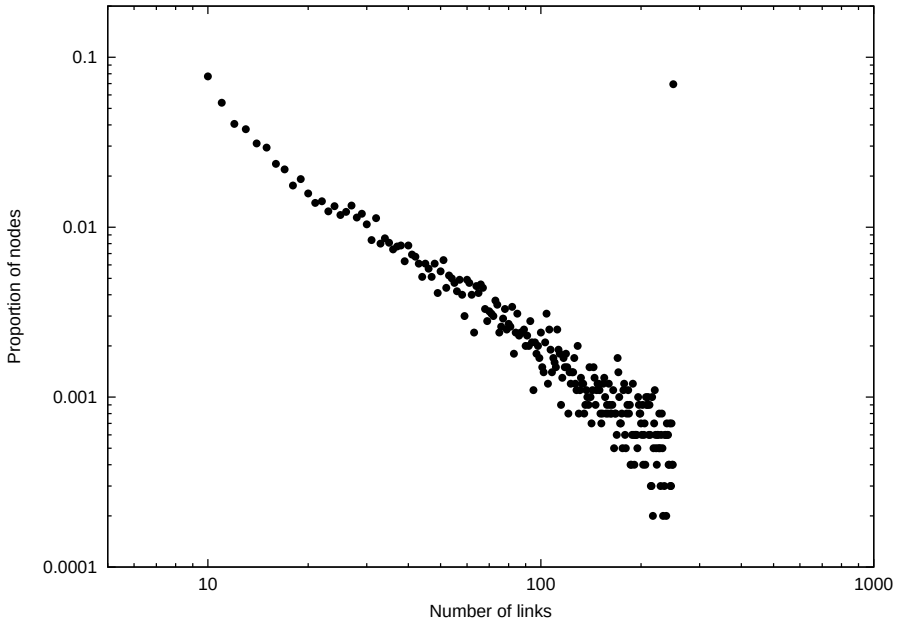


**Fig. 5 –.** *Distribution du nombre de liens parmi les nodes Freenet.*

| Système | Attaqueur | Anonymat de l'émetteur | Anonymat de la clé |
|---|---|---|---|
| Freenet de Base | oreille indiscrète locale | démasqué | démasqué |
| | nodes collaborateurs | plus que suspect | démasqué |
| Freenet+pré-routage | oreille indiscrète locale | démasqué | plus que suspect |
| | nodes collaborateurs | plus que suspect | démasqué |

**Tab. 1 –.** *Anonymat des propriétés de Freenet.*

démasqué et démasqué avec preuves (la partie adverse peut prouver aux autres qui était l'émetteur).

Comme la communication Freenet n'est pas dirigée vers des receveurs spécifiques, l'anonymat de ces derniers est vu de manière plus adaptée comme un anonymat de clés, qui consiste en la dissimulation de la clé requise ou insérée. Malheureusement, comme le routage dépend de la connaissance de la clé, l'anonymat de la clé n'est pas possible dans le schéma de base de Freenet (mais allez voir la discussion sur le " pré-routage " ci-dessous). L'utilisation du hachage comme clé fournit une mesure d'obscurité contre les oreilles indiscrètes fortuites mais reste bien sur vulnérable à une attaque de type dictionnaire puique, pour être utiles, les clés non hachées doivent être largement diffusées.

Les propriétés d'anonymat de Freenet sous cette taxonomie sont montrées dans la Table 1. contre une collaboration des nodes malsains, l'anonymat des émetteurs est préservé au dela de la suspiscion car un node dans un chemin de requête ne peut indiquer si son prédécesseur dans la chaîne a initialisé la requête ou a simplement fait suivre cette dernière. [25] décrit une attaque par probabilité qui pourrait compromettre l'anonymat de l'émetteur, en utilisant une analyse statistique de la probabilité qu'une requête arrivant à un node *a*

