Contrôle de congestion équitable pour le multicast et interface avec le niveau applicatif

Vincent Lucas

UNIVERSITÉ DE STRASBOURG

Laboratoire LSIIT
Université de Strasbourg - CNRS
UMR *N*°7005

Thèse présentée pour obtenir le grade de
Docteur de l'Université de Strasbourg

Discipline : Informatique

Vincent Lucas

Contrôle de congestion équitable pour le multicast et interface avec le niveau applicatif

Soutenance le 21 septembre 2010

Membres du jury

Directeur :
Jean-Jacques Pansiot
Professeur, Université de Strasbourg

Rapporteurs :
Guy Leduc
Professeur, Université de Liège

François Spies
Professeur, Université de Franche-Comté

Examinateurs :
Thomas Noël
Professeur, Université de Strasbourg

Vincent Roca
Chargé de recherches, INRIA Rhône-Alpes

Benoît Hilt
Maître de Conférences, Université de Haute Alsace

À Caroline

REMERCIEMENTS

Le travail réalisé tout au long de cette thèse n'a été possible qu'avec l'aide et le soutien de nombreuses personnes. Je profite de l'occasion qui m'est donnée ici pour leur exprimer toute ma gratitude.

Je tenais en premier lieu à remercier vivement Jean-Jacques Pansiot, mon directeur de thèse, pour son encadrement et sa disponibilité. Merci Jean-Jacques de m'avoir fait partager tes connaissances, ton expérience et ta bonne humeur. Ce fut un réel plaisir de travailler pendant ces années sous ta direction.

Je souhaite remercier chaleureusement Dominique Grad et Benoît Hilt, mes coencadrants de thèse, pour m'avoir prodigué de précieux conseils et pour m'avoir apporté de nombreuses idées au cours de ce travail.

Je remercie Messieurs Guy Leduc, Thomas Noël, Vincent Roca et François Spies pour l'intérêt qu'ils ont porté à mes travaux en acceptant de faire partie de mon jury de thèse.

Je tiens également à remercier toute l'équipe Réseaux et Protocoles du LSIIT. En particulier les doctorants présents et passés, avec qui j'ai beaucoup appris, partagé et apprécié travailler. Merci à Alexander, Damien, Émil, Julien, Julien, Mickaël, Pascal et Romain, pour votre disponibilité et votre gentillesse.

Enfin, je remercie également ma famille et mes amis qui m'ont toujours soutenu, encouragé et sans qui je n'écrirais pas ces lignes.

À tous,

Merci

RÉSUMÉ

Les communications de groupe telles que le multicast IP procurent un moyen efficace pour transmettre simultanément la même donnée vers plusieurs destinataires, comme par exemple pour la diffusion de l'IPTV. Or, il existe une demande pour utiliser le multicast sur des réseaux à capacité variable ou limitée tels que ceux des utilisateurs abonnés à l'ADSL ou ceux utilisés pour les usages mobiles avec un accès sans fil. Dans ces conditions, il est nécessaire de créer un mécanisme de contrôle de congestion pour le multicast capable de partager équitablement la bande passante entre les flux multicast et les flux TCP concurrents.

Cette thèse commence par une étude de la latence du temps d'adhésion aux groupes multicast ainsi que son influence sur les différents protocoles existants de contrôle de congestion pour le multicast. Ainsi, nous proposons de créer M2C un protocole de contrôle de congestion prenant en compte cette latence du temps d'adhésion et capable de supporter le passage à très grande échelle. Côté source, M2C utilise des canaux dynamiques semblables à ceux de WEBRC et chaque récepteur M2C dispose d'une fenêtre de congestion mise à jour par des mécanismes de "Slow Start" et de "Congestion Avoidance" qui permettent d'obtenir de façon robuste un partage équitable de la bande passante. De plus, pour converger rapidement vers le débit équitable, un algorithme de "Fast Start" est utilisé au démarrage des sessions M2C. Par ailleurs, un mécanisme permettant de détecter les évolutions du débit équitable réactive le "Slow Start" afin de converger plus rapidement vers le nouveau débit équitable. Le protocole M2C a été implémenté et évalué sur plusieurs environnements complémentaires : aussi bien sur une plateforme locale spécialement mise en place pour ces tests au sein du laboratoire, qu'à travers Internet entre Strasbourg (France), Trondheim (Norvège) et Louvain-la-Neuve (Belgique). L'évaluation se déroule selon des scenarii rigoureux et réalistes : comme par exemple, la concurrence entre une population variable de flux M2C et TCP, avec ou sans bruit de fond tel que les communications courtes produites par la navigation sur des sites web ou la consultation de courriel. Cela permet de mettre en exergue le comportement de M2C grâce à l'évaluation

de métriques telles que l'utilisation de la bande passante, le taux de pertes, l'équité, etc.

Les canaux dynamiques utilisés par M2C peuvent rendre difficile le développement d'applications. En effet, la source doit être capable d'ordonner ses données en utilisant ces canaux dynamiques dont le débit est recalculé pour chaque paquet envoyé et en permettant à chaque récepteur ayant un débit différent de tirer parti de toutes les données reçues. Ces problèmes sont traités en proposant un séquenceur et une interface de programmation (API) qui permettent d'envoyer les données les plus importantes aux débits reçus par la majorité des récepteurs. Nous avons montré que l'efficacité de ce séquenceur est telle qu'un récepteur recevant N% du débit de la source, alors ce récepteur reçoit les N% les plus importants des données applicatives. De plus, l'API fournie permet à l'application de s'abstraire de connaître les mécanismes utilisés par le séquenceur et par M2C. Cette API est à la fois simple d'utilisation et très efficace pour différents types d'applications. Ainsi, nous l'avons utilisé pour implémenter des logiciels de transfert de fichiers [MUTUAL] et de diffusion de vidéos [MUST], dont les sources sont disponibles publiquement ainsi que celles de M2C :

- [M2C] `http://svnet.unistra.fr/mcc/`
- [MUTUAL] `http://mutual.sourceforge.net/`
- [MUST] `http://must.sourceforge.net/`

ABSTRACT

IP multicast is an efficient solution able to deliver a data simultaneously to several receivers : i.e. IPTV streams. Nonetheless, there is a need to use multicast streams on limited bandwidth capacity networks, such as ADSL ones or wireless access. Thereby, a multicast congestion control mechanism is mandatory in order to fairly share the bandwidth between the competing multicast and TCP streams.

In this thesis we first study the multicast join time latency as well as its impact on multicast congestion control protocols. Thereby, we propose M2C a new scalable multicast congestion control protocol able to deal with the join time latency. M2C uses a WEBRC-like dynamic layering at the source part and each M2C receiver manages its reception rate with a congestion window controlled by a "Slow Start" and a "Congestion Avoidance" mechanisms. Thanks to these mechanisms, M2C behavior is robust and TCP-friendly. Moreover, to quickly converge to the fair rate, M2C is able to return in "Slow Start" when a receiver detects that the current rate is far below the fair one. In addition, a "Fast Start" algorithm is designed to speed up M2C startup. We have implemented M2C and studied its behavior over several network configurations : on a local testbed specially deployed within our laboratory, as well as an interdomain testbed through Internet between Strasbourg (France), Trondheim (Norway) and Louvain-la-Neuve (Belgium). The evaluation is driven by several rigorous and realistic scenarios : i.e. competition between a variable population of M2C and TCP streams, with or without background noise such as short communications generated by web navigation or email browsing, etc. These numerous experiments highlight the good M2C behavior in terms of bandwidth usage, loss rate, fairness, etc.

Besides, it is a challenging task for application programmers to efficiently take advantage of the dynamic channels used by M2C. Indeed, the source must be able to cope with the dynamic channels whose sending rate is updated for each new packet. Moreover, the source has to schedule the application data to allow each receiver to take advantage of the whole received data and thus

independently of the reception rate. To deal with these problems, we propose a sequencer and its API, which assure that the more important the data is, the lower its sending rate is. And thus acquired by the majority of receivers. Experiments have shown the sequencer optimal performances insuring that a receiver subscribed to N% of the source rate obtains the most important N% of the applicative data. Moreover, applications can use this API without knowing anything about the dynamic layering used by M2C. Therefore, this API is easy to use, which makes it suitable for various kind of applications. Indeed, we have demonstrated the usefulness of the API and the underlying congestion control protocol by implementing two experimental applications : a file transfer [MUTUAL] and a video streaming software [MUST]. All source codes are publicly available :

- [M2C] `http://svnet.unistra.fr/mcc/`
- [MUTUAL] `http://mutual.sourceforge.net/`
- [MUST] `http://must.sourceforge.net/`

TABLE DES MATIÈRES

INTRODUCTION GÉNÉRALE

Les travaux présentés dans cette thèse ont pour objectif de créer des mécanismes permettant une utilisation équitable de la bande passante pour des communications de groupe à grande échelle. Plus particulièrement, nous nous sommes intéressés à la création d'un protocole de contrôle de congestion pour le *multicast* utilisant une fenêtre de congestion dont les bénéfices apportés se traduisent par un comportement robuste et équitable envers les autres flux *multicast* et les flux *Transmission Control Protocol (TCP)* concurrents. Cette thèse a notamment donné lieu à l'implémentation et l'expérimentation de ce protocole et d'une interface de programmation simplifiant la réalisation d'applications pour ce type de communication.

CONTEXTE

La diffusion de données dans *Internet* est un problème complexe, en particulier pour les applications qui ont des contraintes de temps, telles que les applications de visioconférence, jeux en réseau, etc. De plus, des contraintes de bande passante viennent s'ajouter pour les applications où le nombre de récepteurs est important, telles que la télévision sur *Internet Protocol (IP)*, ou *IP TeleVision (IPTV)*, l'échange de fichiers pour des applications scientifiques ou bancaires.
L'un des principaux problème de communication pour ces applications réside dans le fait que le nombre de copies de la même donnée transitant sur le réseau correspond au nombre de récepteurs. Par exemple, si une centaine de personnes regardent une chaîne de télévision sur *Internet*, cent copies de cette dernière circuleront sur le réseau.
Pour résoudre ce type de problème, une architecture basée sur le routage *multicast* au niveau *IP* a été proposée. Elle permet à la source de ne diffuser qu'une seule copie des données et charge les routeurs d'en faire la duplication au plus proche des récepteurs.
Ainsi, les communications de groupe procurent un moyen efficace pour transmettre simultanément la même donnée vers plusieurs destinataires. Par exemple, les fournisseurs d'accès à Internet plé-

biscitent largement ce type de communication pour la diffusion de l'*IPTV* en combinaison avec l'*User Datagram Protocol (UDP)*. Comme *UDP* n'offre pas de mécanisme de contrôle de congestion, ces flux risquent de saturer le réseau. C'est pourquoi, ils ne sont pour le moment utilisés qu'au sein de réseaux surdimensionnés. Cependant, il existe une demande pour utiliser le *multicast* sur des réseaux à capacité variable ou limitée tels que ceux des utilisateurs abonnés à l'*Asymmetric Digital Subscriber Line (ADSL)* ou ceux utilisés pour les usages mobiles avec un accès sans fil [Sarikaya 09]. Dans ces conditions, il est nécessaire que les flux *multicast* soient capables de partager équitablement la bande passante entre eux et avec les flux *TCP* concurrents. L'utilisation d'un protocole de contrôle de congestion est donc nécessaire, car les flux *multicast* ne peuvent pas utiliser le protocole *TCP*, ce dernier n'étant pas conçu pour des récepteurs multiples.

Principes du contrôle de congestion *multicast*

Différents protocoles de contrôle de congestion pour le *multicast* ont été proposés afin de partager équitablement la bande passante avec *TCP*. Parmi ces travaux, nous nous sommes focalisés principalement aux mécanismes dont l'efficacité ne dépend pas du nombre de récepteurs, c'est-à-dire aux protocoles supportant le passage à très grande échelle.
La première solution proposée dans la littérature utilise un contrôle de congestion à débit unique pour le groupe de récepteurs : c'est-à-dire que la source émet un unique flux que chaque récepteur reçoit dans sa totalité. Par exemple, *Pragmatic General Multicast Congestion Control (PGMCC)* [Rizzo 00] propose que la source émette au débit qui correspond au débit qu'obtiendrait *TCP* pour le récepteur disposant de la bande passante la plus petite. Quant à la retransmission de paquets perdus, elle se fait par l'envoi d'acquittements négatifs, ou *Negative ACKnowledgement (NACK)*, que tout récepteur peut envoyer. Cette situation limite tous les récepteurs à la bande passante du pire récepteur et provoque ainsi un problème de décroissance du débit en fonction du nombre de récepteurs. Par conséquent, ce type de protocole ne supporte pas le passage à l'échelle car les performances obtenues dépendent du nombre de récepteurs. Outre ce problème, toute méthode de contrôle de congestion *multicast* qui nécessite une signalisation entre les récepteurs et la source, comme ici les *NACK*, ne peut pas être extensible. En effet, plus le nombre de récepteurs augmente, plus le nombre de messages envoyés à la source augmente également, ce qui peut à la longue provoquer un phénomène d'explosion du nombre d'acquittements reçus par la source.
Pour résoudre les problèmes des protocoles à débit unique, plu-

sieurs solutions telles que *Receiver driven, Layered Congestion control (RLC)* [Vicisano 98] ou *Fair Layered Increase/Decrease with Static Layering (FLID-SL)* [Luby 00] proposent que la source décompose son flux en plusieurs couches correspondant chacune à un groupe *multicast*. Ainsi, chaque récepteur peut s'abonner au nombre de groupes *multicast* qui correspond à sa capacité réseau. Par exemple, pour un flux vidéo ajouter un canal augmente la qualité de la vidéo reçue. Dans le cas de la transmission fiable, comme avec *File Delivery over Unidirectional Transport (FLUTE)* [Peltotalo 07], chaque canal supplémentaire augmente le débit du transfert et réduit le temps de téléchargement. Ces abonnements se font donc hiérarchiquement et de façon cumulative, afin que les données obtenues par un récepteur comprennent celles de tout autre récepteur abonné à un débit moindre. Ainsi, le mécanisme de contrôle de congestion est dirigé par chaque récepteur qui gère son propre débit et n'a donc pas besoin d'envoyer de message à la source, ce qui permet de supporter le passage à grande échelle.
Idéalement, cette régulation du débit devrait pouvoir être modulée instantanément. Mais l'utilisation de la signalisation *multicast* pour les abonnements et désabonnements ajoute un délai de traitement loin d'être négligeable.
L'utilisation de canaux dynamiques, comme proposée par *Fair Layered Increase/Decrease with Dynamic Layering (FLID-DL)* [Byers 02], permet de s'affranchir des lenteurs de désabonnement d'un groupe *multicast*. Pour cela, la source émet de façon constante uniquement sur le canal de base. Les autres canaux diminuent régulièrement leur débit d'un facteur donné. Le temps est alors divisé en cycles réguliers et à chaque nouveau cycle un canal est créé au débit maximum et le canal dynamique de débit le plus faible se termine et devient muet. Un récepteur peut ainsi réduire son débit en attendant simplement que les canaux diminuent d'eux-mêmes leur débit. Ainsi, un récepteur ne peut se désabonner que d'un groupe devenu muet. Comme ce groupe n'émet plus de paquet, cela permet de ne pas avoir à se soucier du temps de désabonnement. Si le récepteur ne fait rien d'autre, son débit diminue. S'il veut conserver son débit, il doit s'abonner à un nouveau canal à chaque nouveau cycle. Et s'il veut augmenter son débit, il doit s'abonner à un canal supplémentaire avant qu'un nouveau cycle ne commence.

Étude et évolution du contrôle de congestion pour le *multicast*

Les propositions les plus avancées, comme *Wave and Equation Based Rate Control (WEBRC)* [Luby 02], utilisent de tels canaux dynamiques et prennent en compte l'équité des débits obtenus avec

ceux de *TCP*. Notre recherche a commencé par l'étude de l'équité de ces mécanismes et par leurs implémentations [Lucas 10a] afin de pouvoir les tester hors du simulateur. Cela nous a permis de mettre en évidence plusieurs dysfonctionnements de ces protocoles, dont le plus important est de considérer le temps d'abonnement comme étant représentatif du délai aller-retour des données. Or le temps d'abonnement est sujet à un temps de traitement non négligeable de la signalisation du routage *multicast* pour s'abonner à un groupe. Nous avons montré [Lucas 06] que ce temps de traitement peut dépasser plusieurs secondes sur des réseaux réels dont le temps aller-retour pour des données ne dépasse pas les 50 ms. Ce résultat nous a permis de développer une nouvelle proposition exempte de ces problèmes. Notre solution nommée *Multicast Congestion Control (M2C)* s'inspire des mécanismes de *TCP* permettant une régulation équitable du débit dirigée par une augmentation additive et une réduction multiplicative [Lucas 08].
Le protocole *M2C* a également été implémenté [Lucas 10a] et évalué sur plusieurs environnements complémentaires : aussi bien sur une plateforme locale spécialement mise en place pour ces tests au sein du laboratoire, qu'à travers Internet entre Strasbourg (France), Trondheim (Norvège) et Louvain-la-Neuve (Belgique). L'évaluation se déroule selon des scenarii rigoureux et réalistes : comme par exemple, la concurrence entre une population variable de flux *M2C* et *TCP*, avec ou sans bruit de fond tel que les communications courtes liées à la navigation sur des sites *web* ou la consultation de *courriel*. Cela permet de mettre en exergue le comportement de *M2C* grâce à l'évaluation de métriques telles que l'utilisation de la bande passante, le taux de pertes, l'équité, etc.
Ces expérimentations ont permis de découvrir des faiblesses de la première version de *M2C* qui souffre d'un temps de convergence trop lent vers le débit équitable. Une seconde limitation est que le mécanisme de partage équitable de la bande passante n'est pas assez robuste et donne des résultats variables suivant les configurations des réseaux traversés. Ces limitations furent donc traitées en introduisant un mécanisme d'accélération de convergence couplé à un détecteur de sous utilisation de la bande passante par rapport au débit équitable. En effet, l'analyse détaillée du comportement de *TCP* a permis de définir une estimation [Lucas 09c] précise pour calculer quand une congestion devrait survenir dans une situation de partage équitable de la bande passante. Si aucune congestion n'est détectée pendant le temps estimé, cela signifie que le flux est actuellement largement inférieur au débit équitable et qu'il est donc possible d'utiliser un comportement plus agressif pour converger plus rapidement vers le débit équitable. La validation exhaustive de cette nouvelle version du protocole *M2C* montre que ce dernier est robuste, équitable et converge rapide-

ment vers le débit équitable.

Utilisation du contrôle de congestion *multicast* par les applications

L'utilisation de canaux dynamiques peut rendre difficile le développement d'applications, notamment en ce qui concerne la source qui doit être capable :

- D'utiliser les canaux *multicast* dynamiques dont la variation du débit de chaque groupe *multicast* est calculée pour chaque paquet envoyé.
- D'ordonner ses données pour que chaque récepteur ayant un débit différent puisse utiliser et tirer parti de toutes les données reçues.

En d'autres termes, est-il possible d'envoyer les données de façon unique côté source de telle sorte que chaque récepteur ne reçoive que les données les plus importantes correspondant à ses capacités ?
Pour résoudre ce problème, nous proposons un séquenceur [Lucas 10b] gérant cette complexité. Ce séquenceur répertorie tous les paquets à envoyer pendant la durée d'envoi du tampon applicatif. Ces paquets sont triés par débit afin que les données les plus importantes soient envoyées aux débits les plus faibles. En effet, comme les différents récepteurs s'abonnent de façon hiérarchique et cumulative aux canaux *multicast*, les canaux de plus faible débit sont reçus par la majorité des récepteurs.
Ainsi, ce séquenceur permet de proposer aux applications une interface de programmation (API) très simple nécessitant de ne fournir qu'un tampon de données trié de façon hiérarchique : les données les plus importantes d'abord. L'application est libre de choisir la hiérarchie utilisée, qui n'a pas besoin d'être connue du séquenceur. Inversement l'application peut s'abstraire de connaître tous les mécanismes utilisés pour le contrôle de congestion et notamment celui des canaux dynamiques. Nous avons montré que l'efficacité de ce séquenceur est telle que pour un récepteur recevant N% du débit de la source, alors ce récepteur reçoit les N% les plus importants des données du tampon utilisé par l'application source. Ce séquenceur ainsi que son API ont été implémentés et sont disponibles publiquement [Lucas 10a].
Cette API est à la fois très efficace et simple d'utilisation pour différents types d'applications. Nous avons ainsi implémenté :

- Un transfert de fichier en carrousel [Lucas 10b] [Lucas 09b] qui utilise un ordonnanceur de bloc au niveau applicatif couplé à des codes de corrections de pertes basé sur des matrices à basses densités.

- Une application de diffusion vidéo [Lucas 09a]. En plus du séquenceur, le logiciel de diffusion vidéo utilise au niveau applicatif un encodage hiérarchique couplant une transformation en ondelettes discrètes à une transformation par cosinus discrète.

Plan de thèse

En première partie, cette thèse commence par un état de l'art sur les modèles de communications et sur les mécanismes de contrôle de congestion pour l'*unicast* et le *multicast*.
Ensuite en seconde partie, nous proposons de créer un protocole de contrôle de congestion pour le *multicast* efficace et équitable. Pour cela, nous commençons par évaluer les protocoles existants avant de proposer une solution que nous améliorons incrémentalement.
Enfin en troisième partie, nous nous intéressons à l'organisation des données qu'une application doit fournir pour utiliser ces protocoles de contrôle de congestion pour le *multicast*. Ainsi, nous commençons par définir un *séquenceur* couplé à une *Application Programming Interface (API)* pour faciliter l'utilisation de ces protocoles par les applications. Puis, nous étudions la capacité de 2 types d'applications sensibles aux pertes ou aux délais à utiliser l'*API* proposée.

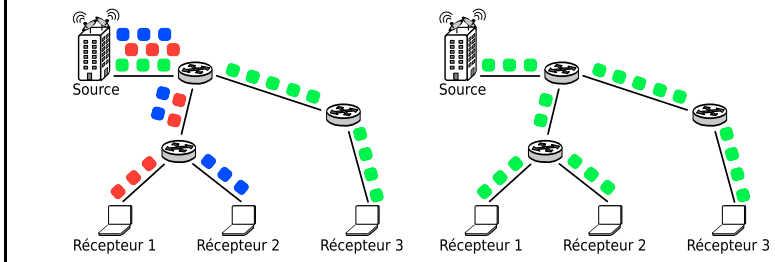
PROBLÉMATIQUE ET ÉTAT DE L'ART

1 MODÈLES DE COMMUNICATIONS

Afin d'illustrer les différents modèles de communication utilisés dans les réseaux *IP*, nous allons illustrer notre propos en prenant l'exemple d'un flux télévisuel populaire diffusé par Internet. Dans ce cas, plusieurs récepteurs souhaitent obtenir les mêmes données au même moment. La source envoie une copie des données pour chaque récepteur du flux : ce type de communication où une copie des données est envoyée à chaque récepteur est appelé *unicast* (cf. figure 1.1(a)). Pour diminuer le nombre de ces copies redondantes, un modèle de communication de groupe a été proposé : le *multicast*. En effet, le *multicast* permet de n'envoyer qu'une seule copie des mêmes données vers tous les récepteurs (cf. figure 1.1(b)).

1.1 L'*unicast*

L'*unicast* est une communication de 1 vers 1, qui permet à 2 machines de dialoguer pour échanger des informations. Les données sont envoyées dans des paquets ayant pour identifiant l'adresse *IP* du récepteur. Cette adresse *IP* sert également de localisateur et permet aux routeurs qui interconnectent les machines de propager ces paquets sur le lien correspondant à la route la plus courte vers le récepteur. Ces routes sont calculées grâce à des algorithmes de routage tel que *Routing Information Protocol (RIP)*, *Open Shortest*



(a) Flux diffusé en *unicast* (b) Flux diffusé en *multicast*

FIGURE 1.1 – Exemple de transfert en *unicast* et en *multicast*

Path First (OSPF), *Intermediate System to Intermediate System (IS-IS)* ou *Border Gateway Protocol (BGP)*. La relative simplicité de l'*unicast*, lui a permis de se développer notamment autour du modèle client-serveur sur lequel se basent la plupart des services sur Internet : par exemple la navigation *web*, la messagerie électronique, etc.
Cependant, pour communiquer les mêmes informations avec plusieurs récepteurs, une machine source doit utiliser plusieurs flux dont les paquets ont pour seule différence l'adresse destination pour chaque récepteur. La source va donc envoyer une copie des données vers chaque récepteur et chaque copie traverse ainsi divers équipements d'interconnexions. Ainsi, ces copies provoquent un surcoût inutile de l'utilisation des ressources de la source et des équipements d'interconnexion.

1.2 LE *multicast*

Le *multicast* permet de transmettre une information vers plusieurs récepteurs en n'émettant qu'une seule copie de cette information. Les applications d'audio ou de vidéo conférence, l'*IPTV*, la réplication de données depuis des sites miroirs, sont autant d'utilisations possibles permettant de tirer profit de ce mode de communication.
La notion de *multicast* existe à plusieurs niveaux, comme pour *Ethernet* ou sur des circuits virtuels comme *Asynchronous Transfer Mode (ATM)*. Dans la suite de cette thèse, nous allons uniquement nous intéresser au *multicast IP* (version 4 ou 6) et plus particulièrement au *multicast* en mode épars basé sur le protocole *Protocol Independent Multicast - Sparse-Mode (PIM-SM)* [Fenner 06].

1.2.1 LE *multicast IP*

L'utilisation du *multicast* au niveau *IP* est possible grâce à des adresses *IPv4* [Albanna 01] et *IPv6*[Hinden 06] particulières, dites adresses *multicast* ou adresses de groupe. Une adresse *multicast IP* sert d'identifiant de groupe et chaque machine souhaitant recevoir ce groupe doit s'abonner à cette adresse *multicast* et l'ajouter à chaque interface abonnée au groupe.
Ensuite, chaque machine doit envoyer des messages *Internet Group Management Protocol (IGMP)* [Cain 02] en *IPv*4 ou *Multicast Listener Discovery (MLD)* [Vida 04] en *IPv*6 pour s'abonner ou se désabonner aux différents groupes. Ainsi en écoutant ces messages les routeurs enregistrent les interfaces (*Outgoing InterFace (OIF)*) par lesquelles les machines s'abonnent à des groupes *multicast*. Puis les routeurs propagent de proche en proche ces abonnements avec le protocole de routage *PIM-SM* [Fenner 06] afin de créer un arbre de diffusion *multicast*. Cette propagation se fait selon les routes

unicast en utilisant les interfaces (*Incoming InterFace (IIF)*) menant vers une racine connue. La racine de l'arbre *multicast* peut être de 2 types différents selon le modèle de communication *multicast* utilisé : *Any-Source Multicast (ASM)* ou *Source-Specific Multicast (SSM)*.
Une fois l'arbre *multicast* construit, la source n'envoie qu'une seule copie des données et les routeurs ont la charge de dupliquer ces données quand plusieurs destinataires sont sur des chemins différents. Ainsi, l'utilisation de ressources des équipements d'interconnexion est optimale. Le *multicast* supprime donc le problème de baisse de performances de l'*unicast* en fonction du nombre de récepteurs d'un même flux. Cette indépendance du nombre de récepteur est la capacité principale du *multicast* appelée passage à l'échelle ou *scalabilité*. Bien que plus récent, nous présenterons d'abord le modèle *SSM* du fait de sa plus grande simplicité.

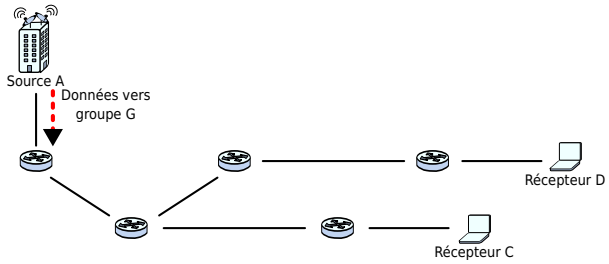
1.2.1.1 LE *Source-Specific Multicast* (*SSM*)

La notion de *SSM* a été introduit par Holbrook [Holbrook 99] [Holbrook 01] où une seule et unique source peut communiquer avec un groupe de récepteurs. Ainsi, la source constitue la racine de l'arbre *multicast* qui est appelé : arbre par source.
La figure 1.2 donne un exemple du fonctionnement d'une diffusion en *SSM* :

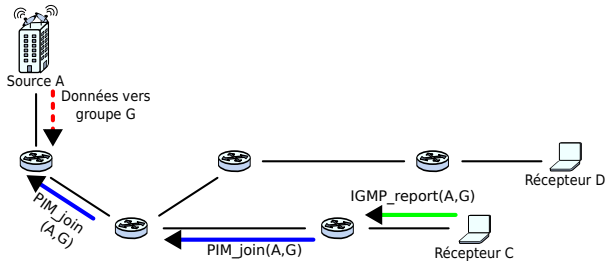
1. Pour chaque groupe, une source émet des paquets à destination d'une adresse *IP multicast* donnée. La source ignore totalement si 0, 1, 2 ou *N* récepteurs sont abonnés au groupe. Il n'est donc pas prévu de faire remonter les demandes d'abonnement jusqu'à la source elle-même. Ainsi même s'il n'y a pas de récepteur, la source envoie les paquets pour le groupe *multicast* qui arrivent par défaut jusqu'au routeur de son réseau local (cf. figure 1.2(a)).

2. Pour s'abonner au groupe *multicast*, un récepteur émet des demandes, ou *report IGMP/MLD* [1], qui spécifient les sources "*S*" et groupes "*G*" *multicast* auxquels le récepteur veut s'abonner ou se désabonner. Dès que le routeur du réseau local du récepteur reçoit cette demande, il marque l'interface sur laquelle il vient de recevoir le message comme étant abonnée au groupe *multicast* demandé (*OIF*).
   Puis, les messages d'abonnements sont propagés de proche en proche en direction de la source jusqu'à atteindre un routeur déjà abonné au groupe ou le routeur du réseau local de la source (cf. figure 1.2(b) et 1.2(d)). Cette propagation

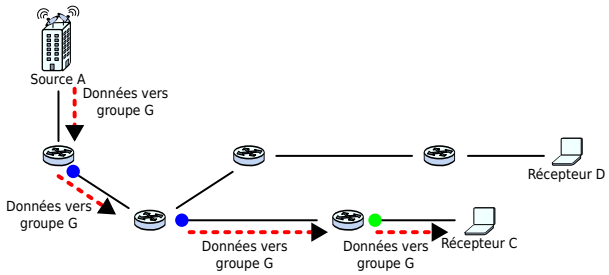
1. En *SSM IGMPv*3 ou *MLDv*2 est nécessaire, car seules ces versions permettent de spécifier une source pour un groupe *multicast*.



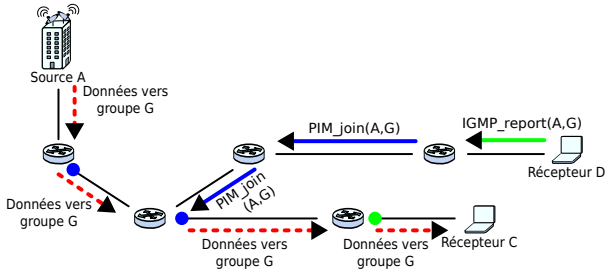
(a) Source sans récepteur



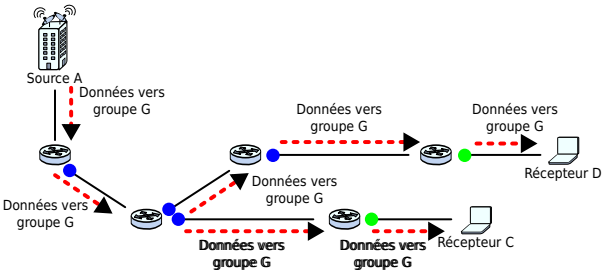
(b) Abonnement du récepteur *C*



(c) Propagation des données vers le récepteur *C*



(d) Abonnement du récepteur *D*



(e) Propagation des données vers les récepteurs *C* et *D*

FIGURE 1.2 – Exemple de fonctionnement du *multicast* en *SSM*

se base sur le routage *unicast* et utilise un protocole de routage *multicast*, dont le plus usité actuellement est le protocole *PIM-SM* [Fenner 06]. Comme pour *IGMP* et *MLD*, *PIM-SM* dispose de messages d'abonnement, ou *join*, et de désabonnement, ou *prune*, pour une ou plusieurs sources et groupes *multicast*. Les chemins entre les différents récepteurs et la source forment un arbre, appelé *arbre par source* dont la racine est la source.

3. Ensuite, les paquets sont propagées en suivant les chemins de l'*arbre par source* calculés lors des abonnements des différents récepteurs (cf. figure 1.2(c) et 1.2(e)). En particulier, si un routeur doit propager des données sur plusieurs liens en même temps (*OIF*), il se charge de faire les duplications nécessaires.

En *SSM*, les adresses *multicast* sont relatives à l'adresse de la source et correspondent donc à un couple : adresse source et adresse de groupe. Ainsi, en *SSM* chaque machine source gère ses propres adresses *multicast*. Les adresses *IP multicast* en *SSM* constituent par défaut les plages `232.0.0.0/8` en *IPv*4 et `FF30:0000::/32` en *IPv*6.

1.2.1.2 L'*Any Source Multicast* (*ASM*)

L'*ASM* est le premier modèle de communication de groupe pour le protocole *IP* apparu avec la thèse de Deering [Deering 91] en 1991, soit une dizaine d'année avant le modèle *SSM*. Les adresses utilisées pour l'*ASM* correspondent aux adresses de classe *D* soit la plage `224.0.0.0/4` en *IPv*4 et `FF00::/8` en *IPv*6, auxquelles il faut retirer les plages dorénavant utilisées pour *SSM*. Pour le modèle *ASM*, la définition de Deering est plus générale que celle faite par Holbrook et permet la communication entre *N* sources et *M* récepteurs.
Prenons l'exemple du modèle *ASM* associé au protocole *PIM-SM*. La racine de l'arbre *multicast* est un routeur servant de point de rendez-vous, ou *Rendez-vous Point (RP)*. D'un côté les abonnements vont donc vers cette racine et de l'autre côté les sources créent des tunnels jusqu'à ce même *RP*. Ainsi le *RP* désencapsule les messages reçus et les propage sur l'arbre *multicast* suivant les informations de routages obtenues grâce aux messages des protocoles *PIM-SM*, *IGMP* ou *MLD*.
Concernant les sources, la figure 1.3 donne un exemple du fonctionnement d'une diffusion en *ASM* :

1. Comme pour le *SSM*, une source émet pour chaque groupe des paquets à destination d'une adresse *IP multicast* donnée et ce même si aucun récepteur n'est abonné au groupe.

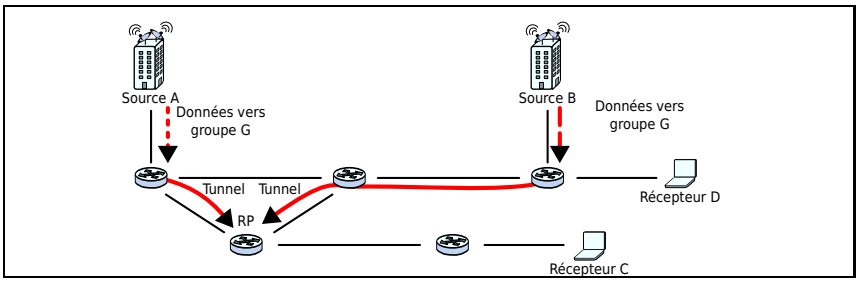
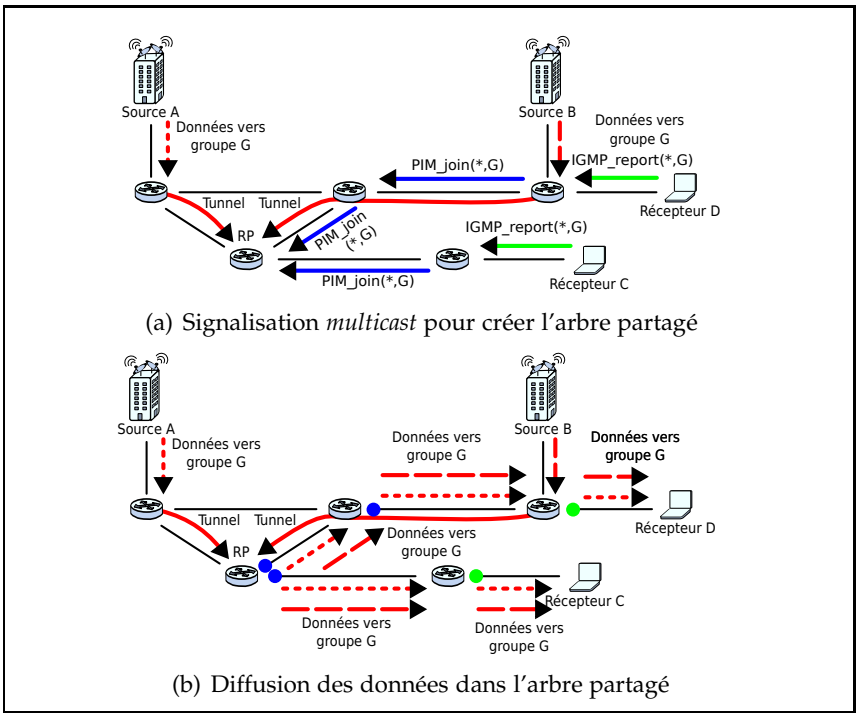


FIGURE 1.3 – Exemple de fonctionnement des sources en *ASM*



(a) Signalisation *multicast* pour créer l'arbre partagé

(b) Diffusion des données dans l'arbre partagé

FIGURE 1.4 – Exemple de fonctionnement des récepteurs en *ASM*

2. Les paquets *multicast* arrivent jusqu'au routeur du réseau local de la source, qui doit déterminer l'adresse du *RP* et lui propager les paquets *multicast* reçus via un tunnel en utilisant des messages *PIM_register*. Ainsi, plusieurs sources d'un même groupe *multicast* peuvent envoyer leurs données au même *RP*.

Ensuite concernant les récepteurs, la figure 1.4 donne un exemple du fonctionnement d'une réception en *ASM* :

1. Les récepteurs utilisent les mêmes mécanismes qu'en *SSM*[2] pour s'abonner ou se désabonner d'un groupe. Pour un groupe

2. À ceci prêt que *ASM* peut fonctionner avec une version précédente d'*IGMP* ou de *MLD*.

multicast "*G*", un récepteur en *ASM* ne s'abonne pas à une source spécifique, mais à toutes les sources "$*$" du groupe (cf. figure 1.4(a)).
Ensuite, le routeur propage cette demande au routeur suivant en direction du *RP* en utilisant le protocole *PIM-SM*. Cet abonnement remonte de routeur en routeur jusqu'à atteindre un routeur déjà abonné au groupe ou le *RP*. Ces différents chemins abonnés entre chaque récepteur et le *RP* forment un arbre, dit "*partagé*", ayant pour racine le *RP*.

2. Les données de chaque source pour le groupe *G* sont reçues par le *RP*. La diffusion depuis le *RP* via l'*arbre partagé* se déroule de la même manière que pour un *arbre par source* (figure 1.4(b)).

Il peut alors arriver que l'*arbre partagé* ne soit pas optimal. Par conséquent en reprenant à partir de l'exemple précédent (figure 1.4), le *RP* et les routeurs directement connectés aux récepteurs peuvent apprendre l'adresse *IP* de la source dès lors qu'un paquet de donnée est reçu. Ces routeurs peuvent alors s'abonner à un *arbre par source* et filtrer cette source sur l'*arbre partagé* une fois que des données sont reçues par cet arbre par source, afin de supprimer les paquets reçus en double.

1.2.2 LE *multicast* APPLICATIF OU OVERLAY

Le faible déploiement du *multicast IP* limite son utilisation. Et comme le *multicast IP* est peu utilisé, les différents fournisseurs d'accès ne désirent pas l'activer. Afin de briser cette boucle sans fin, plusieurs propositions furent faites pour utiliser le *multicast* dans une version applicative ou overlay.
Un réseau overlay constitue une surcouche permettant d'effectuer des opérations de routage au niveau applicatif. Ce sont donc les machines clientes qui se chargent du routage applicatif et notamment des duplications de paquets vers les différents récepteurs. Bien entendu, l'efficacité d'un tel réseau réside dans la répartition de l'utilisation des ressources afin de limiter le nombre de copies sur la même machine et ainsi d'améliorer l'efficacité du système en terme de bande passante. Outre sa facilité de déploiement, la création d'un réseau *multicast* overlay permet également d'utiliser directement le protocole *TCP* entre deux pairs et ainsi de disposer d'un mécanisme de contrôle de congestion de proche en proche.
En revanche, les réseaux overlay sont moins performant du fait que :

- Les paquets n'empruntent pas forcément le plus court chemin (*stretch*).
- Plusieurs copies d'un paquet peuvent traverser les mêmes liens (*stress*).

