Protocole BitTorrent
Master2 Informatique, 2005-2006

Brami Franck & Delattre Arnaud

Table des matières

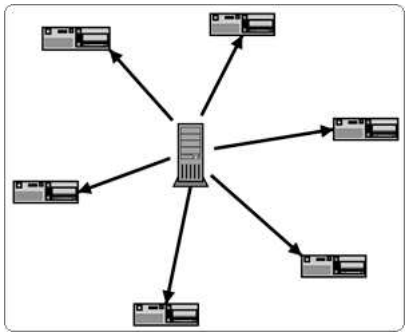
1 Introduction

BitTorrent est un logiciel pair à pair créé par Bram Cohen en 2001. L'idée lui est venue après avoir travailler au sein de la société MojoNation sur un logiciel qui consistait à découper les fichiers confidentiels en morceaux chiffrés et à distribuer ces morceaux à des ordinateurs sur lesquels tournait ce logiciel. Il pensa alors à appliquer le découpage de fichiers pour permettre leur échange. Son objectif principal était de permettre la réplication rapide de gros fichiers et la scalabilité du système.

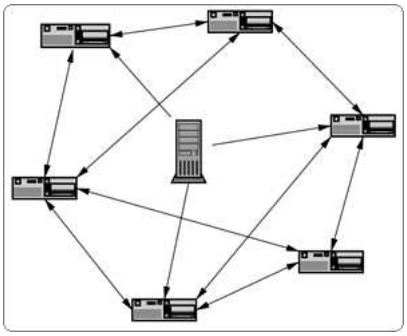
1.1 Idée générale

Lors d'un téléchargement de type clients/serveur, plus les clients sont nombreux, plus la demande est forte côté serveur, moins vite le fichier se propage car le serveur arrive à saturation (soit par le nombre de requêtes, soit par la limitation due à la bande passante).

Ce problème n'est pas dû à la quantité des données, mais au mode de distribution. Avec un téléchargement décentralisé de type BitTorrent, plus le fichier est demandé, et plus il est disponible car tous les clients se partagent les morceaux du fichier qu'ils ont reçu.



Un échange centralisé



Un échange distribué

1.2 Principe de l'échange

La personne désireuse d'échanger un fichier commence par mettre en place un *tracker* (ou s'adresse à un *tracker* existant). Ensuite elle génère un fichier de métadonnées qui contient notamment l'adresse du *tracker*, et met ce fichier à disposition des *leechers*.

Les *leechers* sont ceux qui ne possèdent pas encore le fichier et qui souhaitent l'acquérir. Au contraire, on appellera *seeds*, les personnes possèdant une copie complète du fichier, et prêtes à le partager.

Les *seeds* et les *leechers* se connectent au *tracker* de la même façon. Le *tracker* constitue une sorte d'annuaire chargé de mettre en contact les différents pairs (*seed* ou *leecher*) qui forment les noeuds du réseau. À la différence des autres réseaux p2p, ce réseau n'est pas global, il en existe un pour chaque fichier échangé.

Le fichier à échanger est découpé en pièces et en sous-pièces. Une fois en contact, les *seeds* vont uploader des parties du fichier aux *leechers*. Dès qu'un

leecher arrive à reconstituer une pièce complète, il peut commencer à l'uploader et donc à contribuer à la réplication du fichier sur le réseau.

2 Le fichier de métadonnées

2.1 La diffusion

Contrairement aux autres protocoles de *peer-to-peer*, BitTorrent ne dispose pas d'outil de recherche intégré. Le point de départ d'un téléchargement consiste à récupérer un fichier `.torrent` contenant des méta-informations qui permettront de se connecter au *tracker*. Le plus souvent, ces fichiers sont mis à disposition sur des pages internet et listés chronologiquement. Il y a bien sûr la possibilité d'effectuer des recherches dans ces listes[1]. Mais ce mode de mise à disposition entraine une philosophie d'utilisation différente des logiciels p2p classiques :

- L'utilisateur a tendance à consulter les derniers *torrent* mis à disposition et à télécharger ceux qui l'intéressent. Contrairement aux p2p classiques où l'utilisateur effectue plutôt des recherches ciblées de ce qu'il recherche.
- Les "nouveautés" sont mises en avant.

