BitTorrent
le chaînon manquant des protocoles peer to peer?

Gaëtan de Menten
gdemente@info.fundp.ac.be

Facultés Universitaires Notre-Dame de la Paix - Namur

17 juin 2004

Résumé

BitTorrent est un protocole *peer to peer* de transfert de fichiers. Nous analysons son efficacité et montrons qu'il répond bien à la demande actuelle de pouvoir distribuer de gros fichiers à un grand nombre de clients en imposant une charge modeste sur la source initiale. Nous nous interrogeons également sur sa position et ses perspectives d'avenir par rapport à ses concurrents.

1 Introduction

Le succès d'Internet n'est plus à prouver mais un problème pointe à l'horizon : la quantité de données à distribuer sur Internet est en perpétuelle croissance. Et la partie du problème la plus importante est sans doute la distribution de gros fichiers (souvent du contenu multi-média—video, musique, ...) à un grand nombre d'utilisateurs.

En fait, lorsqu'on y regarde de plus près, le problème ne vient pas de la quantité de données mais plutôt du mode de distribution. En effet, la méthode traditionnelle pour distribuer du contenu est de le mettre sur un serveur web. Les utilisateurs voulant télécharger un fichier à partir d'un de ces serveurs doivent se partager la bande passante de celui-ci. Plus le fichier est populaire, plus la charge est importante sur le serveur[1]. Et à partir d'un certain point, ce fichier devient de moins en moins accessible aux utilisateurs (le(s) serveur(s) étant submergé(s) sous les requêtes ou la bande passante maximale de la connexion de celui/ceux-ci étant atteinte). Le distributeur peut évidemment investir dans un/des serveur(s) plus puissant(s) et/ou dans une augmentation de sa bande passante, mais ces investissements sont coûteux et la limite d'accessibilité n'en serait repoussée qu'un peu plus loin.

D'un autre coté, les programmes de partage de fichiers de type *peer to peer* ont un succès énorme auprès des internautes. D'ou l'idée de résoudre le problème de la distribution de contenu via un protocole de type *peer to peer*. Le protocole BitTorrent [3, 5, 4] a été inventé par Bram Cohen exactement dans ce but :

[1] dans la pratique, il se peut que plusieurs machines se partagent la tâche, mais cela ne change rien à la problématique.

faciliter la distribution de gros fichiers populaires en diminuant la charge sur le serveur original.

Actuellement, BitTorrent jouit d'un succès croissant. En témoignent les nombreux sites ou organisations qui proposent des téléchargements via ce protocole : RedHat, Slackware, etree.org[2], legaltorrents.com, pour n'en citer que quelques uns. Évidemment, et ce malgré que le protocole n'ait pas été conçu pour cela, de nombreux sites utilisent BitTorrent pour distribuer du contenu illégal (films, séries TV, musique et autres).

Dans la suite de cet article, nous présenterons d'abord le protocole BitTorrent en détail. Ensuite, nous étudierons son efficacité et sa capacité à résister à une montée en charge, ainsi que l'influence sur ses performances de divers facteurs extérieurs au protocole et de certains mécanismes mis en oeuvre dans celui-ci . Notre étude s'appuiera principalement sur, d'une part, l'analyse des logs enregistrés lors des cinq premiers mois de la distribution de la version 9 de RedHat via BitTorrent [7], et d'autre part l'analyse d'un modèle théorique, basé sur la théorie des flux, et sa comparaison avec la réalité [8]. Nous continuerons par un bref tour d'horizon des protocoles concurrents à BitTorrent.