Par conséquent, la difficulté des réseaux overlay est de construire et de maintenir un réseau efficace sans perte de connectivité.
De multiples propositions pour le *multicast* applicatif ont été formulées. Certaines collectent des informations de façon centralisée comme *Host-Based Multicast (HBM)* [Roca 01] qui utilise un *RP* virtuel rassemblant toutes les informations sur les noeuds et lui permettant de calculer une topologie optimale. Pour augmenter la stabilité de la connectivité, *HBM* entretient également un taux de liens virtuels redondants associés à un mécanisme de détection de boucles. Du fait de sa centralisation, ce protocole est bien entendu simple d'utilisation mais est également limité du point de vue du passage à l'échelle.
Un grand nombre d'autres travaux ont été réalisés au sujet du *multicast* overlay. Cependant, nous nous limiterons à cette courte présentation car la suite de cette thèse ne concerne que le *multicast* natif.

1.2.3 LE *multicast* HYBRIDE

D'autres propositions visent davantage à instaurer un déploiement incrémental du *multicast IP*. C'est le cas de *Automatic IP Multicast Without Explicit Tunnels (AMT)* [Thaler 10], qui propose de créer des machines relais au sein du réseau *multicast* natif. Ces relais servent à fournir au moyen de tunnels *unicast* une connectivité *multicast* à des réseaux qui en sont isolés, appelés site *AMT*. Pour cela, chaque site *AMT* doit disposer d'une machine servant de passerelle qui crée le tunnel avec le relais le plus proche identifiable grâce à une adresse *anycast*.

1.3 CONCLUSION

La hausse de la consommation de la bande passante a créé une demande de protocoles capables de transmettre des données de 1 vers N. Ainsi, le *multicast IP* s'est notamment imposé comme solution pour pouvoir proposer la télévision sur Internet.
Cependant, le déploiement du *multicast* a été réduit a cette utilisation où seuls les flux du propriétaire du réseau peuvent circuler en *multicast*. En effet s'ils ne sont pas contrôlés, les flux *multicast* sont considérés comme potentiellement dangereux, car :

- D'une part, il manque des moyens de supervision, ce qui fait du *multicast* un service difficile et donc coûteux à garantir aux clients potentiels.
- D'autre part, il manque des mécanismes de contrôle de congestion ce qui rend les flux *multicast* peu flexibles aux changements et restrictions des conditions réseaux.

La suite de cette thèse propose d'étudier les solutions proposées et de soumettre des améliorations afin que les flux *IP multicast* puissent utiliser un mécanisme de contrôle de congestion comparable à celui de *TCP* en *unicast* et donc permettre leur cohabitation.

2

CONTRÔLE DE CONGESTION *UNICAST*

Cette thèse est principalement consacrée aux mécanismes de contrôle de congestion *multicast*. Or, ces mécanismes doivent pouvoir être déployés sur les mêmes réseaux que pour l'*unicast* et donc être compatibles. Ainsi, pour comprendre comment assurer cette compatibilité, ce chapitre détaille les méthodes et principes fondamentaux de contrôle de congestion *unicast*. En outre, certains des mécanismes proposés pour le *multicast* s'inspirent de ceux utilisés pour l'*unicast*.

2.1 PROBLÉMATIQUE

Le contrôle de congestion est un mécanisme qui permet de régir le débit d'un ou de plusieurs flux. Ce mécanisme vise généralement à répondre aux trois exigences suivantes :

- Maximiser l'utilisation de la bande passante.
- Limiter le nombre de paquets perdus, c'est-à-dire ne pas dépasser les capacités des ressources du réseau.
- Partager équitablement la bande passante entre les différents flux parcourant le ou les liens les plus lents, appelés liens faibles.

Le contrôle de congestion permet donc de fournir une qualité de service équitable entre les différents flux en essayant d'utiliser au mieux les ressources réseaux disponibles.

2.2 CONTRÔLE DE CONGESTION DE BOUT EN BOUT

Sur *Internet*, la plupart des mécanismes de contrôle de congestion sont de bout en bout : ce qui signifie que ce sont les machines terminales source et destination qui se chargent de réguler le débit utilisé. Cela fonctionne pour le moment, car les flux non-régulés ne saturent pas à eux seuls les réseaux traversés, ce qui permet aux flux régulés par un contrôle de congestion de se partager la bande passante restante.
Dans ce paragraphe, nous allons décrire les différentes versions et

améliorations du protocole le plus usité actuellement : *Transmission Control Protocol* (*TCP*) [Jacobson 88][Stevens 97]. Même si *TCP* inclut d'autres mécanismes, seuls les moyens servant à réguler le débit nous intéressent.

2.2.1 *TCP* ÉTALON : NEWRENO

TCP connaît beaucoup de variantes dont *TCP-reno* et *TCP-newReno* forment la base. Nous allons donc commencer par décrire le mécanisme de contrôle de congestion de ces versions de *TCP*.
Le contrôle de congestion de *TCP* repose sur le principe de la conservation de paquets. En effet, le but de *TCP* est d'envoyer en même temps un maximum de paquets à travers le réseau en s'assurant que ce dernier peut supporter un tel flux. Ce nombre de paquets maximum à envoyer en même temps sur le réseau forme une fenêtre de congestion, ou *Congestion Window (CW)*. La source ne peut envoyer sur le réseau qu'une quantité de paquets correspondant à la taille de sa fenêtre de congestion. Ainsi, aucun nouveau paquet ne peut plus être émis jusqu'à ce que qu'un acquittement, ou *Acknowledgment (ACK)*, vienne informer la source qu'au moins un des paquets émis à été correctement reçu et à donc quitté le réseau. Ces *ACK* sont émis par le récepteur pour un petit nombre de paquets reçus. De plus ces *ACK* sont cumulatifs, ce qui permet à la source de décaler *CW* et envoyer un ou plusieurs nouveaux paquets pour préserver l'équilibre du nombre de paquets traversant le réseau.
Le problème est de savoir comment calculer la taille idéale de cette fenêtre de congestion pour utiliser au maximum les ressources du réseau, sans pour autant le surcharger et en partageant équitablement ces ressources entre les flux concurrents. Cette découverte de la bande passante disponible se décompose en 2 phases : le démarrage lent ou *Slow Start (SS)* et l'évitement de congestion, ou *Congestion Avoidance (CA)*.

2.2.1.1 LA PHASE DE *Slow start* (*SS*)

La phase de démarrage lent, ou *SS* est utilisée quand le flux ignore totalement quelle est la bande passante disponible : c'est-à-dire au démarrage de la session, ou lorsque les conditions du réseau semblent avoir grandement changé forçant à redécouvrir la nouvelle bande passante disponible. Le *SS* est donc chargé de sonder rapidement quelle est la bande passante disponible.
Pendant cette phase, *CW* commence à la taille de 1 segment de taille *Segment Size (SegSize)*, et est augmentée d'un segment à chaque *ACK* reçu. Ce comportement se traduit par une croissance exponentielle de *CW* qui double pour chaque délai aller-retour, ou

Round Trip Time (RTT) :

$$\begin{aligned} CW &:= \frac{CW}{\beta} \\ \text{avec} \quad \beta &:= 0.5 \end{aligned} \tag{2.1}$$

2.2.1.2 LA PHASE DE *Congestion Avoidance* (CA)

La phase d'évitement de congestion, ou *CA*, est utilisée quand le flux connaît approximativement quel est le débit disponible : ce débit est estimé à la valeur du débit utilisé lors de la dernière perte. Il s'agit pendant cette phase de se rapprocher lentement du débit disponible pour éviter de provoquer une congestion trop importante.
Pendant cette phase, *CW* est augmenté d'un segment à chaque fois que tous les paquets de la *CW* courante sont acquittés. Pour une progression plus régulière de la *CW*, chaque *ACK* reçu augmente cette dernière de :

$$\begin{aligned} CW &:= CW + \alpha * \left(\frac{SegSize * SegSize}{CW} \right) \\ \text{avec} \quad \alpha &:= 1 \end{aligned} \tag{2.2}$$

2.2.1.3 L'ALGORITHME *Additive Increase and Multiplicative Decrease* ($AIMD$)

Concernant le partage équitable de la bande passante, *TCP* utilise un algorithme distribué permettant de faire converger les différents flux partageant les mêmes conditions réseaux vers le même débit équitable. Cet algorithme est basé sur une augmentation additive et une réduction multiplicative, ou *Additive Increase and Multiplicative Decrease (AIMD)*, de *CW*. La partie accroissement de *CW* est assurée par la phase de *CA*, tandis que la diminution est réalisée à chaque congestion, ou *Congestion Event (CE)*, où *CW* est réduite de moitié :

$$\begin{aligned} CW &:= CW * \beta \\ \text{avec} \quad \beta &:= 0.5 \end{aligned} \tag{2.3}$$

Quand *CW* atteint la taille correspondant à un partage équitable de la bande passante, le même débit est obtenu par chaque flux *TCP* de même *RTT* traversant le lien congestionné. Dans ce cas, chaque flux subit les cycles décrits dans [Mathis 97] et représentés à la figure 2.1, composés d'une perte suivie d'une phase d'évitement de congestion. Chaque perte réduit *CW* de moitié (#1 de la figure 2.1). La phase d'évitement de congestion (#2 de la figure 2.1) nécessite $CW * (1 - \beta)$ *RTT* pour atteindre de nouveau la taille de *CW* et ainsi causer une nouvelle perte, signal du commencement

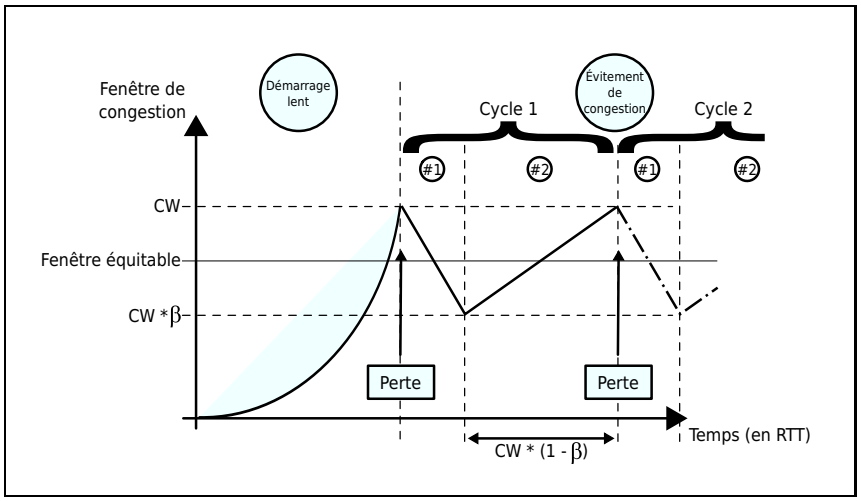


FIGURE 2.1 – Comportement de la fenêtre de congestion de *TCP*

d'un nouveau cycle.
Ainsi si deux flux se partagent la bande passante et que l'un a un débit plus élevé que l'autre, celui ayant le débit le plus élevé subira une réduction de *CW* plus importante à chaque *CE* et mettra plus de temps pour retrouver son débit précédant la congestion, laissant ainsi le flux ayant le plus faible débit grappiller un peu de bande passante à chaque *CE*. Les deux flux convergeront ainsi peu à peu vers la même *CW*. En effet, cette convergence est également assurée par l'augmentation linéaire par *RTT* due à l'accroissement de la fenêtre d'un segment à chaque fois qu'une fenêtre entière est acquittée.

2.2.1.4 GÉNÉRALISATION DES FACTEURS *AIMD*

Pour les mécanismes précédents, nous avons vu que *TCP* utilise un facteur de croissance additif $\alpha := 1$ et un facteur de réduction multiplicatif $\beta := 0.5$. De plus, les analyses détaillées [Yang oo] [Floyd ooa] montrent qu'un protocole de partage de bande passante utilisant un algorithme *AIMD* est équitable avec *TCP* si :

$$\alpha := 3 * \frac{(1-\beta)}{(1+\beta)} \tag{2.4}$$

Cette relation est bien sûr vérifiée par *TCP* lui-même pour lequel $\alpha := 1$ et $\beta := 0.5$.

2.2.1.5 PROBLÈME DES *RTT* HÉTÉROGÈNES

L'augmentation obtenue grâce au mécanisme de *CA* est d'un segment par *RTT* alors que la réduction lors d'une congestion est toujours d'un facteur 2 et est donc indépendante du *RTT*.
Par exemple, si une congestion arrive alors que 2 flux avec des *RTT*

différents se partagent équitablement la bande passante, alors les deux flux réduiront de la même manière leurs *CW*, mais le flux avec le *RTT* le plus long verra sa *CW* augmenter plus lentement. Par conséquent, le flux avec le plus grand *RTT* obtiendra une *CW* plus petite que le flux concurrent.
De plus, une source *TCP* ne peut envoyer plus d'une *CW* en même temps, et comme le premier paquet envoyé de cette *CW* ne sera pas acquitté avant un *RTT*, cela signifie que le débit obtenu est dépendant du *RTT* :

$$debit \simeq \frac{CW}{RTT} \tag{2.5}$$

Ainsi, le flux avec un *RTT* plus important aura non seulement une *CW* réduite, mais également un débit encore plus réduit. La différence de *RTT* influence doublement l'iniquité entre les flux *TCP*.

2.2.1.6 Problème du temps de convergence des réseaux à hauts débits et longs délais

Dans le cas de réseaux à longs délais, la croissance de la *CW* est diminuée aussi bien en phase de *SS*, qu'en phase de *CA*. En outre pour la même *CW*, le débit obtenu est d'autant plus réduit que le *RTT* est important. Si une telle situation se produit sur un réseau disposant d'une bande passante importante, le flux *TCP* mettra énormément de temps pour converger jusqu'à saturer la bande passante disponible. De plus, une fois cette bande passante atteinte et la production d'une congestion, la *CW* est alors réduite d'un facteur 2 et il faut alors repasser par une longue phase de *CA* avant d'obtenir un débit de nouveau proche de la bande passante disponible. Ainsi, le réseau sera la majorité du temps sous-utilisé. Par exemple, pour un réseau disposant d'un débit de 100 Mb/s et d'un temps de propagation de 300 ms, la bande passante est totalement utilisée dès que la *CW* fait 30 Mb, soit environ 20548 segments. Pour saturer le réseau, il faut donc :

- 4.5 secondes ou 15 *RTT* en *SS* pour pouvoir passer d'une *CW* de 1 à 20548.
- Plus de 51 minutes ou 10274 *RTT* en *CA* pour pouvoir passer d'une *CW* de 10274 à 20548.

2.2.2 Techniques de convergence accélérée

Le protocole *CUBIC* [Ha 08] est l'évolution du protocole *Binary Increase Congestion control (BIC)* [Xu 04] et propose une solution pour le temps de convergence de *TCP* pour les réseaux à long délais et à haut débit.

CUBIC utilise deux modes de calculs de *CW* et choisit pour chaque *ACK* le mode lui procurant la plus grande croissance :

- Le mode de compatibilité avec *TCP* est utilisé quand l'algorithme *AIMD* de *TCP* est plus agressif que celui introduit par *CUBIC* : c'est-à-dire quand le rapport délai sur débit est petit. Pour le savoir, *CUBIC* calcule la fenêtre qu'aurait dû avoir un flux *TCP*. Si cette fenêtre est supérieure à celle calculée avec l'algorithme dédié de *CUBIC*, alors la valeur de la nouvelle fenêtre est calculée selon l'estimation d'un *TCP* classique.
- Sinon, les conditions du réseau sont telles qu'un *TCP* classique est trop lent à converger vers le débit équitable. La fonction de croissance linéaire de *TCP* est alors remplacée par une courbe cubique. Cette courbe peut être décomposée en trois parties afin de mieux comprendre les motivations sous-jacentes :
  - La première partie est une approche logarithmique convergeant vers le débit de la dernière perte. Ainsi, si la dernière perte est proche du débit équitable, alors cette approche permet d'utiliser très rapidement une grande partie de la bande passante disponible.
  - La seconde partie est une transition quasiment plane et horizontale autour du débit de la dernière perte. Ainsi, si la dernière perte est proche du débit équitable, le fait de passer un temps important proche de ce débit permet de stabiliser et d'améliorer l'équité du protocole.
  - La troisième partie est une augmentation exponentielle. Ainsi, si la dernière perte est loin du débit équitable, alors une recherche de ce débit équitable est réalisée avec une augmentation dont la rapidité augmente avec le temps depuis la dernière perte.

Ainsi, *CUBIC* permet de garder une bonne équité avec les flux *TCP* traditionnels quand le rapport délai sur débit est faible et permet également d'utiliser plus de bande passante pour les autres réseaux.

2.2.3 TECHNIQUES D'APPROXIMATION BASÉES SUR L'ÉQUATION DE *TCP*

TCP a été le sujet de plusieurs analyses, dont celles de Mathis [Mathis 97] et de Padhye [Padhye 98] qui ont défini le débit de *TCP* en fonction du *RTT* et du taux de pertes. Ces travaux ont ouvert la voie vers la création d'un protocole tel que *TCP-Friendly Rate Control (TFRC)* [Floyd 00b], dont le débit moyen est semblable à celui d'un *TCP* classique, mais dont la fluctuation est réduite, car calculé selon les équations ci-dessous grâce au calcul de moyennes

sur les pertes, le *RTT* et le minuteur de retransmission :

$$debit := \frac{8}{RTT * \sqrt{\frac{2*LR}{3}} + RTO * \left(3 * \sqrt{\frac{3*LR}{8}}\right) * LR * (1 + 32 * LR^2)}$$

avec *Loss Rate (LR)*
Le pourcentage de pertes.
et *Retransmission TimeOut (RTO)*
Le minuteur de retransmission.

2.2.4 TECHNIQUES UTILISANT LE TEMPS DE PROPAGATION

Une autre amélioration possible de *TCP* est d'essayer de ne pas provoquer des pertes pour estimer la bande passante disponible, mais d'essayer d'observer l'évolution du *RTT* ou du débit obtenu pour arrêter la croissance de *CW* avant de provoquer une perte. Ainsi, *VEGAS* [Brakmo 95] propose de comparer le débit obtenu lors du dernier *RTT* avec le débit qu'aurait du obtenir le protocole si la bande passante n'est pas entièrement utilisée et que les tampons des routeurs sont vides, ce qui correspond au débit calculé par le rapport entre la *CW* et le *RTT* minimal observé. Ainsi, si la différence entre le débit réel et le débit souhaité est trop important, alors *CW* est diminuée avant qu'un paquet ne soit perdu. Inversement pour ne pas sous-utiliser la bande passante disponible, si cette différence est trop petite, alors le protocole continue d'accroître *CW*. Et entre les deux seuils, la *CW* reste inchangée.

2.3 CONTRÔLE DE CONGESTION PAR LES ROUTEURS

Le contrôle de congestion, même s'il est principalement effectué de bout en bout, peut être accompagné de mécanismes au sein du réseau. En effet, les routeurs peuvent fournir différentes informations ou réactions permettant d'améliorer le contrôle de congestion effectué sur les machines terminales.

2.3.1 LES STRATÉGIES DES FILES D'ATTENTES

La plupart des routeurs utilisent des files d'attentes du type premier arrivé, premier reparti, ou *First In - First Out (FIFO)*. Ainsi, les versions de *TCP* réactives aux pertes augmentent leur débit jusqu'à saturer entièrement ces files d'attentes avant de provoquer une perte qui sera le signal d'une réduction de débit.
Pour éviter que cette recherche de la perte ne se traduise par une saturation totale des ressources du réseau, plusieurs autres stratégies de file d'attente ont été proposées dont *Random Early Detection (RED)* [Floyd 93] est la plus célèbre. Cette stratégie consiste à fixer

des seuils de remplissage de la file d'attente à partir desquels tout nouveau paquet arrivant aura une chance de plus en plus importante d'être jeté, même si la file d'attente n'est pas entièrement pleine. Ainsi, le flux *TCP* verra sa croissance freinée dès le dépassement du débit disponible, et non pas une fois que le débit émis est largement supérieur au débit disponible dû au temps que la file d'attente se remplisse entièrement. Cela permet de diminuer les oscillations côté *TCP* et de réduire les moments de saturation totale des routeurs.

2.3.2 L'annonce préalable des congestions

Le mécanisme *RED* peut être perçu comme un peu radical, ce qui a poussé à présenter l'idée que le flux *TCP* pouvait être informé d'un début de congestion sans être forcé de provoquer une perte de paquet. Ainsi, le protocole *Explicit Congestion Notification (ECN)* [Floyd 94], propose d'utiliser deux bits d'un champs *DiffServ* du paquet *IP* pour pouvoir marquer les paquets qui traversent au moins un routeur congestionné. Si la source et le récepteur sont capables de comprendre le protocole *ECN*, alors la source en voyant des paquets marqués peut réagir et diminuer son débit pour ainsi tenter de réduire la congestion et limiter le nombre de pertes subies.

2.4 Conclusion sur les mécanismes de *TCP*

Malgré les bénéfices que peuvent procurer des mécanismes comme *RED* ou *ECN*, ces derniers ne sont que très rarement activés car leur utilisation peut s'avérer coûteuse en temps de calcul pour les routeurs.
Concernant *TCP*, la version la plus utilisée est encore *TCP-newReno* même si d'autres versions sont dorénavant activées par défaut, ce qui est le cas pour *CUBIC* sous *Linux*. Or l'évolution de *TCP* est un domaine de recherche très actif ce qui pose problème pour comparer ces différentes solutions.

2.5 Métriques d'évaluation

Pour déterminer l'efficacité d'un protocole de contrôle de congestion, des métriques doivent être définies pour pouvoir analyser son comportement [Floyd 08].

2.5.1 Utilisation maximale bande passante

Un des buts d'un contrôle de congestion est de découvrir quelle est la bande passante disponible pour en tirer profit au maximum.

Ainsi, mesurer l'utilisation de la bande passante, ou *Bandwidth Usage (BWU)*, permet d'observer l'utilisation des ressources réseaux disponibles. Cette utilisation de la bande passante est calculée comme le rapport entre la somme des débits des flux parcourant le lien faible et la bande passante disponible :

$$BWU := \frac{\sum flux\_parcourant\_le\_lien\_faible}{bande\_passante\_disponible}$$

Ainsi, la bande passante est utilisée correctement si $BWU \to 1$ et sous-utilisée si $BWU \ll 1$.

2.5.2 Pourcentage de perte

Il ne suffit pas à un flux d'utiliser un maximum de bande passante pour être efficace : il faut également limiter la surcharge du réseau qui se traduit par un remplissage des tampons des routeurs, qui une fois pleins, sont obligés de jeter les nouveaux paquets arrivant. Ainsi, le pourcentage de perte est un indicateur de la surcharge du réseau. Ce qui sur un réseau "best effort" pourrait se traduire par : temporairement inutilisable.
Le pourcentage de perte (LR) est calculé comme étant le pourcentage entre le nombre de paquets perdus et envoyés :

$$LR := \frac{nombre\_de\_paquets\_perdus}{nombre\_de\_paquets\_recus + nombre\_de\_paquets\_perdus} * 100$$

Un pourcentage de perte important indique un comportement trop agressif des flux qui n'arrivent pas à ménager suffisamment le réseau.

2.5.3 Partage équitable de la bande passante

Un second but des protocoles de contrôle de congestion est de répartir de façon équitable la bande passante entre les différents flux utilisant le ou les mêmes liens faibles. Cette notion d'équité est difficile à cerner car les flux sont équitables :

- S'ils ont le même débit dans le cas où ils partagent les mêmes conditions réseaux : par exemple les différents flux traversent les mêmes liens faibles et ont le même RTT.
- Mais si les flux ont des conditions réseaux différentes, alors le débit équitable est plus complexe à calculer. En effet dans l'exemple de la figure 2.2, la machine D communique simultanément avec A, B et C et ces 3 flux se partagent le lien faible à 10 Mb/s entre $R4$ et $R5$. Cependant le flux $C - D$ ne peut pas dépasser le débit de 1 Mb/s car il passe par un second lien faible entre $R3$ et $R4$. Ainsi, il reste encore 9 Mb/s de libre entre $R4$ et $R5$, qui idéalement seront partagés en 2 flux de 4.5 Mb/s pour $A - D$ et $B - D$.

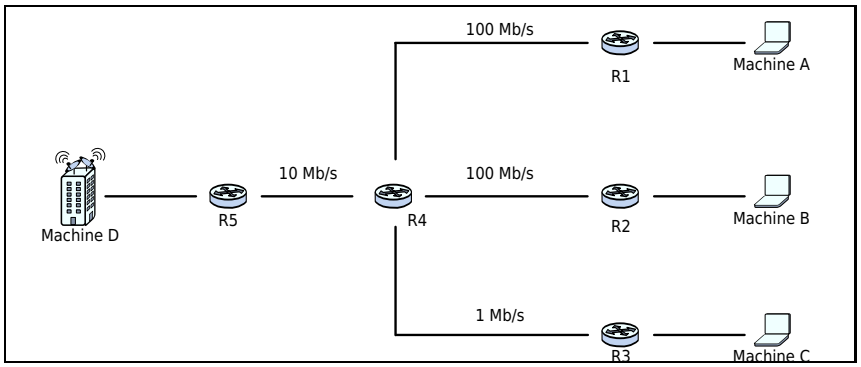


FIGURE 2.2 – Exemple de réseau avec des liens faibles multiples

Ainsi, par la suite l'équité ou *Fairness (F)*, est calculée comme définie par l'indice de Jain [Chiu 89][Jain 98][Jain 99] :

$$F := \frac{\left(\sum_{i=1}^{N} b_i\right)^2}{N \sum_{i=1}^{N} b_i^2}$$

Avec :

- b_i Le débit normalisé de la connexion i.
  Avec $b_i := T_i/O_i$:
  – T_i le débit mesuré de la connexion i.
  – O_i le débit équitable pour la connexion i.
- N Le nombre de connexions.

Cet indice permet de normaliser l'équité entre 0 et 1 :

- Les flux sont parfaitement équitables si $F \approx 1$.
- Les flux sont totalement inéquitables si $F \approx 1/N$, ce qui arrive quand un flux monopolise toute la bande passante.
- Un indice de 0.8 peut signifier par exemple que 80% des flux sont équitables, alors que 20% sont taris.

Enfin, l'équité avec *TCP* reste une notion relative à la version de *TCP* utilisée, car ces dernières ne sont pas forcément équitables entre-elles [Li 07][Munir 07]. Ainsi, dans la suite nous n'utiliserons pour les différents tests que la version $new - RENO$ de *TCP*, qui est la version historique de référence même si cette dernière est de moins en moins utilisée.

2.5.4 TEMPS DE CONVERGENCE

Le temps de convergence ou *Convergence Time (CT)* permet de déterminer la rapidité avec laquelle un flux arrive à obtenir son débit équitable. Le temps de convergence peut servir à évaluer le temps de démarrage pour découvrir la bande passante disponible, ou le temps séparant 2 rafales de pertes. Dans le premier cas, un

temps de convergence important au démarrage signifie que le protocole évalué n'est pas idéal pour les sessions de courte durée, car il est trop lent à converger pour en tirer pleinement parti. Dans le second cas, un temps de convergence important peut être la source d'une mauvaise utilisation de la bande passante, due par exemple à certaines lenteurs du mécanisme de *CA* quand le flux a une bande passante et un *RTT* importants.

2.5.5 TEMPS DE RÉACTION

Le temps de réaction ou *Reaction Time (RT)* correspond à la durée nécessaire pour détecter une congestion et en sortir. Ce temps est difficile à mesurer, mais peut être approximé par la durée des rafales de pertes, où une rafale est calculée comme la durée pendant laquelle toutes les pertes sont séparées d'un temps inférieur à un certain intervalle. Pour *TCP*, cet intervalle peut par exemple être un *RTT*.

2.6 CONCLUSION

En *unicast*, *TCP* est une solution efficace de contrôle de congestion largement déployée et utilisée. Cependant, de multiples variantes de *TCP* ont été proposées car *TCP* n'est pas une solution parfaite et est confrontée à plusieurs problèmes tels que l'iniquité entre flux possédant des *RTT* hétérogènes, ou la lenteur de convergence pour les réseaux à longs délais et hauts débits.
Ainsi, notre but est de créer pour le *multicast* une solution compatible et interopérable avec la solution dominante de *TCP* qui est *TCP-newReno*.

3

CONTRÔLE DE CONGESTION *MULTICAST*

Les communications de groupe procurent un moyen efficace pour transmettre simultanément la même donnée vers plusieurs destinataires. Par exemple, pour la diffusion de l'*IPTV* ce type de communication est largement plébiscité en combinaison avec le protocole *UDP*. Ainsi, ces flux ne sont pour le moment utilisés qu'au sein de réseaux sur-dimensionés.
Cependant, il existe une demande pour utiliser le *multicast* sur des réseaux à capacité variable ou réduite tels que ceux utilisés pour les usages mobiles avec un accès sans fil [Sarikaya 09].
Dans ces conditions, il est nécessaire que les flux *multicast* soient capables de partager équitablement la bande passante avec les flux *TCP* concurrents. La création d'un protocole de contrôle de congestion est donc obligatoire, car les flux *multicast* ne peuvent pas utiliser directement le protocole *TCP* du fait que ce dernier n'est pas conçu pour gérer plusieurs récepteurs simultanément.
Dans ce chapitre, nous allons présenter l'évolution des différents protocoles de contrôle de congestion pour le *multicast* en nous consacrant principalement aux mécanismes dont l'efficacité ne dépend pas du nombre de récepteurs, c'est-à-dire aux protocoles supportant le passage à l'échelle.

3.1 CONTRÔLE DE CONGESTION À DÉBIT UNIQUE

La première solution proposée fut d'utiliser un contrôle de congestion à débit unique pour le groupe de récepteurs : c'est-à-dire que la source émet un unique flux que chaque récepteur reçoit dans sa totalité.
Par exemple, *PGMCC* [Rizzo 00] propose que la source émette au débit qui correspond au débit qu'obtiendrait *TCP* s'il était exécuté sur le récepteur disposant de la bande passante la plus petite. Ce récepteur est élu et est le seul à émettre des *ACK* afin que la source puisse mettre à jour sa *CW* servant uniquement de régulateur de débit. Quant à la retransmission de paquets perdus, elle se fait par l'envoi d'*ACK* négatifs (*NACK*) que tout récepteur peut envoyer.

Cette situation limite tous les autres récepteurs du même groupe à la bande passante du pire récepteur et provoque ainsi un problème de décroissance du débit. Cette décroissance suit une fonction logarithmique inverse par rapport au nombre de récepteurs présents au sein du groupe [Chaintreau 01]. Par conséquent, ce type de protocole est dépendant du nombre de récepteurs et ne peux pas passer à l'échelle.
Une solution permettant de mieux adapter les débits obtenus aux capacités des récepteurs est de répartir les récepteurs en plusieurs groupes suivant leurs débits disponibles, permettant ainsi de limiter la perte de performance par groupe de récepteurs. Cependant, un flux indépendant doit être émis pour chacun de ces groupes, ce qui réduit l'intérêt d'utiliser des flux *multicast*.
Outre ce problème, pour un nombre de groupe important, toute méthode de contrôle de congestion *multicast* qui nécessite une signalisation entre récepteurs et la source ne peut pas être extensible. Ceci car plus le nombre de récepteurs augmente, plus le nombre de messages envoyés à la source augmente également, ce qui peut à la longue provoquer un phénomène d'explosion du nombre d'acquittements reçus par la source.

3.2 CONTRÔLE DE CONGESTION À DÉBITS MULTIPLES

Pour répondre aux problèmes des propositions à débit unique, plusieurs solutions proposent d'utiliser une source qui décompose son flux en plusieurs couches correspondant chacune à un groupe *multicast*. Ainsi, chaque récepteur peut s'abonner au nombre de groupes *multicast* qu'il désire pour obtenir un débit correspondant à ses conditions réseau. Par exemple pour un flux vidéo, ajouter un canal augmente la qualité de la vidéo reçue. Dans le cas de transmission fiable, comme avec FLUTE [Luby 04b][Peltotalo 07], chaque canal supplémentaire augmente le débit du transfert. Ces abonnements se font donc hiérarchiquement et de façon cumulative, afin que le flux reçu par un récepteur soit compris dans celui de tout autre récepteur abonné à un débit plus important.
Ainsi, le mécanisme de contrôle de congestion est dirigé par chaque récepteur qui gère sa propre *CW* et n'a donc pas besoin d'envoyer de messages à la source, ce qui permet de supporter le passage à grande échelle.

3.2.1 CANAUX STATIQUES

Les premières solutions qui supportent le passage à l'échelle, proposent de diviser le flux de la source en plusieurs couches ou canaux, chacun identifié par un groupe *multicast*. Une source uti-

lisant des canaux statiques émet en continu sur chacun de ses canaux à un débit fixe. Le récepteur doit donc s'abonner ou se désabonner pour respectivement augmenter ou réduire son débit.

3.2.1.1 *Receiver-driven Layered Multicast (RLM)*

RLM [McCanne 96b] est le premier protocole à proposer l'utilisation de canaux statiques et permet à chaque récepteur de choisir son débit de façon indépendante.
Ce protocole est très simple et définit les bases qui sont utilisées par tous les autres protocoles de la même famille :

- En cas de congestion, il faut se désabonner d'un canal. Une situation de congestion est facile à caractériser et *RLM* se base sur l'observation des pertes. Ainsi, un numéro de séquence est rajouté aux entêtes des paquets.
- En sous-utilisation de la bande passante disponible, il faut s'abonner à un nouveau canal. Le protocole détermine si la bande passante est sous-évaluée en faisant des essais réguliers pour s'abonner à la couche suivante. Si l'abonnement courant n'est pas suivi d'une perte, alors le récepteur reste abonné au canal et il réduit donc ainsi la sous-utilisation de la bande passante. Sinon, le récepteur se désabonne immédiatement du nouveau canal et programme la tentative d'abonnement en augmentant de façon multiplicative le temps entre deux tentatives successives.

Cependant, se désabonner d'un groupe n'est pas immédiat et peut prolonger la congestion. Le récepteur n'est alors plus capable de déterminer si une congestion persistante est due au fait que le désabonnement n'est toujours pas effectif au niveau du lien congestionné ou s'il doit se désabonner d'un canal supplémentaire.
Un autre problème est que la gestion du débit reçu par les différentes tentatives d'abonnement a été conçue pour limiter les congestions, mais non pas pour partager la bande passante de façon équitable.

3.2.1.2 *Receiver driven, Layered Congestion control scheme (RLC)*

RLC [Vicisano 98] essaye de s'abstraire un peu des problèmes dus au routage *multicast*. En effet, *RLC* utilise les mêmes mécanismes que *RLM*, mis à part que les tentatives d'abonnement au canal suivant sont synchronisées par des bits de synchronisation présents dans les paquets. Comme *RLC* utilise des débits tels qu'à chaque abonnement un récepteur double son débit, ces bits de synchronisations sont deux fois plus rares dans chaque nouveau canal. Cela permet aux récepteurs derrière un même lien faible de converger vers le même débit.
Cependant, l'amélioration la plus importante de *RLC* est le fait

qu'une tentative d'abonnement ne peut se faire que si le récepteur n'a pas eu de pertes pendant une période de rafale. En effet, la source *RLC* envoie régulièrement des rafales de paquets dans tous les canaux afin de doubler temporairement le débit des canaux. Ainsi, chaque récepteur peut tester sans avoir à s'abonner s'il dispose de la bande passante suffisante pour s'abonner au canal suivant. Bien qu'ingénieuse, cette méthode est décriée car la longueur de la rafale ne suffit pas forcément pour provoquer une saturation des tampons du routeur du lien faible.
Toujours pour parer au problème de la latence du temps de désabonnement, *RLC* définit une période de surdité de 10 secondes qui permet à un récepteur qui vient de subir une perte et par conséquent de se désabonner d'un canal, d'être sourd face à l'arrivée de nouvelles pertes. Ainsi, les protocoles de routage *multicast* doivent avoir le temps de rendre le dernier désabonnement effectif en aval du lien faible.
En plus du problème des rafales trop courtes, *RLC* n'adresse toujours pas le problème lié à l'équité envers les autres flux.
Enfin, un dernier problème est qu'une source *RLC* ne propose qu'une granularité très grossière de débits, car les récepteurs ne peuvent que doubler ou diviser par deux leur débit. Ainsi, *RLC* ne propose qu'une liste restreinte de débits qui ne permet pas forcément d'utiliser efficacement et équitablement la bande passante de chaque récepteur.

3.2.1.3 *Fair Layered Increase/Decrease with Static Layering (FLID-SL)*

FLID-SL [Byers 02] est un protocole de contrôle de congestion utilisant des débits statiques, mais utilise une granularité de débit plus faible que *RLC* puisqu'il propose que chaque abonnement accroisse le débit global reçu d'un facteur de $1,3$.
FLID-SL estampille chaque paquet par un numéro d'intervalle noté *Time Slot Interval (TSI)* incrémenté toutes les *Time Slot Duration (TSD)* d'une durée de $0,5$ seconde. Pour *FLID-SL*, chaque changement de TSI marque une décision du récepteur. Il décide d'augmenter son débit, si, pendant le TSI précédent il n'a pas connu de pertes et qu'un marqueur de synchronisation est activé sur son canal le plus élevé. S'il a connu des pertes, le récepteur doit alors réduire son débit. Sinon son débit ne change pas.
Cependant, *FLID-SL* ne présente aucune innovation pour la gestion de la latence de désabonnement d'un groupe *multicast*. En effet, il est indiqué d'utiliser ce protocole uniquement sur des réseaux qui ne présentent pas cette latence. Dans les autres cas, il est conseillé d'utiliser son protocole jumeau : *FLID-DL*, pour *Dynamic Layering*, qui sera présenté dans la partie sur les protocoles

à canaux dynamiques.

3.2.1.4 *Pair receiver-driven cumulative Layered Multicast (PLM)*

PLM [Legout 00b] [Legout 00a] est un protocole à canaux statiques qui ne s'intéresse pas au problème du délai de désabonnement, mais se concentre sur l'estimation du débit équitable. En effet, *PLM* propose d'utiliser le mécanisme des paires de paquets pour estimer le débit équitable en fonction du délai entre la réception des deux paquets d'une même paire :

$$debit := \frac{taille\_paquet}{delai\_inter\_paquets} \tag{3.1}$$

Avec :

- *debit* : la bande passante disponible estimée en bits par seconde, au niveau du lien faible.
- *taille_paquet* : la taille en bits des paquets reçus.
- *delai_inter_paquets* : le délai entre l'arrivée des deux paquets d'une même paire.

PLM évalue le débit courant pour chaque paire de paquets reçue. Ainsi, si le débit évalué est en dessous du débit actuel, alors le récepteur doit réduire ses abonnements. Mais si pendant une seconde, le récepteur évalue uniquement des débits supérieurs, alors le récepteur va s'abonner aux canaux nécessaires pour recevoir le débit le plus proche en dessous du débit minimal calculé pendant cette seconde.
L'évaluation du débit repose sur le principe que les paquets d'une paire sont espacés selon le remplissage de la file d'attente du ou des routeurs en amont du ou des liens faibles. Cela n'est valide que si les routeurs concernés implémentent et activent un mécanisme de file d'attente équitable ou *Fair Queing* [Legout 02]. Malheureusement, ce type de file d'attente est loin d'être intégré dans les différents routeurs actuels, ce qui empêche l'utilisation directe d'une telle méthode.

3.2.1.5 *Explicit Rate Adjustment (ERA)*

ERA [Puangpronpitag 03] est un protocole relativement similaire à *PLM*, mais qui s'inspire également du principe de *TFRC*. Ainsi, ce dernier s'abonne au débit minimum entre :

- Le débit tel que calculé par *PLM*.
- Le débit disponible estimé pour un flux *TCP* :

$$debit\_TCP = \frac{taille\_paquet}{t_{rtt} * \sqrt{\frac{2*l}{3}} + 12 * \sqrt{\frac{3*l}{8}} * l * (1 + 32 * l^2)} \tag{3.2}$$

Avec :

- *debit_TCP* : l'estimation du débit disponible pour un flux *TCP* dans les mêmes conditions (en bits par seconde).
- *taille_paquet* : la taille en bits des paquets reçus.
- t_{rtt} : l'estimation du *RTT* pour ce flux (en secondes). Cette estimation est faite grâce à des estampilles placées dans les paquets et à la synchronisation des horloges de la source et des récepteurs, tel que par *Network Time Protocol (NTP)*.
- l : le taux de pertes (entre 0.0 et 1.0) observé sur une période donnée.

L'utilisation d'une équation du débit de *TCP* permet de rendre le protocole plus équitable avec *TCP* là où *PLM* pourrait être trop agressif.

3.2.1.6 Problème du temps de désabonnement

Le temps de désabonnement d'un groupe *multicast* peut-être particulièrement long. En effet, les protocoles *IGMP* et *MLD* sont conçus pour qu'un routeur maintienne l'abonnement à un groupe tant qu'il n'est pas certain qu'aucun récepteur ne souhaite recevoir le groupe. Ce comportement conservateur se retrouve dans le fait que ces protocoles ont un facteur de robustesse, par défaut de 2 en $IGMPv2/MLDv1$ et en $IGMPv3/MLDv2$, qui spécifie à chaque routeur de répéter ses demandes pour savoir si un récepteur est toujours intéressé par le groupe. L'attente de réponse entre chaque demande est fixé par défaut à 1 seconde en $IGMPv2/MLDv1$ et en $IGMPv3/MLDv2$. Ainsi, cette attente couplée au facteur de robustesse fait qu'un désabonnement du lien local prend 2 secondes en $IGMPv2/MLDv1$ et en $IGMPv3/MLDv2$.
De plus, quand un protocole à canaux statique se désabonne d'un groupe, cela signifie qu'une congestion est apparue. Par conséquent, ce désabonnement doit remonter jusqu'au routeur en amont du lien faible pour supprimer cette congestion. Ainsi, en plus du long temps de désabonnement en $IGMP/MLD$ vient s'ajouter le temps de propager les messages *PIM-SM* pour élaguer l'arbre *multicast*. *PIM-SM* étant un protocole de routage, ce dernier est traité par l'ordonnanceur des messages de contrôle des routeurs et la propagation de la demande de désabonnement peut être retardée, par exemple pour agréger plusieurs messages ensemble. Nous verrons dans le § 4.4 que ce temps peut être long.
Enfin, *PIM-SM* est un protocole directement encapsulé dans le protocole *IP* et ses messages peuvent être perdus, notamment quand comme ici le message doit passer par des routeurs surchargés par la congestion en sens inverse. Sauf en cas d'un autre changement dans les abonnements, le message perdu ne sera rediffusé que 60 secondes plus tard, correspondant au temps par défaut de la variable [Join/Prune-Period] du protocole *PIM-SM*.

