Conception d'une architecture Pair-à-Pair orientée opérateur de services

Radwane Saad

Thèse

présentée pour obtenir le grade de

DOCTEUR de TÉLÉCOM PARISTECH
Spécialité: **Informatique et Réseaux**

Radwane SAAD

Une Architecture Pair-à-Pair orientée Opérateur de Services

Soutenue le 17 Septembre 2010 devant le jury composé de :

Ken CHEN	Université de Paris XIII	*Président*
Bijan JABBARI	George Mason University	*Rapporteur*
Abdelhamid MELLOUK	Université de Paris XII	*Rapporteur*
Patrick BELLOT	Télécom ParisTech	*Examinateur*
Mohammed ACHEMLAL	Orange Labs, France Telecom	*Examinateur*
Youcef BEGRICHE	Université de Paris V	*Invité*
Ahmed SERHROUCHNI	Télécom ParisTech	*Directeur de Thèse*

Thesis

Presented to obtain the degree of

DOCTOR of TÉLÉCOM PARISTECH
Speciality: **Computer Science and Networks**

Radwane SAAD

A Service Provider Oriented Peer-to-Peer Architecture

Defended on the 17th of September, 2010 before the jury composed of:

A mon père Habib, mon unique modèle
A ma mère Maha, ma source de vie
A ma sœur Nassrine, mon âme sœur
A ma sœur Sabrine, mon précieux trésor
A ma moitié Isaure, mon éternel amour

« Tu ne réussiras qu'en partageant ton temps en suivant la règle des 24 pouces des anciens, telles les 24 heures d'une journée : huit heures seront consacrées au divin, huit heures te seront consacrées et huit heures seront consacrées aux autres . . . »

Habib Hussein Saad

« La vie est quête de foi et de lumière . . .
Réussir consiste à savoir tailler sa propre pierre . . .
Les outils sont volonté et mesure . . .
La seule règle est la droiture . . . »

Radwane Habib Saad

Remerciements

Je profite de cette page pour remercier l'ensemble des personnes qui de près ou de loin ont contribués à l'achèvement de cette thèse.

C'est avec la plus grande reconnaissance et le plus grand respect que je remercie infiniment mon directeur de thèse M. Ahmed SERHROUCHNI pour m'avoir si bien dirigé durant ces années avec patience, dévouement et sympathie. Il a su m'apprendre à surmonter avec courage les dures épreuves de la vie d'un doctorant.

Je remercie M. Bijan JABBARI de faire le déplacement de si loin et d'avoir accepté rapporter ma thèse. M. Abdelhamid MELLOUK a également eut la gentillesse d'en être le deuxième rapporteur malgré sa charge importante de travail. Je pense aussi à M. Ken CHEN qui a accepté de faire parti du jury et surtout d'avoir toujours été présent pour me guider et avoir réponse à mes nombreuses questions. Son acuité et sa disponibilité ont été précieuses pour moi. Je remercie M. Patrick BELLOT de Télécom ParisTech qui fera l'honneur d'être un des examinateurs et d'avoir un regard critique sur mes travaux, lui-même étant impliqué dans le monde du pair-à-pair. Je remercie M. Mohammed ACHEMLAL d'avoir accepté d'être dans mon jury et de se déplacer de Caen à cet effet. Il sera le juré le mieux placé pour apprécier les éléments impliquant l'opérateur et le fournisseur de service. Je ne peux oublier un ami M. Youcef BEGRICHE qui a toujours été la pour orienter mes réflexions et m'aider à toucher au formalisme mathématique et pour soutenir mon moral au jour le jour.

Je remercie M. Maroun CHAMOUN sans qui je n'aurais jamais eut la chance d'entamer ce travail au sein de Télécom ParisTech. Il a su m'ouvrir la voie vers le troisième cycle.

Je remercie M. Philippe GODLEWSKI pour sa rigueur hors norme qui m'a aidé à viser plus haut. Je n'oublie pas M. Jean LENEUTRE, M. Ahmad FADLALLAH, Mme Houda LABIOD et Mme Nadia BOUKHATEM pour leur gentillesse, leurs conseils et leur encouragement.

Je remercie mes amis et frères Daniel, Julien et Benoit qui ont m'ont soutenu durant ces années.

Je remercie sincèrement mon oncle Nabil et mes cousins Rayan et Roman pour avoir su m'aider à toujours être sur le droit chemin.

Je ne remercierai jamais assez ma fiancée Isaure GILLOT pour son sincère amour et sa fidèle présence à mes côtés sans lesquels je n'aurai pu avancer.

Je remercie du fond du cœur ceux sans qui je ne serais pas ce que je suis aujourd'hui et ceux qui représentent tout ce que j'ai de plus cher: mon père, modèle de sagesse et de bravoure, ma chère mère pour son amour qui est mon éternelle source de vie, mes sœurs Nassrine et Sabrine qui sauront à jamais me donner la force de grandir.

Je termine en m'inclinant devant l'intercession divine …

Une architecture Pair-à-Pair orientée opérateur de services

Les paradigmes et architectures du pair-à-pair (P2P) sont au centre des réalisations d'applications à grande échelle de tout type. Les architectures à base de résilience, le grid computing ou distribution de traitement, le partage de fichiers, la distribution de données sont ainsi de plus en plus basés sur des infrastructures overlay.

Il est nécessaire d'intégrer un niveau de contrôle sur de telles applications. Ce contrôle peut servir de modèles économiques, intégrer de la sécurité, améliorer la qualité de service (QoS), et cela pour atteindre des objectifs de qualité diversifiés. De telles applications seront ainsi opérées et auront comme maître d'œuvre un opérateur de services.

Dans la pratique actuelle les entités pairs partageant des ressources se placent d'une manière aléatoire sur un large réseau physique (IP). Par ailleurs, certaines applications notamment de distribution de données à contraintes temporelles sont exigeantes en délai et bande passante. Utiliser un tel réseau de recouvrement pour de telles applications nécessite une organisation particulière entre les pairs.

Nous proposons ainsi la conception d'une architecture globale pour la mise en place de telles applications sur des plateformes de type P2P.

Dans ce paradigme il est possible d'isoler trois principales composantes : la première est celle qui concerne le service applicatif proprement dit, la deuxième est le routage (ou la recherche d'information), la troisième est celle qui traite du transport des données.

Nous nous orientons vers une architecture où les réseaux sont divisés en différentes zones ou systèmes autonomes pour Autonomous Systems (ASs) contrôlés par des Opérateurs de Services. Ce travail consiste à optimiser chaque composante du modèle P2P pour atteindre les meilleures performances en se basant sur les différentes exigences des applications. Ces études nous permettent de spécifier des structures pour trois principales contributions. La première a pour objectif de cloisonner le trafic P2P et, après généralisation, d'appliquer un algorithme sensible au contexte où chaque zone ou groupe de pairs (appartenant à un même AS ou partageant les mêmes intérêts ou performances) est basé sur une DHT. La seconde est d'accélérer le transfert des données à l'aide du mécanisme FEC. La troisième est d'intégrer une entité de Contrôle/Gestion qui se charge de gérer les deux précédentes propositions et de varier des paramètres basiques du protocole BitTorrent, utilisé pour la couche transport de l'architecture, afin que l'application utilise le socle du P2P dans les meilleures conditions.

Ces contributions ont pour objectifs principaux de minimiser le temps de téléchargement de la ressource et de diminuer le trafic *peering*, tout en gardant les caractéristiques du P2P à savoir la robustesse et l'interopérabilité.

Nous avons effectué de nombreuses simulations à grande échelle qui valident nos propositions. En effet, nous montrons que cloisonner le trafic peut avoir un apport positif en particulier lorsqu'il est complété par un algorithme de routage sensible au contexte. Une étude complète du mécanisme FEC appliqué au protocole BitTorrent montre combien la correction d'erreur peut accélérer le transfert de données dans certains scénarios, à la fois pour des réseaux homogènes ou hétérogènes. Enfin, nous avons groupé ces contributions pour proposer une architecture P2P orientée Opérateur de Services appelée SPOP pour Service Provider Oriented Peer-to-Peer [1].

Mots-Clés: BitTorrent, DHT, FEC, Localisation, Overlay, Pair-à-Pair, Performance.

1. Pourquoi une telle architecture ?

Les réseaux P2P connaissent une grande expansion et de multiples applications y sont intégrées à moindre coût et avec un meilleur facteur d'échelle. Nous avons comme exemples les applications de partage de fichier qui ne demandent pas particulièrement de QoS ou encore la VoIP et l'IPTV qui au contraire sont des applications temps réel pour lesquelles les réseaux doivent être plus fiables. Le challenge des opérateurs ainsi que des équipementiers est d'intégrer un contrôle et une gestion au trafic P2P dans les réseaux actuels. En effet, le réseau IP n'est pas seul capable d'intégrer de telles possibilités. Ce contrôle est un besoin pour l'opérateur afin de limiter l'importance du trafic inter systèmes autonomes qui peut élever considérablement le coût d'utilisation de son trafic externe. A cela s'ajoute l'importance pour ces opérateurs de services d'améliorer les performances en diminuant le temps de téléchargement de la ressource pour un client ou encore le temps de recherche de celle-ci.

Le modèle P2P peut être décomposé en trois principales composantes qui sont:

- La première composante est celle du transport propre des données dans laquelle on définit la spécification du protocole et les règles d'échange des messages [2]. A ce niveau nous choisissons le protocole BitTorrent comme méthode de transport.
- La deuxième composante concerne le routage et la recherche des ressources. L'utilisation des tables de hachage distribuées (DHT) est de plus en plus intéressante quant à leurs efficacité et robustesse [3].
- La troisième composante décrit le service fournit au niveau applicatif proche de l'utilisateur dépendamment du contrat entre l'opérateur et le client.

La plupart du trafic P2P est généré par des applications qui sont totalement indépendantes des opérateurs de services et de leurs infrastructures. Avec le succès des algorithmes DHT qui touchent directement le routage P2P, des propositions telles que [4] ou [5] définissent un moyen de créer une interface entre un opérateur de services (et son AS) et les entités P2P clients. Cependant, ces solutions sont complexes et nécessitent un nombre important de changement et un manque d'interopérabilité.

SPOP est une solution alliant robustesse, interopérabilité et simplicité. Les étapes suivies sont les suivantes :

- Optimiser le niveau de transport en proposant un mécanisme pour assurer une entropie maximale des segments avec un taux de perte minimal.
- Garantir au niveau routage une complétude pour chaque requête avec le taux le plus élevé en minimisant le nombre de sauts et les messages de signalisation.
- Elaborer des spécifications pour chaque service et ses besoins : 1) contrôler le trafic généré à l'intérieur d'un même AS et entre les ASs pour des raisons de coûts ; 2) garantir un environnement adapté au service nécessitant par exemple un minimum de délai pour des applications temps réels.

Un premier chapitre d'introduction présente la problématique de la thèse ainsi que les solutions et contributions. Un deuxième chapitre présente des généralités sur les réseaux P2P, plus particulièrement sur BitTorrent, ainsi que les différents travaux existants sur les sujets traités dans ce manuscrit (performance, localisation des pairs et correction d'erreur dans BitTorrent). Les trois chapitres suivants détaillent les travaux et contributions de la thèse. L'avant dernier chapitre traite de l'application de sécurité développée pour valider l'architecture SPOP. Le dernier chapitre conclut ce manuscrit.

2. Contributions et travaux

A. hTracker : gestion et contrôle du trafic

Le trafic P2P à l'intérieur d'un même AS est plus facile à contrôler que celui entre les ASs. Lorsqu'un pair au sein d'unAS sollicite des pairs d'un autre AS, le premier AS est chargé de payer le trafic récupéré du deuxième AS. Ce trafic doit donc être réparti entre les pairs de manière équitable. L'augmentation du trafic à l'intérieur d'un même AS peut être causée par un routage qui n'est pas optimisé ou encore la prolifération de messages inutiles. Ce nombre n'est pas contrôlé par l'opérateur. Pour le trafic entre les ASs, le modèle P2P est un modèle distribué et les protocoles tels que BitTorrent implémentent une politique de choix de pairs qui est totalement aléatoire. En effet, une absence quasi-totale de contrôle des protocoles P2P ne permet pas de réguler ce trafic. D'autant plus que le P2P représente plus de 60 % du trafic Internet global, avec 25 à 30 % du trafic global seulement pour BitTorrent (tout clients confondus).

Nous avons fait le choix de BitTorrent qui est probablement le protocole le plus populaire du monde P2P pour le partage de fichiers. Ce choix se fait pour le transfert de l'information dans l'architecture SPOP.

Le système de routage Internet est composé de plusieurs systèmes autonomes. Pour acheminer le trafic Internet les ASs entretiennent des relations entre eux. Des accords de *peering* sont à la base de ces relations. Les principales catégories de ces relations sont : client à fournisseur (C2P), pair à pair (p2p) et sibling à sibling (S2S). Le problème est que le nombre de connexions inter-ASs a un impact sur les performances des pairs et sur le trafic inter-domaines. Le premier impact est dans l'overhead qui augmente linéairement avec le nombre des connexions inter-ASs. Cela peut coûter cher à l'opérateur, en particulier si le trafic ne peut être contrôlé, et c'est le cas des applications P2P. En général le trafic extrait d'un AS externe doit être payé par l'AS d'origine. Habituellement, les ASs larges ne paient pas de la même manière que les plus petits ASs car ils sont souvent le carrefour d'un volume très important de trafic et qu'ils ont donc des accords leur permettant de disposer de facilités avec les ASs de même niveau ou avec des ASs de niveaux inférieurs. Toutefois, lorsque l'AS est de taille relativement moyenne ou petite, il peut avoir à payer si des routes de plus grands ASs sont empruntées.
Le deuxième impact est le ralentissement du téléchargement et ce problème peut être évité en choisissant des pairs situés en majorité dans le même domaine, afin de limiter le temps de propagation lors des échanges ainsi que le trafic inter-domaines. Un aspect complémentaire important est d'avoir des Seeds initiaux rapides offrant une grande diversité de segments.

Notre étude est basée sur des objectifs que pourrait fixer un opérateur pour servir ses clients à l'égard de la qualité de service dans l'engagement contractuel signé entre le prestataire et le client. En même temps, les opérateurs sont tous sous les termes de ce qu'on appelle des stratégies de *peering* qu'ils doivent respecter. L'objectif principal pour les opérateurs est de limiter les surcoûts de leurs systèmes sans pour autant dégrader le service rendu. Certaines applications largement utilisées aujourd'hui ont certaines contraintes temps réel de délai et de bande passante. Des applications comme la VoIP ou l'IPTV sont les meilleurs exemples qui illustrent l'importance de techniques et politiques pour atteindre ces objectifs. Toutefois, il n'est pas si évident et facile pour les applications P2P largement distribuées, dynamiques et sans contrôle d'obtenir un aperçu du comportement de tous les pairs dans le système.

Notre intérêt dans ce travail est de traiter de cette question de localisation, même si nous sommes conscients que ce paramètre n'est pas le seul aspect sur lequel un opérateur doit tenir compte pour respecter ses objectifs de *peering* et de QoS. La méthode originale et la plus utilisée pour contrôler le trafic P2P par les opérateurs est de limiter la bande passante par étranglement. Des dispositifs similaires sont utilisées pour façonner le trafic dans les routeurs de bordure. Cependant, l'inconvénient de ces dispositifs est qu'ils ralentissent les transferts de données et cela ne résout pas les problèmes de localité de chaque pair. On ne peut donc pas ainsi diminuer le trafic inter-domaines. Nous proposons de modifier la politique de sélection aléatoire des pairs par le *Tracker* pour une nouvelle politique plus intéressante. Le principe est de choisir la majorité des pairs du même AS que le Leech qui envoie la demande. Cette méthode, validée par des simulations, permet de réduire considérablement le trafic inter-domaines et le temps de téléchargement. Dans les propositions antérieures, la correspondance des pairs avec leurs AS n'a pas été définie avec précision. On propose une sémantique de cette cartographie avec les *peerIds* de chaque pair.

Cette proposition a été publiée dans [6] (*cf* Figure 1). Nous choisissons de sélectionner des pairs au sein du même AS. On permet ainsi d'ajouter un plan de contrôle et de gestion du trafic au modèle P2P. Pour associer le pair à l'AS auquel il appartient nous proposons une sémantique spécifique pour le *peerId* de chaque pair du réseau à l'aide d'une fonction de HMAC. Une évolution à cette technique permet de varier la taille de la liste de pairs envoyée au pair intéressé. Le contrôle et la gestion du trafic sont effectués par une entité appelé *hTracker*.

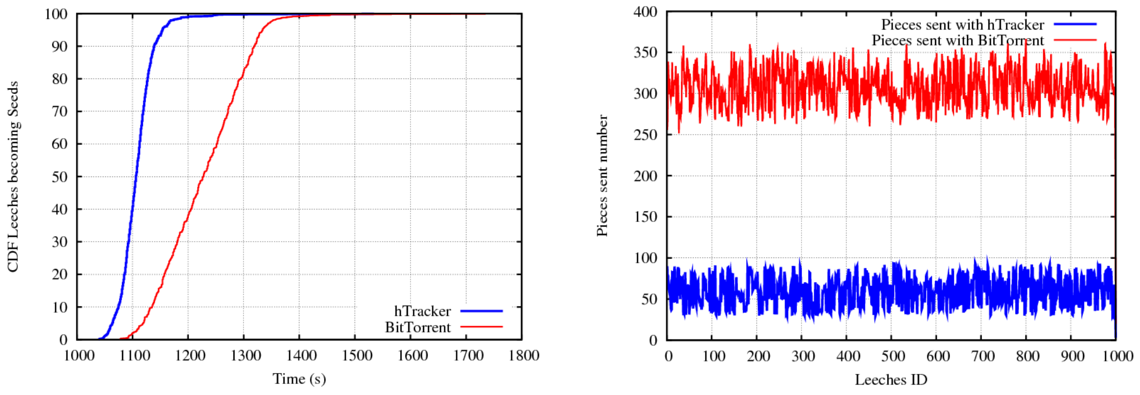


Figure 1. Diminution du temps de téléchargement et du trafic inter-ASs

B. Forward Error Correction : maximiser l'entropie des segments

Il existe un gain de performance entre le codage réseau, le codage de source et BitTorrent sans codage. La perte de l'efficacité du codage de source est principalement due au fait que la propagation des segments au sein du réseau par la duplication introduit des pertes de bande passante, en raison de coûts dans le réseau.

Il y a deux points importants à considérer quand il est d'appliquer un mécanisme d'erreur, en particulier dans notre cas :

- Le premier point est le niveau du codage dans les données. Dans BitTorrent, il est possible d'appliquer le codage au niveau du bloc ou le niveau du segment. L'avantage du codage au niveau plus fin (bloc) est que, pour le remplacement de certains blocs d'un segment, il n'est pas obligatoire de récupérer tout le segment. Il est possible que seuls quelques blocs d'un segment soi remplacés. L'inconvénient est que le traitement est plus important et complexe.

- Le deuxième point est la comparaison entre le codage réseau et le codage FEC. Il est démontré que lorsque le réseau BitTorrent et l'ensemble des pairs est suffisamment important, l'entropie du Rarest First (politique de sélection de segments dans BitTorrent) est proche de 1 et aucun codage est vraiment nécessaire. Le choix dépend des besoins de l'application et du niveau des coûts acceptables par les développeurs qui ont choisi de le mettre en œuvre. Pour certaines applications comme les applications temps réel, les opérateurs veulent s'assurer davantage de garantir que leurs clients puissent choisir d'appliquer le mécanisme de codage, même si le coût et le traitement est plus élevé ou plus complexe. Le choix du codage réseau est également plus compromettant car même si les performances sont meilleures que le codage de source, le traitement est nécessaire au niveau de tous les pairs du réseau puisque le principe est de faire participer tous les pairs au mécanisme de codage. Le principe de codage réseau est d'échanger des informations et de fusionner les données. Un autre inconvénient de choisir le codage réseau est qu'il n'est pas interopérable avec le client BitTorrent générique. Avalanche, qui mettrait en œuvre du codage réseau, ou BitCod qui est le client proposant un codage réseau, sont différents de BitTorrent et proposent leurs propres algorithmes et mécanismes.

Nous choisissons d'appliquer un mécanisme de FEC plus simple et d'évaluer les scénarios avec lesquels il présente des avantages réels sans dégradation des performances du réseau. Il s’agit aussi d’éviter l’ajout excessif de traitement et de complexité. L'interopérabilité avec la version précédente est aussi notre objectif principal.

Prenons un fichier donné, il est fractionné en segments et ces segments sont aussi divisés en blocs dans la spécification originale de BitTorrent. Nous modélisons uniquement l'échange des segments entre pairs sur le simulateur. Soit *k* le nombre de segments qui forment le fichier d'origine, un certain nombre de segments redondants *(n x k)* peuvent être injectés par la source avec *n* le nombre total de segments retirés de l'encodeur FEC. Dans le cas de notre application, c'est le Seed qui fournit ces segments de redondance. Une fois le simulateur en cours d'exécution, au lieu d'injecter des segments *k*, *n* segments sont disponibles en sachant que l'un des segments *n* - *k* peut remplacer n'importe quel segment *k*. Dans ce cas le taux de codage *n/k* donne le pourcentage de redondance. Par exemple, le ratio FEC = *n/k* = 150/100 = 1,5 c'est à dire 50 segments redondants. Notez qu'il n'y a aucun changement d'algorithmes dans le simulateur. L'algorithme Rarest First est appliqué d'abord et dans ce cas, il s'applique au Seed qui injecte les segments *n* et non pas aux segments *k*. Le End Game Mode n'a pas été modélisé, car le problème du dernier segment peut être résolu par la présence de segments codés FEC.

Les travaux de recherches sur les mesures de performance de BitTorrent sont variés et les différents cadres proposés sont généralement basées sur des traces réelles. Nous décidons de nous concentrer sur le niveau segment.
Le simulateur à événements discrets développé est le même que celui utilisé pour la validation de la contribution hTracker. La mise en œuvre du mécanisme de FEC dans notre simulateur a été réalisé par la production de segments spéciaux basés sur les codes Reed Solomon qui sont générés par les Seeds. Il est possible de gérer la prolifération des segments codés et faire varier le rapport FEC. Si la valeur du rapport est égal à 1, cela signifie qu'il n'ya pas de segments FEC injectés dans le réseau. Par exemple, une valeur de ratio de 1,2 signifie que 20% du nombre initial de segments sont ajoutés aux segments initiaux. Pour une ressource de 100 segments, un Seed génère 20 segments supplémentaires et fournit enfin un total de 120 segments. Un pair peut récupérer 100 segments parmi les 120 prévus pour effectuer le téléchargement. Chaque segment téléchargé à partir des 20 segments codés peut compenser l'un des 100 segments d'origine. Dans nos travaux le FEC

est assuré pendant toute la durée des simulations. Il s'agit de conclure sur la façon dont l'ensemble du système réagit avec les segments FEC dans les différentes périodes du téléchargement.

Au niveau transport les principaux problèmes concernent la pénurie de segments due à la capacité faible des clients ou parce qu'elles subissent le problème du dernier bloc. La complétude des requêtes et le téléchargement de ressources dépendent directement des propriétés du réseau à chaque instant et lorsque le réseau est affecté cela touche directement le transport.

La deuxième contribution propose une étude complète de l'implémentation du mécanisme de correction d'erreur Forward Error Correction (FEC) [7] (*cf* Figure 2) plus simple à intégrer que des solutions telles que le Network Coding [8]. Nous montrons ainsi que pour des Leechs relativement rapides en terme de capacité de téléchargement et d'envoi, les segments FEC permettent d'accélérer considérablement la vitesse de récupération de la ressource. Lorsque le nombre de Leechs est trop important et que celui des Seeds ne l'est pas assez, les segments FEC peuvent aussi être une solution de secours pour maximiser l'entropie. Nous montrons tout de même que dans certains cas, où le réseau n'est pas réellement en manque de ressources, l'ajout de FEC peut dégrader le transfert.

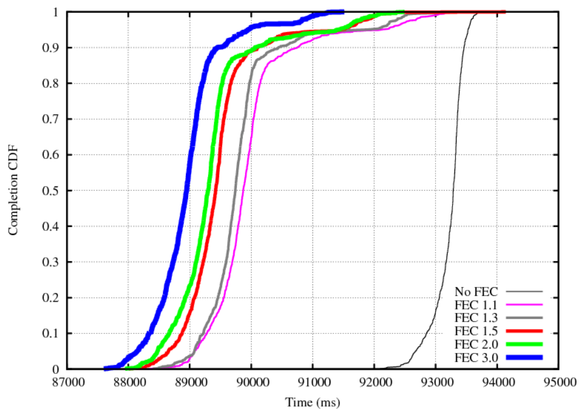


Figure 2. CDF pour 1000 pairs et un fichier de 100 MB avec 1 Seed initial

Pour l'ensemble des résultats de simulation nous choisissons de valider la comparaison des courbes obtenue par l'application d'un test statistique bilatéral.

C. L'architecture SPOP (Service Provider Oriented P2P)

La troisième contribution est la proposition d'une architecture globale introduisant les deux précédentes contributions. En effet, on généralise le contrôle de trafic en proposant un algorithme de routage sensible au contexte proposant la formation de plusieurs groupes basés sur des DHT. On propose un service privilégié pour fournir à certains clients qui le désirent des segments répliqués afin d'améliorer les performances de téléchargement. Enfin, cette dernière contribution propose l'instanciation volontaire de Seeds (fournissant aussi des segments FEC) et de Leechs selon les besoins de l'application pour augmenter la disponibilité et l'entropie des segments, la variation de la taille des segments, et cela pour la composante service du modèle P2P.

De nombreux travaux de recherches actuels dans les réseaux P2P sont axés sur la gestion du trafic et des techniques de la localisation pour le contrôle de l'activité de *peering*. Dans [9] et [10] la sélection des pairs a été modifiée pour choisir les pairs intra-domaine et réduire le trafic échangé entre les ASs. Ces techniques proposent un nouveau concept qui est la coopération entre les opérateurs et les applications P2P. Divers travaux proposent également d'optimiser la sélection par localisation géographique sans nécessairement avoir besoin de cette coopération. Par exemple, Ono [11] et TopBT [12] utilisent l'information de CDNs en se concentrant sur le calcul de performances sans architecture structurée pour les opérateurs. Ces propositions sont orientées client et non pas opérateur.

Les architectures proches de notre proposition sont ALTO P4P [4] et SmoothIT [5]. Tout d'abord, ces architectures sont axées sur le trafic P2P et les questions de qualité de service. SPOP propose également une optimisation du transport et de routage sensible au contexte (context-aware). Ce dernier est ajouté en complément au plan Gestion/Contrôle. Il est important de garder l'interopérabilité et la transparence avec toutes les applications. Voici les principales motivations pour la conception de SPOP :

- P4P propose une coopération entre les opérateurs et les applications P2P afin d'accélérer le téléchargement et optimiser l'utilisation des ressources réseau. Un plan de contrôle et est défini. Le *iTracker* du P4P permet de créer le lien entre le P2P et les opérateurs. Les applications P2P ont un *AppTrackers* qui communiquent avec les *iTrackers* pour obtenir des informations sur les décisions de *peering* (topologie du réseau, fournisseur de politiques et de capacités). Un des problèmes du P4P est que cela peut ralentir les transferts de clients non P4P. Ensuite, la coopération et le partage d'information est une idée nouvelle, mais aucune incitation n'a été proposée pour motiver les consommateurs à partager ces informations. Techniquement, il semble difficile d'intégrer les deux parties.
- SmoothIT partagent les mêmes objectifs clés que le P4P, mais SmoothIT est plus détaillé. En comparaison au P4P, SmoothIT fournit les spécifications pour la coopération entre les opérateurs et les *Trackers* pour les protocoles. En outre, SmoothIT prend en considération des demandes autres que celles du partage de fichiers et considère les contraintes de temps réel des applications. Le problème avec SmoothIT est qu'il nécessite de grandes modifications au niveau des entités comme les routeurs Internet en raison de la complexité de l'architecture.
- SPOP considère trois aspects principaux qui sont la simplicité, l'optimisation des performances, et surtout, l'interopérabilité avec les protocoles existants. Le plan de routage est défini sur un algorithme basé sur une DHT sensible au contexte (*cf* Figure 3) qui peut prendre en compte des paramètres différents et regrouper les pairs. Ce regroupement peut se faire par rapport à l'appartenance à un même domaine comme avec la contribution *hTracker*. Le plan de transport propose un mécanisme de FEC qu'un opérateur de services Internet peut proposer pour accélérer le transfert de données dans le cas où un

manque de ressources est important. Enfin le plan Gestion/Contrôle est basé sur les paramètres existants (politique de sélection des pairs, taille de la liste des pairs, taille des segments, etc …) qui peuvent être ajustés sans ajouter de complexité à l'Internet et l'infrastructure opérateur.

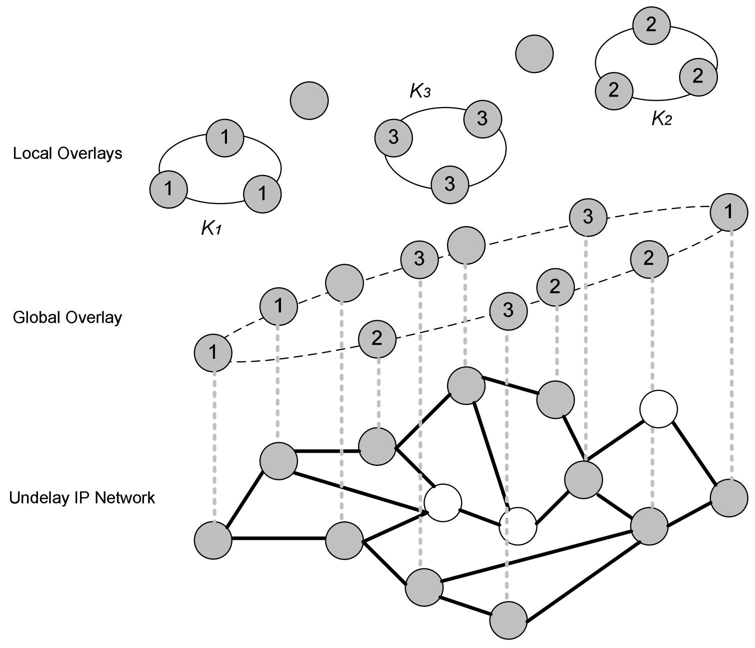


Figure 3. DHT sensible au contexte

3. Validation de SPOP

Les failles de sécurité des réseaux dans l'Internet d'aujourd'hui sont de plus en plus courantes et les attaquants modifient constamment leurs outils pour tenter de passer à travers les systèmes de défense mis en place. Le modèle de l'Internet à été créé dans le but de diminuer tout contrôle du trafic. Ceci peut cependant être considéré de nos jours comme un inconvénient. Plusieurs solutions de sécurité ont été proposées mais ne sont pas efficaces puisqu'elles détectent les attaques bien après que les dégâts ne soient constatés. L'attaque la plus crainte par les opérateurs de services actuellement est le Déni de service (DoS) et surtout dans sa composante distribuée (DDoS). Dans ce cas, un grand nombre d'attaquants sont impliqués, ce qui rend la détection plus difficile et l'impact plus important. Les attaques exploitent à l'origine les points faibles des protocoles. Cependant, l'infrastructure même de l'Internet est de plus en plus la cible d'attaques comme les sites Web, les banques et les fournisseurs d'accès aux services. Des technologies ont prouvées leurs grandes évolutions dans le cas de la détection d'intrusion. Mais au même moment, les intrusions deviennent de plus en plus fines et sophistiquées. Deux exemples concrets sont les attaques de Yahoo en 2000 ou récemment de l'Estonie en 2007. Plusieurs solutions de détection d'intrusion et de filtrage étudient la manière de représenter les attaques mais ces techniques nécessitent un grand nombre de changement dans la structure de base des protocoles Internet. En effet, les principaux logiciels qui rendent la détection difficile en termes de performance et d'efficacité; et dans la plupart des cas les propositions sont basées sur une entité centrale. Mais cette entité peut être également une cible parfaite pour les attaquants et être un point unique de défaillance (*Single Point of Failure*). Toutes ces constatations nous poussent à faire le choix d'introduire la coopération entre des entités de défense d'un réseau pour la détection d'intrusion en particulier et la traçabilité IP. Des systèmes comme DIDS ou NSTAT ont été développés mais n'ont pas réellement pu écarter la nécessité d'une entité centrale d'analyse. Par contre une approche purement distribuée et hiérarchique présente de nombreux avantages par rapport à une approche centralisée.

Nous avons choisi de développer une application de sécurité basée sur SPOP et qui implémente ses principales composantes afin de la valider. L'objectif est de proposer une architecture globale et modulaire qui permette de modéliser l'implémentation d'entités de sécurité pour la défense contre les attaques DDoS. Cette architecture est basée sur la coopération entre les nœuds et un échange d'informations sur un réseau P2P. Ces nœuds dans notre modèle peuvent être des sondes de détection d'intrusion fournissant aux modules P2P les informations d'attaques nécessaire à des applications au plus haut niveau pour réagir selon la menace. La solution proposée cible les attaques distribuées de type DDoS par corrélation des informations sur les trafics suspects détectés par les entités de sécurité distribuées sur l'ensemble du réseau. Dans ce cas, chaque nœud a une vision globale de l'activité d'intrusion par cette collaboration. Pour réussir à définir une telle architecture il est primordial de tenir compte de la performance du système global et de la facilité de déploiement. En effet, le traitement, la bande passante et le stockage doivent être minimisés et un mécanisme de sécurité doit être ajouté pour permettre le contrôle d'accès des entités au sein de l'architecture.

La figure 4 décrit chaque niveau de l'architecture. Nous la présentons avec un degré d'abstraction suffisant pour permettre à chaque niveau de se caractériser par des fonctions et rôles spécifiques et indépendants d'un niveau à un autre. En effet, un équipement peut tout à fait implémenter les fonctionnalités d'une ou plusieurs couches.

Le premier niveau est le plus proche du réseau physique. Nous l'appelons *Niveau Réseau*. Dans ce niveau un équipement appartient au réseau sous-jacent. Pour être plus spécifique, une entité de ce niveau peut être un routeur IP avec les fonctionnalités basiques de routage, d'adressage et de transport du trafic.

Le second niveau est le *Niveau Sécurité*. Ce niveau comprend les entités de sécurité et dans le cas de la détection d'intrusion ce sont les sondes IDS. L'implémentation d'autres modules sont possibles selon le type de solution. Dans le cas d'IDS, lorsque le trafic est analysé et qu'une attaque est détectée, une alerte est générée et une primitive est envoyée au niveau supérieur. Ce dernier fonctionne en suite en réaction aux alertes envoyées par le niveau Sécurité. Notons qu'un module sécurité peut être intégré à un équipement réseau et l'entité en question serait représentée par les deux premiers niveaux de l'architecture. Dans notre architecture nous n'abordons pas la partie réactivité aux alertes.

Le troisième niveau de l'architecture est le *Niveau SPOP* qui inclut la DHT sensible au contexte proposée pour l'indexation et la distribution des informations à travers les nœuds. Ce niveau reçoit les informations collectées concernant le trafic analysé du *Niveau Sécurité*. Lorsqu'une alerte est envoyée par le niveau inférieur, ce qui voudrait dire qu'une attaque ait été détectée, le niveau SPOP indexe les informations concernant ce trafic sur le nœud DHT qui gère ces informations selon la valeur de l'adresse destination IP de la victime (et donc de l'*objectId*). Le nœud avec le *nodeId* le plus proche numériquement se charge de garder ces informations. Nous pouvons dans ce niveau intégrer ce module à un équipement qui détient déjà les modules *Réseau* et *Sécurité*. Dans ce niveau nous avons également l'optimisation du transport de SPOP ainsi que la gestion des paramètres de QoS pour l'application de sécurité développée.

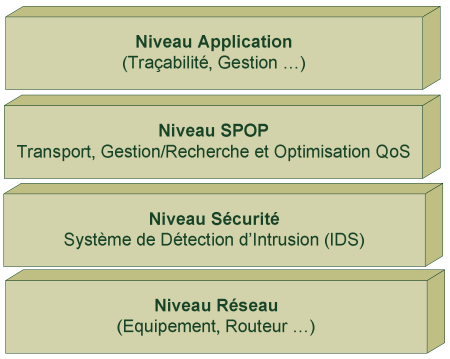
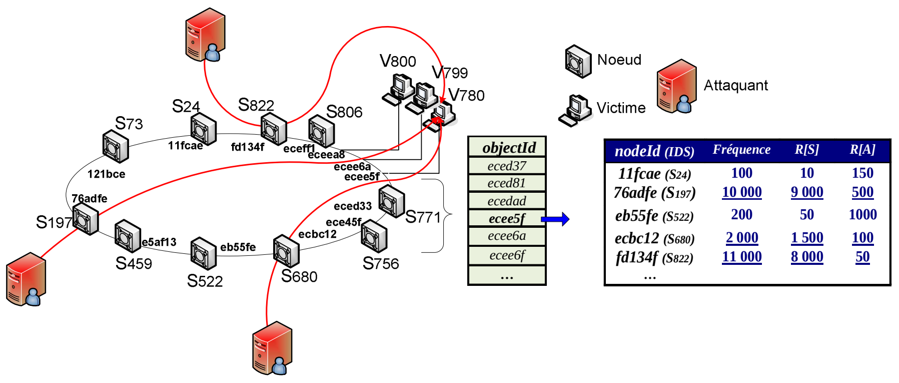


Figure 4. Les niveaux de l'architecture utilisée par l'application de sécurité intégrant SPOP

Le dernier niveau est le *Niveau Application*. Ce niveau est général dans notre architecture. Il peut intégrer toute application susceptible d'utiliser les informations d'alertes de sécurité indexées par le niveau P2P. En effet, nous proposons dans notre cas un mécanisme de traçabilité pour un système global de défense contre les attaques DDoS. Ceci fait partie des perspectives puisqu'une implémentation des trois premiers niveaux a été testé.

En retirant toute entité centrale d'analyse de l'architecture nous proposons une solution complètement décentralisée. Mais le choix de cette méthode doit fournir une certaine garantie sur la bonne corrélation des données indexées pour une détection des attaques DDoS et une réaction efficace à celle-ci. D'où la proposition d'applications qui se chargeraient d'analyser les informations de trafic et décider de la réaction en perspective à nos travaux.

Un exemple de réseau est illustré dans la figure 5, où des nœuds sont organisés logiquement sur un anneau. Pour une plus simple compréhension nous avons représentés les nœuds du niveau *P2P* de notre modèle mais qui intègre également les niveaux Réseau et Sécurité. En effet, un trafic venant d'un attaquant est détecté par le module IDS d'un nœud et c'est le module DHT qui se charge de l'indexation et de la recherche du nœud responsable des flux vers la victime en question. En prenant l'exemple de l'IDS *S771*, nous voyons que la table concerne les victimes d'indexes proche numériquement de *ecee5f*. Ces victimes ne sont pas connectées à *S771*. La distribution logique des identifiants est calculée par la fonction HMAC et les résultats de cette fonction dépendent des adresses IP (des victimes et des nœuds du système de sécurité) et de la clé *K* indépendamment de la position sur le réseau des entités.



objectId
eced37
eced81
ecedad
ecee5f
ecee6a
ecee6f
...

nodeId (IDS)	***Fréquence***	***R[S]***	***R[A]***
11fcae (S24)	**100**	**10**	**150**
76adfe (S197)	**10 000**	**9 000**	**500**
eb55fe (S522)	**200**	**50**	**1000**
ecbc12 (S680)	**2 000**	**1 500**	**100**
fd134f (S822)	**11 000**	**8 000**	**50**
...			

Figure 5. Distribution et Indexation avec Pastry

Trois des nœuds de l'anneau ont détectés un trafic d'attaque ayant pour destination la machine *V780*. Ces nœuds sont *S197*, *S680* et *S822*. Les informations sur cette victime sont gérées par l'IDS *S771* déterminé par l'algorithme DHT. Dans la table de référence nous avons les identifiants des victimes gérées par le nœud courant et chaque *objectId* pointe vers une nouvelle table. La primitive qui permet l'envoi d'un message de publication vers un autre nœud est la primitive *put* qui nécessite la primitive *lookup* pour trouver le nœud responsable de chaque *objectId*. Dans notre exemple, la victime concernée est identifiée par *ecee5f*. La table d'information sauvegarde les différents flux ayant pour destination *V780* avec des informations concernant chacun de ces flux. Le *nodeId* de chaque IDS est le résultat de la fonction HMAC et pour chacun de ces identifiants sont représentés la fréquence, le taux *R[S]* (ratio de paquets SYN) et le taux *R[A]* (ratio de paquets ACK). Trois de ces entrées sont considérées comme des attaques : les IDSs ayant découvert ce trafic malveillant sont *S197, S680* et *S822*.

4. Conclusion et perspectives

L'objectif de cette thèse est d'étudier la conception d'une architecture pour les applications P2P contrôlées et gérées par les opérateurs de services. Le management du trafic P2P et la réduction du temps de téléchargement sont de véritables défis pour les opérateurs de services. Les clients s'attendent à recevoir le meilleur service avec des performances maximales alors que les opérateurs sont chargés de fournir les services avec la meilleurs QoS sans pour autant s'engager avec un coût supplémentaire.

Nous avons fait le choix de BitTorrent pour la composante du transport de messages. La première contribution est *hTracker* où l'on propose un algorithme de sélection de pairs au sein du même AS que le demandeur et pour laquelle nous définissons une sémantique permettant la correspondance entre *peerIds* des clients et ASs auxquels ils appartiennent . En deuxième lieu nous avons effectué une étude complète d'intégration de FEC à BitTorrent. Et enfin la généralisation du cloisonnement de trafic par une DHT sensible au contexte afin d'intégrer au mieux un moyen de regrouper les pairs selon un critère commun et d'optimiser ainsi la recherche de la ressource. Une architecture appelée SPOP englobe ces différentes contributions.

En terme de perspectives il est important que SPOP soit intégrée sur de réels nœuds d'un réseau afin d'étudier les traitements et overhead ajoutés au réseau par les différentes propositions. PlanetLAB [13] peut être une plateforme intéressante pour tester notre solution. Il serait également pratique de dresser un cahier de charges complet de certaines applications telles que l'IPTV et d'adapter SPOP à ce type d'applications en terme de QoS.

Références

[1] R. Saad, A. Serhrouchni and K. Chen, "SPOP: A Service Provider Oriented Peer-to-Peer architecture", à *SoftCOM 2010*, Septembre 2010 Split – Bol, Croatie.

[2] B. Cohen, "Incentives build robustness in BitTorrent", *First Workshop on Economics of Peer-to-Peer Systems*, Juin 2003, Berkeley, Etats-Unis.

[3] J. Li, J. Striblin, T. Gil, et al, "Comparing the performance of distributed hash tables under churn", *IPTPS'04* à *LNCSc 2004*, 26-27 Février 2004, San Diego, Californie, Etats-Unis

[4] H. Xie, A Krishnamurthy, YR Yang et A Silberschatz – "P4P: Proactive Provider Participation for P2P", *Tech. Rep. YALEU/DCS/TR-1377*, Mars 2007, Yale University, Etats-Unis.

[5] K. Pussep, S. Oechsner, O. Abboud, M. Kantor et B. Stiller, "Impact of Self-Organization in Peer-to-Peer Overlays on Underlay Utilization", *Fourth International Conference on Internet and Web Applications and Services (ICIW 2009)*, Mai 2009, Venise, Italie.

[6] R. Saad, A. Serhrouchni and K. Chen, "hTracker: Towards a Service Provider oriented Peer to Peer Architecture", à *NOTERE 2010*, Juin 2010, Tozeur, Tunisie.

[7] R. Saad, A. Serhrouchni, Y. Begriche et K. Chen, "Evaluating Forward Error Correction in BitTorrent Protocol", to appear in *Workshop on Wireless & Internet Services (WISe)* à *LCN 2010*, Octobre 2010, Denver, Etats-Unis.

[8] C. Gkantsidis et P. Rodriguez, "Network coding for large scale content distribution", à *INFOCOM 2005,* Miami, Etats-Unis.

[9] R. Bindal, P. Cao, W. Chan, J. Medved, G. Suwala, T. Bates, et A. Zhang, "Improving Traffic Locality in BitTorrent via Biased Neighbor Selection", *Proceedings IEEE* In*t'l Conf. Distributed Computi*ng *Systems (ICDCS '06J*, Juillet 2006.

[10] I. Papafili, S. Soursos, et G. D. Stamoulis, "Improvement of BitTorrent Performance and Inter-Domain Traffic by Inserting ISP-owned Peers", *6th International workshop on Internet Charg*ing *and QoS Technologies (ICQT'09)*, Mai 2009, Aachen, Allemagne.

[11] D. R. Choffnes et F. E. Bustamante, “Taming the Torrent: A practical approach to reducing cross-ISP traffic in P2P systems”, *Proceedings of ACM SIGCOMM 2008*, Août 2008.

[12] S. Ren, E. Tan, T. Luo, L. Guo, S. Chen, et X. Zhang, "TopBT: a topology-aware and infrastructure-independent BitTorrent client", *Proceedings of* IN*FOCOM'10*, San Diego, Californie.

[13] PlanetLab, http://www.planet-lab.org/

Abstract

The paradigms and architectures of overlay networks and especially Peer-to-Peer (P2P) networks became at the center of all types of large-scale applications achievements. The architectures of resilience, grid computing or distributed processing, file sharing, the distribution of all data types are increasingly based on overlay infrastructures.

It is necessary to incorporate a level of control over such applications. This control can serve as economic models, security, quality of service (QoS), and this in order to achieve various quality objectives. Such applications are would be deployed with an Service Provider as the principal control entity.

In current practice, the entities peers sharing resources are placed in a random way on a large physical network (IP). Furthermore, some applications including data distribution like time constraints ones are demanding on completion time and bandwidth. Using such an overlay network for such applications requires a special organization among peers. We propose the design of an overall architecture for the implementation of such applications on P2P platforms type.

In the P2P paradigm, it is possible to isolate three main components: the first is the proper service application with all its parameters, the second one is for the routing (or information lookup), the third one is for the data transport.

We are oriented toward architectures where networks are divided in different areas or autonomous systems controlled by Internet Service Providers (ISPs). This work consists in optimizing each P2P component to achieve best performances for different application requirements. These results may lead us to specific structures for three principal contributions. The first one has for objective to portion P2P traffic that is generalized by applying a Context-Aware algorithm where each portion or group of peers (peers in the same AS or sharing the same interest or capabilities) is based on a DHT. The second one is to speed up data transfer with Forward Error Correction. The third one is to integrate a Control/Management entity that manages the two previous contributions and that varies BitTorrent protocol parameters depending on the application service on top of the platform. These contributions have the following major goals: to minimize download completion time and the peering traffic between domains while keeping P2P robustness and interoperability.

We have made several large scale simulation studies that validate our propositions. In fact, we showed that portioning the traffic can present positive impacts especially when this portioning is complemented by an optimized routing Context-Aware algorithm. A deep study concerning FEC mechanism applied to BitTorrent protocol proved how much error correction can speed up data transfer in specific scenarios in homogeneous and heterogeneous networks. Finally, we grouped these contributions to propose a Service Provider Oriented P2P (P2P) architecture.

Keywords: BitTorrent, DHT, FEC, Locality, Overlay, Peer-to-Peer, Performance.

Contents

List of figures

List of tables

Chapter 1

Introduction

This chapter introduces this thesis by presenting the context of this research work in P2P networks. The problematic is exposed with the different solutions that exist in the domain. Then we cite the different contributions that are detailed in this thesis with their major advantages before concluding with the manuscript plan.

1.1 Context

In the last few years, the Internet has experienced a huge growth in terms of number of users and integration of new services. However, it has also shown some drawbacks. A model has emerged and fits in with various applications. This model experiencing a real success and that is at the center of many researches is the Peer-to-Peer (P2P) model.

The particularity of P2P networks is that they belong to overlay networks that provide some services by using a specific logical topology and some nodes in the basic Internet infrastructure. The nodes are still working normally: the routing and the packets transport are still working following the network layer protocol but a layer on the top of the basic infrastructure works with its own rules in a totally transparent manner. The principal advantages of Overlay networks and especially P2P networks are scalability because of their distribution and the deployment that does not need high investment. The load is distributed among all peers and their capacities and resources are shared in order to make all peers take advantage of this aggregate of storage capacity, CPU or bandwidth.

P2P networks are experiencing a vast expansion while various types of applications are proposed generally with negligible costs. Applications that have been ascertained in the P2P model are file sharing applications like Naspter [NAP] a few years ago, Emule [KUL&al05] or BitTorrent [BIT] nowadays, instant messaging, VoIP peer-to-peer application like Skype. The higher proportion of traffic exchanged inside and between current Service Providers is the P2P traffic. While music and film production houses have launched a war against illegal content providers and consumers, the ISP have difficulties controlling and managing the P2P traffic, even before differentiating in this traffic the legal one from the illegal one. In the academic research and industrial activity we can distinguish common axes. Principal ones are video streaming, resources managing, semantic overlay networks, resilient overlay networks, signaling traffic optimization, etc.

The current Internet, composed of many Autonomous Systems, is facing an important problem that is the future challenge of OEMs (Orginial Equipment Manufacturers) and ISPs. It is to integrate the undeniable control and management level that they need to mitigate the impact of P2P traffic in current networks. We know that some techniques try to integrate QoS and resource reservation to reach some concurrent objectives, especially in the case of real time applications that need a certain level of quality. The IP protocol is not enough to ensure that these objectives will be reached. In P2P networks, only a well specified and robust control level can provide the necessary environment in which an application that needs to follow some specific objectives can run efficiently.

It is essential to present each P2P component in order to analyze the place of each contribution and the impact of implementing or changing any of its levels. A P2P protocol is composed of three major components:

- The first one is the proper data transport level. In this level are defined the protocol messages specification, the peers exchange rules, etc.
- The second component concerns data lookup and the routing algorithm that helps find the requested resource: how the queries are redirected, is the system logically structured as with Distributed Hash Tables (DHT) [JLI&al04].

- Finally we have the service that is the closest to the end user describing the application in question and the objectives (QoS rules) depending on the agreement signed with the ISP (TLA/SLA).

The majority of ISPs nowadays do not hold on the components described previously. P2P traffic is in general generated by applications that are totally independent and separated from the ISP infrastructure. After the success of structured P2P algorithms like DHT that directly acted into the routing level of P2P model, some projects tried to go further into these concepts by specifying a way to interface an ISP and its Autonomous System with the P2P applications entities. ALTO project gives rise to the P4P architecture [XIE&al07].

Following the needs of P2P protocols in terms of QoS and performance guarantee for both customers and ISPs, we can conclude that a Control/Management level must be added to ISP network architecture and that this level must be an interface to every level of the P2P model. The targets are:

- Optimizing the transport level by proposing some mechanism to ensure maximal packet entropy and the minimum loss ratio.
- Guarantying at the routing level the completion for each query with the higher rate by minimizing the steps number and the signaling packets used by the routing algorithm (overhead).
- Elaborating specifications for each service and its needs to 1) control the traffic that is generated inside the ISP network and that is exchanged between the ISP and the others for some economic reasons; 2) guarantee that the environment is conducive to the service implemented in terms of delay, bandwidth and loss rate depending on each application (real time application or data sharing have not the same constraints).

1.2 Problems and solutions

The main advantage that makes the Internet have such a huge success is the absence of the traffic control that permits deliver data with a best effort service. However, if this is a good point for some applications, other binding applications cannot find the best conditions to provide an optimized service in the basic Internet infrastructure. In P2P networks that are on top of the basic IP infrastructure, clients download resources from other peers at the same time in parallel. This flexibility in choice made P2P ton being robust and scalable. In basic protocols, traffic control is naturally solved by TCP/IP stack protocols. However in different new protocols, especially P2P ones, many ways must be integrated to control and manage this traffic depending on the application.

Traffic that is inside an ISP is easier to control than inter domain traffic. When peers from an AS1 solicit peers from AS2, the first operator responsible of AS1 must pay the traffic that was drawn from AS2 whereas when AS1 peers are using traffic from their AS this problem is not encountered. This does not mean that this traffic is not considered as a cost for AS1. Indeed, this traffic also has a cost and must be fairly controlled among all the peers depending on each QoS class required by each of them. The cause of intra domain traffic overhead is due to two principal problems: 1) When the routing protocol used by the applications is not optimized some signaling messages can disrupt the traffic especially when the queries are flooded inside the network like in Gnutella for instance

[GNU01] 2) Some applications generated useless packets that must be controlled by the operator. For inter domain traffic the distributed aspect of P2P caused the generation of significant inter-domain traffic. In [KAR&al05] the authors show that for BitTorrent protocol, 50 to 90 % of local pieces are taken from other ASs. Some ISPs tend to violate peering agreements caused by traffic unbalance. Either the ISP control the traffic by estimating the needs depending on the applications or the P2P applications designers must adapt their traffic to networks variation. A compromise is here evocated concerning both the Service and the Routing level. In addition to providing traffic engineering at the Service level to control the traffic based on the application needs, it is important to adopt, at the routing level, protocols that minimize the message overhead and that provide the best performance in terms of resource lookup. The choice of DHT can be justified by their robustness and scalability. Many P2P applications propose the integration at the routing level of a structured algorithm like Chord [STO&al01], Pastry [ROW&al01], Tapestry [ZHA&al04], Kadmelia [MAY&al02], etc.

At the transport level the major problems are the following: 1) Some peers experience a lack of resources due to their weak capacities or because they are victim of the last block problem. 2) The queries completion and the resource download directly depend on the network properties at each instant and when the network encountered some difficulties this can affect the transport 3) Some security issues must be considered at the transport level. This security concerns data transport or even the signaling message transport at the routing level. For the first problem some mechanisms and algorithms are proposed to ensure the best entropy and data proliferation but without degrading the network performance. Error correction [RIZ97] and network coding [GKA&al05] are techniques proposed to speed up data access at the transport level. As mentioned previously, the second problem depends on the network environment. This implies that traffic engineering must be established upstream to provide the best network conditions for data transport. Finally some security mechanisms can be integrated against Eclipse [WAL&al02] and Sybil attacks [DOU&al02] and to ensure authentication and data integrity.

1.3 Contributions

The objective of this work is to provide a global architecture for Service Provider oriented P2P applications. We proposed to deal with each P2P level to integrate some mechanisms that can perform each level, based on large scale simulation measures and their analysis. The final goal of this step is to design a global architecture that ISPs can adopt to launch any application based on the P2P model in the best conditions.

The protocol BitTorrent is probably the most famous P2P data sharing protocol and can be regarded as transport method in its own right and not only as a data sharing application. It is used in many applications like Linux distribution installation, IPTV, etc. We decided to make BitTorrent as our transport protocol.

The first contribution of this thesis is the modification of BitTorrent peer selection policy. This proposition has already been proposed in [BIN&al06] and [PAP&al06] where the peers selected for the communication are mainly in the same AS as the peer that is at the origin of the request. This allows to conciliate the traffic and to advocate intra domain traffic at the expense of inter domain traffic that may cost a lot and require some new peer agreements. For the best of our knowledge, no formal specification has been proposed for peer membership in their AS. We decide to change the *peerId* specification in BitTorrent that was forged by the client with a free manner. In our contribution, we will show how we can use HMAC [HMC97] function to generate for each peer a semantic to their *peerId*. This has two main advantages: the first one is that the ISP will be able to control the peers to manage the generated traffic and the second one is that among the *peerId* creation, we will add a security level to the system. This contribution essentially deals with the Service level because it has the objective to facilitate the control of the peers and their needs. We will see in addition to that peer selection policy, that a method is proposed to vary the peers list depending on the launched application.

