Rapport de Stage de Master Réseaux 2004/2005
Université Pierre et Marie Curie (Paris 6)

Streaming Pairs-à-pairs : Impact des Protocoles de Gestion de l'Overlay

Encadrant : Olivier FOURMAUX
Thomas SILVERSTON

Remerciements :
Un grand merci à tous les stagiaires du bureau du 12ème étage (Chloé, Meriem, Rym, Didier, Fehmi, Hicham, Nasreddine, Thomas)
ainsi qu'à toute l'équipe R&P du Lip6, et notamment Olivier Fourmaux.
Je tiens également à remercier tous les étudiants du Master qui m'ont permis de passer une très bonne année, et particulièrement mes binômes au cours des différents projets (Karim Benhamouche, David De Bastos, Erwan Ben Souiden, Kau-huy Chhour, Yassine Chetoui, Surya Arbi, Mohammed Galiani, Wassin Ben Amor).
Ce stage de fin d'étude marque la fin d'un cycle, ainsi je profite de l'occasion qui m'est donnée pour remercier toutes les personnes qui ont compté pour moi pendant toutes ces années et qui m'ont permis -grâce à leur soutien- d'avancer.
Ces remerciements s'adressent en général à toutes les personnes de la '*boustos team*', à tous ceux de l'autre côté...et ceux d'ailleurs ;).

Abstract

Live streaming applications are actually increasing on the Internet. These applications use protocols which involve group communications : one or more sources to many receivers (*multicast*) with real time constraints. At the present time, protocols designed for this kind of communication don't rely on the client/server simple model usually used with the Internet, but organize the receivers on an *overlay* network and receivers are supposed to work in collaboration with each other, that is mean in a *peer-to-peer* network.

First part of my works was to make a thorough study of these protocols that I achieve to classify in three different categories: *source-driven* protocols, *reveiver-driven* protocols and *data-driven* protocols.

Each of these categories manages the overlay differently, so I try to compare them to specify which of these categories is more appropriate to this kind of protocol. That's why I choose to simulate some protocol's behaviour.

After a meticulous study of available simulators, I choose to implement a new simulator for this kind of network and I select two protocols for simulations: a source-driven protocol and a data-driven protocol.

To my knowledge, these works are the firsts to compare, on the same framework, different categories of protocols for live streaming with peer-to-peer network.

From the simulations, I note that nodes origanisation on the overlay influences drastically network global performances, and data-driven approach seems more appropriate for these protocols.

My futures works will be firstly to adapt the configuration of simulator and simulated protocols to an Internet-like configuration to accurate the simulation results, and secondly I will try to correct existing protocols and to design an efficient protocol for this kind of communication.

keywords : Peer-to-peer, Applicative Multicast, Overlay, Streaming, Multimedia, Epidemic (gossip) Algorithm, Simulation

Resume

Les applications de *streaming* en direct sont un type d'application en plein essor actuellement sur Internet. Ces applications utilisent des protocoles permettant des communications de groupe : une ou plusieurs sources vers plusieurs récepteurs (*multicast*), et ce en direct (ou temps réel) donc avec des contraintes temporelles. Actuellement, les protocoles conçus pour ce type de diffusion ne reposent plus sur le simple modèle client/serveur communément utilisé dans Internet mais sur l'organisation des récepteurs en *overlay* et sur leur travail collaboratif c'est à dire en réseaux *pairs-à-pairs*.
La première partie de mes travaux fut de réaliser une étude approfondie de ce type de protocole que je suis parvenu à classer en différentes catégories : les protocoles utilisant une approche *orientée-source*, une approche *orientée-récepteur* ou une approche *orientée-données*.
Chacune de ces approches ayant une gestion de l'overlay différente, j'ai cherché à les comparer afin de déterminer laquelle de ces approches était la plus appropriée pour ce type de transmission. Pour cela j'ai choisi de simuler le comportement de certains de ces protocoles.
Après avoir largement étudié les simulateurs existants, j'ai implémenté un nouveau simulateur de ce type de réseau et j'ai choisi deux protocoles à simuler, l'un utilisant une approche orientée-source, l'autre une approche orientée-données.
Ces travaux sont à ma connaissance les premiers à comparer entre elles, sur une même plateforme de simulation, différentes approches utilisées par les protocoles de streaming en direct via un réseau pairs-à-pairs.
De ces premières simulations, j'ai pu constater que les différentes organisations des noeuds dans l'overlay influencent énormément les performances globales du réseau, et que l'approche-orientée données parait plus appropriée pour ce type de protocole.
Les objectifs futurs de mes travaux seront dans un premier temps d'adapter la configuration du simulateur et des protocoles à des configurations plus proches de celles rencontrées dans Internet afin d'affiner les résultats des simulations, puis dans un second temps de chercher à palier les imperfections de protocoles existants et de concevoir un protocole efficace pour ce type de communication.

Mots clés : Pairs-à-pairs, Multicast applicatif, Overlay, Streaming, Multimédia, Algorithmes épidémiques, Simulation

Table des matières

Table des figures

Chapitre 1

Introduction

Le succès d'internet a engendré de profonds changements dans son utilisation. D'un simple réseau d'interconnexion entre quelques universités américaines pour permettre aux chercheurs de s'échanger rapidement et simplement des informations (Arpanet [36]), il est devenu un réseau d'interconnexion mondiale qui transportent tous types de données (textes, paroles, vidéos), et un élément de plus en plus incontournable dans le commerce et l'économie planétaire.

A mesure que son nombre d'utilisateurs augmentaient, de nouvelles applications apparaissaient et contribuaient à accélérer d'avantage son développement. On peut par exemple citer les browsers webs et le formatage des informations en pages htmls([37]) qui ont permis une utilisation d'internet beaucoup plus simple car ils facilitaient son exploration et la présentation des informations via une interface visuelle. Plus récemment, on peut également citer les applications pairs-à-pairs.

1.1 Pairs-à-Pairs

A l'inverse du traditionnel modèle client/serveur utilisé jusqu'alors dans internet, c'est à dire un modèle où les rôles des participants sont clairement séparés entre les utilisateurs qui souhaitent recevoir un contenu et les fournisseurs de contenu qui le transmettent, les applications pairs-à-pairs sont un type d'application où les rôles des participants ne sont pas a priori définis. Chacun peut jouer à la fois le rôle de client, de serveur ou bien les deux à la fois. Ce modèle permet une plus large diffusion du contenu par rapport au simple modèle client/serveur puisque plutôt que d'être accessible en un unique point du réseau, à savoir le serveur, le contenu peut maintenant être rétrouvé chez des utilisateurs ayant déjà récupéré le contenu. Ainsi, le contenu devient disponible en plusieurs points du réseau à la fois, et il devient plus facilement récupérable par les autres utilisateurs intéressés puisque ce n'est plus uniquement le serveur qui met ses ressources à dispositions de tous les autres récepteurs (bande passante, puissance de calcul), mais les récepteurs qui partagent entre eux leurs ressources

pour permettre la diffusion du contenu. Le fait de partager les ressources des utilisateurs pour les faire collaborer afin de parvenir à un même but commun est le concept clé du pairs-à-pairs. C'est ainsi que les applications pairs-à-pairs de diffusions de fichiers ont très largement fait parler d'eux ces dernières années (Napster[42], Kazaa[44], eDonkey[45], BitTorrent[17]) et ont aussi contribué à accélérer la croissance d'internet.

Concernant les applications pairs-à-pairs sur internet, bien qu'elles soient relativement récentes, il est bon de rappeler que le concept de pairs-à-pairs dans le réseau internet est très ancien. En effet, internet n'est finalement qu'un réseau de routeurs pairs qui collaborent entre eux pour permettre au réseau de conserver sa principale caractéristique : fournir un service *best effort* à ses utilisateurs. En effet, les routeurs échangent entre eux des informations sans qu'il y ait une quelconque notion de hiérarchie entre eux ou de rôles prédéfinis à l'inverse du modèle client/serveur. A une autre échelle du réseau internet, les routeurs administrés au sein d'un même ensemble forment un système autonome (*AS*). le routage entre AS (via *BGP-4* [38]) repose sur l'échange d'informations utiles entre AS qui collaborent sans qu'il y ait non plus une quelconque hiérarchie entre eux. En fait la notion de pairs-à-pairs reposent sur la collaboration entre les différentes entités pour atteindre un but commun à tous, et c'est cette collaboration qui nécessite le partage des ressources.

1.2 Communication de groupe (Multicast)

Parallèlement à ce type de développement, alors qu'Internet est nativement conçu pour des communications point à point, *unicast*, c'est à dire un unique émetteur vers un unique récepteur, l'augmentation du nombre d'utilisateurs a entraîné la nécessité pour internet de disposer de fonctionnalités (mécanismes) de communication de groupe, le *multicast*, c'est à dire un ou plusieurs émetteurs vers un ou plusieurs récepteurs. La communication de groupe nécessitent entre autre de dupliquer tous les messages vers les récepteurs . Il a alors été proposé d'implémenter les fonctionnalités des communications de groupe au niveau réseau (*IP*) puisque dans cette solution, les paquets sont dupliqués au niveau des routeurs ce qui présente l'avantage de ne pas surcharger inutilement les liens du réseau avec plus d'une copie du même message par lien.

Cependant, le déploiement du multicast réseau sur internet a été considérablement freiné pour diverses raisons aussi bien techniques, que politiques ([20]). Tout d'abord, pour que la totalité du réseau Internet soit capable de supporter des communications de groupe au niveau réseau (IP), il faut une évolution complète des infrastructures d'Internet afin que tous les routeurs supportent les mécanismes du multicast. Ces changements d'infrastructures sont coûteux pour les fournisseurs d'accès à internet, et ont ralenti le déploiement du multicast de niveau réseau. En plus, le multicast au niveau réseau nécessite que les routeurs maintiennent un état pour chaque groupe de communication. Le fait de maintenir un état par groupe implique que le coeur du réseau internet doit être capable de certaines tâches complexes ce qui va à l'encontre des principes de

base d'internet : intelligence en bordure du réseau et simplicité dans le coeur du réseau.

Le plan d'adressage d'internet, à l'origine par classe (A, B, C), repose maintenant sur l'agrégation des adresses (*CIDR* [39]) afin d'économiser le nombre d'adresses à attribuer et de diminuer la taille des tables de routage des routeurs. Les adresses multicast (classe D) ne peuvent pas s'agréger comme les adresses unicast (classe A, B, C), puisqu'elles ne sont pas attribuées en correllation avec le plan d'adressage définis pour internet (CIDR). Ainsi, l'utilisation des adresse multicast (classe D) pourrait engendrer une augmentation de la taille des tables de routages ce qui ne permet pas un passage à l'échelle du multicast de niveau réseau.

De plus, les mécanismes de contrôle de congestion et de flots sont très difficilement utilisables pour les flots multicast. En effet, les fournisseurs d'accès à internet pourraient ne pas avoir pleinement connaissance des flots qui circulent dans leur réseau puisqu'ils n'auraient pas connaissance -à priori- des duplicatas des flots crées dans leur réseau. Cette non-connaissance du trafic dans leur réseau pourrait entraîner sa saturation au point qu'ils ne puissent plus fournir aucun service à leurs utilisateurs. Ainsi, les fournisseurs d'accès à internet refusent à leurs clients d'utiliser les mécanismes du multicast de niveau réseau. Cependant, certains fournisseurs d'accès les emploie vers leurs utilisateurs, puisque dans ce sens ils ont connaissance de la quantité de trafic qui circulera dans leur réseau. C'est le cas pour la diffusion de certaines télévision sur internet.
Toutefois, on constate actuellement que le multicast de niveau réseau n'a pas permis de répondre aux besoins d'effectuer des communications de groupe sur internet.

1.3 Multicast Applicatif

Plutôt que d'implémenter les fonctionnalités des communications de groupe au niveau réseau (IP) c'est à dire au niveau des routeurs donc au coeur du réseau internet, une autre solution est de reporter ces fonctionnalités en périphérie du réseau, c'est à dire au niveau des *hôtes*. Ainsi, plutôt que de dupliquer les paquets au niveau des routeurs, ce sont les hôtes eux-mêmes qui se chargent de dupliquer les paquets vers les autres hôtes récepteurs. On appelle ce type d'approche du *multicast applicatif* car les fonctionnalités du multicast sont implémentées au niveau de la couche applicative.
L'avantage de cette méthode par rapport au multicast de niveau réseau est qu'elle peut facilement être déployée puisqu'elle ne nécessite pas de modifications de l'infrastructure d'internet. Il s'agit également d'une solution très flexible car les hôtes en bordure du réseau sont des entités 'intelligentes' capables de plus de traitements que les routeurs de coeur du réseau pour par exemple décider à qui transmettre le message pour que la communication s'effectue le plus efficacement possible. A ce propos, les routeurs de coeur du réseau sont bien sûr suffisamment puissants pour effectuer le genre de traitement que l'on demande aux hôtes, mais leur rôle est de servir d'entités stables pour qu'internet puisse

assurer continuellement son niveau de service de type '*best effort*'. C'est pourquoi les routeurs ne pourraient se permettre d'effectuer d'autres calculs que ceux permettant d'assurer le service d'internet. Par ailleurs, la solution qui consisterait à implémenter les fonctionnalités du multicast dans des routeurs dédiés disséminés sur Internet ne semblent pas non plus être une solution plus viable et déployable que le multicast de niveau réseau([7]).

1.4 Réseau Overlay

Pour permettre aux hôtes de s'organiser afin de réaliser des communications de groupe, chaque hôte considère qu'il est connecté directement à d'autres hôtes qui constituent son voisinage. Cette vision du réseau dont dispose les noeuds est complètement décorellée de la véritable topologie du réseau sous-jacent (Internet). On dit que les noeuds ont une vision abstraite, virtuelle du réseau, et qu'ils s'organisent en un overlay (cf. FIG.1.1).

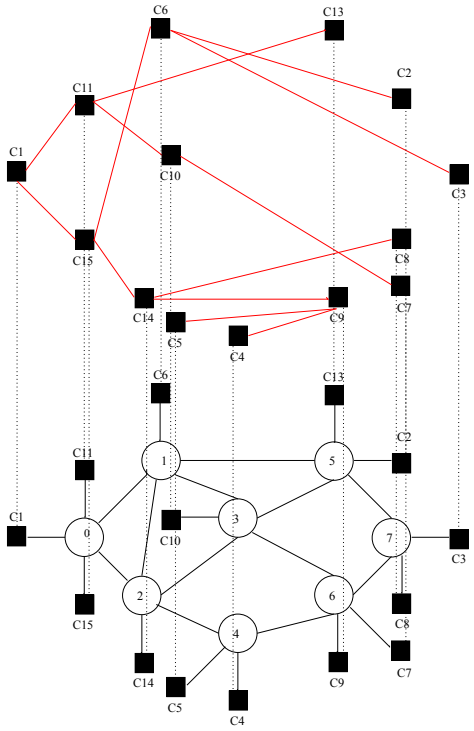

FIG. 1.1 – Vision virtuelle du réseau (Overlay)
En haut : réseau virtuel, En bas : réseau réel.

Dans le cadre du multicast applicatif, comme ce sont les hôtes qui implémentent les fonctionnalités des communications de groupe, l'objectif des hôtes

membres du groupe va être de s'organiser en une topologie virtuelle efficace pour diffuser les messages à tous les membres du groupe. Comme exemple de topologies, on peut citer des topologies en arbre de diffusion, en maillage ou en anneau.

Toutefois, le multicast au niveau applicatif n'est pas aussi performant que le multicast de niveau réseau. En effet, ce sont les hôtes qui dupliquent et transmettent les données aux autres membres du groupe. Suivant la topologie créée dans l'overlay, chaque hôte transmet le message à des voisins, donc comme les hôtes voisins dans l'overlay sont directement connectés -virtuellement-, une seule copie du message traverse les liens de l'overlay. Cependant, des liens différents dans l'overlay peuvent reposer sur de mêmes liens dans le réseau sous jacent. Ainsi, on peut en réalité observer des duplications du message sur les même liens dans le réseau réel ce qui constitue une mauvaise utilisation des ressources du réseau, avec notamment une consommation inutile de la bande passante.

En plus, bien que les hôtes voisins soient directement connectés dans l'overlay, on doit garder à l'esprit qu'ils ne le sont que virtuellement. Ainsi, le chemin dans l'overlay peut être plus couteux en nombre de sauts que le chemin direct, donc entraîner de plus important délais de transmission. (cf : FIG. 1.2)

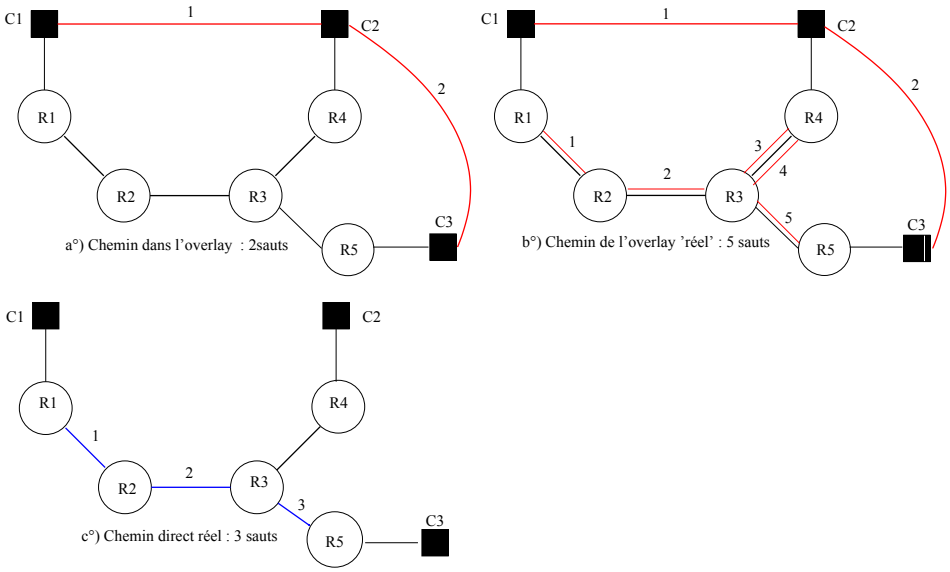

FIG. 1.2 – Nombre de sauts (overlay, réel, direct)

Ainsi, certaines métriques ont été introduites pour quantifier la qualité de la structure de l'overlay.
Parmi ces métriques, on trouve le *stress*, qui correspond au nombre de duplicatas de paquets envoyés sur un même lien du réseau réel, et le *stretch*, qui correspond au rapport entre la longueur réel (en termes de sauts, '*hops*') du chemin

entre l'émetteur et le récepteur dans l'overlay à la longueur du chemin direct dans le réseau réel. Par exemple (cf. FIG. 1.2), la longueur réel du chemin entre l'émetteur et le récepteur dans l'overlay est de 5 sauts (cas b), etla longueur direct du chemin dans le réseau réel est de 3 sauts (cas c), le stretch vaut alors $\frac{5}{3}$.Une optimisation de l'overlay est donc de faire tendre ces métriques vers un : une unique copie du paquet sur chaque lien réel et un chemin dans l'overlay de même longueur que celui du chemin réel.
Naturellement, seul le multicast de niveau réseau (IP) peut avoir ces deux métriques à un. En effet, ces métriques ont été introduites pour comparer le coût des communications dans l'overlay à ceux du multicast de niveau réseau, et le multicast de niveau réseau est justement conçu pour qu'un unique paquet traverse les liens réels. La longueur du chemin est quant à elle toujours égale à un puisqu'au niveau d'internet (réseau réel) il s'agit forcément du chemin direct entre émetteurs et récepteurs.

Enfin, alors que dans le multicast de niveau réseau, les routeurs qui implémentent les fonctionnalités des communications de groupe sont des infrastructures dédiées et relativement stables dans le temps, les hôtes ne sont pas des entités stables dans le temps. Il n'est ainsi pas rare d'observer des hôtes rejoindre et quitter très rapidement le réseau suivant le bon vouloir des utilisateurs (départ explicite) ou même d'observer de nombreuses pannes et autres défaillances de ces hôtes (départs non explicite). De ce comportement, il en résulte une grande dynamicité des hôtes dans le réseau, ce qui le rend assez complexe à gérer puisqu'il faut déployer des mécanismes de tolérance aux pannes efficaces afin que le départ (explicite ou non) d'un ou plusieurs hôtes n'affecte pas tout le réseau overlay.

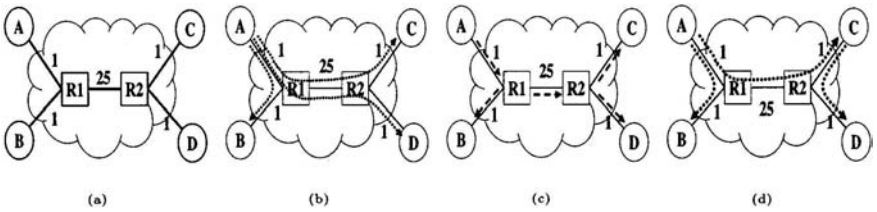

FIG. 1.3 – Illustration des différents types de multicast (plusieurs unicast, niveau réseau, niveau applicatif)

Les différences entre les solutions envisageables pour implémenter les mécanismes du multicast sont présentés dans la figure 1.3. FIG. 1.3(a) présente un ememple de topologie où R1 et R2 sont routeurs pendant que A, B, C, D sont des hôtes. Les délais des liens sont indiqués, et l'on peut remarquer que le lien R1-R2 est un lien très coûteux par rapport aux autres liens. FIG. 1.3(b) illustre la solution naïve du multicast qui effectue plusieurs connexions unicast. Cette solution entraîne une duplication de copies des messages dans les liens, et notamment dans le lien coûteux R1-R2. FIG. 1.3(c) illustre la solution du

multicast niveau réseau. Dans cette solution, les transmissions supplémentaires de paquets sont évitées donc une seule copie du paquet traverse chaque lien. De plus, chaque récepteur reçoit les données avec le même délai que si la source leur transmettait directement en unicast. FIG. 1.3(d) illustre la solution du multicast applicatif où les paquets sont dupliqués au niveau des hôtes. Le nombre de copies est réduit par rapport à la solution naïve du multicast mais pas par rapport au multicast de niveau réseau.

1.5 Multicast Pairs-à-pairs

Les communications de groupe mettent en jeu plusieurs entités dont certaines jouent le rôle d'émetteurs et les autres de récepteurs. Pour effectuer ce type de communication, plusieurs solutions sont envisageables. La plus simple de toutes consiste à effectuer N connections unicast de l'émetteur vers les N récepteurs. Cette solution présente l'avantage d'être utilisable immédiatement mais constitue un important gaspillage en ressources du réseau puisque les mêmes copies des messages sont alors transmises plusieurs fois sur les mêmes liens physiques, ce qui risque de congestionner gravement le réseau. De plus, la source du flot pourrait ne pas avoir les capacités à supporter les N connections unicast, ce qui dans le meilleur des cas empêcherait de nouveaux récepteurs de recevoir le flot de données, et dans le pire des cas empêcherait tous les récepteurs de le recevoir si la source devait s'écrouler devant la charge des récepteurs. Bien que cette solution ne soit pas performante, elle est tout de même actuellement très utilisée sur internet.
Une solution plus appropriée pour ce type de communications de groupe est bien sûr le multicast niveau (IP). Mais, comme on l'a expliqué précédemment, son déploiement limité et le refus des fournisseurs d'accès à internet à accepter ce trafic dans leur réseau empêche son utilisation.
Les solutions actuellement développées sont celles qui placent les fonctionnalités des communications de groupe en bordure du réseau dans les hôtes, c'est à dire le multicast applicatif. Pour ce faire, nous avons vu que les hôtes s'organisaient eux-mêmes dans un réseau overlay qui est un réseau virtuel entre les hôtes qui fait abstraction du réseau internet (IP) sous-jacent.