La diversité dans le choix des fichiers à télécharger est donc plus réduite. Le partage est efficace, mais la durée de vie d'un `torrent` est faible par rapport à la durée de vie sur un réseau p2p classique.

2.2 Encodage utilisé

Le fichier `.torrent` et les réponses du *tracker* (c.f section 3.2) utilisent le format *Bencoding*, largement inspiré de la syntaxe du langage Python (langage utilisé pour le développement du client BitTorrent officiel). L'encodage se fait de la façon suivante :

- Les chaînes de caractères sont de la forme len`:`str où str désigne la chaîne et len sa longueur. Par exemple la chaine "www" sera codée par `3:www`
- Les entiers sont de la forme : `i`int`e` où int désigne la représentation de l'entier en base 10. Par exemple l'entier -3 sera codé par `i-3e`. De ce fait, la taille des entiers n'est pas limitée. Par convention `i-0e` n'est pas valide, ainsi que tous les entiers avec des zéros inutiles en préfixe.
- Les listes sont de la forme : `l`$elem_1 \ldots elem_n$`e` où $elem_i$ représente la forme *bencoded* de l'élément d'indice i. Par exemple `l4:spam4:eggse` correspond à la liste `['spam', 'eggs']`.
- Les dictionnaires (appelation des listes associatives en Python) sont de la forme `d`$key_1 value_1 \ldots key_n value_n$`e`.
  Par exemple `d3:cow3:moo4:spam4:eggse` correspond à `{'cow' : 'moo', 'spam' : 'eggs'}`. Et `d4:spaml1:a1:bee` correspond à `{'spam' : ['a', 'b']}`.
  À noter que les clés doivent être des chaînes de caractères et qu'elles doivent être classées.

Ce format a pour avantage d'être indépendant de l'*endianness* (l'ordre des octets).

[1] Par exemple sur `http ://www.torrentule.com/` ou en utilisant Google avec les attributs de recherche filetype :torrent inurl : mot_cle

2.3 Contenu du fichier

Voici les informations contenues dans un fichier `.torrent` :

```
{announce   =>  url ,
 info       =>  {name           =>  nom ,
                 piece length   =>  taille ,
                 piece          =>  multiple ,
                 length         =>  taille ,
                 files          =>  ({length => length_1, path => path_1 },
                                     ...
                                     {length => length_n, path => path_n })
                                 }
}
```

BitTorrent autorise l'échange de plusieurs fichiers à la fois (et même sous forme arborescente). On parlera de "multi-fichier". En pratique, les différents fichiers sont concaténés les uns aux autres dans l'ordre dans lequel ils apparaissent dans la liste.
url désigne l'URL du *tracker*.
name correspond à un nom suggéré pour le fichier ou alors au nom d'un répertoire dans le cas multi-fichier.
taille correspond à la taille d'une pièce. Toutes les pièces sont de la même taille (excepté la dernière). Le plus souvent, il s'agit d'un multiple de 2, la valeur la plus commune étant $2^8 = 256K$.
La clé `piece` associe une chaine de caractères d'une taille multiple de 20. Il s'agit de la concaténation des empreintes SHA1 de toutes les pièces dans l'ordre de leur indice.

Dans le cas multi-fichier, la clé `length` est absente. La clé `files` est associée à une liste de dictionnaires (un dictionnaire par fichier, indiquant la taille du fichier et son nom sous forme relative).
Dans le cas "mono-fichier", pas de clé `files`, et la clé `length` indique la longueur du fichier, en octets.