The second contribution concerns the transport level of P2P model. We propose to integrate Forward Error Correction (FEC) mechanism to BitTorrent but this integration requires a study. In fact, the impact of FEC varies depending on many parameters such as the peers' number and capacities, the file size, or the FEC ratio. An evaluation study based on large scale simulations has been undertaken to conclude in which scenario FEC has a positive influence on the general performance of the system based on the completion time metric. The simulations have been corroborated by a statistical test.

The third contribution of this Thesis is the proposition of a global architecture that implements both previous contribution and that can be a full model to integrate any application that can be managed by an ISP. We called this architecture the Service Provider oriented Peer-to-Peer (SPOP) architecture. In this contribution we will define and justify the choice the mechanisms that make part of the architecture. While at the transport level we adopted BitTorrent, we chose a Context-Aware DHT at the routing level and a QoS policy structure at the service level with the different policies that the ISP is ready to ensure (peer selection policy, varying the peers set size depending on the feedback, maintaining a threshold for the sliding window maximum delay in streaming applications, etc.). In this last point we define an entity called *hTracker* that is the control entity in an AS responsible of managing and controlling the system. This is the main control point that the ISP has to interact with its members.

To validate our architecture we propose a global security application where intrusion detection entities are distributed following our model and exchanging data information that are used by a service to react after these alerts.

1.4 Thesis plan

The rest of this Thesis is divided as the following. Section two presents a typology and analysis of Overlay and Peer-to-Peer networks, especially BitTorrent protocol that is the transport protocol on which our work is based. The third section describes the first contribution that concern the P2P traffic partitioning and the simulation results that validate our proposition. A FEC mechanism integration to BitTorrent is evaluated and the simulations are appreciated by a statistical test model to corroborate this other contribution in section four. The SPOP architecture that includes previous contributions and some other P2P service level specifications is presented in section cinq. Section six is a case study of a security application developed to validate the SPOP architecture with intrusion detection nodes.

Chapter 2

Overlay and Peer-to-Peer networks: Typology and Analysis

This chapter is a global presentation of Overlay networks and especially P2P networks. We focused on BitTorrent that is the protocol chosen to simulate and measure the efficiency of our contributions and that is implemented at the transport level of the global architecture that finalizes our work. We define BitTorrent specification, entities and algorithms with the different mathematical models that already exist. We detailed also a complete related work section with some performance results that motivate our choice.

2.1 Overlay networks

An overlay network is a virtual network based on one or more existing physical networks, so-called underlying or underlay networks, including the Internet (or sub-networks of the Internet) which is a good example.

The overlay network is formed by a subset of nodes in the underlay network, and a set of logical links between them, allowing a direct communication, while ignoring the topology and network protocols underlay. We call a node every network equipment, terminal or intermediate, of any kind, that acts like a router, client or server.

The overlay networks [LUA&al04] have always emerged when a new service that did not exist in the network had to be established. Thus every computer problem was able to be solved by a layer of indirection, or rather a redirection to a new virtual network that implements the solution. It is possible to design systems based on one or more overlay networks, serving as underlay networks and themselves working over a network like the Internet or the underlay network.

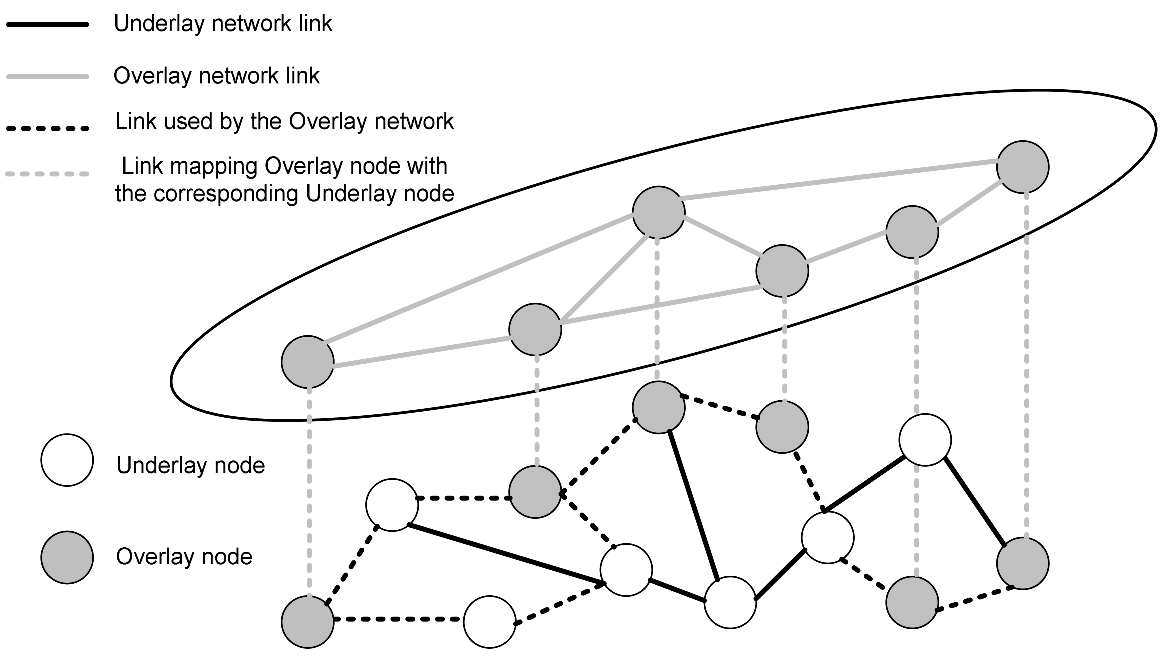

Underlay network link
Overlay network link
Link used by the Overlay network
Link mapping Overlay node with the corresponding Underlay node
Underlay node
Overlay node

Figure 2.1: Overlay network

When the Internet began, it was based on a research network running on top of the PSTN (Public Switched Telecommunications Network). A data application was running over the telephone public networks, and could be considered as an overlay network that added the packet-switched data communication functionality to the basic infrastructure of the PSTN. The Internet Service Providers (ISPs) were the principal actors of the Internet emergence. What we could call in the past the overlay network is our basic Internet network used by other applications and services (like multimedia data applications) to form new generation overlay networks.

As an example we can take the existing protocol Mobile IP [MOB02] where it is possible to physically redirect IP packets. In this system, the home agent is the router itself that has an interface in the same home network where the mobile is connected. This home agent integrates a header to the IP packets to permit their redirection with the same IP routing classical mechanisms.

IPSec [IPS08] technique is also an example of an overlay network constructed over the IP network for some security issues.

We can consider in the OSI model that for every level of the network architecture an overlay network is formed over the inferior level.

Table 2.1: Overlay network classification

Type	**Function**	**Examples**
Peer-to-Peer (P2P)	File sharing, Instant Messaging, IPTV, etc	Gnutella, Kazaa, BitTorrent, P2PLive, etc
Content Delivery Network (CDN)	Content caching to reduce transport delays	Akamai, Digital Island, FreeCast, Contendo, etc
Routing	Reducing routing and resilient delays	Resilient Overlay Network (RON)
Security	Security purposes	Virtual Private Networks (VPNs), FreeNet, IPSec, etc
Experimental and tests	Experiment new protocols and validation	PlanetLab
Other	Various	Multicast (MBone), IPv6 (6Bone), VoIP (Skype), Mobility (Mobile IP), SON, etc

An Overlay is a set of nodes deployed across the Internet that:

- provides a physical infrastructure to one or many applications (in best cases),
- are responsible for forwarding and handling application data in ways that are different from the basic Internet protocol,
- are operated in an organized and coherent way by end-users to provide a specific service,
- are not considered as part of the classical basic Internet infrastructure,
- have their proper routing protocol that is generally independent from the IP routing except for exceptions like topology-aware Overlay networks.

2.1.1 Overlay networks in current Internet networks

We defined an overlay network as existing over the basic Internet infrastructure and providing its own infrastructure and routing protocol. This permit us, based on the OSI model, to see an overlay network as a level in the middle between the IP network layers and the application one. Even if the OSI model provides a reference in the definition and the analysis of protocols, it cannot address all aspects of overlay networks. In fact, it is important to be interested on determining how overlay networks are evolving with the current Internet infrastructure and design.

The Internet is basically composed with hosts that are the end nodes and routers that forward the packets between the hosts. We generally see the Internet network as a group of connected routers with hosts that are connected at the border of this group. The applications are running on the hosts and are totally transparent for routers. The third components that can be defined are the servers that are responsible for providing a specific service for the profit of end users that are connected to the hosts.

In this specific vision we can say that this new possibility of integrating services to the basic Internet infrastructure can be considered as an overlay network using a physical network to provide some services to end users.

2.1.2 Overlay Functionality

It is important to study the functionality of overlay networks when it is necessary to understand their principal objectives. Let's consider the physical infrastructure as the grouping of protocols like TCP/IP or UDP/IP (and even routing protocols like BGP) that permit to form the core network necessary for any kind of applications that may be part of the Internet. The success of the Internet network is due to its capacity to implement any physical support (SONET, wireless, etc.) to provide any kind of applications (data, voice, video, gaming, etc.) with a generally distinguished interoperability.

However, the principal drawback recognized in the Internet is its best effort service. The needs in terms of QoS (Quality of Service) for applications like multicast or delay constraints application like VoIP, Video On Demand (VOD) or IPTV are a real challenge. This is how overlay networks tend to bring the best solutions for these issues that are essential for current applications in the Internet.

2.1.3 Overlay networks emergence

The reasons of the overlay networks emergence are various. The first reason is that overlay networks are born to fit some specific needs that are different from the basic Internet data forwarding needs, as we explained in the previous paragraph. An application can provide a virtual cloud composed of basic physical nodes with a logical topology. This topology would permit to connect an end user of a specific overlay network to communicate with other peers, following some constraints fixed by the Service Provider or the administrator of this overlay network.

Even if the best effort aspect of the current Internet is a major inconvenient, it presents the advantage of providing an easy way to deploy applications in a large number of nodes without any modifications of the infrastructure. However it is a real challenge to make a new application functioning without any problems in terms of scalability, fault tolerance and performance. The objectives of IPv6 were to integrate to IP protocol some mechanisms that could provide the layer that needed IPv4 protocol. However, we also know that this kind of change in the current Internet is a very difficult project. The first solutions that allow developers to test new protocols and applications were overlay networks. We can take the example of PlanetLab [LAB] that permits the integration and the validation of various kinds of applications using specific nodes over the whole Internet network.

2.1.4 Examples

MBone

The MBone [MBO], for Multicast Backbone, is a virtual network on the top of the IP basic infrastructure that permits the integration of Multicast technique over the Internet.

The principle is to logically redirect the traffic to a group of users sharing the same IP address. The MBone is formed by some multicast clouds interconnected by virtual tunnels with routers that have the multicast functionality. In France the MBone have a version called the FMBone.

6Bone

The 6Bone [FIN&al04] has been developed to test the implementation of IPV6 network. The principle is exactly the same as MBone. Some IPV6 networks are interconnected by IPV4 tunnels. The problem is that to apply IPV6 in the Internet requires the change of every routers used in the Internet today because the protocol specification is totally different. A D-day in IPv6, when the entire infrastructure will change, is a must and this solution is hard to imagine and very expensive actually.

RON (Resilient Overlay Network)

RON [AND&al01] is an overlay network that was created to establish a control of the different Internet links quality and that permits the detection of alternative routes when a service is interrupted whatever is the reason (fault, attack, etc). It allows the resilience and the restoration of a connection between nodes in a low time delay when some other protocols like BGP-4 can take several minutes.

SON (Semantic Overlay Network)

The problem of current overlay networks and especially P2P networks that are used for file sharing applications is that the management of the queries volume is difficult to maintain. It is a challenge to minimize the overhead created by an increase of useless requests. One solution is proposed by the Semantic Overlay Network [CRE&al02] that integrate a semantic to each file shared in a P2P network using Bloom Filters [BLO70] to verify the presence of specific information in the file. This semantic can concern the description, the content or the queries history of the file. This semantic signature is integrated to the routing tables that allow performing better resource localization.

CDNs (Content Delivery Networks)

CDN is a network that is formed by several cache servers that provide an optimized management dispositive of high data volume communication flows and permitting the data transport in the best conditions. The major objective is to minimize the bottlenecks in the network. Most of the CDN clients are Service Providers that need to propose reliability and availability in their services. Akamai [AKA] and Cisco [CIS] are the main interested companies in this domain.

OverQoS

OverQoS [SUB&al04] is an architecture that is based on an overlay network that has for objective to enhance the best effort service of the Internet. A traffic aggregate observe the loss rate and limit it using a virtual link called CLVL (Controlled Loss Virtual

Link). The different services provided can be smoothing packet losses, prioritizing packets within an aggregate, guaranty bandwidth and statistical losses.

PlanetLab

PlanetLab [LAB] is an overlay network that connects some specific nodes (around 900) scattered around the world and that was developed as a research test platform in network and distributed systems domains. For each project a slice is created generating the reservation of a virtual network corresponding to a part of the PlanetLab network. Only enterprises and academic institutes can have access to the nodes even if some free services like OpenDHT [RHE&al05] have been launched for public use.

2.2 Peer-to-Peer networks

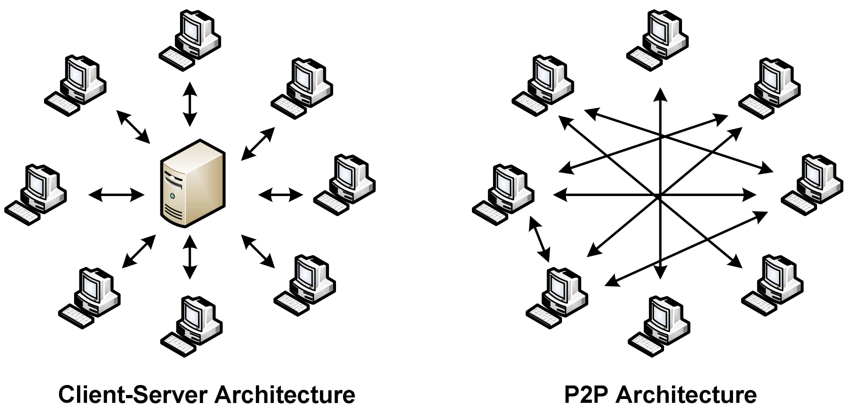


Figure 2.2: Peer-to-Peer model versus Client-Server model [GNT]

In this Thesis, the examples of overlay networks we are interested in are P2P (Peer-to-Peer) networks.

A P2P network is any distributed network composed of nodes that share their resources with other network participants without any central server. A peer is both client and server, supplier and consumer of these resources. This model differs from the classical Client-Server model where only servers provide the service while clients use it. The Figure 2.2 illustrates the difference between both models.

The definition of the word "peer" reminds us the notion of equality in terms of role or function. In fact, in a pure P2P system all peers have the same functions and roles and no one has a higher hierarchical status. In computer networks a peer is equal to a host or a node in a system.

Peer-to-peer is usually associated to illicit file download applications in the Internet, and this is due to the impact that had or still have some applications like Napster [NAP], Kazaa [KAZ] or Emule [KUL&al05]. However, Peer-to-Peer is not only characterized by this kind of use, while sharing content is not prohibited by law if this content is not protected by copyrights.

The *Intel P2P Working Group* defines P2P as the "the sharing of computer resources and services by direct exchange between systems" [INT]. For the SETI@home [KOR01] project members "P2P projects that do not involve communication are inverted Client-

Server", which means that nodes at the edge provide the resources and those at the core coordinate them.

2.2.1 P2P objectives

The P2P have as major objectives, like all information systems, to offer applications and services that are satisfying users' needs. It is important to define and analyze these needs in our work while it is our goal to provide a platform based on the P2P model that can guarantee the best performance and usage conditions for users. We detail in the following the most important points that form the requirements and specifications of a P2P system:

Costs Sharing/Reduction: Centralized systems that serve important number of clients represent the major costs of a system. When this cost is too important, P2P architecture can help to distribute this cost among peers. Taking the example of file sharing application, the storage costs can be distributed among all clients while keeping an essential index to maintain the sharing. This sharing can work thanks to the use of unused resources aggregation (like on SETI@home [KOR01] project). This permits to reduce the most expensive system components. While every peer tends to be autonomous, it is important to keep consistent and balanced costs among all nodes.

Resources aggregation and interoperability: a decentralized approach tends naturally towards resources aggregation. Every node in a P2P system keeps some resources like CPU power or storage capacity. Applications need those resources in important quantities, like Grid Computing or file distribution systems. A distributed system like SETI@home [KOR01] is a good example. When thousands of computer resources are aggregated, the systems are capable of calculating some very complex calculation function. Even in file sharing applications like in Gnutella [GNU01] or BitTorrent [BIT], resources aggregation is the basis of their success. In this case, the principal resources that are aggregated are storage capacity and bandwidth. These resources are available for the community to respectively save some information and transport it quickly. We know that all nodes are not necessarily homogeneous in many aspects. Maintaining interoperability between systems is important to allow communications between heterogeneous peers.

Reliability/Scalablity/Extensibility: P2P networks are devoid of any control of autonomous nodes and this motivates designers to increase the reliability and the extensibility of those systems. Many innovations are introduced to fulfill these objectives. It is essentially at the lookup and routing level that some new algorithms are integrated to previous generic systems. Examples are decentralized structured P2P applications like Chord [STO&al01], Pastry [ROW&al01], CAN [RAT02], etc. They are generally based on a Distributed Hash Table (DHT) that index and distribute the resource among all nodes in a logical deterministic way that optimizes the resource lookup in a P2P system. When it is a question of reliability, it is important to observe the evolution of the system while the peers' number is increasing. A reliable and scalable system must be unchanged even if many peers have joined it. The availability here is an aspect that is essential to quantify the performance of any P2P system.

Autonomy: in most of the cases, users in a distributed system are not forced to depend directly on a central server. Instead they prefer to store most of the information and to keep the treatment done locally. P2P systems support this autonomy level because they are built to let nodes execute the whole work generated by the end user. In fact, in file sharing protocols, the users can exchange data without depending on a central point. The drawback is that without this control, the proliferation of illegal data exchange is encouraged.

Anonymity: Anonymity notions are directly linked to autonomy aspect described previously. A user can prefer that its ISP and anyone else know its activity in P2P systems. Actually some new concepts must be applied to allow ISPs to establish a real survey on peers' activity and traffic generate in its Autonomous System and passing by it. Hiding will tend to be prohibited since it would be easy for a user to break the rules. It is generally difficult in a P2P system to ensure a total anonymity since servers have to know clients and to connect them together in some applications. Freenet [CLA&al00] is the application that provides the best anonymity but it has not been a great success.

Dynamism: P2P systems integrate the fact that their computer environment is extremely dynamic. In fact, the resources like all nodes in the system can join and leave in a random and continuous manner. When an application has to support a dynamic environment, the P2P approach is implicit. File sharing application has to manage the fact that the data is scattered in different nodes that are not connected at the same time.

2.2.2 Architectures

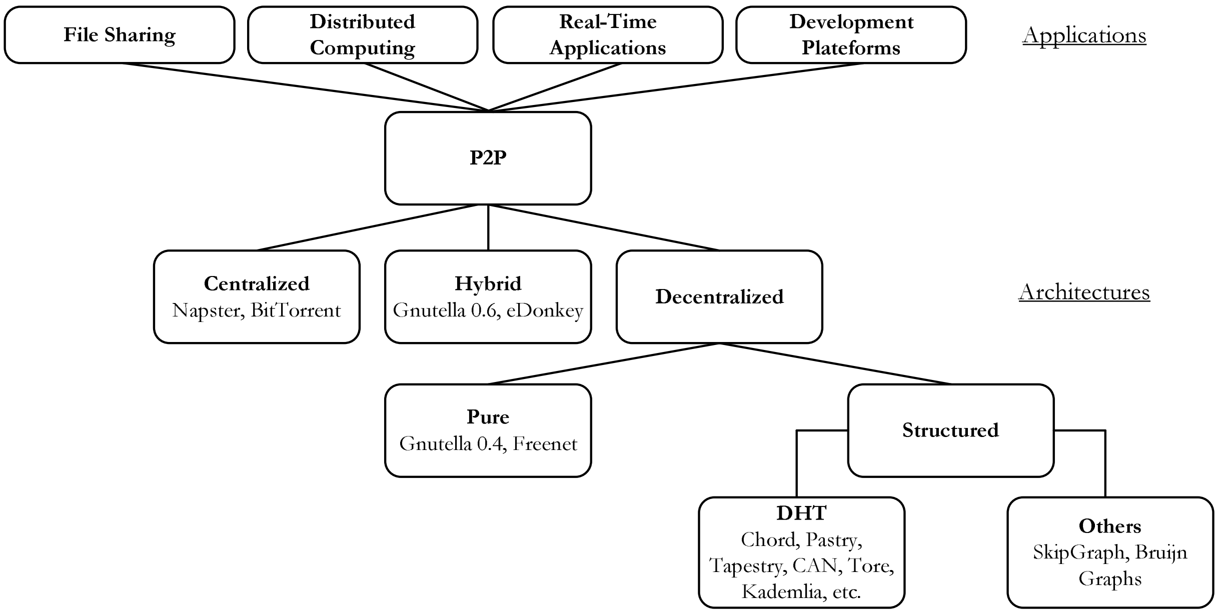


Figure 2.3: Peer-to-Peer applications and architectures

In Figure 2.3 we summarize the different P2P architectures and applications. This list is non exhaustive but references the major elements of P2P networks.

2.2.2.1 Centralized

The most famous P2P network was probably Napster. Its originality lies in the fact that it uses a centralized architecture. This concept has contributed to its success but also to its loss. On paper, such a device is currently the most comfortable solution for sharing files in one community (music, DVD, etc.). However, in reality, this architecture requires such a resource investment that services are rarely of good quality. Either they are saturated, or they are limited in terms of simultaneous users allowed. Concretely, in any centralized architecture, a server is responsible for directly connecting users between them. The value of this technique lies in the centralized indexing of all directories and files shared by subscribers on the network. In general, the update of the database is doing in real time, when a new user connects or leaves the service. It works with the customer as with a conventional search engine: a query is started by entering a keyword. The client get a list of users currently connected to the service and whose files are shared to the search term. Therefore, simply click on one of the titles link to connect directly to the corresponding machine and begin the transfer. Under these conditions no files are stored in the server.

The main advantage of a centralized list of users and files or resources that the peers share is the speed of response from the server. Since it is dedicated to handling queries and data referencing, it is efficient and can respond quickly as it has all the information locally to check if at any user a particular resource can be found. We also note that it simplifies the use because the user has no server to choose as in the case of the Hybrid architecture presented in next section.

The limitations of this system are numerous. That is the reason why the majority of consumer applications do not function in this model. It requires a large investment for the servers on which the entire burden rests. This is a weak point while if the server goes on, all users are deprived of their resources.

To solve the problems of robustness and improve the quality of connection with the server, the central server of the centralized architecture is replaced by a ring server. This prevents the collapse of the network if a failure occurs on a server, because there is always a valid connection to the servers.

Furthermore, the use of multiple servers permits a better distribution of the connection requests and it can therefore limit the drop in bandwidth. Each server can have access to customer information connected to others. Access to shared data is completely transparent to users. The solution improves service availability and robustness of the architecture but it is an important investment.

We decided to put BitTorrent in this architecture category because of the *Tracker* entity while we can put it as apart from all usual classification because of its specificity in terms of functionalities and mechanisms.

2.2.2.2 Decentralized

2.2.2.2.1 Totally decentralized

This architecture is based on network nodes, rather than on a central server. The system for exchanging files is completely decentralized. The software user connects to the computer via the Internet to one or more other users, thereby creating a network. In this way, each user is available to the entire community. This model is more difficult to use than the first one because end users need to find a starting node on the network to connect. Otherwise, the network cannot be used and a peer will have difficulty find another one.

The principle is: a computer "A", with a specific program (that both acts as client and server both), connects to a computer "B" also equipped with this program. "A" and tells him he is "alive". "B" relays this information to all computers to which it is connected, "C", "D", "E" and "F", etc. They will relay the information to turn to computers they are connected, and so immediately with all computers on the network.
Once A is found "alive" by the other members of the peer network, it searches the content of interest in the shared directories of other network members.
The request will be sent to all members of the network, starting with "B", then to all other members. If one computer has this file, it transmits the information to "A". This may well open a direct connection to that computer and download the file.
This model, being decentralized, is much more robust than a centralized model since it is not dependent on the server, potential point of failure of a network. If a user disconnects from the network, the application may be continued to other computers.
Fully decentralized architecture presents some drawbacks. The system is easily overloaded by requests relayed (broadcast) that are multiplied with the number of connected peers. This can reduce the burden carried by the network. The latter is more difficult to control and to administer as it has no central node and must be, if it needs configuration, configure all clients.

It is important to note that on a public network such as Gnutella, for example, the responsibility is fully shared. It can be worn by the head of the software because it has no control over the content shared on the network (which may well be legal to be illegal). This shared responsibility has therefore introduced a concept of community.

One consequence of this architecture and the convergence time of such a network is the slow lookup that requires a high number of messages that is proportional to the number of network elements (and exponentially with depth lookup). However, optimized protocols were established, based on distributed hash table, to conduct lookup in a number of messages increasing logarithmically with the number of network elements, such as CAN, Chord, Freenet [CLA&al00], GNUnet, Tapestry, Pastry, and Symphony. Those protocols follow the decentralized but structured architecture.

2.2.2.2.2 Structured Decentralized model

A decentralized P2P network can be structured when an algorithm controls the logical topology of the network and the way the resource is found by each peer follow this algorithm. The main advantage of this type of architecture is that the peers do not have to flood the network to ask for the resource. This avoids the overhead on the network. The lookup can also be optimized while the resource and the peers are indexed logically following an efficient algorithm. Most of the structured algorithms are based on Distributed Hash Tables (DHT). We will present the most famous and interesting algorithms depending on the type of logical architecture they are based on. These protocols are considered as second generation P2P protocols.

Ring topology

Chord

Chord is the most famous DHT algorithm and the most simple to understand and to implement. It is based on a ring topology unidirectional (clockwise) and like all DHT it stores key-value pairs by indexing keys among all nodes of a network. The objective of Chord is to specify the way keys are assigned to nodes and how a node request for the value of a given key by locating the node responsible of this specific key.

In the Chord ring we have 2^m nodes. The circle can have IDs/keys ranging from 0 to $2^m - 1$. Each node is responsible of some keys and the overall information database is uniformly distributed and in the same identifier space due to what is called the consistent hashing. Each node has a successor and a predecessor. Both keys and nodes are assigned an *m*-bit identifier. For nodes, this is a hash of the node's IP address. For keys, this identifier is a hash of a keyword, such as a file name. A logical ring with positions numbered 0 to 2^{m-1} is formed among nodes. Key *k* is assigned to node successor(*k*), which is the node whose identifier is equal to or follows the identifier of *k*. If there are *N* nodes and *K* keys, then each node is responsible for roughly *K*/*N* keys. When a new node joins or leaves the network, responsibility for *O*(*K*/*N*) keys changes hands. If each node knows only the location of its successor, a linear search over the network could locate a particular key. Chord requires each node to keep a "finger table" containing up to *m* entries. The i^{th} entry of node *n* will contain the address of successor($n + 2^i$). With such a finger table, the number of nodes that must be contacted to find a successor in an *N*-node network is *O*(log*N*). In Figure 2.4 for instance K33 and K35 are managed by N39.

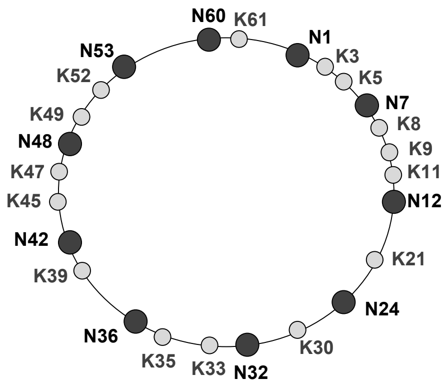


Figure 2.4: Chord algorithm

Pastry

This implementation also consists of connected nodes using a P2P network but that follows a structured Plaxton based algorithm [PLX&al97] for resource lookup. When a node joins the system, it is assigned a unique ID, a 160-bit value. In this 160-bit ID space the distance between two nodes A and B must be *min(A-B mod N, B-A mod N)*. In Pastry the logical topology is also a ring but in this DHT implementation the ring is bidirectional. Thus, the distance between nodes is a minimum distance along a circle from one node to another. The main difference between Pastry and Chord is that Pastry presents a bidirectional ring topology.

The principal objective of Pastry is to manage communication between nodes. The algorithm is demonstrated in the following Figure 2.5. To send a message from a node with node ID 1084 to the address 0128, Pastry first sends the message to a node that has a matching prefix of one digit. The recipient checks the message. If the recipient has its node ID closest to the destination address that means the message is addressed to it. If not, the recipient forwards the message to a node that matches the destination prefix with two digits. And so on the message arrives at its destination in log (N) hops. A node sending message to address 0128 first sends it to any node that has first part of the address 0. Then the message is sent to a node that has prefix 01, then 0128. Since the last node ID digit cannot be matched by any node, the node 0122 processes the message as the node 0128 has the closest node ID. In Pastry each node has a routing table, a leaf set and a neighborhood set. The number of rows in the routing table is equal to the number of node ID digits and the number of rows is one less than the number of distinct digits in base 16 (if this base is chose). In the routing table, the *n^th^* row nodes have a length matching prefix with the current node ID. The leaf set keeps nodes addresses of nodes that have close IDs to the current node. This table helps routing. The neighborhood set has addresses of nodes that are physically close to this node. This table is used for routing updates.

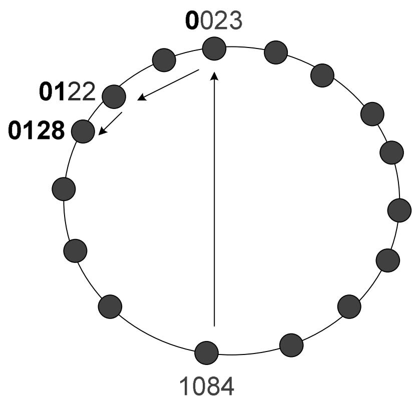


Figure 2.5: Pastry algorithm

Tapestry [ZHA&al04]

In Tapestry the hash key is 160 bits long and the object identification is defined by a GUID (Global Unique Identifier). A recovery graph is created to avoid faults with some replicate objects inserted in the path towards the top of this graph. Different keys are used in Tapestry and the routing table contains nodes that are closer numerically as in Pastry algorithm. Tapestry takes into consideration the physical distance in the network (RTT) but comparably to Pastry which is also a Plaxton [PLX&al97] based algorithm, Tapestry routing is based on the identification suffix and not the prefix

Tore: CAN [RAT02]

The Content Addressable Network is a distributed P2P architecture based on a DHT designed to be scalable, fault-tolerant and self-organizing. The principle is a virtual multi-dimensional Cartesian coordinate space. This space is a virtual logical address independent of the physical layer. The partitioning is done such as every node in the system possesses at least one distinct zone within the overall space. This architecture was one of the first DHT proposed for P2P systems.

Butterfly: Viceroy [MAL&al02]

Viceroy is a routing algorithm based on a Butterfly model. This protocol is an extension that ameliorates Chord algorithm with a multi level dimension topology. Each peer in Viceroy maintains two pointers for the successor and the predecessor at the same level, two pointers for the successor and the predecessor at the reference ring that is usually the level 1 ring, and three pointers for nodes in the right and in the left in lower levels and for the closest node in the lower level. The lookup is done step by step from the node level to the higher levels until the request reach a success.

2.2.2.3 Hybrid model

This model is also called the Super-Node or Super-Peer model. A Super-Peer acts as a server or intermediate node for a group of peers. In this kind of model, this special peer is chosen because it presents some special capabilities (High performance peer in terms of memory and bandwidth) and constitutes the central entity as in centralized model but only for a group of peers. This model is designed to take profit from the advantages of the two types of networks which are the centralized and the decentralized ones. Indeed, this architecture structure reduces the number of connections on each server, thereby avoiding the problems of bandwidth.

On the other hand, the network server uses a mechanism based on decentralized networks to maintain a client directory and a file index based on information from other servers. A server can offer all information contained on the network to any customer. The network is no longer polluted by broadcast frames but the counterpart is that anonymity is no longer assured. Examples of Hybrid architecture protocols are Gnutella version 0.6 [GNU01] and Freenet [CLA&al00]. Some literature puts BitTorrent in this category while we think that BitTorrent is more a centralized architecture protocol because of the *Tracker* entity. In fact, BitTorrent can be apart from usual classifications.

2.2.3 Applications

Distributed computing is particularly used today in finance and biotechnology ; for instance in some financial institutions such as banks, where credit institutions must implement extremely complex simulations for calculations of the market. In the past, financial applications were usually performed during the night. Today, these applications are more real time sensitive and need to be executed on time. Only certain large institutions are able to assume the cost of setting up a system powerful enough to support these simulations. An alternative for smaller structures is the use of P2P systems can use all the computing resources for the calculation of these simulations.

2.2.3.1 File Sharing

The storage and sharing content are the areas where P2P technology has been and still is the most used. With the emergence of some various media files sizes, we needed a way to share these files without being hindered by the limited bandwidth of the time. Distributed storage applications that concern information based on P2P technology offer the following features:

- **Spaces for file sharing:** systems such as Freenet [CLA&al00] potentially enable users unlimited storage capacity thanks to the redundancy management. A file is stored on one or more nodes in the community but is available to all members of the community.
- **High availability of storage space:** duplication and redundancy in some projects can provide a virtually anytime availability of stored files and protection of sensitive files.

- **Anonymity:** Some P2P applications like Freenet [CLA&al00] ensure anonymity of authors and readers of the network.

2.2.3.2 Collaborative Computing

P2P collaborative applications are designed to enable collaboration between users at the application level. The nature of P2P technologies enables effective collaboration between users. The applications in this category include instant messaging, online games and shared applications that can be used in professional environments, educational and personal use.

In collaborative applications peers form a group and start a given task. One group may include two peers who work directly or a wider band. When a change occurs in a peer (the peer sends a chat message for example), an event is generated and sent to the group. At the application level, each peer's interface is updated accordingly. There are many technical challenges that make difficult the implementation of such systems. Like other P2P systems, the location of other peers is a challenge for collaborative systems. Many applications, such as rely on centralized directories that list all peers online. To form a new group, a peer consults the directory and selects the other peers. Other systems, such as Microsoft NetMeeting, can require peers to add members based on their IP address.

Fault tolerance is another challenge for these systems. In the shared applications, messages need to be transmitted with high reliability so that all peers have the same vision of the information. In some cases, the scheduling of messages is also very important. If non-P2P applications are not as strict on these points, P2P applications will need to be rigorous. Indeed, the lack of autonomy of central control may curb if the whole system is not very reliable. One of the solutions used in most P2P systems is to stack the sent messages and messages to be issued.

Finally, real-time constraints are probably the most complicated aspect in collaborative task implementations. Users in such environments, felt directly on time. Unfortunately, in this case, most of the network is involved, rather than the P2P system itself.

2.2.3.3 Real-Time Applications

Real-Time applications are one of the biggest challenges for P2P networks. Current researches are focusing on how to distribute streaming data on P2P with the best performance and by respecting ISP Peering strategies. Streaming engaged high traffic volume that is difficult to manage by ISPs when they do not have control on the applications launched in their network. PPLive [HEI&al06], PeerCast [JIA&al08] or PPStream [WEI&al09] are examples of IPTV P2P applications that are usually used in Asian countries and especially in China. VoD is also a killer application and is generally based on Content Distribution Networks (CDNs) like Akamai [AKA].

2.2.3.4 Development Platforms

Operating systems are becoming less and less essential to run applications. Middleware solutions, such as Java virtual machines or web browsers, are extremely interesting for both end users and developers. This suggests that future systems will depend more and more platforms will be the common denominators between users and the services they want to access.

There are several candidates in competition to become the future platform P2P. The JXTA platform [GON02] and .Net are the two industry heavyweights.

2.3 The BitTorent protocol

The challenges of current P2P applications are to permit sharing of high volume file to a large number of peers. In file sharing communities' efficiency, robustness and scalability are guidelines of file distribution. BitTorrent protocol distinguishes itself from other protocols. It is probably the most popular file-sharing protocol that is currently used in the Internet. It represents more than 60% of the P2P traffic [IPO07]) which is more than 30% of the Internet traffic [CAC05]. Many applications are using BitTorrent-as protocols to deliver various content types. The first particularity of BitTorrent is that it first focuses on how to quickly deliver the content by increasing the pieces entropy while other P2P applications are first interested in the localization of this content. BitTorrent is known for its performances in terms of fast download speed while it is exploited by Linux operating systems that propose a BitTorrent download option to retrieve their distributions. Like in Emule [KUL&al05], BitTorrent uses the multisourcing principle. Data is divided in two levels presenting a good granularity. Technically BitTorrent is different from other P2P protocols: no search engine is integrated to BitTorrent clients and a metadata file is downloaded to have all information concerning how to retrieve the resource. In BitTorrent, integrity is implicit because every piece is verified thanks to SHA1 hashing value for each piece and if a file is corrupted, that means it is at the origin.

Comparing to other P2P applications, the BitTorrent reciprocity and incentive mechanisms create an intuitive virtuous circle during file sharing: data proliferation is fast and peers participate early. A swarm that contains many peers and where upload capacities are important is the best environment for high quality data transport for BitTorrent protocol.

BitTorrent has been imagined and developed by Bram Cohen [COH03] who originally wanted a robust system with high efficiency with the maximal resource utilization. It integrates an equity policy into the protocol that permits only generous users to receive pieces abundantly. All BitTorrent clients are capable to prepare, request and give any resource through a network using BitTorrent protocol: text files, Audio, Video, VoIP and IPTV applications, etc.

Many clients have been developed in all languages and for every platforms and BitTorrent is integrated to web browsers (Opera [OPE]) or other P2P clients (LimeWire [LIM] or Emule [KUL&al05]).

2.3.1 Architecture

BitTorrent is composed of 3 major entities: the *Tracker* that keeps information on peers concerning a specific *Torrent* file; the .torrent that identifies file that have to be downloaded or that have been downloading; and finally the peers that are sharing the resource. BitTorrent, in opposition to other P2P applications, does not have a search engine integrated to the client. The customer has to download the .torrent file corresponding to the resource(s) (file or multiple files) it wants to retrieve. An optional contribution has been added to some clients: the DHT Kademlia [MAY&al02] algorithm permits to distribute data and alleviates that request are sent to a unique *Tracker*. We talk about Trackerless BitTorrent solution.

The flows that characterize a file download between different peers is also called a *Torrent* while the .torrent file is the metadata file containing information about the *Tracker* that has to be contacted and about the file(s) that composed the resource. Following are the network elements that can participate into a BitTorrent network:

- A *Seed* has the entire file and is also uploading parts of it.
- A *Leech* is a peer that is downloading data but that does not have the entire file yet. It can begin to share information with other *Leeches* with the piece it already has before becoming a *Seed* and when a *Leech* becomes a *Seed*, it can stay in the network swarm to share the resource it has entirely.
- The *Tracker* is the entity that lists the peers (*Leeches* or *Seeds*) and the data volume that each one contains for a specific *Torrent* file (for a specific resource). This entity also maintains some statistics on the peers and the *Torrent* by regularly receiving reports from the peers.

A web site hosts *Torrent* files that are requested by the peers to contact the *Tracker* and download data.

A resource in BitTorrent is divided in pieces that have the same size for a specific resource but this size can vary depending on the resource size. The bigger the resource is, the bigger the piece size is. Each piece is also divided in blocks that also have the same size for a specific resource. The default values are 256 kB for the piece size and 16 kB for the block size. The block is the transfer unit but in general, the protocol only considers the piece level granularity because in BitTorrent, a policy called the *Strict Policy* forces a piece to be completely downloaded (all blocks of the current piece) before another piece is downloaded.

Leech states: a *Leech* can be *Interested* (if it is *Interested* by a piece in a distant *Leech*) or *Not Interested*. It can also be *Choked* or *Unchoked*. A peer B chokes a peer A (that is *Choked*) if the peer B decides to block its upload flow towards A (in the case where B accepts to upload data to A, we say that A is *Unchoked* by peer B).

Connection: BitTorrent is based on TCP protocol. A peer can open a maximum of 40 connections in 80 available.

Figure 2.6 represents the architecture entities. We can note that the peer is a *Leech* because it is in the *Interested* state for a resource and that it does not download a piece yet. The first step is the .torrent file download from a Web site that corresponds to the desired resource. This .torrent file downloaded, the peer retrieve from the .torrent the *Tracker* URL responsible of the resource.

The *Tracker* returns a list of peers called the *Peers Set* containing *Leeches* and *Seeds* that the *Leech* downloading will contact to retrieve data pieces. An algorithm is responsible of coordinating the peers selection and pieces selections is also based on some policies that we will detail.

We represent in the following figure an arbitrary scenario to understand the different steps when a *Leech* connects to BitTorrent and wants to download a file. We symbolize the order of contact from this *Leech* to other entities and we do not integrate the policies in this scenario yet.

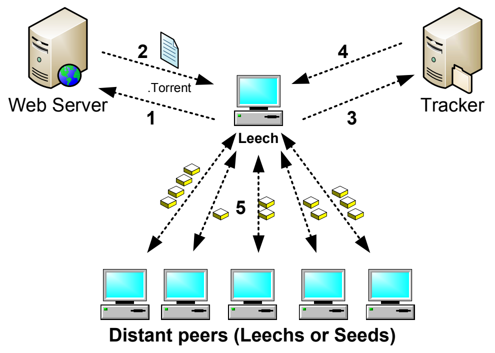


Figure 2.6: BitTorrent Architecture entities

Distant peers that are peers of the P*eers Set* depend directly on the peers' selection algorithm. An *Interested Leech* can contact peers from the *Peers Set* but these peers are the one to decide if they accept to upload pieces to the requesting *Leech*. The following details the different algorithms in BitTorrent.

2.3.2 Functioning and protocol specification

2.3.2.1 How does BitTorrent works

Peers selection algorithm: *Choke Algorithm*

In BitTorrent, no resource reservation is done but every peer is responsible of maximizing its download ratio. This is done by managing to which *Leech* a peer will upload its pieces or not following a policy called *Tit for Tat*. The principle of this policy is to install reciprocity between peers. The more a peer will give to others, the more it will have a chance to receive. The distant peer is selected only if it respects that principle and accepts that others download from its resource pieces. A distant peer that chooses to collaborate is *Unchoked* and the one that does not is *Choked* (To Choker is to block temporarily to send some pieces. A principle penalizes *Free-riders*, that means those that do not participate and only peers that are respecting the game and sending files will be *Unchoked*.

A peer sends to a maximum of five *Leeches* that are *Unchoked*. However these *Leeches* must first be in the *Interested* state. Among these 5 peers, 4 are chosen depending on their Downloading Rate if the local peer is a *Leech*. When it becomes a *Seed*, this choice depends on the Uploading Rate. The cycle changes every 10 seconds and the choice of these peers may also depend on the network configuration and the peers' dynamicity. The rate calculation for each peer is done every 20 seconds. This calculation was done depending on the data volume transfer at long term but this does not exactly correspond to reality because the Bandwidth can be quickly by the availability change for a specific resource.

To avoid the situation where resources are lost during fast *Choke* and *Unchoke*, peers calculate again the peer to Choke every 10 seconds. This duration is just enough for new connections to be opened by TCP and it permits to maximize transfer capacities.

To let a chance for *Leeches* that do not have enough bandwidth to be chosen as *unchoked* peers, a 5th peer called the *Optimistic Unchoked Peer* or *altruist peer* is chosen randomly.

BitTorrent *Tracker* HTTP/HTTPS Protocol.

A peer A downloads the *Torrent film*.torrent from *www.piratebay.org*. It must contact the responsible *Tracker* for this *Torrent* that would be able to inform it and send it back the list of peers to contact to launch the download process.

The peer A sends to the *Tracker* information that are necessary to keep localizing the peer (IP address and port number), its peerId, its state (*started, completed* or *stopped)*, the Uploaded and downloaded data volume in bytes (for the first contact these fields are naturally equal to zero).

It is also important that peer A sends regularly metrics to the *Tracker* allowing to keep some statistics on the torrent.

The *Tracker* makes a random choice of 50 peers (default value that can be changed but that must be between 20 and 80), and sends this list called *Peers Set* to the peer A.

It specifies to A a time interval during which it refuses to be questioned again (in general this interval is fixed to 15 minutes). This time must be longer than the timeout on the http connection. It also informs the local peer A the number of *Seeds* and *Leeches*.

Peer Wire Protocol

At the very beginning of the process, a new peer that joins the *Tracker* and that wants to download the specific file, does not have any available piece at its disposal. The Tit for Tat reciprocity that we previously mentioned requires that a peer has something to share to be able to receive very quickly. For the first piece to be downloaded, peer A randomly chose a peer itself. This distant peer B that peer A chose sends a *have* message to give its availability. Peer A sends an *Interested* message for a random piece. This policy is called the *Random Policy*. The sending peer B can choose to *Unchoke* the peer A or not, depending on peer A's capacities or if it has the chance to be chosen as Optimistic Peer. If peer B *Unchoke* peer A, it accepts to send data to local peer A. The set of peers that chose to *Unchoke* peer A is a subset of the set called the *Neighbor Set* which is the set including to the peers to which the peer A also sends. However, in the primary stage, peer A is a new *Leech* and does not have anything to send. The peer B *Active Peers Set* is the peers set that peer B chose to *Unchoke*.

Let's note that a Strict Policy is applied in BitTorrent at the Blocks level. Blocks of a piece are first downloaded before another piece is downloaded. In fact, we download blocks in the right order from the source. In the case there is a fault for any reason whatsoever (peer B decided for example to *choke* peer A), depending on the implementation the peer A can toggle download to another peer and retrieve remaining blocks or to drop blocks that are already retrieved, to chose to download the same piece from another distant peer.

Random Policy detailed previously for the first piece can be applied to four or five pieces and this in order to quickly obtain some pieces to share to have more chance to download afterwards.

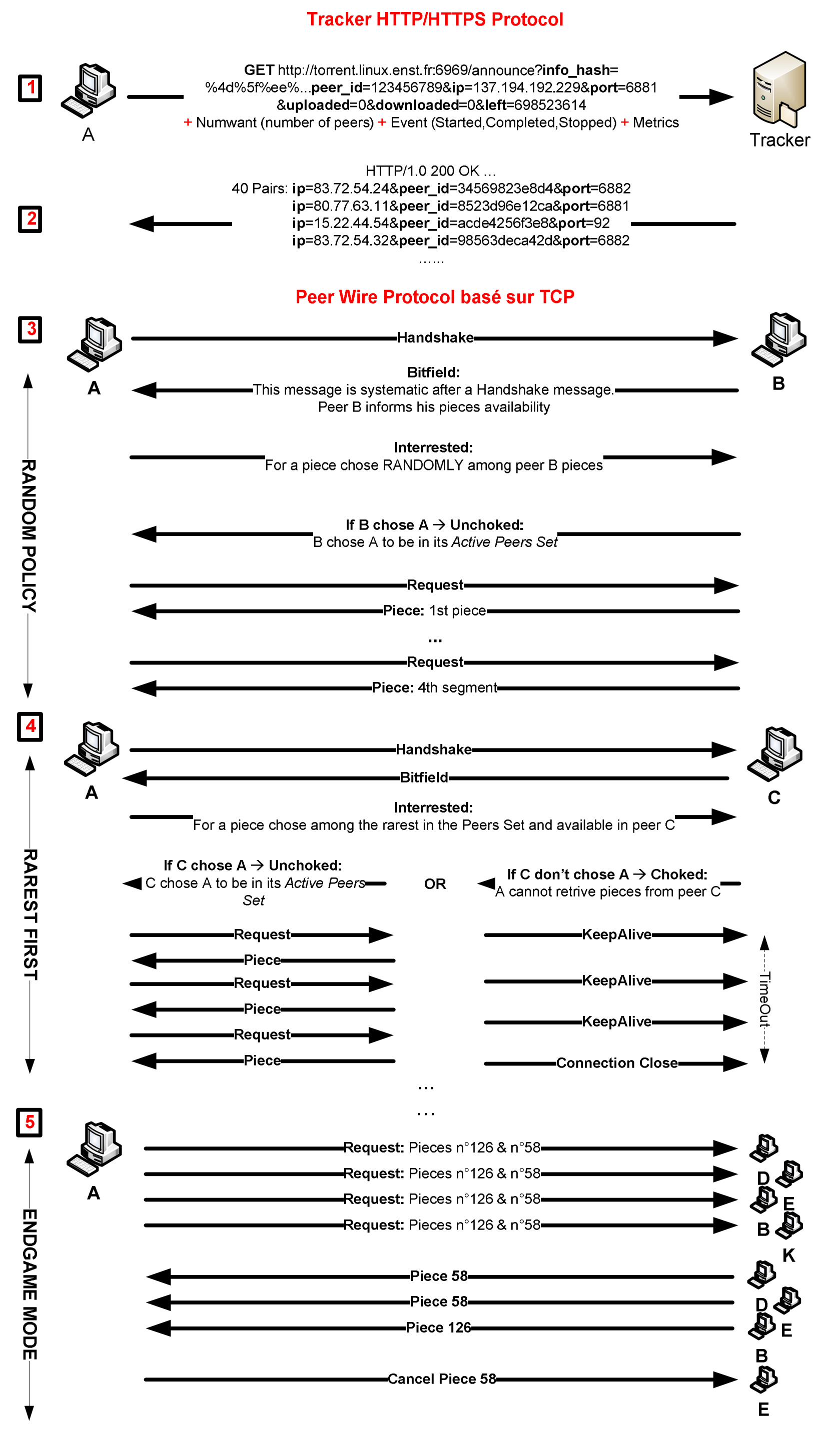

Tracker HTTP/HTTPS Protocol
1
A
GET http://torrent.linux.enst.fr:6969/announce?info_hash=%4d%5f%ee%...peer_id=123456789&ip=137.194.192.229&port=6881&uploaded=0&downloaded=0&left=698523614
+ Numwant (number of peers) + Event (Started,Completed,Stopped) + Metrics
Tracker
2
HTTP/1.0 200 OK …
40 Pairs: ip=83.72.54.24&peer_id=34569823e8d4&port=6882
ip=80.77.63.11&peer_id=8523d96e12ca&port=6881
ip=15.22.44.54&peer_id=acde4256f3e8&port=92
ip=83.72.54.32&peer_id=98563deca42d&port=6882
……
Peer Wire Protocol basé sur TCP
3
A
B
Handshake
Bitfield:
This message is systematic after a Handshake message.
Peer B informs his pieces availability
Interrested:
For a piece chose RANDOMLY among peer B pieces
If B chose A → Unchoked:
B chose A to be in its Active Peers Set
Request
Piece: 1st piece
…
Request
Piece: 4th segment
RANDOM POLICY
4
A
C
Handshake
Bitfield
Interrested:
For a piece chose among the rarest in the Peers Set and available in peer C
If C chose A → Unchoked:
C chose A to be in its Active Peers Set
OR
If C don't chose A → Choked:
A cannot retrive pieces from peer C
Request
Piece
Request
Piece
Request
Piece
KeepAlive
KeepAlive
KeepAlive
Connection Close
TimeOut
RAREST FIRST
…
…
5
A
Request: Pieces n°126 & n°58
Request: Pieces n°126 & n°58
Request: Pieces n°126 & n°58
Request: Pieces n°126 & n°58
D
E
B
K
Piece 58
Piece 58
Piece 126
D
E
B
Cancel Piece 58
E
ENDGAME MODE

Figure 2.7: BitTorrent download scenario

Peer A sends an *Interested* message to indicate that it wants to receive data from B. When B is ready to send, it sends an *Unchoked* message. A can download data by sending *Request* message indicating the piece number that peer A wants (corresponding to the Bitfield index) and the bytes interval (offset and length). To each *request* message, a *Piece* message is followed and corresponds to a sent piece. When a piece is entirely received, peer A calculates the piece hash and verifies if it corresponds to the one present in the metadata *Torrent* information file. If the result is the same, peer A adds the piece to the downloaded file and announces to all its neighbors that it has a new piece by sending *have* messages. Peer A neighbors can this report their interest to these messages with *Interested* or *Not Interested* messages. If peer A is willing to accept some other connections in Upload, its sends a *Unchoked* message in reply to an *Interested* message. The *Choke* message on the contrary makes the peer wait.

The *Choke Algorithm* governs the peers selection in BitTorrent. However, it automatically depends on the peers state (*Interested* or *Not Interested)* and it is different depending on the peer nature (*Leech* or *Seed)* [MR05].

In the scenario represented, the peer A is a *Leech*. The algorithm round duration is 10 seconds and it is called any time a peer change state (*Interested* or *Not Interested)*. The round can also be shorter than 10 seconds. The following steps are also considered:

- At the beginning of every 3 rounds (30 seconds), the algorithm chose a peer as the Optimistic Peer or POUP for *Planned Optimistic Unchoked Peer*. This peer is chosen randomly.
- The algorithm lists peers that are *Interested* and that sent a minimum of one block during the last 30 seconds. This choice is made depending on the downloading peers Download Rate. A peer that did not send during the last 30 seconds is *snubbed* (excluded to guarantee the fact that only active peers are *Unchoked*).
- The 4 fastest peers in terms of Download capacity are chosen and are *Unchoked*.
- If the POUP is part of the 4 *Unchoked* peers, another one must be chosen randomly. If this peer is *Intersted*, it is *Unchoked* and another one is chosen as POUP randomly. If this one is also one of the 4 previously chosen, the previous step is repeated. It is possible that more than 5 peers are *Unchoked* by the algorithm but also 5 peers *Interested* will be *Unchoked* during one round. *Unchoking* peers that are not *Intersted* permit to calculate again the *active peers set* until one of the *Unchoked* peers become *Interested*. The algorithm is repeated every time an *Unchoked* peer becomes *Interested*.

The algorithm in the *Seed* case functions differently. We have the same round principle that lasts 10 seconds. The algorithm is called every time a *Unchoked* peer is *Interested* or *Not Interested* and every time a peer leaves the *Peers Set*.

- The algorithm lists the peers depending on the duration they last in their last *Unchoked* state. The peers that were *Unchoked* recently (less than 20 seconds) or that have some block requests pending are taken into consideration. The Uploading Rate is used as the decision criterion between peers that spend the same time in their last *Unchoked* state.
- The peers that have the better Uploading Rate are listed and the ones that have the better Upload have priority.
- During 2 rounds over 3, the algorithm keeps 4 peers *Unchoked* and 1 POUP is *Unchoked*. For the 3rd round, the algorithm keeps the 4 *Unchoked* peers.
- Peers in the *Active Peers Set* change frequently.

In previous version of BitTorrent, we used to promote peers depending on their best Downloading rate but this is an inconvenient because a peer can monopolize the resources of a *Seed*. For some voluminous *Torrents*, this is not a problem but for small *Torrents* that have only few *Seeds*, one or more *Seeds* can be too much used and pieces propagation of rare pieces would be penalized.