2 Le Protocole

2.1 Présentation

BitTorrent est un protocole *peer to peer* classique en ce sens que ses clients partagent une partie de leur bande passante ascendante pour faciliter la distribution des fichiers qu'ils téléchargent. Il possède également les caractéristiques intéressantes suivantes :

- Son objectif étant uniquement la distribution de contenu (le plus efficacement possible et en limitant la bande passante du fournisseur initial) et non pas la localisation de ce dernier, BitTorrent ne contient pas de fonctionnalité pour trouver un fichier, comme dans les logiciels *peer to peer* classiques.
- A la différence de beaucoup d'autres protocoles *peer to peer*, les clients ne font pas partie d'un réseau global (comprenant tous les utilisateurs du protocole), mais sont plutôt regroupés par fichier : chaque fichier distribué verra son propre groupe d'utilisateurs. Chacun de ces groupes sera organisé autour d'un composant central, le *tracker*, qui sert en quelque sorte d'annuaire dynamique de clients intéressés par la distribution d'un seul fichier.
- Il traite les gros fichiers comme un certain nombre de morceaux[3] et chaque morceau peut être téléchargé d'une source différente.
- Il nécessite la distribution de fichiers .torrent contenant de la métainformation.
- Il contient divers mécanismes (que nous verrons plus loin) pour assurer son efficacité, dont notamment un mécanisme qui récompense les clients qui envoient des données aux autres.

[2] un site communautaire qui distribue des enregistrements de concerts "live" (de groupes qui acceptent ces enregistrement) sous un format avec compression sans perte.

[3] en général de 256KB

D'un point de vue vocabulaire, nous parlerons à partir de maintenant de *client* pour désigner n'importe quel client du réseau, de *peer* pour désigner n'importe quel client distant connecté au client courant, de *seed* pour désigner un client du réseau qui ne fait qu'envoyer des données (et ne télécharge donc rien) et enfin de *leechers*[4] pour désigner les clients qui ne sont pas des *seeds*.

2.2 Scénario typique

Dans un scénario typique, quelqu'un veut distribuer un ou plusieurs fichiers. Pour ce faire, il devra d'abord mettre en place un *tracker* (ou s'arranger avec le propriétaire d'un *tracker* existant). Ensuite, il devra construire le fichier .torrent à l'aide d'un utilitaire ad-hoc. Cela étant fait, il devra rendre accessible le fichier .torrent. La manière la plus couramment utilisée pour ce faire est de mettre ce fichier .torrent sur un serveur web ; une simple page web peut alors servir de catalogue des fichiers disponibles. Enfin, le fournisseur de contenu devra évidemment rendre accessible le contenu du fichier à distribuer. Cela sera fait en démarrant un client BitTorrent sur une machine qui a déjà le fichier.

Les clients BitTorrent des utilisateurs ayant récupéré le fichier .torrent se connecteront alors au *tracker*. Ce dernier leur répondra par une liste d'autres clients BitTorrents intéressé par le même fichier. Et le transfert pourra commencer. A noter que pour faciliter la vie des utilisateurs, le simple fait de télécharger le fichier .torrent lance en général directement le client BitTorrent[5] et démarre le transfert.

Dans les points qui suivent nous allons présenter plus en détails les différents éléments du protocole.

2.3 Structure du fichier .torrent

Un fichier .torrent sert principalement à deux choses : décrire les différents attributs du fichier à distribuer et donner un moyen de contacter le *tracker* qui "supervise" la distribution de ce fichier.

A ce titre, chacun de ces fichiers .torrent devra au moins contenir :

- le nom et la taille du (des) fichier(s) à distribuer
- la taille des morceaux et leur nombre
- une chaîne de caractères contenant, pour chaque morceau, les 20 bytes de leur valeur de hachage (du type SHA1).
- l'URL du *tracker*

2.4 Le protocole entre le *tracker* et les clients

Ce protocole est basé sur le protocole HTTP. Il sert à communiquer entre les *peers* et le *tracker*. Les requêtes du client au *tracker* devront contenir, entre autres :

- des statistiques de transfert du client (nombre de bytes envoyés, reçus et nombre de bytes qui restent à recevoir)

[4] Ceci est le vocable "officiel" du protocole, et malgré ce que l'on pourrait croire, les leechers contribuent à l'upload dans le système.

[5] Pour ce faire, la plupart des clients BitTorrent se définissent comme le "preneur en charge" du type MIME *x-bittorrent* lors de leur installation.