3 Connexion au *tracker*

À partir du fichier `.torrent`, le client connaît l'adresse du *tracker*. Le *tracker* n'est pas impliqué dans la réelle distribution du fichier, il garde les méta-informations concernant les pairs et joue le rôle de point de rendez-vous pour tous les clients du torrent.
Le *tracker* est un serveur HTTP (cela pourrait par exemple être un serveur *Apache* utilisant un module particulier).

3.1 La requête

Tout d'abord, le client envoie une requête `GET` au *tracker*.
Parmi les paramètres de cette requête on trouve :

- info_hash : Une empreinte de 20 octets. Elle correspond à l'entrée "info" du dictionnaire du fichier `.torrent` décrit précédement.
- peer_id : Identifiant unique du client, d'une taille de 20 octets,

- port : Le port sur lequel le client écoute. Par défaut, BitTorrent réserve la plage 6881-6889.
- uploaded : Le nombre total d'octets envoyés par le client.
- downloaded : Le nombre total d'octets téléchargés par le client.
- left : Le nombre d'octets restants à télécharger par le client.

Certaines valeurs sont présentes à titre uniquement statistique, le fait de mentir sur le nombre total d'octets envoyés ne procure aucun bénéfice. À ces paramètres s'ajoutent des paramètres optionnels :

- event : Indique l'état du *tracker* sur le client. Les différentes valeurs sont :
  - `started` : Il s'agit de la première requête au *tracker*. Le *tracker* doit donc créer une nouvelle session pour ce client.
  - `stopped` : Indique au *tracker* que le client souhaite arrêter sa session.
  - `completed` : A envoyer lorsque le client a terminé son téléchargement, ce qui permet au *tracker* de mettre à jour le nombre de *seed*.
- ip : L'adresse IP du client.
- numwant : Nombre de pairs que l'on souhaite récupérer.

3.2 La réponse

Le *tracker* renvoie alors une réponse sous la forme de dictionnaire *bencoded*. Dans le cas d'une erreur :

`{failure_reason =>` *str* `}`

où *str* est une chaîne expliquant pourquoi la requête a failli. Dans les autres cas, le résultat est de la forme :

`{interval =>` *interval* `,`
`peers =>` `({id =>` *id* `, ip =>` *ip* `, port =>` *port* `},`
...
`{id =>` *id* `, ip =>` *ip* `, port =>` *port* `})}`

Il s'agit d'un dictionnaire où *interval* correspond à la fréquence à laquelle le client doit recontacter le *tracker* (le plus souvent toutes les 30 minutes) afin d'obtenir de nouveaux pairs. La clé *peers* renvoie vers une liste de dictionnaires. Chaque élément de la liste représente un pair, avec son identifiant, son adresse et le port sur lequel il écoute.

Remarque : D'autres "clés" (non officielles) peuvent être éventuellement utilisées.

4 Échanges entre les pairs

4.1 La poignée de mains (*handshaking*)

La connexion entre deux pairs commence par une poignée de mains. Voici les différentes informations contenues dans cette poignée de mains :

- *Protocol Name Length* [1 octet] : égale à 19
- *Protocol Name* [variable] : La chaîne "BitTorrent protocol"

- *Reserved Extension Bytes* [8 octets] : octets prévus pour une extension éventuelle du protocole, non utilisés dans les implémentations actuelles.
- *SHA1 Hash of info dictionnary* [20 octets] : correspond à la clé *info* du dictionnaire contenu dans le fichier `.torrent`
- *Peer Id* [20 octets] (uniquement pour la personne qui répond à la poignée de mains) : Il s'agit de l'identifiant du pair, fournit dans la réponse du *tracker*. Si l'identifiant n'est pas le même, la connexion est rompue.