Taking the case of an egoist peer, it can monopolize the *Seed* at the beginning of *Torrent* lifetime and delay its launching.

Pieces selection policy [LEG&al06]**: *Rarest First* and End Game Mode**

This algorithm has for goal to maximize piece entropy and variety for a specific *Torrent* among the peers. It must avoid that exchanged pieces become rare. The principle is to choose to download the rarest pieces first. Each *Leech* keeps a list that contains the information of all peers' pieces number. This list permits to decide which pieces are the rarest. The last block problem must be avoided this policy in collaboration with the *End Game Mode* activated at the end of a session. This latter concerns the end of the download and allows a *Leech* to flood requests to all other peers at the same time to ask for the last blocks that the local peer did not retrieve yet. This is to complete last pieces download when a lack of some blocks is recognized.

Rarest First is the policy that follows the *Random Policy*. It is applied during the rest of the download. The piece selection is important for the BitTorrent system performance. A poor piece selection can have some bad consequences.

The local peer maintains the copies number of each piece in its *Peers Set*. It uses this information to define which are the rarest pieces. If *n* is the copies number of the rarest piece, then each piece id having *n* copies in the *Peers set* will be added to the rarest pieces list. We talk about the *Rarest Pieces Set*. Then the peers download pieces that the distant peers have the less. This technique permits to ensure that peers will have all pieces from distant peers and avoid blocking at the end if peers that had rarest pieces decide to disconnect or leave. Then sending pieces becomes more flexible. We note that therefore, the downloading of pieces that are the most present on the *Peers Set* is done at the end of the session.

The information theory shows that no peer downloading can finish until all pieces of the file have not been sent by the *Seed* [LEG&al05]. In the case there exists a unique *Seed* that has an upload capacity minimal compared to most of the *Leeches* capacity, performances would be better in the case peers are downloading from different peers than the *Seed*. This would avoid some redundant downloading that could limit *Seed* upload number. The *Rarest First* policy is efficient for new pieces download from the *Seed* because a peer that is downloading could see that distant peers have pieces that the *Seed* could send before.

It can happen that the original *Seed* is turned off for some costs reason and allowing *Leeches* to work in Upload. The risk is that a specific piece could not be found in the *Leeches*. But *Rarest First* manages this problem by replicating as soon as possible the rarest pieces to reduce their loss risks.

In Figure 2.7 that represents the downloading scenario, we represented for the *Rarest First* a unique exchange between peer A and peer C for some simplicity reasons but this exchange is not unique in this step. In fact, a *Leech* does multi-sourcing and can download from multiple sources at the same time or one after the other. For the beginning of the process, the first 4 pieces are in general downloaded from the same peer because the new peer worries about having something to share quickly.

It is difficult in BitTorrent to make a standard scenario and to know exactly who between the receiver and the sender chose the peers with whom they will communicate and who instantiate them. In fact the receiver chose its distant peers only if it is *Interested* by some pieces they have. However these distant peers have the last word because they decide at the end to which they will accept to send some pieces. We detailed the *Choke Algorithm* that permits the distant peers to accept to upload to the local peer or not depending on downloading or uploading rate if the sender is a *Leech* or a *Seed*. This is why we propose a state diagram in Figure 2.8 with transitions. At the initial state 1 a *Leech* is *Choked* and *Not Interested*.

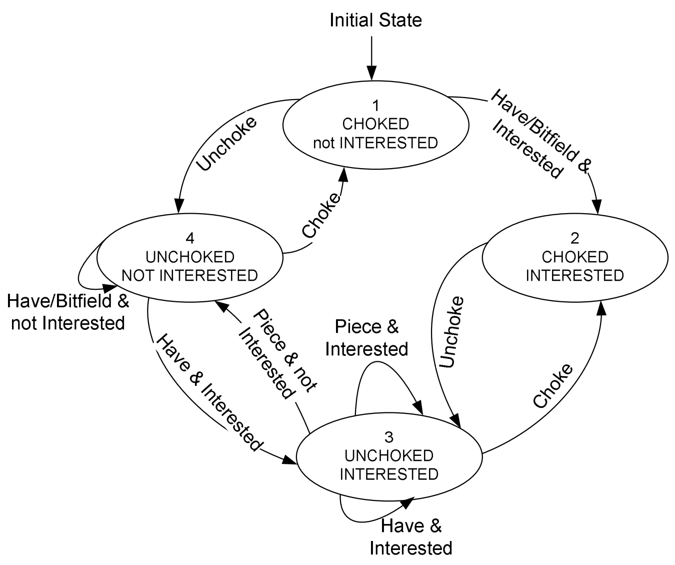


Figure 2.8: BitTorrent connection states for a *Leech*

By receiving a *Have* or *Bitfield* message from a neighbor peer that is in its *Peers Set*, a peer can realize that this neighbor has a piece that interests it. The local peer becomes *Choked* and *Interested* (state 2). From state 1 it can be *Unchoked* by a sender but without being *Intersted* yet. In this state, it can receive a proposition in a *Have* message and become *Interested* to reach state 3. At state 3, the peer can receive a piece or a proposition in a *Have* message from the same peer or another for another piece and it will stay in state 3, or it can receive a piece without being *Interested*. It returns to state 4. We can note that in this state the local peer can receive a *Have* proposition or a *Bitfield* and stay *Not Interested*. The transitions from the state *Unchoked* to *Choked* and inversely are only done between state 1 and 4 and between state 2 and 3.

2.3.2.2 Specification

It is important to study the formal protocol specification and its different messages sent between the entities. The following specification is detailed based on [BTO] and [BTW]. A BitTorrent specification is proposed in its version 1.0 and permits to remove any ambiguity and generality. In the context of any improvement or modification of BitTorrent protocol, the specification fixes the protocol exchanges and facilitates the basic protocol understanding before passing to a possible evolution.

The specification concerns the *Torrent* file and its structure, the exchange protocol between peers (Peer Wire Protocol) and the protocol between the *Tracker* and the peers (*Tracker* HTTP/HTTPS Protocol). We will detail each message structure between the different entities of the architecture.

The data structure is in BEncoding, a concept that is derived from Python language. It specifies and organizes data within a particular format that support four types that are: *String*, *Integer*, *List* and *Dictionary*.

Table 2.2: BEncoding data structure

Type	Description	Format	Example
Strings	Normal Strings [series of continuous characters]	*<string length>*:*<string data>*	*7:overlay*
Integers	Normal integers	**i***<integer>***e**	*I8e* represents 8.
Lists	They are lists of types [strings, integers, lists, dictionaries].	**l***<bencoded type>***e**	*l7:overlay:network***e** represents the list of two strings: ["overlay", "network"]
Dictionaries	They are a mapping of keys to values	**d***<bencoded string><bencoded element>***e** The keys are bencoded **strings**	***d****4spam1:a1:be***e** represents the dictionary {*« spam » => [a,b]*

Metainfo .Torrent file structure

Torrent file is a dictionary that concern a *Torrent* linked to a simple file (simple mode) or to multiple files (multiple mode). The dictionary is formed by 6 fields when 2 are mandatory.

The following table gives the structure of a single-file torrent.

Table 2.3: Single-file *Torrent* data structure in BitTorrent

Key	**Description**
Info	A dictionary that describes the files
- length	Length of file in bytes (integer)
-md5sum(optional)	A 32 character hexadecimal string corresponding to the MD5 sum of the file.
-name	The filename of a string (string)
-piece length	Number of bytes in each piece (integer)
-pieces	String consisting of the concatenation of all 20-byte SHA1 hash values, one per piece.(raw binary encoded)
Announce	The announce URL of the *Tracker*
Announce-list(optional)	This is an extension to the official specification, which is also backwards compatible. This key is used to implement lists of backup *Trackers*.
Creation date (optional)	The creation time of the torrent, in standard Unix epoch format (integer seconds since 1-Jan-1970 00:00:00 UTC)
Comment(optional)	Free form text comments.(string)
Created by(optional)	Name and version of the program used to create.

Table 2.4: Multi-files *Torrent* data structure in BitTorrent

Key	**Description**
Info	A dictionary that describes the files
ofiles	List of dictionaries, one for each file. Each dictionary in this list contains the following keys
- length	Length of file in bytes.(integer)
-md5sum(optional)	A 32 character hexadecimal string corresponding to the MD5 sum of the file.
-path	List containing one or more string elements that together represent the path and filename. Each

	element in the list corresponds to either a directory name or (in the case of the final element) the filename. For example, the file "dir1/dir2/file.ext" would consist of three string elements: "dir1", "dir2", and "file.ext".
oname	Name of the top-most directory in the structure -- the directory which contains all of the files listed in the above **files** list. (string)
opiece length	Number of bytes in each piece(integer)
opieces	String consisting of the concatenation of all 20-byte SHA1 hash values, one per piece.(raw binary encoded)
Announce	The announce URL of the *Tracker*
Announce-list(optional)	This is an extension to the official specification, which is also backwards compatible. This key is used to implement lists of backup *Trackers.*
Creation date (optional)	The creation time of the torrent, in standard Unix epoch format (integer seconds since 1-Jan-1970 00:00:00 UTC)
Comment(optional)	Free form text comments.(string)
Created by(optional)	Name and version of the program used to create.

This is for instance a multiple mode file described in Bencoding:

announce=>http://torrent.linux.duke.edu:6969/announce

creation date=>1089948866

info=><dictionnary> {

files=><list>[
<dictionnary>{length=>159332;path=>[conary, conary-0.1.tar.bz2] }
{length=>57;path=>[linux,alpha,0.1,iso,MD5SUM]}
{length=>1778;path=>[linux,alpha,0.1,iso,README]}
{length=>618993664;path=>[linux,alpha,0.1,iso,speci fix-linux-0.1.iso]}
]
name=>specifix
piece length=>262144
pieces=>47240:uÚÎº2D:×íÎ¡1bõJu(...)

This file *specifix.*torrent is formed by 4 files divided into segments with a size 262144 for each. The *Tracker* is available at *http://*torrent.*linux.enst.fr:6969/announce.*

If the port 6969 is not available, the Web port 80 is used.

Tracker **HTTP/HTTPS protocol**

This protocol defines the communication between a peer and the *Tracker* responsible of the *Torrent* linked to the desired resource [BIT06].

We know that the *Tracker* permits a specific peer to obtain the *Peers Set* where the local peer can find the peers that are downloading the same resource. The *Tracker* acts like an HTTP server to which the local peer gives some parameters to obtain these peers addresses. We can identify the *Tracker* as a HTTP/HTTPS service that answers to HTTP GET messages that the peers send. These http requests contain some metrics that permit the *Tracker* to keep various statistics on the torrent. The *Tracker* response contains Peers list that permits clients to participate on the torrent. Like in all HTTP request, parameters are putted following the *Tracker* URL. Everyone is preceded by an interrogation mark and the *Tracker* URL is the one contained in the *Torrent* file announce URL field described previously. Each parameter is seperated by a « & ».

This is the HTTP GET message structure:

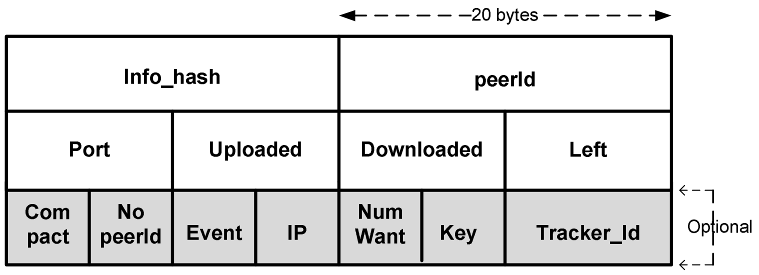


Figure 2.9: Get announce message

Table 2.5: Get announce message fields

Parameter	**Description**
info_hash	20-byte SHA1 hash of the *value* of the **info** key from the Metainfo file. Note that the *value* will be a bencoded dictionary, given the definition of the **info** key above.
peer_id	20-byte string used as a unique ID for the client, generated by the client at startup. This is allowed to be any value, and may be binary data. There are currently no guidelines for generating this peer ID.
port	The port number that the client is listening on. Ports reserved for BitTorrent are typically 6881-6889. Clients may choose to give up if it cannot establish a port within this range.
uploaded	The total amount uploaded so far, encoded in base ten ascii.
downloaded	The total amount downloaded so far, encoded in base ten ascii.
left	The number of bytes this client still has to download, encoded in base ten ascii.
event	If specified, must be one of **started**, **completed**, or **stopped**. If not specified, then this request is one performed at regular intervals.
-started	The first request to the *Tracker* **must** include the event key with the **started** value.
-stopped	Must be sent to the *Tracker* if the client is shutting down gracefully.
-completed	Must be sent to the *Tracker* when the download completes. However, must not be sent if the download was already 100% complete when the client started. Presumably, this is to allow the *Tracker* to increment the "completed downloads" metric based on this event
ip	Optional. The true IP address of the client machine, in dotted quad format or rfc3513 defined hexed IPv6 address. Notes: In general this parameter is not necessary as the address of the client can be determined from the IP address from which the HTTP request came. The parameter is only needed if the IP address that the request came in on is not the

	IP address of the client. This happens if the client is communicating to the *Tracker* through a proxy (or a transparent web proxy/cache.) It is also necessary when both the client and the *Tracker* are on the same local side of a NAT gateway.
numwant	Optional. Number of peers that the client would like to receive from the *Tracker*. This value is permitted to be zero. If omitted, typically defaults to 50 peers.

The *Peers Set* size is 50 by default. This list is sent in the response.

Table 2.6: Get Response message fields

Key	Description
failure reason	If present, then no other keys may be present. The value is a human-readable error message as to why the request failed (string).
interval	Interval in seconds that the client should wait between sending regular requests to the *Tracker*
complete	number of peers with the entire file, i.e. *Seed*ers (integer)
incomplete	number of non-*Seed*er peers, aka "*Leech*ers" (integer)
peers	The value is a list of dictionaries, each with the following keys
-peer id	peer's self-selected ID, as described above for the *Tracker* request (string)
-ip	peer's IP address (either IPv6 or IPv4) or DNS name (string)
-port	peer's port number (integer)

A client can send requests at different intervals than the ones given by the *Tracker* but this is possible at a state change (*stopped* or *comlpeted*) or also if the client needs to know more about peers. The most simple way for a client to have a larger peers number is to precise it in the *numwant* field

Peer Wire Protocol

The Peer Wire Protocol facilitates pieces exchange as described in the **meta-info** file. A client must maintain state information for each connection that it has with a remote peer:

- **Choked**: Whether or not the remote peer has choked this client. When a peer chokes the client, it is a notification that no requests will be answered until the client is “unchoked”. The client should not attempt to send requests for blocks, and it should consider all pending (unanswered) requests to be discarded by the remote peer.

- **Interested**: Whether or not the remote peer is *Interested* in something this client has to offer. This is a notification that the remote peer will begin requesting blocks when the client unchokes it. The states are:

- **am_choking**: this client is choking the peer

- **am_interested**: this client is *Interested* in the peer
- **peer_choking**: peer is choking this client
- **peer_interested**: peer is *Interested* in this client

Client connections start out as "choked" and "*Not Interested*". In other words:

- **am_choking** = 1
- **am_*Interested*** = 0
- **peer_choking** = 1
- **peer_*Interested*** = 0

A block is downloaded by the client when the client is *Interested* in a peer, and that peer is not choking the client. A block is uploaded by a client when the client is not choking a peer, and that peer is *Interested* in the client. It is important for the client to keep its peers informed as to whether or not it is *Interested* in them. This state information should be kept up-to-date with each peer even when the client is choked. This will allow peers to know if the client will begin downloading when it is unchoked (and vice-versa).

Data Types:

Unless specified otherwise, all integers in the peer wire protocol are encoded as four byte big-endian values. This includes the length prefix on all messages that come after the handshake.

Message flow:

The peer wire protocol consists of an initial handshake. After that, peers communicate via an exchange of length-prefixed messages. The length-prefix is an integer as described above.

Handshake:

The handshake is a required message and must be the first message transmitted by the client.

- **handshake**: <pstrlen><pstr><reserved><info_hash><peer_id>
    - **pstrlen**: string length of <pstr>, as a single raw byte .
    - **pstr**: string identifier of the protocol.
    - **reserved**: eight (8) reserved bytes. Each bit in these bytes can be used to change the behavior of the protocol. *An email from Bram suggests that trailing bits should be used first, so that leading bits may be used to change the meaning of trailing bits.*
    - **info_hash**: 20-byte SHA1 hash of the info key in the metainfo file. This is the same info_hash that is transmitted in *Tracker* requests.
    - **peer_id**: 20-byte string used as a unique ID for the client. This is the same peer_id that is transmitted in *Tracker* requests.

In version 1.0 of the BitTorrent protocol, pstrlen=19, and pstr="BitTorrent protocol". The initiator of a connection is expected to transmit their handshake immediately. The recipient may wait for the initiator's handshake; if it is capable of serving multiple *Torrents* simultaneously (torrents are uniquely identified by their info_hash). However, the recipient must respond as soon as it sees the info_hash part of the handshake. The *Tracker*'s NAT-checking feature does not send the peer_id field of the handshake.

HandShakeReply:

If a client receives a handshake with an info_hash that it is not currently serving, then the client must drop the connection.

If the initiator of the connection receives a handshake in which the peer_id does not match the expected peer_id, then the initiator is expected to drop the connection.

If everything matches the receiver responds with handshake message with peer_id field modified to its own ID.

peer_id: There are mainly two conventions of coding client version information into the *peerId*, Azureus-style and Shadow-style. Azureus-style uses the following encoding: '-', two characters for client id, 4 ASCII digits for version number, '-', followed by random numbers. For example: '-AZ2060-'... Shadow-style uses the following encoding: 1 ASCII alphanumeric for client identification, up to five characters for version number followed by three characters (commonly '---'), followed by random characters. Each character in the version string represents a number from 0 to 63. For example: 'S58B-----'... for Shadow's 5.8.11.

Messages:

All of the remaining messages in the protocol take the form of

<length prefix><message ID><payload>

The length prefix is a four byte big-endian value. The message ID is a single decimal character. The payload is message dependent.

Keep-alive: <len=0000>

The *keep-alive* message is a message with zero bytes, specified with the length prefix set to zero. There is no message ID and no payload.

Choke: <len=0001><id=0>

The *choke* message is fixed-length and has no payload.

Unchoke: <len=0001><id=1>

The *unchoke* message is fixed-length and has no payload.

Interested: <len=0001><id=2>

The *Interested* message is fixed-length and has no payload.

Not interested: <len=0001><id=3>

The *Not Interested* message is fixed-length and has no payload.

Have: <len=0005><id=4><piece index>

The **have** message is fixed length. The payload is the zero-based index of a piece that has been successfully downloaded. Intially the peers tell each other about the pieces they have using Bit-Field message later when they have downloaded another piece, they use the "have" message to tell the other peers now it also has this piece.

Bitfield: <len=0001+X><id=5><bitfield>

The *bitfield* message may only be sent immediately after the handshaking sequence is completed, and before any other messages are sent. It is optional, and needs not be sent if a client has no pieces.

The *bitfield* message is variable length, where X is the length of the bitfield. The payload is a bitfield representing the pieces that have been successfully downloaded. The high bit in the first byte corresponds to piece index 0. Bits that are cleared indicated a missing piece, and set bits indicate a valid and available piece. Spare bits at the end are set to zero.

Request: <len=0013><id=6><index><begin><length>

The *request* message is fixed length, and is used to request a block. The payload contains the following information

- index: integer specifying the zero-based piece index
- begin: integer specifying the zero-based byte offset within the piece
- Length: integer specifying the requested length. This value must not exceed 2^{17}bytes, typical values are 2^{15} bytes.

The observant reader will note that a block is typically smaller than a piece (which is commonly >= 2^{18} bytes). A client should close the connection if it receives a request for more than 2^{17} bytes.

Piece: <len=0009+X><id=7><index><begin><block>

The *piece* message is variable length, where X is the length of the block. The payload contains the following information

- index: integer specifying the zero-based piece index
- begin: integer specifying the zero-based byte offset within the piece

- Block: block of data, which is a subset of the piece specified by index.

Cancel: <len=0013><id=8><index><begin><length>

The *cancel* message is fixed length, and is used to cancel block requests. The payload is identical to that of the "request" message. It is typically used during "End Game" (see the Algorithms section below).

2.3.3 Related works on BitTorrent studies

[THE&al04] and [XIA&al10] propose complete surveys on Peer-to-Peer and BitTorrent (for the second one) performances. They provide a good classification of the different studies present in the literature.

BitTorrent has proved its efficiency comparing to other file sharing protocols using the P2P model. Its performance is recognized thanks to its piece and peer policies. Many performance studies exist in the literature but it is not easy to globally appreciate one comparing to another. We will present in this section a general overview of the different and major works that appreciate the popularity and performance of BitTorrent. Then we will also list the contribution on locality-aware techniques and erasure codes applied to BitTorrent protocol. In fact, this section is important while in our work we propose some simulation results concerning a locality-aware technique for peer partitioning and a specific FEC mechanism that speeds up data transfer. The different ways a performance study is performed in BitTorrent literature are various.

The first method is to use measurements based on *Tracker* logs, provided from *Torrent* sites, with traffic analyzers, collecting logs in clients or even experiments a PlanetLab [LAB] network. The most famous works using this method are [IZA&al04], [POU&al04], [LEG&al05] and [DAL&al08]. The main points in these works are the study of peers' evolution in BitTorrent network and traffic volume uploaded and downloaded. Usually the analysis is done in a long period of time (from 4 to 8 months) but measurements present less flexibility than simulation when the simulations number is high enough.

The second method is based on simulation. In [MR05] a complete simulator modeling all BitTorrent mechanisms and algorithms is used to study *Rarest First* policy and *Choke Algorithm*. In [FEL&al04], [URV&al06] and [HAM&al07] authors also propose their own simulator to study the latter mechanisms with the impact of some BitTorrent parameters like the Peers Set size. Our work can position itself in this category while we propose a complete study based on a greatly modified version of [MR05] simulator. We also studied the Peers Set size variation, the Piece size variation, the localization impact on BitTorrent [PBL05]. Our simulations were various exploiting many scenarios with various *Leeches/Seeds* number integrating also FEC mechanism [PBL07]. We also proposed a statistical test validating the results (*cf* Section 4). A global architecture was developed and published in [PBL06]

The third method is to study BitTorrent based on analytical model. Many models have been proposed as fluid model in [QIU&al04], [GUO&al07] and [TIA&al07]. Those proposals have been extended by others as we will detail it in next sections.

We will also present contributions on locality-aware techniques in BitTorrent like

[CHF&al08] and [REN&al10] that are client oriented pluggin while our work is Service Provider oriented. In [BIN&al06], [PAP&al06] or [LEG&al09] authors proposed biased neighbor selection while we add to a new peer selection policy the possibility to adapt some parameters like the Peers Ser size or the Piece size depending on the application needs. The objective is to be close to ALTO [XIE&al07] and SmoothIT [PUS&al09] but without any complexity like in the latter proposal that requires some changes in the infrastructure and the network entities. We prefer a simpler and interoperable solution.

Erasure codes integrating in BitTorrent is a major part of our work. We propose a complete study of their application depending on many parameters in order to evaluate the impact (positive or not) in BitTorrent applications. Works like [BYE&al99] or [LAC&al02] propose also to integrate erasure codes to P2P networks and the actual research is focusing on Network Coding [GKA&al06] that is more complex and where all peers must be implicated. We encourage a solution where only *Seeds* are coding redundancy while we are Service Provider oriented.

First we present some BitTorrent-like protocols that have been proposed and having some similarities in their policies and algorithms. At the origin, these proposals were defined to improve file-sharing systems performance but BitTorrent stays the best protocol and the most famous one for this issue.

In [SHE&al04] Slurpie is designed to reduce the downloading time and increase the system scalability. The idea is to form a mesh but with the specific advantage that download bandwidth is adapted dynamically with a proper estimation technique proposing reports. This bandwidth while varying can change the neighbors' number depending on the estimation. The more the peers' number is important the more Slurpie is considered as efficient. The major problems with Slurpie are that the protocol is much more complex than BitTorrent and it is a must in this protocol to estimate the number of peers in the Slurpie network. However, Slurpie proposes less load on the central server (equivalent to the BitTorrent *Tracker*) and on the principal originally source (equivalent to the *Seed* in BitTorrent), but we do not have information on performances when the network is experiencing a flash-crowd when the peers number is high.

In [SHE&al06] the Fair Optimal eXchange protocol (FOX) has been proposed as an efficient file sharing BitTorrent-like protocol that focuses on fairness. The principle is to achieve a symmetric communication for giving and receiving with some stable communications. A system is able for peers that complete their download to leave by providing some encrypted information that the other peers needed before their departure.

Avalanche presented in [GKA'&al06] is a Microsoft proposal for a BitTorrent-like protocol that integrates Network Coding [GKA&al05] mechanism. When a piece is downloaded by a peer, Avalanche takes into consideration that this download can be interesting for other peers. It avoids that this piece is requested afterwards by another peer. In fact, when a peer download a piece a system that implementing coding inside the network let the neighbor peers having information on pieces they did not especially asked but that they would ask in the future. The principle is some Xor operation between the pieces to mix the overall pieces information.

2.3.3.1 Performance studies based on measurements

In [IZA&al04] the BitTorrent protocol is evaluated during a five months study in

BitTorrent clients' behavior, *Seeds* contribution, the *Seeds/Leeches* ratio, the *Rarest First* and *Choke Algorithm* efficiency. This work is a performance evaluation that focuses principally on the peers' evolution and the traffic volume downloaded and uploaded during the five months analysis. The traces are based on two principal sources of analysis on a unique torrent. The first source performs a global and macroscopic view using a *Tracker* log of thousands of peers and the second one is a session-based analysis that shows more the *Leeches* behavior between each other.

In [POU&al04] the authors studied the peers' number and their relation with *Seeds*, the time peers stay as *Seeds* after completing the download, and the resource lifetime. Like the previous work, they especially focused on peers behavior and content volume distributed in the network. The way they studied BitTorrent was to use Suprnova.org mirror site to retrieve measures during 8 months. The authors work to better understand the performances of a global combination of BitTorrent and Suprnova, the availability and the integrity of this network.

[LEV&al04] is a study where information has been collected from two different *Trackers* to show that BitTorrent can serve large files. The *Torrent* is observed during a four months period and the results are studied in comparison with other P2P systems to show how BitTorrent availability and content management can be efficient for much larger files (multi-gigabyte resource volume size).

In [LEG&al05] the authors proposed a complete and detailed study of BitTorrent and complement it in [LEG&al06] to show the efficiency of *Rarest First* and *Choke Algorithm*, proving that they are enough for BitTorrent performances. The way they studied BitTorrent is by using the original mainline Bram Cohen's client [COH03]. They show with good arguments that *Rarest First* decreases well dynamically rarest pieces in the network and the *Choke Algorithm* using Tit-For-Tat works well at the block level, better than at the bit level ensuring fairness and robustness. They motivate the implementation of such algorithms without introducing any other complex method as erasure codes or network coding. It is true that in the case of complete network with high resource availability and a high number of peers, it is unnecessary to integrate additional mechanisms to BitTorrent. However, P2P networks are dynamic and the free aspect of BitTorrent conduces peers to freely leave and join the network again. We will see that our work treats the transport aspect of P2P by using simple FEC mechanism to speed up data transfer in the case of Service Provider oriented networks and only when it is necessary: lack of resource availability, few peers in the network, real time applications running, etc.

[AND&al05] is a study on the cooperation in BitTorrent. The authors focus on free-riding, *Seeds* importance and sharing ratio. They conclude that free-riders are less important than in other P2P networks like Gnutella for examples. This shows that BitTorrent mechanisms act well against low-sharing *Leeches*. However, the fact that *Leeches* are not uploading is not only due to the fact that they voluntary want to be egoist but because when they decide to upload the peers they contacted cannot receive at the moment or because of the asymmetrical links in the Internet. The main final observation is that BitTorrent is much more cooperative than other P2P protocols.

In [LEG&al07] some experiments on some private *Torrents* show how a peer acts individually in BitTorrent. The study platform used is PlanetLab [LAB] using a modified version of BitTorrent to perform clustering and prove that it can optimize uploading utilization. In fact, the authors show that *Choke Algorithm* contributes to make a peer contacting peers that have the same performance capabilities and more precisely the same bandwidth. This is usually true when the original *Seed* has a bandwidth that is at least the same as the fastest peers downloading the *Torrent* in terms of upload (while the *Seed* doesn't download). This study focuses precisely on implicit clustering of peers in BitTorrent. We also studied various networks as homogeneous and heterogeneous but using a simulation based study that is usually more flexible than measurements based on logs. The authors propose a *Tracker* extension that permits to send the Peers Set with peers that present similar upload bandwidth as the *Leech* requesting. This seems possible only if reports are sent by *Leeches* frequently to let the *Tracker* decide which peers correspond.

In [RAS&al07] some peers have been used to study the BitTorrent performance but not with an overall view. The major results of this work are that the performance distribution of peers is uniform and that the *Choke Algorithm* is the main factor that has an impact on BitTorrent performance. In a BitTorrent network, different types of network can be observed. The particularity of this work is that they explain the advantages and inconvenients of both measurement and simulation studies and provide a measurement methodology to overcome measurement study comparing to simulation which is that in measurement it is difficult to be sure that the log retrieved is really representative. That does not mean, for the authors, that simulations are not complex also while in this case, it is important to simulate all different aspects and mechanisms of the protocol. In our work, we prefer simulation study that is more flexible and we tried to model all mechanisms and algorithms of BitTorrent.

The work in [DAL&al08] proposed a study of these different types of networks. They identify four kinds of network that they studied: Connection Network, Interest Network, Unchoked Network, and Download Network. The platform [LAB] were used to launch a specific BitTorrent client in more than 400 nodes and essentially shows that the initial BitTorrent stage is independent of the overall performance in general. They also explain that this is not obvious to assert that BitTorrent implicitly clustered peers as we could read it in [LEG&al07] except in the initial stage. Their proposition to be closer to this configuration is to imagine what they call small-world networks that are more efficient for spreading information. The investigation put as hypothesis that creating a small-world is not clear. In fact, to create a maximum degree network, it is necessary to maximize the clustering coefficient of a regular graph of these peers. The simpler solution proposed is to implement what they called a small-world *Tracker* that is modified to send to each new *Leech* a Peers Set composed by random peers chosen in a specific group of peers having a maximum neighborhood number degree that has to be respected.

The neighbor selection with the *Choke Algorithm* has been studied in [LEI&al07] and concludes that this random selection does not take into consideration the communication cost and that this results in low transmission rate and high cost. They provided a protocol called ShareStorm that seems to present good performances comparing to BitTorrent but that needs incremental works.

[CHO&al08] presents simulations study that shows how to use more intelligently *Seed* capacity in BitTorrent while improving the performance of contributing nodes. They say that with a specific amount of *Seed*ing capacity (medium) the schemes result in a good impact and degrade the performance of free-riders.

The work presented in [ZHA&al10] study the behavior of BitTorrent in Darknets from macroscopic, medium-scopic and microscopic perspectives. They conclude that the

Seed-to-*Leech* ratios and upload-download ratios are much higher than in public systems. In fact, darknets are private networks where availability is usually much h than in generic public networks.

We also have the studies based on simulation. The following presents the major simulation based studies for BitTorrent-like systems.

2.3.3.2 Performance studies based on simulations

The use of measurement-studies of BitTorrent is difficult when it is important to study every factor that can affect BitTorrent performance. The use of simulations can add more flexibility and efficiency to the study. The variation of parameters is much more interesting while in measurements, the results are fixed and cannot really be adjustable. However, in BitTorrent simulations, most of the cases authors fix some hypotheses to neglect some protocols mechanisms while focusing on the main one without degrading the system and to retrieve the best log information to perform the evaluation the closer from reality.

[FEL&al04] presented a deep study on piece and peer selection strategies and their effects on BitTorrent performance. The authors in this study try to answer to a fundamental problem which is to see if the self-scaling and self-organizing aspects of P2P networks are encouraging to reach a highly efficient, cost effective and robust content distribution protocol. They answer saying that the aggregation of all these resources in P2P networks is not enough while it is important to focus on peers and pieces strategy to reach best performances. The results show that between all different strategies, there is not one that takes advantage on the other. However, they conclude saying that *Rarest First* seems to be the best delivery strategy among all other piece strategies. This joins [LEG&al06] work idea. We will show in our work that *Rarest First* is a very good strategy in comparison to the random one and that adding FEC mechanism in some cases can reveal better performance with *Rarest First* than the *Rarest First* working alone.

[MR05] is a very complete study of BitTorrent mechanisms. The performance is studied during flash-crowd using a simulator that models all peers activities, policies and mechanisms. The results made the authors conclude that BitTorrent is a robust and scalable protocol and that is ensure high uplink bandwidth utilization. The *Rarest First* policy is very important to ensure that new peers have something to share with others. The only drawback of this study is that the number of peers in the simulation was small. We directly extended this work by modifying greatly the simulator that they developed to integrate ASs repartition and complete FEC mechanism.

In [THO&al05] a simulation based analysis is proposed to study the fairness properties of BitTorrent. The metric chosen is the ratio of bytes uploaded to that downloaded by each peer. The authors also propose three modifications to BitTorrent and examine their impact on fairness. The Conditional Optimistic Unchoke, the Multiple Connection Chokes and the Variable Number of Outgoing Connections are all providing positive amelioration to BitTorrent fairness incentive mechanism.

[URV&al06] is a work that evaluates the impact of the overlay topology parameters on BitTorrent performance. The authors show that the Peer Set size and the percentage of outgoing connections have a significant impact on BitTorrent's performance. This study is also a simulation based one that performs measure under flash-crowd scenarios. The drawback of simulating is that we must fix some parameters to study others.

However, when the number of simulations is high enough to allow diversification on the parameters hypothesis, the simulations can be considered as enough representative of the real networks. We will see in our work that we also study the impact of the Peers Set size but fixing some quality and service objective for an ISP running a specific application.

In [WUG&al06] the authors try to find how BitTorrent can be optimal or close to it. They proposed a new distribution scheme called the Centrally Scheduled File Distribution (CSFD) that can considerably decrease the total download time. This distribution scheme is compared to BitTorrent scheme. The authors find that BitTorrent is far to be optimal and that the peer selection is not helping the protocol to optimally decrease the overall distribution. The particularity of this study is that it concerns the dynamics of the built-in control mechanisms. We will see in our work that we chose to play with the peer localization and some FEC mechanism to decrease the total download time instead of changing the protocol algorithms. Our goal is to stay the more interoperable and simple as possible.

[GAR&al06] proposes a system called *2Fast* which solves the problem of freer-riding that affects the download performance, while preserving fairness of bandwidth sharing. The authors propose to form groups of peers that collaborate in downloading a file on behalf of a single group member which can thus use its full download bandwidth. A peer can help other peers in their ongoing downloads and gets help in return during its own downloads.

[WAN&al07] proposes an improvement study that take into consideration the asymmetric bandwidth in Internet and BitTorrent where upload capacity is generally limiting the transfers. The authors show that also integrating, like the previous work, new peers called helpers can add a considerable amelioration as effective as *Seeds*.

[CWU&al07] is a proposal to improve the download time of BitTorrent with a new strategy that replaces the *Rarest First* policy and introduces a strategy based on a greedy concept that a peer assigns each missing piece with the highest priority for next download. The strategy is called the weighty piece selection strategy.

In [HAM&al07] authors follow the work on [URV&al06] and perform an evaluation study on the properties of the distribution overlay in BitTorrent and the relation of this overlay and BitTorrent performances. They used MATLAB to create their own simulator in order to analyze the relation between the overlay properties and the BitTorrent performance. The authors also studied the peers set size and the time for a peer to reach its maximum peers set size. They tried to analyze the overlay generated by BitTorrent showing that this overlay is robust but that this is not a random graph. The authors show two principal problems that may impact BitTorrent overlay: the NATing and the peer exchange creating a chain-like overlay that might impact BitTorrent efficiency.

In [CHE&al09] two main *Seed*ing strategies have been studied in details, based on simulation with a Java simulator and a mathematical model. The original *Seed*ing strategies known in the *Choke Algorithm* is compared to the Time-based *Seed*ing strategy where the *Seed* provides data to *Leeches* during a uniform interval of time. The authors studied the impact of free-riders and introduce another type of egoist *Leeches* that are named Exploiters: those that leave the network just after becoming *Seeds*. They conclude saying that both free-riders and exploiters harm the system despite the *Seed*ing strategies that is used. They also say that the time-based *Seed*ing strategy is fairer than the original one even if this latter is more efficient when the number of egoist peers is small. Our work does not deal with proper *Seed*ing strategies and their impact while we chose to keep

BitTorrent in its original specification.

The last method we know for studying BitTorrent performance is to model the protocol analytically. We detailed the different works known in the literature in the following.

2.3.3.3 Performance studies based on analytical models

The principal famous analytical modeling studies in BitTorrent are classified into two principal categories which are the homogeneous and the heterogeneous ones. In the first category, the peers have the same capabilities in terms of download and upload rates. Most of the models for BitTorrent are fluid-flow models where the approach is to follow the number of peers (*Leeches* and *Seeds*). In our work, the mathematical analytical model is out of scope but it is important to know how BitTorrent can be modeled to understand the peers and the protocol behavior. Following the different proposals based on analytical model.

[QIU&al04] presented a fluid-flow model. It proposes a deterministic model describing the evolution of the *Leeches* and *Seeds* number also. They conclude that the average download time is not related to the arrival rate and that the system scales very well with the peers increasing. They studied the built-in incentive mechanism in BitTorrent and its effect on network performance. They proposed a validation part based on both simulation and real traces obtained from the Internet. In terms of modeling, this paper appears as a complete reference in the literature.

In [GUS&al05] the parameter that is principally studied is the service capacity for transient and steady-state regimes. An assumption has been chosen here: no peers leaves the system and aborted the download. Based on an abstract model of P2P system, the authors demonstrated that the scaling is favorable for the download while load of charge is increasing. A markovian approach has been adopted to model the stationary regime. They concluded by proposing a specific fairness policy that they assume to be better for dynamic P2P system like BitTorrent-like ones. The authors also partially validated their work using traces obtained from a second generation P2P application. This work is also developed in [YAN&al06] and detailed in section 2.3.4.1.

[GUO&al05] is a work inspired from [QIU&al04]. The authors analyzed the file downloading trace files obtained from *Trackers*. They conclude saying that the peer arrival rate to a *Torrent* is exponentially decreased. This result permits them to extend the work in [GUO&al05] for this decreasing rate. They studied various parameters like the *Seed* departures, the evolution of peers in the system and they proposed a graph for analyzing inter-torrent collaboration. [GUO&al07] is a work proposing incremental work comparing to the previous on extensive measurement and trace analysis.

In [ART&al05] an analysis of data dissemination is proposed. The Tit For Tat strategy is ignored and the authors assumed that the peers are homogeneous. We will see in our work that by simulation, we can approach closer to networks reality which is to propose some heterogeneous environments.

The authors in [KUM&al06] presented an analytical model of file sharing in P2P networks also using a fluid-flow model. They proposed to study the advantages of a P2P file sharing protocol comparing to a Client-Server system. They tried to find the minimum download time to finish the distribution to all *Leeches* in the system. They adopted a good heterogeneous approach very close to reality even if some non realistic assumptions have been chosen concerning the bit level of data transfer. However, the study gives a good approximation in terms of performance. Here the study is limited to 10 *Leeches* for a unique *Seed*.

In [ERM&al05] the authors performed a modeling methodology and some measurement to study the entire session characteristics of BitTorrent. They found that BitTorrent session inter-arrivals can be modeled by the hyper-exponential distribution while session duration and sizes can be modeled by the lognormal distribution.

In [FAN&al06] the authors also based on the famous work in [QIU&al04] to propose a model based on a stochastic differential equation approach. They divided peers into three types which are *Leeches* that have a few pieces, *Leeches* that have most of pieces and *Seeds*. They propose to fix a probability to each connection. They analyzed the file completion time and the file availability. Here a discrete-event simulator has been used to validate the results concerning the effect of various parameters like the peers' arrival rate, the *Seeds*' departure or even the transfer bandwidth on performance measures.

[TIA&al07] is a deep and complete work that extended the fluid model in [QIU&al04] to study the peers in different states of the download. This work is detailed in 2.3.4.2.

With java modeling tools, [SAR&al07] presents a queuing model for BitTorrent where each peer is considered as a load dependent host. The service is divided into slots and a request time is equal to the time needed to download one piece. The work focuses on the download behavior and the incentive mechanism.

[PIC&al07] is the first work that proposes a study based on heterogeneous fluid-flow model. To work on different access link capacities, the authors developed a model with two different capacity classes by extending the work in [QIU&al04]. The two classes are high rate and low rate. The performance of heterogeneous networks is compared by the homogeneous ones and they conclude saying that heterogeneous bandwidth can have a good effect on content distribution among peers in some certain scenarios.

In the same optic, the work in [LIA&al07] proposes a model based on heterogeneous networks with also two classes: high and low peers. The authors propose a mathematical model that helps them to predict the average file download delay for both classes of peers. The used the BTSim simulator [BTS] to experiment the proposed model.

2.3.3.4 Contributions on locality-aware for BitTorrent

Recently, the P2P community showed a lot of interest on peer localization and how this can decrease inter-AS traffic and ISP costs. All these propositions choose to modify the original random selection by a specific selection of some peers inside of ISP with a certain fixed threshold. However, for the best of our knowledge, none of the propositions specified a unique manner to map each peer with the AS it belongs to.

The work in [QUR04] proposes a new peer selection based on proximity. The *Tracker* sends information on nearby peers to improve the download. They compare an approach where the requesting peer floods an announcement to discover peers and an approach based on Gossip protocols that use a low-rate probabilistic flooding mechanism.

In [KAR&al05] the authors performed a study on the impact of ISPs on BitTorrent. They proved that BitTorrent is locality unaware and this increases ISP costs. They showed that in BitTorrent, the content is sent 30 to 70% more times and that some mechanisms can help decreasing the inter-ISP traffic.

In [BIN&al06], a new selection called the biased neighbor selection is proposed to improve traffic locality. The idea is to select peers according to each of these AS numbers. In this solution, the *Tracker* forces each new *Leech* to select a majority of its neighbors within the same ISP and only few outside of it. This solution has no proposition for the peers mapping with the AS's.

The authors in [PAP&al06] extended the solution in [BIN&al06] by inserting ISP-Owned peers to enhance performance within an ISP domain. They also compare multiple locality selection with their proposition showing the added-value they bring.

In [AGG&al07] the authors evaluate the feasibility of a solution where the ISP offers an oracle to P2P users. The peers provided a list of neighbors and the oracle ranks them according to certain criteria like proximity or bandwidth.

In [ZHN&al07] the authors explore the use of proximity in the construction of the overlay network and the efficient exchange of the file fragments In BitTorrent with the main goal of reducing download time and resource usage. In our work, we propose a Context-Aware DHT that generalizes the traffic partitioning based on the *hTracker* peer selection policy. Our work also aims to reduce traffic exchanged between ASs.

In [YAM&al07] the authors propose a method to constitute P2P content distribution networks and a reduction in ISP costs by considering the form of the ISP interconnection in its distribution. The authors show that the proposal achieves a reduction in ISP costs and distribution time.

The Ono project [CHF&al08] is a pluggin added to Azureus client taking pressure off international and other long distance transfers, to increase file download speeds. To determine which peers are close by, Ono learns from existing Content Distribution Networks (CDNs) such as Akamai [AKA] and Limelight. Contrarily to other propositions like P4P, Ono does not need any cooperation between ISPs and P2P applications. It is a customer-oriented service.

Some solutions proposing locality enhancement have been implemented in [LEG&al09] and a large study showed how far locality can be pushed and what the traffic economy gain we can have.

[CUI&al09] presents the measurement study of locality-aware P2P solutions over real Internet AS topology using the accesses of nodes in PlanetLab. The authors propose an evaluation of the performances of a set of locality-aware solutions. They point out the necessity to tradeoff between the goals of optimizing AS performance and fairness among peers.

TopBT [REN&al10] for Topology BitTorrent is a topology-aware version of BitTorrent protocol also implemented as a plugin to Azureus (Vuze) client. The protocol has a discovery mechanism that allows peers to discover network proximity peers by sending requests and waiting for responses.

Propositions like ALTO and SmoothIT are mainly focusing on traffic management and QoS issues and data transfer optimization is not mentioned whereas increasing data transfer performance can have positive consequences on peering traffic and QoS agreements respect. Another drawback on these architectures is the difficulty for deployment. In fact, interoperability is not totally ensured and major modifications are needed in the Internet entities like routers. [CHF&al08] and [REN&al10] proved performance results but these solutions are not totally independent while they needed interaction with other infrastructures like CDNs. They are customer-oriented and do not propose collaboration between ISPs and P2P applications. The service provider oriented architecture we will propose in chapter 5, which is the main objective of thesis, introduces a proper mapping for AS membership which is totally independent from other infrastructure.

The ALTO project defined in an IEEE draft a specification for the called P4P [XIE&al07] that permits coordination and collaboration between an ISP in the P2P activities and applications. In this model, an entity called the *iTracker* is added in each AS and permits the communication with peers and the application *Trackers* (like BitTorrent). This proposition is more general to all P2P or overlay applications, while our proposition is specified to BitTorrent.

SmoothIT [PUS&al09] consortium also proposes an architecture relying on various criteria to evaluate the traffic management in P2P systems. However, their architecture seems complex to implement legally and technically.

2.3.3.5 Contributions on erasure codes applied to BitTorrent

A performance study is proposed on [PLA05] where the authors present an assessment for erasure codes in the wide area.

A comparison between replication and erasure codes is defined in [WEA&al02]. In data networks, the FEC is usually used at the two first OSI model layers. In higher layers some techniques such as Checksum are used to detect errors on packets. It is now possible to see encoding and decoding FEC mechanism at the application layer.

The best examples are multicast transport protocols like SRMTP [BLO&al99] or data storage where FEC is integrated to protect data against failures.

In content storage, FEC improves the system performance by distributing the encoded blocks from various sources. The principle is to use some encoded blocks to compensate the loss or the corruption of any block. The objective in this context is to minimize the failures when it is to replace loss blocks by FEC blocks. However, FEC mechanism can be integrated to an Internet protocol or application to speed up data access.

In [BYE&al99] the authors propose a FEC-based alternative for multicast distribution with a parallel access to mirror sites using Tornado Codes [COH&al02]. The data is reconstructed as soon as the customer retrieves the necessary number of blocks. This solution is applied to software distribution and presents some congestion problems.

In [LAC&al02], a solution to speed up data access in P2P networks is developed with dilution of FEC fragments over all the peers based on a data storage scheme. This permits to increase the blocks entropy and choice.

Bullet is a distributed and scalable algorithm proposed in [KOS&al03] where peers self-organize into a high bandwidth overlay mesh. In this algorithm, the peers locate and retrieve data from multiple peers in parallel. The proposal is emulated and the measurements show that compared to tree based solutions, Bullet reduces the need for peers to probe intensively.

In [MAY&al03] the authors propose a simple algorithm that allows big files to be downloaded from multiple sources in P2P. The solution proposes low handshaking overhead between peers. The interest is that when two peers have partially downloaded the resource, they can benefit from each other resource parts. The codes used are linear-time rateless erasure codes.

Current researches on block coding focus on Network Coding [GKA&al06], which is an alternative of FEC. In this mechanism, the blocks are obtained by a combination of resource held by other peers. This technique, which is a channel coding and not a source coding like FEC, is implemented by Avalanche which is the Microsoft BitTorrent-like P2P application.

Except BitCod [BIC&al07] that uses the Network Coding mechanism, no development has been proposed for Avalanche yet, even if in [GKA'&al06] the authors show that this new proposition may surpass FEC performance.

In [LUN&al06] the authors studied the block management in BitTorrent and their circulation. They conclude that the block distribution in BitTorrent is far from being optimal in terms of block frequency and that some blocks dominate the network and that others become extinct nearly. They also studied this distribution depending on the topology because blocks tend to conglomerate. Then, they propose a simple source coding mechanism to achieve a BitTorrent-like network with better performances. The coding use is the one presented in [BYE&al98].

[SPO&al10] is a recent work where the authors explain that in real world applications like real time constraints ones are affected by flash crowds because peers join and leave quickly. They propose a modification to GPS simulator [GPS] integrating LT (Luby-Transform) codes. They prove how the changes make the protocol more robust and help speeding up data transfer.

Comparing to those different works previously described, our work on erasure codes presented in chapter 4 is to provide a complete study of the FEC mechanism impact on BitTorrent in different scenarios with the variation of various parameters. It will be our goal in our global Service Provider oriented platform to allow an ISP to choose the redundancy degree (FEC ratio), depending on the running application and the peers number or availability for instance. We chose FEC instead of Network Coding because the FEC mechanism has proved more maturity until now, it is simpler and also because we prefer a source coding to a channel network coding in order to be able to control the FEC providers easily. In fact, in Network Coding, each peer is implicated to the process.

The main ways source coding can be integrated to BitTorrent is also detailed in section 4.1.

2.3.4 BitTorrent simulators

A simulator is used to simulate communications networks in certain scenarios or situations. The simulators have evolved today to be able to implement the simulation of P2P systems. The simulators such as NS2 [NS2] can be used directly to simulate a P2P protocol while a simulator as OMNET++ [OMN01] was used to produce a simulator called OverSim for P2P systems [OVER07]. The simulators can be classified into two categories: packet-based simulators and simulators application-level. The first category calculates the time, bandwidth and routing for each packet sent or used by the simulator while the second category, does not take each packet into account but the calculation of these same metrics mentioned above are made from a network point to another based on flows.

The criteria for selecting a simulator are mainly:

- The ability to flexibly implement BitTorrent and all its aspects (algorithm, protocol, messages, etc.).
- Having a global view of network.
- Having reasonable results in terms of performance compared to the reality of the network.
- Having a support and documentation for use and/or development. Provide a significant scaling factor.

Some famous simulators are used for general P2P simulations like Peersim [PEE] but we will present a non exhaustive list of simulators specific to BitTorrent protocol.

2.3.4.1 NS-2 for BitTorrent

NS-2 simulator is a famous and complete simulator for networks. For to P2P protocols, many protocols such as Gnutella have been simulated using NS-2. A basic implementation of BitTorrent [EGER AL07 &] was developed by omitting the contact part of the *Tracker* (*Tracker* HTTP Protocol). However all the algorithms are developed in this simulator including the End Game Mode often overlooked.

2.3.4.2 OverSim on OMNeT++

OverSim [OVER07] is a simulation platform for overlay networks. It uses the characteristics of OMNET++ as the GUI. What is interesting with OverSim was the opportunity to interact with the model derived from the underlay INET INET Framework used by OMNET + + and thus be able to change the settings of the current network layer and MAC. OverSim currently implements Chord [STO&al01], Kademlia [MAY&al02] and GIA [GIA03]. The media OverSim consists in a mailing list and a full documentation for installation, use and development of this simulator tuning. It can be used to simulate more than 100,000 nodes. The possibility of changing the underlying network makes it very flexible compared to other simulators. Although most protocols are implemented structured P2P DHT types, it is also possible to simulate P2P unstructured and thus implement BitTorrent. In [OMBIT09], the authors propose an integrated module to OMNET BitTorrent++ with all elements of the protocol (algorithms and messages). Three modules were created. The first one is the *Tracker*, then the Customer *Tracker* (*Tracker* module contacts to join the network) and finally Peer-Wire Protocol module implements the full functionality of the protocol transfer between different peers.

2.3.4.3 GPS

The GPS simulator [GPS] developed in Java is interesting because it offers a graphical interface and visualization nodes and exchanges. However, the number of nodes that can be introduced is limited. GPS was created to simulate P2P protocols and implementation of custom protocol, but its operation mode is based on BitTorrent. It is described as a message-level simulator that works at the application level. *Choke Algorithm* is integrated in GPS. The problem with this simulator is the missing documentation and popularity.

2.3.4.4 BTSim

BTSim [BTS] is a new implementation of the BitTorrent protocol. It is used to simplify the components of BitTorrent by emulation on a single system. The variables of the simulator can be easily changed to adapt the code itself for specific experimentations. It is written in Java and is based on BTOrig (in python). The *Tracker* and the client are two different threads. Depending on the simulation parameters, BTSim can generate different threads for the *Tracker* and the customer on the same system. Sockets created between clients to emulate the behavior of a real socket connection and even the socket created between the *Tracker* and a client. An element of the architecture emulates the algorithm Choke, and the Choker regarding downloading each customer is the downloader that is responsible for managing each connection. The problem is the lack of a scheduler for managing transfer speeds of each client. A timer element is used to emulate the data transfers.

2.3.4.5 Microsoft Research OctoSim BitTorrent simulator

MSR Simulator is a simulator that was created by the authors of [MR05] in Microsoft Research group. The authors are at the origin of the protocol Avalanche [NC'06] we will present in Chapter 4 of this work. MSR is the simulator that we have greatly modified for the implementation of error correcting codes and most of the work and performance studies of BitTorrent.

The simulator is a discrete-time simulator and represents the exchange between the peers of the BitTorrent network. The unit of exchange is only the piece. The finer granularity is the block (pieces fragments) has not been taken into account yet. The simulator models peer activity (Joins, Leaves, exchange of rings) and most of the mechanisms and algorithms of the basic BitTorrent protocol [COH03] (First rarest, Reciprocity Tit-For-Tat, *Choke Algorithm*) in detail.

The network model combines speed downlink and uplink rate for each peer, which allows modeling asymmetric access networks. Thus able to simulate heterogeneous environments change. The simulator uses this rate to calculate the approximate time of the pieces exchange between peers. This calculation takes into account the number of flows sharing the uplink or downlink connection while this number may vary. The calculation for each piece transmitted from the time depending on the flow is quite complicated to do to limit the maximum number of peers.

One *Seed* is created at the beginning by default even if that value can be changed. It is also possible to change the probability of withdrawal of the new peers become *Seeds*. In effect this means that if this value is equal to 1, *Leeches* becoming *Seeds* necessarily leave the network once the download is complete.

Despite the complexity of calculating the simple model given, above simplifications have been made in the network model:

- Every edge of the propagation delay is neglected, only indicator taken into account for sending control packets of small size (for example packages used by peers for the query pieces to their neighbors). We assume that this simplification does not really impact on the results, because the download time is dominated by data traffic (transfer of segments) and the mechanism for sending data from BitTorrent cache much of the latency Traffic control in practice.

- The dynamics of TCP connections at the packet level is not modeled. However, a link is shared by different connections equally, taking the variation in the number of connections into account. Note that the procedure Store and Forward BitTorrent is not affected by this simplification. Basically, the anomalies (TCP timeouts) are not modeled. Note that while one piece is not particularly large (32-256 KB), the flow of data is kept continuously.
- Finally, the bottleneck of shared connections within the network is not represented.
- A simplification has been made in the modeling algorithms of BitTorrent. The End Game Mode has been ignored. It is used to speed up downloading a priori at the end of the file allowing the node to query blocks they want to and that all peers in parallel. Knowing the End Game mode has no effect on the performance plan and it does not solve the Last Block problem [MR05] LEG&al06], it is better to neglect it in the simulation. We remind that the Last Block problem takes place when the peer does not find a peer with the last piece it needs and this may increase the overall download time.