- l'état actuel du client ("vient de commencer le transfert", "est en cours de transfert", "a fini le transfert", ou encore "va quitter")

Le *tracker* répondra à cette requête par un message comprenant d'une part le temps que le client devrait attendre entre deux requêtes au *tracker* et d'autre part une liste de *peers* (un identifiant, une ip et un port par *peer*), qui sont intéressés par le même fichier que le client. Le nombre de *peers* renvoyés est variable (mais par défaut il est égal à 50). Ces *peers* sont choisis au hasard parmi tous les clients intéressés par ce même fichier !

2.5 Le protocole inter-clients

Les points les plus intéressants en sont les suivants :

- C'est un protocole basé sur TCP.
- Chaque fois qu'un client complète un morceau (et qu'il a vérifié que sa valeur de hachage est correcte), il l'annonce à tous les *peers* auquel il est connecté. Ceci permet à chaque client de savoir à tout moment ce que les *peers* auquel il est connecté ont comme morceaux et donc ce qui les intéresse.
- Dans la spécification originale du protocole, chaque client demande aux autres clients les morceaux qui composent le fichier voulu dans un ordre aléatoire. Cette politique fonctionne bien, mais actuellement la plupart des clients utilisent une politique de *rarest*[6] *first*.
- Pour faciliter le transfert, chaque morceau est encore subdivisé en sous-morceaux. Les sous-morceaux d'un même morceau doivent tous être téléchargés dans l'ordre et à partir de la même source (le même client).
- Un client choisit à qui il envoie des données. Quand un client refuse d'envoyer des données à un autre client, on dit que le premier client "choke" ce dernier (et à l'inverse lorsqu'il accepte d'en envoyer, on dit que le récepteur est "unchoked").

2.5.1 Mécanisme d'encouragement au partage

BitTorrent est doté d'un mécanisme qui récompense les clients qui envoient des données. En effet, à tout moment, un client *upload* vers maximum quatre *peers* (on dit qu'ils sont "unchoked"). Ces quatre *peers* doivent évidemment être intéressés par le contenu du client, mais aussi ils doivent être les quatre de chez qui le client *download* le plus vite (principe de réciprocité).

A noter que, pour éviter un changement incessant, un client évalue (et change éventuellement) ses *peers* seulement toutes les dix secondes. D'autre part, une fois qu'un client devient une *seed*, il ne décide plus à quels *peers* il envoie des données sur base de leur vitesse d'*upload* (puisqu'elle est forcément nulle) mais bien sur leur vitesse de *download*.

2.5.2 "Optimistic Unchoking"

Pour avoir la possibilité d'explorer le réseau, il y a en permanence un *peer* qui est "unchoked" quel que soit son taux d'*upload*. Ce *peer* peut changer toutes

[6] parmi les *peers* que le client connaît, évidemment.

les 30 secondes. Lorsqu'une nouvelle connexion est établie avec un *peer*, celui-ci a trois fois plus de chance que les autres d'être sélectionné, ce qui lui offre une opportunité raisonnable d'obtenir un morceau complet.

2.5.3 Superseed

L'algorithme de *super-seeding* [6] a été conçu pour aider un fournisseur de contenu disposant d'une bande passante limitée à "envoyer" un fichier sur le réseau BitTorrent le plus rapidement possible. A noter que cet algorithme ne fait pas partie de la spécification originale du protocole et n'est pas non plus implémenté dans le client officiel. Il a été proposé ultérieurement par John Hoffman[7] et implémenté dans divers clients dont le sien[8].

Lorsqu'un client est en mode "super-seed", il n'agit pas comme une *seed* normale, en laissant les clients demander les morceaux au hasard, mais plutôt essaie d'envoyer tous les morceaux du fichier le plus rapidement possible. La *seed* accomplit ceci en faisant semblant d'être un client normal qui n'a aucun morceau. Elle va pouvoir choisir les morceaux qu'elle distribue en envoyant aux autres clients des messages du type "j'ai reçu tel morceau" pour faire croire à ces clients qu'elle ne dispose que de ce morceau-là. Et pour favoriser les clients qui redistribuent les morceaux qu'ils reçoivent, la *seed* n'envoie un deuxième morceau à un client particulier qu'après avoir vu chez un autre client le dernier morceau qu'elle a envoyé à ce client.