4.2 Les différents messages

keepalive : Il s'agit d'un message de taille 0, envoyé toutes les deux minutes. C'est le seul message qui ne commence pas par un *type*.

choke/unchoke/interested/not interested : Tout au long de l'échange avec un pair, deux bits d'état sont utilisés : choqué (*choked*) ou non, et interressé (*interested*) ou non.
On dira qu'on choque (*choke*) le pair, quand on lui refuse le téléchargement temporairement. Ce refus sera levé quand on le déchoquera (*unchoke*).
Le transfert de données se fait quand l'un des pairs est intéressé et que l'autre n'est pas choqué.
Ces états sont communiqués à l'aide des messages :

code	type	description
0	choke	indique que le client est choqué.
1	unchoke	indique que le client n'est pas choqué.
2	interested	indique que le client est interressé
3	uninterested	indique que le client n'est pas interressé

bitfield : Uniquement envoyé en tant que premier message. Sa charge utile est de type bitmap. Les bits correspondant aux indices des pièces envoyées sont positionnés à 1, à 0 sinon. Le premier octet correspond aux indices allant de 0 à 7, le second de 8 à 15, etc.

type = 5 (bitfield)	xxxxxxxx	...	xxxxxxxx

have : Lorsque le client complète une pièce et valide son empreinte, il envoie ce type de message qui contient l'indice de la pièce. De cette façon, à tout moment, un pair est au courant de ce que possèdent les pairs avec qui il est connecté.

type = 4 (have)	*indice*

request : Pour demander la partie d'une pièce que l'on souhaite recevoir.

type = 6 (request)	*index*	*begin*	*length*

begin désigne l'offset dans la pièce correspondant à *index*, et *length* la longueur qu'on souhaite récupérer.

cancel : Sont de la même forme que les *request message*. Ils sont généralement envoyés uniquement à la fin d'un téléchargement, pendant ce qui est appellé le *endgame mode* et permettent d'annuler le téléchargement d'une sous-pièce (c.f section 4.4.2).

type = 8 (cancel)	*index*	*begin*	*length*

piece : Contient l'indice de la pièce, l'offset à l'intérieur de la pièce ainsi que les données. Il est possible qu'une pièce arrive de façon innatendue lors d'une succession rapide de messages *choke* et *unchoke* ou quand le transfert est très lent.

type = 7 (piece)	*index*	*begin*	*piece* [taille variable]

4.3 Stratégie d'émission

Un pair reçoit du *tracker* une liste d'une cinquantaine de pairs et tente de maintenir une connexion avec en moyenne entre 20 et 40 pairs. Cela ne veut pas dire qu'il y a forcément un échange entre ces pairs. Un pair ne limite pas le nombre de pairs sur lesquels il télécharge simultanément. Par contre, il *upload* uniquement vers 4 pairs simultanément. On dit de ces 4 pairs qu'ils sont *unchoked*, les autres sont *choked*.

4.3.1 Réciprocité des échanges

Les pairs récompensés sont ceux sur lesquels on télécharge le plus. Ainsi, on *unchoke* les pairs proposant le taux d'*upload* le plus élevé (c'est à dire qu'on les autorise à télécharger). Si un pair n'est pas *interested*, il est *choked*, un autre prendra sa place.

Ce "tableau d'honneur" est réévalué toutes les 10 secondes (cela permet d'éviter les "fibrillations" qui sont des successions rapides de *choke* et *unchoke*) et se base sur les taux d'*upload* des 20 dernières secondes.

Dans le cas d'un *seed*, la stratégie est la même, sauf que le classement est établie en favorisant les *leechers* qui téléchargent le plus vite.

4.3.2 *Optimistic-unchoking*

Tant qu'un pair n'a rien reçu, il ne peut rien envoyer. Et s'il n'envoie rien, alors personne ne lui enverra quoi que ce soit. Pour éviter ce cercle vicieux, BitTorrent utilise un algorithme d'*optimistic-unchoking*. Il s'agit là d'envoyer à un pair quelque soit son taux d'émission. Le pair qu'on *unchoke* dans le cadre de l'*optimistic-unchoking* tourne toutes les 30 secondes.

De plus, pour donner la possibilité d'obtenir une pièce à uploader, les nouvelles connexions ont trois fois plus de chances que les autres de bénéficier du *optimistic-unchoking*.