3.2.2 CANAUX DYNAMIQUES

L'utilisation de canaux dynamiques permet de s'affranchir des lenteurs de désabonnement d'un groupe *multicast*. Pour cela, la source émet de façon constante uniquement sur le canal de base. Les autres canaux diminuent régulièrement leur débit d'un facteur donné.
Le temps est alors divisé en cycles réguliers dont la durée est nommée *TSD*. À chaque nouveau cycle, un canal est créé au débit maximum et le canal dynamique de débit le plus faible se termine et devient muet. Un récepteur peut ainsi réduire son débit en attendant simplement que les canaux diminuent d'eux-mêmes leur débit. Ainsi un récepteur ne se désabonne d'un groupe que si ce dernier est muet. Comme ce groupe n'émet plus de paquets, un récepteur n'a plus à se soucier du temps de désabonnement. S'il ne fait rien d'autre, son débit diminue. S'il veut conserver son débit, il doit s'abonner à un nouveau canal. Et s'il veut augmenter son débit, il doit s'abonner au moins à un second canal avant *TSD* secondes.

3.2.2.1 *Fair Layered Increase/Decrease with Dynamic Layering (FLID-DL)*

FLID-DL [Byers 02] est le premier protocole de contrôle de congestion *multicast* à proposer une solution vraiment efficace pour le problème du temps de désabonnement : l'utilisation de canaux dynamiques comme décrit au § 3.2.2 avec un *TSD* de 500 ms.
Hormis ce mécanisme de canaux dynamiques, *FLID-DL* se comporte exactement de la même façon que *FLID-SL* et est donc également sujet au problème d'indépendance du calcul de débit par rapport au *RTT* et n'est donc pas équitable avec *TCP*.

3.2.2.2 *Wave and Equation Base Rate Control (WEBRC)*

Pour résoudre le problème du temps de désabonnement, *WEBRC* [Luby 02] reprend le mécanisme de canaux dynamiques de *FLID-DL*, mais en utilisant une diminution quasi-continue dans le temps. En effet, l'ordonnanceur de *WEBRC* calcule le débit de chaque groupe (cf. figure 3.1) comme étant un nombre de paquets à émettre pendant un *TSI*, qui pour *WEBRC* dure $TSD := 10$ secondes. Ces paquets sont séparés par un délai qui croit exponentiellement, provoquant ainsi une réduction exponentielle du débit par un facteur $P = 0.75$ par *TSD* et le débit courant est recalculé à chaque nouveau paquet, selon la formule suivante :

$$current\_rate \quad = \quad (R_{start} * P^{\frac{t}{TSD}})$$

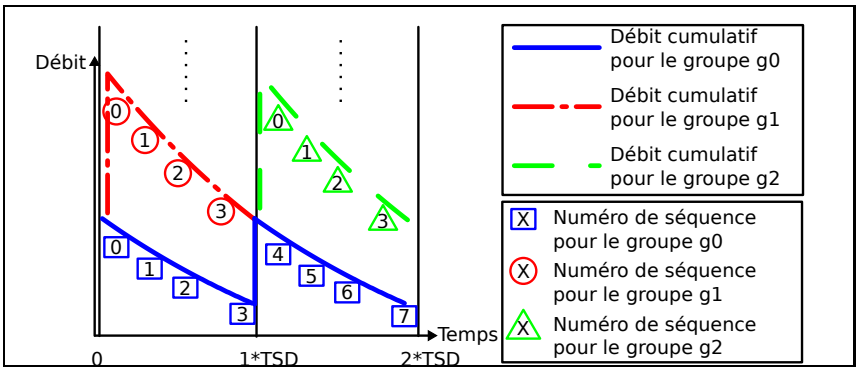


FIGURE 3.1 – L'ordonnanceur de la source de *WEBRC*

Avec R_{start} le débit maximal d'un groupe quand ce dernier devient actif, et t le temps écoulé depuis que le groupe est devenu actif. Ainsi, au bout de *TSD* secondes K groupes deviennent muets avec pour *WEBRC* $K := 1$.
De plus *WEBRC*, s'intéresse également à l'équité envers *TCP* et propose une méthode proche de celle de *TFRC* [Floyd 00b] [Handley 03]. En effet, *WEBRC* décide de potentiellement s'abonner à un nouveau canal à chaque changement d'époque qui arrive 20 fois par *TSD*. Cette décision résulte de la comparaison entre une estimation de la bande passante disponible notée *reqn* et une prévision du débit attendu pour la prochaine époque, notée *arr_p*. Le calcul de *reqn* se fait suivant l'équation 3.3 qui a été proposée dans [Luby 04a] pour approcher la formule de Padhye [Padhye 98] modélisant le débit *TCP*.

$$reqn = \frac{1}{artt * \sqrt{lossp} * (0.816 + 7.35 * lossp * (1 + 32 * lossp^2))} \quad (3.3)$$

reqn repose sur une probabilité de pertes notée *lossp*, et sur une approximation du *RTT multicast* notée *artt*. En effet, comme il n'existe pas de communication bidirectionnelle entre les récepteurs et la source, il est difficile d'estimer directement le *RTT*. Les récepteurs doivent donc se contenter d'une estimation qui pour *WEBRC* est équivalente au temps d'adhésion à un canal. Quant à *arr_p*, il repose sur le débit idéalement reçu à l'époque précédente réduit du facteur de diminution des canaux. Par conséquent, si $arr\_p \leq reqn$, alors le récepteur décide d'augmenter sa réception. Sinon le débit du récepteur diminue tout seul tant que $arr\_p > reqn$.

3.3 CONCLUSION

L'évolution des mécanismes de contrôle de congestion *multicast* a permis de créer des protocoles supportant le passage à l'échelle

Protocole	Description courte	Avantages	Inconvénients
PGMCC	Protocole à débit unique.	Simple et fiable.	Débit limité au récepteur de plus faible débit. Nombre de messages envoyés à la source proportionnel au nombre de récepteurs.
RLM, *RLC*, *FLID-SL*	Protocole en couches à débits statiques.	Débit indépendant pour chaque récepteur. Pas de messages envoyés à la source.	Sensible à la latence du temps de désabonnement. Inéquitable avec *TCP*.
PLM, *ERA*	Protocole en couches à débits statiques.	Utilisation des pairs de paquets pour estimer le débit équitable.	Nécessité du *Fair Queing* au sein des routeurs.
FLID-DL	Protocole en couches à débits dynamiques.	Insensible à la latence du temps de désabonnement.	Inéquitable avec *TCP*.
WEBRC	Protocole en couches à débits dynamiques.	Insensible à la latence du temps de désabonnement. Utilisation de la formule de *TFRC* pour estimer le débit équitable.	Utilisation du temps d'adhésion pour approximer le *RTT*. Fonction d'équité non robuste aux environnements éprouvants.

TABLE 3.1 – Récapitulatif des protocoles de contrôle de congestion pour le *multicast*

(cf. tableau 3.1). La première difficulté pour ces protocoles était de trouver une solution au problème du temps de désabonnement. Ce problème est résolu grâce aux canaux dynamiques.
Cependant, il reste un point important non encore validé qui est l'équité de ces protocoles envers *TCP*. En effet, les résultats présentés pour ces protocoles sont souvent succincts et présentent des scénarios simplistes. Il manque donc une évaluation exhaustive de ces protocoles qui ne peut se faire qu'en définissant des métriques d'évaluation. De plus, les différentes évaluations de ces protocoles n'ont été effectuées que par simulation simplifiant bien souvent des paramètres importants.
Nous allons donc montrer que malgré les progrès réalisés en terme d'équité par *WEBRC*, ce dernier possède encore des défauts que nous allons analyser afin de proposer des solutions adaptées à ces problèmes.

PROPOSITION DU CONTRÔLE DE CONGESTION MULTICAST EFFICACE

4

ÉVALUATION DE L'EXISTANT

4.1 MÉTRIQUES D'ÉVALUATION

L'évaluation et la comparaison de mécanismes nécessite de définir des métriques afin de mettre en évidence les différents comportements des protocoles de contrôle de congestion. En effet, la recherche de métriques adéquates pour évaluer ces protocoles est important et a déjà donné lieu à des discussions au sein de l'*Internet Engineering Task Force (IETF)*. Le résultat de ces discussions se retrouve dans le *Request For Comments (RFC)* 5166 [Floyd 08] qui définit des métriques pour les flux *unicast*. Notre réflexion au sujet des métriques à utiliser pour le *multicast* a abouti sur la conclusion que la plupart de ces métriques sont utilisables aussi bien en *unicast* qu'en *multicast*. Ainsi, il n'est pas nécessaire de les redéfinir : comme pour l'utilisation de la bande passante (*BWU* cf. § 2.5.1) ou le pourcentage de perte (*LR* cf. § 2.5.2). Pour les autres métriques, les paragraphes suivants discutent des points importants qui doivent être pris en compte pour les communications de groupe.
Dans les différentes expérimentations, les résultats de chaque métrique sont présentés de la façon suivante :

$$avg\ (\pm ci)$$

Avec :

avg	La moyenne des résultats obtenus.
ci	La variation correspondant à un intervalle de confiance de 95%.

4.1.1 PARTAGE ÉQUITABLE DE LA BANDE PASSANTE

La façon d'évaluer le partage équitable de la bande passante pour le *multicast* doit être prise en considération. En effet, comme le *multicast* permet à plusieurs récepteurs de recevoir le même flux au même moment, faut-il idéalement :

- Partager la bande passante pour que le flux *multicast* avec N récepteurs récupère un débit qui correspond au débit de N flux *TCP* concurrents ? En effet, si un flux dessert un nombre

important de récepteurs, il est compréhensible que ce flux bénéficie d'une priorité proportionnelle au nombre de récepteurs.
Cela pose cependant deux problèmes. Le premier est qu'il est très difficile de calculer le nombre de récepteurs d'un flux, car les routeurs ne propagent pas de l'un à l'autre le nombre d'abonnés au groupe, mais simplement une annonce d'abonnement à ce groupe. Le second problème est plus politique. En effet, comme les opérateurs ne sont pas actuellement favorables pour activer le *multicast*, il serait bien difficile de les convaincre du bien fondé motivant la "mise en priorité" des flux *multicast* par rapport aux flux *TCP*, même si cela peut se justifier.

- Partager la bande passante comme s'il s'agissait d'un flux *TCP* ? Cette stratégie limite l'intérêt d'utiliser le *multicast* pour l'utilisateur, mais cet intérêt reste entier pour un opérateur et semble beaucoup plus simple à mettre en œuvre aussi bien techniquement que politiquement.

Par conséquent, l'équité sera calculée et comparée de la même façon pour les flux *unicast* et *multicast* (cf. § 2.5.3).
Il est intéressant de noter que l'équité peut être mesurée selon plusieurs échelles de temps. L'objectif des flux *multicast* est d'obtenir une équité sur le long terme car les flux *multicast* sont beaucoup moins réactifs que *TCP*. Cette lenteur est notamment due à l'utilisation des groupes à débits dynamiques et de la signalisation *multicast* pour modifier le débit obtenu.

4.1.2 TEMPS DE CONVERGENCE

Le temps de convergence, ou *CT*, sert à déterminer la rapidité avec laquelle un flux arrive à obtenir son débit équitable (cf. § 2.5.4). *CT* est mesuré en secondes comme le temps écoulé entre :

- *Le moment où les conditions du réseau ont changé.*
  Ce moment correspond soit au démarrage de la session, soit à l'arrivée ou au départ d'un flux concurrent.
- *Le moment ou le flux obtient un débit correspondant à son estimation stable du débit équitable.*
  Cette estimation du débit équitable est calculée comme le débit moyen obtenu durant toute la période où les conditions réseaux demeurent inchangées. Bien entendu, suivant le protocole utilisé cette estimation peut être différente du débit réellement équitable, mais représente en tout cas le débit stable du protocole pour ces conditions.
  De plus, nous estimons que le débit obtenu est proche du débit moyen d'une session s'il se trouve à $\pm 10\%$ de ce dernier.

4.1.3 TEMPS DE RÉACTION

Le temps de réaction, ou *RT*, sert à estimer la durée nécessaire pour résorber une congestion (cf. § 2.5.5). *RT* est approximé par la durée en secondes des rafales de pertes.
Nous nous intéressons principalement à l'évaluation de *RT* après la phase de convergence, car cette dernière produit souvent la congestion la plus importante. Par exemple, au démarrage *TCP* utilise le mécanisme de *SS* qui est souvent suivi d'une congestion plus importante que celles observées durant les phases de *CA*. Dans la suite de cette étude, *RT* est calculé comme étant la durée de la rafale de pertes définie par la durée d'une suite de pertes dont l'intervalle inter-pertes est inférieur à 500 ms.

4.1.4 LE SURCOÛT DE SIGNALISATION

Même si le *RFC* 5166 ne le prend pas en compte, la signalisation du protocole de contrôle de congestion est souvent insignifiante en terme de débit mais peut avoir une importance conséquente en terme de temps de calcul et de traitement par les équipements d'interconnexion. Par exemple, en *unicast* les messages *ACK* peuvent charger le réseau ou perturber le comportement de *TCP* comme dans le cas du problème d'iniquité dans les *Wireless Local Area Network (WLAN)*s [Lopez-Aguilera 08]. En *multicast*, il est donc intéressant d'étudier la signalisation utilisée par les protocoles de routage.
En effet, si les récepteurs des flux *multicast* n'envoient pas de messages vers la source, ils utilisent cependant la signalisation de contrôle *multicast* pour gérer les abonnements et désabonnements des groupes *multicast*. Ainsi, cette métrique sert à évaluer l'impact des mécanismes de contrôle de congestion *multicast* sur le plan de contrôle. Le surcoût de signalisation, ou *Signaling Overhead (SO)*, représente l'évaluation du nombre moyen de messages d'abonnements et de désabonnements émis par le contrôle de congestion par seconde. Comme la signalisation n'est pas importante en terme de débit mais plutôt en terme de temps de calcul pour les équipements d'interconnexion, nous ignorons dans cette partie les messages dont l'utilité unique est d'améliorer la robustesse du routage *multicast* (cf. § 6.7). Or, pour les mécanismes de contrôle de congestion à débits fixes, *SO* fluctue et dépend totalement de la variation du débit demandé. Cependant, il n'en est pas de même pour les mécanismes à débits dynamiques. Les deux paragraphes suivants montrent comment calculer la limite de *SO* pour les protocoles à débits dynamiques.

4.1.4.1 COÛT DE LA SIGNALISATION PAR RÉCEPTEUR

Pour évaluer le nombre de messages de signalisation, nous devons mesurer le nombre d'abonnements et de désabonnements émis par chaque récepteur :

- *Le nombre de désabonnements* : pour les mécanismes à débits dynamiques, la réduction du débit est gérée automatiquement par la source et un récepteur ne doit se désabonner d'un groupe que lorsque ce dernier est muet. Ainsi, si K groupes deviennent muets toutes les TSD secondes, alors pendant le temps total de la session ($time_{total}$ secondes), chaque récepteur doit effectuer un nombre de désabonnement correspondant à nb_{leave} :

$$\begin{aligned} nb_{leave} &= (K * nb\_TSD_{total}) \\ \text{Avec :} & \\ nb\_TSD_{total} &= \frac{time_{total}}{TSD} \end{aligned}$$

- *Le nombre d'abonnements* : le nombre L d'abonnements à des groupes en un TSD caractérise différents comportements :
  - Si $0 \leq L < K$, alors le récepteur est en train de réduire son débit.
  - Si $L = K$, alors le récepteur maintient son débit.
  - Si $L > K$, alors le récepteur augmente son débit.

  Pour trouver le nombre d'abonnements émis, nous pouvons constater que pour recevoir un débit correspondant à L_1 groupes au bout de $time_1$ secondes, le récepteur doit faire un nombre d'abonnements correspondant à $nb\_join_{L_1}$:

$$\begin{aligned} nb\_join_{L_1} &= L_1 + (K * nb\_TSD_1) \\ \text{Avec :} & \\ nb\_TSD_1 &= \frac{time_1}{TSD} \end{aligned}$$

  La partie $(K * nb\_TSD_1)$ repose sur le fait que pour chaque TSD, un récepteur doit s'abonner à K groupes pour conserver son débit. Abonnements auxquels il faut ajouter L_1 abonnements pour obtenir le débit désiré.
  En outre, si le récepteur réduit ensuite son débit à L_2 groupes en $time_2$ secondes, le récepteur doit alors faire un nombre d'abonnements correspondant à $nb\_join_{L_2}$:

$$\begin{aligned} nb\_join_{L_2} &= (L_2 - L_1) + (K * nb\_TSD_2) \\ \text{Avec :} & \\ L_2 &< L_1 \\ nb\_TSD_2 &= \frac{time_2}{TSD} \\ time_2 &\geq \frac{L_1 - L_2}{K} * TSD \end{aligned}$$

Dans ce cas, $(L_2 - L_1)$ représente le nombre de canaux qui doivent devenir muets afin que le débit réduit ne soit pas compensé par les réabonnements à de nouveaux canaux. Ainsi, le récepteur peut réduire son débit au niveau désiré. Pendant cette réduction du débit, le récepteur "économise" donc des abonnements.
En définitive, nous pouvons montrer que le nombre d'abonnements est uniquement relatif à deux paramètres, qui sont :

1. le niveau du groupe actuellement abonné
2. et la durée totale de la session $time_{total}$.

En effet, toutes les fluctuations précédentes du nombre de groupes abonnés s'annulent les unes les autres, car les abonnements effectués en plus pour augmenter le débit, sont ensuite économisés pour le réduire de nouveau :

$$\begin{aligned} nb_{join} &= nb\_join_{L_1} + nb\_join_{L_2} \\ \Leftrightarrow nb_{join} &= L_2 + (K * nb\_TSD_{total}) \end{aligned}$$

Finalement, le nombre total d'abonnements et de désabonnements $nb_{leave\_\&\_join}$ est donc :

$$nb_{leave\_\&\_join} = L_2 + (2K * nb\_TSD_{total})$$

Ce qui signifie que la fréquence de la signalisation engendrée se calcule par :

$$\begin{aligned} freq_{leave\_\&\_join} &:= \frac{nb_{leave\_\&\_join}}{time_{total}} \\ \Leftrightarrow freq_{leave\_\&\_join} &= \frac{L_2}{time_{total}} + \frac{2K}{TSD} \end{aligned}$$

Enfin, comme

$$\lim_{time_{total} \to \infty} \frac{L_2}{time_{total}} = 0$$

et que dans la plupart des cas $L_2 \ll time_{total}$, la fréquence de la signalisation peut être approximée par :

$$\Leftrightarrow freq_{leave\_\&\_join} \simeq \frac{2K}{TSD} \tag{4.1}$$

Ce résultat démontre un avantage important de l'utilisation des groupes à débits dynamiques par rapport aux groupes à débits statiques. En effet, les groupes à débits dynamiques engendrent un taux de signalisation régulier et connu au préalable, alors que les groupes à débits statiques engendrent un taux de signalisation correspondant à la fluctuation du débit du récepteur.

4.1.4.2 COÛT DE LA SIGNALISATION POUR N RÉCEPTEURS

Pour une même session, N récepteurs connectés en *IGMPv3* au même routeur peuvent générer N messages *IGMP_report* pour s'abonner à un même groupe. Cependant, seul le premier reçu crée une nouvelle entrée dans la table de routage *multicast* et est propagé au prochain routeur en amont par un message *PIM_join*. Bien entendu, un comportement identique a lieu quand un routeur reçoit plusieurs *PIM_join* des routeurs en aval.
De la même façon, N récepteurs connectés au même routeur peuvent générer N messages *IGMP_report* pour se désabonner du même groupe. Cependant, seul le dernier reçu provoque un changement d'état du routeur. Celui-ci se désabonne alors du groupe et propage alors au prochain routeur en amont un message *PIM_prune*. Bien entendu, un comportement identique a lieu quand un routeur reçoit plusieurs *PIM_prune* des routeurs en aval. Par ailleurs, même si les récepteurs ne sont pas synchronisés entre eux, le fait d'attendre le dernier message de désabonnement n'est pas problématique, car un récepteur ne quitte un groupe que lorsque ce dernier est muet. Ainsi pendant cette attente, aucun paquet supplémentaire ne traverse le réseau et le routeur garde simplement l'entrée correspondante dans sa table de routage *multicast* un peu plus longtemps.
Grâce à ce phénomène d'agrégation des messages d'abonnement et de désabonnement, la signalisation générée est indépendante du nombre de récepteurs abonnés au groupe. En effet, avec *PIM-SM* un lien entre routeurs ne voit ainsi passer qu'une seule copie pour un abonnement ou un désabonnement et un routeur voit donc le nombre de messages de signalisation borné par son nombre d'interfaces.

4.2 PLATEFORMES D'EXPÉRIMENTATIONS

Toujours dans le but d'évaluer et de comparer les différents protocoles de contrôle de congestion *multicast*, nous avons mis en place plusieurs plateformes d'expérimentations.
En effet, les évaluations existantes des différents protocoles à débit dynamique (*FLID-DL* et *WEBRC*) ont uniquement été réalisées par simulation, ce qui introduit des hypothèses pour simplifier la réalité. Or, il arrive que ces simplifications soient suffisamment importantes au point d'introduire un biais. Ainsi, pour pouvoir évaluer ces protocoles dans des conditions réalistes, nous avons implémenté sous licence libre la librairie *libmcc* [Lucas 10a] comprenant trois protocoles de contrôle de congestion *multicast* :

- L'un à débits statiques : *FLID-SL* dont l'implémentation existe également dans la librairie MCL [INRIA 06].

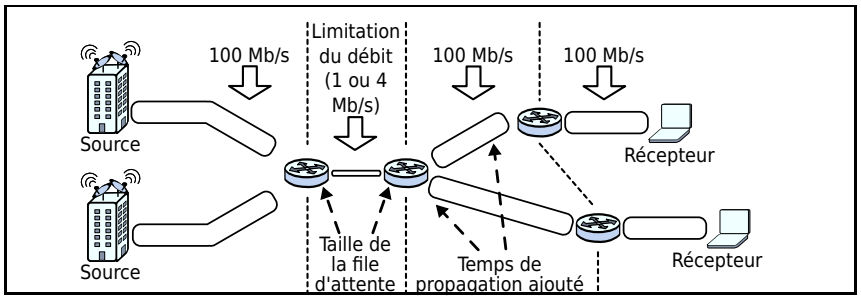


FIGURE 4.1 – Un exemple de la plateforme locale avec 2 émetteurs et 2 récepteurs

- Les deux autres à débits dynamiques : *FLID-DL* et *WEBRC* dont notre implémentation est à notre connaissance la seule actuellement accessible publiquement pour ces protocoles.

Les flux *TCP* concurrents utilisent l'algorithme de contrôle de congestion *new-RENO* avec *SACK* et serviront ainsi de flux témoin, notamment pour l'évaluation de l'équité.

4.2.1 PLATEFORME LOCALE

Cette plateforme est dédiée à des expérimentations où seuls des flux générés artificiellement peuvent circuler et où tous les paramètres (débit, durée, quantité de données échangée, etc.) sont maîtrisés. Cette maquette permet donc de réaliser des expériences sans avoir d'aléas ou de biais venant d'éléments non contrôlés.
Cette maquette (cf. figure 4.1) est composée de routeurs qui sont des stations Linux faisant tourner :

- le service de routage *XORP* (*PIM-SM*, *IGMPv3*),
- le module *NETwork EMulation* (*NETEM*) pour émuler différents *Temps de Propagation Ajouté (TPA)*.
- et le module *Token Bucket Filter* (*TBF*) pour modifier le débit du lien faible ainsi que la taille des files d'attente des routeurs.

Nous découvrirons dans la suite de ce manuscrit que le temps d'abonnement n'est pas forcément équivalent au *RTT* du réseau. Or, une particularité de cette plateforme est l'utilisation de *XORP* comme service de routage qui permet d'avoir un temps d'abonnement équivalent au *RTT*.

4.2.2 PLATEFORME INTERNET STRASBOURG-TRONDHEIM

Cette plateforme permet de réaliser des tests sur un réseau de production où des flux tiers peuvent venir perturber les expérimentations, améliorant ainsi le degré de réalisme des expérimentations.

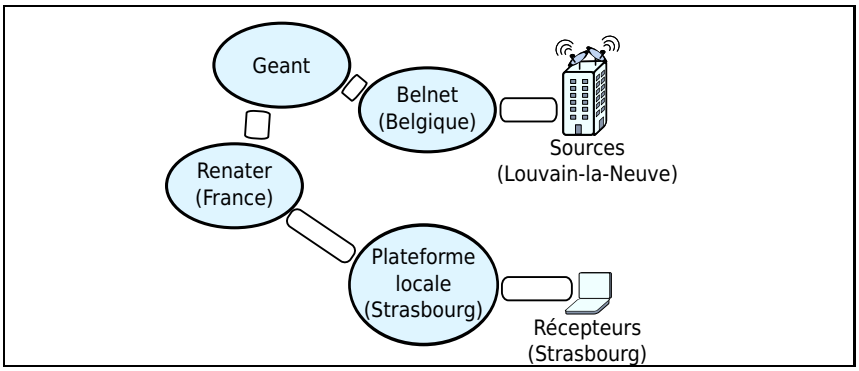


FIGURE 4.2 – Plateforme internet entre Strasbourg et Louvain-la-Neuve

Cette plateforme est déployée au dessus du *m6bone* en passant par des routeurs de production reliant la France à la Norvège et est composée de 17 routeurs. La signalisation *multicast* est assurée avec *PIM-SM IPv6* et un lien faible proche des récepteurs a une capacité d'1 Mb/s mis en place par *traffic-shaping*. Le temps d'adhésion *multicast* y est relativement long ($\approx$ 2200 ms) comparé au *RTT* ($\approx$ 60 ms).

4.2.3 PLATEFORME INTERNET STRASBOURG-LOUVAIN

Cette plateforme a le même but que celle entre Strasbourg et Trondheim et est venue la remplacer pour les années 2009-2010.
Cette plateforme est déployée en *SSM IPv4* en passant par des routeurs de production reliant la France à la Belgique (cf. figure 4.2). Les sources sont placées à Louvain-la-Neuve (Belgique) et sont joignables via le *mbone*. Les flux traversent 13 routeurs de coeur de Belnet (Belgique), Geant (Belgique - Royaume Uni - France) et Renater (France) avant de traverser ceux de la plateforme locale de Strasbourg où les récepteurs sont connectés. Les flux tiers avec et sans mécanisme de contrôle de congestion qui traversent les routeurs de cette plateforme permettent également d'augmenter le réalisme des tests. De plus, cette plateforme dispose aussi d'un temps d'adhésion (300-500 ms) plus long que le *RTT* (30 ms).

4.3 SCENARII PRINCIPAUX

Pour évaluer et mettre en évidence les différentes propriétés et comportements des protocoles de contrôle de congestion, nous avons défini des scenarii pour former un banc d'essais complet.
L'utilisation de la signalisation *multicast* et des groupes à débit dynamique rendent les protocoles de contrôle de congestion *multicast* moins réactifs. Ils sont donc adaptés pour des transferts de longue

durée. C'est pourquoi, chaque test est effectué pendant 10 minutes.

4.3.1 1 FLUX *MULTICAST* FACE À 1 FLUX *TCP*

Ces tests sont réalisés pour montrer l'équité sur le long terme entre 1 flux *multicast* et 1 flux *TCP*. Le but ici est d'avoir un panel de résultats pour une grande combinaison de paramètres influant sur les conditions du réseau en faisant varier les paramètres du tableau 4.1.

Paramètres	Valeurs
Débit du lien faible (Mb/s)	1 ou 4
Files d'attente (# paquets)	5, 10, 20, 40, 60, 80 ou 100
TPA dans chaque sens (ms)	0, 25, 50, 75 ou 100

TABLE 4.1 – Paramètres du scénario : "1 flux *multicast* face à 1 flux *TCP*"

4.3.2 1 FLUX *MULTICAST* FACE À N FLUX *TCP*

Ces tests servent à observer si l'équité évolue en fonction du nombre de flux *TCP* concurrents. Ainsi, 1 flux *multicast* est mis en concurrence avec 1 à 4 flux *TCP*. Les conditions du réseau sont obtenues selon les combinaisons des paramètres données par le tableau 4.2.

Paramètres	Valeurs
Débit du lien faible (Mb/s)	1
Files d'attente (# paquets)	5, 20 ou 80
TPA dans chaque sens (ms)	0 ou 100

TABLE 4.2 – Paramètres du scénario : "1 flux *multicast* face à N flux *TCP*"

4.3.3 M FLUX *MULTICAST* FACE À N FLUX *TCP*

Ce scénario expérimente 5 longs flux concurrents, de telle sorte que : N *TCP* versus $(5 - N)$ *multicast*, $N \in [0..4]$. Cela permet de voir l'évolution de l'équité en fonction de la proportion de flux concurrents d'un certain type : *TCP* ou *multicast*. Les conditions du réseau sont obtenues selon les combinaisons des paramètres données par le tableau 4.3.

4.3.4 TRAFIC EN BRUIT DE FOND

Ce scénario est utilisé pour reproduire un modèle de trafic plus réaliste, comprenant :

Paramètres	Valeurs
Débit du lien faible (Mb/s)	1 ou 4
Files d'attente (# paquets)	10, 25, 50, 75 ou 100
TPA dans chaque sens (ms)	0, 20 ou 50

TABLE 4.3 – Paramètres du scénario : M flux *multicast* face à N flux *TCP*

- Du trafic en bruit de fond, composé de multiples connexions *TCP* de courte durée, telles celles utilisées pour la navigation sur le *web*. Le temps de démarrage de ces flux est défini par un processus Poissonnien générant en moyenne 100 connexions par test. De plus, chaque connexion envoie une quantité de données suivant une distribution exponentielle, de telle sorte que 85% de ces connexions transportent moins de 6 Ko de données.
- Deux long flux (*TCP* ou *multicast*) : $2 * TCP$, $TCP + multicast$ ou $2 * multicast$.

Ce scénario correspond à la situation d'un abonné *ADSL* où le lien faible est le lien le reliant à un *Digital Subscriber Line Access Multiplexer (DSLAM)*. En effet, dans cette situation la majorité des flux sont courts et produits par la navigation sur le *web*, alors que de rares flux de longue durée se partagent la bande passante, comme ceux d'un service de partage de fichier pour l'*unicast* ou ceux d'un flux télévisuel pour le *multicast*. Les conditions du réseau sont obtenues selon les combinaisons des paramètres données par le tableau 4.4.

Paramètres	Valeurs
Débit du lien faible (Mb/s)	1 ou 4
Files d'attente (# paquets)	10, 25, 50, 75 ou 100
TPA dans chaque sens (ms)	0, 20 ou 50

TABLE 4.4 – Paramètres du scénario : "Trafic en bruit de fond"

4.3.5 *RTT* HÉTÉROGÈNES

Ce scénario montre le comportement pour partager la bande passante entre 1 flux *multicast* ou *TCP* et 1 flux disposant d'un *RTT* différent. Deux flux $(s1, s2)$ sont diffusés via deux chemins mais qui traversent le même lien faible : le chemin $s1$ ne voit pas son temps de propagation rallongé, alors que le chemin $s2$ voit son temps de propagation rallongé de 50 ms dans chaque direction. Cela permet de mettre en évidence l'équité obtenue, quand des flux de provenances différentes se croisent pour se partager la bande passante. Les conditions du réseau sont obtenues selon les

combinaisons des paramètres données par le tableau 4.5.

Paramètres	Valeurs
Débit du lien faible (Mb/s)	1 ou 4
Files d'attente (# paquets)	10, 25, 50, 75 ou 100
TPA dans chaque sens (ms)	0 ms pour un chemin et 50 ms pour l'autre

TABLE 4.5 – Paramètres du scénario : "*RTT* hétérogènes"

4.3.6 TEMPS DE CONVERGENCE

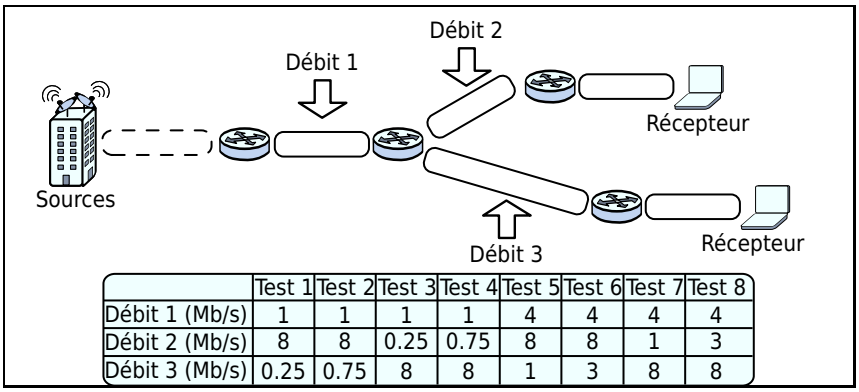
Ce scénario est conçu pour évaluer le temps de convergence au démarrage d'un flux *multicast* unique. Ce type de test permet d'évaluer le temps nécessaire avant d'obtenir un débit proche du débit équitable ce qui, pour une application vidéo, correspond au temps pour lequel l'utilisateur doit patienter avant de recevoir une qualité d'image correspondant aux capacités de son réseau. Ces tests sont réalisés avec et sans trafic de bruit de fond (cf. § 4.3.4). Les conditions du réseau sont obtenues selon les combinaisons des paramètres données par le tableau 4.6.

Paramètres	Valeurs
Débit du lien faible (Mb/s)	1 ou 4
Files d'attente (# paquets)	10, 25, 50, 75 ou 100

TABLE 4.6 – Paramètres du scénario : "Temps de convergence"

4.3.7 LIENS FAIBLES MULTIPLES

Ce scénario montre le partage de bande passante obtenu quand deux flux traversent plusieurs liens faibles dont un en commun (cf. figure 4.3). De cette manière, un flux *multicast* traverse les liens faibles dont les débits sont limités à *Débit 1* et à *Débit 2*, alors qu'un flux *TCP* est limité par les liens faibles de débit *Débit 1* et *Débit 3*. Bien entendu, $Débit\ 2 + Débit\ 3 > Débit\ 1$.
De plus, ce scénario permet d'évaluer la rapidité de convergence (*CT*) quand le flux *multicast* est en compétition avec un long flux *TCP* pouvant générer des rafales de paquets.
Cette suite de tests est effectuée avec les débits indiqués à la figure 4.3 ainsi que les tailles de tampon des routeurs connectés au lien faible de 10, 25, 50, 75 et 100 paquets. De plus, les flux concurrents sont regroupés selon les couples suivants : *TCP* vs. *TCP*, *multicast* vs. *TCP*, *multicast* vs. *multicast*. Tous ces tests sont également réalisés avec et sans trafic de bruit de fond (cf. § 4.3.4).



	Test 1	Test 2	Test 3	Test 4	Test 5	Test 6	Test 7	Test 8
Débit 1 (Mb/s)	1	1	1	1	4	4	4	4
Débit 2 (Mb/s)	8	8	0.25	0.75	8	8	1	3
Débit 3 (Mb/s)	0.25	0.75	8	8	1	3	8	8

FIGURE 4.3 – Paramètres des débits du scénario "Liens faibles multiples"

4.3.8 VARIATION DU NOMBRE DE FLUX CONCURRENTS

Ce scénario met en valeur le temps de convergence d'un flux *multicast* pour atteindre un nouveau débit équitable suite au changement du nombre de flux concurrents. Plus précisément, un flux *multicast* et un second flux (*multicast* ou *TCP*) sont actifs durant toute la durée du test (de 0 à 600 secondes) et un troisième flux *TCP* apparaît ou disparaît au milieu du test (300 secondes).
Dans ce scénario, deux temps de convergence peuvent être évalués :

- Entre 0 et 300 secondes : le temps de convergence correspond au démarrage du flux pour atteindre son débit équitable. Les résultats de ce temps de convergence sont déjà pris en compte dans le scénario avec des liens faibles multiples et ne sera donc pas analysé plus avant pour ce scénario.
- Entre 300 et 600 secondes : le temps de convergence correspond au temps pour le flux d'atteindre son nouveau débit équitable suite à l'apparition ou la disparition du troisième flux. Le critère du temps de convergence est important car il représente la rapidité du protocole pour acquérir ou libérer une partie de la bande passante.

Les conditions du réseau sont obtenues selon les combinaisons des paramètres données par le tableau 4.7.

Paramètres	Valeurs
Débit du lien faible (Mb/s)	1 ou 4
Files d'attente (# paquets)	10, 25, 50, 75 ou 100
TPA dans chaque sens (ms)	0, 20 ou 50

TABLE 4.7 – Paramètres du scénario : "Variation du nombre de flux concurrents"

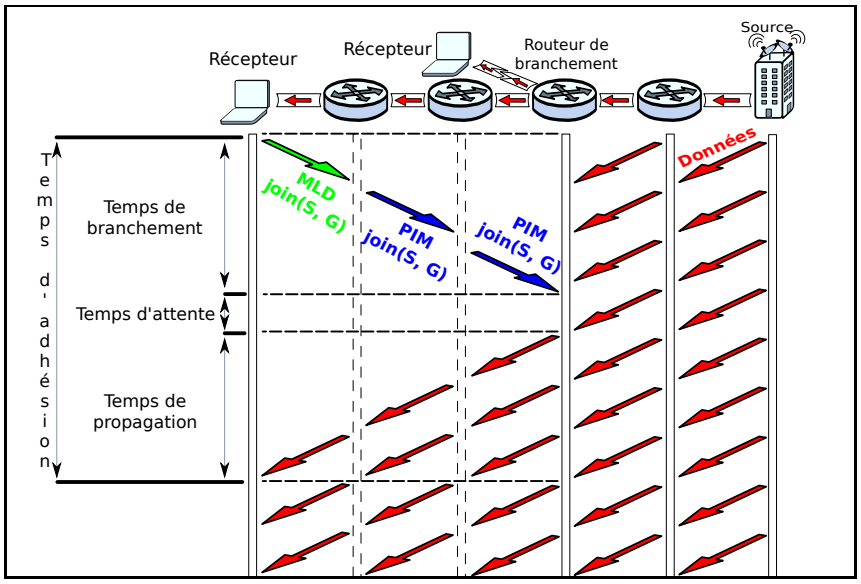


FIGURE 4.4 – Description du temps d'adhésion

4.4 ÉVALUATION DU TEMPS D'ADHÉSION

Les différentes méthodes de contrôle de congestion *multicast* présentées dans l'état de l'art se basent sur l'hypothèse implicite que le temps d'adhésion est négligeable et n'est fonction que du temps de propagation. Cependant, certains chercheurs et utilisateurs du *m6bone*[1] ont remarqué que le comportement réel du temps d'adhésion diffère de cette logique.

La littérature [Quinn 01] définit le temps d'adhésion comme étant le délai séparant le moment où un récepteur envoie un *IGMP /MLD_report* pour s'abonner à un groupe *multicast* et le moment où ce même récepteur reçoit le premier paquet du groupe demandé. Ce temps se compose de trois phases (figure 4.4) :

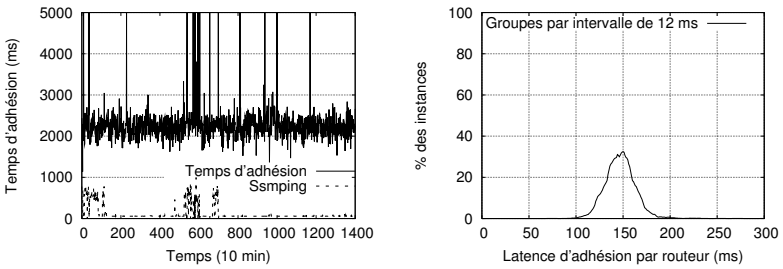
- *Un temps de branchement* :
  Il correspond à l'intervalle de temps qui s'écoule entre la demande d'abonnement à un groupe *multicast* et le moment où la nouvelle branche créée par cette demande est connectée à l'arbre existant du même groupe. Ce temps se découpe lui-même en un temps de signalisation entre récepteur et routeur (*IGMP/MLD_report*) et un temps de signalisation entre routeurs (*PIM_join*). Pour la suite, nous utiliserons le terme *branchement* pour désigner le fait de réaliser une connexion entre la nouvelle branche et l'arbre existant. De même, nous nommerons *routeur de branchement* le routeur au sein duquel se déroule le branchement.