Cet algorithme s'avère être très efficace pour la *seed* initiale (entre 150 et 200% plus efficace—c'est à dire entre 150 et 200% moins de données à envoyer avant de voir apparaître une autre *seed*—que l'algorithme standard) mais ne devrait jamais être utilisé pour les autres *seed*, car il diminue l'efficacité du transfert dans le cas d'un système ayant déjà plusieurs *seeds*.

3 Efficacité

Comme le suggèrent l'étude faite à partir des logs de la distribution des ISO de la RedHat 9 [7], ainsi que l'étude sur un modèle théorique [8], BitTorrent est un protocole très efficace. Ces deux études prouvent également qu'il est capable de résister à une forte augmentation du nombre de peers en très peu de temps.

Une des métriques principales pour mesurer les performances du protocole est la vitesse moyenne de transfert (ou le temps moyen pour terminer le téléchargement, ces deux métriques étant directement liées). La vitesse moyenne sur les 5 mois de mesure fût de 500Kbits/s, ce qui est déjà une preuve indéniable de l'efficacité du protocole.

En ce qui concerne le temps moyen, l'étude théorique nous donne l'expression suivante :

$$T = max\{\frac{1}{c}, \frac{1}{\eta}(\frac{1}{\mu} - \frac{1}{\gamma})\}$$

où T est le temps moyen de téléchargement
c est la bande passante descendante normalisée[9] des peers[10]

[7] aussi connu sous le pseudonyme TheSHAD0W
[8] BitTornado, TheSHAD0W's experimental BitTorrent Client
[9] par la taille du fichier en bytes
[10] L'étude prend comme hypothèse que tous les peers ont la même bande passante.

η est l'efficacité du partage
μ est la bande passante ascendante normalisée des peers
γ est le taux de départ des *seeds*

Cette équation nous procure de nombreuses informations sur le comportement de BitTorrent :

- Le temps moyen de téléchargement est indépendant du taux d'arrivée de nouveaux *leechers*, ce qui est une des preuves que BitTorrent supporte bien la montée en charge.
- Le temps moyen de téléchargement est indépendant du taux de départ des *leechers*.
- Le temps moyen de téléchargement n'est pas toujours contraint par la bande passante ascendante des *peers*. En effet, si le taux de départ des *seeds* (γ) est inférieur à la bande passante ascendante normalisée des peers (μ), alors, c'est la bande passante descendante normalisée (c) qui détermine les performances du réseau, même si c est beaucoup plus grand que μ (ce qui est fréquent, par exemple pour les connexions du type ADSL).
- Au début, si c augmente, T diminue. Par contre, à partir d'un certain point, si on augmente encore c, T ne diminuera plus (c'est à dire que la bande passante descendante n'est plus le facteur limitatif). Une observation similaire peut être faite en ce qui concerne la bande passante ascendante.

On aimerait également connaître l'efficacité du partage, c'est à dire si les *leechers uploadent* autant que leur bande passante leur permet ou si, au contraire, leur bande passante ascendante n'est pas utilisée. On peut également tourner la question de la manière suivante : "un *leecher* de bonne volonté trouve-t-il toujours un autre *leecher* intéressé par les morceaux de fichier qu'il a déjà ?".

Le modèle théorique nous donne ici :

$$\eta \approx 1 - (\frac{logN}{N})^k$$

où η est l'efficacité du partage (compris entre 0 et 1), N est le nombre de morceaux du fichier, et k est le nombre de peers auxquels le client est connecté.

Pour un cas réaliste d'un fichier d'au moins 100MB, N sera de l'ordre de plusieurs centaines (par défaut, les morceaux font chacun 256KB) et donc η sera très proche de 1, même si k est petit (ce qui n'est, en général, pas le cas). On voit également qu'en théorie, quand k augmente, l'efficacité du partage augmente également très lentement. Dans la pratique, il faut tenir compte de l'*overhead* que représente chaque connexion : il y a un point au-delà duquel augmenter le nombre de connexions d'un client (k) n'améliorerait plus les performances et pourrait même les détériorer.