Cette politique optimiste fait également office de "premier pas coopératif", comme évoqué dans la théorie des jeux. Ce premier pas n'est pas totalement gratuit, celui qui le fait espère découvrir de cette façon des pairs offrant de bons taux d'*upload*.

4.3.3 *Superseed*

Le politique *superseed* a pour but d'aider un *seed* à uploader toutes ses pièces le plus rapidement possible, ainsi le fichier est répliqué plus rapidement sur le réseau. Le principe est de toujours fournir des pièces différentes à chaque demande. Comme vu précédement, c'est le pair qui choisit le bout qu'il souhaite télécharger. Pour contourner cela, le *superseed* joue un double jeu envers les pairs en feignant de ne pas avoir certaines pièces (donc imite un pair *leecher* quelconque), il force donc la main sur les morceaux qu'il souhaite envoyer. De plus, pour favoriser les clients qui redistribuent, le *seed* envoie une seule pièce dans un premier temps, et envoie le reste uniquement une fois qu'il aura vu cette première pièce chez un autre pair.

Cet algorithme serait de 150% à 200% plus efficace que l'algorithme normal dans le cas du *seed* initial. Il est par contre totalement proscrit dans le cas des autres *seeds*.

Le *superseed* n'est pas un algorithme de la spécification officielle.

4.4 Stratégie de réception

4.4.1 *Rarest first*

Dans la description officielle du protocole, le choix des pièces à télécharger est aléatoire. Cependant, la plupart des clients appliquent le principe du *rarest first*. Cela veut dire qu'un pair choisit de télécharger la pièce la moins dupliquée parmi les autres pairs. Plusieurs raisons attestent le bien-fondé de ce choix :

- Alléger le *seed* : Cette méthode assure que seules de "nouvelles" pieces sont téléchargées du *seed*. Au début, le *seed* représente un goulot d'étranglement. En téléchargeant les pièces les plus rares, les pieces récupérées du *seed* sont celles qui n'ont pas encore été uploadées par d'autres pairs.
- Activer l'upload : Une pièce rare est très demandée par les autres pairs. Cela maximise donc l'utilité du pair qui en télécharge une, vis-à-vis des autres pairs.
- Les plus répandues en dernier : Il semble raisonnable de télécharger les pièces les plus répandues à la fin. En effet, il y a de fortes chances pour que celles-ci soient encore présentes plus tard.
- Empêcher les pièces rares de disparaître.

4.4.2 *Endgame mode*

En début ou au milieu de la réception d'un fichier, le download à partir d'un pair offrant un taux d'upload faible ne pose pas de problème. En effet, cela ne risque pas de retarder la durée globale du transfert. En revanche, à la fin du téléchargement, le fait de recevoir lentement (alors qu'on pourrait recevoir plus rapidement ces même pièces à partir d'un autre pair) risque d'introduire des délais supplémentaires. Le *Endgame mode* consiste, en fin de téléchargement, à réclamer toutes les sous-pièces manquantes à l'ensemble des pairs. Pour éviter le gaspillage de bande passante dû aux réceptions redondantes, des messages d'annulation (de type *cancel*) sont envoyés.

5 performances, limitations et améliorations

5.1 Succès et performances

BitTorrent semble remplir son objectif : fournir une diffusion rapide et efficace de gros fichiers. De plus, BitTorrent est très scalable, tout en maintenant un taux de téléchargement relativement élevé. Une multitude de clients ont vu le jour, chacun apportant ses modifications ou ses ajouts au protocole officiel, tel Exeem qui a intégré un moteur de recherche de fichiers.

En quelques années, il s'est imposé comme le logiciel de partage de référence. Certains évoque le chiffre de 35% du traffic total de l'internet engendré par des transferts BitTorrent.