Dans le cadre d'une communication de groupe, quelque soit le rôle des entités (émetteurs ou récepteurs), leur but est que la communication ait effectivement lieu, et ce dans les meilleurs conditions pour chaque entité. Toutefois pour les communications de groupe, comme le multicast applicatif ne peut être aussi efficace que le multicast réseau puisque l'overlay sur lequel sont organisés les hôtes est décorélé du réseau sous jacent (internet) et engendre donc une moins bonne utilisation des ressources de ce réseau (bande passante, délais de transmission), il faut ajouter des mécanismes permettant une meilleure utilisation des ressources du réseau réel. Pour cela, on peut envisager d'organiser les hôtes dans l'overlay de façon à ce qu'ils collaborent pour partager judicieusement les ressources disponibles entre eux et pour atteindre l'objectif de la communication de groupe, c'est à dire que l'on peut choisir d'organiser les hôtes en un réseau

collaboratif sur l'overlay, c'est à dire un *réseau pairs-à-pairs sur l'overlay.*

1.6 Les Application de Streaming

Après avoir défini les notions de pairs-à-pairs, de communication de groupe et d'overlay,nous pouvons maintenant nous intéresser aux applications qui nécessitent la mise en oeuvre de ce type de mécanisme.
Comme nous le disions précédemment, internet a connu une croissance phénoménale ces quinze dernières années. Cette croissance a en partie été due à la discrète apparition d'applications ou de services qui ont rendu le réseau attrayant pour de nouveaux utilisateurs. Parmi ces applications on peut citer initialement les messageries (mail), puis les messageries instantanées (ICQ[48], MSN[41]) ou les systèmes pairs-à-pairs d'échange de fichiers (Napster, Kazaa, eDonkey), et pour les services, les transactions bancaires et autres possibilités d'achats en ligne.
Alors que le succès de certaines de ces applications ou services n'était pas attendu, la tendance actuelle pour les acteurs d' internet est d'essayer d'en créer de nouveaux qui permettront d'attirer de nouveaux utilisateurs. On peut par exemple constater qu' actuellement, des télévisions ou des radios tentent de diffuser leur programme sur internet, pour conquérir des parts de marchés et augmenter leur profits. Il existe aussi des communautés d'internautes, qui souhaitent profiter de la proximité et de l'impression de liberté que crée internet avec le public pour développer et diffuser de nouvelles chaînes de télévisions ou de radios.
Ces applications nécessitent des communications de groupe puisqu'elles ont vocation à atteindre un très large public (de plusieurs centaines à plusieurs milliers voire dizaines de milliers de récepteurs actuellement). De plus, elles nécessitent également le transport via internet de flot multimédias que sont le son ou la vidéo. Cependant, à l'inverse des applications pairs-à-pairs de transferts de fichiers qui permettent de télécharger des contenus multimédia mais de ne les consulter qu'une fois le téléchargement terminé, ces applications nécessitent que les flots soient décodés et lus au fur et à mesure qu'on les reçoive. On appelle ce type d'application des applications de streaming en direct.

Le décodage et la lecture simultanés des flots multimédias en direct présentent des contraintes de temps fortes puisqu'il est nécessaire de respecter les instants de lecture pour que l'utilisateur disposent d'un vrai confort d'utilisation. En effet, la réception de paquets du flot après leur instant de lecture engendrera une mauvaise qualité de réception pour l'utilisateur (image saccadée, ralentie...). Le paradoxe à vouloir déployer des applications de streaming sur internet est que le réseau internet est un réseau qui assure un service dit de 'moindre effort', donc qui ne donne aucune garantie sur les délais de livraisons des paquets et les débits dont on pourra disposer. Or les applications de streaming multimédias sont des applications qui consomment beaucoup de bande passante mais surtout ce sont des applications qui nécessitent que des délais de transmission soient garantis, ce qu' internet ne peut assurer puisqu'aucun mécanisme permettant un certain niveau qualité de service n'y est implémenté.

Pourtant, des applications à contraintes de temps fortes comme des applications de téléphonie sur internet ont déjà été déployées, fonctionnent de manière tout à fait satisfaisantes et rencontrent un succès grandissant auprès des utilisateurs d'internet (Skype[46]). C'est d'autant plus remarquable que ce type d'application (téléphonie sur IP) présentent des contraintes de temps encore plus fortes que les applications de streaming puisqu'elles doivent gérer l'interactivité entre l'émetteur et le récepteur. Et, même si il est encore trop tôt pour l'affirmer complètement, on peut presque estimer qu'elles contribuent à attirer de nouveaux utilisateurs vers internet donc à accélérer son développement.
C'est pourquoi, bien qu'aucun mécanisme assurant un certain niveau de qualité de service n'existe sur internet, la viabilité des applications de streaming est tout à fait envisageable sur le réseau Internet, surtout si on les conçoit en utilisant les mécanismes décrits précédemment que sont la construction d'un overlay pour permettre la communication de groupe entre les hôtes, et un réseau pairs-à-pairs pour permettre aux hôtes de partager leurs ressources et d'utiliser au mieux celles du réseau afin qu'ils atteignent le même but commun : la diffusion efficace de flots multimédias en direct (streaming).

Travaux de Stage

Au cours de ce stage de fin d'étude, mes travaux de recherche se sont focalisés sur les protocoles permettant la diffusion de flots multimédias en direct via un réseau pairs-à-pairs construit sur l'overlay. Ces travaux m'ont permis de constater que les protocoles pouvaient utiliser plusieurs approches différentes pour permettre la diffusion des flots à savoir des approches orientés-sources, des approches orientées-récepteurs et des approches orientées-données.

Bien que les topologies construites dans l'overlay par ces protocoles soient similaires (arbre de diffusion, maillage, etc...), leur construction et leur utilisation diffèrent suivant l'approche utilisée. Les protocoles mettant en oeuvre une approche orientée-source construisent leur overlay en fonction de la source du flot, tandis que les protocoles mettant en oeuvre une approche orientée récepteur construisent l'overlay en fonction du récepteur du flot. Enfin, les protocoles mettant en oeuvre les approches orientées données construisent l'overlay en fonction de la disponibilité des données chez les autres pairs du réseau. Par exemple, si la topologie de l'overlay est un arbre de diffusion, la racine de l'arbre sera la source du flot pour une approche orientée source mais elle sera le récepteur du flot dans une approche orientée récepteur. Nous étudierons plus précisément ces types d'approches et certains de ces protocoles dans la seconde partie de ce rapport. Après avoir étudié en détails ce type de protocoles et les différentes approches qui les caractérisent, j'ai souhaité comparer certains de ces protocoles afin de savoir quelles approches seraient les plus efficaces pour la diffusion de flots en direct sur internet.

Tout d'abord, pour pouvoir comparer différents protocoles, on peut convenir de certaines métriques de comparaisons comme les métriques de stress ou de stretch déjà évoquées, mais également du degrée des noeuds dans le réseau over-

lay, du débit possible que l'on peut atteindre dans le réseau, du taux de pertes de paquets ou du temps moyen de réception du premier paquet du flot après l'entrée dans le réseau par les récepteurs (*time to first packet*). Ensuite, pour effectivement comparer les protocoles, plusieurs techniques sont possibles. On peut choisir de les déployer à grandes échelles sur internet et d'effectuer des mesures réelles de leur comportement. On peut également modéliser le comportement des protocoles via des modèles mathématiques et calculer leur comportement à travers ces modèles. On peut aussi déployer des prototypes de protocoles sur des réseaux de type internet comme PlanetLab et effectuer des mesures de certaines métriques. Enfin, on peut simuler le comportement des protocoles grâce à des simulateurs. Cependant, le déploiement à grande échelle des protocoles sur internet est une technique beaucoup trop coûteuse (en temps) à mettre en oeuvre uniquement pour comparer différentes approches des protocoles, et la possibilité de pouvoir effectuer des mesures réelles n'est pas garantie, comme par exemple dans le cas où le protocole conçu ne passerait pas à l'échelle d'internet. De même, le prototypage sur des réseaux de type internet simplifie le problème précédent, mais reste une solution démesurée pour effectuer seulement des comparaisons entre les différentes approches. La modélisation est une technique qui peut s'avérer très efficace si tant est que l'on puisse trouver une manière de modéliser ces protocoles. Or la complexité des protocoles est parfois telle que l'on obtient des modèles que l'on ne peut résoudre simplement (*NP-complet*). Dans le cadre de mes travaux, j'ai donc choisi de simuler les protocoles parce que c'est une technique simple et efficace pour avoir une idée précise du comportement des protocoles pour pouvoir ensuite les comparer. Cependant, la simulation des protocoles n'est possible que si l'on dispose d'un simulateur qui permet de simuler le comportement des protocoles suivant certains contextes et certaines métriques. En l'occurrence, je souhaitais pouvoir simuler des réseaux de grandes tailles (plusieurs centaines à plusieurs milliers de clients) en me limitant à la structure de la topologie de l'overlay et en faisant abstraction des protocoles des couches basses d'internet. Après avoir étudié les simulateurs existants qui sont présentés dans la troisième partie de ce rapport, je suis arrivé à la conclusion qu'aucun ne me permettait de faire assez simplement les mesures et comparaisons que je souhaitais effectuer pour les protocoles de streaming en direct sur internet. J'ai donc implémenté un nouveau simulateur et j'ai porté dessus déjà deux protocoles ayant des approches pour la construction de l'overlay très différentes : l'un utilise l'approche orientée-source (*PeerCast*[1]),l'autre utilise l'approche orientée données (*Donet*[16]). Je n'ai pas retenu l'approche orientée récepteur, car comme nous le verrons dans la seconde partie, elle me parait moins intéressante à évaluer car trop intimement liée au codage des données. Le fonctionnement du simulateur implémenté est présenté dans la quatrième partie de ce rapport, ainsi que la façon dont sont simulés les protocoles. Enfin, la dernière partie de ce rapport présente les premiers résultats obtenus grâce aux simulations. Ils tendent à montrer que les protocoles utilisant une approche orientée-données sont plus efficaces pour la diffusion de flots multimédia en direct que ceux utilisant une approche orientée-source car l'approche orienté-données permet une plus grande robustesse à l'importante dynamicité des membres du réseau over-

lay, donc elle occasionne moins de perturbations pour l'utilisateur et permet ainsi une meilleure qualité de réception du flot.

A ma connaissance, ces travaux sont les premiers à comparer les protocoles entre eux selon l'approche qu'ils utilisent, et via une même plateforme de simulation. Ainsi, on peut observer les comportements des différents protocoles en fonction du même scénario et des même paramètres de simulation, c'est à dire que l'on compare les protocoles en les faisant évoluer sur une même topologie, avec le même nombre de clients dans le réseau et la même dynamicité du réseau. L'utilisation d'une même plateforme de simulation sert donc de dénominateur commun aux simulations pour tous les protocoles et permet ainsi de réellement les comparer.

Pour mes futurs travaux, j'adapterais dans un premier temps les paramètres de configuration du simulateur et des protocoles à des configurations plus proches de celles rencontrées dans internet afin d'affiner les résultats de mes simulations. Je poursuivrais également l'implémentation du simulateur afin d'y ajouter les fonctionnalités lui permettant de réaliser tous types de scénarios. Le simulateur, une fois arrivée à maturité, sera un outils capable de mettre en évidence les imperfections des protocoles existant afin de les améliorer, et sera d'une aide précieuse pour me permettre de concevoir un protocole efficace pour le streaming en direct via un réseau pairs-à-pairs

Chapitre 2

Protocoles de Diffusion de Flots Multimédias en Direct via un Réseau Pairs-à-pairs

Les applications de streaming de contenus multimédias sont des applications ayant vocation à atteindre un grand nombre de récepteurs, donc nécessitant la mise en place des fonctionnalités de communication de groupe (multicast). Pour atteindre cet objectif, les protocoles regroupent les membres du groupe multicast en un overlay dans lequel un réseau pairs-à-pairs est constitué entre les membres du groupe. Les différents protocoles peuvent chacun créer des topologies dans l'overlay radicalement différentes comme des arbres de diffusion entre les hôtes, ou un maillage. Toutefois, c'est principalement l'approche qu'ils utilisent qui les différencie et que l'on a déjà évoqué à savoir les approches orientés sources, récepteurs ou bien données.

2.1 Protocoles orientées-source

Concrètement, quelque soit la topologie constituée dans le réseau overlay, on distingue généralement un plan de contrôle et un plan de données. Le plan de données est utilisé pour la transmission des données à tous les hôtes pairs membres du groupe, et le plan de contrôle contient les fonctionnalités nécessaires à la gestion du groupe et à la dynamicité de celui-ci (adhésion, départ explicite ou non). Généralement, les plans de données découlent des plans de contrôle qui sont établis pour gérer les membres du groupe. Ainsi, dans le cas du protocole Peercast[1], le plan de contrôle est un arbre dont la racine est la source du flot, et le plan de données utilise le même arbre que celui du plan de contrôle. Pour ce qui est du protocole Narada[2], le plan de contrôle est un maillage entre les noeuds de l'overlay, et le plan de données est un arbre de diffusion constitué au dessus de ce maillage, et dont la racine est la source du flot. Il existe d'autres structures

de plan de contrôle comme celle du protocole Nice[3] qui définit une organisation hiérarchique des noeuds en cluster, et sur lequel le plan de données est encore un arbre de diffusion dont la racine est toujours la source du flot. Bien que dans ces trois exemples la structure du plan de contrôle soit différente, le plan de données est toujours un arbre de diffusion centré à la source, et c'est ce qui caractérise les approches orientés-sources ([5, 4]). En fait, les approches orientés sources n'ont fait que réinjecter au niveau applicatif les mécanismes développés pour les communications de groupe au niveau réseau (multicast réseau), en construisant un arbre de diffusion des données sur l'overlay via un plan de contrôle.

2.2 Protocoles orientées-données

A l'inverse, les approches orientés-données ne marquent pas clairement la différence entre leur plan de contrôle et leur plan de données. En effet, les informations de contrôle que s'échangent les membres du groupe concernent la disponibilité des données dans le réseau. Ainsi, chaque membre du groupe choisit lui même les membres de son voisinage en fonction des données qu'il souhaite obtenir. Il n'y a pas à proprement parlé de structures de plan de contrôle ou de données réellement constituées dans l'overlay car ces protocoles utilisent les algorithmes épidémiques([23]) (ou gossip). Ce type d'approche est apparu suite au succès des protocoles pairs-à-pairs comme BitTorrent([17]). BitTorrent est un protocole pairs-à-pairs de diffusion de fichiers volumineux dans lequel les pairs entrent en relation en fonction des données dont ils disposent. Les mécanismes de BitTorrent reposent donc sur la diffusion des données que les pairs possèdent.

De la même manière que l'introduction du concept de pairs-à-pairs, ou bien de certains modèles de mobilité dans les réseaux, semblent découler de l'observation par l'homme de modèles déjà présents dans d'autres sciences ou dans la nature, comme par exemple la vie en communauté des fourmis ou des abeilles pour le pairs-à-pairs ou les poursuites d'individus (modèle de poursuite), les déplacements de population nomade (modèle de nomade) ou de particules dans des fluides (modèle de marche aléatoire, inventé par Einstein pour expliquer le mouvement brownien) ([40]), les algorithmes épidémiques doivent leur origine à l'observation par l'homme des phénomènes d'épidémie de virus comme ceux de la peste ou de la grippe. En effet, la diffusion des virus d'une personne à l'autre se fait aléatoirement, de proche en proche, et converge vers une diffusion totale de la maladie pour créer une épidémie où toute une population est effectivement atteinte. De la même manière, les algorithmes épidémiques ont le fonctionnement suivant : une entité souhaitant transmettre un message vers toutes les autres entités le propagent vers des destinations aléatoirement choisies qui sont alors 'infectées' par le message. A leur tour, elles transmettent les informations à d'autres entités aléatoirement choisies et ainsi de suite jusqu'à ce que toutes les destinations soient atteintes. Au final, ce procédé permet bien au message d'être transmis à tous les utilisateurs malgré l'absence d'infrastructures clairement définies dans l'overlay. On peut ainsi noter que dans le cadre des communications de groupe, une des grandes difficultés à surmonter lorsque

les fonctionnalités de ce type de communication sont implémentées en bordure du réseau (c'est à dire au niveau applicatif chez les hôtes), est la dynamicité des membres du groupe. En effet, il est très compliqué de vouloir structurer l'overlay en une certaine topologie si les noeuds de l'overlay sont très dynamiques, car les départs et arrivées des noeuds peuvent parfois influer sur tout le réseau. C'est pourquoi, l'approche orientée-données qui fonctionne sans réelle structure dans l'overlay, pourrait s'avérer moins sensible à la dynamicité des membres du réseau que l'approche orientée-source, et pourrait donc être l'approche la plus adéquate à utiliser pour les protocoles de diffusions de flot multimédias en direct. En effet, ce type d'approche pourrait limiter les effets d'une des principales difficultés que ces types de protocoles ont à maîtriser, c'est à dire la gestion de la dynamicité du groupe. L'autre principale difficulté restante étant la garantie stricte des délais de réception qu'internet n'est pas censé pouvoir assurer puisqu'il ne fournit qu'un service *'best effort '*.

2.3 Protocoles orientés-récepteur

La dernière approche que l'on peut distinguer est l'approche dite orientée-récepteurs ([12, 6, 13]). Dans ce type d'approche, bien que la structure de la topologie peut également être un arbre, un cluster ou un maillage comme c'est le cas dans l'approche orientée-source, le plan de données de la structure n'est pas centré sur la source du flot mais sur le récepteur. C'est lui qui organise les ressources (membres pairs du réseau) pour assurer la réception du flot dans de bonnes conditions. Cette approche est très souvent liée au codage utilisé pour le flot de données comme le codage en couches (*layered coding*) ou à multiples descriptions (*MDC*). Succinctement, le codage en couches est un codage où la donnée est codée en différentes parties. Il est alors nécessaire de recevoir au moins la couche principale des paquets du flot pour le décoder et le lire, la réception des autres couches permettra alors d'améliorer la qualité de réception. Le codage à multiples descriptions est similaire, mais il permet de ne pas avoir à recevoir une couche en particulier. Chacune permet d'obtenir une certaine qualité de réception qui augmente en fonction du nombre de couches reçues. Ce type d'approche présente l'avantage de permettre à tous les noeuds du réseau d'avoir leur ressources utilisées même si leurs capacités d'émissions sont limités, à la différence des autres approches qui pourraient ne jamais solliciter les ressources d'un noeud jugés trop faibles et solliciter toujours les noeuds à grandes capacités. En effet, même un noeud à faibles capacités pourrait permettre de diffuser les couches de faibles débits (pour le codage en couche) ou bien un seul niveau de couches (pour le MDC). Cependant, il présente –par nature– l'inconvénient d'être trop lié aux techniques de codages des paquets du flot. De plus, ce type d'approche autorise le fait que la qualité de réception pour l'utilisateur ne soit pas parfaite puisque l'on pourrait se contenter uniquement de certaines couches en réception. C'est à dire que c'est une solution qui conçoit -de par sa nature- que le résultat ne puisse être parfait. Or, mes travaux de recherche ainsi que mon utilisation d'applications implémentant des protocoles de diffusion de flot

multimédias en direct sur internet (CoolStreaming[49], Pplive[50]) m'ont permis de constater que la conception d'une solution efficace est envisageable et qu'internet dispose de suffisamment de ressources pour y parvenir ([16]). Ainsi, je n'ai pour l'instant pas jugé nécessaire d'inclure des protocoles utilisant ce type d'approche (orientée-récepteurs) dans mes travaux de simulation, bien qu'ils gardent toutes ma considération car leurs conceptions ont également contribué à faire évoluer la recherche concernant les protocoles de diffusion de flots multimédia en direct et les différentes techniques de codage de l'information.

2.4 Exemples de protocoles

Après avoir exposé les concepts généraux des protocoles de diffusion de flot multimédias en direct et la manière dont il convient de les classer selon leur approche, certains de ces protocoles vont maintenant être présentés en détails, notamment ceux m'ont paru représentatifs de leur catégorie, et surtout ceux dont j'ai éprouvé le besoin de simuler le comportement afin de comparer les différentes approches, et savoir laquelle était la plus adaptée à ce type de communication.

2.4.1 PeerCast

PeerCast[1] est un protocole de diffusion de flots multimédias en direct d'une source vers plusieurs clients. Pour la diffusion du flot, il construit dans un réseau overlay un arbre multicast spécifique à la source dans lequel chaque noeud est client du flot. L'arbre de l'overlay est maintenu et contrôlé à l'aide d'un réseau pairs-à-pairs (P2P) qui permet aux applications réceptrices du flot de faire abstraction de la topologie réelle ainsi que de la dynamicité de l'overlay (adhésions ou départs de clients).

Chaque noeud du réseau overlay joue le rôle d'un routeur multicast : ils répliquent et transmettent les paquets à leurs enfants.

2.4.1.1 Architecture

L'architecture de PeerCast repose sur un réseau P2P qui permet de masquer la topologie du réseau à l'application. Ce réseau P2P est considéré par PeerCast comme une couche de *peering* qui se situe entre la couche transport et la couche applicative. Cette couche permet de contrôler l'arbre de diffusion, ainsi que les arrivées et les départs des noeuds. PeerCast utilise cette couche de *peering* pour séparer les sessions de transferts de données des sessions d'applications.

Les sessions de transferts de données sont établies entre les couches de *peering* de deux noeud et sont composées du canal de données et du canal de contrôle d'un *stream*. Les sessions d'applications commencent quand un client souhaite recevoir un *stream*. Les sessions d'applications sont établies entre la couche de peering et l'application.

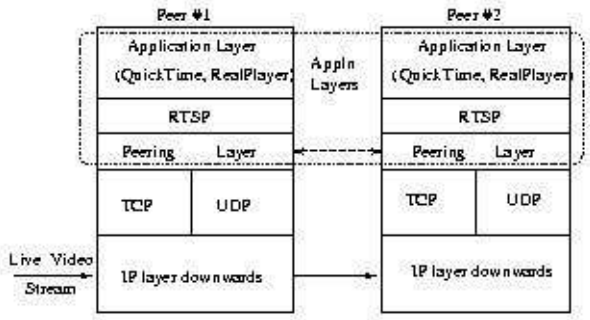

Figure 2.1: Architecture en couches d'un pair

La couche de *peering* sert d'interface à l'application pour lui abstraire tout le réseau et les mécanismes nécessaires à la diffusion du flot. Toutes les communications entre la couche transport et la couche application passent donc par la couche de *peering* rendue par le réseau P2P.