1. Le *m6bone* est une part d'Internet *IPv6* (le *6bone*) où le *multicast* est activé.

- *Un temps d'attente* :
  Il correspond au temps séparant la réalisation du branchement et l'arrivée du prochain paquet du canal au niveau du routeur de branchement.
- *Un temps de propagation* :
  Il correspond à l'intervalle de temps entre l'arrivée du prochain paquet du canal sur le routeur de branchement et la réception du même paquet par le récepteur.

Cette décomposition nous permet de définir les éléments qui peuvent faire varier le temps d'adhésion. La signalisation devant se propager saut par saut jusqu'au routeur de branchement, plus le routeur de branchement est éloigné du récepteur, plus le temps d'adhésion sera important. En second point, on peut remarquer que la probabilité que le routeur de branchement soit proche varie en fonction du nombre de récepteurs actuellement abonnés à la même session. En effet, plus le nombre de récepteurs est élevé, plus l'arbre *multicast* existant a de chances de recouvrir une grande partie du chemin entre la source et le récepteur, et plus le routeur de branchement a de chances d'être proche du récepteur. Troisièmement, les mesures du temps d'adhésion présentent des différences de résultats en fonction de la fréquence des envois des paquets par la source, qui influe sur le temps d'attente. Ainsi, plus le débit de la source est important, moins les mesures des temps d'adhésion fluctuent. Enfin le temps d'adhésion varie en fonction du traitement des messages d'adhésion dans les routeurs, point que nous détaillons ci-dessous.
Comme nous l'avons présenté dans la section précédente, le temps d'adhésion est sensible à plusieurs paramètres. Le traitement des messages d'adhésion au sein des routeurs ne semble pas être effectué de la même manière suivant l'implémentation utilisée. Plusieurs observations concordent, affirmant que contrairement aux implémentations telles que *MRD*6 ou *XORP*[2], les routeurs des grands équipementiers semblent construire relativement lentement l'arbre *multicast*. Pour étudier ce comportement, nous avons effectué des tests sur le m6bone entre Strasbourg et Trondheim en 2006, avec 17 sauts *multicast* entre les machines de tests. Les tests se sont déroulés en *multicast SSM-IPv*6, avec des relevés effectués toutes les 10 minutes par l'outil *multicat* pour le temps d'adhésion et *ssmping* [Venaas 06] pour le temps de propagation.
Les résultats des ces tests sont présentés à la figure 4.5(a). Le temps de propagation est de 50 à 60 ms en temps normal, mais peut atteindre 1000 ms en cas de forte congestion. Le temps d'adhésion reste stable autour de 2.2 secondes, sauf en cas de perte d'un message où ce temps peut alors dépasser les 60 secondes, ce qui corres-

2. *MRD*6 et *XORP* sont des processus de routage *multicast* pour Linux.



(a) Mesures (b) Répartition

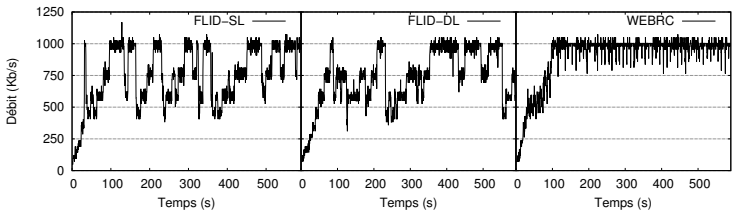
FIGURE 4.5 – Mesures du temps d'adhésion sur le *m6bone*

pond au délai d'envoi périodique de *PIM_join*. Ces 2.2 secondes représentent une éternité en comparaison aux 50 ms du temps de propagation, soit 22 fois l'aller-retour Strasbourg-Trondheim. La courbe de répartitions des temps d'adhésion (figure 4.5(b)) montre qu'un routeur met en moyenne 150 ms pour traiter la demande d'adhésion et la propager au routeur suivant. Ce problème a été discuté pour *IPv*6 dans la liste de diffusion du *m6bone* [Hoerdt 06] et aussi pour *IPv*4 [Namburi 04]. De plus, ce délai n'apparaît que sur certains types de routeurs (par exemple les plus gros routeurs chez Cisco ou Juniper). Une analyse plus fine en utilisant des traces sur Cisco montre qu'un *PIM_join* reçu n'est pas envoyé immédiatement mais est mis en attente pendant un certain temps ce qui permet d'agréger plusieurs messages et de limiter l'utilisation des ressources. Par conséquent, nous devons prendre en considération que le temps d'adhésion n'est pas négligeable et qu'il peut ainsi influer sur le comportement des mécanismes de contrôle de congestion.

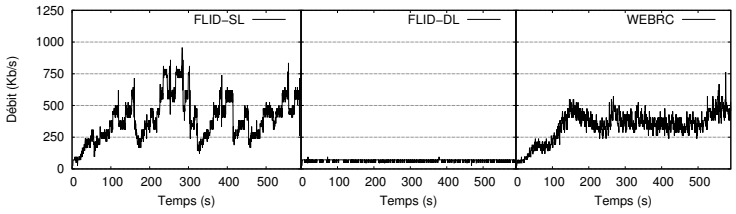
4.5 INFLUENCE DU TEMPS D'ADHÉSION

Afin de vérifier le comportement des mécanismes de contrôle de congestion et notamment face à un temps d'adhésion variable, nous avons procédé à une série de tests pour évaluer les protocoles *FLID-SL*, *FLID-DL* et *WEBRC*.
Pour chaque type de contrôle de congestion, le test se résume à démarrer une session pendant 10 minutes et à observer l'évolution de la bande passante et du temps d'adhésion. Pour observer l'influence de ce dernier, les tests se sont déroulés dans deux environnements :

- Sur la plateforme locale disposant d'un temps d'adhésion proche du *RTT*.



(a) Plateforme locale ($JT \simeq RTT$)



(b) Plateforme Strasbourg-Trondheim ($JT \gg RTT$)

Débit de sessions *FLID-SL*, *FLID-DL* et *WEBRC* en fonction du temps d'adhésion

Plateforme	*FLID-SL*		*FLID-DL*		*WEBRC*	
	BWU	*LR* (%)	*BWU*	*LR* (%)	*BWU*	*LR* (%)
Locale	0.84	0.01	0.76	0.06	0.98	0.01
Strasbourg-Trondheim	0.47	0.21	0.07	0.42	0.38	0.04

(c) Résultats récapitulatifs

FIGURE 4.6 – Résultats de *FLID-SL*, *FLID-DL* et *WEBRC* en fonction du temps d'adhésion

- Sur la plateforme Strasbourg-Trondheim avec 17 routeurs et un temps d'adhésion beaucoup plus long ($\approx$ 2200 ms) que le *RTT* ($\approx$ 60 ms).

La figure 4.6 présente les résultats typiques du débit pour chaque type de contrôle de congestion. Ces résultats montrent des anomalies ainsi que les faiblesses, notamment pour *FLID-DL* et *WEBRC* face à un temps d'adhésion conséquent. Nous allons donc analyser les modifications du comportement de *FLID-SL*, *FLID-DL* et *WEBRC*.

4.5.1 INFLUENCE DU TEMPS D'ADHÉSION POUR *FLID-SL*

Nous avons vu que *FLID-SL* utilise des canaux statiques. Les différents canaux diffusent donc des données de manière constante et un récepteur doit s'abonner au prochain canal pour augmenter son débit. De ce fait un long temps d'adhésion retarde simplement l'arrivée des données. C'est pourquoi *FLID-SL* présente un comportement similaire en local (figure 4.6(a)) et sur le *m6bone* (figure 4.6(b)). Par contre cette lenteur du temps d'adhésion couplée

à quelques pertes isolées nous font obtenir un débit inférieur sur le m6bone (en local $BWU \simeq 0.84$ et $LR \simeq 0.01$ %, sur le m6bone $BWU \simeq 0.47$ et $LR \simeq 0.21$ %).
Ce test sert de référence pour pouvoir comparer les protocoles à canaux statiques avec ceux à canaux dynamiques. Cependant, comme les protocoles à canaux statiques souffrent du problème du temps de désabonnement, ces derniers ne seront plus évalués par la suite. Cela sans compter que la famille des protocoles *FLID-*L* ont un problème d'équité, car le débit obtenu n'est pas inversement propoertionnel au *RTT*.

4.5.2 INFLUENCE DU TEMPS D'ADHÉSION POUR *FLID-DL*

FLID-DL a un fonctionnement comparable à celui de *FLID-SL* en local (figure 4.6(a), $BWU \simeq 0.76$ et $LR \simeq 0.06$ %), mais est totalement différent sur le *m6bone* (figure 4.6(b), $BWU \simeq 0.07$ et $LR \simeq 0.42$ %). Cette différence a pour origine l'utilisation de canaux dynamiques. En effet, un récepteur utilisant *FLID-DL* s'abonne au canal de base pour initier la session, avant de tenter de s'abonner au canal suivant pour augmenter son débit. Or ce canal est dynamique et devient muet au bout de *TSD* secondes, avec une valeur de 0.5 seconde par défaut pour *TSD*. De plus, le temps d'adhésion sur le *m6bone* est de 2.2 secondes et inclut un temps de branchement qui dépasse les *TSD* secondes. Ce temps de branchement supérieur à *TSD* fait qu'aucun paquet du canal demandé ne sera diffusé jusqu'au récepteur. Au changement de *TSI*, le récepteur se désabonne du canal et s'abonne au canal suivant. On se retrouve alors dans la même situation qu'au *TSI* précédent et avec un phénomène qui se répète indéfiniment et donne les résultats de la figure 4.6(b). Par conséquent un temps d'adhésion trop important rend impossible l'utilisation de *FLID-DL*, par un simple problème de démarrage.

4.5.3 INFLUENCE DU TEMPS D'ADHÉSION POUR *WEBRC*

WEBRC présente également une meilleure utilisation de la bande passante en local (figure 4.6(a), $BWU \simeq 0.98$ et $LR \simeq 0.01$ %) que sur la maquette du *m6bone* (figure 4.6(b), $BWU \simeq 0.38$ et $LR \simeq 0.04$ %). Or, *WEBRC* utilise aussi des canaux dynamiques. Comme *FLID-DL*, un canal devient muet toutes les *TSD* secondes, mais par défaut le *TSD* de *WEBRC* vaut 10 secondes. Par conséquent, le problème de démarrage n'apparaîtrait que pour un très grand nombre de sauts (de l'ordre de 60 par exemple) et n'est donc pas visible sur nos tests.
En outre, *WEBRC* présente une autre faiblesse relative au temps d'adhésion. En effet, nous avons vu que *WEBRC* utilise le temps d'adhésion pour le calcul de la bande passante disponible (voir

équation 3.3). Par conséquent le phénomène constaté figure 4.6(b) provient du fait que les temps d'adhésion relevés sont beaucoup plus longs que le temps de propagation. Donc *WEBRC* calcule une bande passante disponible beaucoup plus faible que celle réellement disponible. De plus ce calcul est réalisé pour prévenir des pertes, mais quand l'estimation est trop faible, alors *WEBRC* ne reçoit plus d'informations sur l'état du réseau car il ne provoque alors ni pertes ni variations de propagation des paquets. En effet, le calcul de la bande passante disponible fait que *WEBRC* est plutôt proactif, alors que *FLID-DL* est réactif aux pertes.

4.6 ÉVALUATION DE *WEBRC*

Hormis cette inadéquation de l'utilisation du temps d'adhésion pour calculer l'estimation du *RTT*, nous souhaitons à présent savoir si *WEBRC* est équitable avec *TCP*. Ainsi, nous proposons de réaliser plusieurs séries de tests sur la plateforme locale pour laquelle le temps d'adhésion est semblable au *RTT* et où le problème du § 4.5.3 ne se pose pas.

4.6.1 ÉVALUATION DE LA SIGNALISATION

Dans les tests suivants, chaque session de *WEBRC* génère une signalisation $SO \approx 0.2$ message par seconde, ce qui correspond à la définition de l'équation 4.1.

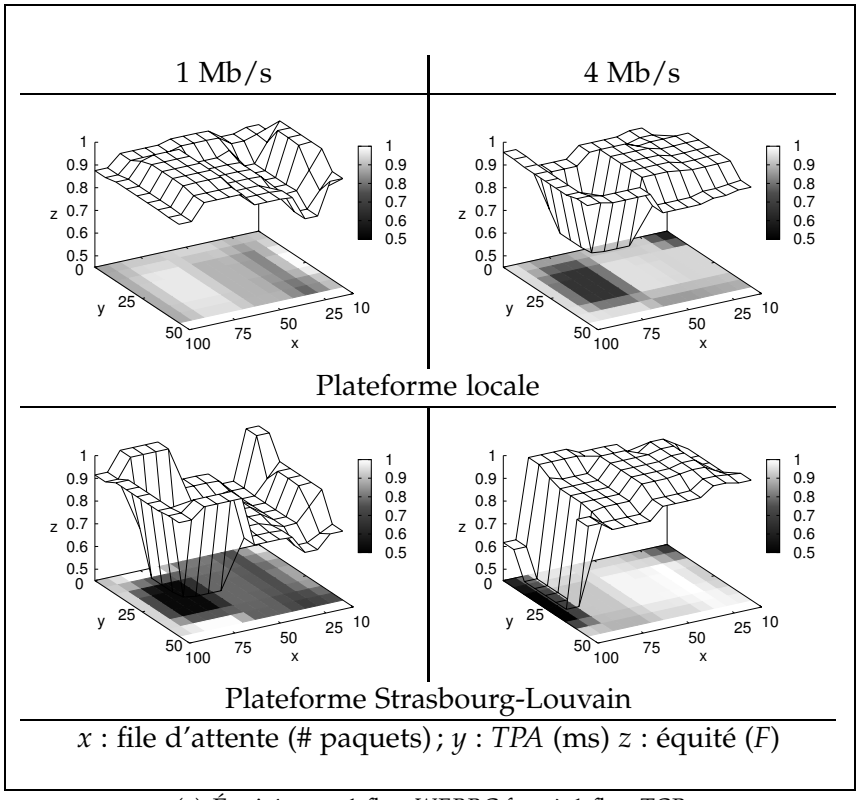
4.6.2 1 FLUX *WEBRC* FACE À 1 FLUX *TCP*

En utilisant la plateforme locale qui ne présente pas le problème du temps d'adhésion, les résultats obtenus (cf. figure 4.7) pour ce scénario (cf. § 4.3.1) montrent que *WEBRC* est généralement assez équitable ($F \simeq 0.90$, $LR_{WEBRC} \simeq 1.02\%$ et $LR_{TCP} \simeq 1.27\%$) face à un unique flux *TCP* concurrent. Ces informations confirment les résultats obtenus en simulation par [Luby 02].
Par contre avec la plateforme *Strasbourg-Louvain*, le problème du temps d'adhésion[3] apparaît, ce qui rend les résultats plus variables et diminue l'équité de *WEBRC* ($F \simeq 0.86$, $LR_{WEBRC} \simeq 0.66\%$ et $LR_{TCP} \simeq 1.46\%$).

4.6.3 1 FLUX *WEBRC* FACE À N FLUX *TCP*

Après avoir expérimenté toutes les combinaisons des paramètres du scénario (cf. § 4.3.1) mettant en concurrence 1 flux *WEBRC* et 1 flux *TCP*, nous avons dégagé 3 comportements pour lesquels

3. Le délai du temps d'adhésion entre Strasbourg et Louvain (300 à 500 ms) est moindre qu'avec Trondheim (plus de 2000 ms).

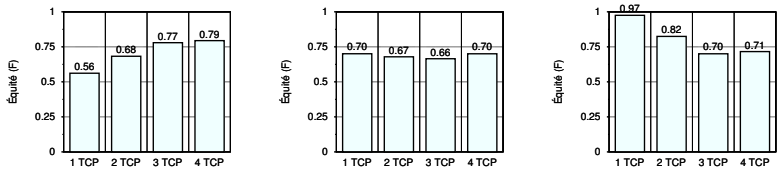


(a) Équité pour 1 flux *WEBRC* face à 1 flux *TCP*

Test	*BWU*	*LR* (%)	*F*	*CT* (s)		*RT* (s)	
				$1Mb/s$	$4Mb/s$	$1Mb/s$	$4Mb/s$
Plateforme locale							
$1 * WEBRC$ $/1 * TCP$	0.99 ($\pm$0.02)	1.02 ($\pm$3.99)	0.90 ($\pm$0.23)	106 ($\pm$66)	180 ($\pm$173)	1 ($\pm$1)	1 ($\pm$1)
Plateforme Strasbourg-Louvain							
$1 * WEBRC$ $/1 * TCP$	0.98 ($\pm$0.05)	0.66 ($\pm$1.25)	0.85 ($\pm$0.32)	51 ($\pm$47)	156 ($\pm$276)	1 ($\pm$1)	1 ($\pm$1)

(b) Résultats récapitulatifs

FIGURE 4.7 – Résultats pour 1 flux *WEBRC* face à 1 flux *TCP*



(a) $TPA < 1$ ms, tampon 5 paquets (b) TPA 100 ms, tampon 20 paquets (c) TPA 100 ms, tampon 80 paquets

Équité pour 1 flux *WEBRC* face à N flux *TCP*

Test	*BWU*	*LR* (%)	*F*	*CT* (s)		*RT* (s)	
				$1Mb/s$	$4Mb/s$	$1Mb/s$	$4Mb/s$
Plateforme locale							
$1 * WEBRC$ / $1 * TCP$	0.99 ($\pm$0.0)	3.02 ($\pm$4.66)	0.75 ($\pm$0.24)	62 ($\pm$78)	*N.A.*	1 ($\pm$1)	*N.A.*
$1 * WEBRC$ / $2 * TCP$	0.99 ($\pm$0.01)	3.14 ($\pm$4.77)	0.73 ($\pm$0.2)	66 ($\pm$152)	*N.A.*	1 ($\pm$1)	*N.A.*
$1 * WEBRC$ / $3 * TCP$	0.99 ($\pm$0.0)	3.32 ($\pm$4.7)	0.71 ($\pm$0.15)	52 ($\pm$127)	*N.A.*	1 ($\pm$0)	*N.A.*
$1 * WEBRC$ / $4 * TCP$	0.98 ($\pm$0.01)	3.23 ($\pm$5.02)	0.73 ($\pm$0.17)	42 ($\pm$84)	*N.A.*	1 ($\pm$0)	*N.A.*

(d) Résultats récapitulatifs

FIGURE 4.8 – Résultats pour 1 flux *WEBRC* face à N flux *TCP*

nous avons observé l'équité de *WEBRC* en fonction du nombre de flux *TCP* concurrents (cf. § 4.3.2) :

- Le comportement 1 apparaît quand les tampons des routeurs sont très petits, comme par exemple à la figure 4.8(a) avec un tampon de 10 paquets. *WEBRC* présente un taux de pertes d'environ 8% et il sous-utilise la bande passante. Par contre dans cette situation, *WEBRC* arrive à profiter de l'augmentation de la concurrence entre flux *TCP* pour converger légèrement vers le débit équitable, mais en reste très éloigné même dans le meilleur des cas : $F \simeq 0.79$ avec 4 flux *TCP* concurrents. Par conséquent, le comportement de *WEBRC* présente un dysfonctionnement qui montre les limitations de cette méthode quand les tampons sont sous-dimensionnés. Cependant, on peut considérer ce scénario comme étant rare et fortement improbable au vu des configurations actuelles des routeurs qui par défaut utilisent bien plus d'espace mémoire pour les files d'attente.
- Le comportement 2 (voir figure 4.8(b)) apparaît avec des files d'attente de plus grande taille et un temps aller entre la source et le récepteur, ou *One Way Delay (OWD)*, inférieur à environ 350 ms. Dans ces conditions, *WEBRC* présente un taux de pertes inférieur à 4%, mais les flux utilisent 4 fois

plus de bande passante que les flux *TCP*. Cependant, dans cette situation le comportement de *WEBRC* est insensible au nombre de flux *TCP* concurrents et son équité ne dépasse pas la barre des $F \simeq 0.70$. Ce comportement montre que l'estimation de *WEBRC* n'est pas précise et ce même pour les configurations de réseaux les plus courantes.

- Le troisième comportement (voir figure 4.8(c)) apparaît quand *OWD* est supérieur à environ 350 ms. *WEBRC* est relativement équitable, même si cette équité diminue quand le nombre de flux *TCP* concurrents augmente. Dans les deux cas le taux de pertes est toujours inférieur à 4%. *WEBRC* est donc finalement plus équitable pour des réseaux à très longs délais.

Le tableau 4.8(d) montre que de façon générale l'équité de *WEBRC* est très variable et que ce dernier ne réagit pas de la même manière pour un nombre de flux concurrents donné en fonction des conditions réseau.

4.6.4 M FLUX *WEBRC* FACE À N FLUX *TCP*

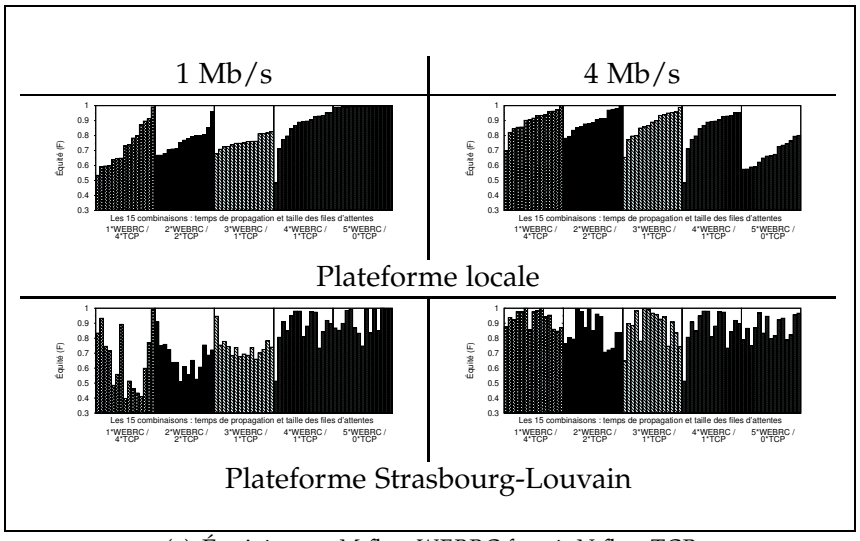
Les résultats (cf. figure 4.9) pour ce scénario (cf. § 4.3.3) montrent que *WEBRC* n'est pas équitable ($0.82 \leq F \leq 0.86$) et est instable quand il est en compétition avec plusieurs autres longs flux. De plus, le taux de pertes de *WEBRC* croît proportionnellement au nombre de flux *WEBRC* en compétition : de $LR_{1*WEBRC+4*TCP} \simeq 3.18\%$ à $LR_{5*WEBRC} \simeq 8.6\%$.
Cependant, le comportement de *WEBRC* en fonction de la proportion de flux $WEBRC/TCP$ dépend des conditions réseaux. En effet, les résultats de la plateforme locale (cf. figure 4.9(a)) montrent qu'à 1Mb/s et à 4Mb/s l'équité de *WEBRC* diminue respectivement avec la diminution ou l'augmentation du nombre d'instances de *TCP*.

4.6.5 TRAFIC EN BRUIT DE FOND

La figure 4.10(a) montre que le "bruit de fond" généré par les flux *TCP* de courte durée perturbe le comportement de *WEBRC*. Or, nous avons observé jusqu'à présent que l'équité des flux *WEBRC* est variable en fonction de certains paramètres comme le nombre ou le type de flux concurrents. Cependant, les résultats obtenus avec le bruit de fond (cf. figure 4.10(a) et 4.10(b)) montrent que :

- L'équité varie presque uniquement entre 2 niveaux extrêmes : soit très équitable, soit totalement inéquitable [4].
- Dans le cas où *WEBRC* n'est pas équitable avec *TCP*, le flux monopolisant la bande passante n'est pas toujours du même

4. obtenir une équité de 0.5 selon l'indice de *Jain* signifie qu'un flux utilise toute la bande passante alors que le second est à un débit nul.

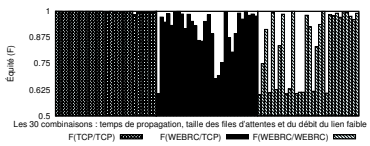


(a) Équité pour M flux *WEBRC* face à N flux *TCP*

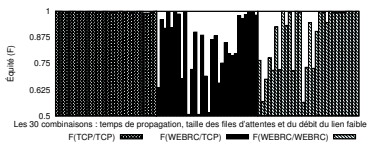
Test	BWU	LR (%)	F	CT (s)		RT (s)	
				$1Mb/s$	$4Mb/s$	$1Mb/s$	$4Mb/s$
Plateforme locale							
$1*WEBRC$ $/4*TCP$	0.99 ($\pm$0.0)	3.18 ($\pm$6.93)	0.82 ($\pm$0.22)	58 ($\pm$54)	110 ($\pm$80)	1 ($\pm$4)	1 ($\pm$2)
$2*WEBRC$ $/3*TCP$	0.99 ($\pm$0.01)	3.42 ($\pm$5.27)	0.83 ($\pm$0.16)	53 ($\pm$40)	119 ($\pm$90)	1 ($\pm$0)	1 ($\pm$0)
$3*WEBRC$ $/2*TCP$	0.99 ($\pm$0.01)	4.06 ($\pm$4.89)	0.81 ($\pm$0.16)	51 ($\pm$33)	101 ($\pm$73)	1 ($\pm$2)	1 ($\pm$2)
$4*WEBRC$ $/1*TCP$	0.98 ($\pm$0.02)	6.17 ($\pm$6.07)	0.86 ($\pm$0.15)	43 ($\pm$36)	104 ($\pm$68)	1 ($\pm$1)	1 ($\pm$0)
$5*WEBRC$ $/0*TCP$	0.94 ($\pm$0.07)	8.6 ($\pm$8.65)	0.84 ($\pm$0.26)	21 ($\pm$15)	64 ($\pm$40)	123 ($\pm$114)	5 ($\pm$6)
Plateforme Strasbourg-Louvain							
$1*WEBRC$ $/4*TCP$	0.99 ($\pm$0.01)	2.46 ($\pm$8.33)	0.8 ($\pm$0.38)	59 ($\pm$59)	117 ($\pm$290)	1 ($\pm$0)	1 ($\pm$3)
$2*WEBRC$ $/3*TCP$	0.99 ($\pm$0.02)	2.34 ($\pm$4.6)	0.76 ($\pm$0.23)	50 ($\pm$40)	78 ($\pm$116)	1 ($\pm$0)	1 ($\pm$0)
$3*WEBRC$ $/2*TCP$	0.98 ($\pm$0.02)	2.95 ($\pm$3.39)	0.81 ($\pm$0.18)	46 ($\pm$51)	109 ($\pm$123)	1 ($\pm$1)	1 ($\pm$1)
$4*WEBRC$ $/1*TCP$	0.98 ($\pm$0.03)	3.99 ($\pm$3.89)	0.85 ($\pm$0.13)	48 ($\pm$124)	104 ($\pm$63)	1 ($\pm$1)	1 ($\pm$0)
$5*WEBRC$ $/0*TCP$	0.93 ($\pm$0.05)	5.69 ($\pm$6.74)	0.89 ($\pm$0.14)	14 ($\pm$6)	72 ($\pm$50)	14 ($\pm$14)	1 ($\pm$1)

(b) Résultats récapitulatifs

FIGURE 4.9 – Résultats pour M flux *WEBRC* face à N flux *TCP*



(a) Plateforme locale



(b) Plateforme Strasbourg-Louvain

Équité de *WEBRC* avec du trafic en bruit de fond

Test	*BWU*	*LR* (%)	*F*	*CT* (s)		*RT* (s)	
				$1Mb/s$	$4Mb/s$	$1Mb/s$	$4Mb/s$
Plateforme locale							
$2 * WEBRC$	0.89 ($\pm$0.15)	1.31 ($\pm$1.05)	0.84 ($\pm$0.23)	55 ($\pm$49)	96 ($\pm$108)	4 ($\pm$21)	4 ($\pm$8)
$1 * WEBRC$ $/1 * TCP$	0.97 ($\pm$0.02)	1.1 ($\pm$3.96)	0.91 ($\pm$0.24)	96 ($\pm$67)	186 ($\pm$127)	1 ($\pm$2)	1 ($\pm$0)
$2 * WEBRC$	0.83 ($\pm$0.18)	0.88 ($\pm$1.26)	0.89 ($\pm$0.32)	27 ($\pm$19)	111 ($\pm$107)	2 ($\pm$5)	1 ($\pm$4)
$1 * WEBRC$ $/1 * TCP$	0.97 ($\pm$0.05)	0.65 ($\pm$1.5)	0.84 ($\pm$0.33)	73 ($\pm$60)	134 ($\pm$406)	1 ($\pm$1)	1 ($\pm$0)

(c) Résultats récapitulatifs

FIGURE 4.10 – Résultats de *WEBRC* avec du trafic en bruit de fond

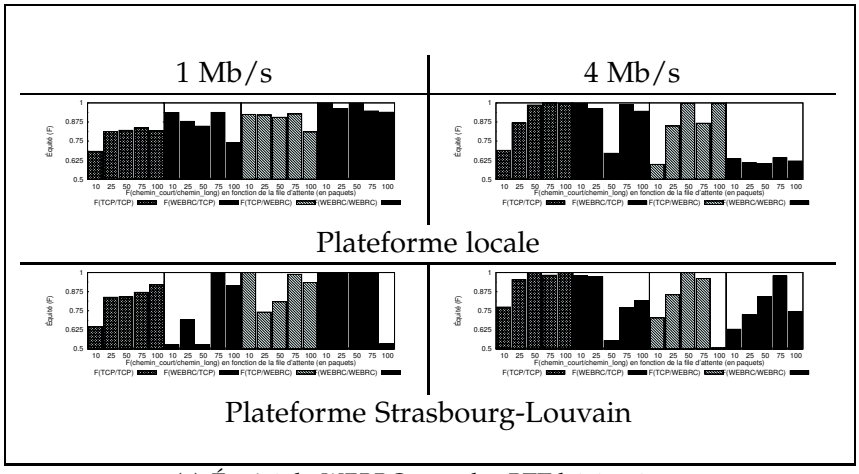
type : dans certains tests *WEBRC* monopolise toute la bande passante, alors que dans d'autres c'est *TCP* qui prive *WEBRC* de bande passante.

- La distribution de l'équité n'est pas relative à un unique paramètre de test utilisé : temps de propagation, taille des files d'attente ou débit du lien.

Les tests où l'iniquité est importante sont des situations particulières où apparaissent des problèmes qui se rapprochent de ceux liés à l'utilisation par *WEBRC* de la formule de calcul du débit de *TFRC*. Ces problèmes proviennent notamment de la façon d'évaluer les pertes par moyenne pondérée et du comportement conservateur de la formule de *TFRC* elle même [Vojnovic 02]. De façon générale selon [Rhee 07], si 2 flux utilisent des débits suffisamment différents, alors ils observeront des taux de pertes différents et cette différence de taux de perte peut alors amplifier la différence de débit. La différence de débit initiale peut notamment provenir du temps de convergence de *WEBRC* qui est à la fois relativement long et surtout très variable (cf. *CT* de la figure 4.10(c)).

4.6.6 *RTT* HÉTÉROGÈNES

Pour ce scénario où les flux ont des *RTT*s hétérogènes (cf. § 4.3.5), *TCP* est connu pour ne pas être équitable. Ainsi, *WEBRC*



(a) Équité de *WEBRC* avec des *RTT* hétérogènes

Test	*BWU*	*LR* (%)	*F*	*CT* (s)		*RT* (s)	
				$1Mb/s$	$4Mb/s$	$1Mb/s$	$4Mb/s$
Plateforme locale							
$2 * WEBRC$	0.9 (±0.11)	0.96 (±1.2)	0.79 (±0.2)	45 (±38)	91 (±31)	8 (±6)	3 (±2)
$1 * TCP/$ $1 * WEBRC$	0.99 (±0.0)	1.62 (±4.25)	0.88 (±0.28)	96 (±92)	161 (±144)	1 (±0)	1 (±0)
$1 * WEBRC$ $/1 * TCP$	0.99 (±0.05)	0.67 (±1.5)	0.89 (±0.22)	129 (±118)	210 (±145)	1 (±0)	1 (±0)
Plateforme Strasbourg-Louvain							
$2 * WEBRC$	0.86 (±0.08)	0.59 (±0.91)	0.84 (±0.31)	33 (±18)	129 (±44)	1 (±2)	1 (±0)
$1 * TCP/$ $1 * WEBRC$	0.99 (±0.02)	0.8 (±1.91)	0.85 (±0.34)	60 (±41)	91 (±88)	1 (±0)	1 (±0)
$1 * WEBRC$ $/1 * TCP$	0.98 (±0.05)	0.41 (±0.93)	0.77 (±0.25)	28 (±33)	121 (±96)	1 (±0)	1 (±1)

(b) Résultats récapitulatifs

FIGURE 4.11 – Résultats de *WEBRC* avec des *RTT* hétérogènes

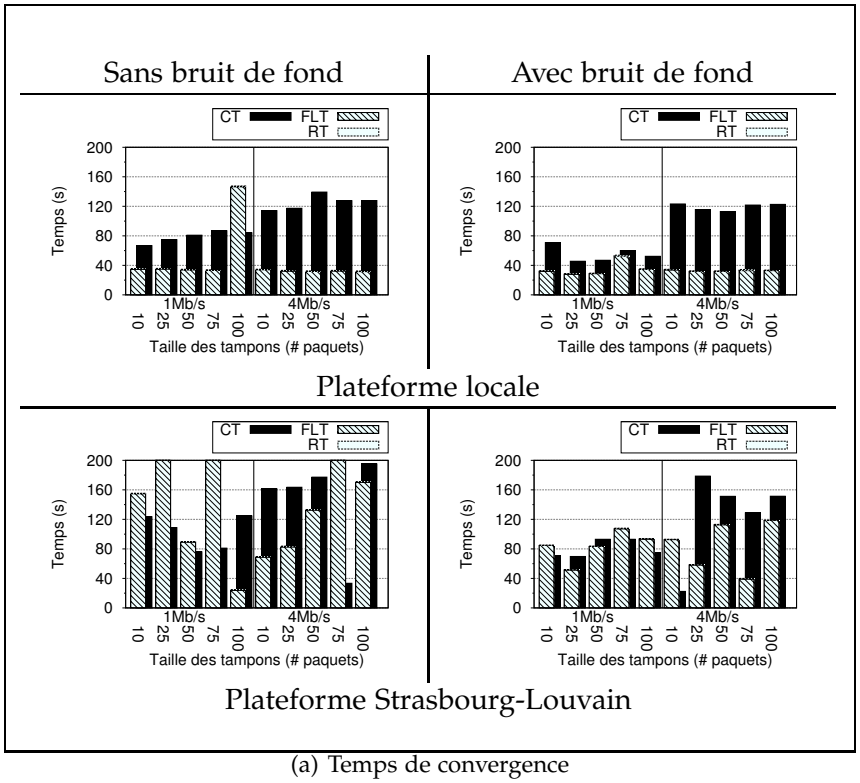
ne déroge pas à la règle et est également inéquitable. De plus, cette inégalité est totale ($F \simeq 0.62$) quand *WEBRC* est en concurrence avec d'autres flux *WEBRC* dans le cas d'un *RTT* très court, tel que celui de la plateforme locale à 4 Mb/s (cf. figure 4.11). En effet comme la différence absolue des *RTT* est toujours de 50 ms, plus le *RTT* de base est court, plus la différence relative est importante.
Cependant, *WEBRC* est toujours globalement sensible aux configurations réseau utilisées. Ainsi en dehors du fait qu'il est normal que l'équité soit faible pour ce type de test, cette variation de l'équité *WEBRC* est notable et confirme la remise en question de la robustesse de ce protocole.

4.6.7 TEMPS DE CONVERGENCE

Les résultats (cf. figure 4.12) pour ce scénario (cf. § 4.3.6) montrent que *WEBRC* converge lentement (sur la plateforme locale entre 40 et 80 secondes pour converger vers 1 Mb/s et $\simeq$ 120 secondes pour 4 Mb/s) et, qui plus est, de façon très variable. Cela même en l'absence de bruit de fond pour perturber la fonction de *Slow Start* de *WEBRC*. Ces lenteurs au démarrage ont un impact important sur l'utilisation de la bande passante ($0.72 \geq BWU \geq 0.83$), car ces temps de convergence correspondent à un temps compris entre 8 et 16% du temps d'expérimentation qui est, rappelons le, de 10 minutes.
Les résultats obtenus sur la plateforme Strasbourg-Louvain sont encore plus variables. En effet, il est normal que *WEBRC* soit sensible aux variations importantes de charge des routeurs du *mbone* s'il est déjà sensible en local aux variations des courts flux *TCP*.
De plus, les tests où le temps avant d'obtenir la première perte, ou *First Loss Time (FLT)*, est largement supérieur au temps de convergence, permettent de mettre en évidence le comportement proactif de *WEBRC* qui dans certains cas ne va jamais provoquer de pertes pour toute la durée du test. Ce comportement proactif présente tout de même l'avantage qu'à la fin de la phase de démarrage, *WEBRC* ne subit quasiment aucune perte.

4.6.8 LIENS FAIBLES MULTIPLES

Les résultats (cf. figure 4.13) pour ce scénario (cf. § 4.3.7) montrent que *WEBRC* est équitable (F en moyenne entre 0.91 et 0.93 pour la plateforme locale) pour partager la bande passante entre plusieurs flux traversant différents liens faibles et ce même avec du bruit de fond. Cependant, cette équité est toujours variable suivant les conditions réseau utilisées.
De plus, on peut observer que la bande passante est sous-utilisée

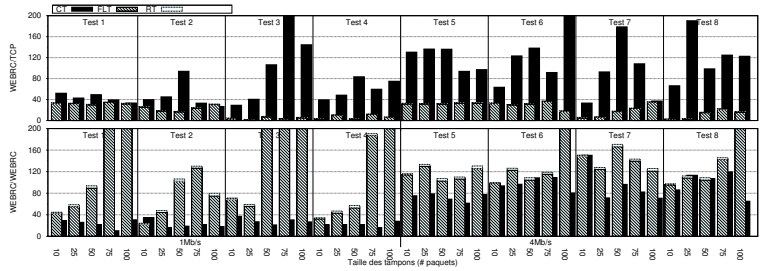


(a) Temps de convergence

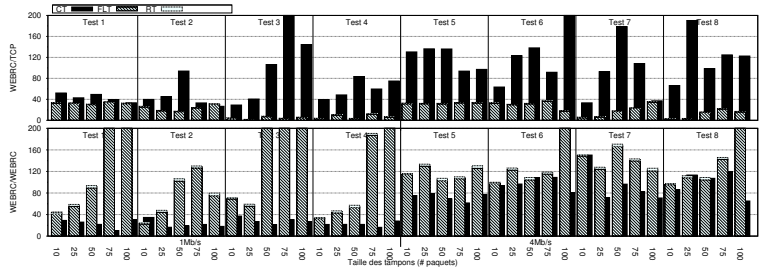
Test	*BWU*	*LR* (%)	*F*	*CT* (s)		*RT* (s)	
				$1Mb/s$	$4Mb/s$	$1Mb/s$	$4Mb/s$
Plateforme locale							
SANS bruit de fond	0.83 (±0.07)	0.19 (±0.26)	*N.A.*	79 (±51)	125 (±33)	1 (±0)	1 (±0)
AVEC bruit de fond	0.82 (±0.11)	0.48 (±1.11)	*N.A.*	55 (±38)	119 (±28)	1 (±1)	1 (±0)
Plateforme Strasbourg-Louvain							
SANS bruit de fond	0.72 (±0.66)	0.19 (±0.24)	*N.A.*	103 (±27)	146 (±113)	1 (±1)	1 (±0)
AVEC bruit de fond	0.75 (±0.69)	0.48 (±0.55)	*N.A.*	80 (±13)	127 (±104)	1 (±0)	1 (±0)

(b) Résultats récapitulatifs

FIGURE 4.12 – Résultats du temps de convergence pour *WEBRC*



(a) Plateforme locale



(b) Plateforme Strasbourg-Louvain

Paramètres de convergence de *WEBRC*

Test	*BWU*	*LR* (%)	*F*	*CT* (s)		*RT* (s)	
				$1Mb/s$	$4Mb/s$	$1Mb/s$	$4Mb/s$
Plateforme locale							
$2 * WEBRC$	0.86 ($\pm$0.14)	1.01 ($\pm$1.33)	0.92 ($\pm$0.27)	24 ($\pm$15)	91 ($\pm$37)	3 ($\pm$5)	3 ($\pm$5)
$1 * WEBRC$ /$1 * TCP$	0.95 ($\pm$0.08)	1.18 ($\pm$4.5)	0.93 ($\pm$0.19)	66 ($\pm$158)	114 ($\pm$87)	1 ($\pm$1)	1 ($\pm$1)
Plateforme Strasbourg-Louvain							
$2 * WEBRC$	0.83 ($\pm$0.11)	0.75 ($\pm$1.29)	0.95 ($\pm$0.24)	30 ($\pm$23)	123 ($\pm$73)	1 ($\pm$4)	1 ($\pm$4)
$1 * WEBRC$ /$1 * TCP$	0.87 ($\pm$0.65)	0.62 ($\pm$2.14)	0.88 ($\pm$0.38)	52 ($\pm$43)	138 ($\pm$113)	1 ($\pm$1)	1 ($\pm$1)

(c) Résultats récapitulatifs

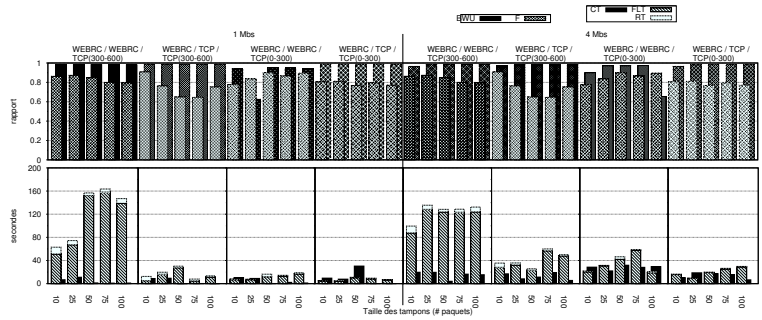
FIGURE 4.13 – *WEBRC* avec des liens faibles multiples

($0.83 \geq BWU \geq 0.86$) quand seuls des flux *WEBRC* sont présents. Cette sous-utilisation reste en grande partie due au temps de convergence qui est toujours important même si le débit à obtenir est au minimum réduit de moitié par rapport au test de convergence simple :

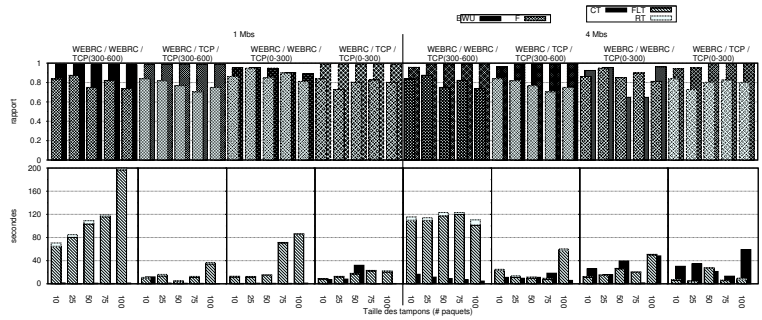
- À 1 Mb/s, la convergence est plus rapide mais prend tout de même 30 à 40 s pour les tests où la bande passante correspond à la bande passante laissée par le flux concurrent qui lui est uniquement limité par le second lien faible (cf. tests 1 et 5), ainsi que pour les tests où le flux concurrent laisse une partie de la bande passante, mais que le débit équitable est supérieur à ce vide (cf. tests 2 et 6).
- Toujours à 1 Mb/s, la convergence est encore de 30 à 40 s quand le flux concurrent n'est pas limité par le second lien faible, mais cela uniquement si le flux concurrent est un flux *WEBRC*. En effet si le flux concurrent est un flux *TCP*, alors le temps de convergence peut croître de façon importante. La raison de ce changement de performances en fonction du type de flux concurrent est dû au comportement proactif de *WEBRC* face aux congestions. Ainsi, quand *WEBRC* est le seul protocole en compétition, alors il est très rare d'observer une perte et donc de ralentir la convergence du protocole : la première perte (*FLT*) arrive toujours après et très souvent longtemps après que les flux aient convergé (*CT*) (cf. figure 4.13(a) et figure 4.13(b)). À l'inverse, il suffit d'un unique flux *TCP* provoquant des pertes pour ralentir *WEBRC*, qui n'est donc pas adapté à une utilisation telle que le cas d'un abonné *ADSL* qui utilise à la fois des logiciels de *Peer to Peer (P2P)* en *TCP* et regarde avec *WEBRC* des flux télévisuels dont la qualité progressive augmente en fonction du débit obtenu.
- Enfin en 4 Mb/s les mêmes comportement sont visibles, mis à part le fait que même sans être ralenti par une perte, *WEBRC* a une convergence lente généralement proche des 70 à 80 s.