soit relancée ou traitée directement et la probabilité que $a$ choisisse un node $b$ particulier pour faire suivre. Cette analyse n'est pas néanmoins directement applicable à Freenet car les chemins de requête ne sont pas construits par des probabilités. Le fait de faire suivre dépend du fait ou non que $a$ possède la donnée de la requête dans son stock de données, plutôt que de la chance. Si une requête est suivie, la table de routage détermine où l'envoyer et il se peut que $a$ fasse suivre toutes les requêtes à $b$, ou ne jamais faire suivre de requête à $b$, ou n'importe quelle proportion de routage entre les deux. Toutefois, la valeur depth peut fournir quelques informations sur le nombre de sauts d'éloignement de l'origine, bien que cela soit brouillé par la valeur aléatoire de la valeur depth initiale et par les moyens de probabilité pour l'augmenter (voir section 4). Des considérations identiques s'appliquent à la valeur hops-to-live. Des investigations plus approfondies sont nécessaires pour éclaircir ces envois.

Contre une oreille indiscrète locale, il ne peut y avoir de protection des messages entre l'utilisateur et le premier node contacté. Puisque le premier node contacté peut agir comme une oreille indiscrète locale, il est recommandé à l'utilisatrice de n'utiliser que le node de sa propre machine comme point d'entée dans le réseau Freenet. Les messages entre les nodes sont cryptés contre l'écoute locale indiscrète, bien qu'une analyse de traffic puisse encore être faite (e.g. une écoute indiscrète peut observer un message sortant sans avoir de message entrant et conclure que la cible en est l'origine).

L'anonymat de la clé et celui plus fort de l'émetteur peut être réalisé en ajoutant un style mélangé de " pré routage " des messages. Dans ce schéma, les messages de base Freenet sont cryptés par une succession de clés publiques qui déterminent la route que le message crypté va suivre (en surpassant le mécanisme de routage normal). Les nodes le long de cette portion de route ne sont capables de déterminer ni l'origine du message, ni son contenu (incluant la clé de requête), comme pour les propriétés d'anonymat du réseau mélangé. Lorsqu'un message atteint le point final de la phase de pré routage, il sera injecté dans le réseau Freenet normal et se comportera comme si le point final était à l'origine du message.

La protection des sources de données est fournie par le reset occasionnel du champ source des données dans les réponses. Le fait qu'un node soit listé comme la source des données pour une clé particulière n'implique pas nécessairement qu'il fournisse réellement la donnée ou même qu'il ait été contacté dans le cas de la requête. Il n'est pas possible de dire si le node a fourni le fichier ou qu'il a simplement fait suivre la requête envoyée par quelqu'un d'autre. En fait, la seule action d'une requête de fichier réussie est de placer celui-ci sur le node qu'il n'y était pas déjà, pour qu'un examen approfondi ne puisse rien révéler de l'état du node avant la requête, et fournir une base légale possible à l'affirmation du fait que les données n'étaient pas là avant que la requête ne les place là. Faire une requête sur un fichier particulier avec un hops-to-live de 1 ne révèle pas directement si le node stockait précédement ou pas le fichier en question car le node continue à faire suivre les messages avec une probabilité de garder le hops-to-live à 1 avec une probabilité finie. Le succès de grands nombres de requêtes pour un fichier donné, néanmoins, peut fournir une assise sur la suspiscion du fait que le fichier était stocké ici précédement.

Les modifications des fichiers demandées par un node hostile dans une chaîne de requêtes sont un danger important et pas seulement à cause de la corruption

du fichier lui-même. Comme les tables de routage sont basées sur les réponses aux requêtes, un node peut tenter de capter le traffic pour lui-même en prétendant avoir des fichiers alors que cela n'est pas le cas et simplement en retournant des données fictives. Pour les données stockées sous clés hachées, cela n'est pas possible car les données non authentiques peuvent être détectées à moins qu'un node trouve une collision de hachage ou construise une signature cryptée valide. Les données stockées sous des clés à signature de texte, néanmoins, sont vulnérables aux attaques de type dictionnaires car les signatures peuvent être faites par quiconque connait la chaîne de description originale.