Le réseau P2P est responsable de la localisation de la source et de l'établissement de la session de transfert de données. Quand une session de transfert de données se termine alors que le client souhaite toujours recevoir le flot, c'est la couche de *peering* (le réseau P2P) qui localise une nouvelle source de données et rétablit une session de transfert de données.

La fonction principale de la couche de *peering* utilisée par PeerCast est la primitive de redirection. PeerCast l'utilise comme moyen simple pour maintenir la connectivité des noeuds dans l'overlay. Son fonctionnement est le suivant : Un noeud p envoie un message de redirection à un noeud c qui est soit en train d'ouvrir une session de transfert de données, soit déjà connecté. Le message spécifie un noeud cible t. Le noeud c ferme alors ses connections en cours vers p et essaye d'établir une session de transfert de données avec le noeud cible t (pour le même flot).

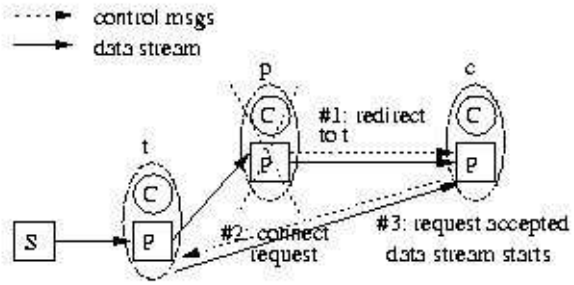

Figure 2.2: Un exemple de redirection quand p quitte le réseau

Un nouveau client qui souhaite recevoir le flot doit être capable de joindre le groupe multicast. PeerCast suppose que pour chaque flot, il existe une URL unique qui permet de retrouver la source du flot.

L'arbre multicast est construit à mesure que des noeuds souhaitent recevoir le flot. Les nouveaux noeuds sont incorporés dans l'arbre et servis par des noeuds qui reçoivent déjà ce flot. Cependant, chaque noeud à des capacités limitées. Un noeud qui n'est pas capable de servir un client avec une qualité acceptable pour le flot sera considéré comme saturé. PeerCast contient donc un mécanisme permettant à un nouveau noeud d'identifier un noeud non saturé et de s'y rattacher. Ce mécanisme utilise principalement la fonction de redirection :

Pour recevoir le flot, un nouveau noeud contacte la source (qu'il connaît via l'URL). La source est la racine de l'arbre multicast du réseau overlay. Si elle n'est pas saturée, elle accepte le noeud comme son fils et la session de transfert des données peut commencer. Sinon, la source redirige le noeud vers un de ses fils. Ensuite, soit le fils n'est pas saturé et il accepte le noeud, soit il est saturé, et redirige le noeud vers un de ses fils. Ce procédé est répété jusqu'à ce que le nouveau noeud trouve un noeud qui l'accepte. Si le noeud n'est pas capable de trouver un noeud non saturé au bout d'un certain nombre de tentatives, la couche de *peering* renvoie à l'application un message d'erreur indiquant qu'aucune ressource n'est disponible.

Un noeud de l'arbre multicast peut souhaiter quitter le réseau, ou être victime de pannes ou autres défaillances. Ces comportements entraînent une partition de l'arbre multicast à laquelle le réseau overlay doit faire face, afin que la diffusion du flot ne soit pas interrompue pour les autres noeuds. Quand un noeud quitte explicitement le réseau, il envoie à tous ses enfants un message de redirection immédiate vers une autre source de flot.

Pour détecter les pannes et autres défaillances, PeerCast utilise un mécan-

isme de *heartbeat* : concrètement, par l'intermédiaire de sa couche de *peering*, un noeud envoie périodiquement des messages à ses parents et enfants pour leur confirmer sa présence. Au bout d'un certain temps sans message, les parents considèrent que leur noeud enfant est mort, clôturent leurs sessions et libèrent leurs ressources ; les enfants quant à eux doivent se retrouver une nouvelle source de flot.

Les différentes techniques pour retrouver une source de flot suite à un départ explicite ou à une défaillance sont explicitées dans la partie suivante.

2.4.1.2 Politiques de gestion de la topologie

Dans PeerCast, la diffusion du *flot* multimédia vers plusieurs clients repose sur la création d'un arbre multicast dans le réseau overlay, et tous les noeuds de l'arbre sont des clients du flot.

Schématiquement, un noeud rejoint l'arbre de diffusion en contactant la source puis en étant redirigé vers un noeud capable de le servir grâce à la primitive de redirection. Quand un noeud quitte l'arbre explicitement, il transmet un message de redirection à ses enfants. S'il quitte l'arbre à le suite d'une défaillance, ses enfants s'en rendent compte au bout d'un certain temps (ils ne détectent plus les messages de *heartbeat*) et recherchent une nouvelle source de flot.

La topologie de l'overlay étant en arbre, chaque noeud n'a qu'une connaissance locale de la topologie : chaque noeud ne connaît que son père ou ses enfants directs. Tous les noeuds connaissent la source, puisque c'est par elle que les noeuds entrent dans le réseau, via l'URL supposée connue de tous par PeerCast.

Ainsi, lors des redirections, deux cas se présentent :
-soit il s'agit d'une redirection d'un noeud qui adhère au flot
-soit il s'agit d'une redirection d'un noeud suite au départ explicite ou à une défaillance de son père

2.4.1.3 Adhésion d'un noeud à l'arbre de diffusion

Un noeud saturé qui ne peut accepter de connexions peut rediriger le noeud seulement vers ses enfants. En effet, la demande de connexion lui est retransmise par son père, ce qui signifie que ni la source, ni aucun des parents ne pouvaient satisfaire la connexion.

Pour choisir vers lequel de ses enfants rediriger le nouveau noeud, trois politiques de redirection sont possibles :

a°) choisir aléatoirement le noeud enfant cible

b°) utiliser un mécanisme de Round-Robin pour choisir le noeud enfant cible

c°) rediriger le client vers l'enfant le plus proche (délai minimum entre le client et lenfant)

2.4.1.4 Départ d'un noeud du groupe

Quand un noeud quitte le réseau explicitement, il ne peut rediriger ses enfants que vers la source du flot ou vers son propre père. Quatre politiques de redirection sont alors possibles :

a°) rediriger tous les enfants vers la source : le message de redirection est transmis récursivement à tous les enfants qui vont tous recommencer la procédure d'adhésion par la source.

b°) rediriger tous les enfants vers le père du noeud qui quitte l'arbre. De la même manière que précédemment, un message de redirection est transmis récursivement, et tous les noeuds recommenceront la procédure d'adhésion à partir de ce noeud.

c°) rediriger uniquement les enfants directs du noeud vers la source. Les descendants restent alors accrochés à leur branche de l'arbre et sont dépendants des enfants directs du noeud.

d°) rediriger uniquement les enfants directs du noeud vers le père du noeud qui quitte l'arbre.

Dans le cas du départ d'un noeud suite à une défaillance, seules les politiques qui redirigent vers la source sont possibles puisque le père du noeud défaillant ne peut pas être connu par les enfants puisque aucun message de redirection ne leur a été envoyé.

D'après les études réalisées par les concepteurs de PeerCast, il semblerait qu'il soit plus judicieux de rediriger les client vers l'enfant le plus proche (délai minimal) lors d'une procédure d'adhésion, et de rediriger seulement les enfants directs d'un noeud vers le père du noeud en cas de départ explicite du réseau overlay, ou vers la source du flot en cas de défaillance du noeud.

2.4.1.5 Conclusion

PeerCast est un protocole de diffusion de streaming multimédia en direct via un réseau P2P. PeerCast diffuse le flot aux clients via un arbre multicast spécifique à la source construit dans un réseau overlay. Le réseau P2P sous-jacent sert de plan de contrôle à l'élaboration de cet arbre.

Le fait que la source du flot soit la racine de l'arbre de l'overlay, et qu'elle soit également l'élément central de tout le mécanisme du protocole (puisque tous les noeuds la contactent pour recevoir le flot) nous permet bien de classer ce protocole comme un protocole *orienté-source*.

2.4.2 Narada

La construction d'un arbre de diffusion multicast dans le réseau overlay est la manière la plus évidente de transmettre un flot multimédia d'une source vers plusieurs récepteurs puisque c'est la réutilisation directe des mécanismes développés pour le multicast de niveau réseau (niveau IP).

Un problème avec cette approche est qu'elle n'est pas robuste aux défaillances des noeuds de l'arbre : une défaillance d'un noeud entraîne la partition de l'arbre, et les noeuds en aval du noeud défaillant ne peuvent plus recevoir le flot jusqu'à ce que des mécanismes coûteux en ressources ne soient déployés pour les raccrocher à l'arbre.

Un autre défaut de cette approche est que si le groupe contient plusieurs sources de diffusion, alors toutes les opérations de gestion du groupe seront répliquées dans chaque arbre, ce qui diminuera fortement les performances du réseau overlay à cause de l'*overhead* engendré pour le contrôle.

Une solution pour améliorer la robustesse de la structure et la fiabilité des transmissions est d'augmenter le nombre de chemins possibles entre la source et les destinations. Ainisi, en cas de défaillance d'un noeud, d'autres chemins alternatifs seront disponibles pour transmettre les messages (contrôles ou données).

De plus, plutôt que de répliquer les mêmes informations de contrôle du groupe dans chaque arbre, il parait judicieux de séparer la transmission des messages de contrôle de la transmission des données. Ainsi, les informations de contrôle du groupe pourraient être utilisées par chaque source sans avoir à les répliquer dans tous les arbres de diffusion.

Narada[2] est un protocole conçu dans cet optique : il sépare le plan de contrôle du plan de données. Ainsi, le plan de contrôle sera construit via un maillage (*mesh*) qui est une structure robuste et fiable du fait de la grande connectivité entre les noeuds. Le plan de données sera quant à lui construit via un arbre de diffusion multicast, lui-même reposant sur le maillage : les arbres de livraison des données seront construits entièrement sur les liens de l'overlay présent dans le maillage. Les informations de gestion du groupe n'auront plus à être répliquées dans chaque arbre, puisque tous les arbres utiliseront le plan de contrôle sous-jacent offert par ce maillage.

2.4.2.1 Gestion du groupe

Pour atteindre un haut niveau de robustesse, chaque membre du groupe maintient une liste de tous les autres membres. Les listes des membres doivent être mises à jour quand un nouveau membre rejoint le groupe ou qu'un membre le quitte.

Chaque membre du groupe génère périodiquement des messages de mise à jour qui contiennent un numéro de séquence incrémenté à chaque nouveau message. Ces messages sont seulement échangés entre voisins. Chaque membre

maintient pour les autres membres les informations suivantes : adresse du membre, dernier numéro de séquence émis par le membre, date à laquelle le message de mise à jour du membre a été reçu. En plus, chaque membre échange périodiquement sa connaissance du groupe avec ses voisins. Ainsi, tous les membres auront une connaissance totale du groupe au bout de quelques itérations.

Lorsqu'un membre n'a pas reçu de mise à jour d'un autre membre pendant un certain temps, il considère que le membre est 'défaillant' ou qu'il y a une partition dans le groupe et déclenche les mécanismes de réparation du maillage (*mesh*).

2.4.2.2 Adhésion au groupe

Lorsqu'un membre souhaite rejoindre le groupe, Narada suppose que le membre est déjà capable d'obtenir une liste de membres du groupe. Narada considère que ce problème n'est pas du ressort du protocole, mais de l'application et de l'utilisateur. Le nouveau membre envoie aléatoirement des messages aux membres qu'il connaît pour qu'ils l'acceptent comme voisin. Il répète ce procédé jusqu'à ce qu'il obtienne une réponse positive de certains des membres. Après avoir rejoint le groupe, le nouveau membre échange des messages de mise à jour avec ses voisins. Les voisins diffuseront leur connaissance du nouveau noeud à leurs voisins : ainsi, la présence du nouveau membre sera propagée à tout le groupe, et il prendra également connaissance de tous les autres membres.

2.4.2.3 Départ ou défaillance d'un membre

Quand un membre quitte le groupe, il l'annonce explicitement à ses voisins et l'information sera propagée de voisins en voisins à tous les autres membres du groupe.

Quand un membre quitte le groupe suite à une défaillance, ses voisins le détectent puisqu'ils ne reçoivent plus de messages de mise à jour. Chacun des voisins essayent alors d'interroger le membre défaillant et si il ne répond pas, les voisins considèrent qu'il est définitivement 'défaillant' et propagent l'information à travers le maillage.

La défaillance d'un membre peut entraîner la partition du maillage (cf: 2.3. avec A qui disparaît)

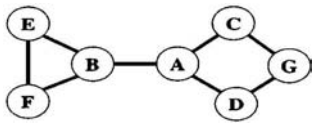

Figure 2.3: Exemple de topologie d'overlay

Dans un cas pareil, les membres détectent la partition car ils ne reçoivent plus aucun message de mise à jour de toute une partie des membres du groupe (ceux qui se situent de l'autre côté de la partition).

Les membres réparent la partition de la façon suivante : chaque membre maintient une liste des membres dont il a arrêté de recevoir des messages de mise à jour. Périodiquement, il efface le premier membre de la liste et le teste. Si le membre répond, un lien est rajouté entre les deux membres, sinon le membre testé est considéré comme 'défaillant' et cette information est propagée au sein du groupe.

2.4.2.4 Performances du maillage

Le maillage construit n'est pas optimale car la sélection initiale des voisins par un nouveau membre est aléatoire donc elle ne tient pas compte de la topologie sous-jacente et de son évolution. De plus, les mécanismes de réparation des partitions peuvent ajouter des liens nécessaires sur le moment mais inutiles quand le maillage sera stabilisé. Enfin, la composition du groupe peut changer très rapidement en raison des différents départs et arrivés de membres dans le réseau.

Par l'intermédiaire d'une fonction testant l'utilité des liens, Narada permet d'optimiser la qualité du maillage en ajoutant ou en supprimant des liens. Pour ajouter un lien, chaque membre teste périodiquement certains des membres non voisins aléatoirement, et évalue l'utilité d'ajouter un nouveau lien.

Pour supprimer un lien, chaque membre teste ses voisins et décide si il doit conserver ou non le lien vers ce voisin.

Les informations seront ensuite propagées dans le maillage via les messages de mise à jour périodiquement transmis.

2.4.2.5 Transmission des données

Narada se sert d'un protocole de vecteur de distance au dessus de la mesh pour router les messages. Pour éviter le problème du coût infini des chemins, chaque membre maintient les chemins complets ainsi que leur coût vers tous les autres membres du groupe. Les messages de mises à jour des tables de routage contiennent à la fois le coût vers la destination et le chemin complet qui permet un tel coût.

Les arbres par source utilisés pour la livraison des données sont construits par '*reverse shortest path*' entre les récepteurs et la source, de la même façon que DVMRP :

Un membre M qui reçoit un paquet de la source S par un voisin V, transmet le paquet seulement si le voisin V est le prochain saut sur le plus court chemin du membre M vers la source S. Le membre M transmet également le paquet à tous ses voisins qui l'utilise comme prochain saut pour atteindre la source.

Cependant, les départs des membres du groupe ou la suppression de liens pour des raisons de performances peuvent entraîner des pertes de paquets, dues à la non convergence des tables de routage de tous les récepteurs.

Pour éviter cela Narada introduit un nouveau coût de routage appelé '*Transmission Passagère*' (*Transient Forward : TF)*. Ce coût est plus élevé que le coût d'une route valide mais n'est pas infini. Un membre qui quitte le groupe annonce un coût de *TF* à tous les membres pour qui il a une route valide. L'opération normale des algorithmes à vecteur de distance fait choisir aux membres une route valide alternative ne comprenant pas le membre qui quitte le groupe puisque le coût du chemin passant par lui a fortement augmenté avec l'annonce du *TF*. Ce mécanisme force donc les autres membres à changer immédiatement leurs chemins pour ne plus utiliser le membre qui quitte le groupe.

2.4.2.6 Conclusion

Narada est un protocole de diffusions de streaming multimédia d'une source vers plusieurs récepteurs. A la différence de PeerCast[1], Narada ne se contente pas de seulement réutiliser les mécanismes du multicast de niveau réseau: Narada associe la robustesse d'une structure maillée pour la gestion du contrôle de l'overlay à l'efficacité d'une structure en arbre pour la transmission des données.

Tout comme PeerCast, Narada construit un arbre de diffusion multicast sur l'overlay pour transmettre les données. D'ailleurs, la source du flot est la racine de l'arbre, et c'est elle qui nécessite la mise en place de la structure maillée qui par sa robustesse, permet d'atteindre un certain niveau de fiabilité lors des transmissions des données. Cest pourquoi on considère que Narada est un protocole *orienté-source.*

2.4.3 Nice

L'objectif de Nice[3] est de construire de manière distribuée, efficace et scalable un arbre de diffusion multicast dans le réseau overlay. Nice organise hiérarchiquement les noeuds de l'overlay en clusters répartis sur plusieurs niveaux. L'opération de base du protocole est de créer et maintenir cette hiérarchie. Cette hiérarchisation des noeuds permet de créer une topologie de contrôle, qui définit implicitement le chemin de transmission des données sur l'overlay.

Logiquement, chaque membre garde un état des autres membres qui sont près de lui dans la hiérarchie, et a des connaissances limitées sur les autres membres de la hiérarchie.

Le protocole Nice utilise le délai de bout en bout entre les noeuds comme métrique de distance. Pour construire la hiérarchie de Nice, les membres qui sont proches selon cette métrique se retrouvent dans la même partie de la hiérarchie.

2.4.3.1 Arrangement Hiérarchique des membres

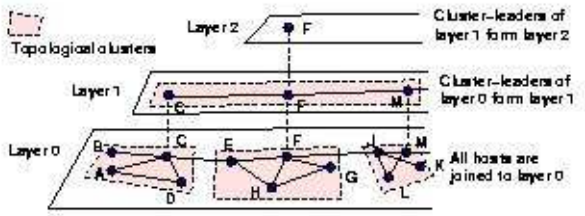

Figure 2.4: Structure hiérarchique des membres dans NICE

La hiérarchie de Nice est créée en affectant des membres à différents niveaux (cf. FIG.2.4)

Les membres de chaque niveau sont partitionnés en clusters. Chaque cluster consiste en un ensemble de membres qui sont proches les uns des autres selon la métrique de distance définie précédemment (délai de bout en bout). De plus, le noeud le plus proche du centre du cluster sera considéré comme le leader du cluster. Le choix du leader du cluster est important car il permet aux nouveaux membres de trouver rapidement leur position dans la hiérarchie.

Les membres sont affectés aux différents niveaux de la façon suivante :
-tous les membres font partie du plus bas niveau (niveau 0)
-les membres de chacun des niveaux sont partitionnés en clusters
-Les leaders de tous les clusters d'un niveau rejoignent le niveau supérieur

Chaque membre conserve l'état de tous les clusters auxquels il appartient et du cluster supérieur auquel appartient son leader (son super-cluster).

Exemple: *(figure 4 avec k=3)*

Les clusters du niveau L0 sont [ABCD], [EFGH] et [JKLM].

Les leaders des clusters de L0 sont C, F et M. Ils forment le niveau 1 et sont regroupés pour créer l'unique cluster du niveau 1 [CFM].

F est le leader de ce cluster et par conséquent il appartient également au niveau 2.

2.4.3.2 Chemins de contrôle et de données.

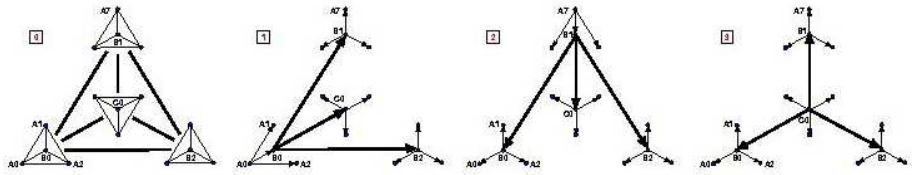

Figure 2.5: Plans de contrôle et de données pour une structure hiérarchique a 2 niveaux. les hôtes A_i sont seulement membres du cluster de niveau 0. les hôtes $B_i$sont membres des niveaux 0 et 1. L'hôte C est le leader du cluster de niveau 1 regroupant les hôtes $B_i$et lui même.

Les voisins sur la topologie de contrôle échangent périodiquement des messages de mise à jour(FIG. 2.5) .

Les liens indiquent les relations de peering entre les membres de l'overlay.

Les clusters du niveau 0 sont composés de 4 membres dont B0, B1, B2 et C0 sont les leaders. C0 est le leader du cluster du niveau 1.

Panel 0 :

Dans la topologie de contrôle, les pairs d'un membre sont les autres membres des clusters de tous les niveaux auxquels il appartient. Par exemple, A0 appartient seulement au niveau 0 et ses pairs de la topologie de contrôle sont les autres membres de son cluster de niveau 0 c'est à dire A1, A2 et B0. Par contre, B0 appartient au niveau 0 et 1 donc ses pairs de la topologie de contrôle sont les autres membres de son cluster de niveau 0 (A0, A1 et A2) et de son cluster de niveau 1 (B1, B2, C0).

Dans cette topologie de contrôle, chaque membre d'un cluster échange des messages de rafraîchissement avec tous les autres membres de son cluster. Ainsi les membres du cluster pourront identifier rapidement les arrivés et les départs dans le cluster.

Dans Nice, le chemin de livraison des données est un arbre dont la racine est la source du flux et est implicitement définie par la topologie de contrôle.

Panel 1 :

A0 envoie les données à tous les membres de son cluster (niveau 0). Le leader de son cluster B0 transmet les données aux membres de son cluster de niveau 1. Les membres de ce cluster de niveaux 1 sont leaders de clusters de niveaux 0 : ils peuvent donc transmettre les données aux membres des autres cluster de niveaux 0.

Les Panels 2 et 3 illustrent le même procédé avec A7 et C0 comme source (respectivement).

2.4.3.3 Invariants du Protocole

Pour que la hiérarchie soit maintenue et que les propriétés décrites précédemment restent valides, le protocole doit maintenir les invariants suivants :

-à chaque niveau, les membres sont regroupés en clusters dont la taille est comprise entre k et 3k+1.

-tous les membres appartiennent au niveau 0, et chaque membre appartient à un seul cluster par niveau.

-les leaders des clusters sont les centres de leur cluster respectif et appartiennent également au niveau immédiatement supérieur.

2.4.3.4 Description du protocole

Nice suppose l'existence d'un point de rendez-vous *RP* que chaque membre devra contacter pour rejoindre le groupe multicast. Ce point de Rendez-vous est toujours le leader de l'unique cluster de plus haut niveau de la hiérarchie.

Arrivée de nouveaux membres

Figure 2.6: Exemple d'adhésion au groupe.A12 rejoint le groupe multicast

Quand un nouveau membre rejoint le groupe multicast, il doit être affecté à un cluster de niveau 0 (*Fig 6*).