4.6.9 VARIATION DU NOMBRE DE FLUX CONCURRENTS

Pour ce scénario (cf. § 4.3.8), nous ne nous intéressons aux résultats obtenus (cf. figure 4.14) qu'à partir du moment où le nombre de flux a changé dans le réseau : c'est-à-dire entre 300 et 600 secondes.
Les résultats obtenus confirment une fois de plus le comportement déjà observé de *WEBRC* : une sous-utilisation de la bande passante ($BWU \leq 0.89$) quand seuls des flux *WEBRC* sont présents et une équité relativement faible ($F \leq 0.87$) et variable selon les condi-



(a) Plateforme locale



(b) Plateforme Strasbourg-Louvain

Équité et paramètres de convergence

Test	*BWU*	*LR* (%)	*F*	*CT* (s)		*RT* (s)	
				$1Mb/s$	$4Mb/s$	$1Mb/s$	$4Mb/s$
Plateforme locale							
*2*WEBRC/ TCP(300-600)*	0.98 (±0.01)	1.84 (±3.02)	0.84 (±0.17)	4 (±28)	15 (±30)	7 (±7)	7 (±7)
*2*WEBRC/ TCP(0-300)*	0.89 (±0.84)	0.79 (±1.23)	0.86 (±0.28)	5 (±22)	28 (±60)	2 (±4)	2 (±4)
WEBRC/TCP/ TCP(300-600)	0.98 (±0.01)	1.92 (±7.87)	0.74 (±0.24)	4 (±18)	12 (±12)	4 (±5)	4 (±5)
WEBRC/TCP/ TCP(0-300)	0.98 (±0.02)	1.09 (±3.33)	0.79 (±0.22)	10 (±79)	14 (±16)	1 (±2)	1 (±2)
Plateforme Strasbourg-Louvain							
*2*WEBRC/ TCP(300-600)*	0.98 (±0.03)	1.39 (±2.25)	0.8 (±0.28)	1 (±3)	10 (±14)	6 (±12)	6 (±12)
*2*WEBRC/ TCP(0-300)*	0.88 (±0.87)	0.58 (±0.99)	0.87 (±0.34)	1 (±1)	26 (±43)	1 (±2)	1 (±2)
WEBRC/TCP/ TCP(300-600)	0.99 (±0.02)	1.02 (±3.14)	0.78 (±0.25)	3 (±27)	11 (±24)	2 (±4)	2 (±4)
WEBRC/TCP/ TCP(0-300)	0.98 (±0.08)	0.71 (±1.38)	0.8 (±0.29)	10 (±85)	31 (±145)	1 (±1)	1 (±1)

(c) Résultats récapitulatifs

FIGURE 4.14 – *WEBRC* : variation du nombre de flux concurrents

tions du réseau.
Ainsi l'étude du temps de convergence vers le nouveau débit équitable n'a pas vraiment de sens du fait que dans la plupart des cas les flux *WEBRC* n'atteignent pas le débit équitable. Cela influe sur la mesure de *CT* qui est relativement court. Ainsi, les flux *WEBRC*, faute de converger vers le débit équitable, ne subissent pas de longue perturbation de débit.

4.7 CONCLUSION

Pour évaluer et comparer les mécanismes de contrôle de congestion *unicast*, des discussions au sein de l'*IETF* ont permis de définir des métriques au sein du *RFC* 5166 [Floyd 08]. Ainsi, nous avons considéré et défini des méthodes pour évaluer de manière rigoureuse des protocoles de contrôle de congestion *multicast* en mettant en place :

- Des critères permettant d'évaluer les différentes propriétés d'un protocole de contrôle de congestion.
- Des scenarii simples et réalistes pour mettre en avant le comportement des protocoles de contrôle de congestion face à différents types d'environnement.
- Des plateformes isolées ou au contraire influencées par des flux extérieurs afin de cibler, soit le comportement propre du protocole testé, soit sa capacité à être robuste face à des flux réels passant par des ressources communes du réseau.

Les conditions réelles des réseaux en inter-domaine nous ont permis de mettre en évidence l'importante durée que pouvait prendre le temps d'abonnement. Ainsi, nous avons mis en exergue deux problèmes pour les mécanismes de contrôle de congestion qui sont le problème du démarrage et le problème de l'estimation du *RTT* en s'appuyant sur le temps d'adhésion.
Parmi les protocoles proposés précédemment, *WEBRC* est celui qui est le plus efficace pour partager équitablement la bande passante. Cependant, les différents tests montrent que *WEBRC* souffre :

- D'une mauvaise estimation du *RTT*, due à l'utilisation du temps d'abonnement qui génère une sous-évaluation du débit disponible. En effet, l'utilisation du temps d'adhésion n'est pas très représentative du temps de propagation alors qu'elle est censée l'approximer. Une solution est donc de remplacer le temps d'adhésion dans l'équation 3.3 par une variable plus proche du temps de propagation.
- D'un problème de robustesse (même pour les expérimentations où le temps d'adhésion est représentatif du *RTT*) car l'équité obtenue est très variable suivant les caractéristiques des réseaux utilisés. Ce problème ressemble entre autre aux problèmes des protocoles utilisant une équation du type de

TFRC [Rhee 07] pour lesquels un partage inéquitable peut s'amplifier notamment à cause de la manière d'évaluer les pertes. Ce problème d'équité est accentué avec la variation et l'augmentation du nombre de flux concurrents.

WEBRC est donc un protocole posant les bases d'un protocole équitable avec *TCP*, mais qui nécessite encore d'être amélioré afin que cette équité puisse perdurer même dans des environnements éprouvants.

Ces différentes difficultés nous mènent à proposer une solution couplant une source de type *WEBRC*, avec un récepteur évaluant sa bande passante de façon plus robuste en réaction aux changements d'état du réseau.

5

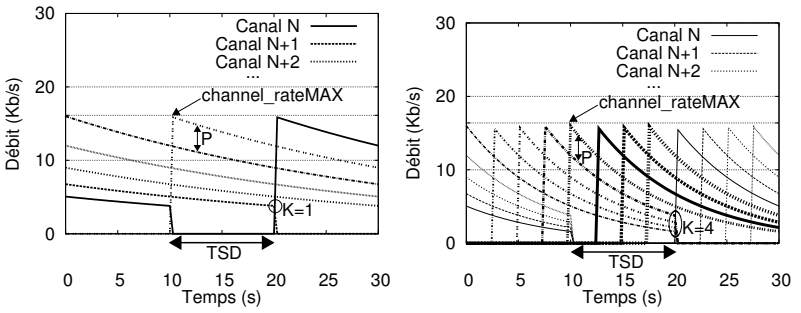
ALGORITHMES DE BASE

Comme pour *TCP*, la création d'un protocole de contrôle de congestion robuste et équitable pour le *multicast* a suivi une progression incrémentale. En effet, cette progression se retrouve dans l'état de l'art qui décrit l'évolution des améliorations réalisées de *Receiver-driven Layered Multicast (RLM)* jusqu'à *WEBRC*. Ainsi, c'est dans cette lignée que s'inscrivent nos travaux. Dans ce chapitre, nous présentons deux protocoles faisant suite à *WEBRC*. Le but principal de nos propositions est d'améliorer la robustesse du contrôle de congestion notamment en terme d'équité.
Nous commençons par présenter une amélioration de la source de *WEBRC* permettant d'accélérer la réaction face aux congestions tout en prenant en compte les spécificités du temps d'abonnement. Puis, nous présentons deux algorithmes côté récepteur, dont le premier se concentre sur l'étude de la gigue, alors que le second se penche sur l'utilisation d'une fenêtre de congestion.

5.1 ALGORITHME DE LA SOURCE

Le mécanisme de la source du protocole de contrôle de congestion *multicast* constitue le point ayant le plus évolué parmi les protocoles cités dans l'état de l'art.
Nous proposons ici d'utiliser la source de *WEBRC* afin d'améliorer la réactivité du protocole face aux congestions tout en prenant en compte le problème du temps d'adhésion. En effet, nous avons vu qu'un protocole comme *FLID-DL* peut subir un problème au démarrage. Ce problème du démarrage est lié à l'utilisation des canaux dynamiques et apparaît quand le temps d'adhésion est supérieur au *TSD* du protocole utilisé. Pour déplacer ou supprimer cette limite, nous pouvons recourir à plusieurs solutions :

- *Augmenter la valeur du TSD :*
  Cette solution permet simplement de repousser le seuil pour lequel le problème apparaît. Cependant, il ne faut pas oublier que la valeur du *TSD* représente aussi le temps de réaction du protocole à une congestion. De ce fait augmenter la valeur du *TSD* signifie ralentir la réaction du protocole face aux congestions.



(a) Source WEBRC

(b) Source avec réduction accélérée des débits

FIGURE 5.1 – Accélération de la réduction dynamique du débit des groupes

- *Effectuer des abonnements simultanés à plusieurs canaux :*
  En effet, le premier canal dynamique se termine au bout de $1 * TSD$ secondes, le second après $2 * TSD$ secondes, etc. S'abonner à plusieurs canaux simultanément permet donc d'augmenter la durée de diffusion des canaux. Cette solution intéressante ne nécessite pas de modifier le comportement de la source et doit donc être prise en compte lors de la conception du récepteur.

Au final, une source comme *WEBRC* avec un *TSD* de 10 secondes permet de supprimer ce problème de démarrage. Cependant, le débit des groupes de la source de *WEBRC* diminue trop lentement pour réagir convenablement à une congestion. En effet, cette diminution est de $1 - P = 25\%$ toutes les $TSD = 10$ secondes (voir figure 5.1(a)), avec $P = 0.75$ le facteur multiplicatif de diminution des canaux.
C'est pourquoi, nous proposons d'utiliser une pente de diminution des canaux de la source plus importante tout en gardant une granularité de débit inter-canal égale à celle de *WEBRC*. De plus, nous conservons *TSD* à 10 secondes pour éviter les problèmes liés au temps d'adhésion. Cela permet de conserver le même premier groupe dynamique pendant *TSD* secondes. Par conséquent, nous proposons de passer le facteur multiplicatif de diminution P de *WEBRC* à P^K. Ainsi, la granularité inter-canal est toujours de P, mais ces derniers diminuent de P^K en un *TSD* (voir figure 5.1(b)). De ce fait les canaux réduisent leur débit K fois plus rapidement et au bout de *TSD* secondes il y a non pas 1 mais K canaux qui deviennent muets.
Ainsi, la diminution du débit d'un canal est proportionelle au

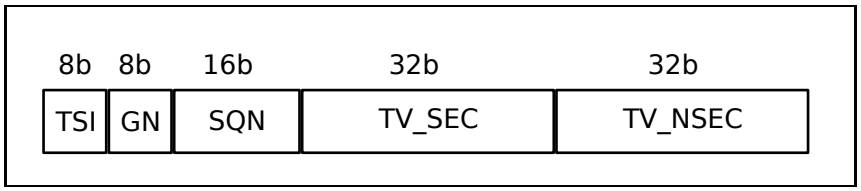


FIGURE 5.2 – En-tête pour la source améliorée

temps t qui correspond au temps écoulé depuis que ce canal a démarré son cycle avec le débit le plus élevé $channel\_rate_{MAX}$:

$$channel\_rate(t) \quad := \quad channel\_rate_{MAX} * \beta(t)$$

Où la réduction du débit est représentée par $\beta(t)$:

$$\beta(t) \quad := \quad P^{\left(\frac{t*K}{TSD}\right)} \tag{5.1}$$

Avec :

$P := 0.75$	Le facteur principal de réduction.
$K := 4$	Le facteur d'accélération de la réduction.
$TSD := 10$	La durée d'une subdivision du temps (s).

Ce facteur K permet donc de définir une accélération de la réduction du débit des canaux, afin d'améliorer la réactivité du protocole face à une congestion. Cependant, le nombre de messages de signalisation d'abonnement et de désabonnement *multicast* est alors K fois plus important que pour *WEBRC*. C'est pourquoi, nous proposons de limiter K à 4, ceci afin de préserver une moyenne acceptable du nombre de messages de signalisation correspondant à $\frac{2*K}{TSD} \simeq 0.8$ message de contrôle par seconde pour un récepteur.
Enfin, les protocoles proposés par la suite nécessitent que les récepteurs calculent le temps de propagation ou sa variation. Il faut donc étendre l'entête de *WEBRC* en y ajoutant un champ contenant une estampille de la date d'émission du paquet. Si cela suffit pour calculer la variation du temps de propagation, cela n'est pas suffisant pour le calcul du temps de propagation proprement dit. Pour ce dernier il faut également veiller à ce que les horloges de la source et des récepteurs soient synchronisées par *NTP* ou par *Global Positioning System (GPS)*.
Ainsi, l'en-tête utilisé (cf. figure 5.2) comprend un identifiant de l'intervalle de temps (TSI), un identifiant du groupe (GN), un numéro de séquence relatif au groupe (SQN) et l'estampille divisée en 2 parties (TV_SEC représentant les secondes et TV_NSEC pour les nanosecondes).

5.2 ALGORITHME RÉCEPTEUR BASÉ SUR LA GIGUE

La première proposition est de créer un *Récepteur Réactif (RR)* aux changements d'état du réseau qui sont observables par la détection de pertes et la variation du temps de propagation. Ce type de mécanisme s'inspire de celui de *TCP-vegas* car il repose sur l'observation de l'évolution de la gigue pour déterminer le niveau de saturation du réseau.

5.2.1 CALCUL DE LA GIGUE ET DÉTECTION DES PERTES

RR doit pouvoir atteindre un débit optimal sans utiliser de prédiction du débit disponible et en limitant les pertes. Le calcul de la variation du temps de propagation notée Δ utilise les estampilles des paquets envoyés par la source notée *estampille*. Ensuite, le récepteur doit relever, pour chaque paquet reçu, la date d'arrivée du paquet noté *reception*. Ainsi entre deux paquets successifs l'équation suivante permet de calculer Δ :

$$\delta_{source} = estampille_{paquet\_n+1} - estampille_{paquet\_n}$$
$$\delta_{recepteur} = reception_{paquet\_n+1} - reception_{paquet\_n}$$
$$\Delta = \delta_{recepteur} - \delta_{source}$$

Comme indiqué précédemment, le calcul de Δ ne nécessite donc pas de synchronisation entre les horloges de la source et des récepteurs.
Une perte est identifiée par un trou dans la séquence des paquets pour un groupe donné. De plus, dans le cas d'un déséquencement où $estampille_{paquet\_n+1} < estampille_{paquet\_n}$, *RR* réagit de la façon suivante :

- Si le déséquencement a lieu entre deux paquets de deux canaux différents, alors il faut considérer cet événement comme une perte.
- Sinon ce déséquencement a lieu pour deux paquets du même canal et a déjà été interprété comme une perte pour le paquet précédent et ne sera pas à nouveau comptabilisé comme tel.

5.2.2 ALGORITHME BASÉ SUR LA VARIATION DE LA GIGUE

Maintenant que nous savons identifier les pertes et calculer Δ, l'algorithme de *RR* est relativement simple :

- Si le récepteur n'est pas déjà en train d'adhérer à un nouveau canal et que pendant une certaine durée notée *attente* il ne connaît pas de pertes ni de variation trop importante de la gigue Δ, alors il augmente son débit en s'abonnant à un nouveau canal.

- Sinon le récepteur ne fait rien et laisse son débit diminuer tout seul.

Nous définissons que la variation de la gigue Δ est stable si elle est inférieure ou égale à un *seuil* correspondant au temps d'émission d'un paquet au débit actuellement reçu.
De plus, comme la source diminue lentement son débit, il faut éviter les adhésions successives trop rapides. Ainsi nous empêchons le récepteur de s'abonner à un nouveau canal pendant 500 ms après chaque adhésion effective [1] à un groupe *multicast*.
Enfin quand le *TSI* change, alors le récepteur se désabonne du canal qui vient de devenir muet.

5.2.3 AVANTAGES DE L'UTILISATION DE LA VARIATION DU TEMPS DE PROPAGATION

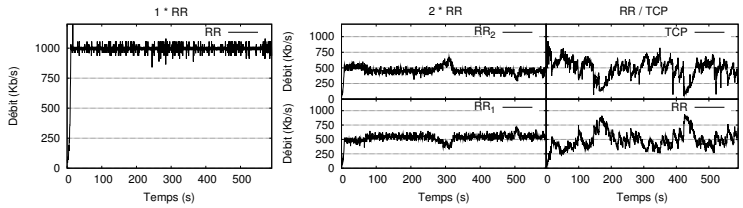
L'utilisation de la variation du temps de propagation est une mesure qui n'est pas affectée par les biais que nous avons constatés pour le temps d'adhésion :

- Les paquets sont estampillés par la source. De ce fait, la variation du temps de propagation est indépendante de la construction de l'arbre *multicast* ainsi que de la position du nœud de branchement. Il n'y a pas d'influence du nombre de récepteurs sur la variation, contrairement au temps d'adhésion.
- Le deuxième point est que la variation du temps de propagation est calculée à partir d'estampilles acheminées par des paquets de données. Ces paquets n'étant pas des paquets de signalisation, les routeurs ne les traiteront pas de manière singulière. Par conséquent, la variation du temps de propagation peut être mise en relation avec les variations du taux de remplissage des files d'attente de chaque routeur.
- En troisième point, la fréquence des paquets n'influe plus sur la justesse du résultat.
- En dernier point, la variation du temps de propagation est calculée à la réception de chaque paquet, contrairement au temps d'adhésion qui n'est relevé qu'à chaque nouvel abonnement à un canal.

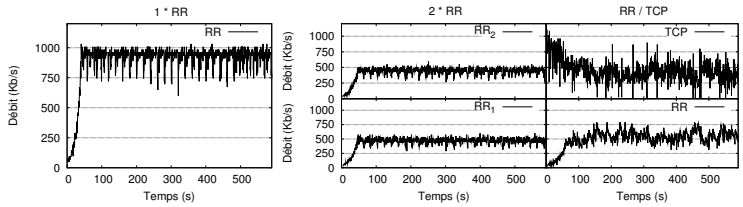
5.2.4 ÉVALUATION DE *RR* : COMPÉTITION ENTRE 2 LONG FLUX

Pour valider notre proposition, nous avons implémenté le mécanisme *RR* et testé son efficacité sur les plateformes locale et Strasbourg-Trondheim. Nous utilisons la source améliorée avec

1. Une adhésion est considérée comme effective à un groupe *multicast* dès que le premier paquet de donnée du groupe nouvellement abonné arrive au récepteur.



(a) Plateforme locale



(b) Plateforme Strasbourg-Trondheim

FIGURE 5.3 – Débit de sessions de *RR* en fonction du temps d'adhésion

l'ajout des estampilles nécessaire au calcul de Δ et $K := 1$ [2]. Le *seuil* correspond au temps inter-paquet moyen au débit reçu pendant les 500 dernières ms. La valeur du paramètre *attente* a été évaluée empiriquement à 148 ms. Les résultats de la figure 5.3 mettent en évidence plusieurs points :

- Pour chacun des tests, *RR* utilise toute la bande passante disponible.
- *RR* converge très rapidement vers le débit optimal. En effet, alors qu'en local *WEBRC* converge en près de 100 secondes, n'en faut que 10 pour *RR* (figure 5.3(a)). Sur le m6bone *RR* converge en 40 secondes (figure 5.3(b)), car il est ralenti par le temps d'adhésion.
- Un point négatif de *RR* est qu'il provoque des pertes plus importantes que *WEBRC*, car *RR* est un protocole réactif aux pertes contrairement à *WEBRC* qui essaye de les prédire. Mais ce taux de pertes reste tout de même inférieur à 1% pour tous les tests n'utilisant pas *TCP* et est inférieur à 5% pour les tests mêlant *RR* et *TCP*. Quand *RR* détecte une importante augmentation de la variation du temps de propagation, il n'augmente pas son débit car il sait qu'un abonnement provoquerait de nombreuses pertes. Malheureusement, la source réduit son débit très lentement et notre prédiction est trop tardive pour éviter les pertes. Cette consta-

2. Lors de la proposition de cet algorithme, nous n'avions pas encore proposé d'amélioration de la source pour augmenter la rapidité de réaction aux congestions. En effet, les motivations de cette amélioration ne sont apparues qu'à partir de l'analyse des résultats ci-après.

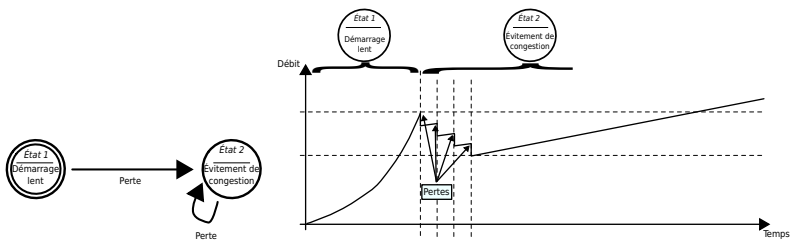
tation constitue la motivation première de notre proposition d'amélioration de la source grâce au facteur *K*, permettant d'accélérer la réduction des canaux tout en conservant le même *TSD*.

- Les résultats de la figure 5.3 montrent que *RR* est équitable envers lui-même et envers *TCP* dans le sens où aucune instance n'usurpe continuellement la bande passante de l'autre. Par contre, les résultats de *RR* en compétition avec *TCP* montrent une variation importante et un manque de stabilité de l'évaluation de la bande passante, alors que ce test se déroule sur la plateforme locale où aucun flux externe aux tests ne vient perturber les résultats.

Ainsi, les résultats de *RR* montrent que l'amélioration de la source peut être utile en cas d'abonnements trop rapides. De plus, le point le plus marquant est que le test avec *TCP* sur la plateforme locale montre une instabilité du protocole, ce qui motive à chercher des propositions pouvant apporter un effet de mémorisation de l'état général du réseau.

5.3 ALGORITHME UTILISANT UNE FENÊTRE DE CONGESTION

Suite aux résultats de *RR* et notamment son instabilité face à un flux *TCP*, nous avons essayé de réutiliser les mécanismes présents dans *TCP* afin d'améliorer le comportement du protocole notamment face aux flux *unicast*. Pour cela, nous nous sommes intéressés aux mécanismes utilisant des fenêtres de congestion.
La principale contribution est de calculer la taille de la fenêtre de congestion en combinant et en adaptant les deux mécanismes *CA* et *SS* (cf. figure 5.4) de *TCP-newReno* afin de créer un protocole de contrôle de congestion capable de converger rapidement vers le débit équitable tout en minimisant les variations de débit une fois ce débit atteint. Le protocole ainsi créé se nomme *M2C* et s'inspire donc de la fenêtre de congestion de *TCP*-newReno. Une première différence est que cette fenêtre est gérée par le récepteur, car c'est ce dernier qui décide s'il augmente ou diminue son débit en fonction de ses abonnements aux différents groupes *multicast*.
De plus, on peut se demander comment un mécanisme du type *AIMD* peut être équitable s'il ne peut pas calculer directement le *RTT* ? En effet, nous avons vu que *WEBRC* propose une estimation du *RTT* biaisée car étant basée sur le temps d'adhésion. Qui plus est, *RR* ne propose pas de solution à ce problème. Ainsi pour *M2C*, nous proposons de calculer une estimation du *RTT*, ou *Estimated Round Trip Time (ERTT)*, basée sur la durée aller des paquets entre la source et chaque récepteur (*OWD*).



(a) Diagramme états-transitions (b) Principe de l'estimation du débit équitable

FIGURE 5.4 – *Contrôle de Congestion Multicast* inspiré de l'analyse de TCP (*M2Cv1*)

5.3.1 PRINCIPES GÉNÉRIQUES DE *M2C*

Nous présentons ici la première version de *M2C*, ou *Multicast Congestion Control version 1 (M2Cv1)*, qui se base sur les mécanismes de *TCP* et utilise une source avec une réduction du débit accélérée d'un facteur $K := 4$.
Chaque récepteur peut s'abonner à plus ou moins de groupes jusqu'à ce que son débit reçu atteigne le débit estimé comme étant équitable, ou *Estimated Fair Rate (EFR)*. Si *EFR* croît très rapidement, alors un récepteur peut être amené à s'abonner à plusieurs groupes simultanément. La difficulté principale de cet algorithme est de déterminer la valeur d'*EFR* en considérant les problèmes énoncés précédemment. L'idée principale pour le calcul de l'*EFR* est dérivée de *TCP* [Jacobson 88][Stevens 97] :

- Un mécanisme d'évitement de congestion (*CA*) permettant à *M2Cv1* de partager équitablement la bande passante. Pour être équitable envers *TCP*, *CA* utilise un mécanisme de croissance linéaire et de réduction multiplicative (*AIMD*). Ainsi, la courbe d'augmentation des débits pendant la phase de *CA* suit une croissance linéaire. Cette augmentation est très douce ce qui permet de limiter les oscillations de débit.
- Un mécanisme de démarrage lent (*SS*) permettant à *M2Cv1* d'approcher le débit équitable plus rapidement. Il est conçu pour remplir le plus rapidement possible les files d'attente du routeur en amont du lien faible. Ainsi, la courbe d'augmentation des débits décrite par *SS* suit une croissance exponentielle.

5.3.2 ESTIMATION DU *RTT* (*ERTT*)

Nous proposons de calculer une estimation du *RTT*, ou *ERTT*, basée sur la mesure de la durée aller, ou *OWD*. Elle correspond à

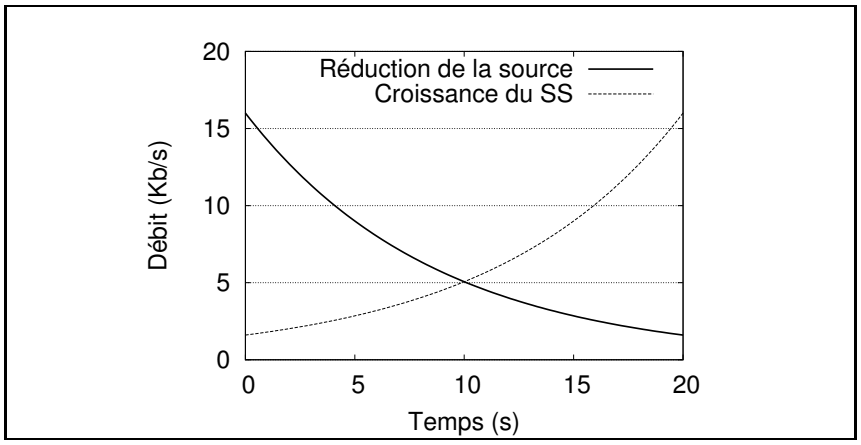


FIGURE 5.5 – Fonction de croissance du *SS* de *M2C*

la différence entre la date d'arrivée du paquet et son estampille. Pour que cette mesure soit correcte, il faut que la source et les récepteurs aient leurs horloges synchronisées, par exemple avec *NTP* ou par *GPS*.
Ainsi, cette estimation du *RTT* est calculée grâce à une moyenne pondérée du *OWD* pour chaque nouveau paquet reçu :

$$ERTT := ERTT * \beta(TIP) + OWD * (1 - \beta(TIP))$$

Avec β correspondant à la définition de l'égalité 5.1 et *Temps Inter-Paquets (TIP)* pour le temps séparant la réception des 2 derniers paquets.

5.3.3 L'ÉTAT DE DÉMARRAGE LENT OU *Slow Start* (*SS*)

La phase de démarrage lent (*état 1* de la figure 5.4(a)) sert à découvrir la bande passante disponible. Comme pour *TCP*, nous utilisons une croissance exponentielle afin d'atteindre rapidement le débit équitable. Ainsi pour chaque paquet reçu, l'estimation du débit équitable, ou *EFR* est mise à jour selon :

$$EFR \quad := \quad \frac{EFR}{\beta(TIP)}$$

Avec $\beta(TIP)$, le facteur de réduction de la source (cf. équation 5.1) en fonction du *TIP*. En effet, contrairement à *TCP*, β n'est pas constant et est recalculé pour chaque paquet reçu.
La croissance d'*EFR* en phase de *SS* représente une courbe (cf. figure 5.5) symétrique selon l'axe des ordonnées à celle de la réduction de la source et limite ainsi la période de congestion suivant chaque phase de *SS*.

5.3.4 L'ÉTAT D'ÉVITEMENT DE CONGESTION OU *Congestion Avoidance* (*CA*)

La phase d'évitement de congestion (*CA*) (*état 2* de la figure 5.4(a)) sert à découvrir progressivement la bande passante disponible. *CA* se base sur l'analyse de *TCP* [Mathis 97][Padhye 98] ainsi que sur la relation permettant à un mécanisme à croissance additive et diminution multiplicative (*AIMD*) d'être équitable avec *TCP* tel que décrit dans [Yang 00]. Pour cela, nous utilisons une croissance linéaire avec laquelle l'*EFR* est actualisée à chaque réception de paquet selon :

$$\begin{aligned} inc\_window &:= \alpha * \frac{packet\_size^2}{window\_size} \\ EFR &:= EFR + \frac{inc\_window}{ERTT} \\ \text{Avec :} & \\ window\_size &:= EFR * ERTT \end{aligned}$$

Le facteur d'augmentation additif α est dérivé de [Yang 00] :

$$\alpha := 3.0 * \frac{1.0 - \beta}{1.0 + \beta}$$

Les formules utilisées pour la phase de *CA* montrent que l'équité avec *TCP* dépend principalement de la précision du calcul de *ERTT*. De plus, l'augmentation en douceur de *EFR* pendant la période de *CA* est assurée par le facteur α qui est dérivé du facteur de diminution des débits de la source (β).

5.3.5 LA TRANSITION LORS D'UNE CONGESTION OU *Congestion Event* (*CE*)

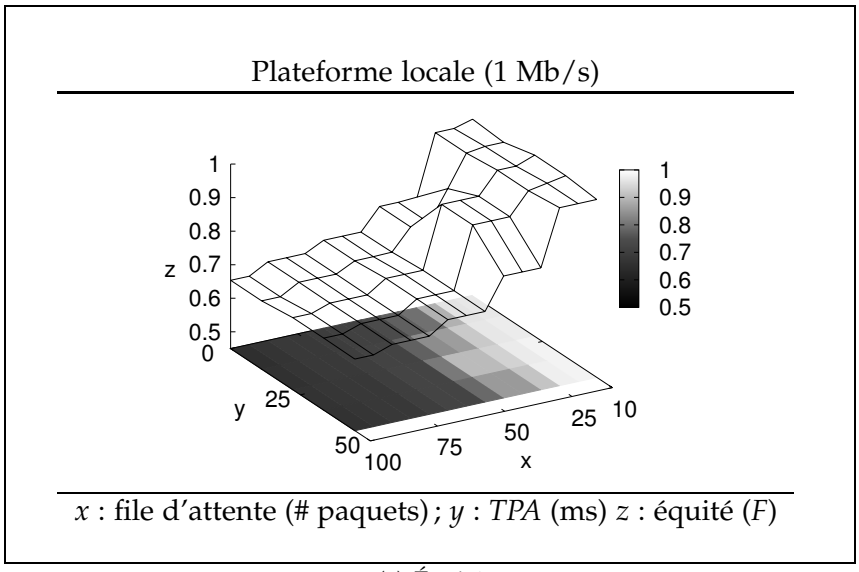
Les différentes congestions ne sont détectées que lors d'une perte de paquet. La réaction lors d'un *CE* est une diminution multiplicative proportionnelle à la réduction continue du débit des canaux de la source :

$$EFR := EFR * \beta$$

Une fois la réduction opérée, le protocole repasse en mode de *CA*.

5.4 ÉVALUATION DE *M2Cv1*

Maintenant que *M2Cv1* est défini, nous pouvons étudier son comportement en évaluant ce dernier face à 1 et *N* flux *TCP* concurrents.

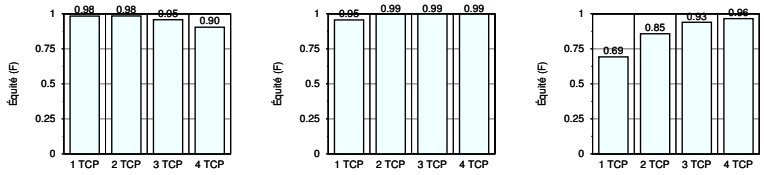


(a) Équité

Test	*BWU*	*LR* (%)	*F*	*CT* (s)		*RT* (s)	
				$1Mb/s$	$4Mb/s$	$1Mb/s$	$4Mb/s$
Plateforme locale (1Mb/s)							
$1 * M2Cv1$ / $1 * TCP$	0.99 ($\pm$0.01)	1.06 ($\pm$3.16)	0.8 ($\pm$0.19)	13 ($\pm$43)	*N.A.*	1 ($\pm$2)	*N.A.*

(b) Résultats récapitulatifs

FIGURE 5.6 – Résultats pour 1 flux *M2Cv1* face à 1 flux *TCP*



(a) $TPA < 1$ ms, tampon 5 paquets (b) TPA 100 ms, tampon 20 paquets (c) TPA 100 ms, tampon 80 paquets

Équité de *M2Cv1*

Test	*BWU*	*LR* (%)	*F*	*CT* (s)		*RT* (s)	
				$1Mb/s$	$4Mb/s$	$1Mb/s$	$4Mb/s$
Plateforme locale							
$1 * M2Cv1$ $/1 * TCP$	0.98 (±0.03)	8.68 (±16.76)	0.88 (±0.19)	10 (±28)	*N.A.*	9 (±90)	*N.A.*
$1 * M2Cv1$ $/2 * TCP$	0.98 (±0.02)	9.68 (±17.83)	0.95 (±0.13)	4 (±8)	*N.A.*	17 (±69)	*N.A.*
$1 * M2Cv1$ $/3 * TCP$	0.98 (±0.01)	10.15 (±18.17)	0.97 (±0.04)	4 (±5)	*N.A.*	18 (±70)	*N.A.*
$1 * M2Cv1$ $/4 * TCP$	0.98 (±0.01)	10.48 (±17.47)	0.96 (±0.09)	3 (±2)	*N.A.*	29 (±117)	*N.A.*

(d) Résultats récapitulatifs

FIGURE 5.7 – Résultats d'1 flux *M2Cv1* face à N flux *TCP*

5.4.1 1 FLUX *M2Cv1* FACE À 1 FLUX *TCP*

Ce scénario a été uniquement évalué sur la plateforme locale et à 1 Mb/s. En effet, les résultats obtenus (cf. figure 5.6 et tableau 5.6(b)) montrent que *M2Cv1* n'est pas équitable pour la grande majorité des configurations ($F \simeq 0.8$) et il est donc inutile de tester plus avant la robustesse de ce protocole en multipliant le nombre de tests.
Cependant même si les résultats obtenus sont médiocres, ceux-ci sont tout de même corrects pour des tailles de tampon comprises entre 10 et 25 paquets. Nous allons donc réaliser les tests avec plusieurs flux *TCP* pour essayer de cerner pourquoi ces différences de comportement apparaissent en fonction de la taille de la file d'attente.

5.4.2 1 FLUX *M2Cv1* FACE À N FLUX *TCP*

Ce scénario de tests a été réalisé pour isoler les résultats et étudier plus finement 3 comportements de *M2Cv1* :

- Le comportement 1 apparaît quand les tampons des routeurs sont très petits, comme par exemple à la figure 5.7(a) avec un tampon de 10 paquets. Certes, *M2Cv1* semble équi-

table, mais le taux de perte d'environ 30% est beaucoup trop élevé. Par conséquent, dans ces conditions *M2Cv1* présente le même comportement que *WEBRC* qui peut se résumer à un dysfonctionnement montrant les limitations de ces méthodes quand les tampons sont sous-dimensionnés. Cependant comme pour *WEBRC*, on peut considérer ce scénario comme étant rare et fortement improbable au vu des configurations actuelles des routeurs qui par défaut utilisent bien plus d'espace mémoire pour les files d'attente.

- Le comportement 2 (voir figure 5.7(b)) apparaît avec des files d'attente de plus grande taille et un *OWD* inférieur à environ 350 ms. Dans ces conditions, *M2Cv1* se montre équitable envers les flux *TCP*, avec un taux de perte inférieur à 4%. Ce comportement montre que l'estimation de *M2Cv1* est plus précise que celle de *WEBRC* pour les configurations de réseaux les plus courantes.
- Le troisième comportement (voir figure 5.7(c)) apparaît quand l'*OWD* est supérieur à environ 350 ms. *M2Cv1* sous-utilise la bande passante, avec un taux de pertes toujours inférieur à 4%. En réalisant des tests de plusieurs heures dans les mêmes conditions, on peut observer que *M2Cv1* est finalement équitable avec *TCP*. Cela signifie que le problème vient du temps de convergence qui ici peut atteindre une heure pour un *RTT* d'une seconde.
  Ce problème de convergence est confirmé par le fait qu'en augmentant le nombre de flux *TCP* concurrents l'équité augmente (cf. tableau 5.7(d)) car le flux *M2Cv1* doit converger vers un débit équitable qui est de moins en moins important.

M2Cv1 est donc équitable pour les configurations de réseaux les plus courantes. Cependant, il souffre d'un problème de lenteur de convergence qui le rend difficilement utilisable pour les réseaux à long délai.

5.5 CONCLUSION

Dans ce chapitre, nous avons proposé 2 mécanismes comme principes de base pour améliorer les protocoles de contrôle de congestion pour le *multicast*.
Le premier est basé sur l'étude de la gigue pour décider s'il doit augmenter son débit. Les premiers résultats pour ce protocole ont montré sa rapidité de convergence vers le débit équitable, mais également une grande instabilité face à un flux *TCP* concurrent.
Ainsi, le second veut apporter une mémoire pour améliorer la stabilité de l'estimation du débit. Pour ce faire, nous proposons d'utiliser une fenêtre de congestion dont l'augmentation est gérée par une adaptation des mécanismes de *SS* et de *CA* de *TCP*-newReno.