Très adapté aux sorties "événementielles" (qui entraînent beaucoup de téléchargements sur une période courte), BitTorrent a trouvé grâce non seulement auprès des échangeurs de contenu illicite, mais également d'organismes plus officiels. En effet, l'ascension de BitTorrent est fortement liée à l'échange non autorisé de documents soumis aux *copyright*. Alors que la liste des sites fermés pour avoir proposé du contenu copyrighté s'allonge, les sites légaux s'intéressent à BitTorrent.Ainsi, beaucoup de distributions Linux ont choisi de privilégier leur diffusion via ce mode. Voici quelques exemples de sites qui diffusent leurs médias par BitTorrent :

- Torrentocracy : Diffusion de débats politiques américains.
- Jamendo : Plate forme de musique libre.
- LinuxTracker : Images iso de distributions linux.
- LegalTorrents : Documents libres.

Parmi les études réalisées, un groupe de l'Institut Eurecom a analysé la "vie" d'un torrent pendant 5 mois. Ce projet concernait le torrent de la distribution Linux Redhat 9, d'une taille de 1.77GB. Les informations proviennent du fichier log du *tracker* ainsi que d'un client modifié. Le log contient les statistiques de plus de 180 000 clients sur 5 mois, dont 50 000 dans les 5 premiers jours.

5.2 Limitations

5.2.1 Les ports inaccessibles

Un client dont les ports utilisés par BitTorrent sont inaccessibles (soit à cause d'un NAT ou d'un *firewall*) se verra désavantagé. En effet, il ne pourra profiter que des connexions dont il est l'initiateur puisque les pairs ne pourront pas le joindre. De plus, deux clients dont les ports sont inaccessibles ne pourront pas communiquer entre eux. Cela revient à se priver d'une partie du réseau.

5.2.2 Le *Free-Riding*

La méthode du *free-riding* consiste à downloader à partir des pairs sans rien uploader en échange. Même si BitTorrent applique le principe de réciprocité des échanges qui semblent empêcher cette méthode,

- L'*optimistic-unchoking* utilisé pour aider les nouveaux arrivants, a pour effet d'aider les *free-rider*. Cependant ce bénéfice apporté par l'*optimistic-unchoking* est relativement faible : selon une étude, le *free-rider* bénéficie

en moyenne d'un taux de téléchargement de 20% de ce qu'il aurait pu espérer obtenir en coopérant.
- L'existence de *seeds* est une porte ouverte à la pratique du *free-riding*. En effet, comme vu précédement, les *seeds* uploadent aux pairs ayant le meilleur taux de download.

5.2.3 Concernant le *tracker*

On peut reprocher au *tracker* son rôle très passif. Il se contente en effet de mettre en contact les pairs de façon totalement aléatoire. Une façon de faire serait par exemple de prendre en compte la localisation géographique ou les pièces qu'un pair peut offrir.

Une autre critique vis-à-vis du *tracker* concerne le fait qu'il constitue un élément centralisé et donc, le seul obstacle à une montée en charge. En effet, même si la bande passante utilisée par le *tracker* est relativement faible, elle constitue tout de même un facteur limitatif à partir d'un certain point.
Une solution proposée dans la dernière version du client officielle est l'utilisation d'un mode *trackerless*.

Dans ce contexte, chaque pair joue le rôle d'un *tracker léger* et maintient une table de routage vers quelques pairs. Ce procédé est basé sur une table de hachage distribuée Kademlia. L'annuaire des pairs est distribué et aucun client n'en possède une vue complète.

Pour atteindre un client, un protocole itératif est utilisé qui permet de se rapprocher de la destination de plus en plus. Pour initier le processus, la table de routage doit contenir au moins une entrée valide.
Les clients acquièrent l'information au contact d'autres clients pendant l'utili sation normale, mais un client nouvellement arrivé n'a aucune porte d'entrée pour intégrer le réseau.