A12 veut rejoindre le groupe multicast :

1. Il contacte *RP* (Panel 0)

2. *RP* lui indique les membres qui sont présents dans le plus haut niveau de la hiérarchie (ici, C0 dans le niveau 2)

3. A12 contacte tous les membres que lui a transmis *RP* pour identifier le plus proche de lui. Dans cet exemple il n'y a que C0, donc C0 est forcément le plus proche.

4. C0 informe A12 qu'il y a 3 autres membres B0, B1 et B2 dans le cluster de niveau 1 (Panel 1)

5. A12 contacte chacun des membres pour identifier le membre le plus proche. (Panel 2)

6. A12 recommence itérativement pour trouver le membre le plus proche de lui au niveau 0 et se joindre à un cluster.

Il faut remarquer que envoyer des requêtes à chaque couche, du plus haut niveau hiérarchique au plus bas, permet de trouver le membre du cluster de niveau 0 le plus proche du membre qui rejoint le groupe.

Comme le leader d'un cluster doit être au centre du cluster, l'arrivée d'un nouveau membre peut modifier le centre du cluster donc entraîner un changement de leader. Le noeud qui perd alors son leadership se retire des couches hiérarchiquement supérieures. Un nouveau leader est choisi pour chacun des clusters affectés par ces modifications. Chaque nouveau leader rejoint le cluster de niveau supérieur (qui était son super-cluster avant qu'il ne devienne leader).

Gestion et optimisation du Cluster

Chaque membre d'un cluster envoie périodiquement un message à tous les pairs de son cluster. Le message contient la distance estimée (délai) par le membre vers tous les autres membres de son cluster.

Le leader du cluster envoie périodiquement l'état de l'adhésion au cluster à tous les membres afin de leur permettre de connaître les arrivée et départs de membres, et de mettre en place de nouvelles relations de peering si nécessaire. Le leader leur envoie également l'état des membres de son cluster de niveau supérieur (leur super-cluster).

Les membres d'un cluster considèrent qu'un membre ne fait plus partie du cluster quand ils n'ont pas reçu de messages de sa part depuis un certain temps.

Les leaders vérifient également périodiquement la taille de leur cluster. Si les clusters sont trop grands, alors ils seront divisés, s'ils sont trop petits ils seront fusionnés avec d'autres clusters. Ces divisions et fusions entraînent nécessairement des changements de leader, puisque les centres des cluster sont modifiés.

Un membre peut rejoindre un mauvais cluster à cause d'éventuelles pertes de messages ou de congestion dans le réseau. C'est pourquoi chaque membre teste périodiquement les membres de son super-cluster (le cluster de niveau supérieur dont son leader fait partie) et vérifie quel membre du super-cluster est le plus proche de lui. Si le membre du super-cluster le plus proche de lui est bien son leader alors il se trouve dans le bon cluster. Sinon, le membre quitte le cluster où

il était attaché et rejoint le cluster dont le leader est le membre du super-cluster qui est le plus proche de lui.

Départ des noeuds et sélection des leaders

Quand un membre quitte le groupe multicast, il envoie un message de départ explicite à tous les clusters auxquels il appartient. Quand un membre est victime d'une défaillance, les pairs de son cluster le détectent en ne recevant plus de message de sa part. Si c'est le leader qui quitte le cluster, les membres du cluster déterminent un nouveau leader.

2.4.3.5 Conclusion

Nice est un protocole qui utilise un système hiérarchique pour le contrôle de l'overlay, et utilise un arbre de diffusion pour transmettre les données. La source du streaming multimédia est alors la racine de l'arbre.

Le protocole Nice, qui construit un système hiérarchique pour le contrôle de l'overlay et un arbre pour transmettre les données, est relativement proche du protocole Narada, qui construit un arbre pour transmettre les données mais contrôle l'overlay via une mesh, dans la mesure où ils séparent les plan de contrôle et de données pour limiter les messages de contrôle qui circulent sur l'overlay.

Plusieurs sources peuvent alors émettre dans l'overlay sans répliquer les informations de contrôle au sein de chaque arbre de diffusion.

Pour les même raisons que Narada, à la structure du plan de contrôle près, Nice est une approche *orientée-source*.

ZigZag

Une optimisation du protocole Nice est le protocole ZigZag[4].

La principale différence entre ces deux protocoles est que Nice utilise toujours le leader du cluster pour transmettre les données aux autres membres alors que ZigZag utilise un leader étranger : un leader d'un autre cluster.

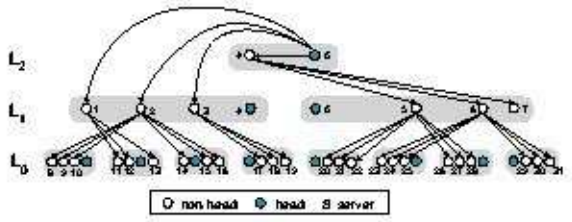

Figure 2.7: ZigZag : Tous les membres non leader des clusters doivent recevoir le contenu de leur leader étranger.

Les auteurs de ZigZag procèdent ainsi car ils démontrent que ZigZag permet d'avoir des noeuds de degrés moins importants dans le pire des cas que Nice. Cela signifie que les noeuds -avec ZigZag- auront moins de connections vers d'autres noeuds, donc qu'ils seront moins vite saturés.

Autrement dit avec Nice, il y aura plus rapidement des noeuds saturés qui gêneront la diffusion du stream multimédia pour les autres noeuds.

ZigZag semble donc apporter des améliorations significatives à Nice pour des applications de streaming multimédia.

2.4.4 Donet

DONET[16] est un protocole conçu pour permettre la diffusion de streaming multimédia en direct d'une source vers plusieurs récepteurs via la construction dun réseau overlay.

Le fonctionnement de DONET peut schématiquement se réduire ainsi : tous les noeuds du réseau échangent périodiquement des informations sur les données dont ils disposent à un ensemble de partenaires. Les noeuds peuvent ainsi récupérer des données, fournir des données ou les deux.

La seule exception est la source du *streaming* qui est toujours un fournisseur de données. La source peut aussi bien être un serveur dédié qu'un noeud du réseau, et est considérée comme le *noeud d'origine*.

2.4.4.1 Arrivée des noeuds et gestion de l'adhésion au réseau

Chaque noeud du réseau a un identifiant unique (par exemple l'adresse IP) et conserve en mémoire une liste partielle des membres actifs du réseau (membership Cache : *mCache*).

Un nouveau noeud souhaitant se joindre au réseau contacte le noeud d'origine. Le noeud d'origine sélectionne aléatoirement de sa *mCache* un *noeud député* vers lequel il va rediriger le nouveau noeud. Le *noeud député* transmet au nouveau noeud une liste de candidats qu'il a aussi choisi aléatoirement dans sa *mCache*.

Le nouveau noeud contacte les candidats pour établir des partenariats dans le réseau. Une fois que des partenariats sont établis entre le nouveau noeud et les candidats sélectionnés par le *noeud député*, le nouveau noeud fait partie du réseau. Il échangera alors avec ses partenaires des informations relatives aux données qu'il possède comme nous le verront ultérieurement.

Afin que le réseau s'accommode à la dynamicité des noeuds, il est nécessaire que la liste des membres dont dispose chaque noeud (*mCache*) soit régulièrement mise à jour. Ainsi, chaque noeud génère périodiquement un message d'adhésion pour annoncer son existence. Les messages d'adhésion contiennent : le numéro de séquence du message, l'identifiant du noeud, le nombre de partenaires du noeud et la durée restante de validité du message.

Les messages d'adhésion sont seulement envoyés à certains noeuds de la *mCache* choisis aléatoirement.

Quand un noeud reçoit un message d'adhésion d'un partenaire, il décrémente la durée de validité du message de la durée qui s'est écoulée depuis la dernière mise à jour. Si le résultat est inférieur ou égale à 0, alors le message n'est plus valable et l'entrée du partenaire est effacée de la *mCache*. Si le résultat est strictement supérieur à zéro, soit le partenaire est nouveau et une nouvelle entrée est créée dans la *mCache*, soit il existe déjà une entrée pour ce partenaire. Le noeud vérifie alors le numéro de séquence du message. Si c'est le même numéro de séquence, alors le message a déjà été reçu et le noeud ne met pas à jour sa *mCache*. Si le numéro de séquence est nouveau (supérieur à l'ancien), le noeud met à jour l'entrée du partenaire dans sa *mCache*.

Une entrée de *mCache* contient les mêmes champs que le message d'adhésion avec en plus la date locale de la dernière mise à jour de l'entrée pour permettre ensuite de tester la validité du message.

La mise à jour de l'entrée de la *mCache* du noeud entraîne la retransmission du message d'adhésion vers d'autres partenaires choisis aléatoirement dans la *mCache* et ainsi de suite.

Ainsi, l'envoi et la retransmission des messages d'adhésion aléatoirement vers les partenaires entraînent une augmentation de la connaissance des membres du réseau. Ainsi, les noeuds auront à leur disposition dans leur *mCache* plus de noeuds vers qui créer des partenariats pour recevoir les données. La figure 2.8 montrent un exemple de partenariats dans le réseau DONET.

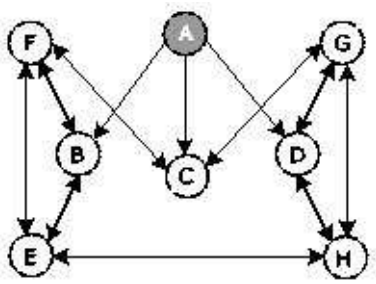

Figure 2.8: Illustration d'un partenariat dans Donet (A est le noeud d'origine).

2.4.4.2 Représentation des données

Les données du stream sont divisées en segments de même longueur. Chaque noeud utilise un vecteur de bits pour représenter les segments qu'ils possèdent en mémoire. Les bits à '1' du vecteur indiquent les segments que le noeud possède, et à les bits à '0' indiquent ceux qu'ils ne possèdent pas.

Ainsi, chaque noeud échange son vecteur de segments à ses partenaires. Le choix des segments que le noeud doit récupérer ou envoyer se fera via un algorithme d'ordonnancement.

La taille du vecteur de segments n'a pas à être trop importante puisque DONET est conçu pour le streaming en direct, donc les segments trop anciens n'ont pas besoin d'être retrouvés.

2.4.4.3 Algorithme d'ordonnancement

En fonction des données dont le noeud dispose et des vecteurs de segments de ses partenaires, l'ordonnanceur permet de choisir les données à récupérer chez les partenaires.

L'ordonnanceur doit prendre en considération deux contraintes : la date de lecture de chaque segment, et les capacités hétérogènes des partenaires. il doit également veiller à ce que le maximum de paquets arrivent avant leur instant de lecture afin de garantir une fluidité dans le lecture du stream.

Le choix judicieux des segments en fonction de leur instant de lecture et des capacités des partenaires est un problème NP-complet. L'algorithme d'ordonnancement est donc construit à l'aide d'un simple heuristique.

Tout d'abord, l'ordonnanceur du noeud calcule, en fonction des vecteurs de segments envoyés par les partenaires, le nombre de fournisseurs potentiels de chaque segment désiré. Puis, l'ordonnanceur classe les segments à récupérer en premier dans l'ordre inverse du nombre de leurs fournisseurs. Ainsi, moins un segment dispose de fournisseurs, plus il sera récupéré prioritairement. Lorsque

un segment a plusieurs fournisseurs, l'ordonnanceur choisit comme fournisseur le partenaire ayant les plus grandes capacités (bande passante).

Pour récupérer les données chez le partenaire choisi, le noeud envoie au partenaire une séquence de vecteur de segments et le partenaire lui envoie les segments correspondants.

On constate ainsi que les données sont récupérées avec DONET de la même manière qu'avec *bitTorrent*[17], en utilisant une politique d'ordonnancement '*rarest first*'.

2.4.4.4 Départ des noeuds et partenariats

Dans un réseau dynamique, les noeuds peuvent partir soit de leur plein gré, soit parce qu'ils sont victimes de pannes ou de défaillances.

Lorsqu'ils partent de plein gré, les noeuds transmettent un message de départ qui a le même format qu'un message d'adhésion, mais dont le nombre de partenaire est fixé à -1. Ce message est transmis aléatoirement aux partenaires du noeud, de la même façon que les messages d'adhésion. Les partenaires qui reçoivent le message effacent le noeud de leur *mCache*. Ainsi, petit à petit, tous les noeuds ont connaissance du départ du noeud et l'effacent de leur *mCache*.

S'il s'agit d'une défaillance, un partenaire qui la détecte envoie un message de départ à la place du noeud défaillant. Ce message sera transmis aux autres noeuds de la même manière que les messages d'adhésion, et les noeuds qui le reçoivent effaceront l'entrée du noeud défaillant de leur mCache.

Cependant, il se peut que plusieurs partenaires aient détecté la défaillance et que plusieurs messages de départ aient été générés. Seul le premier message sera retransmis, et les autres messages de départ reçus seront ignorés.

Périodiquement, chaque noeud établit de nouveaux partenariats avec des noeuds choisis aléatoirement dans sa mCache. Ce procédé permet d'atteindre deux objectif : premièrement cela permet à chaque noeud de maintenir un nombre stable de partenaires dans le cas de départ des noeuds. Deuxièmement, cela permet à chaque noeud de trouver des partenaires de meilleure qualité : c'est à dire des partenaires qui pourront transmettre le plus de segments possibles rapidement. En effet, le noeud examine des noeuds de sa mCache et peut décider de remplacer un de ses partenaires par un noeud de meilleure qualité.

2.4.4.5 Conclusion

DONET crée une 'mesh' dont la topologie fluctue en fonction de la livraison des données entre les partenaires. En effet, un lien (de l'overlay) peut exister entre deux noeuds qui échangent des donnés à un certain instant, puis ne plus exister l'instant suivant car ils ne s'échangent plus de données.

Cependant, les deux noeuds restent partenaires potentiels l'un pour l'autre, chacun possédant l'autre en *mCache*. Comme les noeuds continuent de récupérer des données chez leurs autres partenaires, les données dont ils disposent évoluent ; leur vecteur de segments respectif leur seront éventuellement retransmis, et un nouveau partenariat entre les noeuds pourrait alors à nouveau s'engager si l'un des noeuds en éprouve le besoin : c'est à dire si un des noeuds souhaite récupérer des segments que possèdent l'autre noeud.

Dans cette approche c'est véritablement la disponibilité de la donnée qui crée instantanément la topologie de livraison des données, c'est pourquoi ce protocole est dit *orienté-données*.

Chapter 3

Simulateurs

3.1 La Simulation

Une simulation est une tentative de modéliser un système dans le but de l'étudier scientifiquement. Si les relations dans le modèle (la complexité du modèle) sont assez simples, alors une solution analytique peut être obtenue par des méthodes mathématiques. Cependant la complexité du système réel empêche souvent cela, et alors une simulation est requise pour pouvoir estimer les vraies caractéristiques du système.

Les systèmes pairs-à-pairs comprennent des caractéristiques complexes à modéliser dont il faut rigoureusement tenir compte telles que le nombre de pairs, leur dynamicité, les nombreux comportements qu'ils peuvent présenter ou encore les différentes localisations des contenus.

Actuellement, certaines approches analytiques sont appliquées aux systèmes pairs-à-pairs mais elles requièrent souvent beaucoup d'hypothèses simplificatrices qui ne permettent pas de modéliser les conditions réelles du système. C'est notamment le cas concernant le nombre de pairs que l'on souhaite simuler pour que le système passe à l'échelle.

Pourtant, la nature même d'une simulation est de faire certaines suppositions et abstractions sur le système à simuler. En effet, si cela n'était pas le cas, on pourrait alors tout aussi bien construire le système réel et le déployer afin d'étudier directement son comportement. Or, si le système construit ne satisfaisait pas les conditions de passages à l'échelle, sa conception aura été vaine puisque il ne pourra être utilisé à l'échelle d'internet, ce qui est généralement le but recherché par les systèmes pairs-à-pairs. C'est d'ailleurs ainsi que beaucoup de systèmes pairs-à-pairs ont été conçus par le passé, et c'est pourquoi certains d'entre eux on eu une durée de vie limitée : à cause de leur mauvaise aptitude à passer à l'échelle. C'est ainsi le cas pour le protocole Gnutella ([51]), dont le mécanisme de recherche de contenu entraînait un 'flooding' dans le réseau qui empêchait son passage à l'échelle d'internet. Toutefois, des améliorations ont été ajoutées à ce système afin qu'il puisse être déployé à l'échelle d'internet.

Une compréhension totale du système à simuler est ainsi nécessaire puisque la vraie question à se poser lorsque l'on souhaite simuler un système est de savoir quelles suppositions, abstractions ou hypothèses simplificatrices l'on pourrait faire pour que d'une part le système ne soit pas trop complexe à simuler, et d'autre part pour que les simulations rendent bien compte du comportement qu'aurait eu le système en réalité. Ainsi, on doit veiller à ne pas oublier de paramètres à implémenter dans le simulateurs, ou à ne pas utiliser d'hypothèses trop simplificatrices qui altéreraient dramatiquement le comportement du réseau simulé par rapport à son comportement réel.

De plus, souvent les simulations construisent un modèle et utilisent les résultats d'une unique simulation pour obtenir une réponse (un comportement du système). En fait, Les résultats produits par la plupart des simulateurs seront fortement affectés par les distributions de probabilités des paramètres d'entrée. Spécifiquement aux simulations de système pairs-à-pairs on trouve la topologie de départ, la distributions des contenus et des requêtes parmi les pairs, ou encore le taux de dynamicité des pairs. Pour qu'une simulation soit utilisée et puisse apporter des conclusions valides et crédibles, les différentes suppositions quant aux distributions des paramètres de départ doivent être précautionneusement déterminées.

Une bonne hypothèse pour entrer de bons paramètres de distribution aux simulateurs est d'entrer des valeurs que l'on a mesurer dans l'étude de cas réels. Cependant, si l'on prend le cas d'une topologie à passer au simulateur, cette topologie est représentative de l'état du système à un instant donné mais manque toutefois de particularités spécifiques au développement dynamique du réseau. Autrement dit, même si un paramètre passé en entrée du simulateur semble adéquat, on doit garder à l'esprit que le système entier évoluera au cours de la simulation, et que d'autres distributions -judicieusement choisies- de paramètres seront nécessaire pour permettre de simuler de vrais comportements du réseau, comme par exemple les distribution qui caractérisent le comportement des pairs (pannes, départs, etc...). De la même façon que pour la topologie, déterminer les comportements des pairs via des observations réelles de systèmes pairs-à-pairs semblent être le meilleur moyen pour éviter les trop grandes différences entre le système simulé et le système réel. Cependant, si l'on ne dispose d' aucune information réelle pour déterminer ces distributions, il est alors nécessaire de s'en approcher approximativement en utilisant par exemple des distributions de probabilités qui viennent d'études de systèmes non pairs-à-pairs mais qui présentent des similitudes avec le système pairs-à-pairs que l'on souhaite simuler.

L'utilisation de distributions de probabilités comme paramètres d'entrées au simulateur permet de modéliser les différents comportements des entités du système. Or, à chaque exécution de la simulation, une génération des probabilités différente aura lieu et modifiera radicalement la configuration du système (Dans mon cas, malgré la même topologie et le même nombre de clients, ils ne seront pas attachés sur les mêmes noeuds, la source sera différente, donc le rôle des clients dans la topologie de l'overlay sera différente).

C'est pourquoi, on ne doit pas effectuer une unique exécution de la même simulation pour en déduire un comportement global du réseau car cela corre-

spondrait à un cas particulier de toutes les distributions de probabilités, mais l'on doit effectuer plusieurs exécutions de la même simulation (parfois un grand nombre), afin que les résultats obtenus tendent réellement vers le comportement global du système qui tient effectivement compte des paramètres d'entrée. A l'inverse, on peut éventuellement avoir besoin de fixer certains paramètres d'entrées du simulateur si l'on souhaite étudier des comportements dans un cas très précis, par exemple dans le pire des cas possibles.

Voyons un exemple très simple pour illustrer la nécessité d'effectuer plusieurs exécutions des simulations : le cas d'un tirage à pile ou face. La raison nous permet d'affirmer que les simulations permettront d'obtenir des valeurs de pile pour la moitié des exécutions, et face pour l'autre moitié. Or avec une seule exécution nous n'obtiendrons que l'une ou l'autre des possibilités donc soit pile, soit face. Ainsi, conclure après une unique exécution, que le modèle tendrait à toujours donner par exemple pile serait dénué de tout bon sens et absolument faux.

Ainsi, lorsque l'on souhaite procéder à des des simulations, il convient d'effectuer de nombreuses exécutions de la même simulation et de présenter les résultats en indiquant la variation des valeurs des paramètres de sortie des simulations.

3.2 Etat de l'art des simulateurs de réseau pairs-à-pairs

La plupart des simulateurs de réseau pairs-à-pairs existants permettent de simuler des systèmes d'échange de fichiers. Ainsi, le simulateur *QueryCycle([30])* simule le comportement d'applications pairs-à-pairs de partage de fichier reposant sur un réseau semblable à Gnutella. Il se concentre sur la simulation des requêtes envoyées par les pairs sur le réseau mais ne simule pas le temps sur le système.

SIMP^2([28]) quant à lui est conçu pour fournir un support à une analyse des réseaux pairs-à-pairs ad hoc. L'analyse est limitée à étudier certaines propriétés du réseau tel que le degré des noeuds ou leur faculté à être joignables par les autres noeuds. Il permet également de simuler le comportement des files d'attentes et des pertes de messages. SIMP^2 ne tient pas compte de la dynamicité des noeuds et des autres aspects dynamiques du réseau.

Le simulateur Neurogrid ([25]) est un simulateur conçu à l'origine pour comparer les mécanismes de recherches de contenus des protocoles Freenet, Gnutella et Neurogrid. Cependant, il a également été conçu dans l'optique de ne pas être trop spécifique à un système pairs-à-pairs, et pour être réutilisable et adaptable à d'autres protocoles. Son implémentation se fait via l'utilisation de classes abstraites qui sont destinées à être génériques aux différents système pairs-à-pairs. Ainsi, l'on y trouve des classes abstraites concernant le réseau, les noeuds ou encore les messages. Chacune comprend des méthodes par défaut qui peuvent être étendues ou modifiées afin de simuler le comportement d'autres système pairs-à-pairs. Le moteur de simulation est à évènement discret associé à différentes estampilles temporelles. Chaque évènement engendre d'autres évènements à

des estampilles temporelles futures (supérieures), mais le temps et les délais du réseau ne sont pas réellement simuler.

Le simulateur 3LS([26]) est un simulateur qui décompose le système pairs-à-pairs en 3 niveaux : le niveau réseau, le niveau du protocole et le niveau utilisateur. Ce simulateur transforme la topologie du réseau sous-jacent en une matrice de délais entre les différents noeuds et permet de simuler les délais dans le réseau.