Contrairement à *TCP*, cette fenêtre de congestion est gérée par les récepteurs car en *multicast* chaque récepteur est chargé de décider quel débit obtenir. Les résultats de cet algorithme montrent que le protocole est équitable, mais que la convergence vers cette équité est très longue à obtenir si le *RTT* estimé est important.

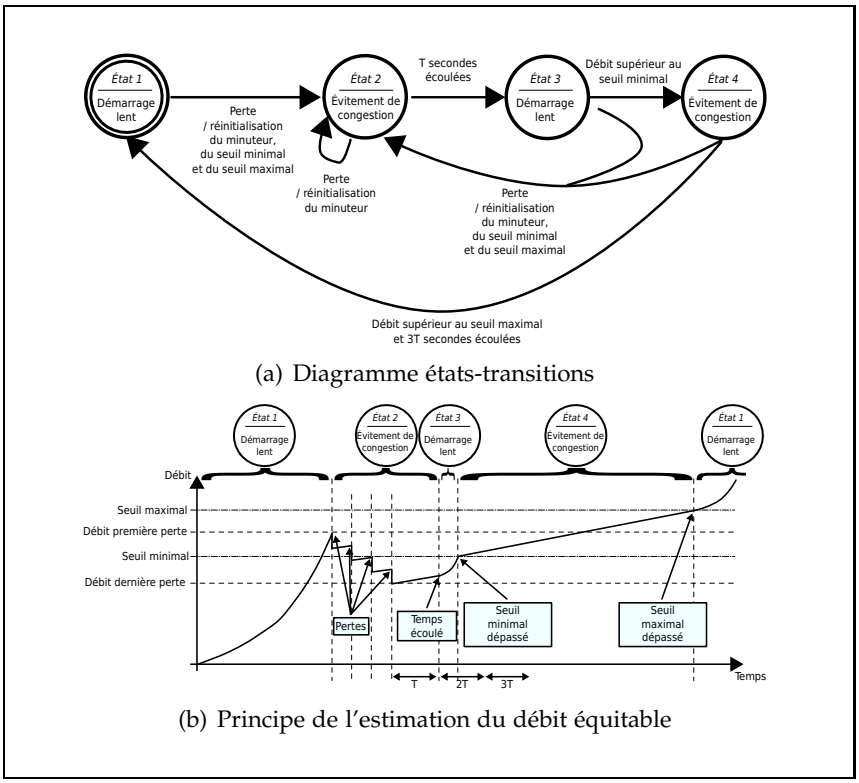
6 AMÉLIORATIONS INCRÉMENTALES

Dans le chapitre précédent nous avons fait deux propositions pour améliorer l'utilisation de la bande passante et l'équité des protocoles de congestion *multicast*. Parmi ces propositions *M2Cv1* dispose d'un mécanisme de fenêtre de congestion côté récepteur qui le rend équitable à très long terme, mais souffre d'un problème de rapidité de convergence vers le débit équitable. Dans ce chapitre, nous proposons donc d'améliorer de façon incrémentale le protocole *M2C* [1], afin d'obtenir un protocole équitable, robuste avec un temps de convergence réduit.

6.1 RÉSURGENCE DU PROBLÈME DES RÉSEAUX À HAUTS DÉBITS ET LONGS DÉLAIS

Si *M2Cv1* est fonctionnel tant que les tampons des routeurs ne sont pas sous-dimensionnés, son temps de convergence très lent le rend inutilisable pour les réseaux où le *RTT* est supérieur à 350 ms. Or, ce problème est semblable à celui de *TCP* pour les réseaux à hauts débits et longs délais, car ce dernier converge également trop lentement pour pouvoir utiliser correctement la bande passante disponible.
En effet, la lenteur de convergence de l'algorithme *M2Cv1* est due à la phase d'évitement de congestion, utilisée pour atteindre en douceur le débit idéal. Ainsi en analysant les traces obtenues avec *M2Cv1*, nous avons constaté qu'il est fréquent qu'une perte interrompe de façon prématurée la phase de démarrage lent, provoquant alors le passage en phase d'évitement de congestion alors que le débit équitable est loin du débit actuellement reçu. Afin de résoudre ce problème, nous proposons le protocole *Multicast Congestion Control version 2 (M2Cv2)* qui améliore *M2Cv1* en ajoutant un mécanisme pour réduire le temps de convergence qui s'inspire de protocoles tels que *BIC* [Xu 04] et *CUBIC*[Ha 08]. *M2Cv2* propose de reprendre le mode de démarrage lent (*SS*) quand la probabilité d'être proche du débit équitable est trop faible. Cette

1. Une description synthétique des différentes améliorations est détaillée dans la version finale de *M2C* jointe en annexe A.



(a) Diagramme états-transitions

(b) Principe de l'estimation du débit équitable

FIGURE 6.1 – Algorithme d'amélioration du temps de convergence de *M2Cv2*

probabilité est évaluée grâce à l'analyse des pertes que peuvent provoquer les flux *TCP* à un débit équitable.

6.1.1 ANALYSE DES PERTES *TCP*

Le but de *M2Cv2* est de déterminer la distance au débit équitable pour reprendre le cas échéant le démarrage lent et ainsi améliorer le temps de convergence. Or, l'analyse des pertes obtenues quand un flux *TCP* est proche du débit équitable montre que le flux subit des pertes :

- Tant que la rafale de pertes n'est pas encore terminée. La rafale est estimée comme terminée quand le délai depuis la dernière perte est supérieur au temps (T) qui correspond à un *RTT*, ce qui se traduit pour *M2Cv2* par $T := ERTT$.
- Quand le débit utilisé est proche de celui auquel la dernière perte a été détectée. Nous définissons que le débit R est proche du débit D de la perte précédente si $seuil_{min} < R < seuil_{max}$ avec $seuil_{min} := D * \beta^2$ et $seuil_{max} := \frac{D}{\beta^2}$.

6.1.2 AMÉLIORATION DU TEMPS DE CONVERGENCE

Avec ces nouvelles définitions nous devons ajouter 2 nouveaux états pour améliorer notre algorithme d'évitement de congestion comme décrit à la figure 6.1(a) :

- La phase de *démarrage lent* (*SS*) à l'*état 3*. Quand la durée depuis la dernière perte est trop longue, c'est-à-dire depuis plus de T secondes. Nous utilisons alors une croissance exponentielle pour converger rapidement vers le débit de la dernière perte.
- La phase d'*évitement de congestion* (*CA*) à l'*état 4*. Quand le débit est proche du débit reçu au moment de la dernière perte, nous ralentissons la croissance de notre estimation. Cela signifie que pour $seuil_{min} < EFR < seuil_{max}$, EFR suit alors l'algorithme de *CA* et augmente linéairement pour découvrir progressivement la bande passante disponible.

De plus, 2 transitions supplémentaires permettent de revenir aux 2 premiers états :

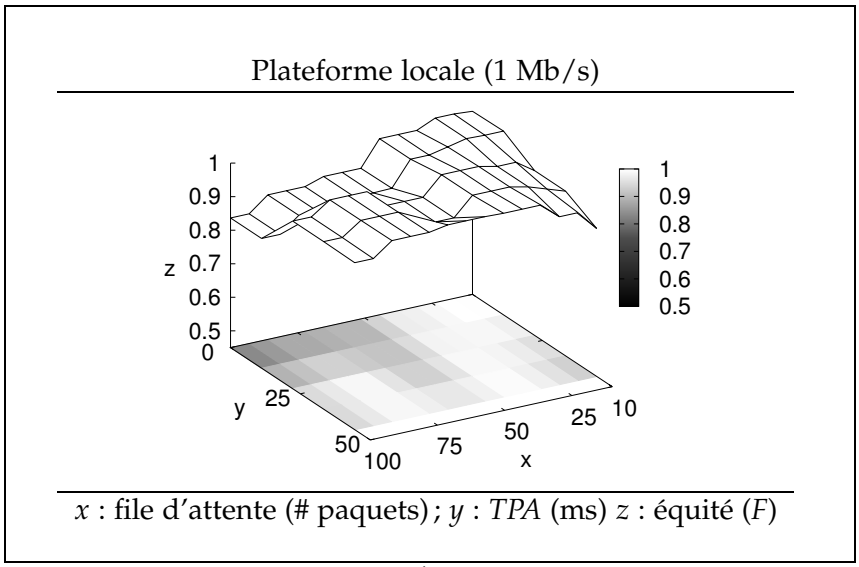
- Si $EFR > seuil_{max}$ et que le temps depuis la dernière perte est supérieur à $\gamma * T$ [2], alors le débit reçu lors de la dernière perte est largement inférieur au débit équitable, ce qui permet de reprendre le démarrage lent de l'*état 1* et ce jusqu'à la prochaine perte.
- Une *perte* provoque toujours un retour à l'*état 2* et réinitialise le minuteur T. De plus, si l'état précédent n'était pas l'*état 2*, nous recalculons les valeurs de $seuil_{min}$ et de $seuil_{max}$. De cette manière, en cas de rafale ces seuils sont fixés à la valeur correspondant au débit lors de la première perte de la rafale.

Ainsi quand le débit est proche du débit équitable, cet algorithme croît linéairement (*états 2 et 4*), ce qui permet d'effectuer un partage équitable de la bande passante. D'autre part quand le débit est éloigné du débit équitable, cet algorithme croît exponentiellement (*états 1 et 3*) pour converger rapidement vers le débit équitable.

6.2 ÉVALUATION DE *M2Cv2*

Pour vérifier l'efficacité des améliorations proposées par *M2Cv2* et comparer son comportement avec *M2Cv1*, nous avons rejoué les scenarii face à 1 et N flux *TCP*.

2. Dans la suite, nous prenons $\gamma = 3$. Comme T est calculé en fonction de OWD_{moyen} qui peut être 2 fois plus petit que le *RTT*, il faut donc au moins que $\gamma \geq 2$ pour équilibrer cette approximation et $\gamma = 3$ permet de ne pas être trop agressif pour la reprise du mode de démarrage lent (*SS*).



(a) Équité

Test	BWU	LR (%)	F	CT (s)		RT (s)	
				$1Mb/s$	$4Mb/s$	$1Mb/s$	$4Mb/s$
Plateforme locale							
$1 * M2Cv2$ / $1 * TCP$	0.99 ($\pm$0.01)	1.94 ($\pm$3.94)	0.95 ($\pm$0.1)	19 ($\pm$37)	$N.A.$	1 ($\pm$2)	$N.A.$

(b) Résultats récapitulatifs

FIGURE 6.2 – Résultats pour 1 flux *M2Cv2* face à 1 flux *TCP*

6.2.1 1 FLUX *M2Cv2* FACE À 1 FLUX *TCP*

Les résultats obtenus (cf. figure 6.2) pour ce scénario (cf. § 4.3.1) montrent que *M2Cv2* est bien plus équitable ($F \simeq 0.95$) et cela de façon robuste pour les différentes configurations du réseau :

- Le comportement 1 apparaît toujours pour des très petites files d'attente. En effet, le comportement de *M2Cv2* est plus agressif que celui de *M2Cv1* uniquement si le temps entre 2 pertes consécutives est trop important. Or comme le taux de pertes avoisine les 30%, il est normal de constater que *M2Cv2* n'utilise pas le retour au *SS* fréquemment. Par conséquent, *M2Cv2* n'est pas plus agressif dans ces conditions et conserve donc son comportement équitable.
- Le comportement 2 a lui fortement progressé. En effet, quasiment tout l'espace des paramètres d'expérimentation correspond à ce comportement où les flux *M2Cv2* sont équitables avec un taux de pertes aux environs de 2%. Ainsi les modifications ont à la fois :
  - permis au protocole de converger plus rapidement vers le débit équitable, le rendant ainsi plus équitable pour les tests où le *RTT* est important ;
  - permis de conserver le comportement équitable du proto-

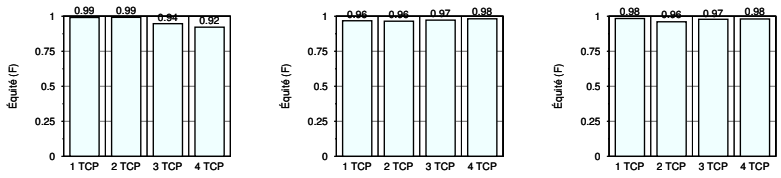
cole quand le *OWD* est inférieur à 350 ms.
Par conséquent, les phases de retour en *SS* sont utilisées à bon escient ce qui empêche les flux *M2Cv2* d'être trop agressif et donc de monopoliser la bande passante.

- Le comportement 3 apparaît encore dans quelques rares configurations où le *RTT* est important et est composé en majorité du temps de mise en file d'attente et en minorité du temps de propagation.
  Pour cette série de tests, il n'existe pas de flux traversant le réseau dans le sens inverse. Par conséquent, le *RTT* est constitué de 2 fois le temps de propagation ajouté (*TPA* aller et retour) et du temps de mise en file d'attente aller. Cependant *ERTT* se base sur le délai en sens unique (*OWD*), qui lui est constitué d'1 seul temps de propagation ajouté ainsi que du temps de mise en file d'attente aller. Or les conditions réseaux du comportement 3 correspondent à un *RTT* où le temps de propagation est négligeable, ce qui signifie que dans ces conditions l'*ERTT* est une estimation fidèle du *RTT*.
  Par conséquent le mécanisme de retour en *SS* n'est vraiment efficace que si l'*ERTT* sous estime le *RTT*. L'analyse des traces du comportement 3 montre que dans le cas contraire *M2Cv2* va passer un temps trop important dans l'état 4 où l'algorithme de *CA* augmente lentement l'*EFR*. Ainsi la condition de limite de débit maximal à obtenir avant de repartir en *SS* permet bien d'améliorer le temps de convergence quand l'iniquité est très importante, mais est une estimation trop grossière pour fonctionner avec des écarts de débit moins conséquents mais néanmoins non négligeables.

Ces résultats confirment donc que le problème principal de *M2Cv1* était bien lié au temps de convergence. Cependant, la présence même rare du comportement 3 laisse présager qu'il est possible de trouver un moyen d'affiner le mécanisme de retour en *SS* pour encore améliorer le temps de convergence du protocole.

6.2.2 1 FLUX *M2Cv2* FACE À *N* FLUX *TCP*

Les mêmes tests que pour *WEBRC* et *M2Cv1* sont réalisés pour *M2Cv2* afin de pouvoir comparer ces protocoles. Les résultats obtenus (cf. figure 6.3) confirment bien que *M2Cv2* résout le problème du temps de convergence pour des réseaux présentant des délais importants. Ainsi, *M2Cv2* reste équitable pour tous les réseaux dont les files d'attente ne sont pas sous-dimensionnées tout en améliorant sensiblement le temps de convergence qui est divisé par un facteur 3 à 5 pour la figure 6.3(b) et converge en moins de 2 minutes là où *M2Cv1* en prenait plus de 60.



(a) $TPA < 1$ ms, tampon 5 paquets (b) $TPA \simeq 100$ ms, tampon 20 paquets (c) $TPA \simeq 100$ ms, tampon 80 paquets

Équité de *M2Cv2*

Test	*BWU*	*LR* (%)	*F*	*CT* (s)		*RT* (s)	
				$1Mb/s$	$4Mb/s$	$1Mb/s$	$4Mb/s$
Plateforme locale							
$1 * M2Cv2$ $/1 * TCP$	0.98 (±0.03)	9.4 (±16.01)	0.98 (±0.03)	20 (±31)	*N.A.*	14 (±96)	*N.A.*
$1 * M2Cv2$ $/2 * TCP$	0.98 (±0.02)	9.88 (±17.9)	0.97 (±0.07)	10 (±29)	*N.A.*	36 (±330)	*N.A.*
$1 * M2Cv2$ $/3 * TCP$	0.98 (±0.02)	10.42 (±17.79)	0.97 (±0.07)	4 (±5)	*N.A.*	12 (±98)	*N.A.*
$1 * M2Cv2$ $/4 * TCP$	0.98 (±0.01)	10.86 (±17.97)	0.96 (±0.1)	4 (±9)	*N.A.*	25 (±140)	*N.A.*

(d) Résultats récapitulatifs

FIGURE 6.3 – Résultats d'1 flux *M2Cv2* face à 1 flux *TCP*

De plus, cette équité est également stable en fonction du nombre de flux *TCP* concurrents. Ces améliorations permettent donc de réduire le temps de convergence tout en stabilisant les flux au débit équitable. Ainsi, c'est la régularité des pertes de *TCP* qui permet de synchroniser indirectement les flux *multicast*.

6.3 PROPRIÉTÉS D'UN PROTOCOLE ROBUSTE

Les résultats précédents montrent que les mécanismes proposés par le protocole *M2Cv2* permettent de réduire le temps de convergence. Cependant en analysant les résultats en détail, nous avons aussi constaté que les conditions pour repartir en *SS* sont trop grossières et ne permettent pas l'utilisation du retour en *SS* si l'écart de débits est trop réduit. De plus, les mécanismes de retour en *SS* sont sensibles aux pertes aléatoires qui peuvent survenir par exemple d'un bruit de fond provoqué par des flux concurrents de courte durée.
C'est pourquoi nous voulons améliorer la robustesse de *M2C*. Pour cela, nous allons conserver le principe de retour en *SS* mais en améliorant le mécanisme permettant la transition entre les phases de *CA* et de *SS*. La difficulté est de créer un déclencheur réactif qui démarre le *SS* dès qu'une différence de débit est détectée, mais qui ne va pas générer de faux positifs en rendant le protocole agressif

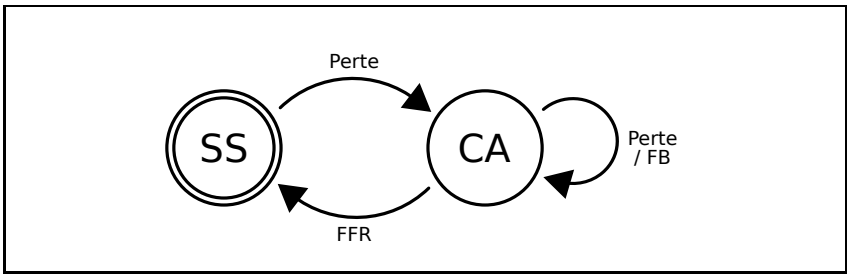


FIGURE 6.4 – Diagramme d'états-transitions de *M2Cv3*

par l'activation de *SS* alors que le débit actuel est proche du débit équitable. De plus, une seconde contrainte est de créer un protocole adapté aux longs flux tels que ceux de l'*IPTV* pour lesquels le protocole doit à la fois :

- Minimiser le temps de convergence pour atteindre rapidement le débit équitable, afin de limiter la période où la variation du débit est importante. De même cette rapidité de convergence est nécessaire quand un flux concurrent s'arrête et libère ainsi de la bande passante, ce qui modifie le niveau du débit équitable.
- Limiter les oscillations, car une fois le débit équitable atteint, le protocole doit adapter son débit de façon à maintenir la qualité de la vidéo et ainsi limiter les nuisances pour l'utilisateur.

Afin de répondre à ces attentes, nous proposons une nouvelle version de *M2C*, appelée *Multicast Congestion Control version 3 (M2Cv3)*, qui améliore à la fois :

- L'estimation du *RTT* afin d'augmenter la précision de la croissance du mécanisme de *CA* et ainsi stabiliser la fonction d'équité.
- La méthode permettant de déterminer quand le débit actuel est loin du débit équitable, afin de pouvoir réactiver l'algorithme de *SS*.
  La question sous-jacente est : comment le protocole peut-il savoir que *EFR* est de loin inférieur au débit équitable et ainsi qu'il doit réactiver l'algorithme de *SS* ? Nous proposons pour cela d'évaluer la durée d'un pseudo cycle de perte de *TCP*.

Au final, *M2Cv3* est une simplification de *M2Cv2* permettant de parfaire l'équité obtenue même dans des situations plus exigeantes. Le diagramme d'états-transitions de *M2Cv3* correspond à la figure 6.4 pour la gestion de l'*EFR*. Ce diagramme ressemble à celui de *TCP*, mais comporte une transition supplémentaire agissant quand l'*EFR* est loin du débit équitable, ou *Far from Fair Rate (FFR)*, qui permet alors au protocole de réactiver le démarrage lent (*SS*).

6.3.1 AMÉLIORATION DE L'ESTIMATION DU *RTT*

L'estimation du *RTT* est une donnée importante qui influence à la fois la pente d'augmentation du mécanisme de *CA*, ainsi que le minuteur décrit ci-dessous pour repartir en *SS*. Ainsi, améliorer *ERTT* signifie à la fois améliorer la fonction d'équité du protocole et maîtriser son agressivité pour le redémarrage en *SS*.
Nous proposons donc d'améliorer la précision du calcul de l'estimation *RTT* (*ERTT*) basée sur :

- La durée aller (*OWD*). Elle correspond à la différence entre la date d'arrivée du paquet et son estampille. Pour que cette mesure fonctionne il faut que la source et les récepteurs aient leurs horloges synchronisées, par exemple avec *NTP* ou par *GPS*.
- L'estimation du temps passé dans les files d'attentes, ou *Estimated Buffering Time (EBT)*. Cette estimation est calculée avec la dernière mesure de la gigue ou *last_jitter*, qui correspond à la différence entre les deux derniers *OWD* :

$$EBT \quad := \quad max(0, EBT + last\_jitter)$$

  Cette mesure est indépendante de la synchronisation des horloges.

Or, le *RTT* est composé :

- d'un temps de propagation aller et retour, ou *forward* (PT_f) et *backward* (PT_b) *Propagation Time*,
- ainsi que d'un temps de mise en tampon aller et retour ou, *forward* (BT_f) et *backward* (BT_b) *Buffering Time*.

Pour une pseudo connexion *TCP*, nous partons du principe que le chemin aller est semblable au chemin retour ($PT_f \simeq PT_b$) et que le chemin retour n'est pas congestionné ($BT_b \simeq 0$), d'où :

$$RTT \simeq (2 * PT) + BT_f$$

Comme $OWD := PT_f + BT_f$ et $EBT \simeq BT_f$:

$$last\_ERTT \quad := \quad 2 * OWD - EBT$$

Et suivant [Jacobson 88], nous calculons *ERTT* comme la moyenne pondérée suivante :

$$ERTT \quad := \quad 0.75 * ERTT + 0.25 * last\_ERTT$$

6.3.2 SOUS-UTILISATION DE LA BANDE PASSANTE OU *Far Below Fair Rate* (FFR)

Nous avons déjà mentionné que la variation du débit en utilisant le mécanisme de *CA* est douce, ce qui mène le protocole à converger lentement vers le débit équitable : par exemple il faut

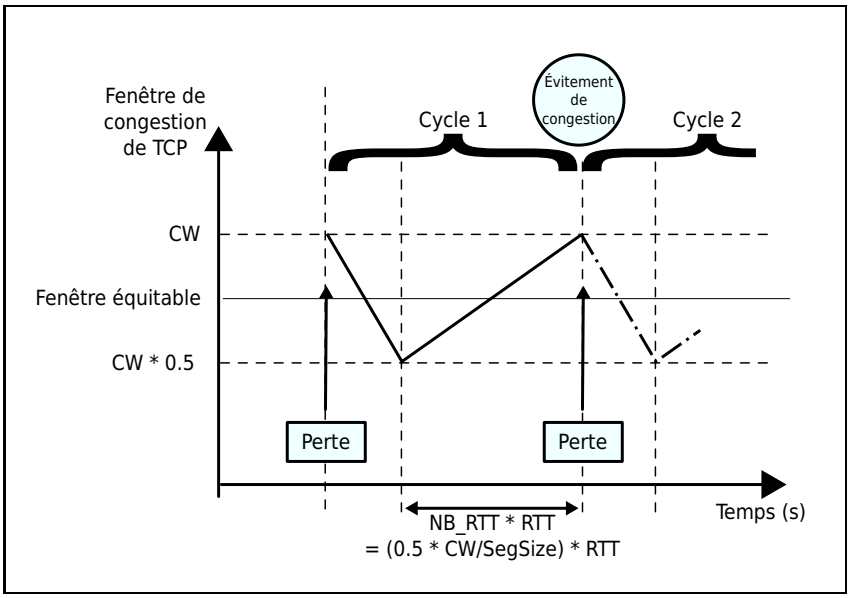


FIGURE 6.5 – Cycle de pertes d'un flux *TCP* au débit équitable

plusieurs heures avec le mécanisme de *CA* pour converger jusqu'à 1 Mb/s avec un *RTT* d'1 seconde.
Ainsi, pour *M2Cv2* nous avons proposé que le protocole utilise :

- Le mécanisme de *CA* lorsque la probabilité d'avoir une perte est importante, pour un instant ou un débit donné. Ainsi, si le protocole est à un débit équitable alors la bande passante est normalement entièrement utilisée et la probabilité d'avoir une perte est importante. Le protocole est alors en *CA* ce qui lui permet de continuer à partager équitablement sa bande passante.
- Le mécanisme de *SS* quand cette probabilité d'avoir une perte est faible. Ainsi, quand le protocole est loin du débit équitable, ce dernier peut utiliser le mécanisme de *SS* pour converger plus rapidement vers le débit équitable.

Pour *M2Cv3*, nous proposons de considérer le problème depuis un autre point de vue. En effet, au lieu de simplement regarder la probabilité de pertes pour éviter que l'utilisation du mécanisme de *SS* rende *M2C* trop agressif, nous proposons d'utiliser un mécanisme capable d'évaluer si le débit actuellement reçu est de loin inférieur au débit équitable (*FFR*).
Quand la bande passante est équitablement partagée avec des flux *TCP* ou des flux considérés comme équitables avec *TCP*, les événements de congestion doivent se produire régulièrement (cf. figure 6.5). Cette régularité nous permet de calculer une estimation de la durée d'un cycle de perte de *TCP*, ou *Transmission Control Protocol Loss Cycle Estimation (TCP_LCE)*. Cette estimation correspond au délai séparant 2 événements de congestion consécutifs

pour un pseudo flux *TCP*.
L'estimation réaliste de ce cycle (*realist_TCP_LCE*), se calcule de la façon suivante, qui est dérivée des études [Mathis 97] [Padhye 98] [Jacobson 88] :

$$\begin{aligned} NB\_RTT &:= 0.5 * \frac{window\_size}{packet\_size} \\ last\_VERTT &:= |last\_ERTT - ERTT| \\ VERTT &:= 0.75 * VERTT + 0.25 * last\_VERTT \\ TCP\_ERTT &:= ERTT + 4 * VERTT \quad \text{[Jacobson 88]} \\ realist\_TCP\_LCE &:= NB\_RTT * TCP\_ERTT \end{aligned}$$

Avec *Variation of Estimated Round Trip Time (VERTT)* la moyenne pondérée de la variation de *ERTT*.
Or, la variation du délai aller (*OWD*) peut être très importante avec un délai d'oscillation potentiellement plus long que la "mémoire" de la moyenne pondérée. Cela peut alors mener à des estimations approximatives de *realist_TCP_LCE* et provoquer des passages non désirés en mode de *SS*. Par conséquent, pour supprimer ce problème, nous proposons que :

- L'estimation *TCP_LCE* soit majorée, tel que :

$$TCP\_LCE := (NB\_RTT * max\_ERTT) + last\_JT$$

  Avec :
  - *maximal Estimated Round Trip Time (max_ERTT)* la valeur maximale de *ERTT* calculée depuis la dernière perte. La variable *max_ERTT* est ainsi utilisée pour lisser les variations de *ERTT*.
  - *last Join Time (last_JT)* la dernière valeur du temps d'adhésion correspondant au délai entre le moment où le récepteur a envoyé un message *(IGMP/MLD)_report* pour s'abonner à un groupe *multicast* et le moment où le premier paquet de ce groupe est reçu.
    Cet ajout est utile quand la durée d'abonnement est largement supérieure à *NumBer of Round Trip Time (NB_RTT)* $*$ *max_ERTT*, car pendant ce délai le débit traversant le routeur en amont du lien faible peut être inférieur au débit souhaité. Par exemple, ce phénomène se produit pour des réseaux avec un *RTT* court et un temps d'adhésion important, tel que ceux de la plateforme entre Strasbourg et Trondheim.
- Pour savoir si le délai du cycle de perte de *TCP* est dépassé, *M2Cv3* doit utiliser un minuteur permettant de connaître le temps écoulé depuis la dernière perte. Or quand un flux observe une perte, cela ne provoque pas forcément la perte d'un paquet pour chaque flux concurrent. Par conséquent,

M2Cv3 doit être capable de réinitialiser le minuteur non seulement quand une perte est détectée, mais encore quand une perte survient uniquement pour un ou des flux concurrents.
Ainsi pour chaque nouveau paquet, *M2Cv3* essaye de déterminer si un flux concurrent a subi un événement de congestion qui n'a pas produit de perte pour *M2Cv3*. Cette situation est considérée comme ayant eu lieu si le tampon des routeurs en amont du lien faible est presque entièrement rempli, ou *Filled Buffer (FB)* :

$$EBT > max\_EBT - VERTT$$

Avec *maximal Estimated Buffering Time (max_EBT)* la valeur maximale calculée pour *EBT* depuis la dernière perte.

Par conséquent, une fois que la valeur du minuteur est supérieure à *TCP_LCE*, *M2Cv3* considère que *EFR* est de trop loin inférieure au débit équitable et repasse alors en mode de *SS*.

6.4 ÉVALUATION DE *M2Cv3*

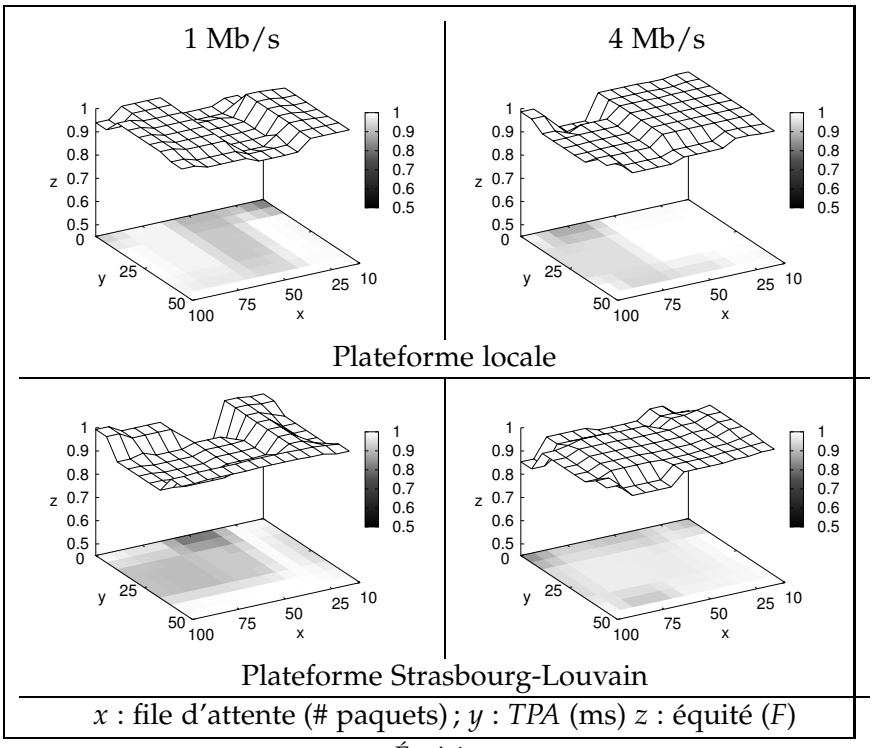
Nous allons maintenant évaluer le comportement de *M2Cv3*. Ainsi, nous allons commencer par tester si les changements apportés à *M2Cv3* ne détériorent pas celui obtenu par *M2Cv2* en réalisant les tests du scénario face à un flux *TCP*. Ensuite, nous étudierons la robustesse de *M2Cv3* avec les scenarii avec *M* flux *M2C* sont face à *N* flux *TCP*, avec du trafic en bruit de fond et avec des flux dont les *RTT* sont hétérogènes

6.4.1 1 FLUX *M2Cv3* FACE À 1 FLUX *TCP*

Afin de vérifier si les mécanismes de *M2Cv3* produisent des résultats équivalents à ceux de *M2Cv2* pour des longs flux stables, nous avons une nouvelle fois rejoué le scénario de concurrence avec un long flux *TCP*. Les résultats obtenus (cf. figure 6.6) confirment que le comportement de *M2Cv3* est équitable ($F \simeq 0.96$) face aux différentes configurations du réseau et ce aussi bien sur la plateforme locale que sur celle entre Strasbourg et Louvain.

6.4.2 *M* FLUX *M2Cv3* FACE À *N* FLUX *TCP*

Ce scénario expérimente le comportement de 5 longs flux en concurrence. Les résultats obtenus (cf. figure 6.7) montrent que *M2Cv3* est équitable ($F \geq 0.94$ en local et $F \geq 0.93$ entre Strasbourg et Louvain) avec de multiples flux concurrents.
On pourrait s'attendre à ce que les tests avec uniquement des flux *M2Cv3* ne soient pas équitable du fait qu'aucun flux *TCP* ne soit

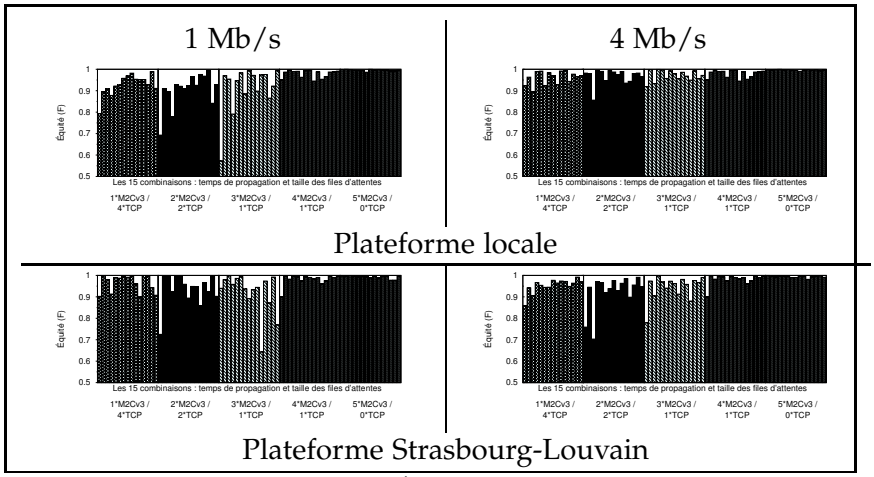


Équité

Test	*BWU*	*LR* (%)	*F*	*CT* (s)		*RT* (s)	
				$1Mb/s$	$4Mb/s$	$1Mb/s$	$4Mb/s$
Plateforme locale							
$1 * M2Cv3$ /$1 * TCP$	0.99 (±0.01)	1.27 (±7.13)	0.97 (±0.08)	34 (±53)	58 (±143)	1 (±2)	0 (±0)
Plateforme Starsbourg-Louvain							
$1 * M2Cv3$ /$1 * TCP$	0.99 (±0.04)	0.73 (±1.61)	0.96 (±0.1)	27 (±34)	53 (±47)	1 (±1)	0 (±0)

(a) Résultats récapitulatifs

FIGURE 6.6 – Résultats pour 1 flux *M2Cv3* face à 1 flux *TCP*

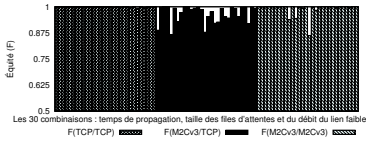


Équité

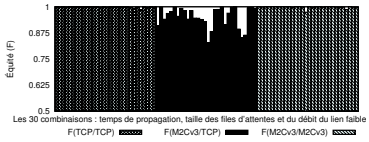
Test	*BWU*	*LR* (%)	*F*	*CT* (s)		*RT* (s)	
				$1Mb/s$	$4Mb/s$	$1Mb/s$	$4Mb/s$
Plateforme locale							
$1*M2Cv3$ $/4*TCP$	0.99 (±0.0)	4.65 (±16.5)	0.94 (±0.07)	16 (±79)	32 (±36)	1 (±3)	0 (±0)
$2*M2Cv3$ $/3*TCP$	0.99 (±0.01)	3.68 (±10.92)	0.94 (±0.16)	22 (±39)	31 (±68)	1 (±1)	0 (±0)
$3*M2Cv3$ $/2*TCP$	0.99 (±0.01)	3.01 (±10.86)	0.94 (±0.15)	19 (±18)	24 (±43)	1 (±1)	0 (±0)
$4*M2Cv3$ $/1*TCP$	0.98 (±0.01)	3.02 (±9.43)	0.97 (±0.09)	26 (±63)	20 (±16)	1 (±1)	0 (±0)
$5*M2Cv3$ $/0*TCP$	0.97 (±0.01)	4.36 (±7.73)	1.0 (±0.01)	12 (±10)	21 (±12)	7 (±2)	0 (±0)
Plateforme Strasbourg-Louvain							
$1*M2Cv3$ $/4*TCP$	0.99 (±0.01)	2.83 (±9.88)	0.96 (±0.06)	34 (±167)	42 (±99)	1 (±1)	0 (±0)
$2*M2Cv3$ $/3*TCP$	0.99 (±0.01)	2.22 (±9.43)	0.94 (±0.08)	25 (±54)	40 (±32)	1 (±1)	0 (±0)
$3*M2Cv3$ $/2*TCP$	0.98 (±0.01)	1.97 (±5.42)	0.93 (±0.16)	29 (±68)	37 (±21)	1 (±1)	0 (±0)
$4*M2Cv3$ $/1*TCP$	0.98 (±0.02)	2.07 (±5.05)	0.97 (±0.08)	24 (±37)	28 (±26)	1 (±1)	0 (±0)
$5*M2Cv3$ $/0*TCP$	0.96 (±0.01)	4.56 (±3.35)	0.99 (±0.02)	13 (±7)	18 (±12)	6 (±3)	0 (±0)

(a) Résultats récapitulatifs

FIGURE 6.7 – Résultats de M flux *M2Cv3* face à N flux *TCP*



(a) Plateforme locale



(b) Plateforme Strasbourg-Louvain

Équité

Test	*BWU*	*LR* (%)	*F*	*CT* (s)		*RT* (s)	
				$1Mb/s$	$4Mb/s$	$1Mb/s$	$4Mb/s$
Plateforme locale							
$2 * M2Cv3$ $/0 * TCP$	0.95 ($\pm$0.03)	4.42 ($\pm$3.92)	0.99 ($\pm$0.05)	19 ($\pm$16)	32 ($\pm$65)	10 ($\pm$2)	9 ($\pm$3)
$1 * M2Cv3$ $/1 * TCP$	0.98 ($\pm$0.01)	1.42 ($\pm$8.73)	0.97 ($\pm$0.09)	35 ($\pm$30)	46 ($\pm$122)	1 ($\pm$0)	1 ($\pm$2)
Plateforme Strasbourg-Louvain							
$2 * M2Cv3$ $/0 * TCP$	0.94 ($\pm$0.04)	3.27 ($\pm$4.07)	1.0 ($\pm$0.01)	13 ($\pm$11)	19 ($\pm$27)	7 ($\pm$4)	1 ($\pm$5)
$1 * M2Cv3$ $/1 * TCP$	0.97 ($\pm$0.03)	0.79 ($\pm$1.76)	0.95 ($\pm$0.1)	30 ($\pm$28)	42 ($\pm$36)	1 ($\pm$1)	1 ($\pm$1)

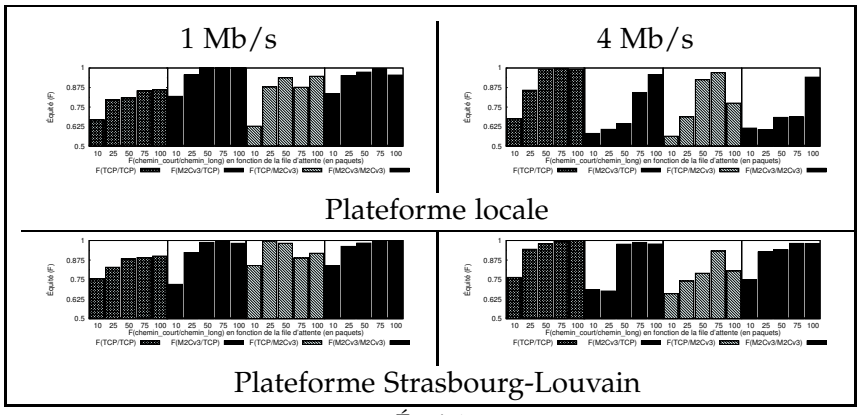
(c) Résultats récapitulatifs

FIGURE 6.8 – Résultats de *M2Cv3* avec du trafic en bruit de fond

présent pour rythmer les cycles de pertes. Or, on peut constater que non seulement les flux *M2Cv3* sont équitables entre-eux, mais encore que le taux de perte généré n'est pas plus important qu'avec les flux *TCP*. Cela signifie bien qu'au niveau rythme des pertes, les flux *M2Cv3* peuvent être considérés comme compatibles avec *TCP*.