Les deux études mentionnées ci-dessus obtiennent des résultats assez forts différents en ce qui concerne le pourcentage de la quantité de données envoyées par les *seeds* comparée à celle transmise par les *leechers*. Une des études cite le chiffre 33%, l'autre 60%. Il faudrait donc faire une étude à plus grande échelle sur plusieurs *torrents*. Quoi qu'il en soit, ce pourcentage ne renseigne

pas précisement sur l'efficacité du protocole car il est principalement dû au rapport entre le nombre de *seeds* et le nombre de *leechers* dans le système.

Une dernière caractéristique notable est qu'il y a apparemment une période d'"échauffement" d'environ 100 secondes avant d'obtenir les premiers morceaux. Nous reviendrons plus tard sur l'importance de cette caractéristique qui pourrait paraître anodine à première vue.

3.1 Efficacité du mécanisme d'encouragement au partage

Ce point est consacré à l'étude de l'impact du mécanisme prévu dans BitTorrent pour encourager le partage de ressources sur la stratégie des utilisateurs.

[8] montre que, si les bandes passantes ascendantes des autres clients sont fixées, la vitesse maximale de téléchargement d'un client est une fonction croissante[11] par rapport à sa propre bande passante ascendante. [7] confirme ce résultat dans la pratique en remarquant que les taux de transferts ascendants et descendants sont positivement en corrélation, ce qui est une preuve que le mécanisme d'encouragement fonctionne.

Mais si on prend comme hypothèse qu'un utilisateur veut maximiser sa vitesse de téléchargement tout en minimisant la quantité de donnée qu'il envoie, quelle proportion de sa bande passante ascendante devrait-il assigner à BitTorrent ?

[8] nous dit que si, dans le réseau, tous les peers sont départageables dans des groupes ayant la même bande passante physique ascendante[12] et que le nombre de peers dans chaque groupe est $> n_u + 1$ ((n_u étant le nombre d'*uploads* concurrents d'un client), alors il existe un point d'équilibre de Nash[13] qui est égal à la bande passante physique de chaque utilisateur. De plus, quelles que soient les valeurs assignées au départ par les utilisateurs, si chacun d'eux ajuste sa bande passante ascendante de manière rationnelle, le système converge vers ce point d'équilibre.

3.2 Comportement des utilisateurs

Comme pour beaucoup d'autres programmes de *peer to peer*, le comportement des utilisateurs joue un rôle crucial dans le succès du protocole. A ce titre, la présence de *seeds* et leur taux de départ du système à une énorme influence sur les performances du système (et même sur sa survie) car elles sont responsables pour une grande proportion du volume "uploadé". Mais il est également très important que les peers "normaux" s'envoient des données l'un l'autre car même si le partage n'était pas très efficace, cela peut jouer un rôle important pour garder le système en vie.

Les utilisateurs de BitTorrent ont en général un comportement altruiste vis à vis du système : d'une part en uploadant, d'autre part en restant connectés (en

[11] mais pas strictement croissante !

[12] Ce qui est un bon modèle de la population actuelle des utilisateurs d'Internet, qui peut être classée en groupes d'utilisateurs utilisant le même type de connexion (modem "classique", ADSL, cable, etc).

[13] S'il y a un ensemble de stratégies possibles pour un "jeu" et qu'on a la propriété qu'aucun agent n'a intérêt à changer sa stratégie lorsque les autres agents gardent leur stratégie inchangée, alors l'ensemble des stratégies choisies par les agents constitue un équilibre de Nash.

moyenne six heures et demi ! [7]) après avoir terminé leur téléchargement. L'upload est encouragé par le protocole lui-même. Quant au fait de rester connecté, cela peut être expliqué par plusieurs facteurs de natures diverses dont des utilisateurs "sympathiques", ou le fait que les logiciels qui implémentent BitTorrent (du moins tout ceux que nous avons essayé) doivent être arrêtés volontairement après la fin du téléchargement, ce qui peut très bien arriver lorsque l'utilisateur n'est pas devant son ordinateur (par ex. pendant la nuit).

3.3 "Optimistic Unchoking" et "Free-riding"

Le *free-riding*[14] est-il possible/avantageux dans le cadre de BitTorrent ? Si l'"optimistic unchoking" n'existait pas, la vitesse qu'un *free-rider* pourrait atteindre serait quasi nulle dans un réseau comprenant uniquement des *leechers*. Mais l'"optimistic unchoking" est un mécanisme nécessaire et ceci rend possible le *free-riding*.