Narses([29]) est un simulateur de réseau au niveau des flots, conçu pour éviter l'overhead des simulateurs de niveau paquet comme c'est le cas pour le simulateur NS-2([35]). Ses développeurs partent du principe que la taille des files d'évènements des simulateurs de niveau paquet peuvent avoir de sérieux impacts sur les performances du simulateur donc du réseau simulé. Il ne simule donc pas le niveau paquet du réseau, et supposent qu'il n'y a pas de goulot d'étranglement dans le coeur du réseau donc qu'il n'y a pas besoin de simuler les routeurs intermédiaires. Cette approche permet d'économiser des ressources de calculs qui pourront alors être utilisées pour simuler des réseaux beaucoup plus grands que ceux simulés avec des simulateurs de niveau paquets. Il permet donc de simuler des réseaux dont la taille passerait à l'échelle d'internet. Toutefois, puisqu'il fait abstraction des couches basses, il ne peut garantir que les comportements des réseaux qu'il simule sont proches de la réalité. En effet, certains simulateurs comme PLP2P ([27]) utilisent l'approche inverse et considèrent que le niveau paquets doit être pris en compte car il pourrait avoir des influences significatives sur le comportement du réseau.

Souvent, les protocoles pairs-à-pairs disposent d'un simulateur de réseau qui leur est dédié (par exemple Aurora([33]) et Serapis ([1]) pour Freenet([31]), FreePastry([32]) pour Pastry ([9])). Le problème général d'avoir un simulateur dédié est que les résultats obtenus avec ce type de simulateurs sont difficiles a valider et souvent impossible a obtenir à nouveau avec un simulateur différent, en raison de toutes les suppositions faites par leurs développeurs et de leur implémentation.

Bien qu'une tendance semble se dessiner pour convenir à une conception de simulateurs génériques permettant de simuler différents réseaux, il n'existe actuellement aucune solution suffisamment avancé que l'on aurait pu utiliser pour les types de protocoles que l'on souhaitait simuler c'est à dire des protocoles de diffusion de flots multimédia en direct via un réseau pairs-à-pairs (utilisant différentes approches).

La plupart des simulateurs présentés ne permettent pas de simuler le niveau flot, ou ne tiennent pas comptes des délais dans le réseau. Ils sont généralement conçus pour simuler le comportement de réseaux pairs-à-pairs utilisant des applications d'échanges de fichier, donc n'ayant pas de contraintes de temps fortes, ce qui les rendaient inutilisables dans le cadre de ces travaux puisque on souhaitait simuler des réseaux utilisant des applications de streaming donc avec de fortes contraintes temporelles.

De plus, chaque simulateur étant la plupart du temps implémenté indépendamment des autres, et en fonction du type de réseau à simuler, cela les rend

très complexes à utiliser. Il est ainsi parfois plus rapide de développer un nouveau simulateur pour qu'il soit bien adapté à l'utilisation qu'on veut en faire, plutôt que d'en utiliser un existant difficilement paramétrables et configurables.

Chapitre 4

Implémentation du Simulateur

Le simulateur que j'ai implémenté, bien qu'adapté aux protocoles de streaming pairs-à-pairs, emploie certains mécanismes utilisés par les simulateurs de réseaux pairs-à-pairs présentés précédemment. Par exemple, mon simulateur utilise pour l'instant les mêmes suppositions que Narses en ce qui concerne le coeur du réseau (aucun goulot d'étranglement dans le coeur du réseau), utilise le même niveau réseau que 3LS (topologie des noeuds du réseau transformée en matrice de délais), et utilise un moteur de simulation comparable à celui de Neurogrid (les évènements génèrent d'autres évènements à des estampilles temporelles futures).

4.1 Architecture

Comme ce type de protocole a pour vocation de diffuser le flot à un grand nombre de récepteurs, et que je souhaite simuler leur comportement à l'échelle d'Internet, il fallait que je puisse simuler des réseaux comprenant plusieurs plusieurs milliers de clients. j'ai ainsi choisi d'implémenter le simulateur en utilisant le C-ANSI comme langage de programmation. En effet, ce langage compilé d'un niveau assez proche du système d'exploitation me permettait de gérer de manière très fine les ressources mémoires afin d'initialiser le plus de clients possibles, et il me garantissait également des temps de traitements relativement courts en comparaison avec des langages de très haut niveau comme les scripts (Perl, Python) ou les langages compilés mais utilisant une machine virtuelle coûteuse en ressource et en temps de traitements (Java, C#).

L'environnement de développement pour ce simulateur est un simple PC 'standard' équipé d'un processeur PENTIUM 4 1.6GHZ avec une mémoire cache de 256ko et de 256Mo de mémoire (RAM). Le système d'exploitation est une distribution Fedora core 2 basé sur un noyau Linux 2.6.10-1. Le compilateur utilisé est GCC. L'implémentation du simulateur et du protocole respecte rigoureusement les dernières normes ANSI et POSIX. Le simulateur ainsi que les deux

protocoles simulés compte environ 7000 lignes de code.

4.2 Simulateur

Le simulateur est implémenté suivant trois niveaux : la topologie du réseau, le moteur de simulation et les protocoles. Mon objectif était de pouvoir comparer différentes approches de protocoles de diffusion de flots multimédia via un réseau pairs-à-pairs. Pour cela, j'ai choisi de simuler le comportement de deux protocoles : PeerCast (orientée-source) et Donet (orienté-donnés).

Invariants du simulateur :

- Les temps de traitements des différentes entitées ne sont pas simulés
- On considère qu'il n'y a pas de pertes de paquets au coeur du réseau
- La source est un des clients choisi aléatoirement
- La source ne connaît jamais de défaillances et ne quitte jamais le réseau .
- La source du flot est connu de tous les clients (via généralement une URL).

4.2.1 Topologie

le simulateur fonctionne avec une topologie a deux niveaux : les noeuds du réseau qui constituent la véritable topologie du réseau, et les clients qui sont aléatoirement attachés à des noeud du réseau avec un délai aléatoirement choisi.

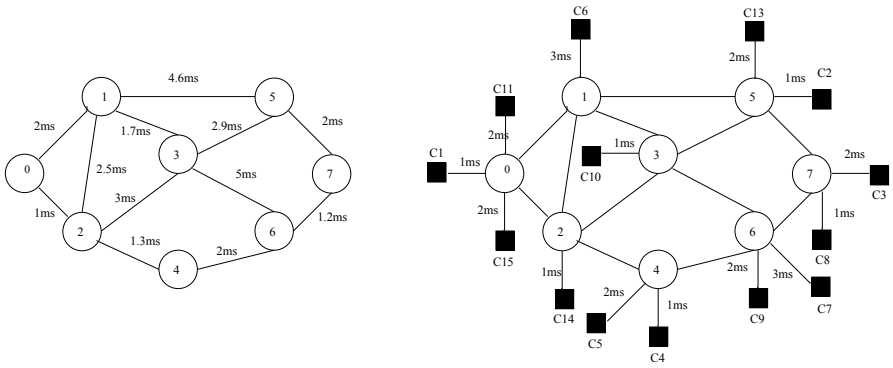

Figure 4.1: Toplogie à deux niveaux.

Concrètement, le simulateur peut reposer sur n'importe quelle type de topologie : topologie de systèmes autonomes (AS), de routeurs, ou encore de points de présence (PoP). Il suffit simplement de transformer la topologie d'entités que l'on souhaite utiliser en une matrice des délais entre chaque noeuds.

Pour calculer le délai entre deux clients, on doit simplement additionner le délai entre les noeuds d'attachement des deux clients (que l'on connaît via la matrice de délais de la topologie) et les délais de chaque clients vers leur noeud d'attachement.

Par exemple dans le figure 4.1, le client 1 (C1) est attaché au noeud 0 (0) avec un délai de 1ms et le client 6 (C6) est attaché au noeud 1 (1) avec un délai de 3ms. Le délai entre les noeuds 0 et 1 est de 2ms. Ainsi, le délai totale entre le client 1 et le client 6 est de 6ms (2+3+1).

Actuellement, j'ai généré une topologie de 1000 AS avec Brite([52]). J'ai choisi ce type de topologie afin d'effectuer mes expériences sur un réseau de type internet. J'ai choisi la granularité des AS car c'est généralement à ce niveau -entre AS- que les délais sont les plus importants, et pas au sein d'une même AS (granularité routeurs) où tout a été configuré pour que le routage soit le plus efficace possible.

Après avoir généré la topologie avec Brite, la transformation en une matrice de délai est très simple en utilisant l'algorithme de Floyd-Warshall(1).

Algorithm 1 Floyd-Warshall (W)

1 $n \leftarrow$ lignes[W]
2 $D^{(0)} \leftarrow$ W
3 pour k $\leftarrow$ 1 à n
4 faire pour i $\leftarrow$ 1 à n
5 faire pour j $\leftarrow$ 1 à n
6 faire $d_{ij}^{(k)} \leftarrow min(d_{ij}^{(k-1)}, d_{ik}^{(k)} + d_{kj}^{(k)})$
7 retourner $D^{(n)}$

Du point de vue de l'implémentation, le simulateur utilise la topologie comme une simple matrice de nombre réel (flottant) . Ainsi, une topologie de 1000 AS occupera 4Mo de mémoire. (1000*1000*4o)

4.2.2 Moteur de simulation

Le simulateur est un simulateur à évènements discrets avec des estampilles temporelles réelles. C'est à dire que chaque estampille temporelle n'est pas une valeur dénuée de sens, mais correspond à un temps calculé dans le réseau en fonction des délais dans la topologie utilisée.

Ce sont les protocoles qui alimentent le simulateur en évènements suivant leurs différents mécanismes.

D'un point de vue strict de son implémentation, le simulateur utilise principalement deux chaînages de structure :

- une liste chaînée de structure de temps(FIG. 4.2)
- une liste chaînée de structure de clients (FIG. 4.3)

Une structure de temps (FIG. 4.2) contient une estampille temporelle (valeur de temps) représentant des dates exprimées en milisecondes, un identifiant de fonctions, un pointeur vers le client qui effectue cette fonction, et bien entendu un pointeur vers la structure de temps suivante de la liste chaînée. Les structures de temps sont ordonnées selon leur estampille temporelle, dans l'ordre croissant puisqu'il s'agit de référence à un temps simulé. Chaque structure occupe une vingtaine d'octets en mémoire.

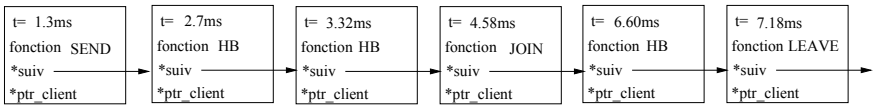

Figure 4.2: Liste chaînée de structure de temps.

Une structure de client (FIG. 4.3) contient principalement un identifiant du client (client id), un identifiant du noeud de la topologie (routeurs, AS, PoP) sur lequel est relié le client, un délai entre l'élément de la topologie et le client et un indicateur d'état du client.

Elle contient également un tableau de pointeur de clients avec lesquels le client pourra organiser ses relations de peering. Ce tableau est bien entendu vide au départ des simulations et sera rempli à mesure que la simulation s'effectue en fonction du protocole. En effet, Suivant le protocole simulé et la structure qu'ils constituent dans l'overlay, les relations de peering entre les clients seront différentes.

La structure de clients contient également une série de compteurs afin de comptabiliser le nombre de messages (de contrôle ou de données) envoyés ou reçus et un pointeur vers une structure de données qui contient l'instant de réception du premier paquet du flot par le client (time to first packet, t2fp), la date de réception du dernier paquet ainsi que son numéro de séquence.

Chaque client possède également un pointeur vers une destination où il souhaite se connecter (Ces éléments n'apparaissent pas afin de ne pas surcharger le schéma). Bien entendu, au début des simulations, pour chaque client cette destination est la source du flot. En effet, les protocoles de streaming considèrent que la source du flot est connu de tous, généralement accessible via une URL ou un mécanisme de ce type. C'est ensuite le protocole qui permettra au client de se procurer le flot parmi ses pairs.

Le fait que la source soit un client (invariant du simulateur) n'est qu'une astuce de programmation. On aurait tout aussi bien pu créer une 'structure source' et l'attachée sur n'importe quelle noeud de la topologie mais pour ne pas accumuler les structures différentes dans le simulateur, la source est donc un client dont l'état est d'être la source du flot.

Enfin, chaque structure de client contient un pointeur vers le client suivant de la liste chaînée, les clients étant ordonnée dans l'ordre croissant de leur identifiant.

Chaque structure de client (en comptant la structure de données) occupe environ 120 octets en mémoire.

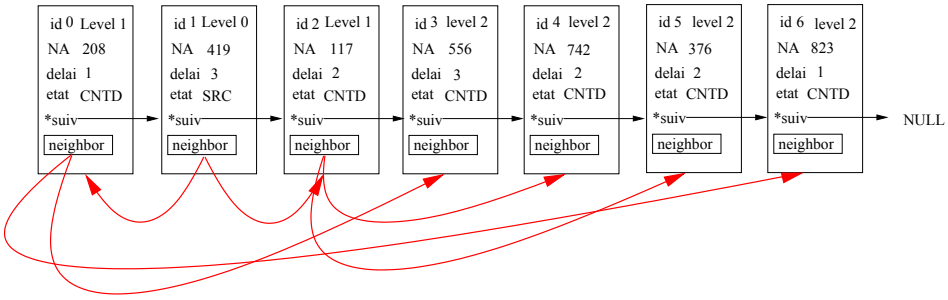

Figure 4.3: Liste chaînée de structure de clients.
Les flêches rouges représentent les relations de peering. Dans cet exemple, la structure de l'overlay est un arbre dont la source est le client 1 (id 1).

Le moteur de simulation fonctionne de la façon suivante :

La liste chaînée de temps est parcourue dans l'ordre des estampilles temporelles croisant, en traitant l'un après l'autre chaque évènement (un seul par case temporelle). Les évènement ajoutent généralement d'autres évènements à des temps futurs. On effectue une simulation jusqu'à ce qu'il n'y ait plus d'évènements à traiter (le *'pointeur suivant'* de la liste de temps est *null*), ou que l'on ait effectué la durée de simulation souhaitée. On doit noter que les évènements qui sont générés au même instant sont traités dans l'ordre de leur arrivée dans la liste d'évènement (gestion en FIFO). Cependant, cela n'affecte en rien les performances de la simulation puisqu'il s'agit de temps simulé. D'ailleurs, les évènements simultanés ne concerne pas les mêmes clients, et sont de plus très rares dans la mesure où chaque client est placé aléatoirement dans la topologie, donc chaque client a des délais vers les autres différents, ce qui fait que deux dates d'évènements sont rarement identiques.

Dans l'exemple de la figure 4.4, un scénario de l'exécution du moteur de simulation est présenté. Les clients ont établi leur relation de peering de façon à construire un arbre dont la source est le client 1 (id 1). Ses deux enfants sont les clients 0 (id 0) et 2 (id 2). Le client 3 (id 3) est un enfant du client 0. Le moteur de simulation lit l'évènement courant qui simule une action SEND par le client 1 (source du flot) à l'instant 35ms. Cet évènement engendre la création dans le liste de structure de temps d'un prochain évènement SEND de la source, 33ms plus tard donc à l'instant 68ms. Si l'on continue ainsi, le prochain évènement simulerait l'instant 37ms de simulation, où un évènement de HeartBeat (HB) serait effectué par la source. Cet évènement n'a pas encore été exécuté donc il n'a pas encore inséré dans la liste de temps le prochain évènement qu'il doit génèrer, c'est à dire le prochain évènement de HeartBeat (HB) de la source à l'instant 1037ms.

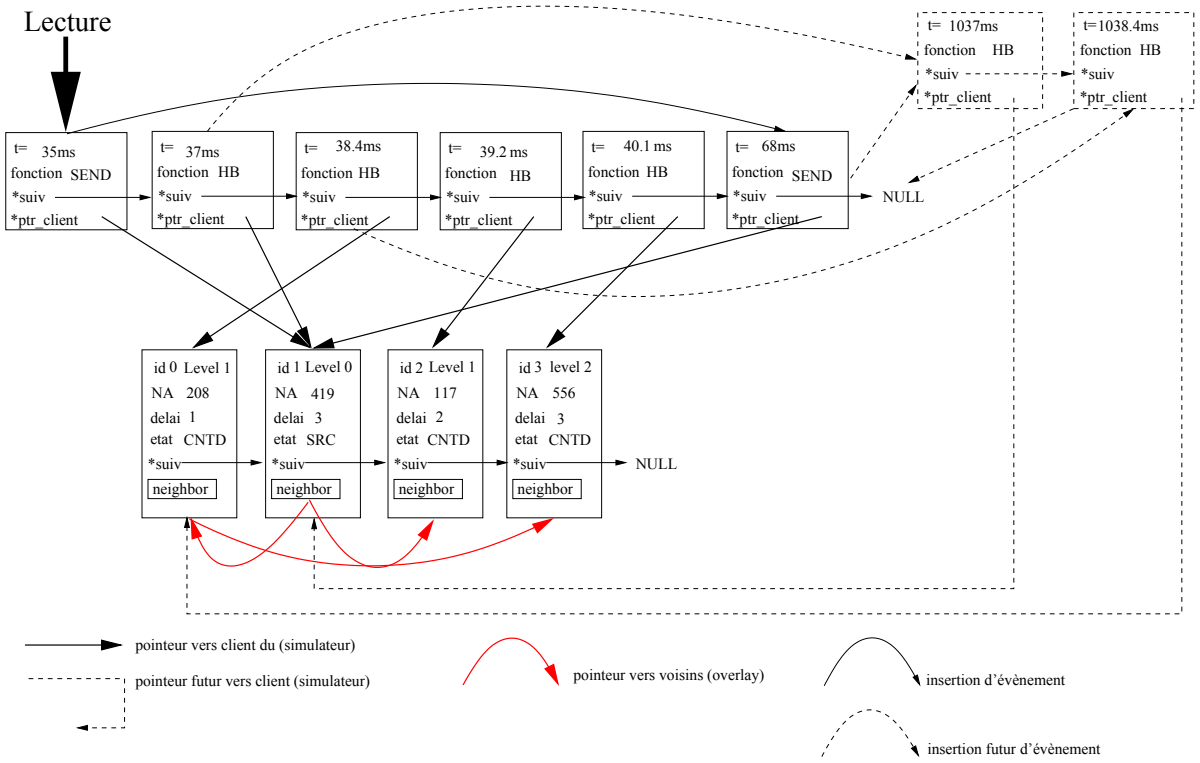

Figure 4.4: Exemple d'exécution du moteur de simulation

4.2.3 Protocoles implémentés

L'objectif de ce simulateur est de mettre en évidence les différences entre un protocole dont l'approche est orientée-source et un protocole dont l'approche est orientée-données. Pour cela, nous avons choisi d'implémenter le protocole PeerCast ([1]) qui est est un protocole orientée-source qui construit dans l'overlay un arbre de diffusion dont la racine est source du flot, et Donet([16]), qui est un protocole orientée-données donc qui ne construit pas à proprement parlé de structure dans l'overlay. Pour Donet, la structure de l'overlay est ainsi un maillage transitoire dont les liens varient en fonction des relations de peering entre les clients, lesquelles sont établies en fonction des données que les clients possèdent.

Ces protocoles ont été choisis car ils sont très représentatifs de leur catégorie (mes travaux de recherche m'ont permis de les choisir parmi plusieurs dizaines de protocoles de streaming pairs-à-pairs). De plus, ils ont soit été publiés dans de grandes conférences (Donet, IEEE Infocomm 2005), soit été très souvent cités par d'autres concepteurs de protocoles (PeerCast), et surtout, ils disposent chacun d'implémentations réelles (CoolStreaming pour Donet et PeerCast pour

Peercast) qui ont déjà été déployées sur internet et été utilisées avec succès par un grand nombre d'utilisateurs.

Bien que ces protocoles organisent les noeuds d'une façon tout à fait différente, certaines fonctions sont communes aux protocoles, même si elles n'effectuent pas les mêmes traitements. Ainsi, les protocoles disposent chacun des fonctions Join ou Leave qui permettent de rejoindre le réseau ou de le quitter, mais leur implémentation ne sera pas la même. Par exemple, dans le cadre du protocole Peercast, adhérer au réseau signifie pour un client de contacter la source et être redirigé dans l'arbre de diffusion de fils en fils jusqu'à trouver un pair capable de l'accepter, tandis que pour le protocole Donet, une fois que le client a contacté la source, il est redirigé vers un des clients députés choisi aléatoirement, qui lui transmet une liste de pairs avec qui commencer les relations de peering. Bien que les traitements ne soient pas les mêmes pour ces deux procédures, il s'agit du même évènement pour le simulateur : l'ajout d'un nouveau membre dans le réseau. C'est pourquoi le moteur de simulation ne connaît que les identifiants des fonctions, et c'est selon le protocole que l'on simule que les traitements seront différents.

Pour l'instant, les fonctions communes des protocoles pour le simulateur sont:

- *Join*, pour rejoindre le groupe
- *Leave*, pour quitter explicitement le groupe
- *Fail*, pour simuler une défaillance d'un client donc un départ non explicite du groupe
- *Hertbeat*, pour l'envoie des messages de contrôles aux pairs
- *Send*, pour l'envoie des messages de données

On peut ainsi constater que le simulateur est capable de simuler la dynamicité des clients.

Pour ce qui est des fonctions spécifiques à chaque protocole, elles sont appelées via les fonctions communes des protocoles pour le simulateur.

Ainsi, si l'on prend la fonction qui permet d'effectuer le placement intelligent ('Smart Placement') des noeud dans l'arbre du protocole PeerCast, cette fonction n'existe pas pour Donet. PeerCast a donc été implémenté de façon à ce que cette fonction ne soit pas effectuée directement par le moteur de simulation, mais via la fonction Join qui l'appelle quand elle en a besoin.

4.2.3.1 Spécifications de PeerCast

Peercast est un protocole dont l'approche est orientée-source. Il construit un arbre dans l'overlay qui sert à la fois de plan de contrôle et de plan de donnée. Comme nous l'avons vu dans la deuxième partie, son mécanisme peut pratiquement se résumer à une fonction de redirection des clients en aval dans le cas

d'une adhésion au réseau ou en amont dans le cas d'un départ explicite ou d'une défaillance.

Comme le suggère les auteurs ([1]), le mécanisme de 'smart placement' a été implémenté pour les adhésions et de 'Grand-Father' dans le cas d'un départ explicite et bien entendu 'Root' dans le cas d'une défaillance. Cela signifie que lors d'une adhésion, si un client ne peut accepter un nouvel enfant, il le redirige vers son enfant le plus proche du nouvel arrivant. Dans le cas d'un départ explicite, le client qui quitte le réseau redirige seulement ses enfants directs vers son propre père (leur grand-père). En cas de défaillance d'un client, ses pairs (parent et enfants), détectent sa défaillance au bout de deux messages de heartbeat non-reçus, lesquels sont émis toute les secondes (comme le suggère encore une fois les auteurs du protocole).

Le nombre de heartbeat à atteindre est un choix arbitraire d'implémentation (naturellement modifiable). On ne simule pas la manière dont chaque client parvient à déterminer lequel de ses enfants est le plus proche du nouvel arrivant puisque les auteurs ne donnent aucun détails sur le sujet, et ne semblent pas inquiet quant aux éventuels problèmes de passage à l'échelle de ce type de mécanisme.