6.4.3 *M2Cv3* : TRAFIC EN BRUIT DE FOND

Les résultats (cf. figure 6.8 et tableau 6.8(c)) pour ce scénario montrent que *M2Cv3* n'est pas perturbé par la présence de courts flux concurrents ($F \geq 0.95$). On peut néanmoins noter que l'équité est plus simple à obtenir entre flux *M2Cv3* plutôt qu'en concurrence avec un flux *TCP*. Ce phénomène provient de la fonction *FFR* dont l'efficacité est réduite ou "ralentie" par le bruit de fond qui arrête plus fréquemment le minuteur. Ainsi, face à un flux *TCP*, le minuteur est doublement ralenti par le bruit de fond et par le long flux *TCP*, alors qu'en concurrence avec le flux *M2Cv3* le bruit de fond ralentit tous les flux *multicast* de la même manière et ceux-ci peuvent alors repartir en *SS* en même temps pour converger vers le débit équitable dès que le bruit de fond réduit sa présence.
Ainsi *M2Cv3* est adapté à une utilisation semblable à celle d'une famille connectée à *Internet* en *ADSL* qui a priori mélange toutes sortes d'utilisation du réseau pour :



Équité

Test	*BWU*	*LR* (%)	*F*	*CT* (s)		*RT* (s)	
				$1Mb/s$	$4Mb/s$	$1Mb/s$	$4Mb/s$
Plateforme locale							
M2Cv3 /*M2Cv3*	0.97 (±0.02)	3.97 (±3.81)	0.82 (±0.22)	20 (±3)	32 (±49)	8 (±3)	9 (±6)
TCP /*M2Cv3*	0.99 (±0.0)	2.21 (±6.82)	0.82 (±0.26)	31 (±44)	42 (±39)	1 (±0)	1 (±0)
M2Cv3 /*TCP*	0.99 (±0.0)	2.38 (±4.71)	0.84 (±0.26)	24 (±28)	27 (±23)	1 (±1)	1 (±2)
Plateforme Starsbourg-Louvain							
M2Cv3 /*M2Cv3*	0.96 (±0.02)	2.94 (±3.31)	0.94 (±0.19)	14 (±10)	21 (±29)	7 (±4)	1 (±1)
TCP /*M2Cv3*	0.99 (±0.01)	0.87 (±1.63)	0.86 (±0.2)	27 (±23)	58 (±65)	1 (±1)	1 (±1)
M2Cv3 /*TCP*	0.99 (±0.01)	1.97 (±6.36)	0.89 (±0.21)	41 (±20)	44 (±39)	1 (±0)	1 (±1)

(a) Résultats récapitulatifs

FIGURE 6.9 – Résultats de *M2Cv3* avec des *RTT* hétérogènes

- naviguer et écrire des mails (flux *TCP* courts),
- échanger des fichiers en pair-à-pair (flux *TCP* longs),
- regarder la télévision (flux *multicast* longs).

6.4.4 *M2Cv3* : *RTT* HÉTÉROGÈNES

Ce scénario montre le comportement d'un flux *M2Cv3* en compétition avec un autre flux dont le *RTT* est différent. Il est bien connu que dans cette situation *TCP* n'est pas équitable et il en va de même pour *M2Cv3* (cf. figure 6.9).
De façon générale, cette iniquité croît avec la différence relative entre les *RTT*s des flux concurrents. De plus cette iniquité est accentuée quand le flux *M2Cv3* traverse le chemin le plus court, car dans ces conditions *M2Cv3* calcule un *FFR* plus petit que le cycle de perte du flux *TCP* concurrent. Cela active prématurément *SS* et le flux *M2Cv3* s'approprie alors une grande partie de la bande passante.
Inversement, plus le *RTT* du flux *M2Cv3* est long par rapport à ce-

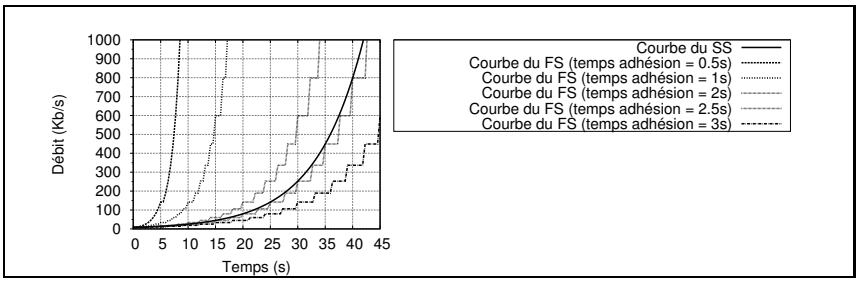


FIGURE 6.10 – Les débits des modes *SS* et *FS*

lui des flux concurrents, plus le minuteur de *FFR* va être surévalué rendant le mécanisme de *FFR* inefficace.
Par contre si le ou les liens faibles se situent en cœur de réseau, alors on peut supposer que :

- Le nombre de flux concurrents est largement supérieur à 2.
- Les flux traversant le cœur de réseau ont un *RTT* assez important. Ainsi, la différence proportionnelle entre les différents *RTT* des flux a de fortes chances d'être réduite, ce qui permet à *M2Cv3* comme à *TCP* d'avoir un comportement plus équitable : en effet avec un *RTT* un peu plus long ($\simeq$ 30 ms), les flux *TCP* et *M2Cv3* sont plus équitables sur la plateforme Strasbourg-Louvain que sur la plateforme locale.

6.5 RAPIDITÉ DE CONVERGENCE AU DÉMARRAGE

Jusqu'à maintenant chaque session de *M2Cv3* commence en mode de *SS*, où l'*EFR* croît de façon exponentielle symétriquement à la réduction du débit des canaux de la source. Ce qui signifie qu'un récepteur double approximativement son débit toutes les 5 secondes (cf. figure 6.10) et cela de façon totalement indépendante du *RTT*. Ainsi, converger vers le débit équitable peut être long et ce, même si le lien faible n'est pas congestionné.
Le comportement idéal serait alors d'augmenter le plus rapidement possible le débit du flux jusqu'à saturer le lien faible. En ce sens, nous proposons *Multicast Congestion Control version 4 (M2Cv4)* qui permet d'améliorer la phase de démarrage de *M2Cv3* avec un nouvel état de démarrage rapide, ou *Fast Start (FS)*. Une fois que toute la bande passante est utilisée, le protocole doit passer en mode de *CA* pour commencer le partage équitable de la bande passante. Par exemple, ce démarrage rapide est utile pour les applications de diffusion de vidéos en résolutions multiples, car il permet de converger rapidement vers la meilleure résolution adaptée aux conditions du réseau et ainsi limiter le temps pendant lequel la qualité de la vidéo varie et la résolution est trop faible.
Avec une croissance rapide du débit reçu, une des difficultés est de pouvoir détecter quand la totalité de la bande passante est utilisée.

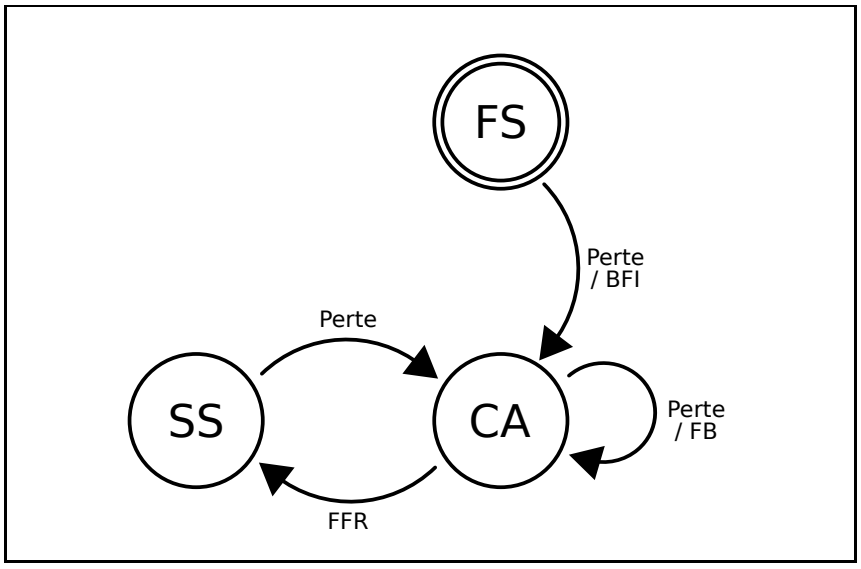


FIGURE 6.11 – Diagramme d'états-transitions de *M2Cv4*

En effet, si ce mécanisme s'arrête trop tardivement, il provoquera de nombreuses pertes en rafales, ce qui peut se traduire pour une application vidéo par un arrêt prolongé sur la dernière image reçue avant les pertes ou par des images affichées avec des portions erronées.

6.5.1 ALGORITHME DU DÉMARRAGE RAPIDE OU *Fast Start* (*FS*)

L'état démarrage rapide (*FS*) est conçu pour utiliser rapidement toute la bande passante disponible. Bien sûr, cette croissance rapide doit être arrêtée dès que le protocole détecte qu'il n'y a plus de bande passante disponible (cf. figure 6.11).
La conception de ce mécanisme se divise en 3 parties :

- *La fonction de croissance.*
  FS est conçu pour s'abonner à un nouveau canal à chaque fois qu'il n'y a plus d'abonnement en cours, c'est-à-dire dès qu'au moins un paquet de données est arrivée pour chaque groupe pour lequel une demande d'adhésion a été émise.
  Ainsi pendant le mode *FS* un récepteur est toujours en train d'essayer de recevoir le prochain groupe *multicast* et ceci indépendemment de la valeur de *EFR*.
  En utilisant le mode de *FS*, le temps de convergence devient alors dépendant et proportionnel au temps d'adhésion (cf. figure 6.10). Ainsi, la croissance de *FS* est plus rapide que celle de *SS* si le temps d'adhésion est < 2.5s (cf. figure 6.10). En effet, la croissance de *SS* est indépendante du temps d'adhésion car *SS* permet à un récepteur avec un long temps d'abonnement de s'abonner à plusieurs groupes simultané-

ment. C'est pourquoi pendant le mode de *FS*, *M2Cv4* continue de faire évoluer l'*EFR* selon la méthode de calcul utilisée pour le *SS*.
Ainsi, le mode *FS* permet de diminuer le temps de convergence au démarrage si le temps d'abonnement est inférieur à 2.5 secondes. Au pire, dans les très rares cas où le temps d'abonnement est supérieur à 2.5 secondes, *FS* est aussi performant que le mode *SS*.

- *La condition d'arrêt.*
  Comme le comportement de *FS* est vraiment agressif, *FS* doit être capable de s'arrêter avant qu'une perte n'apparaisse. L'idée sous-jacente est que *FS* ne peut être employé que si la bande passante n'est pas totalement utilisée. Une fois que cette dernière est entièrement utilisée, *M2Cv4* change d'état et passe alors en *CA* pour commencer à partager équitablement la bande passante.
  Pour détecter de manière proactive si la bande passante est entièrement utilisée, *FS* évalue si les routeurs commencent à mettre des paquets en file d'attente, ou *Buffer Filling In (BFI)*. Cette mise en file d'attente se répercute sur le temps aller (*OWD*), de chaque paquet. Comme la variation de l'*OWD* entre seulement 2 paquets successifs n'est pas significative, *FS* surveille la variation de l'*OWD* pendant une période plus importante. Nous avons déterminé empiriquement que 10 paquets sont suffisants pour déterminer si la variation de l'*OWD* est simplement temporaire ou si les paquets commencent à être mis en file d'attente en prémices d'une congestion. Pour cela, une régression linéaire est calculée sur ces 10 mesures et si le gradient obtenu est > 0.25, alors *M2Cv4* conclut que les routeurs ont commencé à mettre des paquets en file d'attente et passe alors en mode de *CA*.
  Bien entendu, *FS* est également interrompu si une perte survient avant de détecter la mise en file d'attente des paquets. Ce qui peut arriver avec des files d'attente très petite ou si le réseau est déjà saturé.
- *Le débit de reprise.*
  Une fois que la condition d'arrêt a déterminé que la bande passante est entièrement utilisée, l'*EFR* doit être mis à jour. En effet, pendant le mode de *FS* le débit obtenu suite aux adhésions successives ne correspond pas forcément à l'*EFR* calculé. Ainsi à la fin du *FS*, l'*EFR* est mis à jour à la valeur maximale entre :
  - Sa valeur actuelle calculée avec la fonction de croissance de *SS*.
  - Le débit reçu. Pour cela, *FS* calcule le débit obtenu pour les 10 derniers paquets reçus.

6.5.2 *M2Cv4* : TEMPS DE CONVERGENCE

Dans ce scénario nous étudions l'efficacité du mécanisme de *FS* sur *M2C*. Les résultats obtenus (cf. figure 6.12 et 6.13) montrent que le mécanisme de *FS* permet d'améliorer significativement le temps de convergence de *M2C*. Cette amélioration se traduit par plusieurs comportements :

- Sur la plateforme locale, le temps de convergence est réduit de façon significative. Or, cette convergence rapide provoque des rafales de pertes pendant lesquelles l'application peut être incapable d'utiliser correctement les données reçues : par exemple pour une vidéo des pertes de paquets se traduisent par l'affichage d'images comprenant des portions erronées, ce qui en cas de fortes pertes devient très désagréable pour le spectateur. Cependant, l'augmentation du temps de réaction est largement inférieure par rapport au bénéfice engendré pour le temps convergence. Ainsi, le gain dû à l'amélioration du temps de convergence est largement supérieur au temps perdu dû à la rafale de pertes.
  Il est également notable que le bruit de fond n'influence pas le mécanisme de *FS* qui arrive donc à filtrer les variations temporaires du temps aller pour ne pas passer en mode de *CA* avant d'avoir atteint la saturation de la bande passante.
  De plus, on peut remarquer que la durée de réaction aux pertes suivant le mécanisme de *FS* est quasiment indépendante de la taille de la file d'attente des routeurs. Ainsi, la condition d'arrêt de *FS* détecte bien le commencement de la mise en file d'attente des paquets. Le récepteur passe donc en *CA* au lieu de continuer à augmenter le débit demandé, ce qui provoquerait une rafale de perte proportionnelle au débit atteint et donc par conséquent proportionnelle à la taille de la file d'attente des routeurs.
- Sur la plateforme Strasbourg-Louvain, le temps de convergence est également réduit, mais les performances obtenues sont beaucoup plus variables. En effet, on peut observer des tests où la première perte est obtenue largement avant d'avoir convergé jusqu'à un débit proche du débit équitable. Ainsi, les flux concurrents traversant le *mbone* sont visiblement assez importants pour provoquer des congestions sur un autre lien que le lien faible de la plateforme. Ce qui bien entendu arrête le processus de *FS* et force le récepteur à continuer de découvrir la bande passante disponible en n'utilisant que les mécanismes de *CA* et de *SS*.

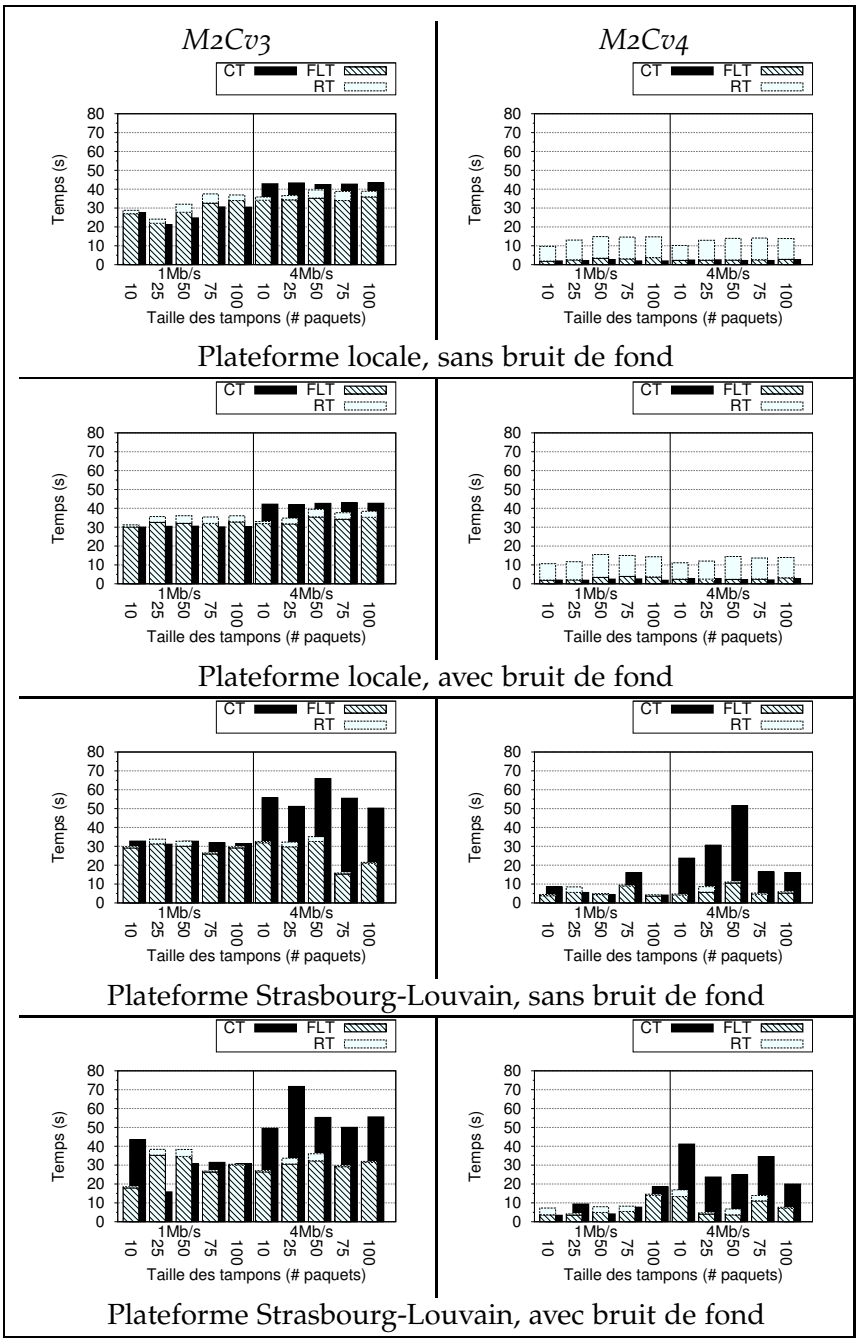


FIGURE 6.12 – Temps de convergence de *M2Cv3* et *M2Cv4*

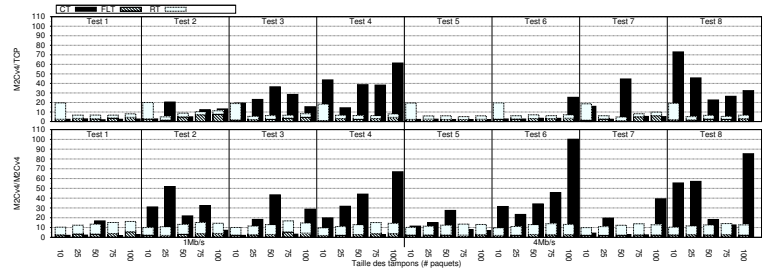
Test	*BWU*	*LR* (%)	*F*	*CT* (s)		*RT* (s)	
				$1Mb/s$	$4Mb/s$	$1Mb/s$	$4Mb/s$
Plateforme locale							
M2Cv3 sans bruit de fond	0.94 (±0.03)	3.77 (±3.71)	*N.A.*	27 (±21)	43 (±2)	3 (±10)	3 (±10)
M2Cv3 avec bruit de fond	0.93 (±0.01)	4.48 (±3.16)	*N.A.*	30 (±1)	42 (±2)	3 (±10)	3 (±10)
Plateforme Strasbourg-Louvain							
M2Cv3 sans bruit de fond	0.93 (±0.03)	2.14 (±3.18)	*N.A.*	32 (±3)	56 (±24)	1 (±7)	1 (±7)
M2Cv3 avec bruit de fond	0.84 (±0.71)	2.51 (±2.73)	*N.A.*	30 (±29)	56 (±22)	2 (±9)	2 (±9)

(a) Résultats récapitulatifs de *M2Cv3*

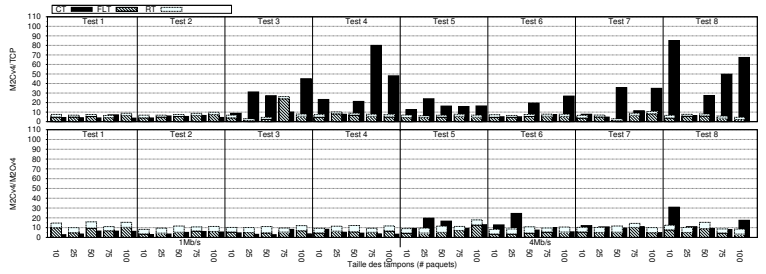
Test	*BWU*	*LR* (%)	*F*	*CT* (s)		*RT* (s)	
				$1Mb/s$	$4Mb/s$	$1Mb/s$	$4Mb/s$
Plateforme locale							
M2Cv4 sans bruit de fond	0.98 (±0.06)	4.17 (±3.78)	*N.A.*	2 (±1)	2 (±1)	10 (±4)	10 (±4)
M2Cv4 avec bruit de fond	0.97 (±0.02)	5.28 (±2.78)	*N.A.*	2 (±1)	2 (±1)	10 (±4)	10 (±4)
Plateforme Strasbourg-Louvain							
M2Cv4 sans bruit de fond	0.97 (±0.03)	2.19 (±2.63)	*N.A.*	8 (±19)	28 (±35)	1 (±4)	1 (±4)
M2Cv4 avec bruit de fond	0.95 (±0.02)	2.66 (±3.71)	*N.A.*	9 (±24)	29 (±28)	2 (±7)	2 (±7)

(b) Résultats récapitulatifs de *M2Cv4*

FIGURE 6.13 – Résultats comparatifs de *M2Cv3* et *M2Cv4*pour le scénario du temps de convergence



(a) Plateforme locale



(b) Plateforme Strasbourg-Louvain

Paramètres de convergence de *M2Cv4*

Test	*BWU*	*LR* (%)	*F*	*CT* (s)		*RT* (s)	
				$1Mb/s$	$4Mb/s$	$1Mb/s$	$4Mb/s$
Plateforme locale							
$2 * M2Cv4$	0.98 (±0.02)	4.91 (±3.45)	0.98 (±0.05)	22 (±41)	30 (±61)	10 (±4)	10 (±4)
$1 * M2Cv4$ $/1 * TCP$	0.99 (±0.01)	2.84 (±7.0)	0.96 (±0.12)	19 (±33)	16 (±38)	6 (±6)	6 (±6)
Plateforme Strasbourg-Louvain							
$2 * M2Cv4$	0.97 (±0.02)	3.5 (±3.81)	1.0 (±0.02)	5 (±5)	13 (±14)	5 (±6)	5 (±6)
$1 * M2Cv4$ $/1 * TCP$	0.98 (±0.02)	1.8 (±3.41)	0.98 (±0.05)	18 (±43)	24 (±61)	2 (±6)	2 (±6)

(c) Résultats récapitulatifs

FIGURE 6.14 – Résultats de *M2Cv4* avec des liens faibles multiples

6.6 ÉVALUATION DE *M2Cv4* DANS UN ENVIRONNEMENT ÉPROUVANT

Nous avons présenté l'évolution d'un protocole de contrôle de congestion pour le *multicast* jusqu'à obtenir *M2Cv4* qui répond aux attentes d'équité et de rapidité de convergence dans des situations spécifiques correspondants aux comportements à tester. Cette partie doit permettre de compléter l'évaluation de *M2Cv4* dans des environnements exigeants, capables de mettre à l'épreuve ses différentes capacités.

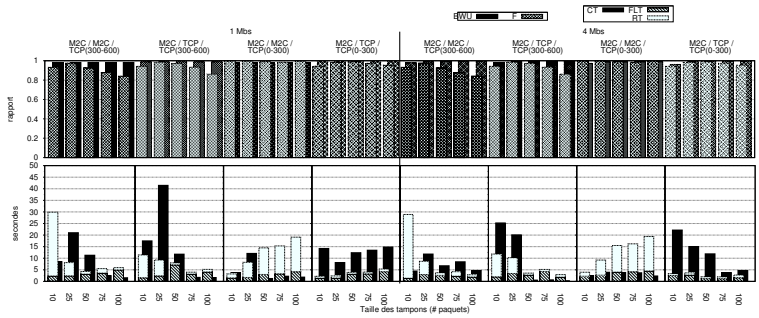
6.6.1 *M2Cv4* : LIENS FAIBLES MULTIPLES

Ce scénario permet de mettre en évidence le comportement de *M2Cv4* en présence de toutes les situations précédentes (différents temps ajoutés, tailles de file d'attente, avec ou sans bruit de fond) et avec des liens faibles multiples dont les temps de mise en file d'attente sont différents selon le flux, produisant ainsi des *RTT* hétérogènes.
Les résultats obtenus (cf. figure 6.14 et tableau 6.14(c)) pour ce scénario, montrent que *M2Cv4* utilise correctement la bande passante ($BWU \geq 0.97$) et ce même s'il n'y a pas de flux *TCP* concurrent. De plus, cette bande passante est équitablement répartie entre les flux concurrents ($F \geq 0.96$).
Par contre le temps de convergence est relatif à la configuration des débits utilisés :

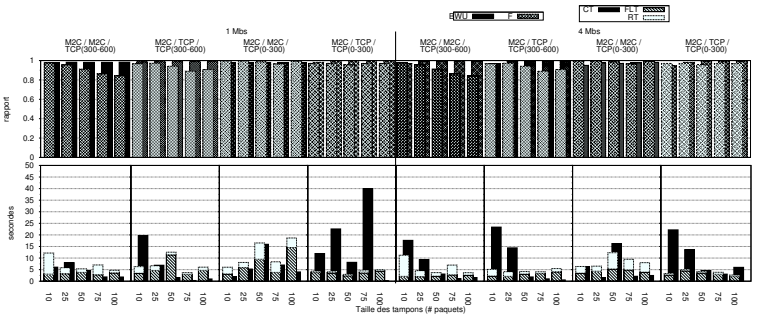
- La convergence est rapide pour les tests où le débit équitable correspond à la bande passante laissée par le flux concurrent qui lui est uniquement limité par le deuxième lien faible (cf. tests 1 et 5).
  CT est également court mais de façon variable quand le flux concurrent laisse une partie de la bande passante, mais que le débit équitable est supérieur à ce vide (cf. tests 2 et 6). En effet, dans ces situations le mécanisme de *FS* doit arrêté quand le débit disponible est atteint afin de permette au mécanisme de *CA* de partager la bande passante de façon équitable. Or, pour les tests 2 et 6 le débit disponible est inférieur au débit équitable et il faut attendre le retour en *SS* avant de pouvoir atteindre ce dernier.
- La convergence est plus lente et variable quand le flux concurrent n'est pas limité par le second lien faible. Dans ces conditions, le temps de convergence est plus dépendant du mécanisme de *SS* que de celui de *FS*.

6.6.2 *M2Cv4* : VARIATION DU NOMBRE DE FLUX CONCURRENTS

Ce scénario a pour but de mettre en évidence le comportements des protocoles après qu'un changement important ait eu lieu dans le réseau. Ici, nous analyserons donc uniquement les résultats obtenus après le démarrage ou l'arrêt du troisième flux, c'est-à-dire entre 300 et 600 secondes.
Dans ces conditions, les résultats obtenus (cf. figure 6.15 et tableau 6.15(c)) montrent que *M2Cv4* utilise entièrement la bande passante ($BWU \geq 0.98$) et ce même s'il n'y a pas de flux *TCP* concurrents.
En outre concernant l'équité avec les flux concurrents, deux comportements sont identifiables :



(a) Plateforme locale



(b) Plateforme Strasbourg-Louvain

Équité et paramètres de convergence

Test	*BWU*	*LR* (%)	*F*	*CT* (s)		*RT* (s)	
				$1Mb/s$	$4Mb/s$	$1Mb/s$	$4Mb/s$
Plateforme locale							
*2*M2Cv4/ TCP(300-600)*	0.99 (±0.01)	2.59 (±8.41)	0.91 (±0.09)	9 (±17)	7 (±7)	7 (±8)	7 (±8)
*2*M2Cv4/ TCP(0-300)*	0.98 (±0.01)	3.22 (±4.04)	0.99 (±0.03)	4 (±22)	3 (±3)	9 (±9)	9 (±9)
M2Cv4/TCP/ TCP(300-600)	0.99 (±0.01)	2.7 (±10.66)	0.94 (±0.15)	15 (±39)	9 (±43)	4 (±31)	4 (±31)
M2Cv4/TCP/ TCP(0-300)	0.99 (±0.01)	1.21 (±5.95)	0.97 (±0.07)	13 (±29)	12 (±31)	1 (±1)	1 (±1)
Plateforme Strasbourg-Louvain							
*2*M2Cv4/ TCP(300-600)*	0.98 (±0.02)	1.66 (±2.37)	0.91 (±0.12)	4 (±11)	7 (±23)	4 (±12)	4 (±12)
*2*M2Cv4/ TCP(0-300)*	0.98 (±0.03)	2.44 (±3.36)	0.99 (±0.05)	7 (±35)	6 (±34)	4 (±10)	4 (±10)
M2Cv4/TCP/ TCP(300-600)	0.99 (±0.01)	1.35 (±3.95)	0.94 (±0.17)	6 (±44)	8 (±34)	2 (±5)	2 (±5)
M2Cv4/TCP/ TCP(0-300)	0.98 (±0.04)	0.71 (±1.33)	0.97 (±0.06)	17 (±102)	10 (±31)	1 (±0)	1 (±0)

(c) Résultats récapitulatifs

FIGURE 6.15 – Résultats de *M2Cv4* : variation du nombre de flux concurrents

- Un comportement très équitable ($F \geq 0.97$) pour les tests où le troisième flux s'arrête à 300 secondes et libère alors un tiers de la bande passante. Dans cette situation, les flux *M2Cv4* doivent être capable de récupérer de façon équitable la bande passante qui vient d'être libérée. Or, la méthode d'acquisition de la bande passante de *M2Cv4* est efficace, notamment grâce au mécanisme *FFR*.
  Sans flux *TCP* concurrent, la convergence de *M2Cv4* est alors plus courte mais provoque une rafale de pertes plus importante. Finalement, après convergence et réaction à la congestion la session est réellement utilisable au nouveau débit équitable a peu près au même moment (< 20 secondes) avec ou sans flux *TCP* concurrent.
- Un comportement plus conservateur pour les tests où le troisième flux commence à 300 secondes et doit donc acquérir un tiers de la bande passante. Dans cette situation, les flux *M2Cv4* réduisent lentement leurs débits avec une rapidité relative au rythme des pertes produites par les flux *TCP* concurrents. Or, quand le *RTT* augmente la fréquence des pertes *TCP* diminue, ce qui ralentit la réduction du débit utilisé par *M2Cv4*. Ainsi, les 300 secondes d'expérimentations dans ces conditions ne sont pas forcément suffisantes pour que *M2Cv4* puisse atteindre son nouveau débit équitable. En effet, les résultats obtenus montrent une équité qui décroît quasiment jusqu'à $F \simeq 0.8$ quand le *RTT* est à plus d'1 seconde.

Finalement, le comportement de *M2Cv4* est rapide pour acquérir de la bande passante et plus lent pour la libérer. Cela permet donc de stabiliser le débit reçu : par exemple pour un flux vidéo encodé hiérarchiquement, ce comportement permet d'obtenir rapidement la qualité maximale correspondant aux capacités du réseau tout en réduisant en douceur cette qualité quand d'autres flux entrent en concurrence.

6.7 SIGNALISATION ENGENDRÉE PAR *M2C*

La signalisation de *M2C* est régulée par l'utilisation d'une source dont les groupes *multicast* sont à débit dynamique. Ainsi comme *M2C* utilise un *TSD* de 10 secondes et un facteur d'accélération de diminution de débit $K := 4$, sa fréquence de signalisation est alors de $SO := 0.8$ messages par secondes et par récepteur. Cette fréquence de signalisation correspond à celle engendrée par récepteur pendant les différents tests.
Nous avons déjà mentionné que le débit de la signalisation est quasiment insignifiant et que la charge principale engendrée par la signalisation est le coût du calcul des tables de routage à chaque

changement d'état. Il est donc notable que cette mesure de la signalisation ne prend en compte que les messages générés par le protocole de contrôle de congestion *multicast*, pour lesquels il résulte un changement d'état des tables de routage *multicast*. Ainsi, cette mesure ne prend pas en compte :

- Les différents messages envoyés par le routeur comme les *IGMP_query* généraux [Cain 02] (envoyés par défaut tous les $Query\_Interval = 125$ secondes) et spécifiques (envoyés quand le routeur vient de recevoir la demande de désabonnement du dernier client d'un groupe).
- Le facteur de robustesse de *IGMP*, qui par défaut provoque l'envoi d'une deuxième copie des différents *IGMP_report* et des *IGMP_query* spécifiques.

Un autre point important pour la signalisation est sa fiabilité. En effet, les messages *IGMP* et *PIM-SM* n'utilisent pas de protocoles communications fiables [3]. Par conséquent, les différents paquets de signalisation peuvent donc être perdus, auquel cas :

- Si un message de désabonnement est perdu, alors un protocole de contrôle de congestion *multicast* utilisant des groupes à débit dynamique n'est pas perturbé. En effet, l'utilisation de groupes dynamiques oblige les récepteurs à ne se désabonner que lorsque le groupe est muet. Par conséquent, aucun paquet non sollicité ne traversera les routeurs pour lesquels la demande de désabonnement n'est pas arrivée.
  Il faut noter qu'en l'absence de demande de désabonnement explicite, les routeurs gardent l'entrée correspondante au groupe dans leur table de routage *multicast* par défaut :
  - pendant $(Robustness\_Variable * Query\_Interval) + Query\_Response\_Interval = (2 * 125) + 10 = 260$ secondes si l'abonnement provient d'un message *IGMP*,
  - ou pendant $PIM\_Keepalive\_Period = 210$ secondes si l'abonnement provient d'un message *PIM-SM*.

  Ainsi, la perte d'un message de désabonnement ne peut poser problème que si la source réutilise le même groupe *multicast* avant que ces minuteurs ne soient expirés. En effet, dans ce cas les routeurs encore abonnés recevraient les paquets du groupe *multicast* une fois celui-ci de nouveau actif. Or, comme un groupe dynamique commence un cycle à plein débit cela pourrait alors provoquer une congestion pour les routeurs encore abonnés. Cependant, ce scénario ne peut pas survenir avec *WEBRC* et *M2C* qui définissent un temps de silence de 300 secondes avant de réutiliser un groupe *multicast*.

3. La fiabilité pour *IGMP* est limité au facteur de robustesse qui définit le nombre d'occurrences d'un message à envoyer.

- Si un message d'abonnement est perdu, alors cela se répercute sur le nombre de paquets reçus, ce qui crée alors une différence entre le débit auquel la source est en train d'émettre et le débit qui traverse réellement le réseau. Cela fausse alors les calculs des récepteurs, qui peuvent alors surévaluer la bande passante disponible pendant le temps du dysfonctionnement.
  En *IGMP*, un récepteur doit attendre un nouveau *IGMP_query* général (toutes les $Query\_Interval = 125$ secondes) pour remanifester son désir de s'abonner à un groupe. Avec *PIM-SM*, il faut attendre la retransmission périodique (toutes les $t\_periodic = 60$ secondes) des messages *JOIN / PRUNE*. Or, comme un canal dynamique réduit approximativement son débit de moitié toutes les 5 secondes, ces retransmissions d'abonnements périodiques sont quasiment inutiles, car ils arriveront en moyenne après que le groupe soit devenu muet ou au mieux quand ce dernier n'aura plus qu'un débit minimal. Afin de corriger ce type de problèmes, une solution proposée [Farinacci 10] est de pouvoir utiliser *PIM-SM* au dessus de *TCP* ou de *Stream Control Transmission Protocol (SCTP)* (cf. RFC4960 [Stewart 07]).
  À noter que contrairement à *WEBRC*, *M2C* peut continuer à s'abonner aux groupes *multicast* de débit supérieur pour limiter l'impact de ce problème à une différence de débit correspondant uniquement aux débits des groupes pour lesquels les abonnements se sont perdus, ce qui peut se révéler utile pour les applications capables d'utiliser les données du groupe $N+1$ sans recevoir le groupe N, comme par exemple pour le logiciel de transfert de fichiers présenté ci-dessous (cf. § 8.1).

6.8 ANALOGIES ET DIFFÉRENCES PRINCIPALES ENTRE *M2C* ET *TCP-newReno*

Nous avons fondé et développé *M2C* en partant du principe que pour être équitable nous devions créer les différents mécanismes de *M2C* en nous basant sur ceux de *TCP-newReno*. Dans ce paragraphe, nous allons faire la synthèse des analogies et différences des mécanismes entre ces 2 protocoles :

- Tout d'abord, les 2 protocoles utilisent une fenêtre de congestion (*CW*). Cependant contrairement à *TCP*, ce sont les récepteurs *M2C* qui gèrent cette fenêtre car chaque récepteur calcule son débit équitable et s'abonne au nombre de groupe correspondant.

- Ensuite, *M2C* tout comme *TCP-newReno* utilise la phase de *CA* pour gérer la taille de *CW* afin de partager équitablement la bande passante via un mécanisme de type *AIMD*. Cependant, *M2C* utilise un coefficient de réduction (β) différent de celui de *TCP* car la réduction du débit d'un flux *M2C* est dépendante de la réduction de la source pour le débit de chaque groupe *multicast*. Ainsi, pour *M2C* le coefficient β varie en fonction du temps inter-paquet, alors que pour *TCP* β est constant et est égal à 0.5.
  Par conséquent, le coefficient de croissance (α) est également différent entre *M2C* et *TCP*, mais est adapté par rapport au β utilisé pour rester compatible et équitable avec *TCP*.
- Puis, il n'existe pas de *RTT* pour les flux *M2C* car ces derniers n'envoient pas de messages vers la source afin de conserver la capacité du *multicast IP* à supporter le passage à très grande échelle. Or, estimer le *RTT* est essentiel pour rester équitable avec *TCP* car la fonction de croissance du processus *AIMD* est dépendante du *RTT*. Ainsi, *M2C* calcule une estimation du *RTT* (*ERTT*) grâce au temps aller (*OWD*) et au temps de mise en file d'attente (*EBT*) des paquets de données reçus. Ainsi, la synchronisation des horloges de la source et des récepteurs est nécessaire pour la mesure de *OWD*.
- À noter que contrairement à *TCP*, *M2C* ne réinitialise pas *EFR* et ne retourne pas en phase de *SS* quand une importante congestion est détectée. Cela car un récepteur est incapable de modifier rapidement le débit reçu pour que ce dernier corresponde à l'*EFR* minimale. En effet, la source rythme la réduction des débits de tous les récepteurs par l'utilisation des canaux dynamiques. De plus, se désabonner de chaque groupe prend un temps conséquent et un récepteur est incapable de savoir quand le routeur en amont du lien congestionné est réellement désabonné.
- Par contre, *M2C* retourne en phase de *SS* quand son débit est trop faible comparativement au débit équitable, ce qui permet de converger plus rapidement vers le débit équitable. Ce mécanisme n'existe pas pour *TCPnewReno* mais des mécanismes équivalents ont été intégrés dans les différentes versions de *TCP* adressant le problème de la rapidité de convergence pour des réseaux à grand débit et à long délai.
- Enfin à l'instar de *TCP*, les mécanismes de *CA* et de *SS* ne suffisent pas à *M2C* pour démarrer et converger rapidement vers le débit équitable. Ainsi, le mécanisme de *FS* permet à chaque nouveau flux *M2C* de découvrir rapidement la bande passante disponible.

6.9 CONCLUSION

Les protocoles de contrôle de congestion *multicast* se sont efforcés les uns après les autres d'améliorer la source pour s'affranchir de différents problèmes, tel que le problème du temps de désabonnement.
Cependant, pendant tout ce travail la capacité des récepteurs à obtenir un débit équitable n'a été que très peu évaluée. Comme les évaluations précédentes ont toutes été réalisées par simulations, nous avons mis en place plusieurs plateformes pour réaliser des expérimentations dans un environnement réaliste. C'est ainsi que le problème du temps d'adhésion fut pris en compte ainsi que son impact sur les différents protocoles de contrôle de congestion *multicast*.
En plus de ce problème, nous avons évalué de façon exhaustive les performances de *WEBRC* qui est le seul protocole à prendre en considération dans sa conception le problème du partage équitable de la bande passante. Or, même si dans les cas les plus statiques *WEBRC* est équitable, cette équité n'est pas robuste et présente des faiblesses quand le nombre et la dynamique des flux concurrents augmentent.
À partir de ce constat, il était nécessaire de travailler sur les algorithmes côté récepteur pour améliorer l'équité des flux *multicast*. Nous avons donc proposé 2 protocoles basiques : l'un se basant sur l'étude des variations de la gigue (*RR*) et l'autre utilisant un mécanisme de fenêtre de congestion côté récepteurs (*M2C*). Afin de supprimer des problèmes de convergence, d'augmenter la robustesse et d'accélérer le démarrage, nous avons proposé des améliorations incrémentales pour *M2C*. L'évaluation exhaustive de ce protocole montre sa capacité à partager équitablement la bande passante de façon robuste avec une courte durée de convergence.

Organisation des données pour l'application

7 INTERFACE AVEC L'APPLICATION

M2C est un protocole de contrôle de congestion dont nous avons démontré l'efficacité, notamment en terme d'équité. Cependant, savoir utiliser efficacement un protocole de contrôle de congestion à débits dynamiques n'est pas trivial car même en parfaite connaissance de ce type de protocole, il n'est pas évident d'imaginer comment une application peut en tirer parti. Nous allons donc commencer par étudier plus précisément ce qui rend l'utilisation de ces protocoles complexe à programmer au sein du niveau applicatif. Puis, nous proposerons et évaluerons une solution générique adaptée à différents types d'applications.