[8] nous montre que la vitesse maximale de téléchargement qu'un *free-rider* peut atteindre est environ égale à $\frac{1}{n_u+1}$ de la vitesse de téléchargement qu'il pourrait avoir s'il se comportait normalement (n_u étant le nombre d'*uploads* concurrents d'un client). Dans l'implémentation actuelle de BitTorrent (ou du moins dans le client officiel), n_u vaut 4 par défaut et donc un *free-rider* atteindrait seulement 20% de la vitesse maximale de téléchargement qu'il pourrait espérer s'il agissait normalement. Etant donné que dans leur exemple ils ont considéré qu'il n'y avait que des *leechers* et pas de *seed*, une situation réelle serait sans doute pire que cela (du point de vue du système). En effet, le free-rider pourrait en même temps télécharger à partir d'une ou plusieurs *seed(s)* et profiter de l'"optimistic unchoking" des autres *peers*.

On pourrait penser à augmenter n_u pour diminuer la vitesse maximale qu'un "free-rider" peut obtenir, mais il ne faut pas oublier qu'augmenter n_u a un coût : plus de connexions TCP devraient alors partager la même bande passante, ce qui pourrait mener à de mauvaises performances.

3.4 Firewall ?

Un facteur qui peut s'avérer déterminant dans l'efficacité du protocole pour un client particulier est le fait d'avoir le(s) port(s) sur le(s)quel travaille BitTorrent bloqué(s) en entrée par un *firewall* ou un NAT[15]. En effet, un tel client a deux désavantages : premièrement, il a moins de *peers* possible, car il ne peut pas se connecter à un autre client qui a, lui aussi, ses ports bloqués. Ensuite, personne ne sait le contacter, donc les connexions qu'il aura seront uniquement celles qu'il aura initiées. S'il y a suffisamment de clients "non-bloqués" pour ce "torrent", il finira par atteindre sa limite de connexions, mais cela prendra plus longtemps que pour un client "non-bloqué".

Étant donné que nous n'avons pas réussi à obtenir une estimation fiable de la proportion des machines dont les ports sont bloqués[16], il est difficile d'évaluer l'impact exact de ces blocages.

[14] Le *free-riding* est le fait qu'un client essaie de profiter du système en téléchargeant sans rien envoyer en retour.

[15] Ceci est bien entendu valable pour n'importe quel protocole *peer to peer*.

[16] des sources informelles et peu fiables annoncent des proportions qui varient entre 18% et plus de 50% !

3.5 Quelques critiques

La critique principale qu'on peut faire à BitTorrent est que l'utilisation d'un *tracker* unique est un obstacle à une montée en charge sans problème. Même si la bande passante utilisée par le *tracker* est limitée, il arrive un point ou c'est elle qui limite la croissance du système.

Une seconde faiblesse est que la vitesse du téléchargement est faible au début et prend un certain temps avant de "décoller". Ceci est principalement dû au fait que le téléchargement du premier morceau d'un fichier peut être assez long et pendant ce temps, le client ne peut (forcément) rien *uploader* (d'ou une vitesse de download faible).

On pourrait également critiquer le fait que le *tracker* ne fait pas grand-chose, à part servir d'annuaire pour les différents clients intéressés par un même fichier et collecter passivement des données. Il pourrait, par exemple, essayer de décider plus "intelligemment" quels clients connecter entre eux[17]. Le problème c'est que ce genre de décision devrait être basée sur les données que les clients envoient au *tracker*, et l'on verrait peut-être apparaître dans ce cas-là des clients modifiés pour "tricher". Une solution à ce problème serait que les clients reportent au *tracker* la quantité de données qu'ils ont reçue de chaque *peer*. Mais cette solution n'est peut-être pas faisable pour des raisons de bande passante ; celle utilisée par le *tracker* étant déjà le facteur le plus limitatif de la montée en charge, il vaut sans doute mieux ne pas empirer ce problème...