La diffusion des messages de données s'effectuent de la façon suivante : dès qu'un premier client est connecté dans le réseau (donc à la source pour le premier), celle ci commence à émettre des données. A mesure que les clients terminent leur procédure d'adhésion, ils reçoivent les paquets du flot. Il est à noter que dans PeerCast, il n'y a aucune notion de mémorisation des données.

Par exemple (FIG. 4.5), supposons que chaque entité peut accepter 3 enfants et que tous les clients rejoignent le réseau à l'instant 0 de la simulation (*panel 0*).

Supposons que le premier client à terminer sa procédure d'adhésion au réseau (client 0) se situe a un délai de 5ms de la source, alors la source a reçu la requête d'adhésion à l'instant 5ms et génère le premier paquet du flot qu'elle envoie immédiatement en réponse à la requête du premier client (*panel 1*). En effet, si la réponse à une requête d'adhésion est positive, la réponse contient le premier paquet de données. Ce mécanisme est une supposition de ma part puisque le protocole ne précise rien à ce sujet mais je l'ai utilisé car c'est un comportement analogue au requête HTTP.

Supposons que le second client (client 1) se trouve à 5.5ms de la source, la source a les ressources pour accepter le second client, mais reçoit sa requête à l'instant 5.5ms. Or elle a déjà émis son premier paquet, et dans PeerCast il n'y a pas de mécanisme de mémorisation des données. La source n'envoie donc pas encore de paquet au second client (*panel 2*)

A l'instant t=10ms, le premier client reçoit bien la réponse de la source contenant le premier paquet de données. Ainsi, la métrique d'obtention du premier paquet pour le premier client sera de 10ms (time to first packet, t2fp) (*panel 3*).

A l'instant t=11ms, le second client ne reçoit pas de messages de données, mais sait qu'il a bien adhérer au réseau. (*panel 4*)

Supposons que la source émette un paquet toutes les 33ms (comme les au-

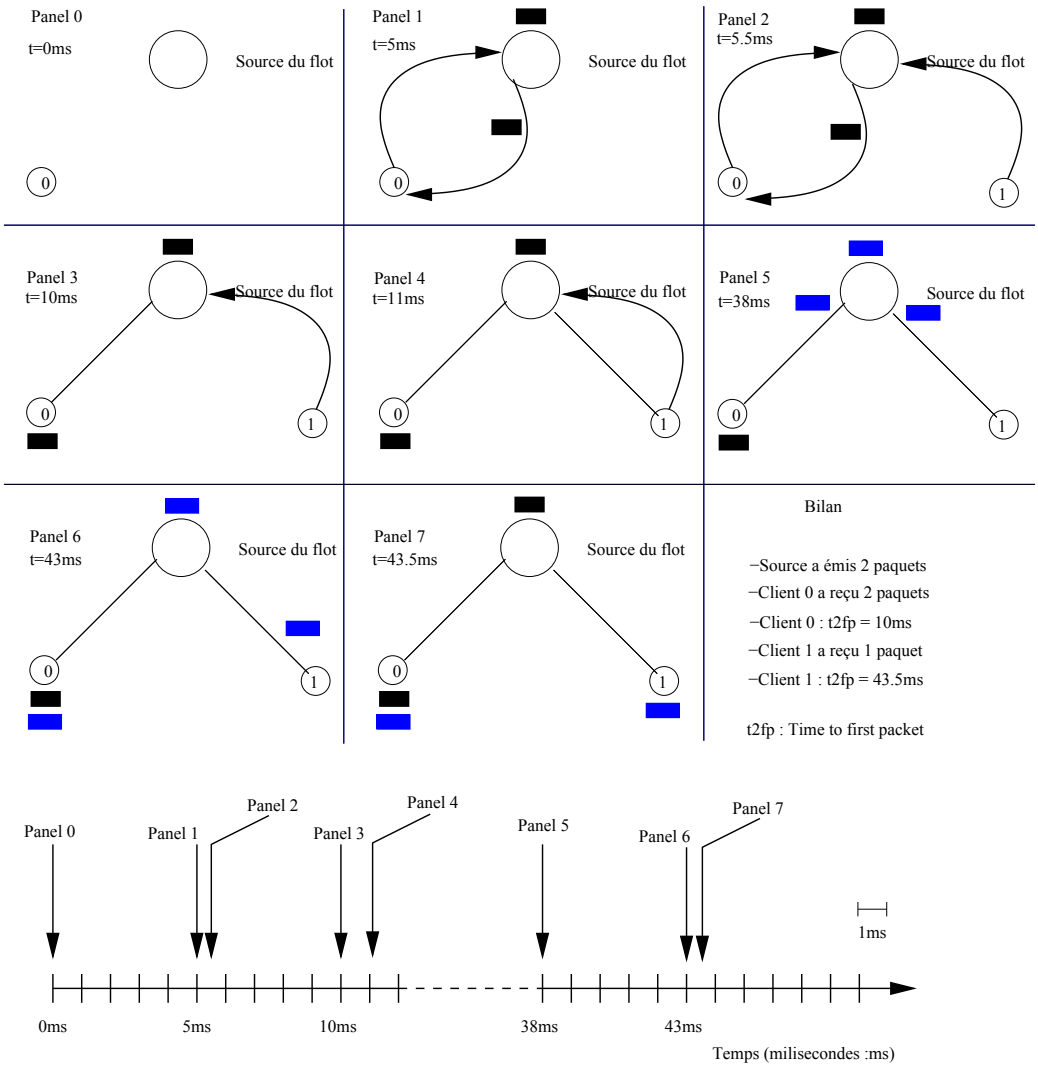

Figure 4.5: Implémentation de la transmission des données pour Peercast.

teurs le suggèrent pour un flot vidéo à 500kb/s), la source génère un second paquet à l'instant 38ms et l'envoie à tous ses enfants (*panel 5*).

le premier client recevra le second paquet à l'instant 43ms (38ms + 5ms de délai entre la source et le premier client). (*panel 6*)

Le second client recevra son premier paquet à l'instant 43.5ms (time to first packet t2fp) (*panel 7*). En fait, dans ce protocole, les clients se synchronisent sur leur source de flot (source réelle du flot ou bien pair).

Un autre élément important d'implémentation concerne les départs explicites (FIG. 4.6). Si un client quitte le réseau, il transmet à ses fils directs l'adresse de son propre père (leur grand père) afin qu'ils s'y reconnectent, si celui-ci dispose encore de suffisamment de ressources. Dans le cas de départs simultanés de plusieurs pairs, il se peut que père et fils quitte simultanément le réseau. Ainsi, le fils transmettra à ses fils directs l'adresse de son propre père. Or celui ci a également quitter le réseau.

Dans ce cas là, je considère que les clients qui doivent se reconnecter, comme il s'agit d'un départ explicite, sont plus alertes que s'il s'agissait d'une défaillance d'un pair. Ainsi, s'ils n'ont toujours pas de réponse de la part de leur grand père (vers qui ils ont été redirigés), ils le comprennent au bout d'un seul message d'heartbeat non reçu, et ils recommencent la procédure d'adhésion par la source.

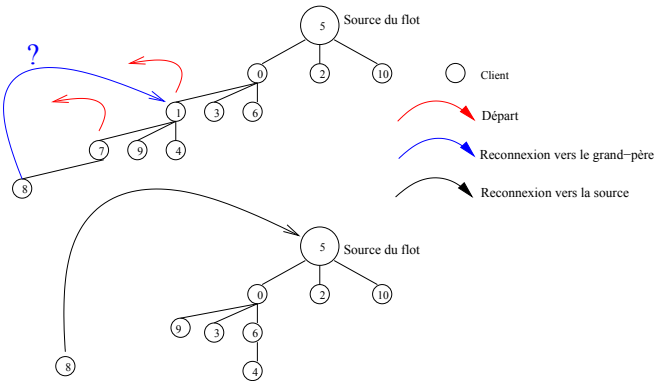

Figure 4.6: Implémentation des départs simultanés pour Peercast

4.2.3.2 Spécifications de Donet

Le procoole Donet est un protocole orientée données. Il ne construit pas de structure fixe dans l'overlay, mais construit un maillage transitoire dont les liens varient en fonction de la disponibilité des données. L'adhésion d'un client au groupe se fait en contactant la source, puis un client député, puis enfin en entamant les relations de peering avec d'autres clients.

Or les auteurs de Donet ne font aucune supposition sur le choix de ces

députés, et n'indiquent pas explicitement comment le flot est fourni au maillage.

Dans mon implémentation, j'ai choisi d'élire dix députés, qui sont les dix premiers clients à terminer leur procédure d'adhésion dans le réseau. Ces dix députés sont ainsi directement connectés à la source et permettent ainsi de pouvoir diffuser le flot dans le maillage. Les autres clients seront ainsi connecter à des pairs (députés ou non). J'utilise ce mécanisme car il est relativement analogue à celui utilisé par le protocole Slurpie([53]), protocole de distribution de fichier orientée-données (bitTorrent like). Dans Slurpie, chaque client (député ou non) à une certaine probabilité de se connecter à la source pour recevoir des blocs non disponibles dans le maillage. Ainsi, les connections entre le maillage et la source fluctuent également en fonction de la disponibilité des données. Mais Slurpie est un protocole de diffusion de fichier volumineux, et pas un protocole de diffusion de flot multimédia en direct. C'est pourquoi j'ai choisi que les connections des députés à la source soient pour l'instant fixes, car il me semble que les contraintes de temps des protocoles de streaming pairs-à-pairs ne permettent d'effectuer des changements de pairs trop souvent. Toutefois, à La différence de la source, les députés sont de vrais clients libres de quitter le réseau explicitement ou non. S'ils quittent le réseau, chacun sera alors remplacé par un autre client du réseau.

Ainsi, mon implémentation de Donet construit une espèce de hiérarchie à deux niveaux (FIG. 4.7): un premier niveau similaire à un arbre de diffusion entre la source et les dix députés, et un deuxième niveaux entre les députés et le maillage. Les députés constituent ainsi la jonction entre le maillage et la source, et permettent de fournir le flot au maillage.

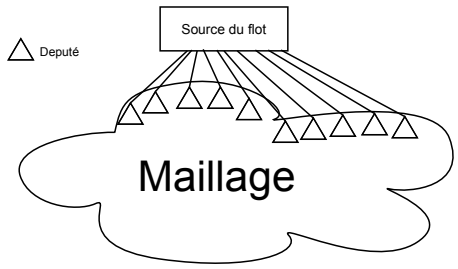

Figure 4.7: Implémentation de la structure de l'overlay de Donet.

De la même manière que le choix de dix députés est un choix d'implémentation, j'ai choisi de faire transmettre par les députés aux nouveaux clients un nombre de pairs en fonction de la taille du réseau. Ainsi, si le réseau est constitué d'au plus cents clients, les clients recevront trois pairs de la part des députés. Si le réseau est constitué d'au plus mille clients, ils recevront six pairs, et dix pairs pour des réseaux comprenant d'avantage de clients. J'ai choisi de différencier

le nombre de relation de peering suivant la taille du réseau parce que je considère que plus il y a de clients dans le réseau, plus un client aura la possibilité d'entamer de nouvelles relations de peering, alors que si le réseau est plus réduit, il n'aura pas autant d'opportunités.

Enfin, l'implémentation de Donet -pour l'instant- ne traite qu'une seule relation de peering à la fois, les autres pairs d'un client lui permettent de retrouver plus rapidement le flot en cas de pannes ou défaillances de son fournisseur du flot. Ainsi, parmi la liste de pairs qu'un client obtient, le pair choisit le pair le plus proche de lui pour récupérer le flot. Celui-ci devient alors son pair actif. Dans le cas du départ (explicite ou non) de son pair actif, le client choisit dans sa liste de pairs le pair le plus approprié à lui transmettre la suite de flot. Ce choix est fait suivant la proximité, mais bien sur suivant la disponibilité de la donnée qu'il recherche. Ainsi, un client choisira le client encore capable de lui fournir les paquets qu'il n'a pas reçu du fait du départ de son pair actif.

Si un client ne connaît plus aucun pair capable de lui fournir le flot, cela signifie que tous ses pairs ont quitté le réseau. Le client recommence alors une procédure d'adhésion au réseau.

J'ai choisi une taille de buffer de cinq secondes, sachant que c'est bien en dessous de ce qui est implémenté dans la réalité. En effet, la plupart des applications possèdent des buffer de plusieurs dizaines de secondes. J'ai choisi une si petite valeur pour rendre compte très vite des éventuelles pertes dans le réseau. Les résultats de la partie suivante montrent que même avec une si petite valeur, l'approche orientée donnée, permet de limiter considérablement les pertes de paquet dans le réseau, donc permet une meilleure qualité de réception pour l'utilisateur.

4.3 Optimisations futures

Le simulateur implémenté permet de simuler les protocoles de diffusion de flots multimédia via un réseau pairs-à-pairs. Deux protocoles ont été implémentés et simulés avec succès, comme nous le verrons dans la partie suivante.

Ce simulateur permet bien de simuler de grand réseaux puisque l'on peut simuler des réseaux comprenant 10000 clients, sans dépasser 8.5% de taux d'occupation mémoire (pour un flot d'une minute). On pourrait ainsi simuler des réseaux encore plus grands, sans être trop rapidement limité par la mémoire de la plateforme de simulation.

Par contre, les temps d'exécutions des simulations augmentent en fonction du nombre de clients. C'est du au fait qu'aucune optimisation pour parcourir les listes chaînées du simulateur n'a été utilisée. Comme optimisation, on pourrait choisir de conserver des pointeurs au milieu des listes afin d'insérer des éléments par dichotomie, plutôt que de parcourir toute la liste. On pourrait également libérer les ressources des structures de temps qui ont déjà été traitées. Cela permettrait également d'affiner la gestion de la mémoire.

En réalité, certaines de ces fonctions d'optimisation sont déjà implémentés mais elles doivent subir encore quelques tests avant d'être utilisées.

Le simulateur est un outil que je souhaite améliorer continuellement. En plus des optimisations à apporter du point de vue de son implémentation, je compte lui ajouter une interface afin de rendre son utilisation simple pour un utilisateur. De plus, je compte adapter ses paramètres d'entrées (topologie, délai d'attachement aux noeuds) afin qu'ils correspondent le plus possible à des paramètres observables sur internet.

Chapitre 5

Résultats des Simulations

En simulant le comportement de protocoles de diffusion de flots multimédias via un réseau pairs-à-pairs, on peut mesurer les performances globales de ces réseaux. Ainsi, on disposera de métriques pour comparer les différentes approches utilisées par ces protocoles afin de déterminer laquelle est la plus efficace pour ce type de transmission.

Typiquement, les protocoles de diffusion de flots multimédias en direct doivent permettre aux utilisateurs de recevoir un flot de bonne qualité, c'est à dire leur permettre de recevoir un flot continu. En d'autres termes, les protocoles doivent limiter les pertes de paquets dans le réseau et les arrivées des paquets après leur instant de lecture. De plus, s'agissant de diffusion de flots en direct, les protocoles doivent permettre aux utilisateurs de recevoir les premiers paquets du flot rapidement après leur entrée dans le réseau. Ceci est d'autant plus vrai si la demande de réception du flot fait suite à une défaillance subie par le client dans le réseau.

Par conséquent nous avons effectué des simulations pour obtenir un comportement global des protocoles suivant principalement deux métriques : le délai de réception du premier paquet après l'entrée du client dans le réseau (time to first packet *t2fp*) et le taux de pertes de paquets de données du réseau.

Le procotole Donet (FIG. 5.1)(approche orientée-données) ainsi que deux implémentations du protocole PeerCast ont été simulés, l'un construisant un arbre dont chaque noeud peut avoir trois enfants (FIG. 5.2) (PeerCast C3), l'autre construisant un arbre dont chaque noeud peut avoir 10 enfants (PeerCast C10) (FIG. 5.2).

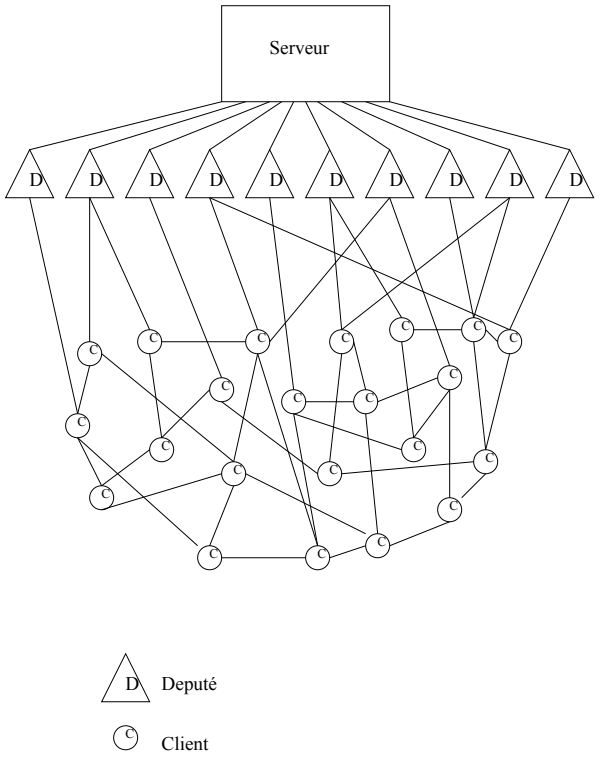

Figure 5.1: Exemple d'overlay construit par Donet.

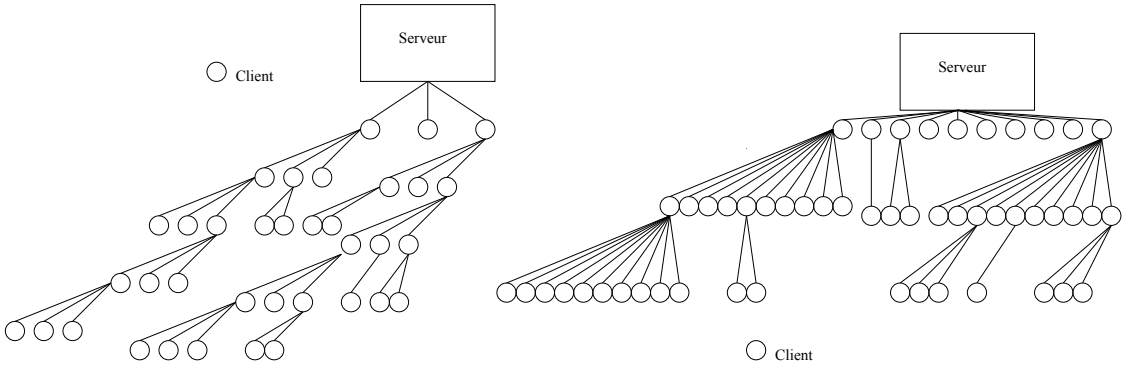

Figure 5.2: Exemple d'overlay construit par PeerCast.
A gauche, trois enfants par noeuds (PeerCast-C3). A droite, dix enfants par noeuds (PeerCast-C10)

Les invariants suivants sont communs à toutes les expériences :

- La topologie utilisée est une topologie de 1000 AS générée par Brite. C'est la même pour les trois protocoles simulés
- Chaque Client est aléatoirement attaché à un noeud de la topologie. Le délai vers le noeud d'attachement est aléatoirement généré entre 1 et 3ms
- Tous les clients entrent dans le réseau à l'instant 0.
- La source génère un paquet toute les 33ms.
- le buffer de Donet est de 5ms
- les messages d'heartbeat sont émis toutes les secondes

5.1 Délai de réception du premier paquet du flot

Dans cette expérience, nous cherchons à savoir quelle est le délai de réception du premier paquet du flot (t2fpmoy) en fonction des protocoles employées.

Paramètres de simulations

Pour cette expérience, nous avons fait varier le taille du réseau de 1 à 1600 clients.

Nous avons également effectué cent exécutions de chaque simulation afin de moyenner les valeurs mesurées pour homogénéiser le comportement du réseau. En effet, la distribution aléatoire des clients, de leur délai vers leur noeud d'attachement et de la source sur la topologie du réseau peuvent modifier les résultats des simulations.

5.1.1 Présentation des résultats

Les trois premiers graphiques(FIG. 5.3, 5.4, 5.5) sont spécifiques à chaque protocole et représentent le délai moyen de réception du premier paquet en fonction du nombre de clients dans le réseau, le délai de réception maximum obtenu dans le réseau en fonction du nombre de clients dans le réseau et l'écart type des délais de réception du premier paquet du flot. Le quatrième graphique (FIG. 5.6) reprend simplement les délais de réception moyen des trois protocoles afin de permettre au lecteur de les comparer plus facilement.

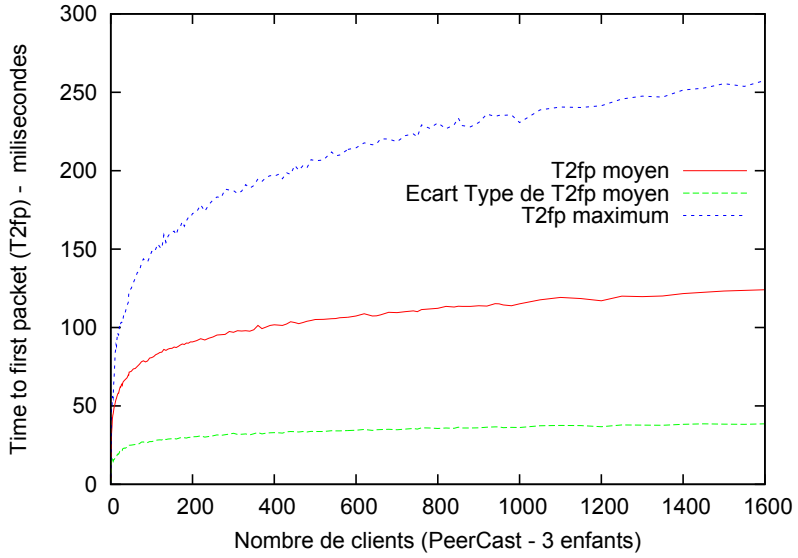

Figure 5.3: Délai de réception du premier paquet du flot pour PeerCast avec trois enfants (t2fp).

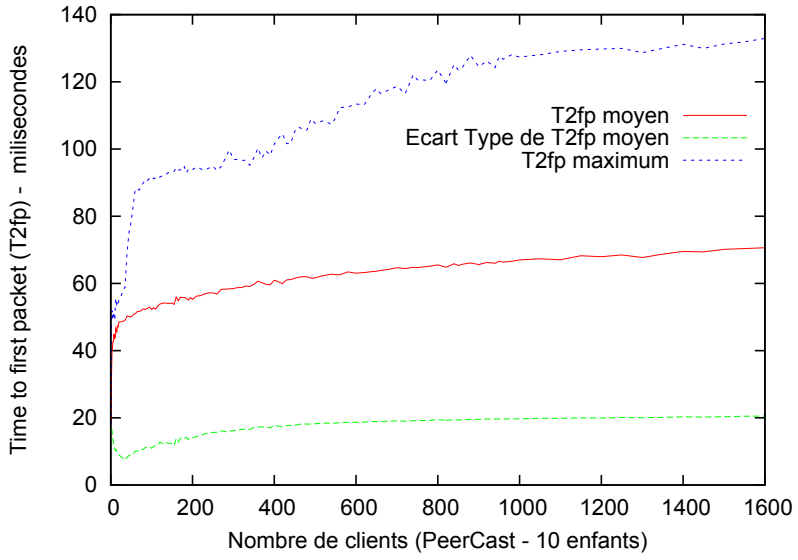

Figure 5.4: Délai de réception du premier paquet du flot pour PeerCast avec dix enfants (t2fp).