The simulator we will use during the rest of this thesis work is a greatly modified version of MSR Simulator.

Chapter 3

P2P traffic partitioning: a contribution

*This chapter presents the first contribution of this thesis. Peer-to-Peer protocols and especially Bit*Torrent *do not take into consideration the importance of the large traffic volumes exchanged between Autonomous Systems (ASs). This constrains Internet Service Providers (ISPs) to restrict Bit*Torrent *connections in a way of decreasing their peering costs. The difficulty in implementing such a mechanism is that most of the P2P applications are distributed and without any real control. We propose an approach which is based on the introduction of a control entity called hTracker. The latter consists in adding a control level in Bit*Torrent *with a new peer selection policy, in order to reduce the download time and as well as the inter-ASs traffic. The principle adopted here is to fraction the traffic by using locality information and to make each peer select its neighbors relying on the AS it belongs to. We also propose a formal peerId specification calculated using an HMAC function to map each peer with its AS membership. The large-scale simulations published in* [PBL05] *show valuable results that validate our solution.*

P2P applications are widely used and generate a quite large part of traffic among the overall traffic in the Internet. Among them, BitTorrent (more than 60% of the P2P traffic [IPO07]) represents more than 30% of the Internet traffic [CAC05]. The P2P model is known for having many advantages compared to the Client-Server model: the scalability due to the good load distribution and well balanced aspect of the system, the possibility of using all peers' cumulative resources at the same time, and finally, the easy deployment of such systems.

However, a significant drawback of P2P model is that the overlay level is independent of the underlay level. This brings traffic engineering problems due to the absence of a real management for redundant and over traffic generated within an AS and especially between ASs in the Internet. An autonomous system is an administrative entity that groups a collection of IP addresses and routing prefixes under the control of one or more ISPs that share the same routing policy (and protocol in general). Some peering agreements exist for the traffic volume exchanged between ISPs [NOR03] but in general the traffic problems and their generated over cost are entirely borne by the ISPs themselves. The shaping devices techniques [BIN&al06] are interesting in a way that they do not need any modification of the *Tracker* or the client. However, these techniques imply that every ISP integrates the system alongside the edge routers of the ISPs. Another disadvantage in this method is that the *Tracker*, which is the entity that helps peers to contact each others, is not directly responsible of how a new joining peer choose its neighbors and this is against the BitTorrent protocol philosophy. The peer can also manipulate the *Tracker*'s response which presents a big risk in terms of security.

The BitTorrent protocol specifies algorithms for peer selection and piece selection that are the main keys of its success [MR05]. However, a problem still exists in the original BitTorrent client version concerning the neighbors' selection. In fact, when a new peer joins the network after downloading a desired *Torrent* file from a *Torrent* web database, it must contact the *Tracker* responsible for this torrent. The main function of this *Tracker* is to keep information on peers for each *Torrent* it manages. In response to the new peer's request the *Tracker* send a list of 50 peers. These peers, that constitute a list called the *Peers Set*, are chosen randomly from a larger list of peers having data for the specific torrent. No locality information is taken into account, as a consequence, the peers forming the *Peers Set* are generally chosen from different domains.

This random selection has consequences on the vast inter-domain traffic. A way to decrease this amount is to force the *Tracker* to send a *Peers Set* such as most of them are inside the same AS as for the peer to which the *Peers Set* is to be sent. Even if this technique increases the intra-domain traffic, the service provider can manage its own overhead and (equipment investment), the main advantage resides in the reduction of traffic passing through other ASs, and so the cost. Another advantage is the better download duration since the peers contacted are physically (in terms of Bandwidth and RTT) closer. This approach is commonly described as *locality-aware* techniques. The impact on implementing this solution in terms of modification is mostly in the *Tracker*.

Most of the current P2P architectures choose a weak control of the *Peers Set* in the sense that the latter is set randomly. Our proposal consists in an enhanced control of the peer set. We introduce into the BitTorrent infrastructure a modified *Tracker*, called *hTracker*. The main difference with a classical BitTorrent *Tracker* resides in the addition of a control level on the choice of the peers. Currently, *Trackers* are the only entities in a P2P system, like BitTorrent, through which a system level control could be provided. In our proposal, the *hTracker* will send a specific *Peers Set* (that is not randomly chosen anymore) to each *Leech*. The choice is done in order to adjust the activity of the peers following some criteria that we will discuss in this paper. One of the goals is to reduce the traffic between peers of the *hTracker*'s AS (the AS to which is attached) and the other ASs.

The second contribution is the definition of the way to map the peers with their AS. Most of the previous works propose to use BGP routers information [LEG&al09], or Internet topology maps to identify ISPs [BIN&al06]. This problem is not considered as primordial in the existing solutions in BitTorrent locality. We propose a method to generate a unique *peerId* for each peer, with a formal semantic. This *peerId* will indicate in particular the AS it belongs to. More precisely, we propose to generate part of the *peerId* with an HMAC function [HMC97]. Each AS, say *ASj*, is associated to a unique private key *Kj*.

Now, let us combine the *hTracker* and the *HMAC-based* semantic *peerId* proposals: when a new *Leech* joins the network, it forges its peerId by applying this HMAC function to its IP Address and then contacts its *hTracker* to obtain the *Peers Set*. By executing the HMAC function with the AS key the *hTracker* verifies the AS membership of the peer. This is a way for the *hTracker* to choose within the same AS most of the peers in the *Peers Set* it sends to the new *Leeches*. The semantic proposed can vary, but the idea is to integrate the AS membership to the *peerId*. The key *Kj* adds a security level when it would be possible for a peer using only its AS identification to choose any of it to spoof its membership.

In order to validate our proposal, we developed a large-scale discrete-event simulator for BitTorrent with the majority of its algorithms and policies. We can inject more than 5000 peers in a multi-domain network component based on a specific study of real propagation delays between ASs in the Internet. We validate our results by choosing two main criteria which are the completion (download) time and the traffic volume sent from an AS to another. The results confirm our design objectives.

3.1 BitTorrent and peers locality

The contribution concerns the *Tracker* and the way it chooses the *Peers Set*. In order to understand it, we present here a brief recall of the general BitTorrent functioning and its principal algorithms based on the major specifications and papers.

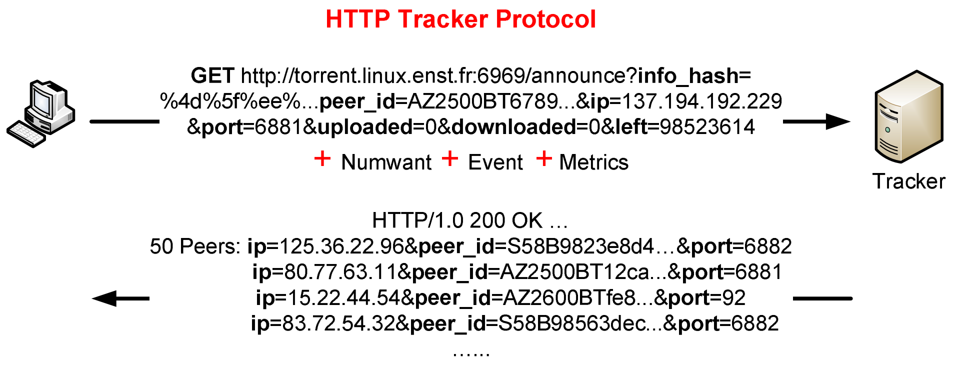
BitTorrent [COH03] is probably the most popular file-sharing protocol used in Internet for large files distribution with a very good scale factor. A metadata file called a *Torrent* is downloaded by the user from a web page. This file contains the address of the *Tracker* responsible for the resource the user is interested in and the hash results of each resource's piece for data integrity verification. The piece size in BitTorrent depends on the resource size but is the same for each piece of the same resource (usually 256 kB). Each piece is also divided into blocks (16 kB in general for a 256 kB piece). Even if the block is the transfer unit of the protocol, a user first finishes to download a piece before downloading another one. The *Tracker* is the entity responsible for keeping information on peers: those having only parts of the resource are called *Leeches* and those having the entire resource are called *Seeds* [LEG&al05].

When a *Leech* wants to download a resource, it must first contact the *Tracker* using the *HTTP Tracker protocol* [BTO] to receive a list called the *Peers Set* that contains, in the regular BitTorrent version, a random list of some peers that the *Leech* will be able to contact for downloading the pieces. The size of this *Peers Set* is by default equal to 50 and can contain *Leeches* and *Seeds*. An interval is initiated by the *Tracker* to fix the time (usually 10 min) after which the *Leech* must contact the *Tracker* again and send a report. This report contains upload and download data volumes, and the peer state. However, when the *Peers Set* size is under 20, the *Leech* must automatically contact the *Tracker* to ask for a new list. This size cannot exceed 80 in the original BitTorrent Specification [BTW]. The peers in the *Peers Set* are peers that can be scattered throughout several ASs or that are more specifically ISP domains.

The main algorithm that governs the peers choosing policy and directly the data exchange between the peers is the *Choke Algorithm* [LEG&al06]. A *Leech A* can proceed to a handshake with each one of its neighbors. The exchange between peers is based on the *Peer wire protocol* [BTO]. The contacted peer *B* sends a *Bitfield* message containing a bitmap of the pieces it has and the pieces it still needs to download. If *A* is *Interested* in one or more specific piece(s) in *B* bitmap, *A* sends an *Interested* message for the piece(s) to *B*. The contacted peer *B* can accept or refuse to send the piece depending on the *Choke Algorithm*. This algorithm determines for each peer a so-called *Active Peers Set*, which is the set of peers to whom a peer accept to send during a round (time cycle). We say that *B unchokes A* when *B* accepts to send data to *A*. If not, then *B* will *choke A*. This decision depends on the download rate of *B* from *A* when B is a *Leech*. If *B* is a *Seed*, the decision depends on the upload rate from *B* to *A* and the data amount previously sent. *B* can upload to a maximum of 5 peers where 4 peers are chosen every 10 s because they present the best download rate from *B* (if B is a *Leech* for example) and 1 peer called the *optimistic unchoke peer* is randomly chosen every 30 s to a let a chance to weak rate peers to participate.

The other main BitTorrent algorithm concerning the piece policy is the *Rarest First* policy [LEG&al06]. A requesting peer selects the pieces to download by choosing the pieces that are the least replicated or distributed among all the peers in the *Peers Set*. This mechanism contributes to accelerate the replication of rarest pieces and to increase their entropy in the network.

In Figure 3.1 we show the procedure launched by a new *Leech* to contact the *Tracker* and the response including the *Peers Set*.



```
GET http://torrent.linux.enst.fr:6969/announce?info_hash=
%4d%5f%ee%...peer_id=AZ2500BT6789...&ip=137.194.192.229
&port=6881&uploaded=0&downloaded=0&left=98523614
```

```
HTTP/1.0 200 OK …
50 Peers: ip=125.36.22.96&peer_id=S58B9823e8d4…&port=6882
          ip=80.77.63.11&peer_id=AZ2500BT12ca...&port=6881
          ip=15.22.44.54&peer_id=AZ2600BTfe8...&port=92
          ip=83.72.54.32&peer_id=S58B98563dec...&port=6882
......
```

Figure 3.1: First exchange between a *Leech* and a *Tracker*

We focused on the *Get Announce* message to analyze the message fields and especially the *peerId* specification before presenting the modification proposed in our work.

The following is a brief description of the most important fields of the *Get* message [BTW]. For more details *cf* Section 2.3.2.2.

peer_id: 20-bytes string used as a unique ID for the client, generated by the client at startup. This ID can any value, and may be binary data. There are mainly two conventions of coding client version information into the *peerId*, Azureus-style and Shadow-style. Azureus-style uses the following encoding: '-', two characters for client id, 4 ASCII digits for version number, '-', followed by random numbers. For example: '-AZ2060-'... Shadow-style uses the following encoding: 1 ASCII alphanumeric for client identification, up to five characters for version number followed by three characters (commonly '---'), followed by random characters. Each character in the version string represents a number from 0 to 63. For example: 'S58B-----'... for Shadow's 5.8.11.

port: The port number that the client is listening on. Ports reserved for BitTorrent are typically 6881-6889.

uploaded: The total amount uploaded (since the client sent the 'started' event to the *Tracker*)

downloaded: The total amount downloaded (since the client sent the 'started' event to the *Tracker*)

left: The number of bytes this client still has to download

compact: Setting this to 1 indicates that the client accepts a compact response

no_peer_id: Indicates that the *Tracker* can omit *peer_id* field in peers dictionary. This option is ignored if *compact* is enabled.

event: If specified, must be one of *started*, *completed*, *stopped*, If not specified, then this request is one performed at regular intervals. *started:* the first request to the *Tracker* must include the event key with this value. *stopped:* must be sent to the *Tracker* if the client is shutting down gracefully. *completed:* must be sent to the *Tracker* when the download completes. However, *completed* must not be sent if the download was already 100% complete when the client started.

numwant: Number of peers that the client would like to receive from the *Tracker*. This value is permitted to be zero. If omitted, the typically default value is 50 peers.

key: An additional identifier that is not shared with any users. It is intended to allow clients to prove their identity if their IP address changes.

Many studies propose performance measurements of BitTorrent protocol. In [IZA&al04] the authors show the different states in the life of BitTorrent Network. The results are collected on a five months long period involving thousands of peers. Some metrics are proposed to evaluate the performance of BitTorrent original algorithms. In fact, the neighbor selection is based on a random mechanism in this case. [MEU&al10] is a BitTorrent measurement study focusing on the data availability, the pieces integrity, the flash crowd reactions and the download performance.

In the following, we present some more concrete examples concerning traffic management in P2P systems while in section 2.3.3.4 we presented related works on locality-aware techniques for BitTorrent.

3.1.1 AURORA [ASA&al09]

Aurora is for Autonomous System Relationship-aware Overlay Routing Architecture in P2P CDNs. The CDNs using P2P networks are developed and deployed for efficient transport of content across the Internet. However, those CDNs do not consider the political and economic routing through different autonomous systems, but what is remarkable is that they consume a huge amount of network resources.

The metrics that are generally taken into account in this kind of study is the RTT and the number of hops for the establishment of peer selection algorithm and the relation between ASs is not taken into account. The protocol that determines the routing is BGP. ASs can be classified into 3 main categories in terms of traffic and economic cost:

- The Transit ASs: An AS that uses access to another AS against a load of money. We will later broach the subject of C2P (Costumer to Provider).
- The Peering ASs: a pair of ASs can directly exchange traffic between ASs and that traffic is free. However, if the traffic becomes unbalanced, a cost can be generated.
- The Sibling ASs: Many ASs belong to the same organization, even if they are managed separately from the administrative point of view. In this case, no charge is considered.

AURORA proposes an inter-domain control based on overlay routing metrics to monitor this traffic.

The cost-based method is first proposed. In this case, it performs a calculation of distances at underlay network of inter-links all ASs through the path between two ASs arbitrary. This method is not difficult to calculate. The challenge is to reveal the true cost of these links, since such information is generally private service providers. A method based on a metric magnitude precisely the problem of ASs undisclosed. The relationships between ASs can be estimated mathematically. The results show significant gains in using such a method, unlike the random selection of peers, or using the calculation of RTT or hops count. A peer selection algorithm based on these metrics significantly reduces inter-domain traffic.

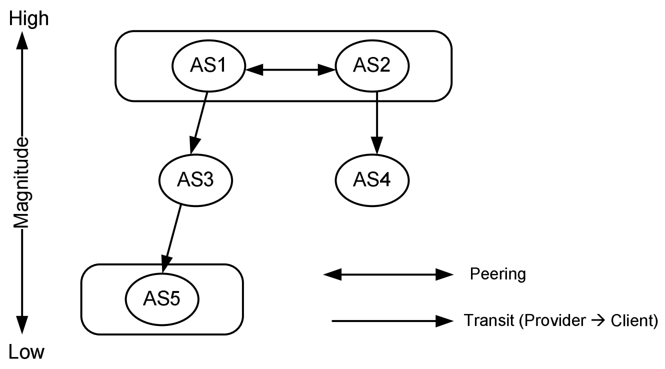


Figure 3.2: Valley-Free Path topology and AS magnitude in AURORA

3.1.2 Ono

The Ono project [CHF&al08] is a software service that allows clients to efficiently identify peers that are physically close. A pluggin is added to Azureus client taking pressure off international and other long distance transfers, to increase file download speeds. Ono seems to increase download rates by between 31% and 207% on average, depending on whether the client is on an overloaded network or with large available bandwidth. The origin of this work is Aqualab that made Ono available as an open *Tracker* and with the source open. Ono is open source and does not require additional infrastructure. To determine which peers are close by, Ono learns from existing Content Distribution Networks (CDNs) such as Akamai [AKA] and Limelight. It assumes that if tow hosts are sent to the same CDN server, they are likely to be close to each other. Contrarily to other propositions like P4P, Ono does not need any cooperation between ISPs and P2P applications.

3.1.3 TopBT

TopBT for Topology BitTorrent is a topology-aware version of BitTorrent protocol also implemented as a plugin to Azureus (Vuze) client. The protocol has a discovery mechanism that allows peers to discover network proximity peers by sending requests and waiting for responses. TopBT is developed with the main objective of reducing unnecessary traffic and maintaining at the same time the download speeds. The authors show in [REN&al10] that they were able to decrease 25% of the unnecessary traffic. The principles are based on discovering path proximity with some probes using TTLs.

3.2 *hTracker*: Management and Control on a BitTorrent network

3.2.1 Problem Statement and Objectives

The global Internet routing system is composed of various Autonomous Systems (ASs). To route traffic in the Internet ASs maintains relationships between them. Peering and business agreements are the base of those relationships. The major categories of those relationships are customer to provider (c2p), peer to peer (p2p) and sibling to sibling (s2s) [DIM&al07].

The problem is that inter-ASs connections number has an impact on peers performance and inter-domains traffic. This impact is multiple and concerns many aspects [LEG&al09].

The first impact is in the overhead that increases linearly with the inter-ASs connections number. This can cost a lot to the ISP, especially if the traffic cannot be controlled like in P2P applications. We note that in general, traffic retrieved from an external AS have to be paid somehow by the origin AS. Usually, large ASs do not pay like smaller ASs because they represent the crossroads of so many traffic that they have agreements allowing them to have facilities with ASs that are at the same level or inferior levels. However, when an AS is of relatively middle or small size, they may have to pay the bill when they have to use bigger ASs traffic roads.

The second impact is the slowdown and this problem can be avoided by choosing peers located in majority within the same domain, in order to limit the propagation delay during exchanges as well as inter-domain traffic. A complementary important aspect is to have fast initial *Seeds* providing high pieces diversity.

Our study is based on some objectives that an ISP could fix to serve its clients with respect of the Quality of Service in the contractual engagement signed between the Service Provider and the customer. At the same time, ISPs are all under the terms of what we call peering strategies [PER08] that they have to respect. The principal goal for ISPs is to limit the over costs of their systems without degrading the service provided. Some applications widely used nowadays have some constraints on delay time and bandwidth. Applications like VoIP or IPTV are the best examples that illustrate the importance for an ISP to follow some objectives and to add the technical and policy measures to reach these objectives. However, it is not so obvious and easy in P2P applications that are widely distributed, dynamic and control-less to preview the behavior of every peer in the system.

Our interest in this work here is to treat this locality issue even if we are conscious that the locality parameter is not the only aspect that an ISP must take into consideration to respect its peering and QoS objectives. The original and most used method to control P2P traffic (and especially BitTorrent) by ISPs is to limit the bandwidth by throttling. Devices like [PCK] are used to shape the traffic in the edge routers. However, the downside of these devices is that they slow down the data transfers and this does not solve the problems of improving the locality of each peer by ISPs; thus cannot participate to decrease the inter-domain traffic. We propose to change the *Tracker* peer selection policy for a new one. Like in [BIN&al06] and [PAP&al06], the majority of the chosen peers are picked up from the same AS as the *Leech* that sends the request. This method, validated by simulations, permits to considerably reduce the inter-domain traffic and the completion time. In previous propositions, the mapping of peers with their AS was not defined precisely. Another good point in this chapter is the definition of this mapping with the *peerId* of each peer.

3.2.2 The *hTracker* peer selection policy

The global BitTorrent network is formed by thousands of peers geographically distributed among different ASs in the Internet. The *Tracker* entity has no specific domain membership. When a *Leech* contacts a *Tracker*, it can't preview that this *Tracker* belongs to the same AS. In the generic BitTorrent specification the *Tracker* sends a *Peers Set* where peers are chosen with a total random manner. Implementing the *hTracker* proposition add a control level so that the *Tracker* contacted by a new *Leech* is the *hTracker* in the AS this *Leech* is member of. Each AS is associated to a key *Kj* that identifies the domain. An *hTracker j* in the *ASj* must send to every new *Leech* a *Peers Set* with a default size of 50 peers.

The new *hTracker* policy consists in choosing *50 - k* peers that belong to *ASd* and *k* peers that are outside *ASd*. The parameter *k* is variable and depends on the ISP objectives and peering agreements. The less the *k* value is, the less *ASd Leeches* will contact peers that are outside *ASd* and the less inter-ASs traffic will be generated. If *k* = 0, the partitioning is maximum and none of the peers are chosen outside the ISP domain. The value of *k* must be set by the ISP that have to take into consideration the number of possible sources in his AS for a specific resource. In fact, if we suppose that for a given AS *k*=1 and that none of the sources is available outside of this AS, the download will be penalized. At the same time if *k* is close to the maximum (50), the ISP must be sure that all sources inside the AS are available.

The Figure 3.3 represents an example of 4 ASs where 3 ASs use the *hTracker* locality mechanism by choosing peers within the same ISP while the other *AS4* use the traditional BitTorrent implementation and chooses randomly the peers.

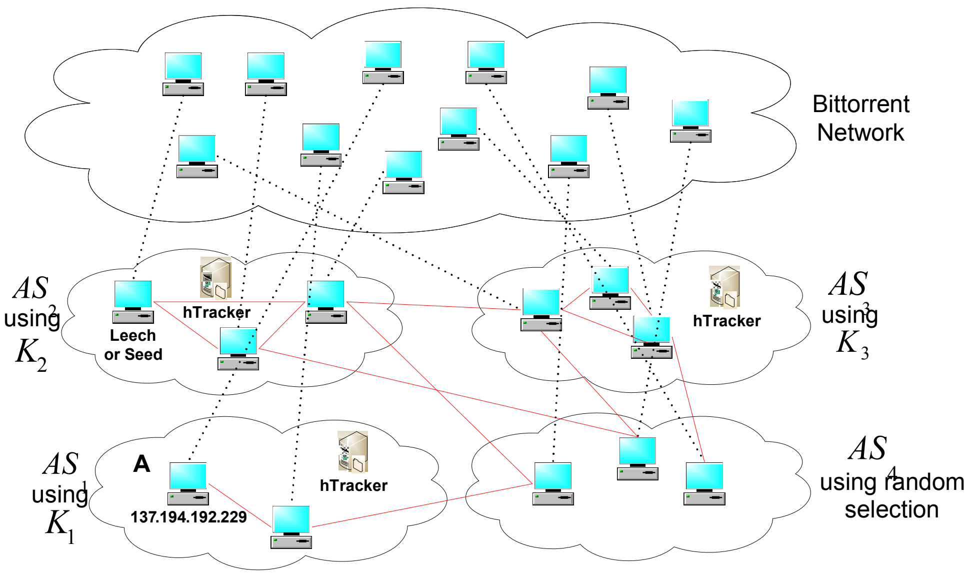


Figure 3.3: Graph representing an example of a 4 ASs BitTorrent Network including 3 ASs using *hTracker* solution and one generic AS.

3.2.3 Mapping the AS membership with the *peerId*

It is primordial in our solution to specify a formal way to map the peers with their AS. We use the HMAC function [HMC97]. It is a mechanism for message authentication using cryptographic hash functions and the function used can be any one of the known hash functions. Usually, HMAC is used to preserve the messages integrity transmitted over a network. The Hash function *H* is associated to a key *K*. The result of HMAC function using SHA1 Hash function is a 20-byte output. This corresponds to the *peerId* field size. The information that is used to generate the *peerId* is the *Leech* IP address. In P2P protocols some applications prefer anonymity, so we propose to make this field mandatory and not optional. If it is not possible the ISP can choose to use the HMAC function on information different from the IP address. However, this one is still the more practical information for calculation. The *Leech peerId* generation process must be changed in the client to the novel *HMAC* generation technique. This *HMAC* function is the same used by the *AS1 hTracker* with *K1* to verify each peer membership. This is how the *hTracker* will choose the peers that are within the *AS1* and those that are outside.

For some security reasons and to add an authentication mechanism in our architecture, we decide not to choose the key *Km* as the key used by all members of the *ASm*. This would facilitate attacks knowing that a unique key is much easier to have especially that the number of P2P members in an AS is usually large. So we choose to use *Km* with the HMAC function to generate a key *Kd* for each peer present in the *ASm*. The key *Km* is not known by the peers and is the private AS key. When a peer joins the AS, it sends its information to the *hTracker* that generates the key *Kd* by using the same HMAC function applied to the peer IP address with the private key *Km*. The peer generate the *peerId* by applying the HMAC function to the IP address using the *Kd* peer key. The *Kd* must be sent to each peer by the *hTracker* via a secure link.

We compare in Figure 3.4 the Azureus style method to our method. We kept the first 8 bytes for transparency with previous clients and interoperability. The *Trackers* keep gathering the statistics information process. In fact the *Client_info* field informs the *Tracker* on the client and its version. The remaining bytes that were randomly generated are formalized to keep an information on the AS membership with the ASN and a calculation of the last 8 bytes with a private key shared between the *Leeches* and the *hTracker*.

Without this security level, it would be easy for any *Leech*, just with the ASN, to pretend being a member of a specific AS even if it is not. The generic *peerId* using Azureus style would be as follow: ***peerId(A)= -AZ2060-256f9ec43a77***

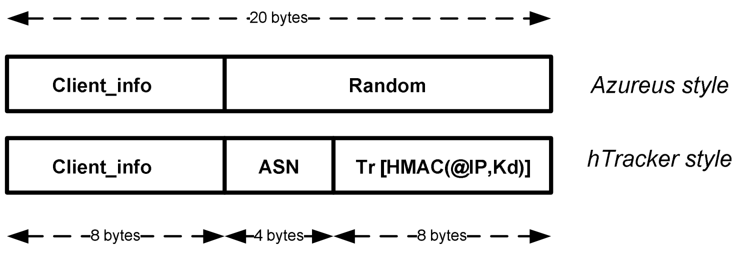


Figure 3.4: *hTracker peerId* specification

Using the HMAC function on the peer A IP address we have for example the following for the generation peer A key *KA* and *peerId(A)*:

HMAC(137.194.192.229,K1)=KA

HMAC(137.194.192.229,KA)=1e43f557d89ac69e49

This result can be truncated by a specific method without losing the entropy and with a very low collision probability [PRE&al95]. The truncation result is not formed by the same HMAC result digits. So we have:

Truncation [HMAC(137.194.192.229,KA)]=a4b5f633

We can take the example of the France Telecom AS, which has the *AS Number (ASN)* 5511 (1587 in hexadecimal). ASN were specified [19] in 16 bits then extended to 32 bits (4 bytes). We finally have in *hTracker* style:

peerId(A)= -AZ2060-1587a4b5f633

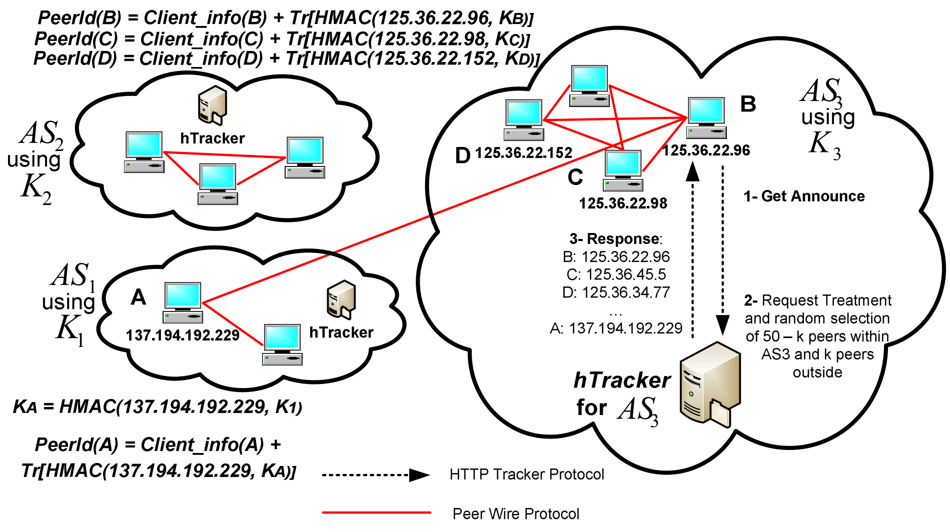

PeerId(B) = Client_info(B) + Tr[HMAC(125.36.22.96, KB)]
PeerId(C) = Client_info(C) + Tr[HMAC(125.36.22.98, KC)]
PeerId(D) = Client_info(D) + Tr[HMAC(125.36.22.152, KD)]
AS2 using K2
hTracker
AS3 using K3
B
D 125.36.22.152
C
125.36.22.96
125.36.22.98
1- Get Announce
3- Response:
B: 125.36.22.96
C: 125.36.45.5
D: 125.36.34.77
...
A: 137.194.192.229
2- Request Treatment and random selection of 50 – k peers within AS3 and k peers outside
AS1 using K1
A
137.194.192.229
hTracker
hTracker for AS3
KA = HMAC(137.194.192.229, K1)
PeerId(A) = Client_info(A) + Tr[HMAC(137.194.192.229, KA)]
HTTP Tracker Protocol
Peer Wire Protocol

Figure 3.5: *hTracker* policy scenario example

In Figure 3.5, we represented the network with 3 ASs that use our solution. The *Leech B* contacts the *hTracker* by sending a *Get Announce* message. In our solution, the IP Address field which is usually optional in the generic BitTorrent solution is forced to be given by the *Leech*. The *hTracker* can retrieve the IP address of this peer in its AS but we decided this to eliminate any routing or NAT problems that could prevent the *hTracker* to calculate the *peerId* with the *Leech* IP address.

The *hTracker* verifies if *Leech B* is in the same Autonomous System by calculating *peerId(B)* with the key *K3* and the same formula. If the authentication is verified, it sends a *Peers Set* with *k=1*, and *B* receives in this list containing 50 peers within *AS3* and 1 peer outside *AS3*, which is the peer A in *AS1*.

3.3 Simulations and results

The discrete-event simulator we developed in C# models peer activity (joins, leaves, pieces exchanges) and the majority of BitTorrent mechanisms (*Rarest First* , *Choke Algorithm*, etc.). Each peer is associated to a download rate and an upload rate that can be set. It is possible to initiate some percentage to have different peer types (depending on its download and upload capacity). This allows us to model homogeneous and heterogeneous networks. A delay calculation is modeled in the simulator to take into account the number of connections that share the different links. Some simplifications have been undertaken like the non-consideration the dynamics of TCP connections and TCP timeouts or interior network bottlenecks. However we consider that a link is shared equally with each connection. The *End Game Mode* [BTO] is not in the sense that this algorithm is launched at the really end of the download and does not really prevent from the last block problem. This simplification does not affect the BitTorrent protocol measures. The peers exchange pieces and the block transfer is not modeled knowing that in BitTorrent a piece has to be downloaded entirely before another piece download is launched. So we decide to integrate the FEC mechanism in our simulator at the piece level.

An important contribution in the development of this simulator compared to several other simulators is that we modeled the network propagation delay even if the download time is dominated by the pieces exchange transmission. This propagation delay component was primordial to simulate a large-scale network with different ASs.

The method we used to add this element to our simulator was a realistic study by taking real delay propagation in the real Internet ASs. We based our calculation on the CAIDA [CLA&al99] statistics and by using a performing traceroute utility from the PingER project [PIN] to provide a table that gives the better picture of current propagation delays in the Internet from our Paris-located lab. In fact, we had to consider from the best to the worst possible RTT, modeling a relevant set of ASs. The measures have been taken several times at different hours of the day to take into consideration the traffic variation because of the time difference between France and other countries. Table I shows the results classified by each AS's rank, referenced in [CAI].

We simulate a homogeneous network and a heterogeneous network. Our results are validated by choosing two main criteria which are the completion time and the traffic volume sent.

Table 3.1: Internet Autonomous Systems RTT measurements

Autonomous System	ASN	Rank	Domain Name	IP Address	RTT (ms)
GBLX Global Crossing Ltd. (US)	3549	1	globalcrossing.com	207.218.55.80	102
AT&T WroldNet Services (US)	7018	8	att.com	12.122.2.125	111
TINET-BACKBONE Tinet SpA (DE)	3257	14	tinet.de	82.165.73.172	23
SWISSCOM Swisscom Ltd. (EU)	3303	22	swisscom.ch	138.190.35.25	251
OBIT-AS Obit Telecommunications (RU)	1299	26	obit.ru	85.114.2.98	147
Bell Canada (CA)	6539	47	bell.ca	207.35.184.46	96
OPENTRANSIT France Telecom – Orange (FR)	5511	87	opentransit.net	193.251.151.57	5
VNN-AS Vietnam Posts and Telecommunications (VN)	7643	168	vietnamnet.vn	203.162.71.74	388
REANNS-NAT-AS-NZ National research Network (NZ)	38022	187	karen.net.nz	203.89.182.33	289
KREONET2-AS-KR Korean Institute of Science and Technology Information (KR)	17579	198	kreonet.re.kr	134.75.30.253	444
IDM Autonomous System (LB)	9051	2707	idm.net.lb	193.199.135.50	232
STEL-AS-AP SamoaTel (WS)	17993	7610	samoa.ws	123.176.73.5	487
Comores Telecom (KM)	36939	21621	comorestelecom.km	80.231.195.5	664

3.3.1 Homogeneous network

We dressed in Table 3.2 the parameters chosen for a homogeneous network simulation. We simulated the behavior of the generic BitTorrent protocol in a network composed of the 13 ASs given in Table 3.1 and their propagation delays. For a generic BitTorrent protocol the parameter *k* is omitted and the peers chosen are totally random while in the *hTracker* solution, *k* can take the value 1, 4 or 12 over the total number of 50, depending on the autonomous system objectives. A Service Provider should fix a unique value of *k* but this is not a must. In fact, for some resource which the pieces are rare, *k* may be variable depending on this resource or the *Leech*. That is why we chose to diversify the *Leeches* by associating a different value of *k* (1, 4 or 12) equally and uniformly among them.

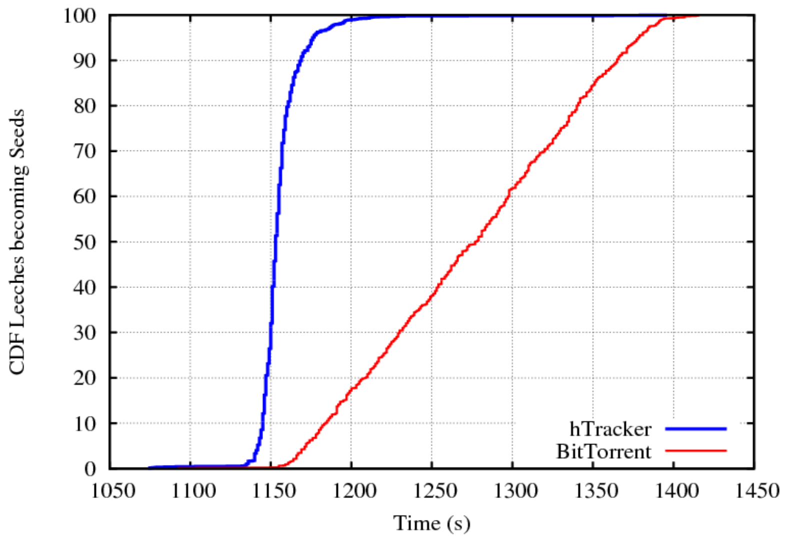
Table 3.2: Parameters for homogeneous networks simulation with *hTracker* solution

Parameter	Value
Number of *Leeches*	1000
Leeches Capacities (Down/Up)	800 kbps/400kbps
Number of initial *Seeds*[1]	1 or 20
Seeds Upload capacity	1500 kbps
File Size	100 MBytes (819200 kbits)
Seed leaving probability	1
Leech abort probability	0
Peers Set size	50
k (peers from other ASs)[1]	1, 4 and 12
Unchoked Connections per peer	5 (4 regulars and 1 optimistic)
Number of Autonomous Systems	13 (*cf.* TABLE I)

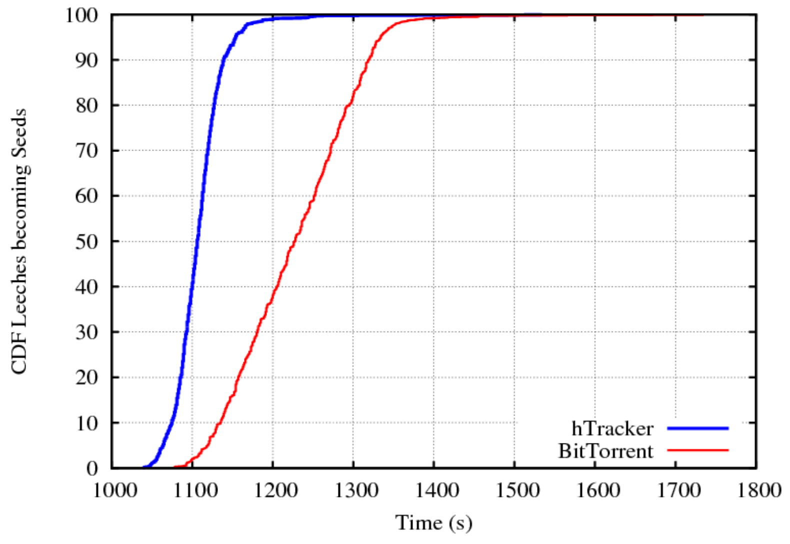
[1] Two simulations: one with 1 initial *Seed* and one with 20 initial *Seeds*

[2] Only in the *hTracker* simulation case, while in the generic BitTorrent *k* is omitted

We simulated a case with 1 initial *Seed* and a case with 20 initial *Seeds* to identify the impact of the initial *Seeds* numbers in homogeneous and heterogeneous systems for our solution comparing to BitTorrent. The results for completion times are shown in Figure 3.6 and for packets sent from our AS to other ASs in Figure 3.7. The CDF is the Cumulative Distribution Function corresponding in our case to the percentage from 0 to 1 of the *Leeches* that become *Seeds*.



(a)



(b)

Figure 3.6: Completion times CDF for BitTorrent and *hTracker* with 1 and 20 initial *Seed*(s) in a homogeneous network

By analyzing Figure 3.6, we see a real difference and a performance gain between BitTorrent using *hTracker* and the generic BitTorrent protocol. We verified this difference for (a) and (b) using an average statistical test with an error probability equal to 5%. This validates that the difference observed in the graph is real and not random. For space reasons and because this is not the scope of this paper, we will not detail the statistical average test process. This process is used for the next results.

We can note that increasing the initial *Seeds* number performs the completion time especially at the beginning of the download process. This is due to the characteristic of the *Seed* to have the entire pieces.

In Figure 3.7, we see that there is a large gap between the two graphs. The more important variation in the regular BitTorrent case is due to the variation of *k*.

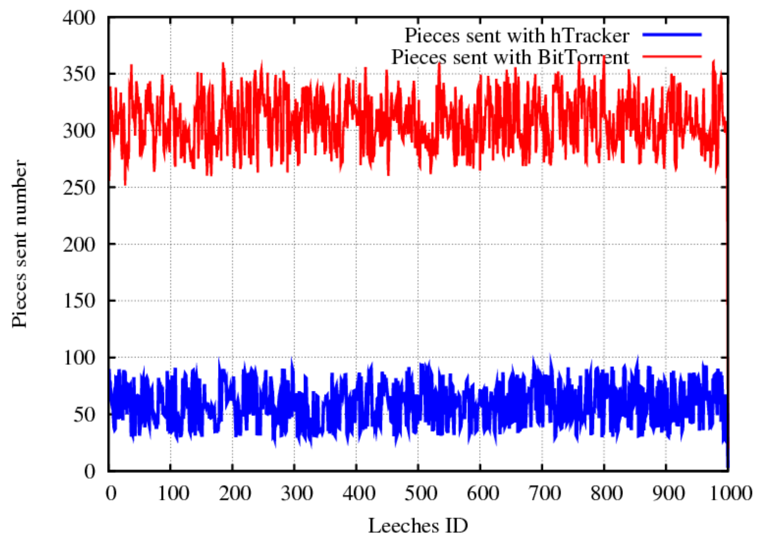


Figure 3.7: Packets sent to other ASs in a homogeneous network for both solutions with 20 initial *Seeds*.

The importance of next case is that most of current networks are composed of nodes with different bandwidth capacities that are unpredictable by an ISP.

3.3.2 Heterogeneous network

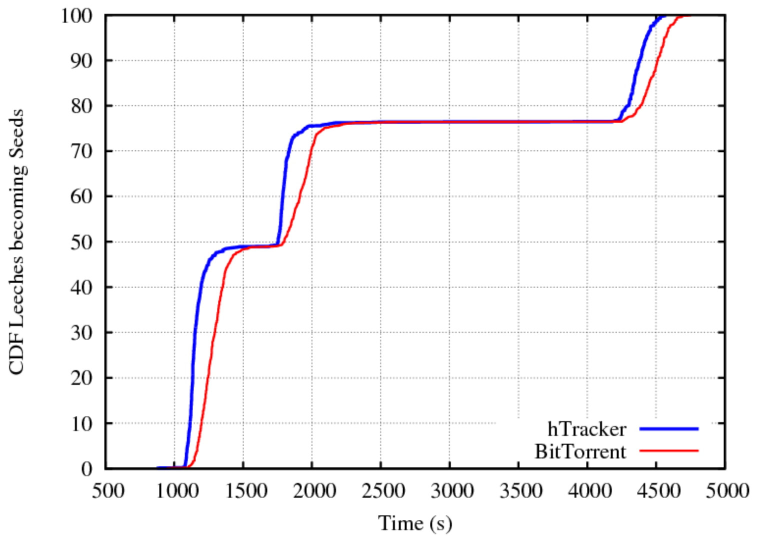
Table 3.3: Parameters for heterogeneous networks simulation with *hTracker* solution

Parameter	Value
Number of *Leeches*	1000
Leeches Capacities (Down/Up)	25% of 200 kbps/100kbps
	25% of 500 kbps/200kbps
	25% of 800 kbps/400kbps
	25% of 1000 kbps/500kbps
Number of initial *Seeds*[1]	1 or 20
Seeds Upload capacity	1500 kbps
File Size	100 MBytes (819200 kbits)
Seed leaving probability	1
Leech abort probability	0
Peers Set size	50
k (peers from other ASs)[1]	1, 4 and 12
Unchoked Connections per peer	5 (4 regulars and 1 optimistic)
Number of Autonomous Systems	13 (*cf.* TABLE I)

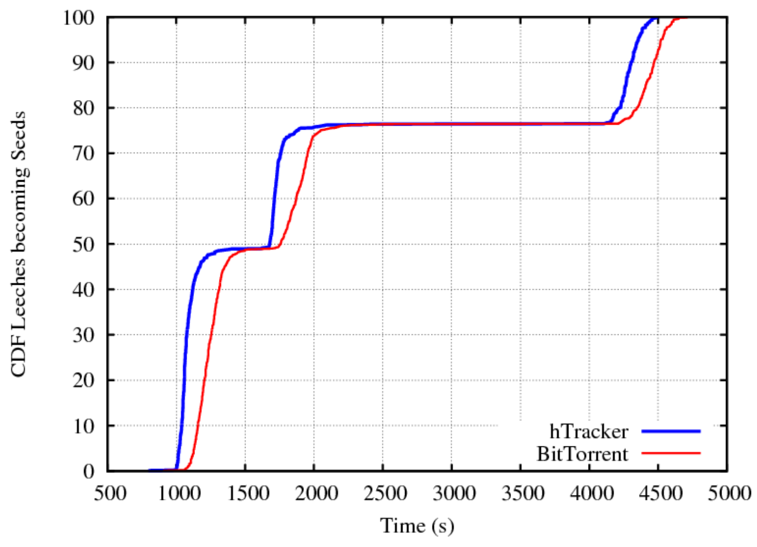
[1] Two simulations: one with 1 initial *Seed* and one with 20 initial *Seeds*
[2] Only in the *hTracker* simulation case, while in the generic BitTorrent *k* is omitted

We varied the *Leeches* capacity and we instantiated 25 % of each *Leech* type (4 types are chosen, *cf.* TABLE III). The other parameters are kept without any changes.

We note some levels in the completion times graphs (Figure 3.8) due to the various *Leech* types placed. These steps are more visible in the case of 20 initial *Seeds* where the faster *Leeches* (1000kbps/500kbps) can differentiate themselves easily.



(a)



(b)

Figure 3.8: Completion times CDF for BitTorrent and *hTracker* with 1 and 20 initial *Seed*(s) in a heterogeneous network

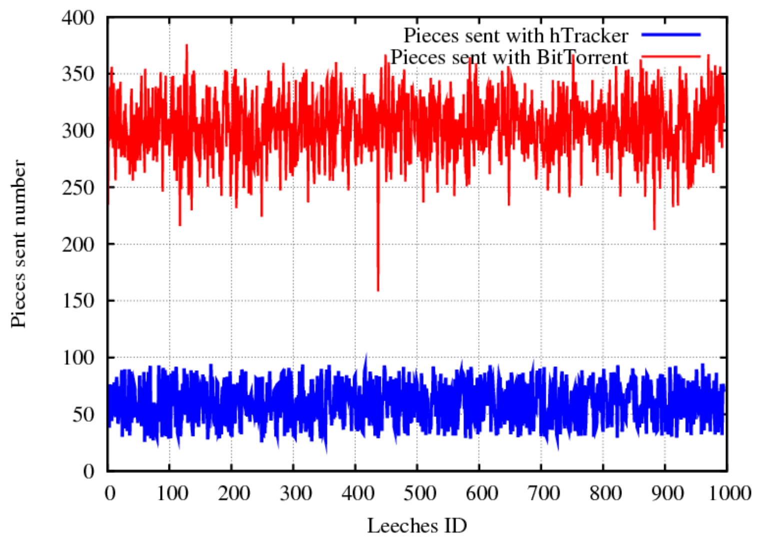


Figure 3.9: Packets sent to other ASs in a heterogeneous network for both solutions with 20 initial *Seeds*.

Concerning the packets number sent by the *Leeches*, the gap is approximately the same (Figure 3.9). However, the regular BitTorrent graph presents a smaller minimum than in the case of a homogeneous network.

Our main objective was to prove that our solution reduces two principal metrics: the completion time and the inter-domain traffic. For the first one, the gain is more relevant in the homogeneous network. The performance increase is between 10 and 20 % depending on the file size and the *Leeches* and *Seeds* number. We voluntary took a difficult case where the *Leeches* number is important and the portion of *Seeds* is small compared to the *Leeches* number. Concerning the packets sent to other domains the results are relatively similar in both networks. In fact, studying the packets sent from our AS to other ones by simulation gives an important evaluation of the inter-domain traffic.

The basic key of BitTorrent exchange is the *Choke Algorithm* based on the reciprocal *tit-for-tat* policy. When a peer chooses its *Active Peers Set* regularly from its *Active Peers*, it implicitly chooses the peers with which it will be in contact in both ways. This is due to the way BitTorrent system becomes steady by making peers exchange in a reciprocal manner. We saw that the packets sent are decreased from 2 to 6 times. This implies a diminution of the inter-ASs traffic, hence a decrease of the over costs for the ASs.

The locality parameter treated in this paper is not the only one that directly impacts the performance of the *Leeches* but it is a major element for inter-domain traffic. We can note that partitioning with *hTracker* implies an increase of the internal traffic in each AS. We consider that this is not a problem and that saving a part of the over costs cannot be done without a supplementary cost elsewhere. There is an economic compromise to be done.

We saw that our solution proposes an interesting way to decrease the over cost but a drawback persists in our strategy. When *k* is chosen, a client may be forced to use peers presenting bad capacities inside the same AS despite that he can find better ones in other ASs. This issue can be solved by integrating a more performing statistics report procedure for the *hTracker*. This is a part of our future work.

We proposed a technique based on the *HMAC* function to map a peer with its AS. The key does not play any direct role in locality but it is a determinist and robust way to map the peers with their AS and to add a security mechanism to our solution. Indeed, the peers keys derived from the AS key are unique.

3.4 Conclusion

While P2P is becoming more popular, the IP traffic that is crossing between ASs increases abundantly. This implies a need for ISPs to cooperate with P2P applications for adding a control level that is usually absent in this paradigm. In our paper, we propose a scheme to improve the BitTorrent locality by introducing an entity called the *hTracker* and a way to associate a peer with its AS using HMAC function. These two contributions permit to move towards a Service Provider oriented P2P architecture where ISPs manage the peers they serve with a semantic association included in the *peerId* definition.

Taking into consideration real applications constraints like IPTV and following ISP objectives, it is important to discover some other aspects to improve other P2P components like the data transport or even the routing process. The next chapter focuses on performing error correction in BitTorrent and in order to achieve a more flexible and complete solution.

Chapter 4

Evaluating FEC mechanism on BitTorrent protocol: Measures and analysis

Various applications are using BitTorrent-like protocols to deliver the resource and implement techniques to perform a reliable data transmission. Forward Error Correction (FEC) is an efficient mechanism used for this goal. This chapter proposes a performance evaluation of FEC implemented on BitTorrent protocol. The same simulator has been modified to evaluate the improvement of integrating this mechanism depending on many factors like the Leeches/Seeds number and capacities, the network nature (homogeneous or heterogeneous), the resource size, and the FEC redundancy ratio. The completion time metric shows that FEC is a method that accelerates the data access in some specific network configurations. On the contrary, this technique can also disrupt the system in some cases since it introduces an overhead. This contribution has been published in [PBL07].

This chapter proposes a performance evaluation of Forward Error Correction (FEC) implemented on BitTorrent protocol. The same simulation framework has been modified to implement FEC and to evaluate its improvement depending on many factors like the *Leeches/Seeds* number and capacities, the network nature (homogeneous or heterogeneous), the resource size, and the FEC redundancy ratio. The completion time metric shows that FEC is a method that accelerates the data access in some specific network configurations. On the contrary, this technique can also disrupt the system in some cases since it introduces an overhead.

BitTorrent is implemented in various applications. In [LEG&al05] and [IZA&al04] relevant measurement studies show the protocol recognized performances that are theoretically revealed in works like in [QIU&al04] or [GUO&al05]. Despite these characteristics and like in all P2P protocols some significant drawbacks exist. These issues arise because the overlay level (P2P control) is independent of the underlay level (data transport over the internet). This brings traffic engineering problems due to the absence of a real management and control of the peers especially when the applications require compliance with some QoS (Quality of Service) constraints.

A major problem in BitTorrent is at the transport level and concerns the resource pieces availability. Even if the *Local Rarest First* policy [LEG&al06] ensures good pieces entropy, it is nonetheless true that in some configurations BitTorrent network can experience a lack of resource and clients would hardly and slowly download the desired content. Just as scalability, robustness and fault tolerance, performance in content access over BitTorrent networks is a major issue for users because downloading the entire resource with the minimum completion time is their principal objective.

One of the approaches consists in using replicated copies [ABE&al02] [PLX&al97] that can be downloaded from a specific server. However, a better alternative for this technique is to provide some pieces that are not exact copies of the origin resource bulks. Instead of sending real copies, the FEC mechanism is an optimized and efficient substitute of other redundancy approach to speed-up the resource access in content distribution networks. The particularity of FEC is that the redundant blocks are coded from the origin ones so that any coded blocks can replace a block of the origin resource. Basically, FEC techniques are used for information error protection and transmission losses [RIZ97], for example in Wireless transmissions or in physical data storage (e.g., CD, DVD, RDRAM). However, in the context of content distribution, FEC can be used to speedup data in different servers or mirror sites [BYE&al99] [LAC&al02] especially in P2P networks where the storage is distributed [ABE&al02].

In distributed systems like P2P ones the use of FEC techniques is addressed differently. Indeed, the major point that differs from classical networks is that the number of clients in these systems is large. The quantity of data exchanged can easily reach important values and so the design of such a system becomes a real challenge. We know that implementing a download accelerator mechanism and a system that increases the piece availability must be done without degrading the generic system and by keeping the system interoperable with classical protocols.

We will present a detailed measurement study based on large-scale simulations to evaluate the real advantages of FEC implementation on BitTorrent protocol. We will show with which types of scenario the protocol experiences a considerable gain and on the contrary in which cases the system can quickly be degraded by an overhead that implicates important latency during downloads. This work is a study that permits to appreciate the impact of FEC coding in BitTorrent. These results can be then relatively generalized to other P2P content distribution protocols.

A related work presentation on erasure codes applied to BitTorrent was given in Section 2.3.3.5.

4.1 Error Correction mechanism

The FEC (Forward Error Correction) policy is to encode a data message composed by *k* packets in *n* packets knowing that it is possible to reconstruct the original message of *k* packets by receiving any of any the n provided. The number of packets sent is increased by a factor of *n/k*. In the case the losses are still below the rate of redundancy injected into the network, the retransmission procedure can be completely avoided. In fact, the FEC main goal in today's applications is to replace costly retransmissions. Correcting codes used today can be divided into two categories: block codes that operate on blocks of data and independently on each block and convolutional codes where transaction is on the blocks but the result depends on the current message, but also the *m* previous blocks. A famous example of block codes is the Reed-Solomon code which is a linear cyclic code. The encoding is systematic: part of the entry is found at the output of the encoder. The operation is performed on message blocks of *k* information bits each. A message is represented by a binary *k-tuple u (u1, u2, ..., uk)*. The total number of messages is 2^k. The encoder transforms each message *u* independently in *n-tuple v = (v1, v2, ..., vn)*, called the code word. The k elements of the code word *v* are identical to the original message *u*. The receiver is able to reconstitute at least *k* of *n* elements of code words.

In the BitTorrent case we consider the larger parts of file for a default piece size of 256 KB. In fact, even if the block is the transfer unit in the protocol the Strict Policy (*cf* section 2.3.1) makes us choose the piece as the transfer element on which we apply the FEC mechanism. Note that the introduction of coded segments is transparent to the *Choke Algorithm* peers with which peer performs a data transfer. The two main limitations of Reed Solomon codes are the small block size and coding\decoding time. The parameters *k* and *n* are playing very important roles also in the overall effectiveness:

- A great value of *n* is favorable because it reduces the probability that packet is replicated at the receiver but increasing it is increasing automatically the number of packets in the network and this can represent an important undesirable overhead.
- A great value of *k* increases the correction capability of FEC code. A redundant packet can recover an error only in a block to which it is attached. The correction capability is inversely proportional to the number of blocks forming a portioned file. Ideally files should be encoded in a single block.

- Reed Solomon codes are limited by Galois Field size. For example, for a matrix 2^8 with $n \leq 256$: for a packet size of FEC 1kB and producing as many redundant packets as packets of origin *n = 2k* and operating on blocks with a maximum size of 128 kB, any file exceeding this threshold must be portioned. Using a matrix 2^{16} solves this problem because $n \leq 65536$ but the disadvantage is that the coding\decoding time is very high in this case.