Et enfin, on peut critiquer le fait que le protocole est trop "statique" : de nombreux paramètres sont fixés alors qu'ils pourraient sans doute être adaptés automatiquement pour une meilleure efficacité. Nous pensons en particulier au nombre de *peers* auxquels se connecte chaque client, le nombre de clients "unchoked", les différents *timings* du protocole, etc...

4 Concurrence et avenir du protocole

Dans cette section, nous allons présenter divers protocoles concurrents à BitTorrent. Au vu de l'impossibilité de les présenter tous dans le cadre de cet article, nous nous limiterons aux quatre protocoles (dont nous avons connaissance) les plus proches de BitTorrent d'un point de vue objectifs.

Un premier protocole concurrent est Slurpie [10]. C'est un protocole en développement qui a les mêmes objectifs que BitTorrent et y est similaire pour certains aspects. Les différences les plus notables sont qu'il n'y a pas de *tracker* mais bien un *Topology Server* auquel tous les clients se connectent (quel que soit le fichier qui les intéressent), qu'un client peut faire varier le nombre de *peers* auquel il est connecté si le besoin en est, que la bande passante de la source est indépendante du nombre de clients dans le système, et enfin que Slurpie a besoin d'estimer le nombre de clients dans le système pour pouvoir fonctionner.

A en croire ses concepteurs, Slurpie est plus efficace que BitTorrent, ce qui serait prouvé par leurs expériences en laboratoire. Quoique Slurpie a l'air en effet extrêmement efficace et est *probablement* en effet plus efficace que BitTorrent nous avons certaines réserves à émettre à ce sujet. D'une part, leurs expériences en laboratoire n'ont pas été faites dans des conditions qui reflètent vraiment la réalité (relativement peu de *peers*, et bandes passantes de ces derniers irréalistes

[17] par exemple, en fonction des bandes passantes estimées de chaque client

par rapport à un usage "dans le monde réel"). D'autre part, même si leur *Topology Server* est un peu moins gourmand en bande passante qu'un *tracker* (qui est déjà relativement léger), le fait d'avoir un seul serveur nous paraît être un obstacle majeur à une montée en charge. Même s'ils abordent eux-mêmes le sujet et proposent de distribuer la fonction de leur *Topology Server* sur plusieurs machines par des techniques de *Distributed Hash Table*, la solution "un tracker par (groupe de) fichier(s)" utilisée dans BitTorrent nous paraît nettement plus simple et élégante.

Konspire2b [9] est un autre protocole concurrent. Il a une utilité légèrement différente (il utilise un modèle de *publish-subscribe*) par rapport à BitTorrent, mais qui pourrait être facilement adapté pour remplir la fonction de ce dernier (ou vice-versa). Un fait intéressant est que ses développeurs déclarent que Konspire2b est plus efficace que BitTorrent. Nous avons de sérieux doutes à ce sujet. En effet, ce protocole utilisant un modèle en arbre dont chaque noeud ne transfère qu'a un seul autre noeud en même temps, il n'y a donc toujours qu'une partie des clients intéressés qui participent à un transfert. Et seuls les clients qui ont le fichier en entier utilisent leur bande passante ascendante ! Nous pourrions passer notre temps à réfuter les arguments donnés par les développeurs de Konspire2b dans leur "preuve" que leur protocole est plus efficace, mais cela sortirait du cadre de cet article.

PDTP (Peer Distributed Transfer Protocol) [2, 1], quant à lui, est un protocole qui a l'air très prometteur mais qui est encore en plein développement (encore plusieurs mois avant d'avoir une première implémentation complète). La principale amélioration par rapport à BitTorrent se situe apparemment dans la multiplicité des composants déployables : un réseau PDTP peu contenir plusieurs serveurs (l'équivalent des *trackers* de BitTorrent), plusieurs *Piece Proxy* (qui aident la *Source*—équivalent de la *seed* initiale—à diffuser les données) et des *Proxy* (qui permettraient apparemment d'éliminer le problème des clients derrière un firewall).