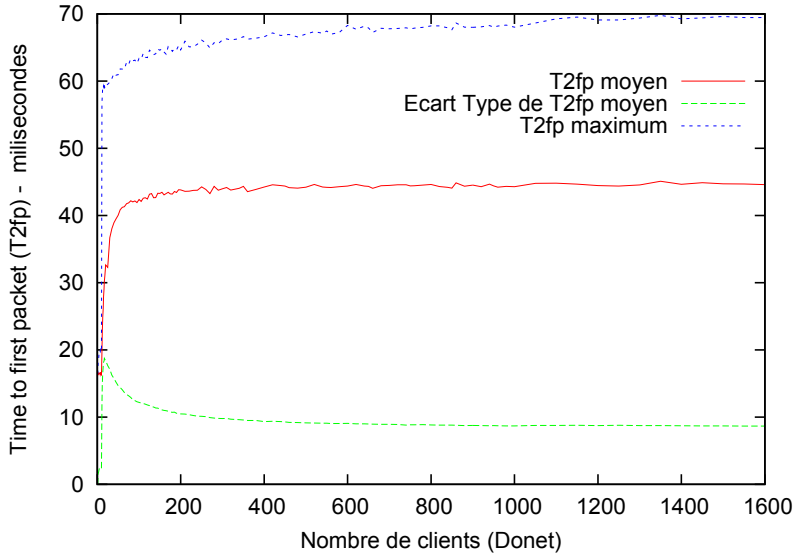

Figure 5.5: Délai de réception du premier paquet du flot pour Donet.

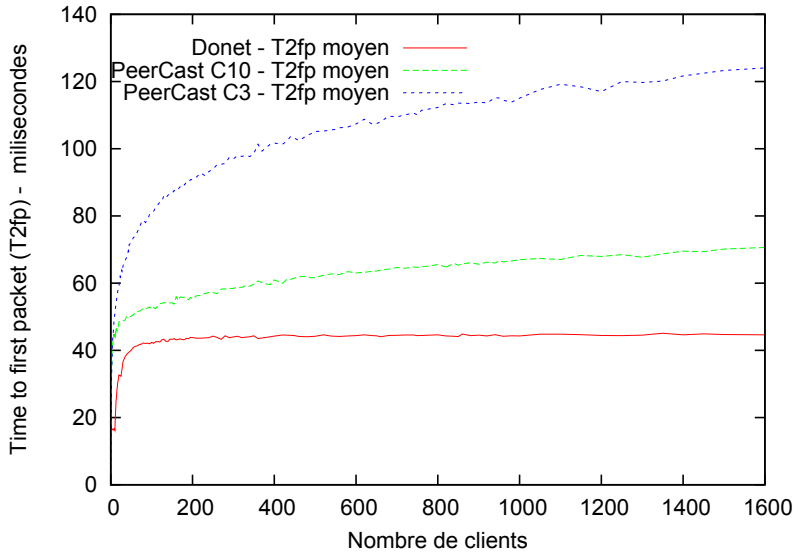

Figure 5.6: Récapitulatif des délais de réception pour les trois protocoles.

5.1.2 Observation des résultats

On observe que le délai moyen de réception du premier paquet (t2fp moyen) pour le protocole Peercast (3 ou 10 enfants par noeuds) augmente de façon logarithmique en fonction du nombre clients dans le réseau.

Pour Donet, le délai moyen de réception du premier paquet augmente jusqu'à environ cinquante clients, puis se stabilise et reste pratiquement constant malgré l'augmentation des clients dans le réseau.

Le délai de réception maximum du premier paquet (t2fp maximum) pour PeerCast-C3 semble avoir le même comportement que le délai moyen de réception : une croissance logarithmique en fonction du nombre de clients, mais à des valeurs de délais plus élevées.

Pour Donet, le délai maximum de réception du premier paquet augmente en fonction du nombre de clients dans le réseau. Le délai maximum et moyen pour Donet ne semblent donc pas avoir le même comportement.

Pour Peercast-C10, bien que le délai de réception maximum pourrait être assimilé à une croissance logarithmique en fonction du nombre de clients dans le réseau, on observe cependant une sorte d'augmentation par palier :

une croissance rapide du délai maximum de réception du premier paquet jusqu'à environ 100 clients. Une croissance plus faible de 100 à 380 clients dans le réseau (premier palier), puis une croissance plus rapide des délais jusqu'à 1000 clients, et enfin une augmentation plus faible jusqu'à 1600 clients (second palier).

5.1.3 Interprétation des résultats

Tout d'abord, on observe que le délai moyen de réception du premier paquet est plus court avec Donet qu'avec les autres implémentations de PeerCast.

Concernant PeerCast, on observe que le délai moyen de réception du premier paquet du flot est plus important pour l'implémentation de PeerCast avec trois enfants (PeerCast-C3) que pour l'implémentation avec dix enfants (PeerCast-C10).

Dans les deux cas, un arbre de diffusion est construit dans l'overlay et chaque client, pour recevoir le flot, doit parcourir l'arbre jusqu'à trouver un noeud disposant de suffisamment de ressources pour l'accepter.

Or, dans l'implémentation avec dix enfants, chaque noeud accepte plus de clients donc utilise plus ressources pour accepter un client que dans l'implémentation avec trois enfants. Ainsi, avec PeerCast-C10, les ressources disponibles dans le réseau sont plus importantes, donc un client aura moins de requêtes à effectuer pour trouver un pair capable de lui fournir le flot qu'avec PeerCast-C3. Avec PeerCast-C10, un client parcourra donc un arbre moins profond qu'avec PeerCast-C3 pour recevoir le flot. (cf FIG. 5.2). Ainsi, en moyenne le délai de réception du premier paquet du flot sera inférieur pour une implémentation de PeerCast avec dix enfants par rapport à une implémentation avec trois enfants.

Plus généralement, plus les noeuds peuvent accepter d'enfants dans l'arbre, plus le délai moyen de réception du premier paquet sera court. En effet, plus un

noeud accepte d'enfants, et plus il met de ressources à la disposition des autres pairs du réseau (Bande passante, temps de traitement etc.). Ainsi, il est normal que les performances du réseau soit meilleures quand celui-ci disposent de plus de ressources. C'est cette mise à disposition des ressources pour les autres pairs du réseau qui est le concept même du pairs-à-pairs. On vérifie donc ce concept : plus les pairs partagent de ressources et plus les performances du réseau sont meilleures pour tous les pairs.

Pour Donet, on observe une augmentation du délai moyen de réception jusqu'à une cinquante de clients, puis le délai devient constant. Ce résultat peut s'expliquer de la façon suivante :

La procédure d'adhésion du protocole implique qu'un client contacte la source, puis un député, puis des pairs qui vont alors lui transmettre le flot. Ce protocole nécessite aux clients d'effectuer trois requêtes.

Pourtant, selon l'implémentation du protocole sur le simulateur, afin de diffuser le flot dans le maillage conçu par Donet, les dix premiers clients à contacter la source (donc les dix plus proches de la source) sont tous élus députés. Ils n'ont donc qu'une seule requête à effectuer pour récupérer le flot directement de la source. Puis, les clients supplémentaires suivent le protocole normalement via l'échange des trois requêtes. Les députés ont donc un délai de réception du premier paquet très rapide par rapport au client normaux puisque composé d'une unique requête plutôt que trois.

Quand le réseau est de petite taille, la proportion de député est importante, mais plus la taille du réseau augmente, et plus la proportion de députés diminue. Par exemple, si le réseau comprend 20 membres, alors la moitié seront des députés. Si le réseau comprend 40 membres, seulement le quart seront députés. Pour mille membres, seulement 1% des clients seront députés. Donc à l'inverse plus la taille du réseau augmente, plus la proportion de clients qui effectuent trois requête pour recevoir le flot augmente (clients non députés).

Par exemple, si la taille du réseau est de dix clients, on comptera 10 Requêtes dans le réseau pour recevoir le flot, soit une moyenne de 1 requête par clients.

Avec 20 clients : $10 * 1 + 10 * 3 = 40$ requêtes donc 2 requêtes par clients en moyenne. On peut généraliser le nombre de requêtes dans le réseau pour recevoir le flot par : $10 * 1 + (N - 10) * 3 = (3N - 20)$ requêtes, où N est le nombre de clients dans le réseau.

Ainsi, plus la taille du réseau augmente, plus le nombre de requêtes pour recevoir le flot tendra vers 3 requêtes (cf. TAB 5.1), c'est à dire le cas normal, sans les hypothèses d'implémentation du protocole. En effet, $\lim \frac{(3*N-20)}{N} \rightarrow 3$.

Nombre de clients	Nombre de Requêtes	Nombre moyen de Requêtes
10	10	1
20	40	2
50	130	2.6
100	280	2.8
200	580	2.9
1000	2980	2.98

Table 5.1: Nombre de Requêtes dans le réseau et Nombre moyen de Requêtes par client

En réalité, on n'observe pas une forte croissance du délai moyen de réception du premier paquet jusqu'à un certain nombre de clients dans le réseau (une cinquantaine de clients), mais on observe l'effet de la moyenne des t2fp en fonction du nombre de clients ***selon l'implémentation du protocole***.

En effet, pour un réseau de petite taille, si 10 clients effectuent une seule requête pour recevoir le flot pendant que les quelques autres clients en effectue trois, cela affecte grandement le délai moyen de réception du premier paquet du flot. Cependant, à partir d'un certain nombre de clients (une cinquantaine), les faibles délais de réception du premier paquet (t2fp) des dix députés ne seront plus perceptibles parmi les durée de réception trois fois plus longues (3 requêtes contre une) des autres clients.

L'augmentation que l'on distingue est seulement due à l'implémentation du protocole, et n'est plus perceptible dès que les députés ne représentent plus une portion significative des membres du réseau.

On peut ainsi considérer que le délai moyen de réception du premier paquet tend vers le délai moyen des trois requêtes dans le réseau. On peut ainsi observer que le délai moyen des trois requêtes dans le réseau est relativement constant sur cette topologie, et que Donet permet un délai de réception du premier paquet du flot constant pour tous les clients.

Pour ce qui est des délais maximum de réception du premier paquet, cela correspond au pire cas de réception du premier paquet par un client. Dans le cas de PeerCast-C3, ces délais augmentent de façon logarithmique en fonction du nombre de clients dans le réseau ce qui était attendu compte tenu du délai moyen de réception du premier paquet.

La croissance logarithmique pour PeerCast-C10 se vérifie grossièrement, même si l'on peut noter un certain effet de palier. Je pense que comme PeerCast ne construit pas les arbres de façon à ce que chaque niveau soit complètement rempli avant d'insérer des clients dans les niveaux supérieurs (ceci étant du au SmartPlacement du protocole), on peut alors observer des différences dans la construction de l'arbre en fonction de la répartition des clients sur la topologie. Ainsi, à partir d'un certain nombre de clients, on pourrait construire des arbres suffisamment différents pour que cent exécutions de la simulation ne suffisent pas à réellement rendre compte du comportement global du réseau dans le pire des cas. Par exemple, pour 500 clients dans le réseau, on pourrait obtenir

Nombre de Clients	T2fp moy (ms)	T2fp max (ms)
200	43.80	65.65
350	44.34	66.62
500	44.21	67.01
750	44.40	67.77
1000	44.29	68.03
1200	44.45	69.10
1600	44.60	69.45

Table 5.2: Résultats de simulation pour Donet. Délai de réception du premier paquet du flot en moyenne et maximum en fonction du nombre de clients dans le réseau.

un arbre atteignant le troisième niveaux de profondeur (meilleur cas), mais on pourrait aussi obtenir un arbre atteignant le cinquième niveau de profondeur (la racine des arbres étant toujours au niveau 0). Pour un client, être inséré dans le réseau au troisième niveau (donc en trois requêtes) ou au cinquième niveau (donc en 5 requêtes) signifie un délai de réception du premier paquet qui passerait pratiquement du simple au double (5requêtes au lieu de trois requêtes).

Aussi, je pense que le délai maximum de réception du premier paquet pour un client dans le réseau est logarithmique avec PeerCast-C10, mais que la moyenne de cent exécutions de la même simulation ne parvient pas à parfaitement rendre compte du comportement global du réseau dans ce pire cas là ce qui explique les effets de paliers observés. Mais si l'on prend la courbe dans son ensemble, on observe bien une croissance logarithmique comme pour PeerCast-C3.

De plus, je pense que l'on n'observe pas ce phénomène de palier pour PeerCast-C3, car chaque noeud peut accepter moins de clients. On ne retrouvera donc pas autant de différences dans les construction d'arbres que pour PeerCast-C10, donc la moyenne de cent exécutions suffit à rendre compte du comportement global du réseau pour mesurer le délai de réception maximum.

Dans le cas de Donet ([16]), la durée moyenne de réception du premier paquet (*t2fp)* est relativement constante, mais la durée maximum semble légèrement augmenter en fonction du nombre de clients. Or, les auteurs du protocole démontrent que dans l'overlay constitué par Donet, les délais de livraison des données sont bornées par $\log N$ (N étant le nombre de clients). C'est sans doute cette borne que l'on a mesuré avec le délai maximum de réception du premier paquet du flot.

Finalement, on observe que Donet permet un délai de réception du premier paquet plus court que pour PeerCast, ce qui en fait un protocole plus performant selon cette métrique. Ce gain en performance est due à l'approche employée par Donet pour mettre les pairs en relation. Avec Donet, les clients du réseau effectuent leurs relations de peering avec des pairs qui leur ont été attribués aléatoirement. Ainsi, il est souvent probable pour un client d'être directement

dirigé vers un pair ayant assez de ressources pour l'accepter, contrairement à PeerCast où chaque client parcourt la structure d'arbre de l'overlay depuis la source jusqu'à trouver un pair ayant assez de ressources pour l'accepter. Ainsi, avec Donet, le délai moyen de réception du premier paquet du flot est relativement constant alors que pour PeerCast il augmente en fonction du nombre de clients dans le réseau.

5.2 Répartition des clients dans la structure de l'overlay

Cette expérience présente pour chacun des trois protocoles la répartition des clients en fonction de leur distance par rapport à la source (en termes de sauts), c'est à dire en fonction du niveau que chaque client occupe dans la structure de l'overlay.

Paramètres de Simulation

Dans cette expérience, la taille du réseau est fixée à 1000 clients.

Nous avons effectué cent exécutions des simulations afin d'obtenir un comportement global des réseaux

5.2.1 Présentation des résultats

Ce graphique(cf FIG. 5.7) présente la répartition des clients en fonction de leur niveau dans la structure de l'overlay pour chaque protocole.

L'axe des abscisses correspond au niveau dans la structure de l'overlay.

L'axe des ordonnées correspond à la proportion des noeuds (en pourcentage)

5.2.2 Observation des résultats

En ce qui concerne PeerCast-C10(5.7), on observe un pic de clients au troisième et quatrième niveau de la structure de l'overlay ce qui signifie que la plupart des clients se trouvent à trois ou quatre sauts de la racine de l'arbre construit sur l'overlay (la source). Certains clients peuvent également se trouver au septième niveau donc à sept sauts de la racine de l'arbre.

Pour PeerCast-C3, la répartition est plus équitablement partagé entres les 5èmes et dixième niveaux, ce qui correspond a autant de sauts vers la racine. Cependant, certains clients se retrouvent au 16ème niveau, ce qui correspond à une distance de 16 sauts de la source.

Pour Donet, la répartition des clients se situent entre les 5ème et onzième niveau, certaines clients atteignant même le niveau 18, soit autant de saut par rapport à la source.

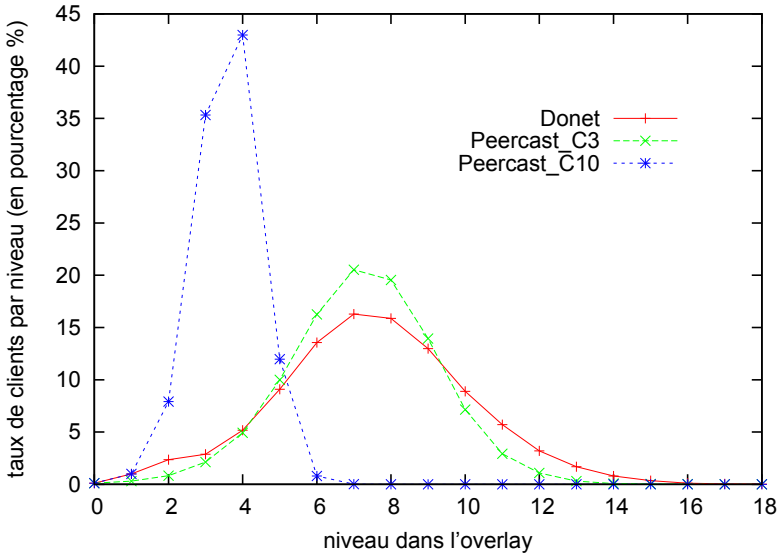

Figure 5.7: Répartition des noeuds dans leur structure d'overlay pour les trois protocoles (Donet, PeerCas-C3, PeerCast-C10).

5.2.3 Interprétation des résultats

Concernant les différentes implémentations de PeerCast, on constate que PeerCast-C10 permet d'obtenir des clients plus proches de la source que PeerCast-C3. Ce qui est normal car chaque client peut accepter plus d'enfants, donc l'arbre construit sera moins profond pour un réseau de même taille. Cette observation corrobore bien le fait que les délais moyens de réception du premier paquet pour PeerCast-C10 sont inférieurs à ceux de PeerCast C3. En effet, les clients avec PeerCast-C10 sont plus proches de la source que dans PeerCast-C3, donc les messages sont transmis en effectuant moins de sauts, donc les délais de transmission sont plus court.

On peut toutefois remarquer pour un réseau de 1000 noeuds, certains clients peuvent se retrouver à des niveaux assez élevées dans l'arbre. Or, si l'on construisait un arbre équilibré, atteindre le troisième niveau serait suffisant pour PeerCast-C10, et le sixième niveau pour PeerCast-C3. Ce phénomène est du à l'utilisation de la fonction de Smart-Placement recommandé par les concepteurs du protocole. Avec cette fonction, les clients vont être redirigés vers le pair le plus proche d'eux, même si ses ressources sont déjà occupées et que des ressources sont disponibles chez les pairs du même niveau. Ainsi, un client pourra 's'enfoncer' dans l'arbre plutôt que de trouver un autre pair dans un niveau supérieur.

En réalité, cette fonction est peut être plus satisfaisante que le choix aléatoire du pair de redirection ou le choix périodique du pair (Round Robin), mais elle

peut éventuellement aboutir à une mauvaise utilisation des ressources du réseau, avec des pairs proches de la source (en nombre de sauts dans l'overlay) qui n'ont pas utilisé leurs ressources tandis que des clients s'enfoncent dans les profondeurs de l'arborescence. Toutefois, la fonction de Smart Placement place les clients en fonction de leur proximité (en termes de délais) avec les autres noeuds. Ainsi, plusieurs sauts dans l'overlay peuvent éventuellement être moins coûteux en délai qu'un unique saut vers un autre client très éloigné.

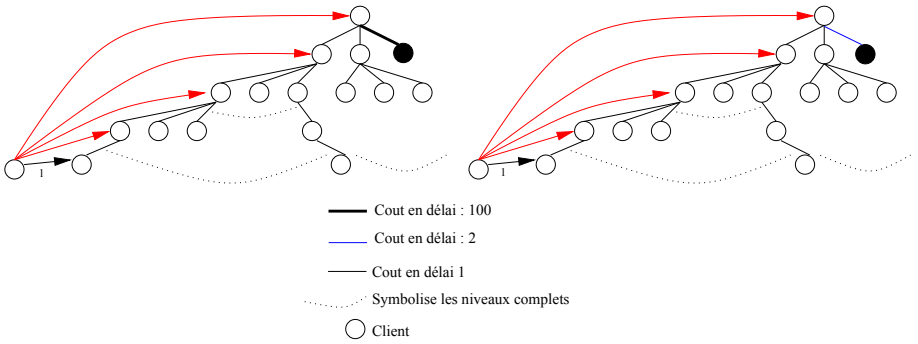

Figure 5.8: Effet du *Smart Placement de* PeerCast. A gauche, le Smart Placement permet de bien placer le client. A droite, le client s'enfonce dans l'arbre.

Par exemple, sur la figure (cf. FIG. 5.8), le client qui cherche à rejoindre le réseau s'enfonce dans l'arbre sans utiliser les ressources mises à disposition par le noeud noir. Dans le schéma de gauche, le client va se retrouver à cinq sauts de la source mais à un délai dont le coût est 5 alors que s'il s'était accroché au noeud noir, le coût en délai aurait été de 101 malgré seulement deux sauts. Par contre, dans le schéma de droite, si le client avait utilisé les ressources du client noir, il se serait retrouvé au deuxième niveau avec un délai dont le coût aurait été de 3 plutôt que de se retrouver au cinquième niveau avec un délai dont le coût est 5. Cet exemple illustre les limites de la fonction Smart Placement. Dans l'exemple de gauche, elle a bien remplie son rôle, et le nombre de sauts dans l'overlay n'implique pas nécessairement de plus grand délai de transmission. Par contre, dans le schéma de droite, son utilisation a conduit à une mauvaise utilisation des ressources, puisque le client s'enfonce dans l'arbre plutôt que de profiter des ressources du noeud noir. Pour améliorer cette fonctionnalité, il faudrait qu'elle dispose d'informations sur l'état des ressources des pairs vers qui elle redirige le nouveau client.

Pour ce qui est de Donet, la répartition des clients par rapport à la source est similaire à la répartition des clients avec PeerCast-C3. Pourtant, on a observé que les délais moyens de réception du premier paquet sont inférieurs à ceux des deux implémentations de PeerCast.

Ainsi, avec Donet, chaque noeud récupère le paquet par rapport à son pair fournisseur de flot, sans que la distance par rapport à la source n'intervienne.

Dans Le maillage constitué par l'approche orientée-données, le flot est diffusé de façon à ce que chaque client puisse le récupérer plus rapidement qu'avec l'approche orientée-source, et le nombre de saut vers la source n'a pas d'impact pour ce type de structure d'overlay contrairement à la structure d'arbre dont la racine est la source construite par PeerCast.

5.3 Pertes de paquets de données en fonction de la dynamicité des clients

Dans cette expérience, on cherche à tester les effets de la dynamicité des clients sur la transmission des données dans le réseau. Pour cela, on a simulé pendant une minute la diffusion d'un flot sur réseau, et on a fait quitter explicitement (*LEAVE*) des clients pendant la simulation.