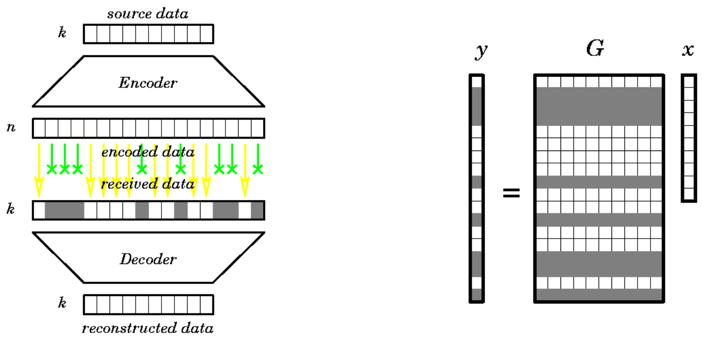


Figure 4.1: GF Matrix in FEC

We can note that the main works presented shows that the End Game Mode is often neglected in the simulation. In fact, this mode is activated at the end of the download during which a peer floods queries to all peers for all remaining pieces. However, this brings overhead and the mode is effective only if the pair is exceptionally at low download speed. The results show that except for this scenario End Game Mode is not very useful.

4.1.1 FEC at the end

The work in [MR05] presents a complete study of BitTorrent on the basis of simulations and shows an incremental work on FEC mechanism integrated to BitTorrent at the piece level. We consider a small number of peers that have already downloaded the vast majority of the pieces just before becoming *Seed* and the left *Peers Set.* We can imagine that at a specific time when the network is overloaded and that exchanges are important, they decide to join the set. They are seeking as some specific parts, while most nodes are looking to all pieces.

The authors took the example of 1,000 peers that join the set the first ten seconds. A node is then added every 200 seconds. Each node is Pre*Seed*ed with a random selection of pieces k% (k% of pieces received entirely). Ideally, these peers have more than *(1 – k<%) T* time to spend to download the remaining songs, T is the time to download the entire file (in this case T = 2000 seconds). However, in practice this time can be more important because the remaining pieces must be found. In this case, a *Seed* with an upload of 6000kBps is selected and injected into the network at least one copy of each piece. The less the number of pieces remaining to download is the more download time is. The first reason is that each piece takes a while before being sent to the *Seed* network peers indirectly given by the maximum uploads competitor number (5). The second reason is that a pre*Seed*ed peer searches for specific pieces and wants it to be replicated quickly. However, the *Rarest First* ensures that all pieces are equally replicated and therefore none is rare. This resource sharing pieces decrease the distribution rate for specific pieces of the desired Pre*Seed*ed peers involving significant download time.

The use of FEC and the injection of a large number of encoded pieces in the equivalent system allow Pre*Seed*ed to have a large selection of pieces to download and the loss of download time is avoided. Note that the injection of additional pieces does not waste bandwidth in *Seed* since each unique piece sent by the *Seed* is equally useful for everyone. By repeating the experiment with an additional contribution of 100% FEC, a significant improvement is shown in download time. At 95% of the download it goes from 5 (without FEC) to 2 (with FEC) for the download time.

This work is interesting but not complete concerning the FEC mechanism integration, even if the presentation of BitTorrent functioning and basics are very well studied and evaluated. An evolution of this paper is presented in next paragraph. The same authors propose a different coding that is not at the source but provided by the network peers themselves. This contribution is the Network Coding.

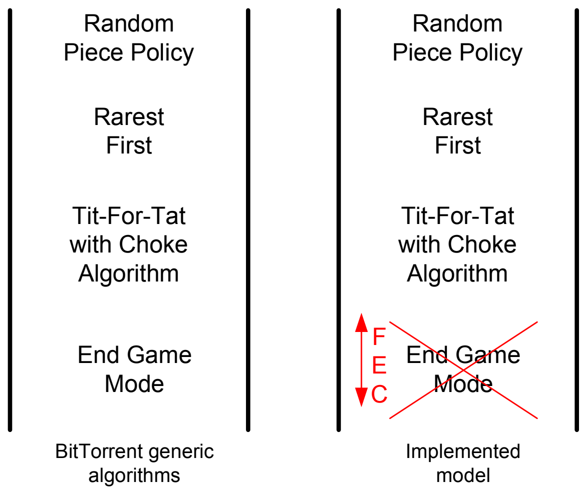


Figure 4.2: FEC at the end

4.1.2 Network Coding

This is a proposal for a new scheme for large content distribution based on Network Coding [GKA&al05]. All peers can send encoded blocks of data to allow more efficient dissemination. In FEC mechanism only some specific peers (some *Seeds*) where performing source coding. In large networks and unstructured overlay, peers need to make decisions on sending data on the node local information only. The work compares coding techniques at the peers of the network to those where the coding is done only at the level of the source. They show a gain of 20-30% with network coding compared to coding at the source like FEC on the download time and better yield of 2 to 3 times compared to normal data transmission without coding.

In this type of study, there are many parameters to consider. Indeed, there is a need for scalability and bandwidth increases considerably at each arrival of a new peer. Requests and transfers are made by all peers. The peers share their resources, the system capacity increases and this restricts the scalability.

BitTorrent is in fact the best example of cooperative system in sharing content. Despite its success, BiTorrent suffers from problems due to large and heterogeneous populations during critical periods of the network. Therefore, the network coding is proposed to remedy such deficiencies.

The network coding scheme is proposed to improve the bandwidth usage in a given network. It allows intermediate nodes to encode the pieces sent. Each time a client needs to send a packet to another client, the source generates and sends a linear combination of all information. A remote client can reconstruct the original information if it has received enough independent linear combinations of packets. The difficulty is to find a pattern of propagation package that minimizes the download time for a given client. This is what the scheduler in some systems, so that the proposed network coding avoids having this type of mechanism.

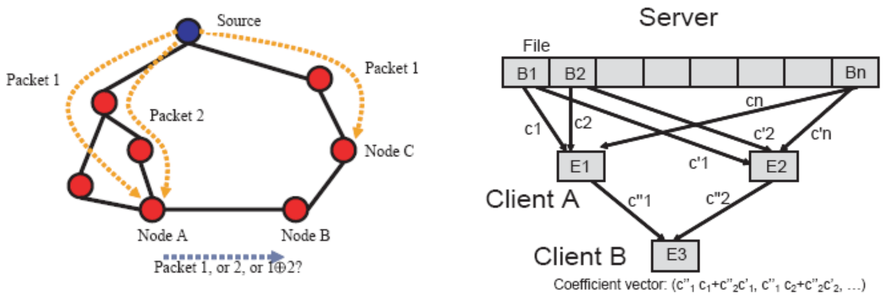


Figure 4.3: Network Coding (Figure from [GKA&al05])

Let's consider the example in the previous scheme where the node A receives the packets 1 and 2 of the source. If network coding is not used the Node B will download the package 1 or 2 of A with an equal probability. While the Node B downloads a package of Node A, Node C downloads the package 1 independently. If Node B decides to receive a package of A, nodes B and C have the same package and a link between them can be used. In cases where network coding is used in Node B downloads a linear combination of packages 1 and 2 and one that can be used by C. Node B could download the package 2 of A and use effectively the link with C. However, without a good knowledge transfer to the rest of the network (difficult in a large and complex) Node B can determine the correct package to download. However, network coding helps a lot to accomplish this task. It is important to note that the decision packages to generate and send does not require any additional information on downloads in the rest of the network.

4.1.3 Digital Fountain: FEC applied on Blocks

Until now we presented works that applied FEC on pieces. In the source coding, a proposal called Digital Fountain [BYE&al98] has been made in where error coding is used to allow peers to reconstruct the original content of size n from any n symbols of a large number of coded symbols. In this case the coding is applied to the blocks and the coding source is considered as unlimited.

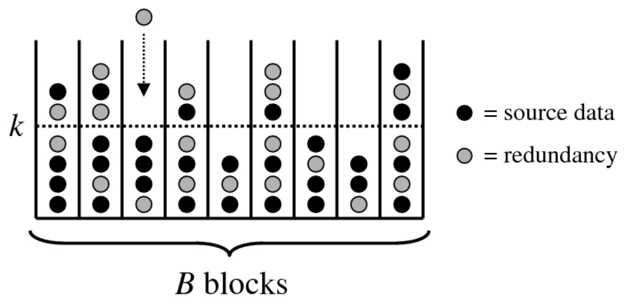


Figure 4.4: Digital Fountain mechanism (Figure from [BYE&al98])

In the previous diagram, we present the structure of a file F divided into B blocks. We have a number of pieces for file *K*. And $B = K / k$ with k the length of the blocks. We move from some blocks of k pieces to blocks of $k + l$ pieces. The choice of k is crucial here. The smaller this value is the faster as the decoding is but the number of elements to decode becomes more important and this can be a problem in terms of processing.

4.2 Speed up data access in Peer-to-Peer content distribution networks

In the previous section, we presented a work that shows the performance gain of network coding compared to source coding and the BitTorrent without any coding. The loss of coding efficiency at the source is mainly due to the fact that the spread of the pieces within the network by duplication (without the interior peers treat them as in the case of network coding) introduces some loss bandwidth, due to overhead in the network of redundant encoded packets that may replace packets that data peers seek in the network without results.

There are two important points to think about when it is to apply an Error Coding mechanism, especially in our case.

The first point is the level of the coding in the data. In BitTorrent, we can apply coding at the block level or the piece level. The advantage of coding at the smaller level (block) is that for replacing some blocks of a piece, it is not an obligation to get the entire piece. It is possible that only some blocks of a piece need to be replaced. The disadvantage is that the treatment is more important and complex in the block case than for the piece case considering the same file size.

The second point is the comparison between Network Coding and FEC coding. First of all, when it is to compare *Rarest First* implementation in BitTorrent without any coding to a version where coding is implemented to increase the piece entropy, some treatment is added to the peers that perform the coding. It is proved in [LEG&al06] that when the BitTorrent network and the Peers Set is important enough, *Rarest First* entropy is close to 1 and no coding is really necessary. The choice depends on the application need and the level of costs acceptable by the developers that chose to implement it. Some applications like real time applications or some ISPs that want to ensure more guarantees for their customers may choose to implement coding mechanism, even if the cost and the treatment is higher or more complex. The choice of Network Coding is also more compromising because even if the performances are better than source coding, the treatment is necessary in all peers of the network since the principle is to make all peers participate to the coding mechanism. The source-based approaches consider the networks as in effect channels with ergodic erasures or errors, and code over them, attempting to reduce excessive redundancy. The data is expanded, not combined to adapt to topology and capacity. Network coding principle is to fuse data exchange information. Another drawback of choosing Network Coding is that it is not interoperable with generic BitTorrent client. Avalanche [GKA'&al06] that would implement Network Coding or BitCod that is the only real network coding client, are different from BitTorrent and propose their own algorithms and mechanisms.

We will choose to apply FEC mechanism that is simpler and evaluate in which cases it presents some real advantages and how it is interesting to integrate, if without degrading the network performance and the adding too much complexity and treatment to the protocol and the network. Interoperability with previous version is also our principal objective.

4.3 The simulation framework: *Seeds* providing FEC

Let's consider a given file, it is fractioned into pieces and these pieces are also divided into blocks in the original specification of BitTorrent. We only model the pieces exchange between peers. Let *k* be the number of pieces that form the original file, a number of *n-k* redundant pieces can be injected by the source with *n* the total number of pieces removed from the FEC encoder. In the case of our implementation, it is therefore the *Seed* of origin that provides these pieces *nk* redundancy. Once the simulator is running, instead of injecting *k* pieces, *n* pieces are available knowing that any of the N-K pieces can replace any pieces of the *k* pieces. In this case *n*/*k* or coding rate gives the percentage of redundancy. For example, the FEC ratio = *n*/*k* = 150/100 = 1.5 = 50 pieces and *n* redundant ones. Note that none of the algorithms change in the simulator. The algorithm is applied with the *Rarest First* and in this case, it applies to the *Seed* that injects *n* pieces and not to *k* pieces anymore. However, we can turn this algorithm off. The End Game Mode has been neglected as we mentioned earlier since the problem of the Last segment can be adjusted by the presence of FEC coded pieces.

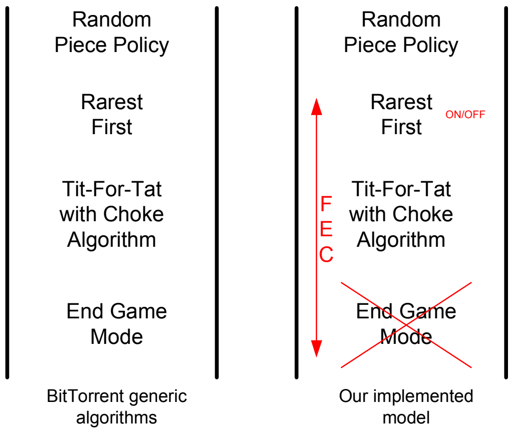


Figure 4.5: Model implemented in the simulator for FEC mechanism

Performance measurements in BitTorrent researches are various and the different frameworks proposed are generally based on real world experiments [IZA&al04] or simulations at the flow level. We decide, like in [EGR&al07] where the authors developed a BitTorrent simulator over NS2, to focus on the piece level.

The discrete-event simulator we presented in chapter 3 is the same used in this chapter. The implementation of FEC mechanism in our BitTorrent simulator was realized by the generation of special pieces based on Reed Solomon codes that are generated by the *Seeds*. It is possible to manage the proliferation of encoded pieces and to vary the FEC pieces ratio. If the ratio value equals *1*, that means no FEC pieces are injected in the network. For example, a ratio value of *1.2* means that a portion of 20% of the original pieces number is added to the pieces. For a 100 pieces resource, a *Seed* generates 20 additional pieces (redundant pieces) and finally provides a total of 120 pieces. A peer can retrieve 100 pieces from the 120 provided to complete the download. Every piece downloaded from the 20 encoded pieces can compensate any of the 100 original pieces. In the next sections, the FEC is provided during the entire duration of the simulations. This is to conclude on how the whole system reacts with the FEC pieces in the different periods of the download time duration.

For some recalls and for a better understanding of this paper, the following presents some basic aspects of BitTorrent. The *Leech* and the *Seed* have different roles in the BitTorrent network behavior. The *Choke Algorithm* is a real motivation to the *Leeches* participation because of tit-for-tat reciprocity characteristic. We call this motivation the incentive [10]. Furthermore, the *Seeds* have the entire resource and are only uploading pieces. The more they upload, the less *Leeches* will be able to participate, and so to receive resources from other peers. However, the compromise is that *Seeds* are essential in the BitTorrent activity and permit to increase consistently the protocol performance. We can see in Figure 4.6 that the gain in the completion time is significant. In Figure 4.7, we notice that in increasing the *Seeds* number, we see the decrease of the *Leeches* sent segments average. When a unique *Seed* is present, this average is close to the resource pieces number (400 in the example). For this simulation, we took 100 heterogeneous *Leeches*. The resource size is a 100 MB size file. In the 1 initial *Seed* simulation 76 *Leeches* completed the download and in the 50 *Seeds* simulation 82 *Leeches* completed the download.

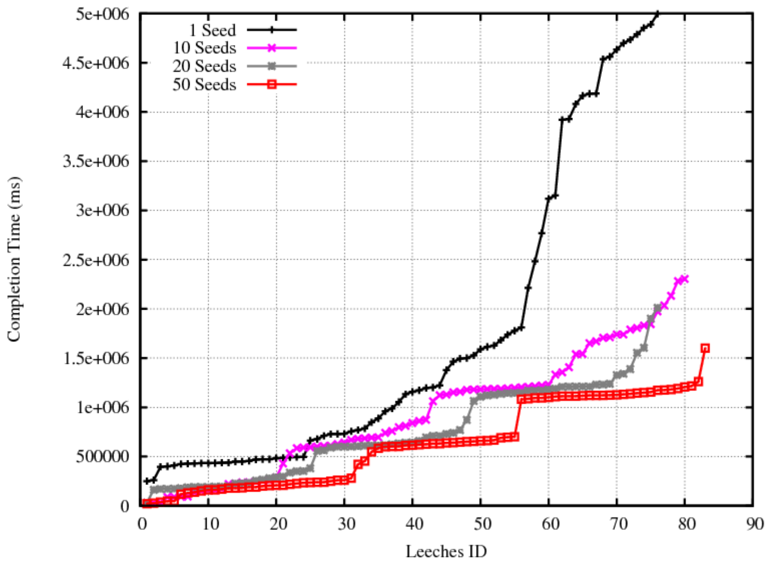


Figure 4.6: Completion time depending on the *Seeds* number

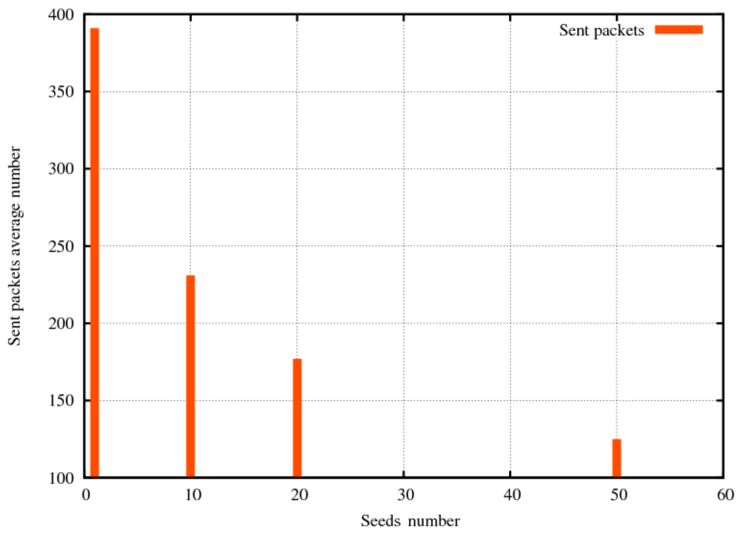


Figure 4.7: Sent pieces depending on the *Seeds* number

We showed that the *Seeds* important activity decreases the traffic between *Leeches*, even if the BitTorrent behavior tends to motivate exchange between *Leeches*.

The next two sections present the simulations that we launched to evaluate FEC mechanism in BitTorrent protocol. We choose to simulate two different environments: a homogeneous network (peers have the same upload and download capacities) and a heterogeneous network (with 5 different peer types). In fact, the implementation of FEC can vary from a network nature to another. We will study the influence of the file size, the *Leeches/Seeds* number and capacities and the FEC ratio. Based on the results we will discuss the impact of these parameters in code correction within BitTorrent-like protocols.

***Rarest First* Policy and FEC mechanism**

Rarest First policy ensures the best entropy for pieces distribution in BitTorrent protocol. When random policy is applied we saw that performances are degraded. When FEC is integrates with the Random policy we see that performances are similar with *Rarest First* policy. When *Rarest First* is complemented by FEC mechanism we see a notable performance gain. However it is important to appreciate the integration of FEC mechanism that can be an overhead in the network depending on the FEC ratio.

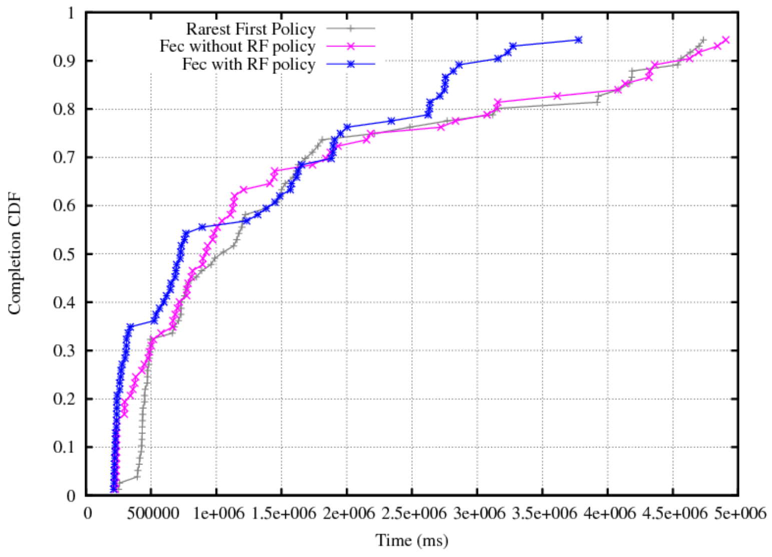


Figure 4.8: *Rarest First* Policy and FEC mechanism

4.3.1 Homogeneous Networks

We consider in the first part of our simulations that the network is a homogeneous network constitute of *Leeches* with exactly the same capacities. We fix some parameters in TABLE I and decide to vary the *Leeches/Seeds* number, and the file size. In every scenario we simulated every case with 4 different FEC ratios and compared them to a *No FEC* simulation and to each others.

In Figure 4.9 we simulate a small network with 50 *Leeches*. In subfigures (a) and (b) we fixed a 100 MB file and we vary the initial *Seeds* number. In (a) we have only one initial *Seed* as the unique media source while in (b) the *Seeds* number is equal to 5. For each graph we represent 5 different simulations: the first one corresponds to the original

BitTorrent transport protocol with no additional FEC. For the four other simulations, we took some different ratios: *1.1*, *1.2*, *1.3* and *1.5*.

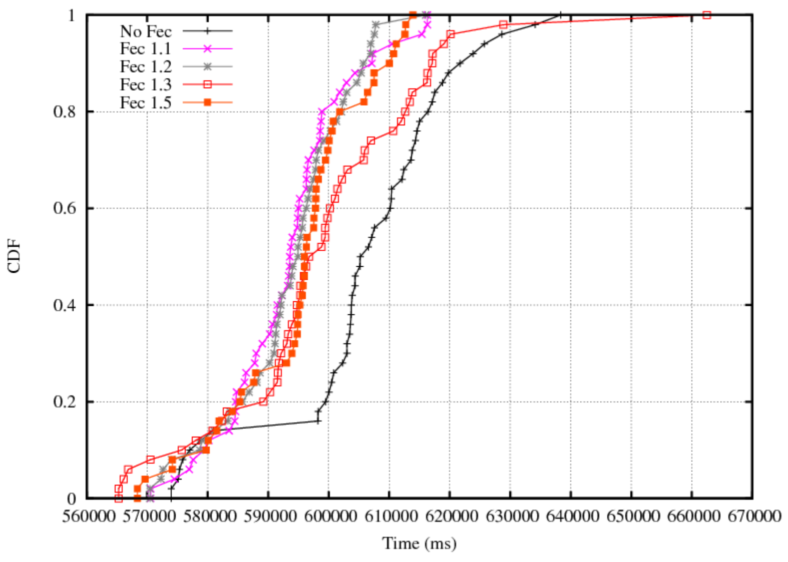
Table 4.1: Parameters for homogeneous networks simulation with FEC mechanism

Parameter	Value
Number of *Leeches*	1000
Leeches Capacities (Down/Up)	800 kbps/400kbps
Number of initial *Seeds*[1]	1 or 20
Seeds Upload capacity	1500 kbps
File Size	100 MBytes (819200 kbits)
Seed leaving probability	1
Leech abort probability	0
Peers Set size	50
Unchoked Connections per peer	5 (4 regulars and 1 optimistic)

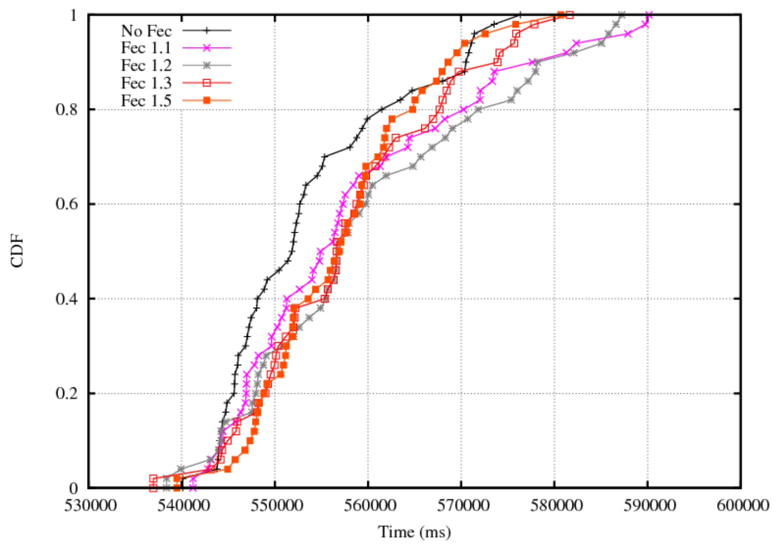
We evaluate the different performance results on the completion portion of peers during all the simulation duration (until all the 50 peers have finished their download). The CDF varies from 0 to 1 (1 representing 100% of the peers that completed the download).

We can note that FEC has a positive impact on subfigure (a) case and that it varies from a ratio to another. The low ratios (like *1.1* and *1.3*) show the best performance in the sense that for the same instant time, we have more *Leeches* that became *Seeds* than for the higher ratios.

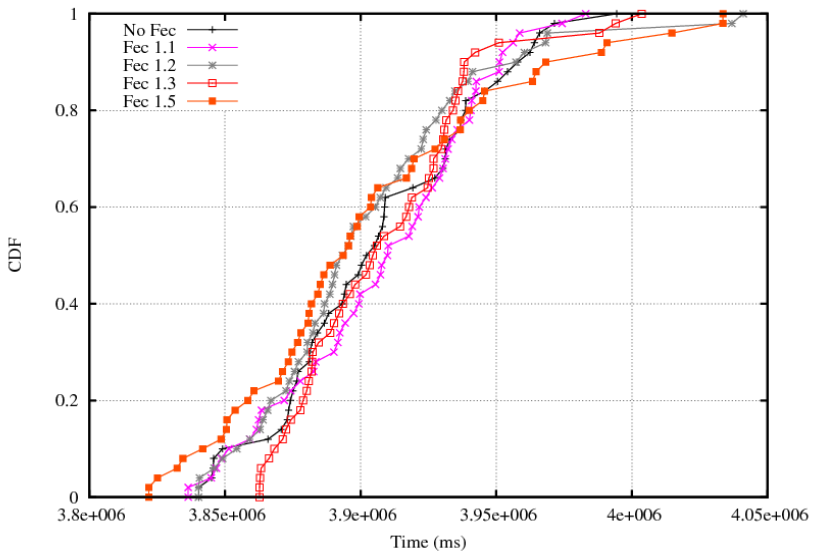
In (b) we increase the *Seeds* number for the same parameters and we show that FEC degrades the performance of the system. The *Seeds* number is enough for the 50 *Leeches* to correctly download the resource. The FEC is introducing here an overhead. The subfigure (c) presents the same configuration but with a 700 MB file. We note that the results are not deterministic and that during the three first quarters of the simulation, the *1.5* ratio applied has a positive benefit. Compared to (a), where lower ratios have some positive impact on the performance, when the file size is increased, the FEC mechanism is not necessary.



(a) *1 Seed* and 100 MB File



(b) 5 *Seeds* and 100 MB File



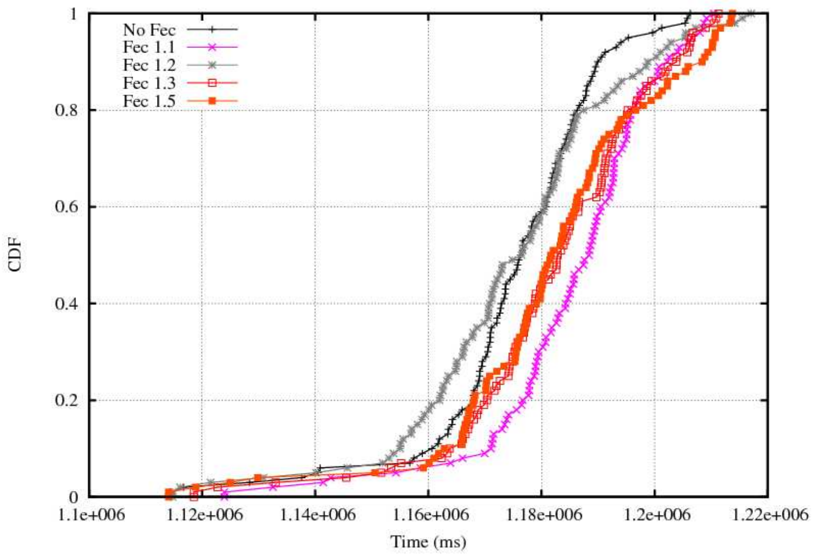
(c) 1 *Seed* and 700 MB File

Figure 4.9: *Leeches* that become *Seeds* in a 50 normal *Leeches* swarm (homogeneous network)

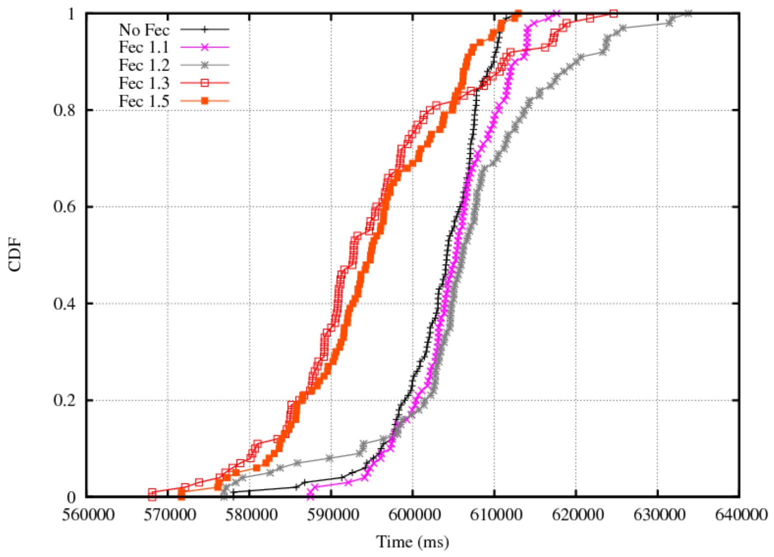
In Figure 4.10, the swarm configured is constituted by 100 *Leeches*. In this battery of simulations, we decided to vary the speed capacities. In (a) the *Leeches* have weaker capacities than in Figure 4.9 simulations. We note that in this case the FEC tends to add an overhead to the system. On the contrary in subfigure (b), for *1.3* and *1.5* ratios, FEC increases the general performance of the system. We can observe that in subfigure (b) and subfigure (a), the *Leeches* number is the unique parameter that varies and that in both cases FEC has a good impact on *Leeches* completion.

The case observed in subfigure (c) where the initial *Seed* speed is increased does not present major performance amelioration. However, for the *1.5* ratio, until that 50% of the *Leeches* have finished their download, the completion is better but for the principal other ratios the results are similar. We can note that in this case FEC acts like an accelerator process when the ratio is large enough. This is due to the increase of the *Seed* speed.

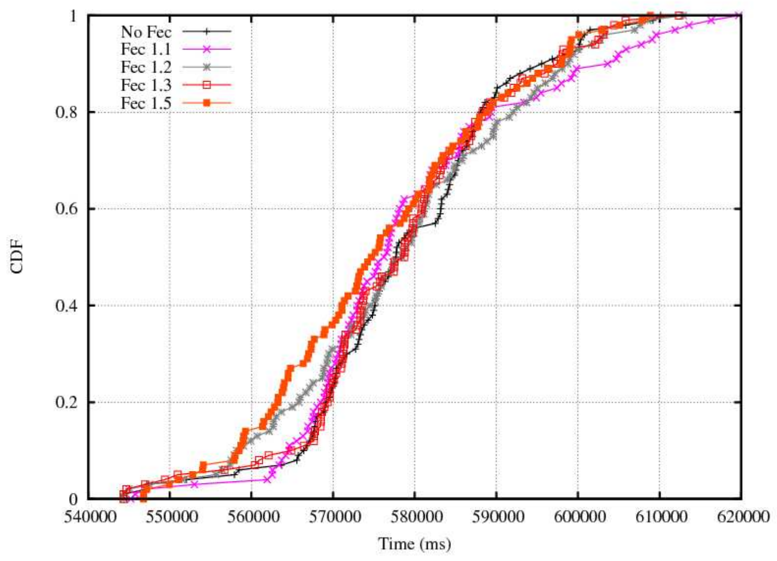
The Figure 4.11 represents a swarm with 1000 *Leeches* with only one initial *Seed*. In this case, we can see that, except for the lower ratio *1.1*, we gain in performance. For the *2.0* ratio, the result degraded comparing to *1.3* or *1.5*. In fact, *2.0* means that there is the same number of coded pieces as origin pieces, which is an important overhead (the number of pieces is doubled). However, there is only one initial *Seed* in this swarm for a very large number of *Leeches*. This shows that when there is penury of resource, the FEC can be a good backup solution.



(a) 1 normal *Seed*, 100 MB file, and weak *Leeches* (*cf* TABLE I)



(b) 1 normal *Seed*, 100 MB file, and normal *Leeches*



(c) 1 high speed *Seed*, 100 MB file, and normal *Leeches*

Figure 4.10: *Leeches* that become *Seeds* in a 100 *Leeches* swarm

From the different configurations presented previously we can conclude as following for a homogeneous network:

- For a swarm configuration with only one initial *Seed* that is the first to provide the resource and a 100 MB file size, FEC increases the system performance, mostly when a large number of *Leeches* are downloading the resource. When the *Leeches* number is small, providing some weak ratios (*1.1* or *1.2*) has a positive impact on the download. Increasing the *Leeches* number made FEC interesting only if the redundant pieces are large enough (ratios *equal or* greater than *1.3*).
- We noted that increasing the *Seeds* number while implementing FEC has a negative impact on the system. This result shows that FEC is more interesting as a backup solution when a lack of resource is found in the swarm. In the same manner, increasing a unique *Seed* upload capacity presents the same impact as increasing their number at the beginning of the downloading. In fact in these cases, FEC is not affecting positively the system performances.
- In the same way, we found that when the *Leeches* capacities are weaker, an overhead is also added and that degrades the system. FEC provides a performance gain if *Leeches* have good download and upload capacities. FEC is acting like a *booster* for peers that have already good enough capacities.
- The file size parameter is also affecting the download. For a higher file size the FEC increases the system performance only when the ratio is high enough.

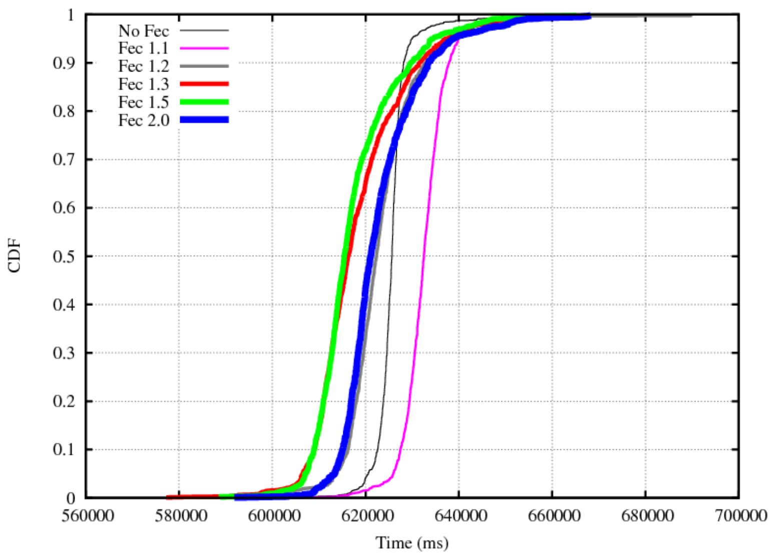


Figure 4.11: *Leeches* that become *Seeds* in a 1000 normal *Leeches* swarm (homogeneous network)

In Figure 4.12 we simulate a very high speed network configuration to show how FEC mechanism can be interesting to speed up data transfer also when *Leeches* in the swarm are already presenting high capacities.

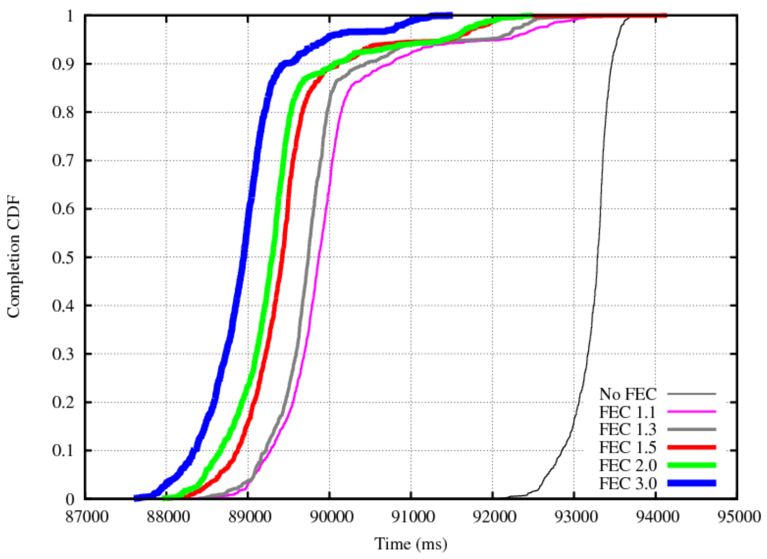


Figure 4.12: Very high speed scenario with FEC mechanism

4.3.2 Heterogeneous Networks

The second part of our simulation work consists in creating a heterogeneous configuration to evaluate the FEC performance in a system where *Leeches* have different capacities. This scenario is closer to real networks where clients can have different connections and so on for download and upload capacities.

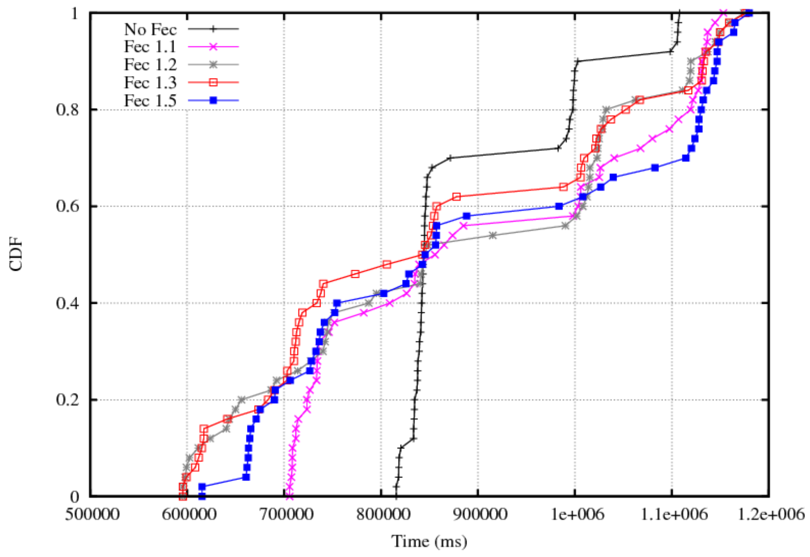
The parameters in which next simulations are based Figured in TABLE II. The principal objective in this configuration is to show how FEC mechanism reacts with the different *Leeches*.

The Figure 4.13 represents three graphs with different *Leeches* numbers that are the same chosen previously. The file size is 100 MB and only one initial *Seed* is providing the resource.

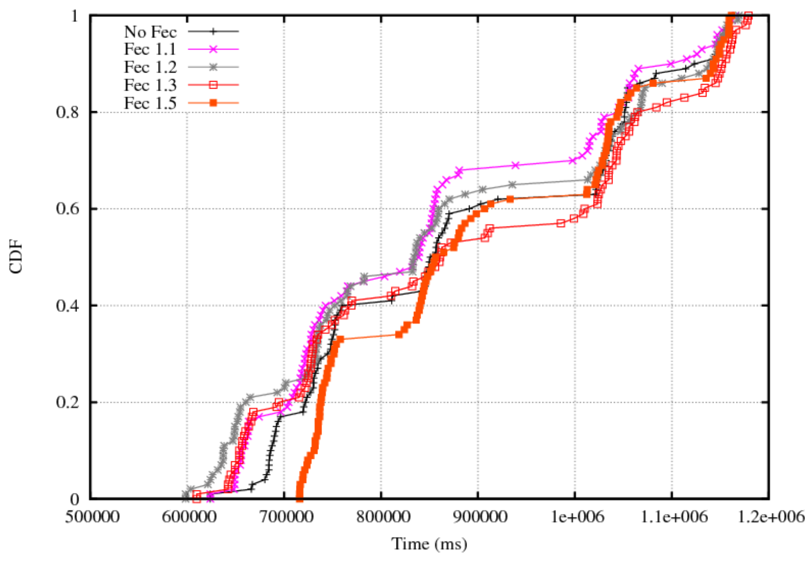
Table 4.2: Parameters for heterogeneous networks simulation with FEC mechanism

Parameter	Value
Number of *Leeches*	1000
Leeches Capacities (Down/Up)	25% of 200 kbps/100kbps 25% of 500 kbps/200kbps 25% of 800 kbps/400kbps 25% of 1000 kbps/500kbps
Number of initial *Seeds*[1]	1 or 20
Seeds Upload capacity	1500 kbps
File Size	100 MBytes (819200 kbits)
Seed leaving probability	1
Leech abort probability	0
Peers Set size	50
Unchoked Connections per peer	5 (4 regulars and 1 optimistic)

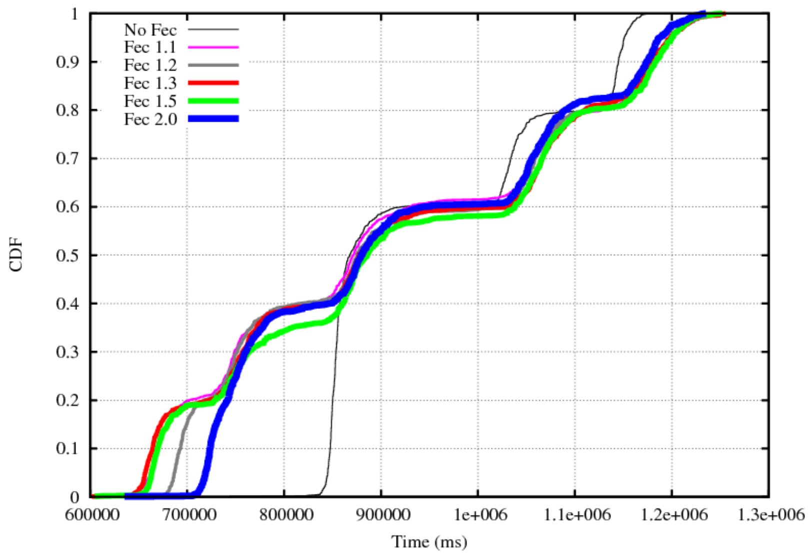
In subfigure (a) the results clearly show that FEC is interesting for high speed *Leeches*. We can easily distinguish by the steps observed in each simulation that represent the *Leeches* types differentiated by their capacities. After 850 000 ms the FEC is degrading the system and for type 1 to 3 (*cf* TABLE II), the *No FEC* simulation presents some better performance. This confirms what we conclude concerning the *Leeches* speed and the positive effect on fast peers.



(a) 50 *Leeches* swarm



(b) 100 *Leeches* swarm



(c) 1000 *Leeches* swarm

Figure 4.13: *Leeches* that become *Seeds*: 1 initial *Seed* and 100 MB file (heterogeneous network)

In subfigure (b) where 100 *Leeches* are running, we can note that for ratios *1.1* and *1.2,* the FEC is interesting even if the difference is not considerable. On the contrary, for higher ratios, the system is disrupts. Increasing the *Leeches* number degrades the performance of higher capacity *Leeches* but for the low capacity *Leeches* the FEC pieces have increased the completion portion compared to the No FEC simulation. This result is relative in the sense that adding redundant pieces is interesting if the ratio is important (till *1.2*) for fear of introducing an overhead to the network.

In subfigure (c) the swarm is large and is constitute by 1000 *Leeches*. In this case we saw the particularity of FEC mechanism that is observed in the major simulation scenarios which is the accelerator function of this technique. When *Leeches* have high enough capacities, FEC provides a better startup in all the download process. This gain is visible when the *Seeds* number is weak and especially equal one or when these provider peers capacity is weak.

We perform deeper studies on FEC mechanism choosing a FEC ratio equals to 1.3 applied in a swarm of 100 *Leeches* then 2000 *Leeches* for a 100 MB resource size. In Table 4.3 we detailed the information showing the impact of *Seeds* number increase on the *Leeches* participation and the download completion time for the 100 *Leeches* example. In Table 4.4 we note that for 2000 *Leeches* the impact is also clear even if in this case the availability is much more necessary while the *Leeches* number is higher.

Table 4.3: Comparing results between generic BitTorrent and BitTorrent with FEC mechanism (100 *Leeches*)

NO FEC	1 Initial *Seed*	10 initial *Seeds*	20 initial *Seeds*	50 initial *Seeds*
Leeches number that completed the download	76	79	75	82
Download average time	1 695 711 ms	933 761 ms	743 176 ms	633 827 ms

Sent pieces average number per *Leech*	391 (100 096 KB)
	MAX= 2065
	MIN= 97
		231 (59 136 KB)
		MAX=1353
		MIN= 36
			175 (44 800 KB)
			MAX= 775
			MIN= 11
				125 (32 000 KB)
				MAX= 497
				MIN= 25

Sent pieces average number per initial *Seed*	732 (187 392 KB)	735 (188 160 KB)	717 (183 552 KB)	740 (189 440 KB)
FEC 1.3	1 initial *Seed*	10 initial *Seeds*	20 initial *Seeds*	50 initial *Seeds*
Leeches number that completed the download	76	87	83	86
Download average time	1 432 473 ms	782 342 ms	623 941 ms	551 635 ms

Sent pieces average number per *Leech*	399 (102 144 KB)
	MAX= 2201
	MIN= 124
		217 (55 552 KB)
		MAX= 1353
		MIN= 67
			197 (50 432 KB)
			MAX= 683
			MIN= 53
				134 (34 304 KB)
				MAX= 675
				MIN= 1

Sent pieces average number per initial *Seed*	735 (188 160 KB)	752 (192 512 KB)	765 (195 840 KB)	675 (172 800 KB)

Table 4.4: Comparing results between generic BitTorrent and BitTorrent with FEC mechanism (2000 *Leeches*)

NO FEC	1 *Seed* Initial	10 *Seeds* Initiaux	20 *Seeds* Initiaux	50 *Seeds* Initiaux
Leeches number that completed the download	1530	1502	1531	1572
Download average time	1 530 858 ms	1 432 288 ms	1 444 976 ms	1 228 195 ms

Sent pieces average number per *Leech*	399 (102 144 KB)
	MAX= 2402
	MIN= 73
		382 (87 792 KB)
		MAX= 2591
		MIN= 67
			363 (92 928 KB)
			MAX= 2341
			MIN= 68
				331 (84 736 KB)
				MAX= 2043
				MIN= 64

Sent pieces average number per initial *Seed*	692 (177 152 KB)	672 (172 032 KB)	663 (169 728 KB)	704 (180 224 KB)
FEC 1.3	1 *Seed* Initial	10 *Seeds* Initiaux	20 *Seeds* Initiaux	50 *Seeds* Initiaux
Leeches number that completed the download	1536	1512	1559	1599
Download average time	1 499 011 ms	1 416 139 ms	1 307 567 ms	1 170 657 ms

Sent pieces average number per *Leech*	400 (102 400 KB)
	MAX= 2935
	MIN= 74
		386 (98 816 KB)
		MAX= 2660
		MIN= 67
			369 (94 464 KB)
			MAX= 2433
			MIN= 49
				334 (85 504 KB)
				MAX= 2643
				MIN= 42

Sent pieces average number per initial *Seed*	684 (175 104 KB)	678 (173 568 KB)	692 (177 152 KB)	666 (170 496 KB)

4.4 Statistical test model validating the results

We have two populations: P_1 that is characterizing the peers in a basic BitTorrent system without FEC and the second population P_2 where a FEC mechanism is integrated. We choose in this section to apply the test to Figure 4.10 simulations as an example. In this case P_1 represents the No FEC simulation and P_2 the simulation with ratio *1.5*.

In these two populations we studied the completion time noted T. We take a sample in P_1 and a sample in the population P_2. The samples sizes are respectively n_1 and n_2.

The issue is to know if the difference observed on both samples comes from a real difference between the two populations or if it is due to random phenomena.

Let's consider m_1 (respectively m_2) the average value of T in the population P_1 (respectively P_2). We apply a Bilateral Test [RIO&al98] as the following:

The objective is to test $H_0: m_1 = m_2$ against $H_1: m_1 \neq m_2$.

We have two samples: the first issue from the first population with the size n_1 and the second issue from the second population with the size n_2. For our example $n_1 = n_2 = 1000$.

We take the precaution to fix the risk of the first kind which is the probability to reject wrongly H_0.

We note $\overline{T}_{n_1}, \overline{T}_{n_2}$ the empirical average of the completion time in the two populations and the level of significance $d = \overline{T}_{n_1}, -\overline{T}_{n_2}$, the problem is to know if d is significatively non equal to 0 or not.

In the case of big samples like here, the central limit theorem (CTL) [28] allows us to say that for any law of T, the laws of $\overline{T}_{n_1}$, and $\overline{T}_{n_2}$, are respectively normal laws with parameters $(m_1, \frac{\sigma_1}{\sqrt{n_1}})$ and $(m_2, \frac{\sigma_2}{\sqrt{n_2}})$.

We note that: $\overline{T}_{n_1} \rightsquigarrow \mathcal{N}(m_1, \frac{\sigma_1}{\sqrt{n_1}})$ and $\overline{T}_{n_2} \rightsquigarrow \mathcal{N}(m_2, \frac{\sigma_2}{\sqrt{n_2}})$

In the other case,

$$E(d) = E(\overline{T}_{n_1} - \overline{T}_{n_2}) = E(\overline{T}_{n_1}) - E(\overline{T}_{n_2}) \qquad (1)$$

$$Var(d) = Var(\overline{T}_{n_1} - \overline{T}_{n_2}) = Var(\overline{T}_{n_1}) + Var(\overline{T}_{n_2}) \qquad (2)$$

Using (1) and (2), we have:

$E(d) = m_1 - m_2$ and $Var(d) = \frac{\sigma_1^2}{n_1} + \frac{\sigma_2^2}{n_2}$

Since the sum and the difference of two random variables following normal laws also follow normal laws, the law of *d* is a normal law with parameters $m_1 - m_2$ and $\sqrt{\frac{\sigma_1^2}{n_1} + \frac{\sigma_2^2}{n_2}}$. $d \rightsquigarrow \mathcal{N}(m_1 - m_2, \sqrt{\frac{\sigma_1^2}{n_1} + \frac{\sigma_2^2}{n_2}})$

Under the hypothesis H_0, we have $d \rightsquigarrow \mathcal{N}(0, \sqrt{\frac{\sigma_1^2}{n_1} + \frac{\sigma_2^2}{n_2}})$.

We search for the region of rejection of the hypothesis H_0.

It is a bilateral test, this region has the form $\{|d| \geq k\}$ and we have to determine *k*.

The risk of the first kind α is the probability to reject wrongly H_0, i.e the probability to decide D_1 (to accept H_1) knowing that H_0 is true.

$$\alpha = P(D_1/H_0) = P(|d| \geq k/H_0) = P(d \leq -k/H_0) + P(d \geq k/H_0)$$

$$\alpha = P\left(d \le -k \middle/ d \rightsquigarrow \mathcal{N}\left(0, \sqrt{\frac{\sigma_1^2}{n_1} + \frac{\sigma_2^2}{n_2}}\right)\right) + P\left(d \ge k \middle/ d \rightsquigarrow \mathcal{N}\left(0, \sqrt{\frac{\sigma_1^2}{n_1} + \frac{\sigma_2^2}{n_2}}\right)\right)$$

$$\alpha =$$
$$P\left(\frac{d}{\sqrt{\frac{\sigma_1^2}{n_1}+\frac{\sigma_2^2}{n_2}}} \le -\frac{k}{\sqrt{\frac{\sigma_1^2}{n_1}+\frac{\sigma_2^2}{n_2}}} \middle/ d \rightsquigarrow \mathcal{N}\left(0, \sqrt{\frac{\sigma_1^2}{n_1} + \frac{\sigma_2^2}{n_2}}\right)\right) +$$
$$P\left(\frac{d}{\sqrt{\frac{\sigma_1^2}{n_1}+\frac{\sigma_2^2}{n_2}}} \ge \frac{k}{\sqrt{\frac{\sigma_1^2}{n_1}+\frac{\sigma_2^2}{n_2}}} \middle/ d \rightsquigarrow \mathcal{N}\left(0, \sqrt{\frac{\sigma_1^2}{n_1} + \frac{\sigma_2^2}{n_2}}\right)\right)$$

$$\alpha =$$
$$P\left(\frac{d}{\sqrt{\frac{\sigma_1^2}{n_1}+\frac{\sigma_2^2}{n_2}}} \le -\frac{k}{\sqrt{\frac{\sigma_1^2}{n_1}+\frac{\sigma_2^2}{n_2}}} \middle/ \frac{d}{\sqrt{\frac{\sigma_1^2}{n_1}+\frac{\sigma_2^2}{n_2}}} \rightsquigarrow \mathcal{N}(0,1)\right) +$$
$$P\left(\frac{d}{\sqrt{\frac{\sigma_1^2}{n_1}+\frac{\sigma_2^2}{n_2}}} \ge \frac{k}{\sqrt{\frac{\sigma_1^2}{n_1}+\frac{\sigma_2^2}{n_2}}} \middle/ \frac{d}{\sqrt{\frac{\sigma_1^2}{n_1}+\frac{\sigma_2^2}{n_2}}} \rightsquigarrow \mathcal{N}(0,1)\right)$$

Noting ϕ the distribution function of the normal law, we have:

$$\alpha = \phi\left(-\frac{k}{\sqrt{\frac{\sigma_1^2}{n_1}+\frac{\sigma_2^2}{n_2}}}\right) + 1 - \phi\left(\frac{k}{\sqrt{\frac{\sigma_1^2}{n_1}+\frac{\sigma_2^2}{n_2}}}\right)$$

$$\alpha = 1 + \phi\left(-\frac{k}{\sqrt{\frac{\sigma_1^2}{n_1}+\frac{\sigma_2^2}{n_2}}}\right) - \phi\left(\frac{k}{\sqrt{\frac{\sigma_1^2}{n_1}+\frac{\sigma_2^2}{n_2}}}\right)$$

$$\alpha = 1 + 1 - 2\phi\left(\frac{k}{\sqrt{\frac{\sigma_1^2}{n_1}+\frac{\sigma_2^2}{n_2}}}\right)$$

$$\alpha = 2\left(1 - \phi\left(\frac{k}{\sqrt{\frac{\sigma_1^2}{n_1}+\frac{\sigma_2^2}{n_2}}}\right)\right) \text{ and } \phi\left(\frac{k}{\sqrt{\frac{\sigma_1^2}{n_1}+\frac{\sigma_2^2}{n_2}}}\right) = 1 - \frac{\alpha}{2}$$

Setting $K_{\frac{\alpha}{2}} = \frac{k}{\sqrt{\frac{\sigma_1^2}{n_1}+\frac{\sigma_2^2}{n_2}}}$, we have : $k = K_{\frac{\alpha}{2}} \times \sqrt{\frac{\sigma_1^2}{n_1}+\frac{\sigma_2^2}{n_2}}$ and $\phi\left(K_{\frac{\alpha}{2}}\right) = 1 - \frac{\alpha}{2}$.