Il y a deux méthodes pour qu'un client qui se connecte obtienne des contacts pour le système *trackerless* :

1. La première méthode utilise les *trackers* classiques. Un client BitTorrent supportant le système *trackerless* donnera des informations sur des contacts avec le nouveau client pour le système *trackerless*.
2. la seconde méthode, permet de démarrer le système *trackerless*, en employant les informations du contact stockées dans les fichiers *torrent* trackerless. Quand vous créez un `.torrent` *trackerless*, les informations des contacts de votre table de routage sont automatiquement incluses pour aider les nouveaux clients qui ne possèdent pas de contacts. Une méthode alternative est d'inclure les informations de clients connus comme étant stables ou alors d'utiliser des clients dédiés spécifiquement au routage.

5.3 Les concurrents de BitTorrent

5.3.1 EDonkey

Le protocole Edonkey malgré son âge vieillissant et les fermetures à répétition de certains de ses serveurs les plus utilisés, est son principal concurrent

car toujours très actif. Malgré tout, il ne dispose pas du soutien de sites de distributions de contenu légal, dont bénéficie BitTorrent.

5.3.2 Slurpie

Slurpie utilise un protocole développé dans le même but que BitTorrent utilisant à la place du *tracker*, un *topology server* permettant de se connecter à des utilisateurs quelque soit le fichier recherché et d'en augmenter le nombre en fonction du besoin. Les algorithmes mis en oeuvre par Slurpie sont beaucoup plus complexes que ceux de BitTorrent et nécessitent d'estimer le nombre de pairs sur le réseau. Malheureusement le projet Slurpie a été abandonné depuis plus de 2 ans.

5.3.3 PDTP (Peer Distributed Transfer Protocol) ou Squall

La principale amélioration par rapport à BitTorrent se situe dans la multiplicité des composants déployables : un réseau PDTP peut contenir plusieurs serveurs, l'équivalent des *trackers*, plusieurs *Piece Proxy*, qui aident la source à diffuser les données et des *proxy* pour passer les firewalls

5.3.4 Avalanche

Avanche est le nom de code d'un projet de protocole pair à pair developpé par Microsoft dans l'espoir de remplacer BitTorrent. Son domaine concerne particulièrement la télévision à la demande et la distribution de *patches* et d'applications. Avalanche se base sur le principe de *Network Coding*, qui utilise l'échange de combinaisons linéaires de pièces. Les annonces de Microsoft semblent prometteuses, reste à savoir si le résultat sera à la hauteur, à moins qu'il ne s'agisse d'un simple "*vaporware*".

6 Conclusion

Bram Cohen a réussi à mettre au point un protocole particulièrement efficace qui marque un tournant dans le mode de distribution pair à pair. Le succès est au rendez-vous puisque de nombreux organismes ont choisi ce protocole. De plus, cela semble n'être que le début avec la nécéssité et la demande d'échanger de plus en plus de contenu (le développement des systèmes de *Video-on-demand* et autres formes de téléchargements légaux).

Cependant BitTorrent n'est pas optimal et est loin d'être parfait. De nombreux protocoles visent à le remplacer. Certains se vantent d'être meilleurs, d'autres disposent de l'appui de puissants groupes tel Microsoft. Mais BitTorrent continue d'évoluer, Bram Cohen lui-même continue d'améliorer son protocole en introduisant de nouvelles fonctionnalités, comme récemment le mode dit *trackerless*.

Références

[1] B. Cohen, *BitTorrent protocol specification.*
http ://www.bittorrent.com/protocol.html.

[2] B. Cohen, *Incentives Build Robustness in BitTorrent.*
http ://www.bittorrent.com/bittorrentecon.pdf.

[3] G. de Menten *BitTorrent le chaînon manquant des protocoles peer to peer ? ,2004*

[4] M. Izal, G. Urvoy-Keller, E.W. Biersack, P.A. Felber, A. Al Hamra, and L. Garcès-Erice *Dissecting BitTorrent : Five Months in a Torrent's Lifetime*

[5] A. R. Bharambe, C. Herley, V. N. Padmanabhan *Analyzing and Improving a BitTorrent Network's Performance Machanisms*

[6] J. Keller *BitTorrent*

[7] J. A. Johnsen, L. E. Karlsen, S. S. Birkeland *Peer-to-peer networking with BitTorrent*