Paramètres de simulation

Topologie: Afin de comparer rigoureusement les protocoles entre eux, on a choisi une certaine configuration de la topologie du réseau, et effectué toutes les expériences sur cette même topologie. C'est à dire que l'on a généré une disposition de tous les clients et la source sur le réseau, ainsi que tous leurs délais vers leurs noeud d'attachement, et l'on a conservé cette topologie pour chaque expérience avec les 3 protocoles. Ainsi, dans chacune des expériences suivantes, il s'agit rigoureusement de la même topologie de clients et bien sur de noeuds (AS). Nous n'avons donc qu'une unique exécution de chaque simulation à effectuer.

Nombre de clients: La taille du réseau est fixé à 1000 clients.

Dynamicité: Les clients qui quittent le réseau le quitte au bout de 30 secondes de simulation.

5.3.1 Présentation des résultats

Pour mesurer l'effet de la dynamicité des clients sur la transmission des données dans le réseau, on a effectué deux types d'expériences pour chaque protocole :

-une expérience où l'on augmente à chaque simulation le nombre de clients qui quitte le réseau(cf. FIG. 5.9).

-une expérience où l'on fait quitter tous les clients d'un certain niveau de la structure créée sur l'overlay (cf. FIG. 5.10).

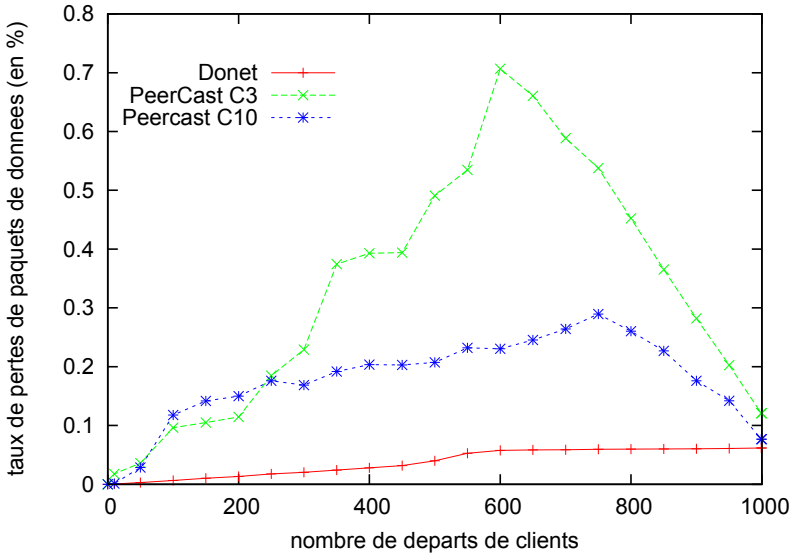

Figure 5.9: Taux de pertes (en %) de paquets de données en fonction du nombre de départs des clients pour les trois protocoles (Donet, PeerCast-C3, PeerCast-C10).

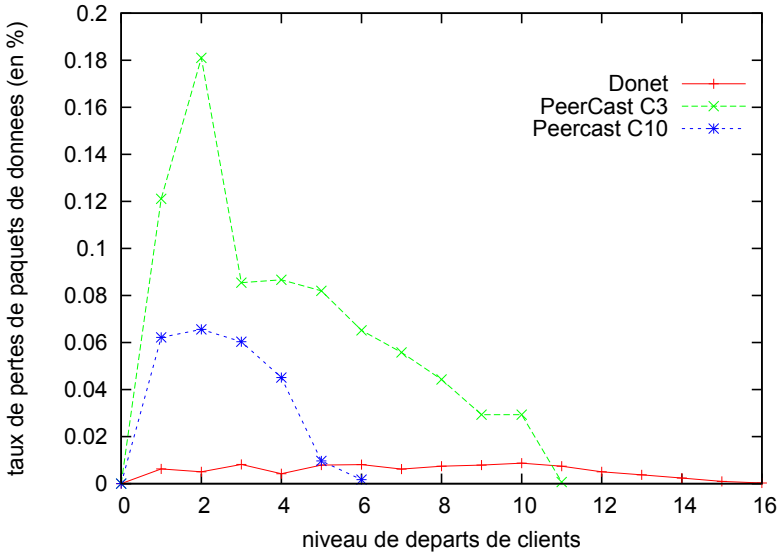

Figure 5.10: Taux de pertes (en %) de paquets de données en fonction du niveau de départs des clients pour les trois protocoles (donet, PeerCast-C3, PeerCast-C10).

5.3.2 Obervation des résultats

Dans le graphique 5.9, on observe que pour les 3 protocoles, le taux de pertes de paquets de données augmentent en fonction du nombre de départ de clients.

Cependant, le taux de pertes de données pour le protocole Donet est toujours très inférieur au taux de pertes des implémentations de PeerCast. Entre les différentes implémentations de PeerCast, le taux de pertes de l'implémentation de PeerCast-C10 est aussi très inférieur à l'implémentation PeerCast-C3.

Dans la courbe 5.10, on observe que pour le protocole PeerCast, le taux de pertes varie en fonction du niveau de départ des clients. Pour Donet, le taux de pertes en fonction du niveau de départ semble relativement constant. A nouveau, le taux de pertes de paquets de données pour Donet est toujours très inférieur au taux de pertes de PeerCast et de ces différentes implémentations. Le taux de pertes rencontrés par PeerCast-C10 est également très inférieur à celui de PeerCast-C3.

5.3.3 Interprétation des résultats

Comme on peut le constater avec les deux expériences, le taux de pertes rencontrés par Donet est très inférieur au taux de pertes rencontrés par PeerCast, quelque soit son implémentation.

Cela est du à la redondance de pairs dont disposent les clients du réseau Donet. En effet, les clients se sont vus attribués un certain nombre de pairs (6 pairs pour un réseau de 1000 clients selon mon implémentation de Donet) pour effectuer leurs relations de peering et récupérer le flot. Si le pair qui leur fournit le flot quitte le réseau (de manière explicite dans cette expérience), le pair pourra choisir un autre pair dont il connaît l'existence pour continuer à recevoir le flot. De plus, si tous les pairs d'un client ont quitté le réseau, le client recommencera une procédure d'adhésion en partant du député ou de la source afin de récupérer le flot. Or on a vu que le délai de réception du premier paquet du flot pour le protocole Donet est inférieur à ceux de PeerCast, donc même si le client doit recommencer la totalité ou une partie de la procédure d'adhésion, le client retrouvera rapidement un pair capable de lui fournir le flot et ne subira pas donc pas beaucoup de pertes.

Pour ce qui concerne PeerCast, lors d'un départ explicite, les clients sont redirigés vers leur grand-père, ou à la source si le grand père a quitté aussi le réseau. Puis ils tentent de trouver un nouveau pair capable de les accepter en parcourant l'arbre à partir de la source ou du grand père. En effet, contrairement à Donet où les clients ont immédiatement le choix entre plusieurs pairs pour retrouver le flot, les clients de PeerCast ne connaissent que leur grand-père ou la source pour récupérer le flot. Or si le réseau compte une grande quantité de départs, source et grand pères ne pourront pas accepter toutes les nouvelles demandes de connexions des clients. Les clients devront alors parcourir l'arbre jusqu'à trouver un pair capable de les accepter et de leur fournir à nouveau le flot. La récupération du flot par un client est ainsi plus longue que pour Donet, d'autant que l'on a vu que le délai de réception du premier paquet pour

PeerCast est plus important que pour Donet. Ainsi, comme avec PeerCast les clients prennent plus de temps à récupérer le flot, ils subissent plus de pertes qu'avec Donet.

De plus, entre les deux implémentations de PeerCast, c'est bien entendu PeerCast-C3 qui subit le plus de pertes. La raison est que les grand-pères et la source vers qui sont redirigés les clients dont le père est parti disposent de moins de ressources pour accepter les clients que dansl'implémentation avec dix enfants. Aussi, avec cette implémentation, les clients devront presque systématiquement parcourir l'arbre pour récupérer le flot. Or l'arbre de diffusion de PeerCast-C3 est plus profond et entraîne un délai moyen de réception du premier paquet plus important que celui de PeerCast-C10. Autrement dit, les clients avec PeerCast-C3 mettront plus de temps à trouver un pair capable de leur fournir le flot, donc subiront plus de pertes qu'avec PeerCast-C10.

Quand les clients quittent le réseau en fonction de leur niveau dans la structure de l'overlay (cf. FIG. 5.10), on peut constater que pour Donet cela n'a pas beaucoup de conséquences sur l'augmentation du taux de pertes de paquet de données alors que cela en a sur les implémentations de PeerCast.

Cela est du au fait que dans PeerCast, la structure formée est un arbre de diffusion où les clients en aval dépendent directement des clients en amont. Ainsi, si certains niveaux quittent tous ensemble le réseau, cela peut pénaliser la plus grande partie du réseau en aval et occasionner beaucoup de pertes. Par exemple, les départs de clients au niveau deux ou trois pour PeerCast-C10 occasionnent beaucoup de pertes car beaucoup de clients du réseau sont en aval (d'un niveau supérieur à deux ou trois dans la hiérarchie, donc directement dépendant des clients qui quittent le réseau). Ils ne pourront pas tous se reconnecter à leur grand-père donc devront parcourir une partie de l'arbre ce qui coûteux du point de vue des pertes de paquet puisque cela prend un certain temps. De même, les départs de tous les noeuds de niveau 2 à 6 pour PeerCast-C3 occasionnent de nombreuses pertes puisque la plupart des clients sont en aval. Pour PeerCast-C3 (cf. FIG. 5.11), si tous les clients du niveau 2 quittent le réseau. Alors pour seulement 9 départs au maximum (puisque l'arbre n'est pas forcément équilibré), pratiquement tout le réseau sera déconnecté. Les 27 enfants directs (maximum) des 9 noeuds qui quittent le réseau vont contacter leur 3 grand-pères qui ne pourront accepter que 9 clients au maximum. Les 18 autres enfants directs (qui ont également des enfants) vont donc devoir parcourir l'arbre de diffusion pour récupérer le flot ce qui va prendre du temps et entraîner des pertes de paquet pour beaucoup de client du réseau. Cet exemple est illustré sur la figure 5.11.

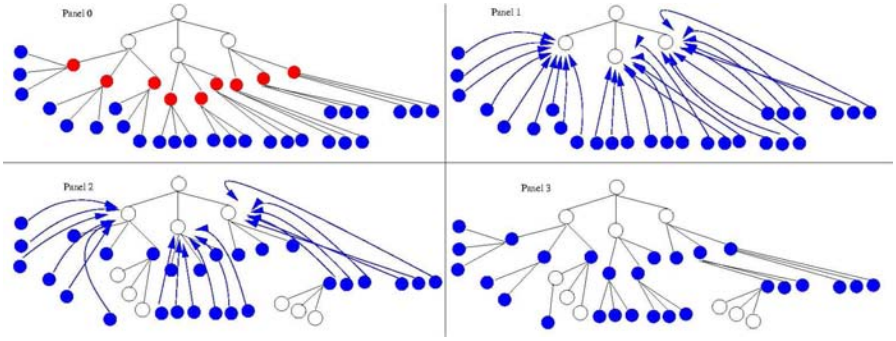

Figure 5.11: Exemple de départ de tous les noeuds d'un même niveau pour PeerCast-C3 (niveau2).

panel 0 : topologie de départ. Les noeuds rouges quittent le réseau. Les éventuels fils des noeuds bleus ne sont pas représentés. *panel 1*: tous les fils directs (bleus) contactent leur grand-père. Les éventuels fils des noeuds bleus ne sont pas représentés. *panel 2*: certains fils directs ont pu se reconnecter. Les autres doivent parcourir l'arbre. *panel 3*: exemple de reconnection des clients qui ont subis des départs.

Sinon on constate bien que les départs des clients des derniers niveaux n'occasionnent aucune perte dans le réseau puisque ces clients ne servent le flot pour personne.

Pour ce qui concerne Donet, la structure crée dans l'overlay permet d'éviter les dépendances entre niveaux que l'on retrouve dans les structures d'arbre. A la différence de PeerCast, dans la structure crée par Donet dans l'overlay, les départs de clients n'ont qu'un impact local : le départ d'un client affecte seulement les clients qui avaient initiés une relation de peering active avec ce client, c'est à dire seulement les clients qui recevaient directement le flot par son intermédiaire.

Dans le cas de départ explicites, le temps que mettent les clients pour retrouver le flot entraîne les pertes de paquet, et l'on a vu que Donet récupère le flot beaucoup plus rapidement que les implémentations de PeerCast, donc il subira beaucoup moins de pertes.

Au cours de ces expériences, on a pu constater que Donet subissait des pertes nettement moins importantes que PeerCast. Ainsi, du point de vue de cette métrique il est beaucoup plus performant que PeerCast.

5.4 Conclusion

Bien que PeerCast et Donet ont tout deux pour vocation la diffusion de flots multimédias en direct via un réseau pairs-à-pairs, les différentes approches qu'ils utilisent ont un sérieux impact sur les performances globales des réseaux. On

peut constater que l'approche orientée-données, grâce à l'utilisation d'algorithmes épidémiques, surpasse toujours l'approche orientée source, approche qui ne fait que réinjecter au niveau applicatif les mécanismes développés pour le multicast de niveau réseau.

L'approche orientée-données obtient de meilleures performances que l'approche orienté-source car elle est rendue moins dépendante de la structure de l'overlay que l'approche orienté-source et elle dispose d'informations redondantes, ce qui lui permet d'être plus réactive et moins sensible à la dynamicité des clients du réseau. Or, la gestion de la dynamicité est l'un des problèmes principaux à résoudre pour implémenter les fonctionnalités des communications de groupe au niveau applicatif (multicast applicatif). Ainsi, les protocoles qui utilisent une approche orienté-données devraient permettre de résoudre ce problème de dynamicité des clients des communications de groupe de niveau applicatif.

L'approche orientée-données permet notamment de recevoir le flot plus rapidement, ce qui pour une diffusion en direct est très important, mais surtout cela permet de retrouver le flot plus rapidement en cas de départs ou de défaillances des pairs du réseau. Ainsi, cette approche permet des taux de pertes nettement inférieur à l'approche orienté-source, ce qui permet de remplir le principal objectif des applications reposant sur ce type de protocole : la diffusion à l'utilisateur un flot de bonne qualité.

Bien que ces travaux ont été réalisé sur une topologie de réseau de type Internet, il convient de vérifier la justesse de leurs résultats en réalisant des expériences aux conditions d'Internet. A la suite de ces travaux, j'adapterais la configuration du simulateur à une configuration plus proche de celle observée sur Internet, en basant le simulateur sur une topologie d'AS réelle, avec des distributions des clients sur la toplogie et de leur dynamicité conforment à ce que l'on observe sur Internet. De plus, les délais simulés entre les clients et leur noeud d'attachements seront adaptés afin d'être plus proche de ceux mesurés sur Internet.

De plus, comme l'approche orientée-données introduit de la redondance d'informations dans le réseau, nous chercherons à estimer, au cours de nos travaux futurs, si cette redondance n'est pas trop coûteuse pour un déploiement à large échelle. Nous chercherons également à quantifier précisément les ressources que nécessitent ce type d'application afin de vérifier que leur déploiement est bien envisageable actuellement sur Internet.

Chapitre 6

Conclusion

Les applications de diffusion de flots multimédias en direct sont des applications de plus en plus utilisées sur Internet. A nouveau, ce type d'application utilise le réseau Internet contre toute attente, puisque Internet ne permet qu'un service 'best effort' et que ces applications présentent de fortes contraintes temporelles. Aussi, les protocoles utilisés par ce type d'application doivent être méticuleusement conçus afin que les utilisateurs puissent recevoir des flots de bonne qualité.

Ce type d'application demande beaucoup de ressources, c'est pourquoi les protocoles développés actuellement font collaborer les clients entre eux afin d'augmenter les ressources disponibles et permettre la réception correcte du flot pour tous les utilisateurs.

Nos travaux nous ont permis de constater que les protocoles de diffusions de flots multimédias en direct via un réseau pairs-à-pairs pouvaient utiliser des approches très différentes, dont l'impact sur les performances du réseau et la qualité de réception de l'utilisateur était flagrante.

Afin de déterminer quelle approche convenait le mieux pour ce type de protocole, nous avons simulé le comportement de certains d'entre eux pour mesurer l'impact de l'approche utilisée sur le réseau.

Nos travaux nous permettent de constater que pour ce type de protocole, l'approche orientée-données semblent plus apte que l'approche orientée-source à atteindre les objectifs de ce type de communication : la réception par l'utilisateur d'un flot de bonne qualité. En effet, ce type de communication nécessite l'implémentation des fonctionnalités de communication de groupe au niveau des hôtes. Or, les hôtes ont un comportement sur Internet très dynamiques et imprévisibles. L'approche orienté-données, via l'utilisation d'algorithmes épidémiques, permet -comme nous l'avons vu- de moins structurer les clients dans l'overlay et utilise des mécanismes de transmission de messages aléatoires et la redondance d'informations qui permettent de prévenir les effets de la dynamicité des clients et de l'anticiper. C'est pourquoi, L'approche orientée-donnée semble toute disposée à permettre ce type de communication.

A la suite de ces travaux, nous configurerons les paramètres du simulateur à des valeurs plus proches de celles rencontrées sur Internet pour affiner les simu-

lations et nos observations de leurs résultats. Cet outil pourra nous permettre dans un premier temps d'améliorer le comportement de protocoles existants afin de les adapter à ce type de communication, et aussi de concevoir des protocoles parfaitement étudiés pour ces communications.

Bibliographie

[1] PeerCast : Streaming Live Media Over Peers, Deshpande Hrishikesh, Bawa Mayank, Garcia-Molina Hector,Mars 2002

[2] Narada : A Case for End System Multicast, Chu, Rao, Seshan, Zhang, in "Measurement and Modeling of Computer Systems", 2000

[3] Nice : Scalable Application Layer Multicast, Banerje, Bhattacharjee, Kommareddy, SIGCOMM, Août 2002

[4] ZigZag : An Efficient Peer-to-Peer Scheme for Media Streaming, Tran, Hua, Do, INFOCOM 2003

[5] Bullet : High Bandwith Dta Dissemination Using an Overlay Mesh, Kostic, Rodriguez, Albrecht, Vadhat, SOSP 2003

[6] Promise : P2P Media Streaming Using CollectCast, Hefeeda, Habib, Botev, Xu, Bhargava, MM 2003

[7] A comparative Study of Multicast Protocols : Top, Bottom, or in the middle ?, Lao, Cui, Gerla, Maggiorini, Technical Report TR040054 UCLA, Janvier 2005

[8] On Peert-to-Peer Media Streaming, Xu, Heffeda, Hambrusch, Bharargave, ICDCS 2002,Juillet 2002 (Acceptance : 18%).

[9] Pastry : Scalable, decentralized, object location and routing for large scale p2p systems, Rownstron, Druschel ; IFIP 2001

[10] Chord : A Scalable P2P Lookup Service for Internet Application, Stoica, Morris, Karger, Kaashoek, Balakrishnan, SIGCOMM 2001

[11] CAN : A Scalable Content Adressable Network, Ratnassamy, Francis, Handley, Karp, SIGCOMM 2001

[12] PALS : P2P Adaptive Layered Streaming, Rejaie, Ortega, NOSSDAV 2003

[13] PeerStreaming : A Practical Receiver-Driven P2P Media Streaming System, Li

[14] PRM : Resilient Multicast Using Overlays, Banerjee, Seungjoon, Bhattacharjee, Srinivasan, SIGMETRICS 2003

[15] MetaStream : Topology-Aware P2P On-Demand Streaming, Zhang, Butt, Hu, IFIP 2005

[16] DONET : Data-Driven Overlay Streaming : Design, Implementaion, and Experience, Zhang, Liu, Li, Yum, INFOCOM 2005

[17] bitTorrent : Incentives Build Robustness in BitTorrent, Cohen, 2003

[18] A Comparative Study of an application Layer Multicast Protocols, Banerjee, Bhattacharjee, 2003

[19] Overview of Overlay Multicast Protocoles, Dennis M. Moen, http ://netlab.gmu.edu/XOM, 2004

[20] Deployment issues for the IP multicast service and architecture, Diot, Levine, Lyles, Kassem, Balensiefe, IEEE Network p78-88, 2000

[21] Chainsaw : Eliminating Trees from Overlay Multicast, Pai, Kumar, Tamilmani, Sambamurthy, Mohr, IPTPS 2005

[22] Araneola : A Scalable Reliable Multicast System for Dynamic Environments, 3rd IEEE International Symposium on Network Computing and Applications (IEEE NCA), pages 5-14, Août/September 2004

[23] Efficient epidemic-style protocols for reliable and scalable multicast, Gupta, Kermarec, Ganesh, SRDS 2002

[24] SplitStream : High-Bandwith Multicast in Cooperative E,vironnements, Castro, Druschel, Kermarrec, Nandi, Rowstron, Singh, SOSP Octobre 2003

[25] An extendible Open Source P2P Simulator, Sam Joseph, P2P Journal, Novembre 2003

[26] 3LS - A Peer-to-Peer Network Simulator, Ting, Deters, P2P 2003

[27] Mapping Peer Behavior to Packet-level Details : A framework for Packet-level Simulations of Peer-to-Peer Systems, MASCOTS 2003

[28] Modeling and Simulation of Adhoc/P2P Resource Sharing, Kant, Iyer

[29] Narses : A Scalable Flow_Base Networks Simulator, Novembre 2002

[30] Simulating a File Sharing P2P Network, First Workshop on Semantics in P2P and Grid Computing, December 2002

[31] Freenet : A Distributed Anonymous Information Storage and Retrieval System, Clarke, Sandberg, Wiley Hong, 1999

[32] Free Pastry :Peer-to-Peer Application Development Substrate, http ://freepastry.rice.edu/FreePastry, 2005

[33] http ://cvs.sourceforge.net/viewcvs.py/freenet/aurora/

[1] http ://cvs.sourceforge.net/viewcvs.py/freenet/Serapis/freenet/

[35] http ://www.isi.edu/nsnam/ns/

[36] http ://www.dei.isep.ipp.pt/docs/arpa.html

[37] RFC 2854 : The 'text/html' Media Type

[38] RFC 1771 : A Border Gateway Protocol 4 (BGP-4)

[39] RFC 1817 : CIDR and Classful Routing

[40] Bibliography Bibliography A Survey of Mobility for Ad Hoc Network Research, Camp, Boleng, Davies, in WCMC 2002

[41] http ://www.msn.com

[42] http ://www.napster.com

[44] http ://www.kazaa.com

[45] http ://www.edonkey.com

[46] http ://www.skype.com

[47] http ://www.planetlab.com

[48] http ://www.icq.com

[49] http ://www.coolstreaming.org

[50] http ://www.pplive.com

[51] http ://www.gnutella.com

[52] http ://www.cs.bu.edu/brite

[53] Slurpie : A Cooperative Bulk Data Transfert Protocol, Sherwood, Braud, Bhattacharjee, Infocom 2004