Then the region of rejection of the hypothesis H_0 is:

$$\left\{|d| \geq K_{\frac{\alpha}{2}} \times \sqrt{\frac{\sigma_1^2}{n_1}+\frac{\sigma_2^2}{n_2}}\right\} = \left\{d \leq -K_{\frac{\alpha}{2}} \times \sqrt{\frac{\sigma_1^2}{n_1}+\frac{\sigma_2^2}{n_2}}\right\} \cup \left\{d \geq K_{\frac{\alpha}{2}} \times \sqrt{\frac{\sigma_1^2}{n_1}+\frac{\sigma_2^2}{n_2}}\right\}$$

In the following, we take $\alpha = 1\%$. So $K_{\frac{\alpha}{2}} = 2{,}575$.

So the region of rejection is:

$$\left\{|d| \geq K_{\frac{\alpha}{2}} \times \sqrt{\frac{\sigma_1^2}{n_1}+\frac{\sigma_2^2}{n_2}}\right\} = \left\{\frac{|d|}{\sqrt{\frac{\sigma_1^2}{n_1}+\frac{\sigma_2^2}{n_2}}} \geq 2{,}575\right\}$$

We demonstrate how to compare two populations with a bilateral statistical test. We can now calculate the region of rejection with the simulation results that we used to obtain Figure 4.10, especially the No FEC and the *1.5* ratio graphs. They represent the two populations that we want to compare.

The numeric application is the following:

$$\frac{|d|}{\sqrt{\frac{\sigma_1^2}{n_1}+\frac{\sigma_2^2}{n_2}}} = \frac{\overline{T}_{n_1'} - \overline{T}_{n_2}}{\sqrt{\frac{\sigma_1^2}{n_1}+\frac{\sigma_2^2}{n_2}}} = \frac{625755.108 - 618376.709}{\sqrt{\frac{17454106}{1000}+\frac{489694730}{1000}}}$$

Then we have, for a risk $\alpha = 1\%$:

$$\frac{|d|}{\sqrt{\frac{\sigma_1^2}{n_1}+\frac{\sigma_2^2}{n_2}}} = 10.360827 > 2.575$$

Finally $H_0: m_1 = m_2$ is rejected against $H_1: m_1 \neq m_2$, which is the hypothesis accepted.

Adding *1.5* ratio FEC pieces in a 1000 *Leeches* swarm with one initial *Seed* providing a 100 MB file presents some difference with a classical BitTorrent network without FEC. This difference in terms of performance was demonstrated by the bilateral statistical test. In the same way it is possible to compare and analyze every simulation with each other. In our example this difference is seen as positive because FEC mechanism completion times are better than the *No FEC* ones.

4.5 Conclusion

The FEC mechanism is based upon erasure codes and permits in many applications to support reliable data transfer or error correction in a network that experienced high level losses. Our paper is a simulation and evaluation study of this mechanism and its performance in BitTorrent-like protocols.

A simulation framework was developed to configure many scenarios with homogeneous and heterogeneous networks. In both configurations the results are similar and proved that FEC increase the performance of *Leeches* when these downloading peers have good enough capacities. This technique cannot replace BitTorrent algorithms as *rarest first* but can be a complementary technique. It boosts the download when the network experience lack of resource if a unique initial *Seed* is present in the swarm or when the *Seeds* capacity are weak.

The results permit us to note that the more the swarm is large and the more the resource size is consequent, the more the redundant pieces ratio must be important to add a positive benefit into the system. At the contrary, FEC mechanism can degrade the general performance.

We previously work on the BitTorrent performance and contribute by providing a new selection policy inside the same autonomous system and a formal specification to map peers with their domain membership. In the future we will be interested on providing a unique architecture based on a BitTorrent-like protocol. We propose to integrate in the solution the localization component and a FEC mechanism at the transport level when the network experience some download difficulties.

Chapter 5

SPOP: a Service Provider Oriented P2P architecture

This chapter presents a Service Provider Oriented P2P (SPOP) architecture that takes into account the contributions presented in the two previous chapters concerning two of the three P2P main components: the Transport and the Service components. For the first component we propose to integrate FEC mechanism as a parallel service that can be provided by an ISP and for the second one we added a Control/Management level where the hTracker entity could vary some protocol parameters (Peers Ser size, Piece size, Traffic partitioning) to adapt the network environment depending on applications constraints. In addition we integrate a technique that optimizes the third component which is the routing/lookup component. This latter consists in implementing a DHT that indexes the resources. The particularity is that this DHT is Context-Aware and generalizes the hTracker contribution. Peers are regrouped depending on criteria (AS membership for instance in hTracker) and traffic partitioning is completed by resource lookup optimization. We validated by simulation the Context-Aware DHT technique that was only at the specification stage. The architecture specification is available in [PBL06].

5.1 Introduction and Objectives

Distributed applications like file sharing, Grid computing or even real time applications like VoD or IPTV use an important volume of the network capacity, memory or CPU. In the case of P2P networks, unlike Client-Server model, resources are distributed and the communication is directly done between the customers. The data volume is sometimes very high and needs best performances to be delivered as soon as possible.

In parallel to this, the lack of traffic control is both what made the strength of the Internet today and what is problematic for some applications. It is important to understand how applications use ISPs resources. The application is only interested in indicating the destination of traffic and the ISP is responsible for conducting the Traffic Engineering to determine the most optimal way and to satisfy some economic goals.

An advantage with P2P is that the resources are often duplicated and available at several point of the network. However, the main difference is that in classical P2P networks, the application does not have reliable information about the underlay network. Generally, the peers' selection for downloading the resource is done randomly or depending on basic information. Before entering the real download process, the requesting peer or any other entity in the network cannot evaluate the download rate since the transfer did not begin. This calculation can be a major parameter for some real time applications that needs to choose some peers depending on their capacities and performances in providing quickly the resource.

Choosing peers without any information on the underlay network has two major drawbacks: first, performances are not optimal since the peers are not chosen depending on any quality criteria and second, the network can experience traffic problems that are visible as link overload spikes due to congestion and this generates over costs for ISPs in addition to the bad service provided.

We previously studied two major components of P2P networks and contributions associated to these components. In chapter 3, we proposed a new scheme for peer management with a semantic for peer AS membership and in Chapter 4, we evaluated the efficiency of FEC mechanism on pieces transport for BitTorrent. In this chapter, we propose a global architecture that groups our different contributions and evaluation studies. We will also see how some parameters can be adjusted to propose some performance optimization depending on the application requirements: Peers Set size, Piece size, *Seeds* number, etc.

The principal objective is to propose a solution that fits with ISPs current requirements and guidelines for P2P activity. Each application has some specific constraints that the networks and so on the ISPs must take into consideration to provide the best service or at least the service promised in the customer agreement. We have two cases: in the first case, the P2P application is totally independent from the ISP and in the second case, the P2P application is provided by the ISP. In the first case, it is more difficult to control the peers' activity while when the ISP is providing the service, it can manage and control more easily the traffic, the transport and the parameters that can have an impact on the QoS offered to the service customers.

The principal issues that motivate our work are directly related to each P2P component. The global SPOP architecture results from our different contributions and evaluations in addition to some algorithms and incremental work that we will develop in this chapter. Taking into consideration the levels architecture of a P2P network, we worked on improving the Transport level integrating FEC as a support service provided in complement to the original data transfer, applying a DHT in the routing level to optimize data lookup and finally managing the traffic and some protocol parameters to follow applications requirements.

At the transport level, the objective is to improve the completion time and solve both First Block and Last Block problems. The FEC mechanism accelerates the download when a peer just joins the network and this allows it to have more quickly some pieces to exchange at the beginning. When a *Leech* is experiencing a lack of pieces, in this case FEC can also provide it a rescue service. However with this solution, the FEC ratio depends on many parameters and must be adjusted to avoid overheard that could disrupt the system and make it loose in performance. In our architecture, FEC will be provided by a specific network with homogeneous and fast *Seeds* available to everyone. This contribution will be detailed in this chapter.

At the routing level, we propose to use a structured algorithm and especially the DHT Chord which is simple and robust. Chord will provide optimization of the resource lookup. The idea is to structure the data lookup in parallel with the localization assured by the *Tracker*. We will see in this level that we can propose an enhanced version of Chord that is Context-Aware and that takes into account some criteria to perform the lookup locally in groups before trying the global lookup.

At the Control and Management level, we introduce the *hTracker* entity that replaces the BitTorrent generic *Tracker*. In this contribution, as we saw in chapter 3, some minor modifications have to be applied in the *Tracker* and the Client. The objective here is to optimize the download performance and the peering traffic by changing the peer selection policy mainly. We can also adapt some parameters like the piece size of the resource, the peers set size or the peers number to provide some additional performance improvement when the completion time has to be decreased if the application is time sensitive.

These propositions are doubtless interesting and require no major implementation difficulty. However, all above contributions and propositions must respect the interoperability with previous architecture. We saw in chapter 3 that for instance the *peerId* specification change is transparent with generic *peerdId* specifications. That is the case of the FEC mechanism that provides pieces that can totally replace any of the resource pieces without generating any conflict.

To summarize the following architecture, we mainly care on transport and completion performance, lookup optimization, security, QoS requirements, costs saving and interoperability. The P2P model, on which this architecture is based, ensures scalability, robustness, load balancing and resources aggregation for any application on top of this system.

This chapter is divided as follows. The next section gives the related work and existing architectures that propose some solutions to P2P collaboration with ISPs. Then we will detail the FEC pieces network provider; and introduce the routing optimization with the Context-Aware DHT algorithm. Finally, we define the Control and Management plan with its objectives and results.

5.2 Related Work and existing architectures

The particularity of P2P networks is that they are not associated to any formal standard that describes the real problems of P2P components with some solutions based on some architectures. Some propositions have been done and every one defines two major components that are:

- A discovery mechanism for resource discovery before the data transfer.
- A protocol used by the P2P applications to send some requests that allow them to retrieve information on the underlay level and that will make them apply the peers selection algorithm.

In the following we will present the two main architectures: P4P architecture derived from ALTO project and SmoothIT.

5.2.1 The ALTO Project: P4P architecture

The ALTO (Application Layer Traffic Optimization) project goal is to provide information that a P2P application can use to make a better decision when it is question of peers' selection. This work can also be used by non-P2P applications. The principal objectives are to reduce resource consumption in the underlay network and improve performances with the information. In fact, it is usually difficult for application entities to retrieve reliable information on the underlay network for two principal reasons: first the mechanisms of measurements calculation directly in the network are complex and second the ISPs could help applications by providing these information but this is not their do not especially want to give many details in their infrastructure.

Definitions and functioning

Resource: Content (file or file piece), a process server (for execution of a video stream for example), in which an application can have access. This resource must always be available and should be replicated. Several peers can manage these resources simultaneously.
ALTO Service: Service providers may be required to provide the same resource. The *ALTO* service guide serves user with information on service providers to know how to choose the resource and to optimize client performance. This can also improve resource consumption at the network level underlay.
ALTO Customer: A logical entity that can launch ALTO queries. Depending on the architecture the application can be integrated into a client consumer of the resource and/or the resource directory.
ALTO client protocol: allows sending ALTO queries and responses between the client and the server.
Supplying protocol: It is used to supply the ALTO server with information. Initialization

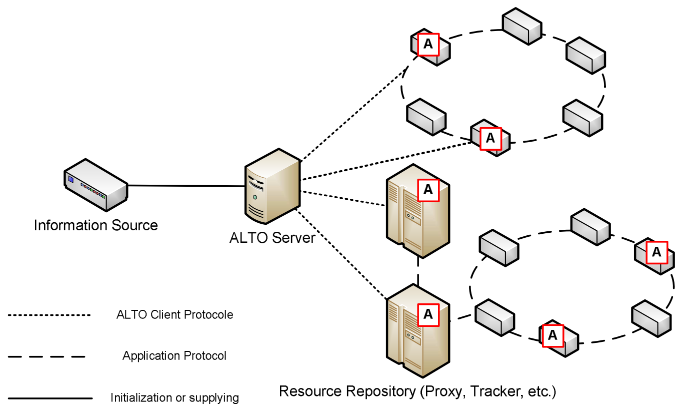


Figure 5.1: ALTO architecture

P4P Specification [XIE&al07]

The ALTO Project defines the P4P specification following the traffic problematic in P2P applications. In fact, traffic control is a real challenge in P2P. First for the intra-domain exchange, the strategy of several P2P applications causes dispersion and crossing paths unnecessary traffic within the same ISP. It is possible according to some calculations to reduce until a rate of 0.8 the number of hops traveled without degrading application performance.

Second, for inter-domain cases, P2P network can generate a significant volume of Internet traffic or route traffic between operators. Study shows on BitTorrent that 50 to 90% of the local tracks for active users are from outside. It is also the case for ISPs which does not compensate access provider for P2P traffic which can cause a significant imbalance resulting in breach of peering. Such inefficiency of inter-domain traffic can disrupt the ISP economics.

ISPs try to estimate traffic patterns and determine routing based but all of these efforts could be avoided if P2P traffic adapts their network changes and so this would thus result to potential oscillations in traffic and route decisions.

Generally, the P2P model exposes a fundamental problem related to the control of Internet traffic: the emerging applications can be flexible in the way the data exchange is done. If end users are encouraged to participate in resource optimization systems could not continue to be opaque but will need to provide a communication channel for traffic control collaboration.

P4P is a simple architecture that presents many interfaces for communication between networks and applications: static network policy, p4p-distances reflecting network policy and network status, and network capabilities. The objective is to allow network providers and applications to optimize their performance while preserving privacy.

The p4p-distance interface is the interface through which a service provider shall inform the applications about cost in the same AS or on the inter-domain links. The p-distances gives an overview of the network status and preferences for the application traffic and can be used to capture a number of interesting metrics like for instance the maximum utilization of backbone and favorite cross-domain links. Applications use these distances to form the connectivity and communication modes to choose efficient network when it is possible.

Three plans are proposed in the architecture: the Control plan, the Management plan and the Data Plan that is optional.

The management plan has a monitor function for the control plan behavior. The control plan is the most important and in this plan P4P introduces the concept of i*Trackers* that divides responsibilities traffic control between applications and service providers and this makes P4P a scalable and easily deployable. Each network provider can be commercial, a university, etc. It maintains an i*Tracker* in its network. This i*Tracker* offers an information portal on the network provider. To get the address of the i*Tracker* you just need to run a DNS query. It can also have multiple i*Tracker* in the same area for security reasons and scalability. This represents the application is what we call the app*Tracker* for P2P. Taking the example of an application based on P2P like Bittorrent *Tracker*, app*Trackers* interact with i*Trackers* and distribute information of P4P control plane peers while for P2P application without *Tracker* and that does not have central app*Tracker* but depends on a DHT, peers given the necessary information directly from i*Trackers*. In both cases, peers can assist in the distribution of information (via the gossips). The i*Tracker* can be run on a trusted third party rather than by the supplier itself. It may also be an integrator that meets the aggregation of information from several i*Trackers* for interaction with applications. P4P does not dictate exactly the same information but provides a common messaging framework.

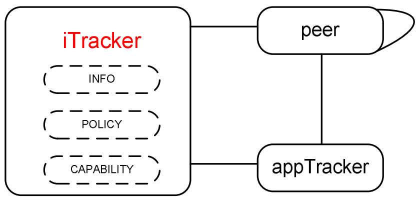


Figure 5.2: *iTracker* interface in P4P

***iTrackers* interface examples**

Policy interface enables applications to obtain network usage policies. To give two examples of policies for network usage policy, we use coarse-grained time-of-day to identify the different uses of specific links or congestion and levels of high use.

P4p-distance interface allows others to query cost and distance as peer networks. Interface capability allows others (peers or content providers) to request the capabilities of network providers. For example, a supplier may offer different classes of service, application servers or caches in its network. An app*Tracker* can ask several i*Trackers* in different popular areas for the application servers or caches that could help the acceleration of P2P content distribution.

A supplier may choose to integrate multiple interfaces and can also define access control to its interfaces for security reasons. For example, a deployment model can be established where ISPs restrict access only to trusted app*Trackers* and integrators. The supplier may also lead to an access control content (for example, the interface capability) to avoid being involved in the distribution of content.

In the following example, app*Tracker* sends a query asking i*Tracker* B Network Provider to reserve capacity for distributing content. This allocates the server at the network and returns its address to the app*Tracker*. This one will be able to insert the server for peers sets returned to peers in network B.

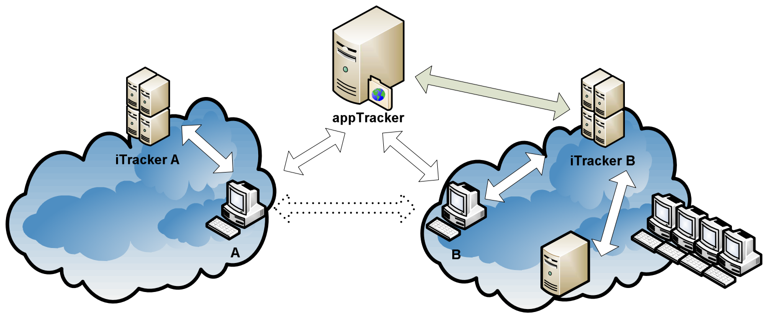


Figure 5.3: P4P architecture

The *iTracker* is decomposed as a network topology G = (V, E) with V a list of nodes and E is the list of links. V is characterized by an opaque ID: the PID. PIDs can be composed by several different types. The first type is an aggregation node that represents a set of customers. A PID aggregation may represent a network Point of Presence (PoP) or a set of customers with a network status (for example the same level of congestion). A PID is a PID aggregation visible from the outside. A client made the request to the network (to the i*Tracker* or the supply system) to map its IP address and the PID number of the AS. If the mapping of IP to PID is dynamic, the client can be refreshed periodically mapping. The i*Tracker* provides p-distance on each link of the level of PID. The external view of the i*Tracker* applications is a full mesh and an application may be just interested in a subset of these peers.

5.2.2 The SmoothIT architecture

The SmoothIT [PUS&al09] project objective is to define and develop some mechanisms called Economical Traffic Management (ETM) to optimize the traffic impact on overlay applications on ISPs. The method is to let network operators cooperate and application users. When the overlay is provided by another operator than the network operator, which is generally the case, we also have the overlay operator included in this cooperation. SmoothIT consortium launched a deep study of the requirements for P2P applications and their generated traffic. They classified overlay applications and gave an overview on overlay applications describing their different features and characteristics (in terms of traffic especially). They provided a technical, economic and functional study of overlay application. They finally define the requirements to design a specific architecture that ameliorates P2P traffic via cooperation between the different parts.

The first point was to take seven different applications types and some overlays application example to know what is the best way to implement ETM more efficiently. Then they specified some classification criteria indicating the overlay application relevance for the consortium that are finally used for the application selection. Then, after a deep study, some real requirements have been derived. Finally, they present an application classification for P2P streaming applications.

The application types and application examples chosen are File Sharing (eDonkey, BitTorrent, Gnutella, etc.), P2P Video-on-Demand (Vuze, PeerCast, End System Multicast, etc.), P2P Live TV (PPLive, etc.), P2P Voice-over-IP (Skype), P2P Gaming, CDNs (Akamai) and VPNs (Hamachi).

The criteria are: Technical Criteria (Traffic intensity, Source Code Availability, Traffic recognition and Emulation), Optimization Potential (Locality Information, QoS Provisionning, and End-user Controllability) and finally Non-Technical Criteria (Popularity and ISP Costs, Additional Charging, and Legal Content).

Based on this study, SmoothIT proposes an overall solution based on some attributes that have been selected in order to perform an analysis of their solution: legal issues that evaluate the viability, feasibility to deploy the architecture, complexity and scalability to show that the proposition is not too much complex to be deployed, optimization potential to evaluate the improvement in the overlay application domain and finally the innovation.

The components of the architecture are the following:

- a SIS Server is the core entity of the architecture and it is responsible for receiving request from the overlay application, to perform calculations based on the different applied policies.
- the Configuration Database that stores ISP policies that an ISP can configure for the SIS architecture.
- The Metering component that collects any information from the network that is required by the ETM and all other components.
- The Security Component that performs some security mechanism.
- The QoS manager that checks availability of all the network resources and permits to guarantee them if necessary depending on the application.

The approach used by the SmoothIT is a scenario based on a honey pot of the design space. This approach shows a high optimization potential but the problem is the deployment potential and the technical complexity and face to the legal issues. The components are presented with their interaction and a specification of a simple initial version of a protocol is also provided.

5.3 Why proposing SPOP ?

Most of current researches in P2P networks are focused on traffic management and locality aware techniques to control the Peering activity. In [BIN&al06] and [PAP&al06] the peering selection has been modified to choose intra-domain peers and decrease the traffic exchanged between ASs. These techniques propose a new concept that is the cooperation between ISPs and P2P applications. Various works also propose to optimize BitTorrent performance with geographic locality based selection without necessarily having this cooperation. For instance Ono [CHF&al08] and TopBT [REN&al10] use CDNs information to optimize download completion time focusing on performance purposes without any structured architecture for ISPs. These propositions are client oriented.

The architectures that are close to our proposition are ALTO P4P [XIE&al07] and SmoothIT [PUS&al09] that we described previously. First of all, these architectures are focusing on P2P traffic and QoS issues while SPOP proposes a transport optimization and a Context-Aware routing plan in addition to the Control/Management control with complete interoperability and transparency for any service application that would be designed under this architecture. We can note main point concerning our choice as follows:

- P4P proposes cooperation between ISPs and P2P applications in order to accelerate download and optimize network resources utilization. A control plan and an optional data plan are defined. The *iTrackers* allow P4P to divide traffic responsibilities between P2P and ISP. P2P applications have *AppTrackers* that communicate with *iTrackers* to obtain information on peering decisions (network topology, provider policies and capabilities). One of the problems with P4P is that it might slow down transfers of non P4P customers even if it helps illegal file sharing. Second, the cooperation and information sharing is a novel idea but no incentives have been proposed to motivate customers to share these information. Technically, this seems difficult to integrate to both parts.
- SmoothIT share similar key objectives as P4P but SmoothIT framework is more detailed even if SmoothIT does not also provide a full architecture description. Comparably to P4P, SmoothIT provides specification for cooperation between ISPs *Trackers* and for protocols. Moreover, SmoothIT takes into consideration other applications than file sharing and considers real time constraints applications. The problem with SmoothIT is that it needs great modifications into Internet entities like routers due to the complexity of the architecture.
- SPOP considers 3 main aspects that are simplicity, performance optimization and mostly interoperability with the existing protocols: the routing plan is based on a Context-Aware algorithm that can take into consideration various parameters to regroup peers and not only the domain membership like P4P and SmoothIT, the transport plan proposes a FEC mechanism that ISP can provide to accelerate data transfer in some lack of resources cases, and finally the Control/Management level is based on existing parameters (Peers selection policy, Peers Set size, Piece size, etc.) that can be adjusted without adding complexity to the Internet and ISP infrastructure (that would introduce extra costs when new entities have to be integrated).

5.4 Service Provider Oriented P2P (SPOP): a simple, robust and interoperable architecture

5.4.1 SPOP architecture plans

The SPOP architecture is designed to provide an optimized platform for P2P applications that are service provider oriented. Nowadays the integration of new services and applications is a big challenge: each application has its proper constraints and functioning and ISPs must be prepared to provide the best environment for their deployment.

We previously discussed the importance and the difficulty to manage P2P applications and services and mainly the traffic that they can generate. The proposition with SPOP is to define an architecture that optimizes the three main components of a P2P application that are: the transport, the routing/lookup and the service. For the service component we saw the traffic management in BitTorrent-like protocols that we will complement with some contributions on existing BitTorrent parameters; for the transport component we studied the positive impact of FEC mechanism as a transport accelerator; for the routing/lookup component we will introduce a mechanism that generalizes our work on traffic management. In fact it is a context-aware algorithm that will also optimize the data indexing and lookup in addition to partitioning the traffic when the context is the AS membership studied in chapter 3. In the following Figure 5.4 we present the different SPOP plans with their main objectives.

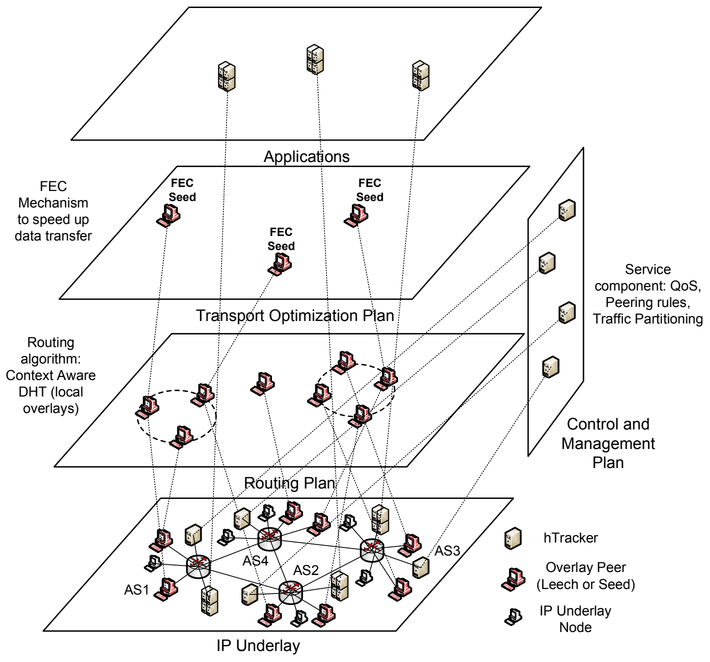
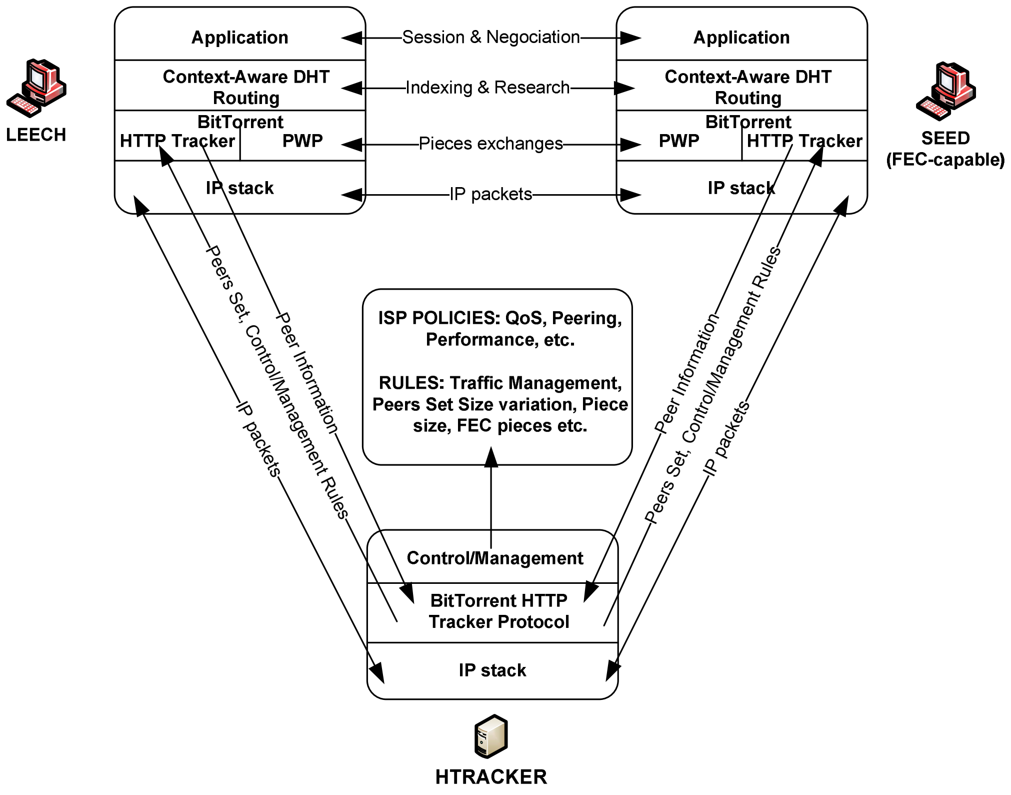

Applications
FEC Mechanism to speed up data transfer
FEC Seed
FEC Seed
FEC Seed
Transport Optimization Plan
Routing algorithm: Context Aware DHT (local overlays)
Routing Plan
Service component: QoS, Peering rules, Traffic Partitioning
Control and Management Plan
AS1
AS4
AS2
AS3
IP Underlay
hTracker
Overlay Peer (Leech or Seed)
IP Underlay Node

Figure 5.4: SPOP architecture

The main goal in proposing this global architecture is to provide a simple and interoperable architecture. Optimizing P2P applications must keep the functioning of existing protocols and architectures without degrading their performance. Dynamicity and heterogeneity of P2P model force every work and proposal to be interoperable with generic foundations of this model which has proved its success.

Interoperability is assured in our architecture by the fact that BitTorrent main protocols and rules have not changed. We present in Figure 5.5 the 3 main entities that compose BitTorrent with the light modifications that do not affect the current functioning of the protocol. We see that some peers in the network are integrated to provide some FEC pieces. These peers are the *Seeds* as we studied it in chapter 4. For the routing plan, we integrated a Context-Aware routing DHT algorithm that will perform lookup optimization in every peer of the architecture (*Leech* and *Seed*). We can note that some BitTorrent clients already exist based on a DHT algorithm: Kademlia [MAY&al02]. The clients that propose this feature are various: Vuze, rTorrent, µTorrent, BitComet, KTorrent, etc. These clients are Trackerless but we voluntary let the *Tracker* in our architecture while this *Tracker* called *hTracker* has an extraordinary Control/Management plan that keeps ISP Policies and that dynamically applies some rules on the peers behaviors: managing the traffic when Peering strategies change, varying BitTorrent parameters like the Peers Set size or the piece size, and this without disrupting the network functioning or any need to modify IP entities like routers. These are the main arguments that motivate us to propose such architecture.



Application
Session & Negociation
Application
Context-Aware DHT Routing
Indexing & Research
Context-Aware DHT Routing
LEECH
BitTorrent
HTTP Tracker
PWP
Pieces exchanges
BitTorrent
PWP
HTTP Tracker
SEED (FEC-capable)
IP stack
IP packets
IP stack
Peers Set, Control/Management Rules
Peer Information
IP packets
ISP POLICIES: QoS, Peering, Performance, etc.
RULES: Traffic Management, Peers Set Size variation, Piece size, FEC pieces etc.
Peer Information
Peers Set, Control/Management Rules
IP packets
Control/Management
BitTorrent HTTP Tracker Protocol
IP stack
HTRACKER

Figure 5.5: SPOP entities

5.4.2 A Context-Aware Routing plan

We saw in chapter 3 how partitioning traffic in a BitTorrent network can increase the download performance and control the Peering traffic. In this study, the parameter that permitted to decide how to choose the peers were the AS membership.

We propose in the following a solution that generalizes the precedent work where peers are regrouped by different other specific parameters. The principle is to implement a DHT that takes into consideration a network context. This context can be the AS membership like in the traffic partitioning contribution defined in chapter 3. In addition to increase download performance and decrease peering costs, the DHT characteristics ensure a lookup optimization. The DHT that is defined is an improved Pastry DHT, where a global network that was usually structured in one ring, is formed by multiple local rings. The number of rings depends on the number of groups that are formed in the network. This number depends on the context that an ISP would decide to create. In fact a group can represent an AS, or a group sharing the same traffic type, or a group sharing the same interest, or even a group sharing the same performances. We will show how this Context-Aware DHT allows to optimize resource lookup and to decrease considerably messages overhead in addition of presenting the same advantages presented in Chapter 3.

The objectives are the same and are to introduce cooperation between ISP and P2P application in a way of optimizing performances and decreasing economic costs due to over traffic: signaling traffic and peering data traffic.

The principle in creating a Context-Aware P2P is to enhance routing process in a DHT algorithm. The contribution we will present next is detailed in [FAY&al08] but has not been validated by simulation or implementation. We propose in this section to show some simulation validation for this proposition. The idea is to consider a generic DHT implementing Pastry for instance and to integrate a semantic that allows the algorithm to take into consideration the underlay network in the routing process following some parameters. These parameters can be various and can be simple or complex. The main change comparably to a DHT is that we replace the hash function by HMAC function studied in chapter 3. In fact, we previously integrated a semantic to the *peerId* to associate a peer to its AS membership while here we generalize this concept by associating a peer to any other parameter that makes it belong to a group (sharing the same interest, having the same performances, appertaining to the same AS like in chapter 3, etc.). The advantage is that we also add a DHT based routing that optimizes the lookup performances. We will also see that the Context-Aware DHT proposed shows even better performances than a generic DHT. Here are the proposition parameters:

- The hash function is still the same as the implemented DHT.
- *Km* is a specific key that characterizes the context. This key is the same as in chapter 3 but here the context is more general that the AS membership and can represent any other context.
- The generation of every *nodeId* depends directly on this key *Km*. Comparably to the proposition in chapter 3 the *nodeId* will undergo the same process as the *peerId*. However, in the DHT context, a major difference is that the value of the ID has an importance because it determines the node position in the ring and when the *objectId* are generated they are managed by the nodes depending on these values. On the contrary in chapter 3, the *peerId* value has no real importance while it identifies the peer and its AS membership. This is why we decide in the Context-Aware DHT to only use the context key *Km* to generate the *nodeId* while in chapter 3 the key *Km* was used to generate a key *Kd* used to generate each peer *peerId*. Key *Km* is used here to generate *nodeIds* and *objectIds* in a specific network that implemented our architecture.
- A key *Km* can be simple if it represents a unique criteria or parameter. In *hTracker* traffic partitioning proposition *Km* was a simple key that represented the AS membership. This key can also be a locality parameter, the application type, a communication group, etc.
  Key *Km* can be composed if it represents many parameters and so on can be derived into many simple keys.

The advantage of using and HMAC function is that the result has the same size as the hash function, which is totally transparent and interoperable. However, the advantage is that the key *Km* can here associate a peer to the context it belongs to. The goal in the Context-Aware DHT is to provide a local routing in every context before performing the global routing that is launched only if locally the resource has not been found. The routing process and algorithm do not change but are targeted to local groups before being executed globally if the resource is not detectable locally. We will demonstrate by some simulations that searching locally a resource in each context is more efficient in terms of performance and messages overhead than searching it in the whole network. In fact, imagine an IP underlay network constituted of hundreds or even thousands of peers, regrouping peers by some criteria that they share and performing resource lookup in the created group (that are obviously smaller than the overall network) make participate less peers in the lookup process: the number of hops the DHT routing process will generate is automatically smaller and the signaling traffic will also be reduced.

We consider in this architecture that a peer is active globally in the network but can be part of one or more secondary overlays that are the different groups they belong to. We will mention local overlays where the same routing algorithm used in the global network is also used locally. Next Figure 5.6 illustrates the principle of local context regrouping.

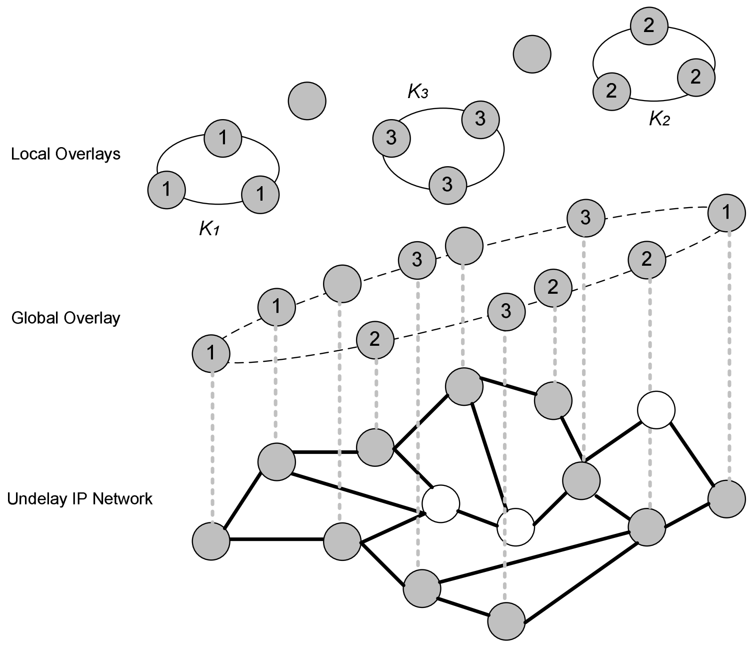


Figure 5.6: Context-Aware DHT model

The Global Overlay is divided into 3 local overlays while some nodes in the Global Overlay can be free without being associated to a specific zone. In this case, the lookup resource is done at the Global Overlay level. A peer can be associated to many local overlays. The lookup is first executed locally, and then tested globally if the first step failed. In the previous figure, we take a simple example with 3 groups or local overlays. We note that if a node belongs to more than one local overlay, it must have one *nodeId* per local overlay it is associated to. To avoid conflicts problems, we suggest using the HMAC key generation only at local overlays.

Routing in the Context-Aware DHT is the same as in the global level but applied locally. When a peer is requesting an object (resource) it begins by calculating its key using local overlay *Km* where it is. Then, it launches a request in this overlay using the corresponding DHT algorithm. Then the request will be done only at the local overlay. If the request is not a success, then it is done at the global overlay using the algorithm and the table that belongs to the global overlay. When the node finds the resource only at the global overlay it executes an *objectId* calculation to share the resource locally using *Km*. If for instance a node in local overlay 1 is in action, it would use *K1* to calculate the *objectId* of the resource that has a different *objectId* in the global overlay.

The Context-Aware DHT contribution requires managing the nodes joins and leaves since the overlay is divided into many other overlays. The problem is that when a peer decides to join a local overlay, it must verify if an active peer exists in this local overlay. The first step is to identify the different local overlays it belongs to. In the case where a local overlay the peer wants to join does not exist, the peer has to create the necessary environment and a local table called the Zone Table that identifies the zone and that references few nodes belonging to this local overlay. If the local overlay already exists, the peer retrieves the existing Zone Table.

The advantage of this proposition integrated at the P2P Routing plan allows adding a DHT that is Context-Aware. The DHT algorithm is still the same even if some light modifications are applied. These modifications are related to the semantic and the ID generation.

Experiment: We simulate a network *N* made of 1000 heterogeneous *Leeches* and 10 *Seeds*.

- *Scenario 1:* the network is a generic BitTorrent network.
- *Scenario 2:* we implemented an original Pastry algorithm (where all peers are in a unique global overlay. Our choice for the DHT is bambooDHT [RHE&al05] which is the standalone version of OpenDHT [RHE&al05]

- *Scenario 3:* we implemented the Context-Aware Pastry by modifying the bambooDHT API to simulate local overlays.

In the three presented cases we retrieve information on the completion time of the resource download and the number of messages that were generate for every lookup process.

In the third case the configuration was a global overlay made of 5 local overlays and 100 single *Leeches* that are not member of any local overlay. These local overlays can be characterized by any context parameter like for instance the AS membership (*cf* Chapter 3). This work would be a generalization of Chapter 3 contribution with the advantage of integrating not only a simple DHT routing algorithm but also a Context-Aware one.

The following table is the *Leeches* and *Seeds* distribution in *N* for scenario 3:

Table 5.1: Peers distribution in N for Context-Aware DHT simulation

AS	Peers
AS	400 *Leeches* + 4 *Seeds*
AS1	300 *Leeches* + 3 *Seeds*
AS2	100 *Leeches* + 1 *Seed*
AS3	70 *Leeches* + 1 *Seed*
AS4	30 *Leeches* + 1 *Seed*
AS5	100 *Leeches* + 1 *Seed*

The Download/Upload capacities distribution is the following in each AS to simulate heterogeneity.

15% of 200 kbps/100 kbps, 15% of 400 kbps/200 kbps, 15% of 500 kbps/400 kbps, 15% of 600 kbps/500 kbps, 15% of 800 kbps/600 kbps, 15% of 1000 kbps/800 kbps and 10 % of 1100 kbps/900 kbps.

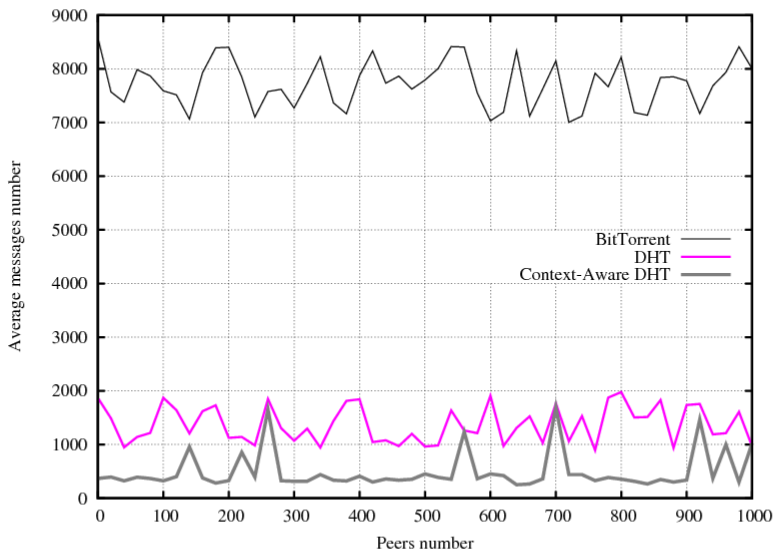


Figure 5.7: Average message number comparison between BitTorrent, generic DHT algorithm and Context-Aware DHT

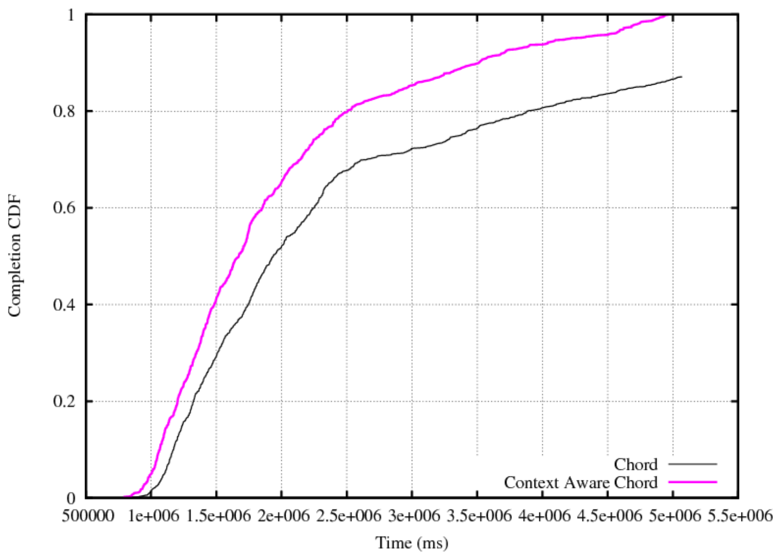


Figure 5.8: Completion download time comparison between generic DHT algorithm and Context-Aware DHT

In this experiment we evaluate the impact of restricting piece lookup and peering traffic exchange in each local overlay a peer is member of. In the first scenario the peers' communication is totally free so that no restriction has been fixed in the choice of the peers.

In the second scenario we only integrated a DHT algorithm to optimize the lookup component.

In the third scenario we did a hypothesis: a peer first chooses to search for peers that are inside its local overlay then if the search is not a success, the peer contacts other peers in the global overlay.

The Figure 5.7 shows that Context-Aware DHT minimizes considerably the message number generated for resource lookup. This solution is also better than implementing a generic DHT. We also retrieved the completion time for each *Leech* in Figure 5.8. In the third scenario, we grouped and ordered the results to compare them with Scenario 1 results in a CDF curve for each of the second and the third scenario. Let's note that for the download completion time, there is no real difference between scenario one and scenario two while the only difference is the way the resource and the node are founded. For the average number of messages implicated in the lookup process, we compared the three scenarios.

We note that for a 5000 seconds simulation, in Scenario 2 the completion reaches only 0.83 while for Scenario 3, all *Leeches* have completed their download. We can also note a gap between the two curves and this gap is increasing over time. In fact, at the end of the simulation, we intuitively know that *Leeches* that have the worst capacities are the ones that finish their download late. The more a *Leech* is weak in link capacity, the better it will be advantaged by Scenario 3. It seems easy to understand because in Scenario 2, a *Leech* requests pieces from peers that are in the same local overlay, so that are physically closer.

This Context-Aware DHT is also a good solution to decrease the Peering traffic when the different groups are ASs. This conclusion is the same as the one obtained in chapter 3 concerning Peering traffic exchanged.

5.4.3 ISP oriented FEC service for the transport plan

We propose in this section to present the impact of FEC mechanism as a provided service. As we saw previously, one of the principal issue of SPOP architecture is to provide a solution for peering traffic between ASs while optimizing the pieces transport and keeping interoperability with classical BitTorrent networks. We evaluated in chapter 4 the integration of FEC mechanism in BitTorrent protocol and its impact on completion time and we showed that FEC acts like a download accelerator. In the following the objectives are to demonstrate that the instantiation of *Seeds* that integrate FEC mechanism in a specific network can optimize the Peering traffic management and increase data transfer performance. FEC is proposed as a service that an ISP can propose for some economic reasons.

Experiment

We take the same peer distribution in *N* and we simulate a network where FEC mechanism an optional service provided by a specific network. We suppose that a subset of *N* called *N2* is one of these ASs. We have *N=N1 U N2* and take we *N2=AS3* is composed by homogeneous *Leeches* (1500 kbps/1200 kbps). *N2* is a network that provides only FEC pieces available for all peers of the system. FEC is provided by *Seeds* because to provide FEC pieces that can replace any of the original pieces, the source must have the entire resource, which is the case of *Seeds*. We fix the FEC ratio to 1.3 and vary the *Seeds* number (*x)* to see also how *Seeds* number increase can has some consequence in the performance of FEC integration in this specific case. *Seeds* have the same capacities as in Experiment 1.
We compare this scenario 2 to scenario 1 that is the same as Experiment 1 Scenario 1 with different *Seeds* number value (*x*):

- *Scenario 1:* The same as Experiment 1 with the *Seeds* number *x* varying.

- *Scenario 2:* The piece lookup for a specific *Leech* is restricted to the AS it belongs to.

Table 5.2: Peers distribution for FEC service solution

AS	Peers
AS1	400 *Leeches*
AS2	300 *Leeches*
AS3 = *N2*	100 *Leeches* + *x Seeds*
AS4	70 *Leeches*
AS5	30 *Leeches*
Others in *N1*	100 *Leeches*

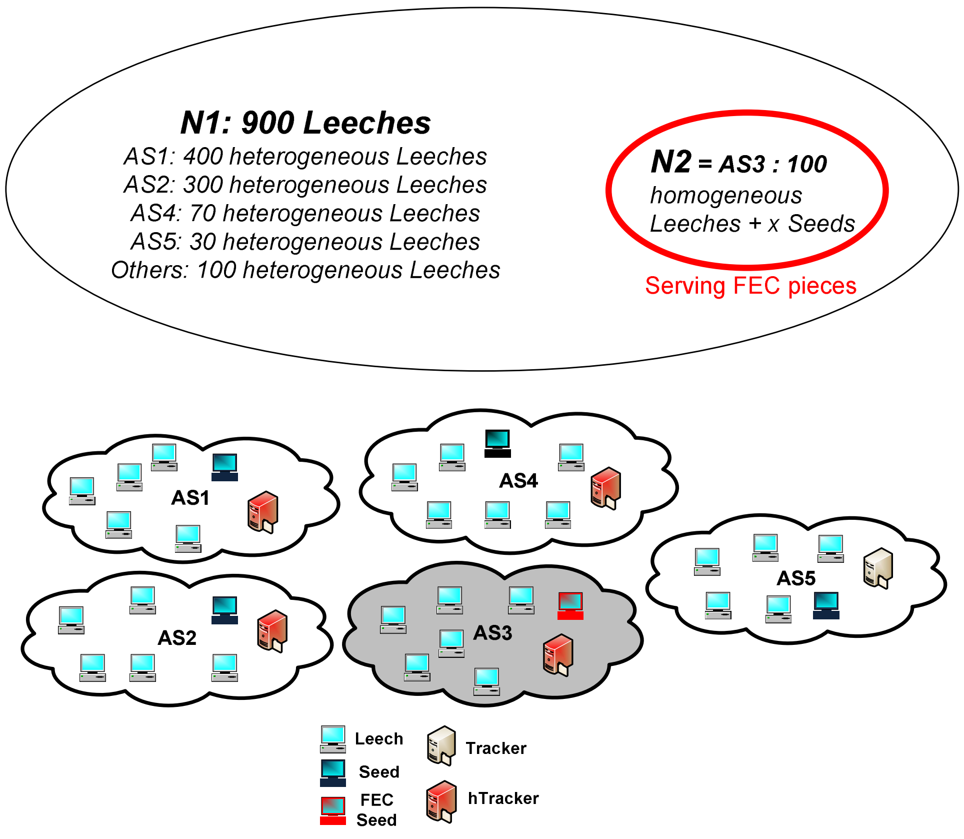

N1: 900 Leeches
AS1: 400 heterogeneous Leeches
AS2: 300 heterogeneous Leeches
AS4: 70 heterogeneous Leeches
AS5: 30 heterogeneous Leeches
Others: 100 heterogeneous Leeches
N2 = AS3 : 100 homogeneous Leeches + x Seeds
Serving FEC pieces
AS1
AS2
AS3
AS4
AS5
Leech
Seed
FEC Seed
Tracker
hTracker

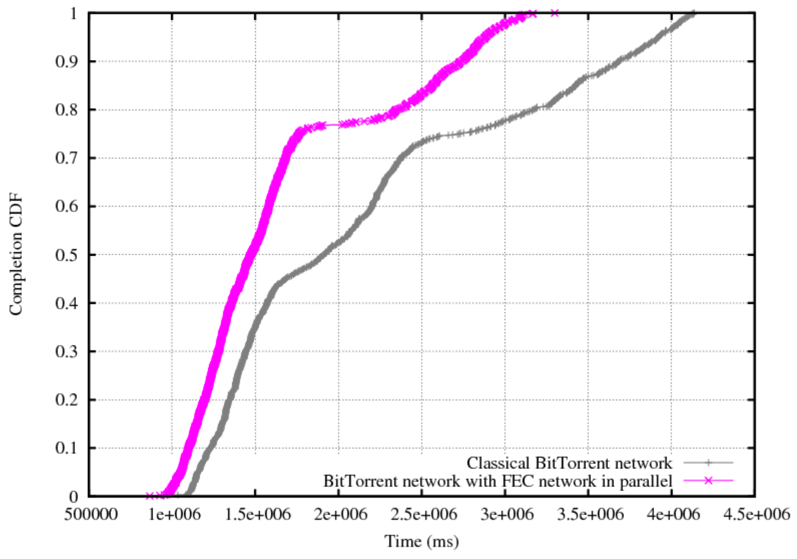
Figure 5.9: Network configuration for FEC service simulation

The interoperability is visible here whether for the *hTracker* solution or the FEC one. AS5 is devoid of both solutions we propose and peers in this AS can exchange data from others and even take profit from AS3 FEC service.

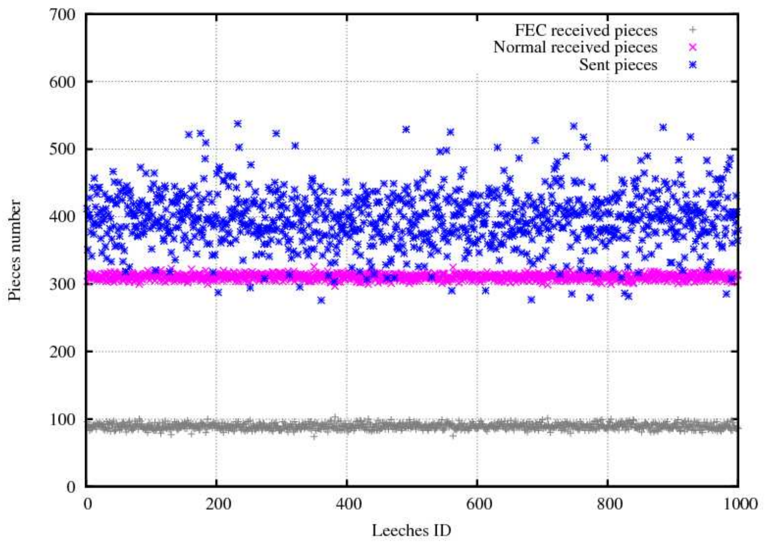
The experiment proposed here has for objectives to evaluate the FEC mechanism as a service provided by an ISP for instance. We will see that this has an impact on completion times for peers and peering traffic exchanged between the ASs. We also decide to vary the *Seeds* number *x* in the AS3 that provide FEC pieces. In this configuration all peers can download pieces from all other ASs and of course from AS3.

In the first simulations, we fix *x=1 Seed*. We note a real difference for completion times: more than 20 % most of the time. For traffic exchanged between ASs in Scenario 2, we retrieve information on the pieces number that every *Leech* downloadw from *N2* and from *N1* network. We note that the portion of pieces download from both parts is the same for every *Leech*. Each *Leech* downloads between 80 and 100 pieces from *N2* and so on 300 to 320 pieces from *N1*. The portion downloaded from *N2* represents approximately 30% of the pieces.

We vary *x* taking values 5 and 20 and we note that the same portion of FEC pieces is downloaded. The difference is observed in the Sent pieces number and this is due to the increase of the *Seeds* number. The *Seeds* send more pieces and this decreases the *Leeches* participation.