Finalement, un certain nombre d'attaques destinées à limiter le service peuvent être prévues. La menace la plus significative est qu'un attaquant tentera de remplir toute la capacité de stockage du réseau en insérant un grand nombre de fichiers pourris. Une possibilité intéressante pour contrer cette attaque est le schéma Hash Cash[20]. Essentiellement, le schéma demande à l'inséreur un long calcul comme " paiement " avant qu'une insertion ne soit acceptée et donc cela permet de ralentir une attaque. Une autre alternative est de diviser la zone de stockage en deux sections, une pour les nouvelles insertions et une pour les fichiers " établis " (définis comme les fichiers ayant reçu au moins un certain nombre de requêtes). Les nouvelles insertions ne peuvent que déplacer d'autres nouvelles insertions, pas les fichiers établis. De cette manière, un flot d'insertions de fichier pourris pourrait paralyser temporairement les opérations d'insertions mais ne déplacerait pas les fichiers existants. Il est difficile pour un attaquant de légitimer artificiellement ses fichiers (pourris) en les demandant plusieurs fois car ses requêtes seront satisfaites par le premier node à détenir les données et ne procédera pas plus avant. Il ne peut pas envoyer des requêtes directement aux autres nodes possédant ses fichiers car leurs identités lui sont cachées. Néanmoins, adopter ce schéma peut rendre difficile la survie des nouveaux inserts sains suffisament longtemps pour être demandés par les autres et devenir établis.

Les attaquant peuvent tenter de remplacer les fichiers existants en insérant des versions alternatives sous la même clé. De telles attaques ne sont pas possibles contre une clé hachée ou une clé à sous-identifiant personnel car cela demande de trouver une collision de hachage ou de créer avac succès une signature cryptée. Une attaque contre une clé de texte descriptif, d'un autre coté, peut résulter en deux versions existant sur le réseau. La manière dont les nodes réagissent à la collision lors de l'insertion (décrite dans la section 3.3) est destinée à rendre de telles attaques plus difficiles. Le succès d'une attaque par remplacement peut être mesuré par le ratio des corruptions sur les versions saines résultant dans le système. Néanmoins, plus l'attaquant tente de faire circuler des copies corrompues (en positionnant le hops-to-live à une grande valeur à l'insertion), plus la chance qu'une collision à l'insertion sera rencontrée, ce qui créera une augmentation des copies saines.

## 7 Conclusions

Le réseau Freenet fournit un moyen efficace de stocker et de retrouver des informations. En utilisant des nodes coopératifs répartis sur plusieurs ordinateurs en conjonction avec un algorithme de routage adaptatif efficace, il garde les informations anonymes et disponibles toute en restant hautement capable de s'adapter aux changements d'échelle. Un déploiement d'une version de test est en

cours et est déjà un succès avec quelques dizaines de milliers de copies téléchargées et plusieurs fichiers déjà en circulation. À cause de la nature du système, il est impossible de dire combien il y a d'utilisateurs ou avec quelle efficacité le système d'insertion et de requêtes fonctionne mais les preuves anecdotiques sont de loin positives. Nous sommes en train de travailler sur l'implémentation d'une suite de visualisations et de simulations qui permettra des tests plus rigoureux du protocole et des algorithmes de routage. Une simulation plus réaliste est nécessaire pour modéliser les effets des départs et des arrivées des nodes simultannés, des variations dans les capacités et la bande passante des nodes et des réseaux de plus grande taille. Nous voudrions aussi implémenter une infrastructure de clé publique pour authentifier les nodes et créer un mécanisme de recherche.

## 8 remerciements

Une partie de ce document est basée sur le travail supporté par le National Science Foundation Graduate Research Fellowship.