Enfin, Swarmcast est un protocole qui, malgré sa similitude et son apparition quasi simultanée avec BitTorrent a un succès moindre. Son fonctionnement est quasi identique à BitTorrent, si ce n'est que le client Swarmcast arrête d'envoyer des données lorsqu'il possède le fichier en entier et que dans Swarmcast, de l'information de parité est ajoutée à chaque morceau avant d'être envoyé. Cette information de parité, qui sert à éviter d'obtenir des fichiers corrompus, est encodée de manière telle que chaque morceau fait 1.5 fois sa taille. D'un autre coté, il n'est nécessaire d'avoir tous les morceaux d'un fichier pour pouvoir le reconstituer.

Les raisons du succès moindre de Swarmcast par rapport à BitTorrent sont un mystère pour nous... Serait-ce son passage en *open-source* après BitTorrent ? Serait-ce sa vocation plus commerciale dès le départ ? Ou alors tout simplement une efficacité moindre ? N'ayant pu tester Swarmcast, nous ne pouvons malheureusement pas nous prononcer sur ce point.

Qu'adviendra-t-il de BitTorrent face à ces protocoles ? Plus précisément, ce protocole va-t-il survivre à l'arrivée de Slurpie et PDTP, qui sont, selon nous, les menaces les plus plausibles pour BitTorrent... Il est quasiment impossible de le savoir car cela dépend de pas mal de facteurs et pas uniquement de l'efficacité de chaque protocole.

Une chose est sûre, c'est que Bram Cohen et les autres développeurs de la communauté BitTorrent ne vont pas "se laisser faire". En effet, ils sont déjà

en train de concevoir une nouvelle mouture du protocole pour palier à certains problèmes du protocole actuel. Plus spécifiquement, cette nouvelle mouture résoudra (ou du moins améliorera), entre autres, le problème de l'acquisition du premier morceau grâce à une taille de morceaux plus petite. Pour que le fichier .torrent ne soit pas trop gros, ils comptent utiliser un arbre de Merkle à la place des valeurs de hachage de type SHA1 utilisées actuellement. Le changement de la taille des blocs devrait avoir comme conséquence de diminuer la "période de pré-chauffage" (et donc d'obtenir des taux de transferts élevés plus rapidement).

5 Conclusion

Nous avons présenté le protocole BitTorrent et montré qu'il remplit parfaitement sa mission en étant (très) efficace dans son domaine. Cependant il reste des possibilités d'amélioration. C'est pourquoi nous pensons qu'à long terme, il va probablement laisser la place à d'autres protocoles. Malgré cela, on peut d'ores et déjà dire qu'il aura marqué une étape importante dans le monde des protocoles *peer to peer*.

Références

[1] T. Arcieri. *PDTP Network Illustrations*. http ://pdtp.org/networks.php.

[2] T. Arcieri. *PDTP Protocol Specification*. http ://pdtp.org/protocol.php.

[3] B. Cohen. *BitTorrent protocol specification*. http ://bitconjurer.org/BitTorrent/protocol.html.

[4] B. Cohen. *Incentives Build Robustness in BitTorrent*. http ://bitconjurer.org/BitTorrent/bittorrentecon.pdf, May 2003.

[5] B. Cohen and the BitTorrent community. *BitTorrent protocol specification*. http ://wiki.theory.org/index.php/BitTorrentSpecification.

[6] J. Hoffman. *About Super-Seed Mode*. http ://bittornado.com/docs/superseed.txt.

[7] M. Izal, G. Urvoy-Keller, E.W. Biersack, P.A. Felber, A. Al Hamra, and L. Garcés-Erice. Dissecting bittorrent : Five months in a torrent's lifetime. In *Proceedings of Passive and Active Measurements 2004 (PAM2004)*, April 2004.

[8] Dongyu Qiu and R. Srikant. Modeling and performance analysis of bittorrent-like peer-to-peer networks. In *Proceedings of ACM SIGCOMM 2004*, August 2004.

[9] J. Rohrer. *How konspire2b works*. http ://konspire.sourceforge.net/howItWorks.shtml.

[10] R. Sherwood, R. Braud, and B. Bhattacharjee. Slurpie : A cooperative bulk data transfer protocol. In *Proceedings of IEEE INFOCOM*, March 2004.