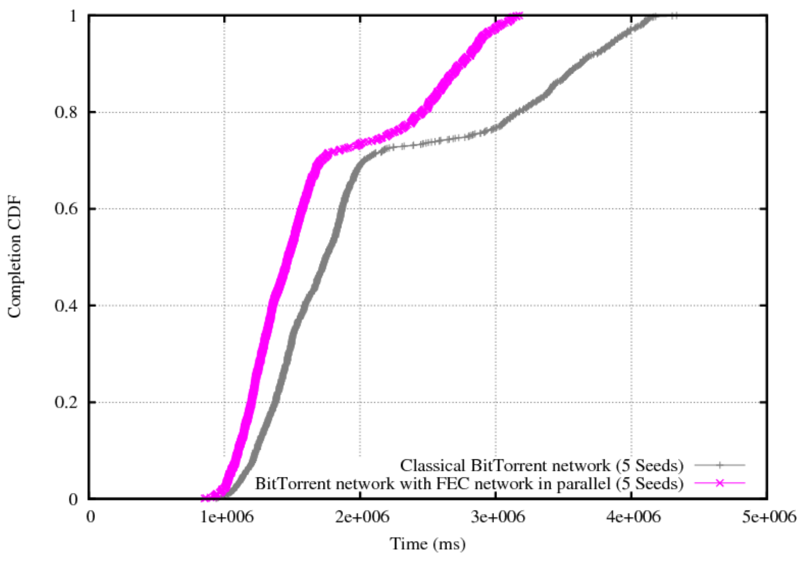


(a) Completion CDF

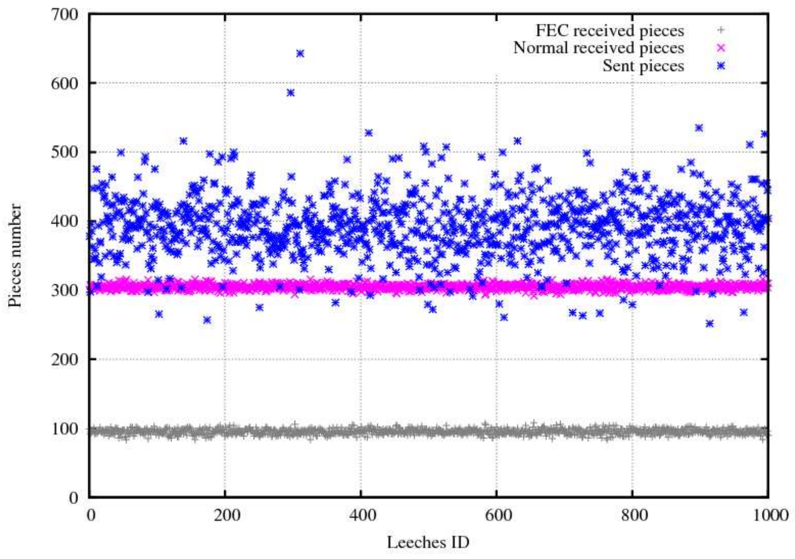


(b) Pieces distribution

Figure 5.10: 1 *Seed* simulation for FEC service

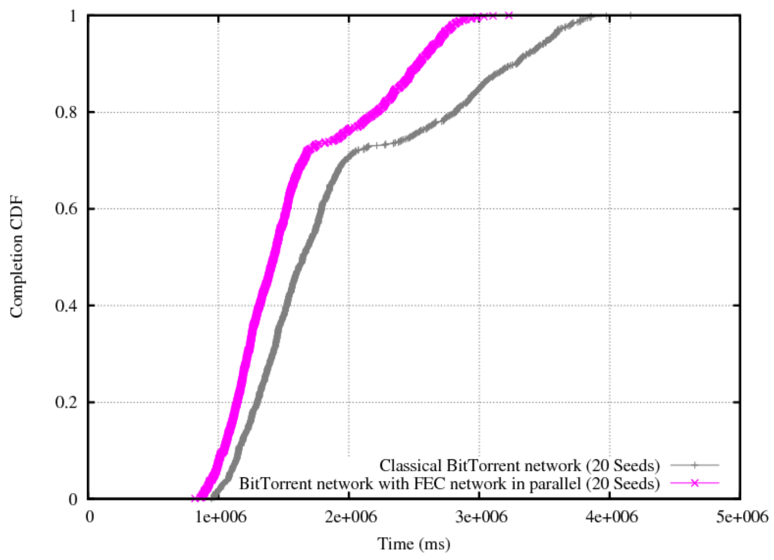


(a) Completion CDF

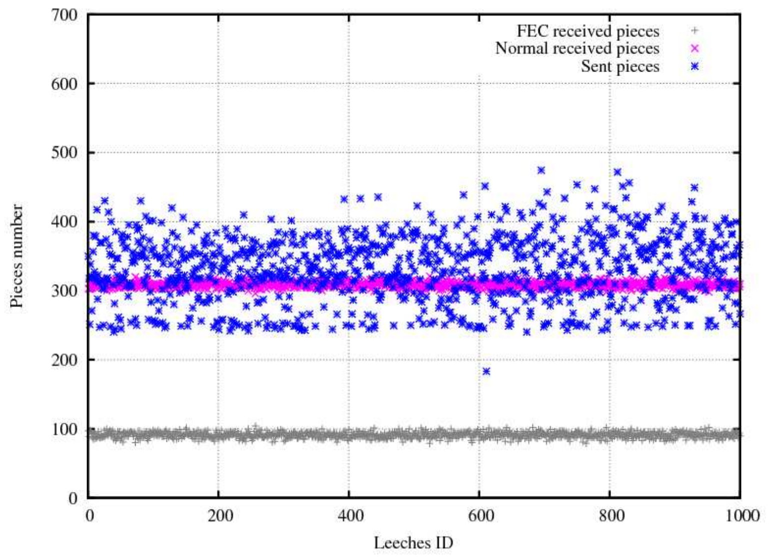


(b) Pieces distribution

Figure 5.11: 5 *Seeds* simulation for FEC service



(a) Completion CDF



(b) Pieces distribution

Figure 5.12: 20 *Seeds* simulation for FEC service

Following are the advantages and the drawbacks of this proposition where a FEC network is feeding peers with pieces to accelerate the resource download. The evaluation study shows that we can reach more than 20 % gain in the completion time and the peering traffic taken from the FEC service provider is approximately equal to the FEC ratio available, which is 30% in our case. We have 25% of the whole traffic that is FEC. This is an interesting way to reach better performances with some economic interest for the ISP that decides to provide this FEC service. However, the major point in this study is the interoperability of this solution. In fact we do not have to change the BitTorrent architecture. The only change is the implementation of FEC mechanism in some *Seeds* with an optimized ratio. We saw in Chapter 4 that when the ratio is exceeding a limit, some overhead can disrupt the system overall performance. The FEC is applied at the piece level which is easier to manage and this method costs less than applying FEC to the blocks, the treatment necessary would be higher.

5.4.4 Control/Management plan

This plan integrated to SPOP architecture is based on the *hTracker* entity. These control entity receives reports from peers and manages the download and upload activity of every peer in the network for a specific torrent. In section 2.3.2 we saw that the *Tracker* in BitTorrent keeps interaction with the peers and receive some message with an interval that can vary. This interval in our architecture must be reduced to its minimum to allow the *hTracker* to receive information concerning the peers' activity.

SPOP Control/Management plan is interoperable and the BitTorrent protocol specification is not changed at all. We propose in this plan to work on the following main parameters:

- Peers Set selection policy with *hTracker* policy that propose a neighbor selection inside the same AS. We can generalize this idea here and propose a selection inside the same context or local overlay since the Context-Aware DHT is integrated at the Routing Plan.
- Peers Set size: some simulation results show that varying the Peers Set size can impact download duration. *hTracker* can vary this Peers Set size to establish an balanced activity among the peers. Some fast peers can need less activity than low peers in real time applications.
- Piece size: increasing the piece size impacts on the download completion time positively but the only reason this parameter is not always varied is that the less the large piece size, the more pieces number would be high. This implies more treatment since every piece is checked for integrity after being downloaded.
- The FEC ratio: when FEC mechanism is chosen in some *Seeds* for speeding up data transfer, a ratio must be fixed depending on parameters like *Leeches* and *Seeds* number, file size, etc. (*cf* section 4.3). *hTracker* entity that has a general view can be the intermediate control entity between the ISP and the peers to configure the FEC ratio that the *Seeds* will provide.
- *Seeds* number: we previously saw that the ISP can instantiate voluntary some FEC *Seeds* that would be compensating lack of resources or the needs for specific applications (like time constraints: IPTV or VoD). This parameter must be decided and controlled by the Control/Management level indirectly with the *hTracker* entity.

We note that these parameters are all simple parameters that can be easily varied without any more infrastructure deployment or physical entities change. That is the interoperable goal of SPOP architecture.

5.4.4.1 Peers Set size Variation

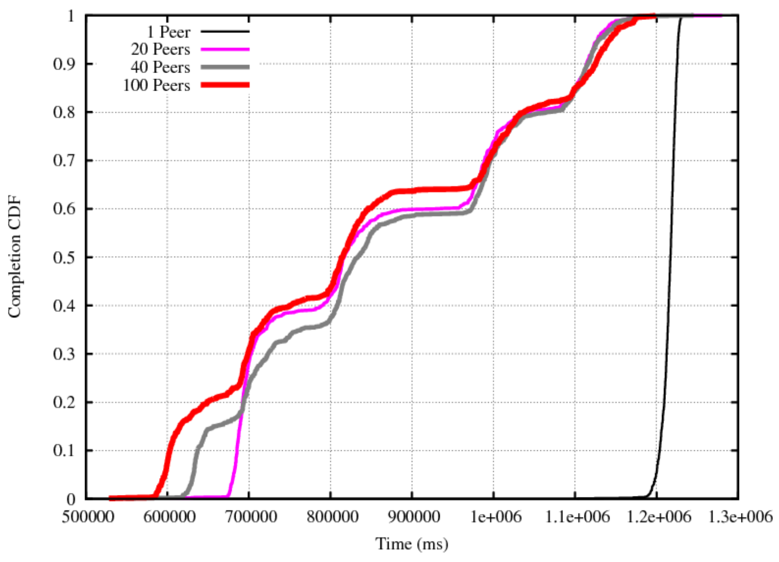


Figure 5.13: Peers Set size variation impact on completion download

We decide to study how the variation of the Peers Set size managed by the *hTracker* can change the download completion time. In fact, for some applications, we can imagine that some *Leeches* are faster than others. In a heterogeneous network and depending on the reports sent by the peers, the *hTracker* can decide to vary the *Peers Set* size and to associate a size to each peer. This list can be sent again during the download if a peer state changes. In Figure 5.13 we vary the *Peers Set* size and observed the comportment in a heterogeneous network (1000 *Leeches* and 20 *Seeds*). The results for a 100 MB resource size show that increasing the Peers Set size ameliorates considerably the completion time. This is due to the reciprocity in the *Choke Algorithm* that promotes the peers activity when the set of peers the local peer will communicate is higher.

5.4.4.2 Pieces size Variation

We study here the Pieces size variation impact on the completion time. The Figure 5.14 show how a 10 MB resource is downloaded depending on the piece size from 16 KB (which is the minimum in BitTorrent) to 1024 KB. We took the 10 MB size in order to simulate a sliding video window in real time application case. As previously we can imagine that in IPTV applications taking small piece size can optimize the download performance even if this increases the pieces number. The treatment for integrity verification would cost more in time and resource usage. This depends on the ISP objectives and the peers capabilities using the application.

In Figure 5.15 we represent the average completion time depending on pieces size chosen also between 16 KB and 1024 KB. In this figure we better saw how this variation can impact on performances.

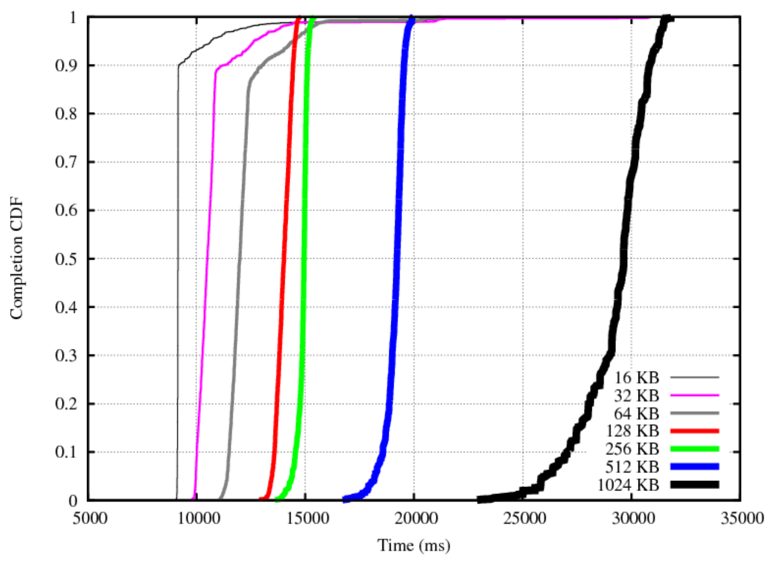


Figure 5.14: Piece size variation impact on completion download

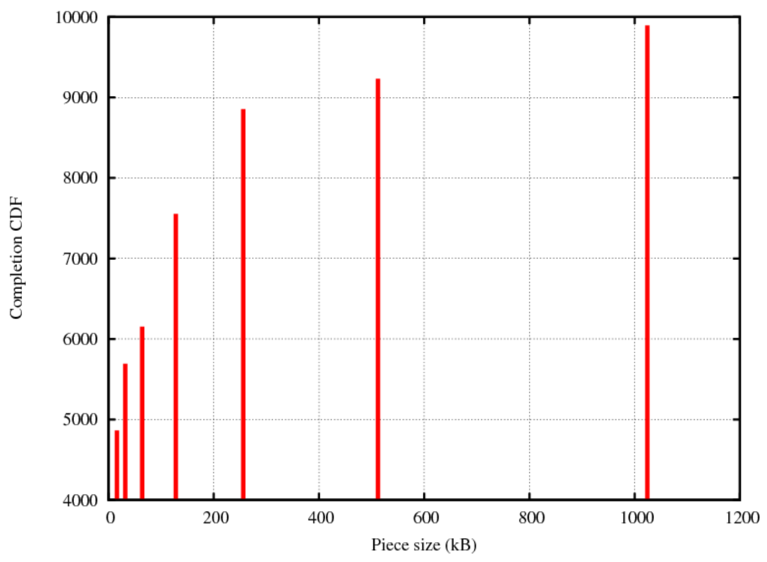


Figure 5.15: Piece size variation impact on completion download (average time)

5.4.4.3 Instantiating *Seeds* or *Leeches* to compensate lack of resources

Seeds in BitTorrent are essential while they have the entire resource. In a configuration where the network has enough *Seeds*, no lack of resources will be experienced. The problem is that in BitTorrent, *Leeches* are more solicited because the Tit-For-Tat policy in the *Choke Algorithm* is based on reciprocity. Leeches are downloading and uploading and *Seeds* are only providing pieces. It is common in BitTorrent that *Leeches* becoming *Seeds* decide to leave the network except in some Darknets [ZHA&al10] or networks that propose a recompense for every data volume provided. This is the case of Torrent411 for instance [TOR] where each client has a profile and gains some advantages depending on the Upload/Download data volumes ratio. This is a good way to motivate peers to stay in the network also after having downloaded the entire resource in the *Seed* state. In

[PAP&al06] the authors propose the insertion of ISP-owned peers to compensate resources download problems.

However, even when some *Seeds* are present in the swarm, certain applications need more availability also depending on the number of *Leeches* that want to retrieve the entire resource. This is why it could be interesting to propose that an ISP instantiates some *Seeds* that have the specificity to also provide FEC pieces when the network and the applications running need a certain availability ratio to guarantee best performance in terms of Download Completion time. For the peering traffic these ISP instantiated *Seeds* can motivate peers in the same AS or controlled by the same ISP to download pieces from those new *Seeds* instead of contacting outside peers that could increase the external traffic.

Table 5.3: Study of the *Seeds/Leeches* ratio for a 10 MB resource with the condition: completion download time < 5 s

Seeds/Leeches	1 initial *Seed*	10 initial *Seeds*	20 initial *Seeds*
Case 1 ≈ 50 *Leeches*	42/52 = 0.807	43/51 = 0.8431	48/52 = 0.923
Case 2 ≈ 100 *Leeches*	61/104 = 0.586	88/106 = 0.758	99/106 = 0.896
Case 3 ≈ 300 *Leeches*	236/334 = 0.706	270/333 = 0.811	266/334 = 0.796
Case 4 ≈ 500 *Leeches*	616/810 = 0.760	760/844 = 0.901	748/830 = 0.901
Case 5 ≈ 1000 *Leeches*	684/1000 =0.684	778/1000 = 0.778	801/1000 = 0.801

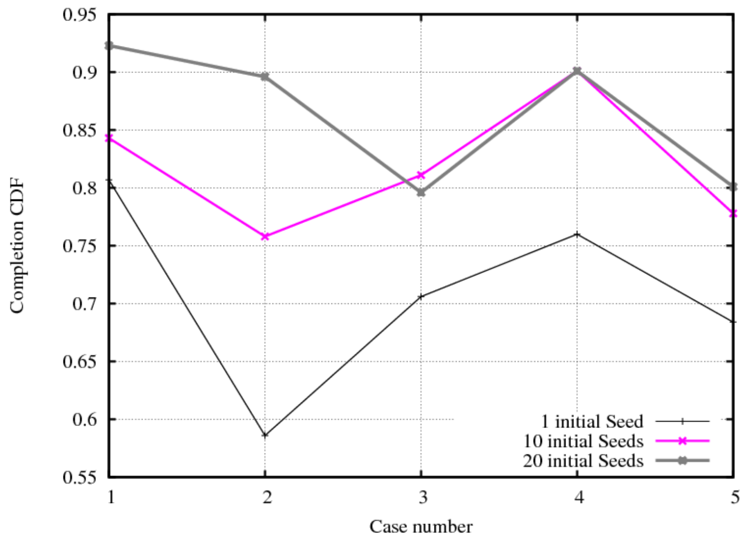


Figure 5.16: Study of the *Seeds/Leeches* ratio for a 10 MB resource

We saw in BitTorrent the importance of *Seeds*. In Figure 5.16 based on Table 5.3 simulation results, we study the impact of the *Seeds/Leeches* ratio variation on the completion in BitTorrent. In fact, a 10 MB resource representing a video window for instance must be downloaded in 5 seconds in these types of application. For each case, representing a *Leeches* number, we vary the initial *Seeds* (1, 10 and 20) and we calculated the percentage of *Leeches* that are respecting the condition: completion download time must be less than 5 s.

We noted that in the 1 initial *Seed* scenarios for each case the results are less interesting than for the 10 and 20 initial *Seeds*. In these two latter scenarios, case 1 and 2 are different while case 3 to 5 are similar. We can conclude saying that the *Seeds* number is primordial but when the number of *Leeches* is increasing, it is possible to have a performance gain thanks to BitTorrent Tit-for-Tat mechanism. Case 1 presents the best results and case 2 the worst. We can also see that the ratio is not enough, while for the same ratio but with different *Seeds*, number performances are not similar. Having a *Seeds/Leeches* ratio of 10/1000 is more interesting than having a ratio of 1/100 even if the ratios are equals. This simulation let us conclude also that instantiating *Seeds* is obviously interesting for the performances but instantiating *Leeches* can also help because *Leeches* are privileged when we know that they provide reciprocity and motivate the fast data dissemination. In fact, we see in case 4 that with 800 *Leeches* we have better results than in case 3 with 300 *Leeches*. However, the more *Leeches* we have, the more peers that need to download the resource we have. Case 5 with 1000 *Leeches* presents a more degrading case than case 4.

Chapter 6

Global network security system: an application

Intrusion detection and filtering are necessary mechanisms to secure networks which can be deployed in many ways. Nowadays, we are witnessing an important increase in attacks among which distributed denial-of-service (DDoS) that easily flood the victims from multiple paths. The major drawback of the existing detection techniques for DDoS attacks is that their entities work in isolation. In this chapter, we propose an efficient and distributed collaborative architecture that allows placement and cooperation among security defense entities to address the main security challenges. The use of content based DHT (Distributed Hash Table) algorithm permits to improve the scalability and the load balancing of the whole system. This modular architecture has been implemented on IDS (Intrusion Detection System) entities with the DHT Pastry protocol and has shown a promising performance. [PBL2] and [PBL3] are publications that present the work detailed in this chapter.

6.1 Introduction

In today's Internet, network security vulnerabilities are becoming common and attackers constantly modify their tools to bypass the security systems. The Internet paradigm was at the origin to avoid any control on the network traffic. This is considered as the strength of the Best effort IP architecture but can also be considered as a weakness at the same time. Many security solutions have been proposed but are still not perfectly efficient because of the time needed to detect the attack. The most dreaded attacks by current service providers are Denial of Service attacks and especially their distributed form (DDoS) [MIR&al02]. In this case, a large number of attackers are implicated in the process and make the detection more and more difficult and the impact is obviously bigger. The attacks originally exploited the weakness of the protocols; but now they start to attack the infrastructure of the Internet like the Web sites, banks and Internet service providers [MIR&al04]. Some technologies have proved an important evolution in the case of intrusion detection but at the same time the malicious intrusion becomes more sophisticated [CER03]. We can take the examples of the Yahoo attack in 2000 or recently the massive attack on Estonia [EST07]. Many solutions of intrusion detection and filtering focus on the way to prevent attacks and have been defined but those techniques need many changes of the original structure of the Internet protocols. This relative efficiency of prevention is not enough to eliminate DDoS attacks. The placement of some edge IDS as filtering and detection devices is necessary. The principal issues recognized in defending wide area networks are that current software complexity and failures make the intrusion detection difficult in terms of performance and effectiveness; and for most of the cases, current defense propositions are based on a central server that becomes a target for attackers and a single point of failure. All those challenges have proved the importance of the introduction of cooperation among the defense security entities in general. This can be for an intrusion detection system in particular or for IP traceback nodes. Some systems have been developed but could not eliminate the central entity necessity. Indeed, the data are collected and sent to a central server that analyses the information. On the contrary, a fully distributed and hierarchical architecture approach presents many advantages.

The P2P model offers the promise to exploit all the resources of vast numbers of hosts. The distribution of data storage among several nodes, gives to this model two main advantages, in comparison with a centralized scheme; first, it reduces the possibility of storage overload at some points and second, it does not have a single point of failure. The use of a P2P algorithm can also be justified by its robustness, high scalability, and fast resource lookup. Many solutions have been proposed, which can be classified into structured and non-structured solutions regarding resource localization methods. Protocols developed on structured P2P networks have recently gained popularity for the implementation of large-scale distributed systems. Most of these approaches are based on hash tables which in turn can be centralized or distributed (named DHT for distributed hash table). We propose the use of a DHT that can efficiently route resource information on the victims and to share data on the claimed attacks among the peer [ZHU&al01]. This scheme is more flexible and can scale to a very large number of peers exchanging control messages without introducing additional overhead to the whole system. In the proposed architecture, we have the choice in the usage of many DHT solutions like Chord [STO&al01], Pastry [ROW&al01], Tapestry [ZHA&al04], Kademlia [MAY&al02],

etc. However, we recommend putting this application over the SPOP architecture described in chapter 5 where a more efficient DHT is provided: a context-aware DHT. Here the context regrouping the peers is the global security system proposed using the detection nodes. In fact, the objective is to solve the DDoS problem in a distributed manner, in which a scalable and efficient intrusion detection scheme cooperates with an accurate traceback scheme to progressively deploy filtering rules upstream until reaching the sources of attacks. As various deployment points bring different benefits, by combining their strengths and coordinating their actions, a distributed system can achieve a successful defense.

6.2 DDoS Attacks Overview

A denial of service attack is launched to make a computer or network unable to provide normal services [STE&al03]. Denials of service attacks continue to be a significant threat for today's Internet as they are growing in number and sophistication. Some recent studies developed by Moore et al. [MOO&al01] estimated the DoS activity by a backscatter method on packet traces and showed that more than 2000 DoS attacks are launched every week. The problem is that it becomes very easy for any Internet user to create disruptions using limited resources. Moreover, the attack damages are increased by the distributed computing techniques. Many existing systems are successful in one aspect of defense, but none of them offers a comprehensive solution. In such a context, there is a tremendous need for distributed and cooperative defense architecture in order to avoid the threat of DoS attacks.

When a DDoS defense system is deployed at the victim network, it is difficult, due to the aggregation, to identify attack packets at the ingress [14] of the targeted network although this deployment can facilitate the observation of the victim. This is why it is important to push the detection upstream to the ingress points of the service provider. This implicitly implies the distribution of the detection scheme among several locations, which raises the problem of how to coordinate the different detection systems. Some systems like DIDS [SNA&al99] or NSTAT [KEM&al97] have been proposed to work in a distributed environment. In these propositions, the audit of the data collected is done in several points of the network and the analysis is executed by a central location. With CSM [WHI&al96] and AAFID [ZAM&al00], the usage of distributed analysis agents is very relative. Current IDS do not offer a global solution that satisfies users' need in copying with the evolution of the attack types.

The deployment of DoS defense system at the attack source network cannot permit the collection of necessary information about the attack traffic and thus detection at this level will not be efficient. On the other hand, attack flows can be stopped before they enter the Internet core. And this is why response can be more effective at the attack source level. The mechanism for identifying the sources of attacks and for limiting the rate of malicious flows is commonly known as traceback mechanism.
Current DoS solutions are many, ranging from host-based solutions to network and infrastructure solutions. Our architecture is basically proposed for DoS detection and IP traceback solutions. For DoS detection, we have 2 main groups:

- The signature-based detection schemes that search for a known identity or signature for each attack event [SNO]. This category is not efficient against new types of attacks.
- Anomaly based detection schemes [GIL&al01] that detect anomalies caused by DDoS attacks. In this case a model must be established according to standard protocol normal system activities.

In general, the intrusion detection entities are deployed on hosts or routers and the agent is deployed at a single point or network-based where the agents cooperate either in a centralized [KEM&al97] or a decentralized [WHI&al96] manner. A decentralized approach is more scalable but needs more complex communication schemes to effectively share the information between the detection entities.

For IP traceback schemes, we have two main classes:

- Backtracking techniques [HAZ&al06] that work in a hop by hop manner to construct a summary of routed flow. In this class we have the proactive measures category where the flow is generated independently from the presence of the attacks and the reactive measures where the summary is generated on demand.
- Flow extension techniques bring additional information to flows during their travel. We have the in-band messaging (packet marking), that can be probabilistic [LIU&al03] or deterministic [BEL&al03], and the out-of-band messaging that sends the traceback data in separated packets.

Some proposed DoS solutions have a global scope. They start from the victim side where detection is most suitable and propagate attack alerts through intermediate networks in order to deploy filtering rules as near as possible to attack source networks. Mahajan et al. [MAHa&l02] proposes pushback (also implemented in [IOA&al02]) as a complete method to deal with DoS. In this proposition, DoS are treated as a congestion-control problem. A new functionality is added to each router to detect and preferentially drop packets that probably belong to an attack. Upstream routers are also notified to drop such packets (hence the term pushback) in order that router resources can be used to route legitimate traffic. This is an interesting approach but router vendors did not show interest in implementing this scheme. A draft was proposed at IETF which expired in 2002. Cotroeno et al. [COT&al01] use the same concept of pushback in defense "propagation". They propose ASSYST, a distributed system, in which network routers cooperate in order to react to DoS attacks in a flexible and dynamic fashion. DefCOM [MIR&al05] is a distributed collaborative framework to defend against flooding DDoS attack. As a global architecture, it combines the advantages of source-end, victim-end and core defenses and allows the existing heterogeneous defense systems to cooperate through an overlay. Nodes collaborate by exchanging messages, marking packets for high or low priority handling, and prioritizing marked traffic. However, it was not clearly described how to authenticate and establish economic cooperative relationship across different management domains.
We will present our architecture with its added-value characteristics compared to other propositions.

6.3 The Peer-to-Peer Collaborative Defense system

The objective is to propose a global architecture that permits an efficient Intrusion Detection System where participants can exchange information in a P2P method, providing services to a traceback application that strengthens the network security against DDoS attacks; or at least permits a fast and effective reaction to this kind of threats.
The proposed solution is designed to elaborate the defense against these large scale attacks by the correlation of the suspicious evidence provided and stored by the architecture entities from different geographical locations. Each participant gains a global view of the intrusion activity through this collaboration. To perform this objective, we took into consideration certain requirements in terms of performance and deployment. In fact, the processing, the bandwidth used and the storage must be minimized and conceptual security mechanism must be added to permit the access control for each entity in the architecture.

6.3.1 Problem Statement

A DDoS attack is usually characterized by a high traffic rate, an IP spoofing and several paths are taken to reach the victim. These elements are particular in a distributed attack. Our system proposes that the detection relies on the most frequently routed destination IP addresses during a short time period (Δ) on different network points. Each IDS deployed on the network and especially in the ingress of a network, will analyze the traffic that passes by the router which this IDS is added too. In fact, the same equipment can also do both jobs. Each detection system *Si* is responsible of a Sub-Netwok *SNi*. *Si* monitors the traffic to/from *SNi* to detect potential attacks on *SNi*. The whole detection system network *S* is formed by the set of detection system entities *Si*. We have *S={Si|i =1,2,3, …,n}*. Each of the *Si* is a node in a peer-to-peer system that forms our collaborative intrusion detection system (DS). The sharing of information between the entities, that we will present in the following sections, must be periodic and this shared evidence can take many forms. We consider that it is the destination IP address. For this, a DS checks the header of each IP packet captured and records this address with a counter that keeps the number of packets going to this destination in the last period of time (Δ). We will have for each *Si* the set <*Target IP @, Counter,* Δ >. The problem is that we must know how to decide if the traffic is issued by an attack. Many models like behavioral models or probabilistic models help to study the nature of different attacks. It is not beyond the scope of our work. We will consider that when the counter exceeds a certain limit, the IDS considered the traffic as malicious. The counter is a vague definition of what will be the decision factor. We will separate it in 3 principal variables that are the packet flow frequency *F*, the SYN packet ratio *R[T]* and the ACK packet ratio *R[A]*. Those values will really determine the malicious aspect of a flow.

6.3.2 The Distributed Hash Table algorithm

The development and widespread usage of peer-to-peer networks is mainly due to their efficient overlay routing and the location function, especially in global storage utilities and applications. We propose to apply an algorithm that defines a deterministic way to efficiently store and share the collected data between the nodes. These nodes are able to form a distributed and decentralized network with a dynamic adaptation without affecting the whole functioning of the network.

In order to realize the flexible and balanced collection of information among the nodes, we required an efficient approach that can scale to a large number of peers exchanging many control messages. The choice of a DHT algorithm satisfies these requirements. The semantics and the API [DAB&al03] specification are also developed for a public usage. Most of the solutions use the consistent hashing providing a balanced sharing among the peers due to the hash function. An advantage is that the joins and leave of nodes do not affect the whole geometry of the network. DHTs are used by many applicative routing solutions such as Chord [STO&al01], Can [RAT02], Pastry [ROW&al01], Tapestry [ZHA&al04], etc. These solutions present a relevant robustness since the global functioning of the network is totally independent of each application node.

A DHT network is logically divided using generally a circular ring space of indexes. In the space, we assigned for each node a key or identifier called a *nodeId* which informs about the logical position of the node. For example in Pastry it is a 128-bit identifier and the maximum number of possible nodes in a circle is 2^{128}. *nodeIds* are simply the hash value of the node IP address. This is why we talk about hash table indexes to identify each node of the DHT network. The assignment of indexes is done randomly but in a deterministic manner; the result of the hash function cannot be predicted and for each value to which the function is applied, a unique Id is generated. In the same logical space, resources are identified by an *objectId* and this *objectId* is the result of the same hash function applied to the name of the resource or a part of it. For applications like storage sharing or communication, a P2P network indexes some resources in each node by distributing the *objectIds* among the different nodes. So that a node will be responsible for a part of the whole resource indexes present in the network. This is why each DHT builds some rules to decide how to assign keys or identifiers and how to locate them via some tables kept by each node. In the case of Pastry, the *objectIds* are kept by the node with the closest *nodeId* numerically.

6.3.3 Description of the architecture

Our principal contribution in this chapter is the proposition of the architecture that serves as a model to implement a security system against DDoS attacks. This model targets detection systems with the possibility to place some applications at the very top level that can use the collected data of the detection system. The innovation in this case is the addition of a new level between the application one and the detection system which is the use of the DHT. This new level permits to index the information and to distribute it among the participants instead of implementing a central collection entity.
The Figure 6.1 describes each level of the architecture. We present it with an abstraction degree that permits us to put both independent and specific functions on each level. In fact, a node can contain one or more levels. We will detail this later by giving examples.
The first level is the closest to the physical network. We called it the *Network Level*. In this level, equipment belongs to the underlay network. To be more specific, an entity in this level can for example be an IP router with the basic routing and addressing functionalities.

The second level is the *Security Level*. In this level, we can classify our detection system entities. The implementation of all solutions in terms of detection system can be done here. This level's functions are the ones of detection. When the analyzed traffic in this level is detected as an attack, an alert is triggered and a primitive is sent to the upper level. This level will react to the alert accordingly. We can note that a detection system can be integrated to an equipment and the concerned entity will be represented by the two first levels of the architecture.

The third level of the model is the *P2P Level* which includes the Context-Aware DHT proposed in our architecture to index and to distribute the information among the nodes depending on the group the peers are members of. In our case, the context share is the fact that these peers are associated to Detection Systems that retrieve information and send alerts managed by the upper application. This level receives the information collected concerning the analyzed traffic from the *Security Level*. When an alert is sent by the lower level, which means that an attack is detected, the P2P level treats the received data and indexes the information on the specific DHT node (identified by a *nodeId*) depending on the *objectId* calculated. Transport can be optimized following the SPOP transport optimization properties presented in chapter 5 and we also have the possibility to vary the transport protocol parameters for better QoS and decreasing the overhead and/or the peering traffic in the application is distributed in many ASs.

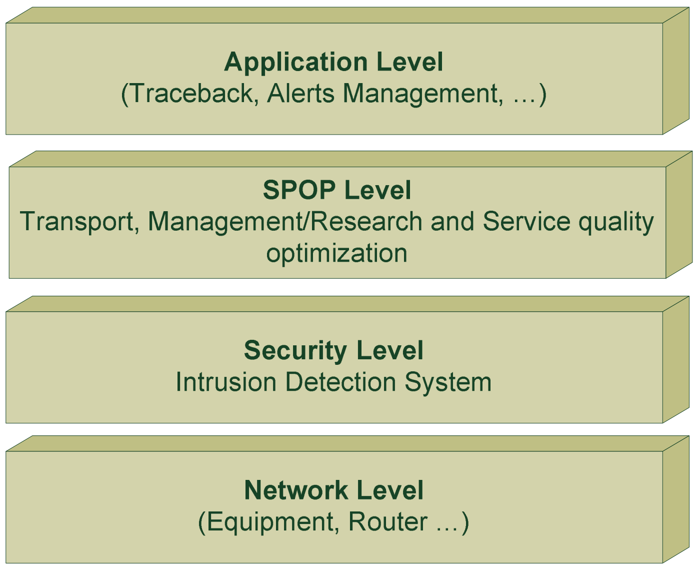

Application Level
(Traceback, Alerts Management, …)
SPOP Level
Transport, Management/Research and Service quality optimization
Security Level
Intrusion Detection System
Network Level
(Equipment, Router …)

Figure 6.1: The architecture levels

The last level is the *Application Level*. This last level is general in the description for our architecture. It can implement all possible management systems that use the indexed information on attacks by the DHT level to react. We propose in our case to add a traceback application solution to integrate more complete defense architecture than only a detection system. This traceback module is a part of the future works since the three first levels of the architecture were implemented as we will see in the next sections.

By removing any central analysis entity in the architecture, we propose a fully distributed solution. However, in choosing this method we must be sure that the collected data are correlated to ensure a better detection of DDoS attacks and also a reaction to them. Indeed, we propose that some applications retrieve the traffic information to analyze it deciding what to do.

Relying on the proposed model, we presented in Figure 6.2 a network that includes some Sub-Networks and some attackers in the same level. Each of these entities is linked to the Internet by a Network Equipment that can include an IDS *Si* which is responsible for the Sub-Network *SNi*. *Si* and the network equipment associated to it can be in the same hardware entity but we represented both of them with different schemes to separate every module depending on its functions. In particular, some of the network equipments are not associated to any detection system. In association with our model, the first layer in the example represents the 2 first levels that are the *Network Level* and the *Security Level*.

In the second layer, we illustrated the DHT ring that manages the *Underlay Network* by collecting the information on the attacks and indexing them in a distributed way. This is *Management Network*. The ring contains the DHT nodes modules and represents them in a logical manner. In DHT algorithms the representation is not always a ring or circular but in most of the case, this scheme is used in a simple way to show the logical overlay network of the DHT modules.

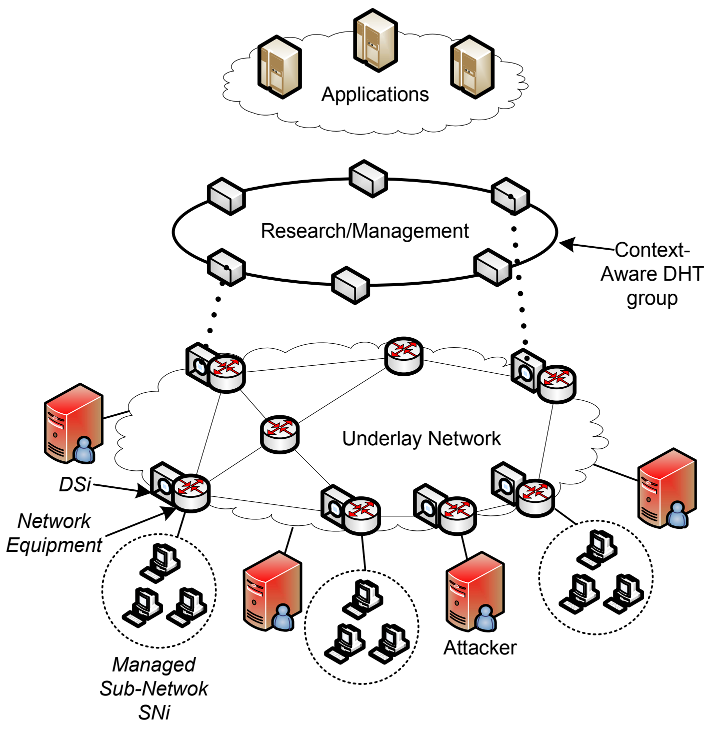


Figure 6.2: The architecture entities

The applications are in a unique layer in the top according to the architecture levels presented before. The entities in this layer are specific to every application that can use the information collected by the nodes in the *Underlay Network* and indexed by the DHT algorithm. It is a reaction to the detection of DDoS attacks. The action can be some maintenance and management or in some cases it can be a more detailed study of the information by applying a mathematical or a compartmental model on the whole collected traffic. In fact, at the top level, we have an aggregate flow of traffic to analyze but with some information given by the detection system entities and distributed by the DHT nodes.

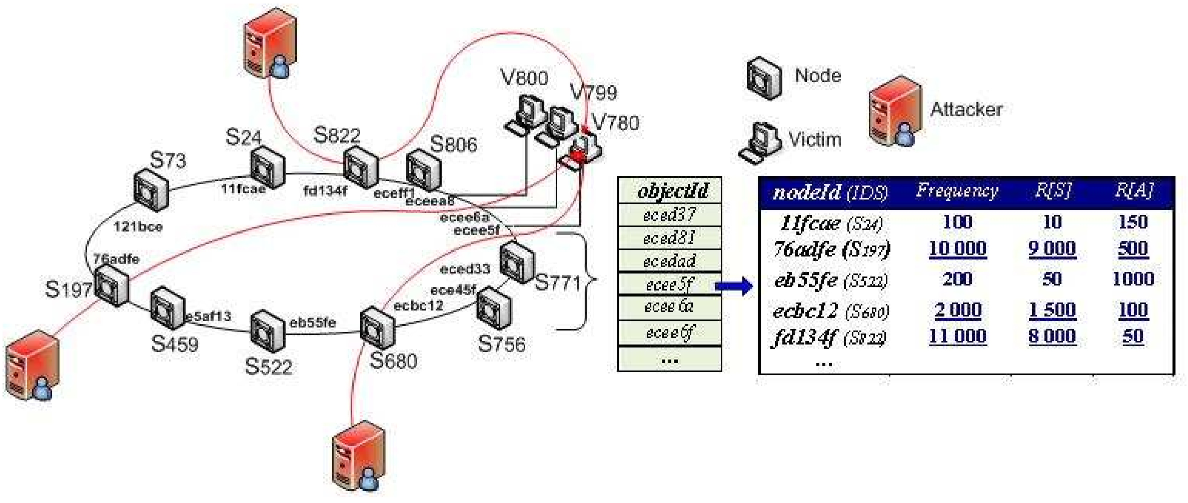
6.3.3.1 Principles and functioning

To illustrate our proposition, we chose the Pastry protocol to place the resources in a decentralized way on all the nodes. The resources are represented by data that the detection systems share and look for when they cooperate to exchange attacks information. As mentioned before, every IDS shares information about suspicious activities detected on its Sub-Network. We can imagine that all the information on each victim *V* is a database and this database is divided into small data tables distributed among all the IDS nodes. Pastry protocol acts in a manner to choose how to distribute those tables and how to refund them. As we presented before, Pastry uses a hash function *H* to calculate the *Id* of the resources and the *Id* of the nodes which contains those resources. We propose that the victims (*Vk*) addresses are used to calculate the *objectId* as for the IDS *Id* (*nodeId*); so we have the hash function applied to the victims and IDS addresses: *objectId= H(@Vk)* and *nodeId =H(@Sk)*. In the rest of the work, *Vk* and *Sk* are given randomly with an integer value for an easier comprehension of the algorithm but the indexes respect the 2^b base which is *16* because $b = 4$. The results of those formulas are 128 bits keys distributed in the same space logically placed in a ring. In this ring like in the original Pastry implementation, we have the victims' indexes placed between the detection systems' indexes and each *objectId* is located in the IDS with an index numerically closest to the concerned victim index. In fact, the placement in the ring of the victim indexes is completely independent of the physical position of the nodes. In our solution, the resources represented by the base *16* words and associated to an IDS *Si* are neither physically, nor geographically near the IDS that keeps the resources on those victims. The important aspect of the proposition is that the shared resource is the attack information on victims in a way that each Pastry node will store information received by other Pastry nodes on victims the current node is responsible for.

As a contribution to Pastry protocol, we propose that instead of using a hash function we choose a HMAC function [HMC97]. The good thing in HMAC is that the Hash function *H* can be used without any modifications. Implemented to the proposed solution, we can take the example of a resource *R* comparing the hash function to the HMAC one. In fact, in spite of having *H(R)=I*, we will have *H(K, R) = I'*. Using in our proposition the HMAC function adds to a new security level. This type of implementation permits access control between the detection systems and this option can be used in all possible applications that use a DHT algorithm. Finally, we have the indexes calculated with the formulas *objectId = H(K, @Vk)* and *nodeId = H(K, @Sk)*.

An example of the network is illustrated in the Figure 6.3, where some nodes are organized logically in a Pastry ring. For a better understanding, we represented nodes that belong to the *P2P level* of the model and that also integrate the *Security* and *Network* levels. In fact, a traffic coming from an attacker will be detected by the node, implicitly its IDS module, and its DHT module will do the indexing job. Each node is connected to a Sub-Network physically but logically it is not absolutely keeping information on the nodes (possible victims) physically connected to it. In fact, taking the example of the IDS *S771*, we see that the data table is for the victims for which the *objectIds* are numerically the closest to *ecee5f*. Those victims are not connected to *S771*. The logical distribution of *Ids* is calculated with the HMAC function and the results of this function depend on the IP address used as entry and the key *K*, independently of the geographic and physical links.

Three of the nodes noticed a suspect traffic flow with *V780* as destination. These nodes are *S197*, *S680* and *S822*. The victim's information is kept by the IDS *S771* determined by the DHT HMAC function. In the reference table we have the victims' indexes that are managed by this current node and each *objectId* points to another table. The primitive that permits to send a publish message on another node is the *put* function [DAB&al03] that needs a *lookup* function [DAB&al03] to find the responsible node for a specific victim *objectId*.



objectId
eced37
eced81
ecedad
ecee5f
ecee6a
ecee6f
...

nodeId (IDS)	Frequency	R[S]	R[A]
11fcae (S24)	100	10	150
76adfe (S197)	10 000	9 000	500
eb55fe (S522)	200	50	1000
ecbc12 (S680)	2 000	1 500	100
fd134f (S822)	11 000	8 000	50
...			

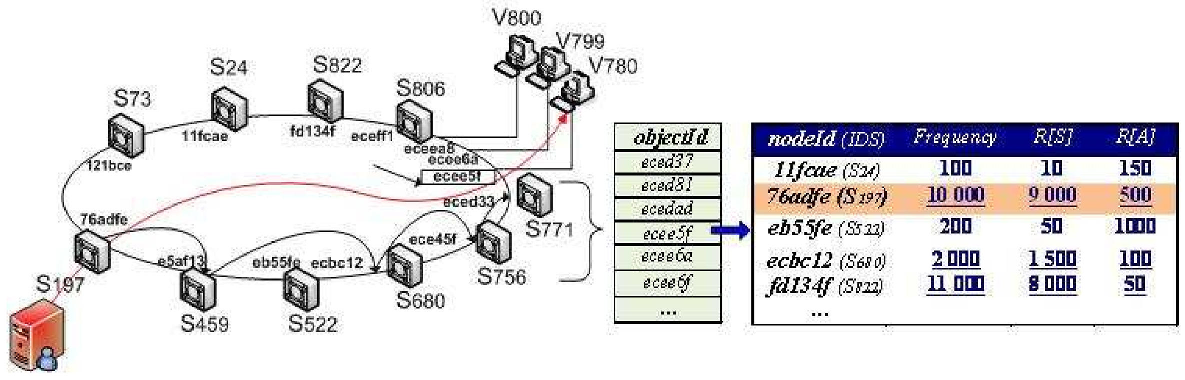
Figure 6.3: Nodes distribution and indexing with Pastry protocol

In our example, the concerned victim has an *Id* value equal to *ecee5f*. The information table saves the different flows where *V780* is the destination with some information concerning each one. We represented in the table the *nodeId* of the IDS which are the result of the HMAC function applied to each IDS address. For each IDS, we have the packet flow frequency, the SYN packets ratio (*R[S]*) and the ACK packets ratio (*R[A]*). Three of the entries are considered as possible attacks: the detection systems that saw a suspect traffic are *S197, S680* and *S822*. The other entries that are not underlined are not considered as suspect peers.

The table that stores for each node the traffic detected and sent by the other nodes in the ring permits to correlate the entire information on a specific victim. In fact, we can be sure that for any victim of the network, one node will store the traffic data for this node. Of course this information can be replicated on other nodes by the *replication* function [DAB&al03] of the DHT.

We represented in the Figure 6.4 the attacks without showing how all the IDS, from where the attacks passed, inform the IDS *S771* and how they are looking for it with the Pastry protocol.

In the Figure 3.4 the *lookup* primitive [DAB&al03] is detailed for a resource's responsible ID. One of the 3 attacks that pass by *S197* makes it looking up for the victim's responsible node. *V780* with the index *ecee5f* is managed by *S771* that has the index *eced33*. As we previously mentioned, the first table stores the victims' *objectIds* managed by *S771* and for each of this table's entries, a second table keeps the information on presumed attacks. In this last table, we save the *nodeIds* of the detection systems that detected the traffic and the information on each of these traffic flows. We can note that if any node of the ring needs to retrieve information kept by another node, the *get* function [] is used after a lookup of this particular node.



objectId
eced37
eced81
ecedad
ecee5f
ecee6a
ecee6f
...

nodeId (IDS)	Frequency	R[S]	R[A]
11fcae (S24)	100	10	150
76adfe (S197)	10 000	9 000	500
eb55fe (S522)	200	50	1000
ecbc12 (S680)	2 000	1 500	100
fd134f (S822)	11 000	8 000	50
...			

Figure 6.4: Lookup for V780's IDS from IDS S197

6.3.3.2 Simulation and results

To test our solution, we focused on two levels of our architecture which are the *Security level* and the P2P *level*. For the *Security Level,* we implemented the open source and non commercial detection system Snort [SNO]. Snort can perform detection and prevention by applying a packet logging and real-time traffic analysis on IP networks. Snort uses the *libcap* library to capture the network packets. After decoding the packets to have the protocols information, a preprocessor (SPAD) classifies and transforms the data to facilitate the treatment by Snort. After this, the detection engine applies the rules (signatures for attacks). Then the output modules generated an alert and logged the information.

Many solutions can be used to implement a DHT node locally and all those solutions propose a global scheme for a large-scale testing of the proposed applications. We chose *BambooDHT* that proposes a light-modified version of Pastry protocol that we used to illustrate the principles of our architecture. Some nodes in the PlanetLab [LAB] platform form the *OpenDHT* [RHE&al05] network that is a large-scale implementation of *BambooDHT* nodes. This platform can be used in the future work that is currently in progress in our laboratory for a larger implementation study.

The first step of our evaluation was to take a group of nodes that run our version of *bambooDHT* with the use of its API developed in Java to route the message between the nodes. On every node, a system socket is developed to permit the link between the Snort module and the DHT one. This brings the indexing and distribution function by the *P2P level* every time the *Security level* generates an alert primitive because it detects a suspect traffic. To simulate an attack, we generate a high number of ICMP packets sent to a specific victim. In this case, we only took the case of one type of attack which is *ICMP Flooding*. In the developed program, we fixed a threshold that must be reached to decide that the traffic is probably malicious.

We illustrated in Figure 6.5 the evolution of the ICMP packets ratio and more precisely their bandwidth consumption in a victim node of our evaluation system during a created *ICMP Flooding* attack. We saw that during the attack, the increase of ICMP packets reach 75% of the bandwidth many times.

To evaluate our implementation, we tested the decentralized solution and compared it with a centralized solution where a central node is receiving all of the data by the nodes that integrated the detection systems. To study the comparison, we varied the size of the records that every node generates and wants to store. In the centralized solution, all the storage is done on the central node. We showed on the Figure 3.6 the delay time in each approach for a group of 10 nodes and a group of 30 nodes.

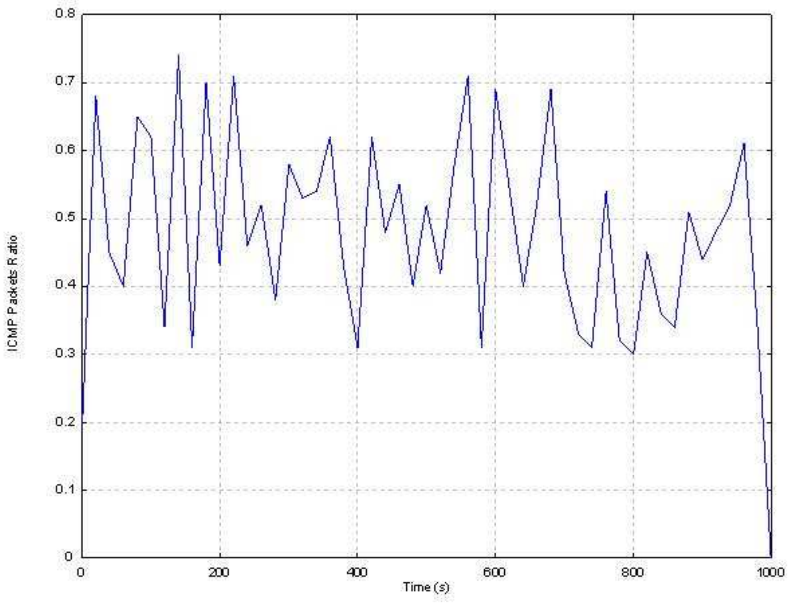


Figure 6.5: Bandwidth Consumption of ICMP Packets during an *ICMP Flooding* attack

This delay time is the time taken between the sending of a publish message by a node to the destination node (central or distributed according to each approach). We saw that from a certain number of records (100) the distributed approach out-performs the centralized one. Passing from 10 nodes to 30 nodes reveals the better adaptation to the variation of the nodes number for the decentralized solution. The centralized graph presents a particular increase when the records number is important. We launched the decentralized system with 10 nodes during less than one hour (2390 seconds exactly), then we collected the packets arrival and departure to show the homogeneity of the P2P system associated to the IDS solution. In fact, we can see in the Figure 6.7 that the variation is between 650 and 930 packets. The packets number includes the signaling messages, the requests and responses only for the *P2P Level* during the attacks detection. It does not include the packets flow from the outside entities. An important result is also that, for each node, the sent packets number is always very close to the received one, due to the stable aspect of the system.

6.3.3.3 Discussion

The modular model proposed here presents many advantages at each level we specified. The *Security Level* can implement any IDS module that can provide data information and alerts on possible attacks. For the *P2P Level*, it can be based on any DHT algorithm that permits the efficient distribution and exchange of data among the different nodes of the architecture. We also know that those algorithms are adequate to a distributed cooperative system that must be scalable without affecting the overall system.

Nevertheless, we are aware that the results previously presented cannot be the only arguments to validate a solution comparing another. We must study the load-balancing and the peer number variation in a large-scale area with more than few nodes. In fact, a vast implementation is needed with more scenarios in order to design a mathematic or compartmental that will really validate the proposition. An issue is that fixing a threshold is not enough to decide if a flow is an attack, because each of the distributed flows can be under this limit and present a danger to the victim when they are aggregated. These steps are, as we said earlier, currently in development.

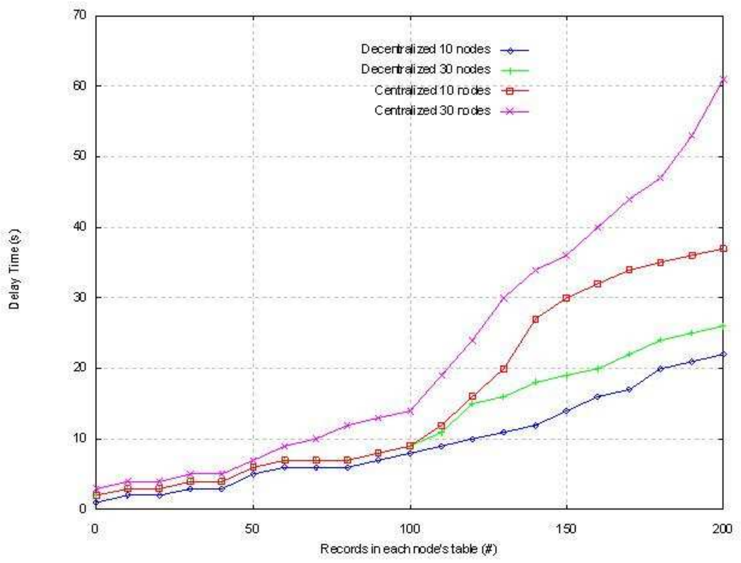


Figure 6.6: Comparison of the architecture with a centralized approach

Concerning the *Application Level*, the principal objective is to provide a system that can analyze the information distributed in the lower levels and especially in the *P2P Level*. It is important to elaborate a complete and secure application that can react to the detected attacks, which information is stored in the lower levels detailed previously. For the traceback mechanism example, some nodes could analyze the information on possible attacks and perform a traceability of every flow by marking packets to detect the attack sources. For keeping the same model constraint, we recommend a distributed traceback mechanism that will be compared to current traceback solutions and studied to be more efficient in terms of scalability and precision. The information correlated in the Context-Aware DHT algorithm can serve to possibly mark abnormal detected packets and to limit the rate of malicious flows by identifying the sources of attacks. Comparing to [SNA&al99] or [KEM&al97] we are proposing a distributed treatment of the attack traffic information in the application layer where any node can correlate its information with any other one. This could avoid a central analysis in addition to the distributed audit that we already performed. Like [BEL&al03] we must elaborate an out-of-band manner to send the traceback data in separate packets between the *Security Level* IDS first, then between the *Application Level* entities. We think about developing an extension to our *BambooDHT* program that adds the *Application Level* module functions for the traceback mechanism. A more detailed study is in progress.

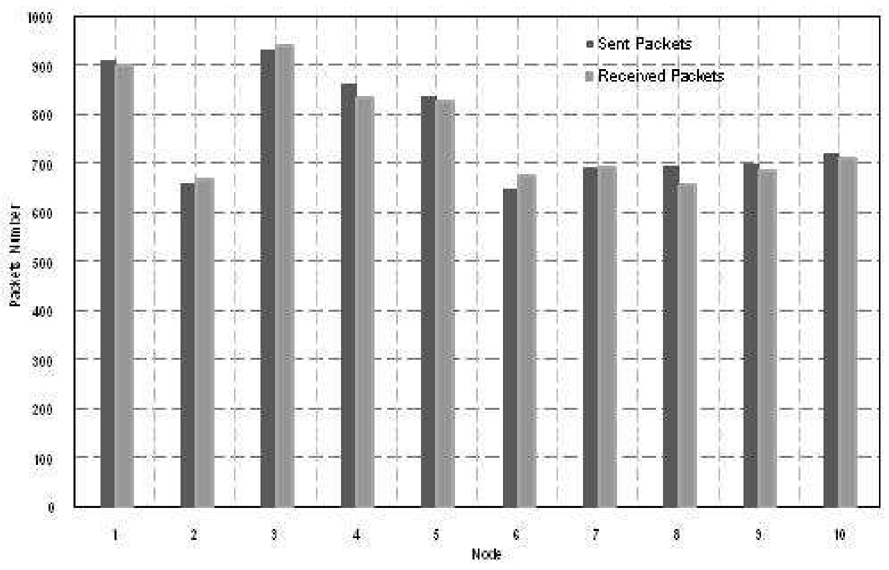


Figure 6.7: Sent and received packets number for 10 nodes during 2390 s.