## Références

1. S. Adler, "The Slashdot effect: an analysis of three Internet publications," *Linux Gazette* issue 38, March 1999.
2. Akamai, `http://www.akamai.com/` (2000).
3. R. Albert, H. Jeong, and A. Barabási, "Error and attack tolerance of complex networks," *Nature* **406**, 378-382 (2000).
4. American National Standards Institute, American National Standard X9.30-199X: *Public-Key Cryptography Using Irreversible Algorithms for the Financial Services Industry: Part 1: The Digital Signature Algorithm (DSA)*. American Bankers Association (1993). X9.30.2-1997: *Public Key Cryptography for the Financial Services Industry - Part 2: The Secure Hash Algorithm (SHA-1)* (1997).
5. R.J. Anderson, " The Eternity service ", dans *Proceedings of the 1st International Conference on the Theory and Applications of Cryptology (PRAGOCRYPT '96)*, Prague, république Tchèque (1996).
6. Anonymizer, `http://www.anonymizer.com/` (2000).
7. O. Berthold, H. Federrath, and S. Köpsell, "Web MIXes: a system for anonymous and unobservable Internet access," in *Proceedings of the Workshop on Design Issues in Anonymity and Unobservability*, Berkeley, CA, USA. Springer: New York (2001).
8. D.L. Chaum, " Untraceable electronic mail, return addresses, and digital pseudonyms ", *Communications of the ACM* **24**(2), 84-88 (1981).
9. Y. Chen, J. Edler, A. Goldberg, A. Gottlieb, S. Sobti, et P. Yianilos, " A prototype implementation of archival intermemory ", dans *Proceedings of the Fourth ACM Conference on Digital Libraries (DL '99)*, Berkeley, CA, USA. ACM Press: New York (1999).
10. B. Chor, O. Goldreich, E. Kushilevitz, et M. Sudan, " Private information retrieval ", *Journal of the ACM* **45**(6), 965-982 (1998).
11. Church of Spiritual Technology (Scientology) v. Dataweb *et al.*, Cause No. 96/1048, Court de justice de la Hague, Hollande (1999).
12. I. Clarke, " A distributed decentralised information storage and retrieval system ", rapport non publié, Division Informatique, Université d'Edinburgh (1999). Disponible sur `http://www.freenetproject.org/` (2000).

13. L. Cottrell, "Frequently asked questions about Mixmaster remailers," `http://www.obscura.com/~loki/remailer/mixmaster-faq.html` (2000).
14. R. Dingledine, M.J. Freedman, and D. Molnar, "The Free Haven project: distributed anonymous storage service," in *Proceedings of the Workshop on Design Issues in Anonymity and Unobservability*, Berkeley, CA, USA. Springer: New York (2001).
15. Distributed.net, `http://www.distributed.net/` (2000).
16. D.J. Ellard, J.M. Megquier, et L. Park, " The INDIA protocol ", `http://www.eecs.harvard.edu/~ellard/India-WWW/` (2000).
17. Gnutella, `http://gnutella.wego.com/` (2000).
18. I. Goldberg et D. Wagner, "TAZ servers and the rewebber network: enabling anonymous publishing on the world wide web," *First Monday* **3**(4) (1998).
19. D. Goldschlag, M. Reed, et P. Syverson, " Onion routing for anonymous and private Internet connections ", *Communications of the ACM* **42**(2), 39-41 (1999).
20. Hash Cash, `http://www.cypherspace.org/~adam/hashcash/` (2000).
21. T. Hong, "Performance," in *Peer-to-Peer*, ed. by A. Oram. O'Reilly: Sebastopol, CA, USA (2001).
22. B.A. Huberman et L.A. Adamic, "Internet: growth dynamics of the world-wide web," *Nature* **401**, 131 (1999).
23. S. Milgram, "The small world problem," *Psychology Today* **1**(1), 60-67 (1967).
24. Napster, `http://www.napster.com/` (2000).
25. M.K. Reiter et A.D. Rubin, " Anonymous web transactions with Crowds ", *Communications of the ACM* **42**(2), 32-38 (1999).
26. The Rewebber, `http://www.rewebber.de/` (2000).
27. M. Richtel et S. Robinson, " Several web sites are attacked on day after assault shut Yahoo ", *The New York Times*, 9 Février 2000.
28. J. Rosen, " The eroded self ", *The New York Times*, 30 Avril 2000.
29. A.S. Tanenbaum, *Modern Operating Systems*. Prentice-Hall: Upper Saddle River, NJ, USA (1992).
30. M. Waldman, A.D. Rubin, et L.F. Cranor, " Publius: A robust, tamper-evident, censorship-resistant and source-anonymous web publishing system ", sur *Proceedings of the Ninth USENIX Security Symposium*, Denver, CO, USA (2000).
31. D. Watts et S. Strogatz, "Collective dynamics of 'small-world' networks," *Nature* **393**, 440-442 (1998).
32. Zero-Knowledge Systems, `http://www.zks.net/` (2000).