6.4 Conclusion

In this chapter, we proposed a modular architecture for a collaborative defense against DDoS attacks relying on the performance and scalability of DHT algorithms. This model was designed with the aim of proposing the integration of an intrusion detection system that can bring an attacks' information collect service to some applications like traceback. An efficient DHT indexing protocol is used to interface these two levels of our architecture. Our proposition overcomes the challenges of the collaboration. The load-balancing is ensured by the consistent hashing of the DHT and every node keeps information on specific victims. The replication primitive permits the copy of this information in other nodes for some security reasons, even if our distributed architecture removes the single point of failure major problem. Finally, the integration of an HMAC function adds a robust access control with the sharing of a key in association with the original hash function. However, the scope of this chapter is the description of the architecture and the way that the lower levels offer services to the application level. The management and the traceback application that can retrieve and analyze the data sent by the lower levels to react to the possible attacks must be more specifically specified, and this is a major issue for future work.

Chapter 7

General conclusion and perspectives

This last chapter concludes this thesis by summarizing the objectives and the contributions presented. It also gives the different perspectives that could be elaborated to provide some more validation results.

7.1 Conclusion

The main objective of this thesis was to focus on studying the design of an architecture for P2P applications that is controlled and managed by a Service Provider. Managing P2P traffic and reducing completion download time are a real challenge for ISPs. While customers are interested in having the best service with the maximal performance, providers must keep their services working efficiently with the best QoS without experiencing consequent over costs.

We divided the P2P model in three main components that are the transport level, the routing/lookup level and the service level. We took into account each of these components and especially the way to propose real improvements to them.

First of all we presented a general state of art of overlay networks and Peer-to-Peer networks. We focused on BitTorrent which is the protocol chosen for the transport in our architecture. A complete related work section shows the different performance studies and the contributions in locality-aware techniques and erasure codes mechanisms in this protocol.

The first contribution is a new peer selection policy with a specific Autonomous System mapping for each peer in BitTorrent. Large scale simulations validated some interesting results on how to decrease download completion time and peering traffic between domains. This partitioning is based on a special entity called the *hTracker*.

The second contribution was to perform a deep study of the integration of erasure codes techniques in BitTorrent. This evaluation work permits us to know in which special cases it can be interesting to use mechanism. The simulation based results validated with a statistical test show that FEC can speed up data transfer when the peers experience a lack of resources. However these peers must be fast enough to take profit from this efficient technique provided by BitTorrent *Seeds*.

The third contribution is the proper architecture implementing a generalization of the partitioning proposal. In fact we extended it in a Context-Aware algorilthm that has the characteristic to be a DHT optimizing lookup and routing. The architecture implements a FEC service for peers that can be offered a faster service. Finally, we propose to use *hTracker* entity at the P2P Service component level to prepare the environment for some QoS constraints if some peers are demanding. This Service level proposes also the changing of some basic parameters while keeping interoperability with existing architectures.

The final step of our work was to implement an application that validates the Service Provider Oriented P2P (SPOP) architecture. The application is a global security system that indexes and manages intrusion detection nodes. These entities send information on detected DDoS attacks in a distributed and scalable manner.

We are aware that the global Service Provider oriented architecture proposed requires to studying management and control policies if an ISP decide to implement it. Even if this perform a real optimization level is referred to each application requirements, the solution introduces some overhead and more treatment. Thus it is important to note that this solution must be evaluated in terms of implementation and design costs. This study, that can be a perspective work, must be taken into account by an ISP that decides to choose SPOP. More technical perspectives are presented in the next section.

7.2 Perspectives

The SPOP architecture was validated by large scale simulations. The main perspective is to implement in the real world the different components of the proposed solution. P2P dynamicity and scalability can be studied using some real nodes in experiment platforms like the PlanetLab one.

The application implemented in SPOP to validate this architecture was also tested in our labs and requires a real world large scale implementation in PlanetLab nodes. This application requires also a service level that has not been defined in yet. In fact we focused on how the alerts and information on attacks are stored in a distributed manner but no service has been already developed to use this data in order to apply a traceback system.

In erasure codes research domain Network Coding aims to be a good alternative to FEC mechanism. However it requires that each peer implicates itself to the error correction. Even if our architecture is Service Provider oriented, it can be interesting to study an alternative that allow peers to use Network Coding and to dress a complete comparison of both solutions while it is certain that FEC is still a good value.

One interesting issue is to adapt SPOP architecture to IPTV P2P applications. The service level adaptation proposed is necessary but not enough while it is important to formalize the video sliding window and the pieces transfer in the BitTorrent transport protocol.

List of Publications

[PBL1] G. Doyen, D. Gaïti, R. Khatoun, L. Merghem-Boulahia, R. Saad and A. Serhouchni, "A cooperative agent approach based on a peer-to-peer model for DDoS attacks detection and reaction", in *Proceedings of NAEC 2008*, September 25 - 28, 2008 in Riva Del Garda, Italy.

[PBL2] R. Saad, F. Naït-Abdesselam and A. Serhrouchni, "A Peer-To-Peer Collaborative Architecture to Defend against DDoS Attacks", in *Proceedings of LCN 2008,* October 13 - 17, 2008 in Montreal, Canada.

[PBL3] R. Saad, F. Naït-Abdesselam and A. Serhrouchni, "Une Architecture Peer-to-Peer de Défense Contre les Attaques DDoS", in *Proceedings of SAR-SSI 2008*, October 13 - 17, 2008 in Loctudy, France.

[PBL4] G. Doyen, D. Gaïti, R. Khatoun, R. Saad and A. Serhrouchni, "Decentralized Alerts Correlation Approach for DDoS Intrusion Detection", in *Proceedings of NTMS 2008,* November 5 - 7, 2008 in Tangier, Morroco.

[PBL5] R. Saad, A. Serhrouchni and K. Chen, "*hTracker*: Towards a Service Provider oriented Peer to Peer Architecture", in *Proceedings of NOTERE 2010*, May 31st - June 2nd, 2010 in Tozeur, Tunisia.

[PBL6] R. Saad, A. Serhrouchni and K. Chen, "SPOP: A Service Provider Oriented Peer-to-Peer architecture", to appear in *SoftCOM 2010*, September 23 - 25, Split – Bol, Croatia.

[PBL7] R. Saad, A. Serhrouchni, Y. Begriche and K. Chen, "Evaluating Forward Error Correction in BitTorrent Protocol", to appear in *Workshop on Wireless & Internet Services (WISe)* in *Proceedings of LCN 2010*, October 11 - 14, 2010 in Denver, USA.

Bibliography

[ABE&al02] K. Aberer, M. Punceva, M. Hauswirth, R. Schmidt, "Improving Data Access in P2P Systems", *IEEE Internet Computing 6,1*, pp. 58–67, Jan./Feb. 2002.

[AGG&al07] V. Aggarwal , A. Feldmann , C. Scheideler, "Can ISPS and P2P users cooperate for improved performance?", ACM SIGCOMM Computer Communication Review, v.37 n.3, July 2007.

[AKA] Akamai, http://www.akamai.com.

[AND&al01] D. Andersen G., H. Balakrishnan, F. Kaashoek, et al., "Resilient Overlay Networks", in *ACM. 18th Symposium on Operating Systems Principles (SOSP),* 21-24 october 2001, Banff, Canada. New-York : ACM Press, 2001, pp.131-145.

[AND&al05] N. Andrade, M. Mowbray, A. Lima, G. Wagner, M. Ripeanu, "Influences on cooperation in BitTorrent communities", in *proceeding of the 2005 ACM SIGCOMM workshop on Economics of peer-to-peer systems*, August 22-22, 2005, Philadelphia, Pennsylvania, USA.

[ART&al05] D. Arthur and R. Panigraphy, "Analyzing the efficiency of BitTorrent and related peer-to-peer networks," in *Proceedings Seventeenth annual ACMSIAM symposium on Discrete algorithms*, January 2005, pp. 961–969.

[ASA&al09] H. Asai, H. Esaki, "AURORA: Autonomous System relationships-aware overlay routing architecture in P2P CDNs", Asia Future Internet summer school, Jeju, Korea, 2009.

[BEL&al03] A. Belenky, N. Ansarin, "Tracing multiple attackers with deterministic packet marking (DPM)", in *Proceedings of IEEE PacRim vol. 1, pp. 49-52,* August 2003.

[BIC&al07] D. Bickson, R. Borer: The BitCod Client: A BitTorrent Clone using Network Coding. Peer-to-Peer Computing 2007: 231-232.

[BIN&al06] R. Bindal, P. Cao, W. Chan, J. Medved, G. Suwala, T. Bates, and A. Zhang, "Improving Traffic Locality in BitTorrent via Biased Neighbor Selection", in *Proceedings IEEE Int'l Conf. Distributed Computing Systems (ICDCS '06),* pp. 1-9, July, 2006.

[BIT] BitTorrent, http://www.bittorrent.com.

[BLO&al99] S. Block, K. Chen, P. Godlewski, and A. Serhrouchni, "Some Design Issues of SRMTP, a Scalable Reliable Multicast Transport Protocol", In *Helmut Leopold and Narciso Garcia, Proceedings ECMAST'99*, Madrid, Spain, May 1999. Springer Verlag.

[BLO70] Bloom H. (1970). Space/Time Trade-offs in Hash Coding with Allowable Errors. Commun. ACM 13, 7 (Jul. 1970), 422-426.

[BTO] The Official Protocol Specification, http://www.bittorrent.org/beps/bep_0003.html.

[BTS] BitTorrent Simulator, http://www.ug.bcc.bilkent.edu.tr/~e_yildirim/.

[BTW] The Unofficial BitTorrent Protocol Specification v1.0, http://wiki.theory.org/BitTorrentSpecification

[BYE&al98] J. W. Byers, M. Luby, M. Mitzenmacher, A. Rege, "A Digital Fountain Approach to Reliable Distribution of Bulk Data", *ACM SIGCOMM' 98*, September 2–4, 1998.

[BYE&al99] J. W. Byers, M. Luby, M. Mitzenmacher, "Accessing Multiple Mirror Sites in Parallel: Using Tornado Codes to Speed Up downloads", IN*FOCOM 99*, 1999.

[BYE&al02] J. W. Byers, J. Considine, M. Mitzenmacher, and S. Rost, "Informed content delivery across adaptive overlay networks," in *SIGCOMM, Pittsburgh, PA, ACM,* 2002.

[CAC05] CacheLogic. CacheLogic – advanced solutions for peer-to-peer networks, 2005.

[CAI] CAIDA, The Cooperative Association for Internet Data Analysis, http://as-rank.caida.org/.

[CER03] CERT Coordination Center, "Module 2 - Internet Security Overview", 2003.

[CHE&al09] Xinuo Chen, Stephen A. Jarvis, "Analysing BitTorrent's *Seed*ing Strategies," cse, vol. 2, pp.140-149, 2009 International Conference on Computational Science and Engineering, 2009.

[CHF&al08] D. R. Choffnes and F. E. Bustamante, "Taming the Torrent: A practical approach to reducing cross-ISP traffic in P2P systems", in *Proceed*ings *of ACM SIGCOMM 2008*, August 2008.

[CHO&al08] A.L.H. Chow, L. Golubchik, V.Misra, "Improving BitTorrent: a Simple Approach", in *the 7th international workshop on Peer-to-Peer Systems (IPTPS'08)*, Tampa Bay, Florida, February 2008.

[CIS] Cisco, http://www.cisco.com

[CLA&al99] K. Claffy and S. McCreary, "Internet measurement and data analysis: passive and active measurement", *Univ. of California, CAIDA*, 1999.

[CLA&al00] I. Clark , O. Sandberg, B. Willey, et al, "Freenet: A distributed anonymous Information storage and retrieval system", in FEDERRATH H., editor. *Design*ing *Privacy Enhanc*ing *Technologies: International workshop on Design Issues* in *Anonymity and Unobservability*, Berkeley, California, USA, 25-26 juillet 2000.

[COH&al02] E. Cohen and S. Shenker "Replication Strategies in Unstructured Peer-to-peer Networks", *ACM SIGCOMM 2002*, August 2002.

[COH03] B. Cohen, "Incentives build robustness in BitTorrent," in *Proceedings* First Workshop on Economics of Peer-to-Peer Systems, Berkeley, USA, June 2003.

[COT&al01] L. Peluso, D. Cotroneo, S. P. Romano, G. Ventre, "ASSYST: an Active Security System against DoS attacks", *Technical Report. Dept.of Computer Sciences*, University of Napoli, Italy, April 2001.

[CRE&al02] A. Crespo and H. Garcia-Molina, "Semantic Overlay Networks for P2P Systems", *Technical report, Computer Science Department, Stanford University*, 2002.

[CUE&al02] F. Cuenca-Acuna, C. Peery, P. Martin et D. Thu Nguyen, "PlanetP : Using Gossiping to Build Content Addressable Peer-to-Peer Information Sharing Communities", in *12th IEEE* In*ternational Symposium on High Performance Distributed Comput*ing *(HPDC-12)*, 2002.

[CUI&al09] B. Liu , Y. Cui , Y. Lu , Y. Xue, "Locality-awareness in BitTorrent-like P2P applications", *IEEE Transactions on Multimedia*, v.11 n.3, p.361-371, April 2009.

[CWU&al07] C.-J. Wu, C.-Y. Li, and J.-M. Ho, "Improving the download time of BitTorrent-like systems," in *IEEE* In*ternational Conference on Communications 2007 (ICC 2007)*, Glasgow, Scotland, June 2007.

[DAB&al03] F. Dabek, B. Zhao, P. Druschel, and I. Stoica. Towards a common API for structured peer-to-peer overlays, in *2nd International workshop on Peer-to-Peer Systems (IPTPS'02)*, February 2003.

[DAL&al07] C. Dale and J. Liu, "A Measurement Study of Piece Population in BitTorrent," in *IEEE GLOBECOM'07*.

[DAL&al08] C. Dale, J. Liu, J. Peters, and B. Li, "Evolution and enhancement of BitTorrent network topologies," *Quality of Service, 2008. IWQoS 2008. 16th International workshop on*, pp. 1–10, June 2008.

[DIM&al07] X. Dimitropoulos, D. Krioukov, M. Fomenkov, B. Huffaker, Y. Hyun, K. Claffy, and G. Riley, "AS Relationships: Inference and Validation", in *ACM SIGCOMM Computer Communication Review (CCR)*, vol.37, no.1, pp. 29-40, January 2007.

[DOU&al02] J.R. Douceur, "The Sybil Attack", in *1st International workshop on Peer-to-Peer Systems (IPTPS '02)*, Cambridge, MA, 2002.

[EGR&al07] K. Eger, T. Hoßfeld, A. Binzenhöfer, G. Kunzmann: "Efficient Simulation of Large-Scale P2P Networks: Packet-level vs. Flow-level Simulations", *2nd Workshop on the Use of P2P, GRID and Agents for the Development of Content Networks (UPGRADE-*

CN'07), in *conjunction with 16th IEEE HPDC (High Performance Distributed Computing),* June 25-29, 2007, Monterey Bay California, USA.

[ERM&al05] D. Erman, D. Ilie, and A. Popescu, "BitTorrent session characteristics and models," presented at *the 3rd Int. Working Conf. Performance Modelling and Evaluation of Heterogeneous Networks (HET-NETs'05)*, Ilkley, U.K., 2005.

[EST07] Cyber Assaults on Estonia Typify a New Battle Tactic, by Peter Finn, Washington Post Foreign Service, May 19, 2007.

[FAN&al06] B. Fan, D.-M. Chiu, and J. Lui, "Stochastic analysis and file availability enhancement for BT-like file sharing systems," *Quality of Service, 2006.IWQoS 2006. 14th IEEE International workshop on*, pp. 30–39, June 2006.

[FAY&al08] M. Fayçal and A. Serhrouchni, "An Efficient Management Technique for Peer-to-Peer Networks", in In*ternational Conference on Software Telecommunications and Computer Networks*, SoftCom2008, Split-Dubrovnik, Croatia, September 2008.

[FEL&al04] P. A. Felber and E. W. Biersack, "Self-scaling networks for content distribution," in In*t. Workshop on Self-* Properties* in *Complex* In*formation Systems*, Bertinoro, Italy, May-June 2004.

[FER&al98] P. Ferguson and D. Senie, "Network ingress filtering: defeating denial of service attacks which employ IP source address spoofing". In *IETF, RFC 2267*, Janurary 1998.

[FIN&al04] R. Fink and R. Hinden, "RFC 3701: 6bone (IPv6 Testing Address Allocation) Phaseout", March 2004.

[GAR&al06] P. Garbacki, A. Iosup, D. Epema, M. van Steen: 2Fast: Collaborative downloads in P2P networks. In *Proceedings of P2P 2006*, 2006.

[GAR] P. Garcia, C. Pairot, R. Mondejar, et al. "PlanetSim: A New Overlay network Simulation Framework", in *4th Software Engineering and Middleware (SEM'04), 20-21 septembre 2004*, Linz, Autriche. New York : Springer-Verlag, number 3437, pp. 123-136.

[GIA03] Y. Chawathe, S. Ratnasamy, L. Breslau, N. Lanham, S. Shenker, "Making Gnutella-like P2P systems scalable", in *Proceedings of SIGCOMM, ACM*, 2003.

[GIL&al01] T.M. Gil and M. Poleto, "MULTOPS: a data-structure for bandwidth attack detection" In *Proceedings of 10th Usenix Security Symposium, Washington, DC, pp. 23–38,* August 2001.

[GKA&al05] C. Gkantsidis and P. Rodriguez, "Network coding for large scale content distribution", in *proceedings of* IN*FOCOM 2005.*

[GKA&al06] C. Gkantsidis, J. Miller, and P. Rodriguez, "Anatomy of a P2P Content Distribution System with Network Coding", in *Proceedings of the 5th International workshop on Peer-to-Peer Systems (IPTPS 2006)*, 2006.

[GKA'&al06] C. Gkantsidis , J. Miller , P. Rodriguez, "Comprehensive view of a live network coding P2P system", in *Proceedings of the 6th ACM SIGCOMM conference on Internet measurement*, October 25-27, 2006, Rio de Janeriro, Brazil

[GNU01] M. Ripeanu, "Peer-to-Peer Architecture Case Study: Gnutella Network", in *IEEE 1st* In*ternational Conference on Peer-to-Peer Computi*ng *(P2P'01)*, 27-29 août 2001, Linköpings, Suède. Washington DC : IEEE Computer Society, 2002, pp. 99-100.

[GNT] http://www.generation-nt.com/.

[GON02] L. Gong, "Project JXTA: A Technology Overview", Palo Alto : SUN Microsystems, Inc., 2002, 12 p.

[GPS] W. Yang, N. Abu-Ghazaleh, "GPS: A General Peer-to-Peer Simulator and its Use for Modeling BitTorrent", Department of Computer Science, Binghamton University.

[GUO&al05] L. Guo, S. Chen, Z. Xiao, E. Tan, X. Ding, and X. Zhang, "Measurements, analysis, and modeling of bittorrent systems", in *Proceedings of ACM SIGCOMM Internet Measurement Conference (IMC'05)*, Oct. 2005.

[GUO&al07] L. Guo, S. Chen, Z. Xiao, E. Tan, X. Ding, and X. Zhang, "A performance study of bittorrent-like peer-to-peer systems", in *IEEE Journal on Selected Areas in Communications (IEEE JSAC),* first Quarter of 2007.

[GUS&al05] G. de Veciana, G., and X. Yang, "Fairness, incentives, and performance in peer-to-peer networks," in *41st Annual Allerton Conference on Communication, Control and Computi*ng, Monticello, October 2003.

[HAZ&al06] H. Hazeyama, Y. Kadobayashi, D. Miyamoto, and M. Oe. An autonomous architecture for inter-domain Traceback across the borders of network operation. In *Proceedings of 11th IEEE Symposium on Computers and Communications (ISCC '06), pages 378–385*, June 2006.

[HAM&al07] A. Al Hamra, A. Legout, and C. Barakat, "Understanding the properties of the BitTorrent overlay," INRIA, Sophia Antipolis, Tech. Rep., July 2007.

[HEI&al06] X. Hei, C. Liang, J. Liang, Y. Liu, and K.W. Ross, 'Insightsinto PPLive: A Measurement Study of a Large-Scale P2P IPTV System", in IPTV Workshop, International World Wide Web Conference, May 2006.

[HOF03] J.Hoffman, "About Super-*Seed* mode" http://bittornado.com.

[HMC97] H. Krawczyk, M. Bellare, R. Canetti, "RFC 2104: HMAC: Keyed-Hashing for Message Authentication", February 1997.

[INT] “Intel Peer-to-peer working group”, http://www.p2pwg.org/

[IOA&al02] J. Ioannidis and S. M. Bellovin, “Implementing pushback : Routerbased defense against ddos attacks”, in *NDSS. The Internet Society*, 2002.

[IPO09] Ipoque, Available online at: http://www.ipoque.com/resources/internet-studies/internet-study-2008_2009.

[IPS08] S. Kent and R. Atkinson, “RFC 2401: Security Architecture for the Internet Protocol”, November 2008.

[IZA&al04] M. Izal, G. Urvoy-Keller, E. W Biersack, P. A Felber, A. Al Hamra and L. Garces-Erice, “Dissecting BitTorrent: five months in a torrent’s lifetime”, in *PAM’2004, 5th annual Passive & Active Measurement Workshop*, April 19-20, 2004, Antibes Juan-les-Pins, France.

[JIA&al08] J. Zhang, L. Liu , L. Ramaswamy , C. Pu, “PeerCast: Churn-resilient end system multicast on heterogeneous overlay networks”, *Journal of Network and Computer Applications*, v.31 n.4, p.821-850, November, 2008

[JLI&al04] J. Li, J. Stribling and T. Gil, et al, “Comparing the performance of distributed hash tables under churn”, In S. SCHENKER, G. M. VOELKER, editors, in *3rd International workshop on Peer-to-Peer Systems (IPTPS’04)* in *LNCSc 2004*, 26-27 february 2004, San Diego, California. New York : Springer-Verlag, number 3279, pp. 87-99.

[JON04] G. L. Jones, “On the Markov chain central limit theorem”, Probab. Surv. 1 299 320. 2004.

[JUN&al05] S. Jun and M. Ahamad, “Incentives in Bittorrent induce Free-Riding”, in *Proceeding of the 2005 ACM SIGCOMM workshop on Economics of peer-to-peer systems*, August 22-22, 2005, Philadelphia, Pennsylvania, USA .

[KAR&al05] T. Karagiannis, P. Rodriguez, D. Papagiannaki, “Should Internet Service Providers Fear Peer-Assisted Content Distribution?”, Internet Measurement Conference (IMC), Berkeley, CA, USA, October, 2005.

[KAZ] T. Hargreaves, The FastTrack Protocol, http://www.kazaa.com.

[KEM&al97] RA. Kemmerer, “NSTAT: a model-based real-time network intrusion detection system” in *Technical Report TRCS97-18, Reliable Software Group, Department of Computer Science, University of California at Santa Barbara*, 1997.

[KOR01] E. Korpela, “SETI@home - Massively Distributed Computing for SETI”, in *Computing in Science & Engineering*, January 2001, p. 78-83.

[KOS&al03] D. Kostic, A. Rodriguez, J. Albrecht, and A. Vahdat, “Bullet: High bandwidth data dissemination using an overlay mesh,” in *Symposium on Operating Systems Principles (SOSP)*, 2003.

[KUL&al05] Y. Kulbak and D. Bickson The eMule Protocol Specification. *HUJI-CSE-LTR 2005-3. Jérusalem : The Hebrew University of Jerusalem*, 2005, 68 p.

[KUM&al06] R. Kumar and K. Ross, "Peer-assisted file distribution: The minimum distribution time," *Hot Topics* in *Web Systems and Technologies, 2006. HOTWEB '06. 1st IEEE Workshop on*, pp. 1–11, Nov. 2006.

[LAB] PlanetLab, http://www.planet-lab.org/

[LAC&al02] J. Lacan, L. Lancérica, L. Dairaine, "When FEC Speed Up Data Access in P2P Networks", in *Proceedings of IDMS-PROMS, LNCS*, Springer, 2002.

[LEI&al07] Y. Lei, L. Yang, Q. Jiang, and C. Wu, "Experimental views on neighbor selection in BitTorrent," *Network and Parallel Computing Workshops,2007. NPC Workshops. IFIP International Conference on*, pp. 813–818, Sept. 2007.

[LEG&al05] A. Legout, G. Urvoy-Keller, and P. Michiardi, "Understanding BitTorrent: An Experimental Perspective", *Technical Report (inria-00000156, version 3 - 9 November 2005),* INRIA, Sophia Antipolis, November 2005.

[LEG&al06] A. Legout, G. Urvoy-Keller, and P. Michiardi, "*Rarest First* and *Choke Algorithms* Are Enough", in *Proceedings of ACM SIGCOMM/USENIX IMC'2006,* October 25--27, 2006, Rio de Janeiro, Brazil.

[LEG&al07] A. Legout, N. Liogkas, E. Kohler, and L. Zhang, "Clustering and sharing incentives in BitTorrent systems," in *SIGMETRICS '07: Proceedings of the 2007 ACM SIGMETRICS international conference on Measurement and modeling of computer systems.* New York, NY, USA: ACM, 2007, pp. 301–312.

[LEG&al08] P. Marciniak, N. Liogkas, A. Legout, E. Kohler, "Small Is Not Always Beautiful", in *Proceedings of IPTPS'2008,* February 25--26, 2008, Tampa Bay, FL, USA.

[LEG&al09] S. Le Blond, A. Legout, W. Dabbous, "Pushing BitTorrent Locality to the Limit", *Technical Report (inria-00343822, version 2-12 May 2009), INRIA, Sophia Antipolis,* May 2009.

[LEV&al04] A. Bellissimo, B. N. Levine, and P. Shenoy, "Exploring the use of BitTorrent as the basis for a large trace repository," *Technical report, University of Massachusetts Amherst, Dept. of Computer Science*, Tech. Rep., June 2004.

[LIA&al07] W.-C. Liao, F. Papadopoulos, and K. Psounis, "Performance analysis of BitTorrent-like systems with heterogeneous users," *Performance Evaluation*, vol. 64, no. 9-12, pp. 876 – 891, 2007.

[LIM] Limewire, http://www.limewire.com.

[LIU&al03] J. Liu, Z. Lee, and Y. Chung, "Efficient dynamic probabilistic packet marking for IP traceback", in *Proceedingd of the 11th International Conf. Networks (ICON 2003), Sydney, Australia, , pp.475-480*, September 2003.

[LUA&al04] E. K. Lua, J. Crowcroft and M. Pias, et al, "A Survey and Comparison of Peer-to-Peer Overlay Network Schemes", *IEEE Communications Survey and Tutorial*, 2004, pp. 72-93.

[LUN&al06] Tom H. Luan, and Danny H. K. Tsang, "A simulation study of block management in BitTorrent", *Proceedings of International Conference on Scalable Information Systems (Infoscale)*, Hong Kong, May 2006.

[MAH&al02] R. Mahajan, S. M. Bellovin, S. Floyd, J. Ioannidis, V. Paxson, and S. Shenker, "Controlling high bandwidth aggregates", in *The network SIGCOMM Comput. Commun. Rev., 32(3) :62_73*, 2002.

[MAL&al02] D. Malkhi , M. Naor and D. Ratajczak, "Viceroy: a scalable and dynamic emulation of the Butterfly", In *ACM. 21st symposium on Principles of Distributed Computing (PODC'02)*, 21-24 july 2002, Monterey, California.

[MAY&al02] P. Maymounkov and D. Mazieres, "Kademlia: A peer-to-peer information system based on the XOR metric", in *1st International workshop on Peer-to-Peer Systems (IPTPS'02)*, 07-08 mars 2002, Cambridge, Massachusetts, USA.

[MAY&al03] P. Maymounkov and D. Mazieres, "Rateless Codes and Big Downloads. In *IPTPS'03*, February 2003.

[MBO] H. Eriksson, "MBone: The Multicast Backbone", *Communications of the ACM*, 1994, vol. 37, n° 8, pp. 54-60.

[MEU&al10] M. Meulpolder, L. D'Acunto, M. Capota, M. Wojciechowski, J.A. Pouwelse, D.H.J. Epema, and H.J. Sips, "Public and Private BitTorrent Communities: A Measurement Study", in *9th Int'l Workshop on Peer-to-Peer Systems (IPTPS'10)*, April 2010.

[MIR&al02] J. Mirkovic, PhD Proposal, "D-WARD: DDoS Network Attack Recognition and Defense", January 23, 2002.

[MIR&al04] J. Mirkovic , S. Dietrich , D. Dittrich , P. Reiher, "Internet Denial of Service: Attack and Defense Mechanisms" (*Radia Perlman Computer Networking and Security)*, Prentice Hall PTR, Upper Saddle River, NJ, 2004.

[MIR&al05] J. Mirkovic, M. Robinson, P. Reiher, and G. Oikonomou, "Distributed Defense Against DDOS Attacks", *University of Delaware CIS Department Technical Report CIS-TR-2005-02*, 2005.

[MOB02] C. Perkins, Nokia research center, "IP Mobility Support for IPv4", RFC 3220, IETF, 2002.

[MOO&al01] D. Moore, G. Voelker, and S. Savage, "Inferring Internet Denial of Service Activity". In *Proceedings of the 2001 USENIX Security Symposium, Washington D.C.*, August 2001.

[MR05] A. R. Bharambe, C. Herley, and V. N. Padmanabhan, "Analyzing and improving BitTorrent performance", *Technical Report MSR-TR-2005-03, Microsoft Research*, February 2005.

[NAP] Napster, http://www.napster.com.

[NBR] Cisco Systems Incorporated. Network based application recognition (nbar).

[NOR03] W. B. Norton, "The evolution of the u.s. internet peering system", 2003.

[NS2] NS-2, http://www.isi.edu/nsnam/ns.

[OMN01] Varga, András, "The OMNeT++ Discrete Event Simulation System", in *Proceedings of the European Simulation Multiconference (ESM'2001),* http://www.omnetpp.org.

[OMN09] K. Katsaros, V. P. Kemerlis, C. Stais and G. Xylomenos, "A BitTorrent Module for the OMNeT++ Simulator," in *Proceedings 17th Annual Meeting of the IEEE International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems (MASCOTS),* London, UK, September 2009.

[OPE] Opera, http://www.opera.com.

[OVER07] I.Baumgart, B. Heep, S. Krause, OverSim: A Flexible Overlay Network Simulation Framework, in *Proceedings of 10th IEEE Global Internet Symposium (GI '07)* in *conjunction with IEEE INFOCOM 2007*, Anchorage, AK, USA, May 2007, http://www.oversim.org.

[PAP&al06] I. Papafili, S. Soursos, G. D. Stamoulis, "Improvement of BitTorrent Performance and Inter-Domain Traffic by Inserting ISP-owned Peers", in *6th International workshop on Internet Charging and QoS Technologies (ICQT'09)*, Aachen, Germany, May 2009.

[PCK] Packeteer packetshaper, http://www.packeteer.com

[PEE] M. Jelasity, A. Montresor and G. P. Jesi, "PeerSim P2P Simulator", http://peersim.sourceforge.net/.

[PER08] "Opentransit 2008 Peering Policy", http://vision.opentransit.net/docs/

[PIA&al07] M. Piatek, T. Isdal, T. Anderson, A. Krishnamurthy, and A. Venkataramani. "Do incentives build robustness in BitTorrent?", in *Proceedings of NSDI'07*, Cambridge, MA, April 2007.

[PIC&al07] F. Lo Piccolo and G. Neglia, "The effect of heterogeneous link capacities in BitTorrent-like file sharing systems," *Peer-to-Peer Systems, 2004. International workshop on Hot Topics* in, pp. 40–47, Oct. 2004.

[PIN] The PingER Project, http://www-iepm.slac.stanford.edu/pinger/.

[PLA05] J. S. Plank, "Assessing the Performance of Erasure Codes in the Wide-Area", in *Proceedings of the 2005 International Conference on Dependable Systems and Networks*, p.182-187, June 28-July 01, 2005.

[PLX&al97] C. G. Plaxton, R. Rajaraman, A. W. Richa, "Accessing Nearby Copies of Replicated Objects in Distributed Environment", *Proceedings* ACM Symp. Parallel Algorithms and Architectures, ACM Press, New York, June 1997.

[POU&al04] J.A. Pouwelse, P. Garbacki, D.H.J. Epema, H.J. Sips, "The BitTorrent P2P file-sharing system: Measurements and Analysis", Department of Computer Science, Delft University of Technology the Netherlands.

[POU&al06] J. Pouwelse, P. Garbacki, J. Wang, A. Bakker, J. Yang, A. Iosup, D. H. J. Epema, M. Reinders, M. van Steen, and H. Sips, "Tribler: A social-based peer-to-peer system", in *Proceedings of IPTPS*, 2006.

[PRE&al95] B. Preneel, P.C. van Oorschot, "MDx-MAC and building fast MACs from hash functions" *Crypto'95, Springer LNCS vol.963*, 1995.

[PUS&al09] K. Pussep, S. Oechsner, O. Abboud, M. Kantor and B. Stiller, "*Impact of Self-Organization* in *Peer-to-Peer Overlays on Underlay Utilization*", Fourth International Conference on Internet and Web Applications and Services (ICIW 2009), Venice, Italy Ma y 2009.

[PWG] Peer to Peer Working Group, 2003, http://p2p.internet2.edu/

[QIU&al04] D.Qiu and R. Srikant, "Modeling and performance analysis of BitTorrent-like peer-to-peer networks", in *Proceedings of ACM SIGCOMM 2004*, August 2004.

[QUR04] A. Qureshi, "Exploring Proximity Based Peer Selection in BitTorrent-like Protocol," MIT 6.824 student project, 2004.

[RAS&al07] A. Rasti and R. Rejaie, "Understanding peer-level performance in BitTorrent: A measurement study," *Computer Communications and Networks, 2007. ICCCN 2007. Proceedings of 16th International Conference on*, pp. 109–114, Aug. 2007.

[RAT02] S. Ratnasamy, "A Scalable Content-Addressable Network", *PhD Thesis*, University of California at Berkeley, 2002, 108 p.

[REE&al60] I. S. Reed and G. Solomon, "Polynomial Codes Over Certain Finite Field", J. SIAM, vol. 8, pp. 300–304, 1960.

[REN&al10] S. Ren, E. Tan, T. Luo, L. Guo, S. Chen, and X. Zhang, "TopBT: a topology-aware and infrastructure-independent BitTorrent client", in *proceedings of* IN*FOCOM'10*, San Diego, California.

[RHE&al05] S. Rhea, B. Godfrey, B. Karp, J. Kubiatowicz, S. Ratnasamy, S. Shenker, I. Stoica, and H. Yu, "OpenDHT: A Public DHT Service and Its Uses", in *Proceedings of ACM Special* In*terest Group on Data Communications Conference (SIGCOMM 2005)*, Philadelphia, Pennsylvania, pp. 73-84, August 2005.

[RIO&al98] B. Riou and P. Landais, "Principes des tests d'hypothèse en statistique: α, β et P", Ann Fr Anesth Réanim 17 (1998), pp. 1168–1180.

[RIZ97] L. Rizzo, "Effective Erasure Codes for Reliable Computer Communication Protocols", In *Computer Communication Review*, April 1997.

[ROS&al08] P. Dhungel, D. Wu, B. Schonhorst, and K. W. Ross, "A Measurement Study of Atatcks on BitTorrent *Leech*ers", In *Proceedings of 7th International workshop on Peer-to-Peer Systems*, 2008.

[ROW&al01] A. Rowstron and P. Druschel, "Pastry: Scalable, decentralized object location and routing for large-scale peer-to-peer systems", in *IFIP/ACM. 18th* In*ternational Conference on Distributed Systems Platforms (Middleware'01)*, 12-16 novembre 2001, Heidelberg, Allemagne. Berlin : Springer, 2001, pp. 329-350.

[SAR&al07] P. Saraswat and P. Batra, "An empirical performance evaluation and modelling of BitTorrent peer-to-peer file sharing system using queuing network models," Research project, Advanced Learning and Research Institute, Switzerland, 2007.

[SHE&al04] R. Sherwood, R. Braud, and B. Bhattacharjee, "Slurpie: a cooperative bulk data transfer protocol," IN*FOCOM 2004. Twenty-third AnnualJ*oin*t Conference of the IEEE Computer and Communications Societies*, vol. 2, pp. 941–951, March 2004.

[SHE&al06] D. Levin, R. Sherwood, and B. Bhattacharjee, "Fair file swarming with FOX," in In *Fifth International workshop on Peer-to-peer Systems (IPTPS 2006)*, 2006.

[SNA&al99] SR. Snapp & al. "DIDS (Distributed Intrusion Detection System)-motivation architecture and an early prototype", in *Proceedings of the 14th national computer security conference, Wash*ington *DC*, October 1999.

[SNO] Snort: The Open Source Network Intrusion Detection System, http://www.snort.org.

[SPO&al10] S. Spoto, R Gaeta, M. Grangetto, M. Sereno, "Bittorrent and fountain codes: friends or foes ?", in *Proceedings in IDPDPS 2010*, Atlanta 2010.

[STE&al03] L.D. Stein, J.N. Stewart, "The World Wide Web Security FAQ, version 1.7, February 23rd, 2003", http://www.w3.org/Security/faq/

[STO&al01] I. Stoica, R. Morris, and D. Liben-Nowell, "Chord: A Scalable Peer-to-peer Lookup Protocol for Internet Applications", in *ACM. SIGCOMM 2001 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication*, 27-31 august 2001, UC San Diego, California, USA: ACM Press, 2001, pp. 149-160.

[SUB&al04] L. Subramanian, I. Stoica and R. Katz, "OverQoS: An Overlay Based Architecture for Enhancing Internet QoS", in *1st Symposium on Networked Systems Design and Implementation (NSDI 2004)*, March 29-31, 2004, San Francisco, California, USA, Proceedings 2004.

[THE&al04] S. Androutsellis-Theotokis and D. Spinellis, "A survey of peer-to-peer content distribution technologies", in *ACM Computing Surveys*, 36(4):335–371, December 2004.

[THO&al05] R. Thommes, M. Coates, "Bittorrent fairness: analysis and improvements", in *Proceedings of Workshop Internet, Telecom. and Signal Proceedings*, Noosa, Australia, 2005.

[TIA&al07] Y. Tian, D.Wu and K.W. Ng, "Performance analysis and improvement for Bittorrent-like sharing systems", 2007.

[TOR] *Torrent* 411, http://torrent411.com

[URV&al06] G. Urvoy-Keller and P. Michiardi, "Impact of inner parameters and overlay structure on the performance of BitTorrent", IN*FOCOM 2006.25th IEEE* In*ternational Conference on Computer Communications.Proceed*ings, pp. 1–6, April 2006.

[XIE&al07] H. Xie, A Krishnamurthy, YR Yang and A Silberschatz - "P4P: Proactive Provider Participation for P2P", *Tech. Rep. YALEU/DCS/TR-1377*, Yale University March 2007.

[WAL&al02] D.S. Wallach, "A Survey of Peer-to-Peer Security Issues", *International Symposium on Software Security Tokyo*, Japan, 2002.

[WAN&al07] J. Wang, C. Yeo, V. Prabhakaran, and K. Ramchandran. On the role of helpers in peer-to-peer file download systems: design, analysis, and simulation. In *Proceedings of IPTPS*, 2007.

[WEA&al02] H. Weatherspoon and J. Kubiatowicz, "Erasure Coding vs. Replication: A Quantitative Comparison", R*evised Papers from the First International workshop on Peer-to-Peer Systems*, p.328-338, March 07-08, 2002.

[WEI&al09] W. Liang , J. Bi , R. Wu , Z. Li , C. Li, "On characterizing PPStream: measurement and analysis of P2P IPTV under large-scale broadcasting", in *proceed*i*ngs of the 28th IEEE conference on Global telecommunications, p.3552-3557*, November 30-December 04, 2009, Honolulu, Hawaii, USA.

[WHI&al96] G. B. White, E. A. Fisch, U. W. Pooch, "Cooperating security managers: A peer-based intrusion detection system". *IEEE Network*, 10(1):20-23, January / February 1996.

[WUG&al06] G. Wu and T. cker Chiueh, "How efficient is BitTorrent?" in *Multimedia Comput*ing *and Network*ing *2006*, S. Chandra and C. Griwodz, Eds., vol. 6071, no. 1. SPIE, 2006, p. 607100.

[YAM&al07] S. Yamazaki, H. Tode, K. Murakami, "CAT: A Cost-Aware BitTorrent", *In 32nd IEEE Conference on Local Computer Networks*, pp. 226–227, 2007.

[YAN&al03] B. Yang and H. Garcia-Molina, "Designing a Super-Peer Network", in *IEEE. 21st* In*ternational Council for Open and Distance Education (ICDE'03)*, 05-08 mars 2003, Bangalore, India. Washington DC : IEEE Computer Society, 2003, pp. 49-60.

[XIA&al10] L. Xia and J. K. Muppala, A Survey of BitTorrent Performance, in *IEEE Communications Surveys and Tutorials, to appear.*

[YAN&al06] X. Yang and G. de Veciana, "Performance of peer-topeer networks: Service capacity and role of resource sharing policies," *Performance Evaluation*, vol. 63, no. 3, pp. 175 – 194, 2006, p2P Computing Systems.

[ZAM&al00] E. H. Spafford, and D. Zamboni, "Intrusion detection using autonomous agents", In *Computer Networks, vol. 34, No. 4, pp. 547-570, 2000.*

[ZHA&al04] B. Y. Zhao, L. Huang, J. Stribling, et al, "Tapestry: A Resilient Global-Scale Overlay for Service Deployment", January 2004, vol 22, nº 1, pp. 41-53.

[ZHN&al07] L. Zhang, J. K. Muppala, W. Tu, "Exploiting Proximity in Cooperative Download of Large Files in Peer-to-Peer Networks," iciw, pp.1, Second International Conference on Internet and Web Applications and Services (ICIW'07), 2007

[ZHA&al10] C. Zhang, P. Dhungel, Di Wu, Z. Liu,and K.W. Ross, "BitTorrent Darknets," in *Proceedings of Infocom 2010*, San Diego, 2010.

[ZHU&al01] S. Q. Zhuang, B. Y. Zhao, A. D. Joseph, R. H. Katz, and J. D. Kubiatowicz, "Bayeux: An architecture ZHA&al05e for scalable and fault-tolerant wide-area data dissemination", in *Proceed*ings *of NOSSDAV* (June 2001).

Appendix: Related works on BitTorrent

Performance studies	**Measurement**	**[IZA&al04]:** peers' evolution and the traffic volume downloaded and uploaded during the five months analysis. **[POU&al04]:** peers' number and their relation with *Seeds*, the time peers stay as *Seeds* after completing the download, and the resource lifetime. Eight months analysis. **[LEV&al04]:** information has been collected from two different *Trackers* to show that BitTorrent can serve large files. The *Torrent* is observed during a four months period. **[LEG&al05]:** Complete study of BitTorrent mechanisms. **[AND&al05]:** a study on the cooperation in BitTorrent. The authors focuses on free-riding, *Seeds* importance and sharing ratio. Free-riders are less important than in other P2P networks. **[LEG&al06]:** show the efficiency of *Rarest First* and *Choke Algorithm*, proving that they are enough for BitTorrent performances. **[LEG&al07]:** experiments on some private *Torrents* show how a peer acts individually in BitTorrent. **[RAS&al07]**: some peers used to study the BitTorrent performance but not with an overall view. The performance distribution of peers is uniform and that the *Choke Algorithm* is the main factor that has an impact on BitTorrent performance. **[LEI&al07]:** The neighbor selection with the *Choke Algorithm* has been studied. This random selection does not take into consideration the communication cost and that this results in low transmission rate and high cost. **[PIA&al07]:** study of BitTorrent incentive. The authors demonstrate that all peers contribute resources that do not directly improve their performance. They develop BitTyrant BitTorrent client. **[DAL&al08]:** proposed a study of these different types of networks. They also explain that this is not obvious to say that BitTorrent implicitly clustered peers as we could read it in [LEG&al07] except in the initial stage. **[ZHA&al10]**: the behavior study of BitTorrent in Darknets from macroscopic, medium-scopic and microscopic perspectives.
	Simulation	**[FEL&al04]:** deep study on piece and peer selection strategies and their effects on BitTorrent performance. See if the self-scaling and self-organizing aspects of P2P networks are encouraging to reach a highly efficient, cost effective and robust content distribution protocol. **[MR05]:** a very complete study of BitTorrent mechanisms. The performance is studied during flash-crowd using a simulator that models all peers activities, policies and mechanisms. C# simulator. **[THO&al05]:** a simulation based analysis proposed to study the fairness properties of BitTorrent. The authors also propose three modifications to BitTorrent and examine their impact on fairness. They show that these modifications provide positive amelioration to BitTorrent fairness incentive mechanism. **[JUN&al05]:** study of the incentive mechanism in the *Choke Algorithm* showing that the high number of Free-riders in BitTorrent. There is a lot of egoist peers that do not participate to the *Torrent* evolution and that finish to download faster than other peers. **[URV&al06]:** work that evaluates the impact of the overlay topology parameters on BitTorrent performance. The authors show that the Peer Set size and the percentage of outgoing connections have a significant impact on the BitTorrent's performance. **[WUG&al06]:** the authors try to find how BitTorrent can be optimal or close to it. They proposed a new distribution scheme called the Centrally Scheduled File Distribution (CSFD) that can decrease considerably the total download time. BitTorrent is far to be optimal and the peer selection is not helping the protocol to decrease optimally

		the overall distribution. **[GAR&al06]:** *2Fast* solves the problem of freer-riding that affect the download performance, while preserving fairness of bandwidth sharing. The authors propose to form groups of peers that collaborate in downloading a file on behalf of a single group member which can thus use its full download bandwidth. A peer can help other peers in their ongoing downloads and get in return help during its own downloads. **[CWU&al07]:** improve the download time of BitTorrent with a new strategy that replace the *Rarest First* policy and introduces a strategy based on a greedy concept that a peer assigns each missing piece with the highest priority for next download. The strategy is called the weighty piece selection strategy. **[WAN&al07]:** improvement study that take into consideration the asymmetric bandwidth in Internet and BitTorrent where upload capacity is generally limiting the transfers. The authors show that integrating also new peers called helpers can add a considerable amelioration as effective as *Seeds*. **[HAM&al07]:** authors follow the work on [URV&al06] and perform an evaluation study on the properties of the distribution overlay in BitTorrent and the relation of this overlay and BitTorrent performances. MATLAB simulation. **[CHO&al08]:** presents simulations study that show how to use more intelligently *Seed* capacity in BitTorrent while improving the performance of contributing nodes. **[CHE&al09]:** two main *Seed*ing strategies have been studied in details based on simulation with a Java simulator and a mathematical model. The original *Seed*ing strategies known in the *Choke Algorithm* is compared to the time-based *Seed*ing strategy where the *Seed* provide data to *Leeches* during a uniform interval of time. Both free-riders and exploiters harm the system despite the *Seed*ing strategies that is used.
	Analytical	**[QIU&al04]**: fluid-flow model. This work proposes a deterministic model describing the evolution of the *Leeches* and *Seeds* number also. They conclude that the average download time is not related to the arrival rate and that the system scales very well with the peers increasing. They proposed a validation part based on both simulation and real traces obtained from the Internet. **[GUS&al05]:** the parameter that is studied principally is the service capacity for transient and steady-state regimes. Authors concluded by proposing a specific fairness policy that they assume to be better for dynamic P2P system like BitTorrent-like ones. The authors also validated partially their work using traces obtained from a second generation P2P application. **[ART&al05]:** analysis of data dissemination is proposed. The Tit For Tat strategy is ignored and the authors assumed that the peers are homogeneous. **[ERM&al05]:** modeling methodology and some measurement to study the entire session characteristics of BitTorrent. BitTorrent session inter-arrivals can be modeled by the hyper-exponential distribution while session duration and sizes can be modeled by the lognormal distribution. **[GUO&al05]**: a work inspired from [QIU&al04]. The authors analyzed the file downloading trace files obtained from *Trackers*. They conclude saying that the peer arrival rate to a *Torrent* is exponentially decreases. **[YAN&al06]:** extension of [GUS&al05] work. **[KUM&al06]**: an analytical model of file sharing in P2P networks using also a fluid-flow model. They proposed to study the advantages of a P2P file sharing protocol comparing to a Client-Server system. They tried to find the minimum download time to finish the distribution to all *Leeches* in the system. **[FAN&al06]:** the authors also based on the famous work in [QIU&al04]: to propose a model based on a stochastic differential equation approach. They divided peers into three types which are

		Leeches that have a few pieces, *Leeches* that have most of pieces and *Seeds*. **[GUO&al07]**: a work proposing incremental work comparing to the precedent on extensive measurement and trace analysis. This result permits them to extend the work in [GUO&al05]. **[TIA&al07]:** a deep and complete work that extended the fluid model in [QIU&al04] to study the peers in different states of the download. **[SAR&al07]:** present a queuing model for BitTorrent where each peer is considered as a load dependent host. The service is divided into slots and a request time is equal to the time needed to download one piece. The work focuses on the download behavior and the incentive mechanism. **[PIC&al07]:** the first work that proposes a study based on heterogeneous fluid-flow model. To work on different access link capacities the authors developed a model with two different capacity classes by extending the work in [QIU&al04]. **[LIA&al07]:** propose a model based on heterogeneous networks with also two classes: high and low peers. The authors propose a mathematical model that helps them to predict the average file download delay for both classes of peers. The used the BTSim simulator [BTS] to experiment the proposed model.
BitTorrent-like Protocols		***Slurpie*** **[SHE&al04]**: Mesh where download bandwidth is adapted dynamically with estimation technique. ***FOX*** **[SHE&al06]**: Protocol that focuses on fairness and symmetric communication. ***Avalanche*** **[GKA'&al06]**: Protocol proposed by Microsoft and integrating Network Coding.
Locality-Aware techniques for BT		**[QUR04]:** proposal of a new peer selection based on proximity. The *Tracker* sends information on nearby peers to improve the download. They compare an approach where the requesting peer floods an announcement to discover peers and an approach based on Gossip protocols that use a low-rate probabilistic flooding mechanism. **[KAR&al05]:** study on the impact of ISPs on BitTorrent. BitTorrent is locality unaware and this increases ISP costs. Their showed that in BitTorrent the content is sent 30 to 70% more times and that some mechanisms can help decreasing the inter-ISP traffic. **[BIN&al06]:** biased neighbor selection to improve traffic locality. Select peers according to each of these AS numbers. The *Tracker* forces each new *Leech* to select a majority of his neighbors within the same ISP and only few outside of it. This solution has no proposition for the peers mapping with the AS's. **[PAP&al06]:** extended the solution in [BIN&al06] by inserting ISP-Owned peers to enhance performance within an ISP domain. **[POU&al06]:** proposes a novel social-based P2P system that exploits social phenomena by maintaining social networks and using these in content discovery and delivery. Tribler is composed by a set of extensions to BitTorrent. **[AGG&al07]:** evaluation of the feasibility of a solution where the ISP offers an oracle to P2P users. The peers provided a list of neighbors and the oracle ranks them according to certain criteria like proximity or bandwidth. **[ZHN&al07]:** explore the use of proximity in the construction of the overlay network and the efficient exchange of the file fragments In BitTorrent with the main goal of reducing download time and resource usage. **[YAM&al07]:** proposal for a method to constitute P2P content distribution networks and a reduction in ISP costs by considering the form of the ISP interconnection in its distribution. The authors show that the proposal achieves a reduction in ISP costs and distribution time. ***Ono*** **[CHF&al08]:** pluggin added to Azureus client considering long distance transfers to increase file download speeds. Ono learns from existing Content Distribution Networks (CDNs) such as Akamai

	[AKA] and Limelight. It is a customer oriented service. **[CUI&al09]:** measurement study of locality-aware P2P solutions over real Internet AS topology using the accesses of nodes in PlanetLab. The authors propose an evaluation of the performances of a set of locality-aware solutions. They point out the necessity to tradeoff between the goals of optimizing AS performance and fairness among peers. **[LEG&al09]:** locality enhancement and a large study showed how far locality can be pushed and what the traffic economy gain we can have. ***TopBT*** **[REN&al10]:** Topology BitTorrent is a topology-aware version of BitTorrent protocol implemented also as a plugin to Azureus (Vuze) client also that discovers network proximity peers by sending requests and waiting for responses. Customer oriented service.
Erasure Codes in BT	**[PLA05]:** A performance study where the authors present an assessment for erasure codes in the wide area. **[WEA&al02]:** comparison between replication and erasure codes. **[BYE&al99]:** FEC-based alternative for multicast distribution with a parallel access to mirror sites using Tornado Codes [COH&al02]. **[LAC&al02]:** a solution to speed up data access in P2P networks is developed with dilution of FEC fragments over all the peers based on a data storage scheme. This permits to increase the blocks entropy and choice. **[KOS&al03]:** Bullet is a distributed and scalable algorithm proposed in where peers self-organize into a high bandwidth overlay mesh. In this algorithm the peers locate and retrieve data from multiple peers in parallel. **[MAY&al03]:** the authors propose a simple algorithm that allows big files to be downloaded from multiple sources in P2P. The solution proposes low handshaking overhead between peers. The interest is that when two peers have partially downloaded the resource, they can benefit from each other resource parts. **[GKA&al06]:** Network Coding which is an alternative of FEC. In this mechanism the blocks are obtained by a combination of resource held by other peers. This technique, which is a channel coding and not a source coding like FEC, is implemented by Avalanche which is the Microsoft BitTorrent-like P2P application. **[BIC&al07]:** BiCod uses the Network Coding mechanism, no development has been proposed for Avalanche yet even if in [GKA'&al06] the authors show that this new proposition may surpass FEC performance. **[LUN&al06]:** the authors studied the block management in BitTorrent and their circulation. They conclude that the block distribution in BitTorrent is far from being optimal in terms of block frequency and that some blocks dominate the network and that others become extinct nearly. The coding use is the one presented in [BYE&al98]. **[SPO&al10]:** is a recent work where the authors explain that in real world applications like real time constraints ones are affected by flash crowds because peers join and leave quickly. They propose a modification to GPS simulator [GPS] integrating LT (Luby-Transform) codes.
Architectures for ISP and P2P collaboration	***ALTO*** **[XIE&al07]:** project defined in an IEEE draft a specification for the called P4P that permit coordination and collaboration between an ISP in the P2P activities and applications. This proposition is more general to all P2P or overlay applications. ***SmoothIT*** **[PUS&al09]** consortium proposes also an architecture relying on various criteria to evaluate the traffic management in P2P systems. However their architecture seems complex to implement legally and technically.