7.1 UTILISATION DES PROTOCOLES DE CONTRÔLE DE CONGESTION À DÉBITS DYNAMIQUES

M2C ainsi que tous les autres protocoles de contrôle de congestion pour le *multicast* à débits multiples disposent d'une source qui émet un unique flux de données. Cependant ce flux est diffusé par plusieurs groupes *multicast*, ce qui permet à chaque récepteur d'obtenir un débit adapté à ses capacités. Ainsi, chaque récepteur obtient une quantité différente de données envoyées par la source. Par conséquent, malgré des résultats prometteurs notamment en terme d'équité, il est difficile pour un développeur de créer une application pour utiliser efficacement ces protocoles afin d'envoyer les données de façon à ce qu'indépendamment de la quantité obtenue, les données reçues soient toujours utiles et utilisables par chaque récepteur. Ce problème peut alors être résumé par l'interrogation suivante :

- Est-il possible d'envoyer les données applicatives de manière à ce que chaque récepteur obtienne uniquement les données les plus importantes qui lui sont utiles ?

Or, chaque récepteur s'abonne de façon cumulative et hiérarchique, un récepteur ne peut s'abonner au groupe N qu'après s'être abonné à tous les groupes de 0 à $N-1$. Ainsi, les données envoyées aux groupes les plus bas de la hiérarchie sont reçues par la plupart des récepteurs. Par conséquent, l'application doit être capable d'en-

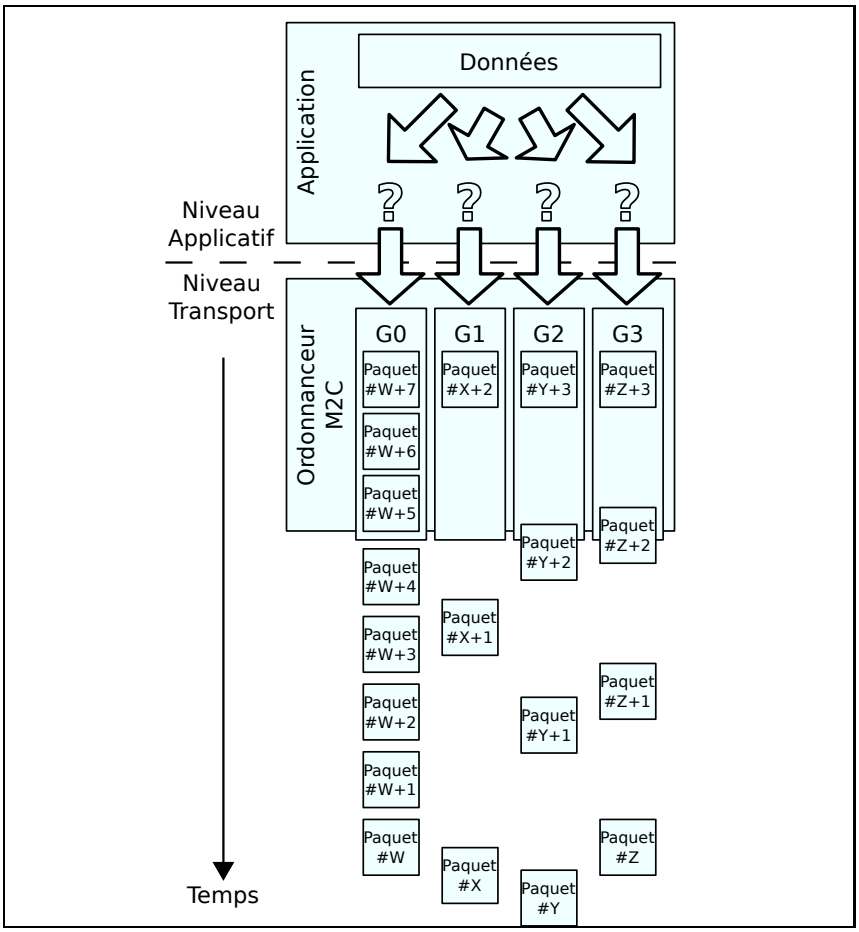


FIGURE 7.1 – Diagramme de flux de la source *M2C*

voyer les données les plus importantes via les groupes les plus bas de la hiérarchie.

Cependant, l'utilisation de groupes à débits dynamiques rajoute une difficulté supplémentaire pour effectuer cette répartition des données envoyées par la source. En effet, la source de *M2C*, tout comme celle de *WEBRC*, utilise un ordonnanceur qui se présente sous la forme suivante (cf. figure 7.1) :

- En entrée, l'ordonnanceur de *M2C* nécessite une liste de tampons de données. Chaque tampon est attribué à un groupe dynamique qui va envoyer un maximum de données en fonction de l'ordonnanceur décrit ci-dessous.
- L'ordonnanceur de *M2C* regarde pour chaque groupe la date d'émission du prochain paquet. Ainsi, si un paquet est prêt à être émis pour le groupe souhaité, alors l'ordonnanceur copie un maximum de données dans le paquet avant de l'envoyer. Par contre, aucune donnée ne sera envoyée s'il n'y a pas de paquet ordonnancé pour le groupe souhaité. Si un paquet est envoyé, alors l'ordonnanceur calcule la date d'émis-

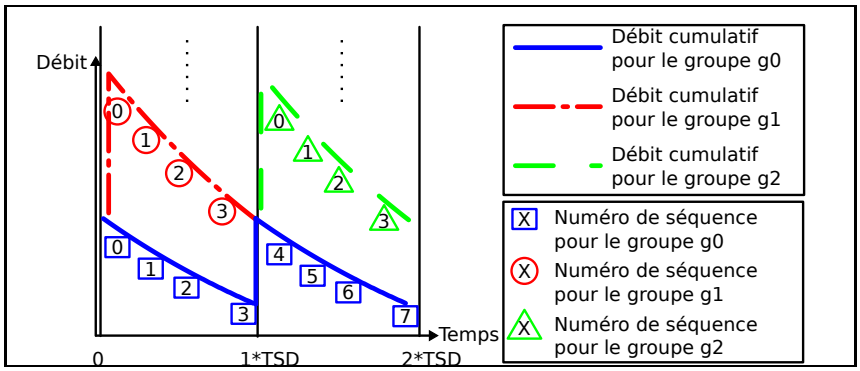


FIGURE 7.2 – L'ordonnanceur de paquets de la source pour $K = 1$

sion du prochain paquet pour le débit actuel du groupe.

- Après avoir parcouru tous les groupes, l'ordonnanceur retourne, à l'application appelante, la liste du nombre d'octets envoyés pour chaque groupe. et renvoie à l'application le nombre de donnée envoyé pour chaque groupe.

Ainsi, l'interface de *M2C* est difficile à utiliser car pour envoyer ses données de façon hiérarchique l'application doit connaître les dates d'émission des paquets et les débits des différents groupes afin de ne pas se retrouver face au problème où une application souhaite envoyer les données #1, #2 et #3 sur les 3 premiers groupes *multicast* alors que seuls les groupes 1 et 3 ont des paquets prêts à être émis. Dans cette situation un récepteur abonné aux groupes 1, 2 et 3 recevra les données #1 et #3, ce qui constitue un trou dans la séquence, qui selon l'application peut rendre les données #3 inutilisables.
Pour comprendre la solution proposée ci-dessous, nous allons revenir en détail sur les mécanismes de l'ordonnanceur de *M2C* pour la source et sur l'adhésion hiérarchique aux groupes pour les récepteurs. Cette description est basée sur *M2C*, mais est également valable pour *WEBRC* quand $K = 1$.

7.1.1 CÔTÉ SOURCE

Nous avons vu que la source envoie les données en utilisant des groupes à débits dynamiques, où chaque groupe commence avec un débit élevé et diminue progressivement son débit jusqu'à atteindre un seuil minimal. Une fois ce seuil atteint, le groupe cesse d'émettre et devient alors muet. Seul le groupe de base (cf. $g0$ à la figure 7.2) ne devient jamais muet.
La source utilise également une subdivision temporelle nommée *TSI*, qui change toutes les *TSD* secondes. À chaque nouveau *TSI*, K groupes deviennent alors muets et K nouveaux démarrent aux K plus hauts débits.

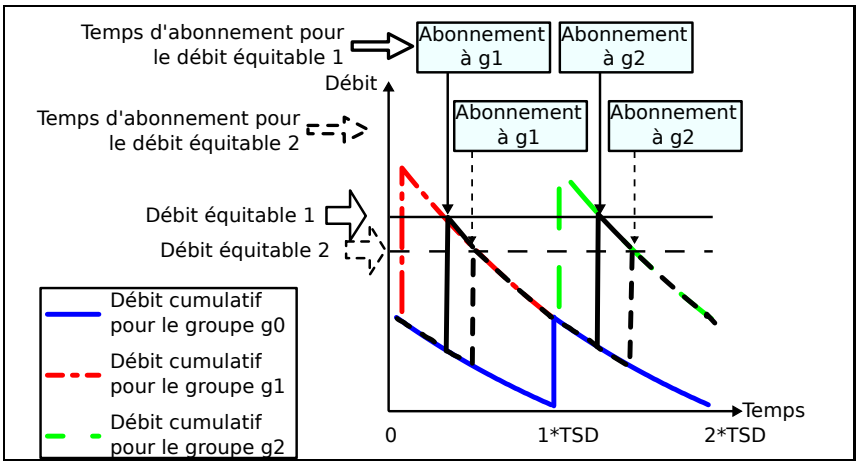


FIGURE 7.3 – Granularité fine des débits grâce aux débits dynamiques de la source

C'est alors l'ordonnanceur de paquets (cf. figure 7.2) qui calcule le débit courant d'un groupe comme étant un nombre de paquets à envoyer pour un *TSI* donné avec une croissance exponentielle du délai inter-paquets. Ce qui provoque une réduction exponentielle du débit d'un facteur P^K par *TSD*. Le débit courant est recalculé pour chaque paquet, tel que décrit dans la partie sur la source améliorée :

$$current\_rate \quad = \quad (R_{start} * P^{\frac{(t*K)}{TSD}})$$

avec R_{start} le débit maximal du groupe quand il devient actif et t le temps écoulé depuis que le groupe est devenu actif.

7.1.2 CÔTÉ RÉCEPTEUR

Il est notable, que les récepteurs d'une source à débits statiques peuvent uniquement recevoir un ensemble de débits dont la granularité est grossière et correspond à la granularité des débits entre les groupes émis par la source. Alors que *M2C* ainsi que tous les autres protocoles utilisant des groupes à débits dynamiques, permettent à chaque récepteur d'obtenir quasiment n'importe quel débit désiré en modulant uniquement en retardant ou en avançant les abonnements et ce indépendemment de la granularité des débits entre les groupes émis par la source. En effet, la figure 7.3 montre qu'un récepteur peut obtenir le *débit 1* en s'abonnant aux groupes aux moments spécifiés, alors qu'un autre récepteur peut obtenir le *débit 2* ($<$ *débit 1*) simplement en retardant ses abonnements et ce avec un écart entre le *débit 1* et le *débit 2* plus

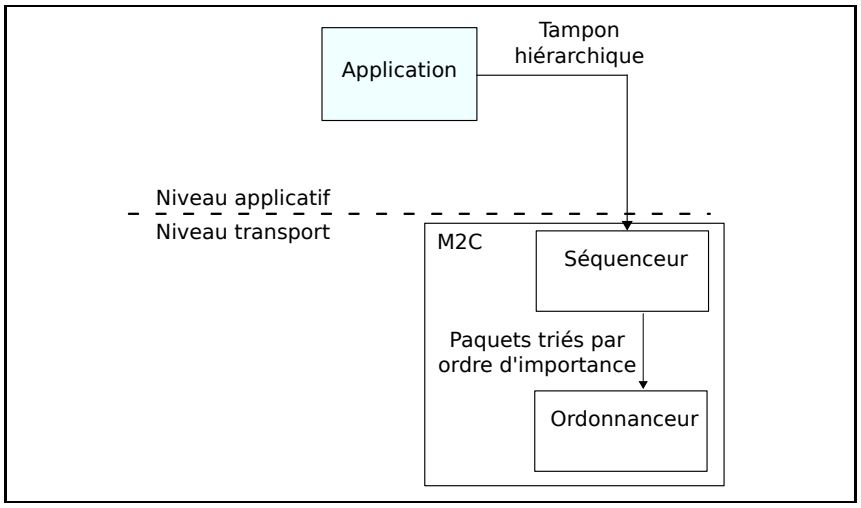


FIGURE 7.4 – Diagramme de flux d'un logiciel utilisant *M2C*

petit qu'entre les débits cumulatifs [1]de $g1$ et $g2$.
Ainsi si l'application le permet, la source doit être capable d'envoyer des données utiles pour n'importe quel débit reçu.

7.2 OBJECTIF

Nous proposons de créer un séquenceur (cf. figure 7.4) qui fait concorder de façon optimale les données applicatives avec les groupes à débits dynamiques. Les applications peuvent aisément bénéficier de ce séquenceur au moyen d'une interface de programmation, ou *API*, créée pour masquer le fonctionnement et la complexité des groupes à canaux dynamiques. Cette *API* permet d'utiliser *M2C* avec différents types d'applications, tel que un logiciel de transfert de fichiers ou un logiciel de diffusion de vidéos, que nous détaillerons dans le chapitre suivant.
En effet, l'*API* proposée est simple d'utilisation car elle fonctionne tant que l'application sus-jacente est capable de fournir un tampon de données contenant les données les plus importantes en premier, la taille du tampon et le délai pendant lequel le protocole de contrôle de congestion peut envoyer ce tampon. L'application est libre de trier les informations du tampon avec une précision fine ou grossière. De plus, cette précision est totalement indépendante de la granularité des débits utilisés pour les groupes *multicast*. Ainsi, différentes applications peuvent utiliser cette *API* sans savoir quoi que ce soit à propos des débits utilisés et de leurs dynamicités. Respectivement, le séquenceur est indépendant de l'application, car il fonctionne sans savoir de quelle manière les données ont été

1. Le débit cumulatif pour le groupe de niveau L correspond à $\sum_{l=0}^{L}(rate_l)$, où $rate_l$ est le débit courant pour le groupe l.

encodées. Le séquenceur envoie alors les données les plus importantes en premier, ce qui permet à un récepteur abonné à N% du flux de la source, d'obtenir les N premiers % de chaque tampon envoyé par la source.

7.3 PRINCIPE DU SÉQUENCEUR

Pour illustrer le principe du séquenceur, imaginons que nous souhaitions envoyer une vidéo en utilisant *M2C*, où chaque récepteur obtient une résolution de la vidéo correspondant au débit reçu. Pour chaque image, le logiciel de diffusion de vidéo génère un tampon de données contenant les différentes résolutions de l'image. Comme les données envoyées aux groupes les plus bas dans la hiérarchie sont reçues par la majorité des récepteurs, notre but est d'envoyer les données les plus importantes à ces groupes de rang inférieur. Comme le débit de chaque groupe change continuellement, une définition plus générique et à la fois plus précise est que la source doit envoyer les données les plus importantes au débit le plus bas possible. Or, ce calcul est trop spécifique et complexe pour être géré au niveau applicatif.
De manière plus précise, le séquenceur liste tous les paquets qui seront transmis pendant le temps disponible pour envoyer le tampon. Puis, le séquenceur trie les paquets par débit et les fait correspondre aux données applicatives. Nous proposons de coupler un tel séquenceur avec une *API* générique et facilement utilisable par des applications diverses : temps réel, sensible aux pertes, etc. Le but de cette *API* est de limiter le travail de l'application à la génération du tampon trié par ordre d'importance décroissant.

7.4 DÉTAIL DU FONCTIONNEMENT DU SÉQUENCEUR

Maintenant que nous avons vu les généralités du séquenceur, il est intéressant de voir en détail comment est gérée la correspondance entre les données applicatives et les paquets ordonnancés par le contrôle de congestion *multicast*.

7.4.1 PARAMÈTRES DE L'*API*

Le séquenceur fournit une *API* simple d'utilisation, qui ne nécessite que 3 paramètres :

- Un tampon ($buffer$) de données encodées hiérarchiquement. Il est important de noter que le séquenceur n'a pas besoin de connaître le schéma d'encodage utilisé par l'application.
- Le temps alloué ($buffer\_time$) pour envoyer le tampon de données. Par exemple, si le tampon contient l'image d'une vidéo, alors $buffer\_time$ correspond certainement à la durée

de cette image.
Dans le cas d'un transfert de fichiers ou tout autre application où la valeur de *buffer_time* n'est pas triviale, cette dernière peut être inférée à partir de la hiérarchie utilisée par l'application. Comme il paraît souhaitable que chaque récepteur puisse au moins recevoir les données du premier niveau de la hiérarchie applicative, la valeur minimale pour *buffer_time* doit correspondre au temps nécessaire pour envoyer les données de ce premier niveau de la hiérarchie au débit minimal du contrôle de congestion [2].
Ainsi hormis pour le débit minimal, la valeur de *buffer_time* est indépendante des paramètres utilisés pour le contrôle de congestion, tel que le *TSD*. La seule limitation est que la valeur de *buffer_time* doit être supérieure au temps d'émission d'un paquet au débit minimal du contrôle de congestion. Le raisonnement sous-jacent à cette limitation est qu'un récepteur limité à la réception du débit minimal doit obtenir au moins un paquet par *buffer*.

- La taille (*buffer_length*) du tampon de données. Pour un codec vidéo, *buffer_length* dépend de la compression de chaque image. Pour un transfert de fichiers, la taille idéale pour *buffer_length* correspond à la quantité de données envoyée pendant *buffer_time* au débit maximal du contrôle de congestion.

En utilisant cette *API*, l'application est assurée qu'un récepteur qui reçoit *N*% du débit maximal du contrôle de congestion, reçoit alors les premiers *N*% du *buffer*.
Enfin, les mécanismes de contrôle de congestion *multicast* sont adaptés pour les flux de longue durée. Ainsi, une application est censée envoyer un nouveau *buffer* périodiquement toutes les *buffer_time* secondes.

7.4.2 Le séquenceur de paquets

Le séquenceur commence par récupérer la liste des paquets fournie par l'ordonnanceur du contrôle de congestion, pour les paquets dont l'envoi est programmé pendant la durée de *buffer_time* (cf. état initial de la figure 7.5). Cette liste contient les attributs suivants pour chaque paquet :

- La quantité de données applicatives contenues par ce paquet.
- Le groupe *multicast* auquel le paquet sera envoyé.
- La date d'émission future du paquet.
- Le débit cumulatif auquel le paquet sera envoyé.

2. Le débit minimal et maximal du contrôle de congestion sont définis par l'application à l'instanciation du contrôle de congestion.

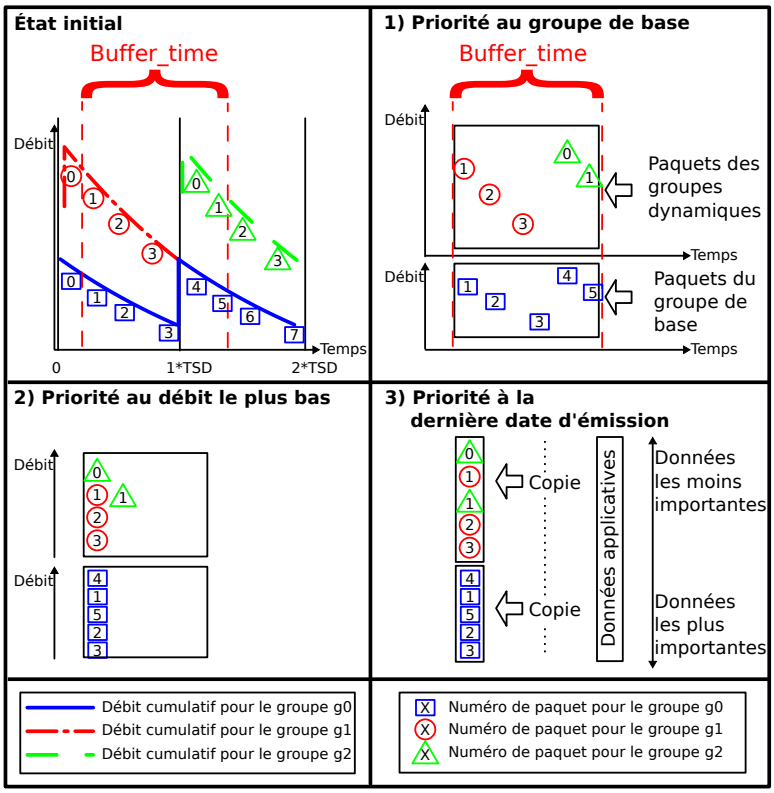


FIGURE 7.5 – Algorithme du séquenceur

Puis, le séquenceur calcule la répartition optimale pour envoyer les données les plus importantes au débit le plus faible possible. À cette fin, le séquenceur trie les paquets selon les 3 critères suivants (cf. état 1, 2 et 3 de la figure 7.5) :

1. *Priorité au groupe de base :*
   Les paquets du groupe de base sont prioritaires en comparaison aux autres groupes. Cela même si des paquets du groupe de base sont envoyés avec un débit supérieur à certains autres groupes. Cette priorité surpasse la règle donnant la priorité au débit le plus bas, car tous les récepteurs sans exception sont toujours abonnés au groupe de base.

2. *Priorité au débit le plus bas :*
   Ensuite, le séquenceur trie par débit les paquets du groupe de base, puis de tous les autres groupes. Cette règle est le coeur du séquenceur dont le but est d'envoyer les données les plus importantes au débit le plus faible possible.

3. *Priorité à la dernière date d'émission :*
   Si après les 2 premiers tris plusieurs paquets ont encore la même priorité, alors ces derniers sont triés par ordre décroissant de date d'émission : les données les plus importantes sont envoyées en dernier. Cette règle repose sur le fait qu'en

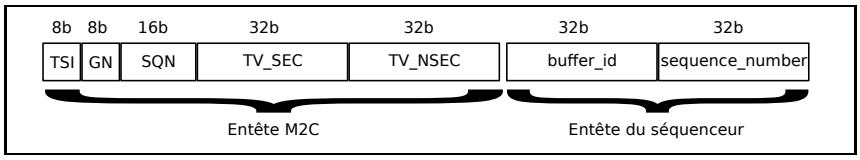


FIGURE 7.6 – En-tête des paquets *M2C* en utilisant le séquenceur

fonction du moment d'adhésion à un groupe, un récepteur peut ne recevoir que les derniers paquets de ce groupe.

Une fois les paquets triés, les données applicatives sont copiées dans ces paquets, ce qui garantit la diffusion au débit le plus faible possible des données les plus importantes. Finalement, l'ordonnanceur du contrôle de congestion envoie ces paquets aux dates d'émission calculées précédemment.

7.4.3 EN-TÊTE DU SÉQUENCEUR ET INTERFACE CÔTÉ RÉCEPTEUR

Pour émettre un buffer, le séquenceur marque chaque paquet d'un en-tête spécifique. Cet entête est composé (cf. figure 7.6) :

- D'un identifiant de tampon (*buffer_id*).
- D'un index indiquant la position dans le tampon des données transportées (*sequence_number*).

Le récepteur mémorise alors dans un tampon de réception toutes les données reçues pour un même identifiant de tampon. Dès qu'un paquet est reçu avec un identifiant de tampon différent, les données mémorisées dans le tampon de réception sont alors transférées à l'application. Pour récupérer ces données, l'application dispose d'une *API* fournissant 2 fonctions d'accès différentes :

- La première renvoie pour chaque tampon une liste contenant toutes les parties reçues.
- La seconde renvoie uniquement la première partie contiguë reçue commençant dès le début du tampon, c'est-à-dire un préfixe sans trou commençant à l'index 0.

7.5 ÉVALUATION DU SÉQUENCEUR

Pour évaluer les performances du séquenceur, il nous faut d'abord définir des critères d'évaluation ainsi qu'un ensemble de tests. Ces tests sont réalisés avec notre implémentation du séquenceur qui est incluse dans la librairie *MCC* [Lucas 10a] afin d'analyser le comportement du séquenceur au sein :

- D'un réseau idéal : sans perte.
- D'un réseau plus réaliste : avec des pertes générées par des flux concurrents partageant le même lien faible.

Cette évaluation ne fournit pas de comparaison avec des travaux précédents, parce que d'après nos connaissances aucune autre so-

lution traitant de ce problème n'a été proposée jusqu'à maintenant. En effet, les applications actuelles utilisant le *multicast* émettent en *UDP* à un débit constant. Cette situation ne peut mener qu'à deux résultats possibles :

- La bande passante disponible pour chaque récepteur est suffisamment grande pour contenir le débit du flux. Auquel cas, la séquence utilisée n'a aucun effet car toutes les données sont reçues.
- La bande passante disponible pour chaque récepteur n'est pas suffisante pour contenir le débit du flux. Auquel cas, le récepteur ne peut pas recevoir correctement le flux et par la même occasion des congestions importantes peuvent survenir et affecter les flux concurrents.

7.5.1 Critères d'évaluation du séquenceur

Deux critères sont définis afin d'évaluer les performances du séquenceur :

- Le nombre de données contiguës reçues, ou *Continuous Buffer Length* (CBL).
  Pour le tampon ($buffer$) courant, le CBL est calculé comme le ratio entre :
  - La taille en octets de la première partie contiguë commençant à l'index 0.
  - Le nombre total d'octets reçus.

  Par exemple, si un utilisateur reçoit les données correspondantes aux intervalles $[0-1000[$, $[2000-4000[$ et $[7000-14000[$, alors $CBL := 1000/10000$. Dans ce cas les applications, telles que certains clients vidéo qui nécessitent de recevoir les données de façon contiguë pour assurer le décodage, sont alors uniquement capable d'utiliser $\simeq$ 10% des données reçues.
- Le nombre de données éparses, ou *Sparse Buffer Length* (SBL).
  Pour le tampon ($buffer$) courant, le SBL est calculé comme le ratio entre :
  - Le nombre total d'octets reçus.
  - L'index le plus élevé parmi les données reçues.

  Pour le même exemple que précédemment, $SBL := 10000/14000$. Dans ce cas une application, telle qu'un logiciel de transfert de fichier qui sait utiliser toutes les données reçues même si ces dernières ne sont pas contiguës, a alors reçu les données les plus importantes avec une efficacité de $\simeq$ 71%.

7.5.2 Expérimentations sans pertes

Afin d'évaluer le comportement du séquenceur dans des conditions de réseau idéal, nous avons conduit une série de tests pour

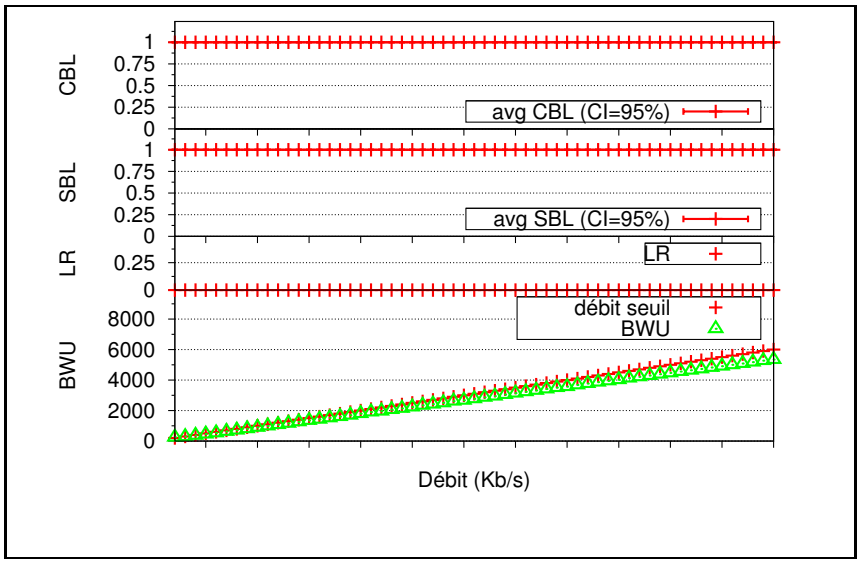


FIGURE 7.7 – Résultats de CBL et SBL avec des conditions idéales du réseau

lesquels aucun paquet n'a été perdu. Une source *M2C* est démarrée avec un débit minimal de 128 Kb/s, un débit maximal de 6 Mb/s et en utilisant une durée pour chaque tampon de $buffer\_time := 100$ ms. Un récepteur est alors démarré sur le même ordinateur afin de supprimer tout lien faible ainsi que le délai d'adhésion *multicast*. Cela supprime donc tous les biais que le réseau peut générer et permet d'obtenir des résultats correspondant uniquement aux performances du séquenceur.
Le récepteur est limité en débit par un seuil statique déterminé par l'application réceptrice qui limite ainsi l'évaluation du débit équitable (EFR) de *M2C*. Une série de tests est réalisée avec 58 débits seuils entre 200 Kb/s et 6 Mb/s avec un pas de 100 Kb/s, chaque test durant 600s.

Les résultats obtenus (cf. figure 7.7) montrent que les performances du séquenceur sont optimales : $CBL \simeq 1.0$ et $SBL \simeq 1.0$ avec une variation comprise entre 10^{-4} et 10^{-5} pour un intervalle de confiance de 95%. Les rares tampons reçus de façon sous-optimale n'apparaissent que lors de la phase de démarrage rapide (FS), quand le récepteur s'abonne à plusieurs groupes *multicast* pendant le même $buffer\_time$.
De plus, ces bonnes performances combinées avec la fine résolution des débits testés montrent que le débit des données utiles et utilisables est indépendant de la résolution grossière des débits des groupes *multicast*. En effet, ces tests montrent 58 débits reçus différents, alors que la source fournit une hiérarchie composée seulement de 14 groupes *multicast* diffusés simultanément.

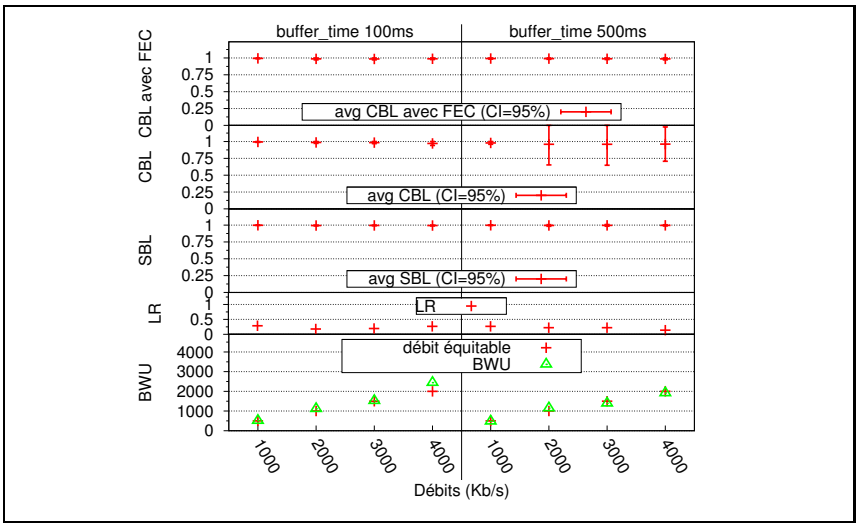


FIGURE 7.8 – Évaluation de CBL et de SBL avec des conditions réseaux réalistes.

7.5.3 EXPÉRIMENTATIONS AVEC PERTES

Afin d'évaluer le comportement du séquenceur avec des conditions réseaux réalistes, nous avons réutilisé la plateforme locale avec le scénario du "trafic en bruit de fond" ou pour les longs flux nous nous limitons au test ou un flux *TCP* est en concurrence avec un flux *M2C* utilisant le séquenceur pendant 10 minutes. Dorénavant, le récepteur *M2C* ne voit plus son débit limité par le niveau applicatif, ce qui va permettre aux flux *M2C* de produire des congestions. Les conditions du réseau sont obtenues selon les combinaisons des paramètres données par le tableau 7.1.

Paramètres	Valeurs
Débit du lien faible (Mb/s)	1, 2, 3 ou 4
Files d'attente (# paquets)	75 ou 100
TPA dans chaque sens (ms)	0, 20 ou 50
La durée d'émission du tampon de donnée ($buffer\_time$ en ms)	100 ou 500

TABLE 7.1 – Paramètres du scénario "Expérimentations du séquenceur avec pertes"

La figure 7.8 montre l'impact des pertes de paquets sur l'efficacité obtenue au niveau applicatif. En effet, comme les tests précédents ont prouvé que le séquenceur organise les données à envoyer de façon optimale, l'inefficacité calculable par $(1 - SBL)$ correspond au taux de pertes observé : le taux de perte varie entre 0.14% et 0.29%, alors que SBL varie entre 0.997 et 0.996. Ainsi, la présence de pertes n'a qu'un effet négligeable pour les applications capable

d'utiliser toutes les données reçues même si elles ne sont pas contiguës.
De plus, la moyenne de CBL est bonne (de 0.96 à 0.99) car la plupart du temps il n'y a pas de pertes pendant la durée correspondant à un $buffer\_time$. Cependant, CBL est plus sensible aux pertes du réseau car il considère pour un tampon que toutes les données reçues après la première perte sont inutilisables. Ainsi, même avec de rares phénomènes de congestions, la variation de CBL est importante (jusqu'à 0.31). En effet, ce phénomène s'explique notamment parce que les protocoles de contrôle de congestion pour le *multicast* réagissent plus lentement que *TCP* aux congestions et créent alors des rafales de pertes. Comme l'ordonnanceur de la source entrelace les paquets des différents groupes, ces pertes sont uniformément réparties sur l'ensemble du tampon de donnée ($buffer$), ce qui inclut le début de ce tampon. Ainsi dès lors que des pertes peuvent survenir sur le réseau, et ce même avec un séquenceur optimal, l'efficacité obtenue au niveau applicatif est vraiment très bonne la plupart du temps, mais les rares tampons reçus pendant une congestion sont quasiment inutilisables par les logiciels capable de ne gérer que les données reçues de façon contiguë. Ce problème est similaire à celui des pertes pour un flux vidéo : quand une perte survient, l'image est décodée avec des pixels erronés, où l'erreur est proportionelle à l'importance du coefficient perdu.

Afin de gérer ce problème, nous proposons d'utiliser un code *Forward Error Correction (FEC)* (cf. figure 7.9) basé sur l'algorithme de Reed Solomon [Rizzo 97] qui permet de générer n symboles à partir de k symboles source. L'avantage de ce type de mécanisme est que seuls k symboles distincts reçus parmi n sont suffisants pour retrouver les k symboles source. Nous avons donc amélioré l'API du séquenceur pour permettre à l'application de configurer ces paramètres k et n. Ainsi, chaque application peut choisir son niveau de redondance désiré ou aucune redondance si cela est inutile : par exemple si la redondance est gérée au niveau applicatif, ce qui peut être plus adapté à la hiérarchie utilisée pour encoder chaque tampon. Les résultats obtenus (cf. le haut de la figure 7.8) montrent qu'un taux de redondance de 10% permettant de corriger 2 pertes dans un bloc (avec $k = 18$ et $n = 20$) est suffisant pour améliorer significativement l'efficacité au niveau applicatif. Le pire résultat avec le code FEC montre une bonne moyenne pour le CBL de 0.98 et avec une variation de 0.01. En effet, le code FEC est extrêmement efficace parce que l'entrelacement effectué par la source pour les paquets des différents groupes se révèle alors être un avantage car il est vraiment peu probable de perdre plusieurs paquets pour des données consécutives d'un tampon. De plus, le temps de calcul pour le code Reed Solomon est limité par l'utili-

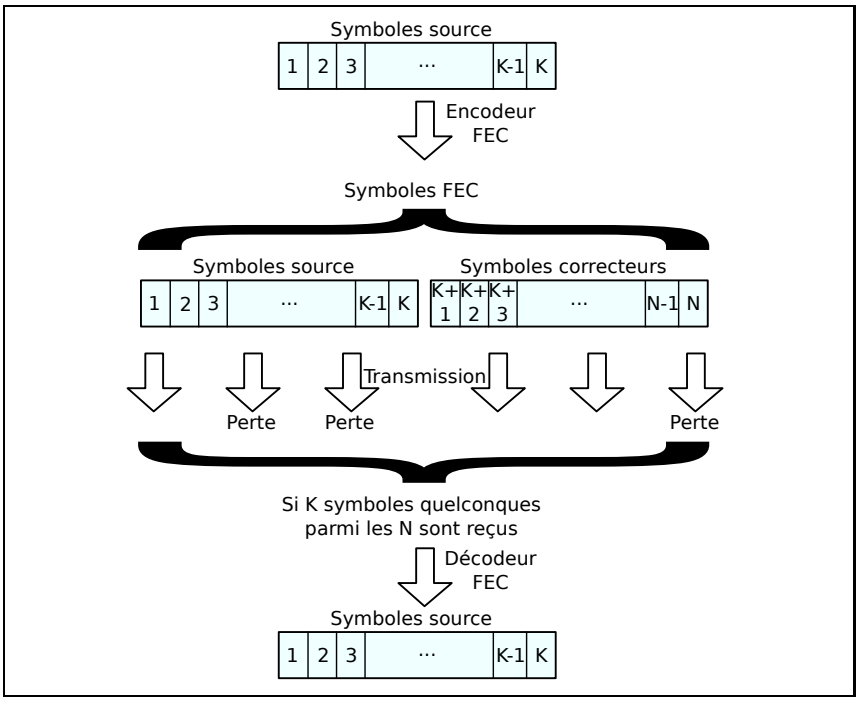


FIGURE 7.9 – Fonctionnement d'un code *FEC*

sation d'un nombre de symbole réduit [Rizzo 97], ce qui est le cas dans notre exemple avec $n = 20$.

7.6 CONCLUSION

Nous avons montré que les protocoles de contrôle de congestion *multicast* et notamment ceux utilisant des groupes à débits dynamiques sont très difficiles à utiliser :

- Comment la source doit-elle gérer l'envoi des données sur plusieurs groupes simultanément afin que chaque récepteur puisse ne recevoir que des données utiles et ce indépendamment du débit reçu ?

Nous avons donc proposé un séquenceur couplé à une *API* permettant d'utiliser simplement les protocoles de contrôle de congestion *multicast*. En effet, ce séquenceur permet à l'application de n'avoir qu'à générer un tampon de données ordonnées par importance. Puis le séquenceur se charge de faire concorder de façon optimale les données applicatives avec les données envoyées par les groupes *multicast*. Ainsi, le séquenceur assure que si un récepteur reçoit N% du débit de la source, alors ce récepteur reçoit les N% les plus importants des données envoyées. L'application n'a pas besoin de savoir comment fonctionne le protocole de contrôle et inversement le séquenceur n'a pas besoin de savoir qu'elle hiérarchie a été utilisée pour encoder les données au niveau appli-

catif.
Nous avons ensuite évalué ce séquenceur après avoir défini les critères d'évaluation ainsi que deux scenarii. Le premier montre le comportement optimal du séquenceur pour des conditions idéales du réseau. Cependant, le second montre que même avec un séquenceur optimal, l'apparition de pertes de paquets peut fortement réduire les performances des applications capables uniquement de gérer des données contiguës. Mais des expérimentations supplémentaires avec un taux de redondance de 10% permettant de corriger des rafales de 2 pertes est suffisant pour améliorer de façon significative l'efficacité pour de telles applications sur un réseau où des paquets peuvent être perdus.

8

UTILISATION PAR L'APPLICATION

Le séquenceur et son *API* permettent aux applications d'utiliser les protocoles de contrôle de congestion pour le *multicast* en ignorant les mécanismes employés au niveau transport. Cependant, il reste encore une contrainte que l'application doit remplir : générer des tampons de données triés par ordre d'importance.
Bien évidemment, plusieurs questions sous-jacentes se posent :

- Existe-t-il des applications capables de générer de tels tampons ? Si oui, est-ce que tout type d'application (sensible aux pertes, sensible aux délais) en est capable ?
- La hiérarchie utilisée pour générer les tampons est-elle assez fine pour permettre à chaque récepteur d'obtenir des données utiles correspondant au débit reçu ?
- La méthode d'encodage des données est-elle efficace ? L'encodage hiérarchique est-il moins performant qu'un encodage linéaire pour la même qualité ?

Dans ce chapitre, nous allons étudier 2 types d'applications qui peuvent tirer partie de cette *API*. Le but est que le *séquenceur* couplé avec l'*ordonnanceur au niveau transport* du mécanisme de contrôle de congestion *multicast* puissent être réutilisés pour diverses applications. Ainsi, ce couple contrôle de congestion et séquenceur doivent procurer des fonctionnalités similaires à celles de *TCP*, exceptée la fiabilité qui peut être gérée entièrement ou partiellement au niveau applicatif. Cette dernière peut être gérée par un *encodeur FEC* ainsi que par un mécanisme de retransmission cyclique appelé également boucle de carrousel, ou *carousel* [Peltotalo 07].

8.1 EXEMPLE D'APPLICATION : TRANSFERT DE FICHIERS

Pour montrer comment une application peut tirer parti du *séquenceur*, nous proposons d'étudier la création et les performances d'un transfert de fichiers. Cela permet de mettre en avant un exemple d'application sensible aux pertes.
Nous avons implémenté un logiciel permettant le transfert efficace de fichiers avec une granularité fine des débits utilisables. Ainsi,

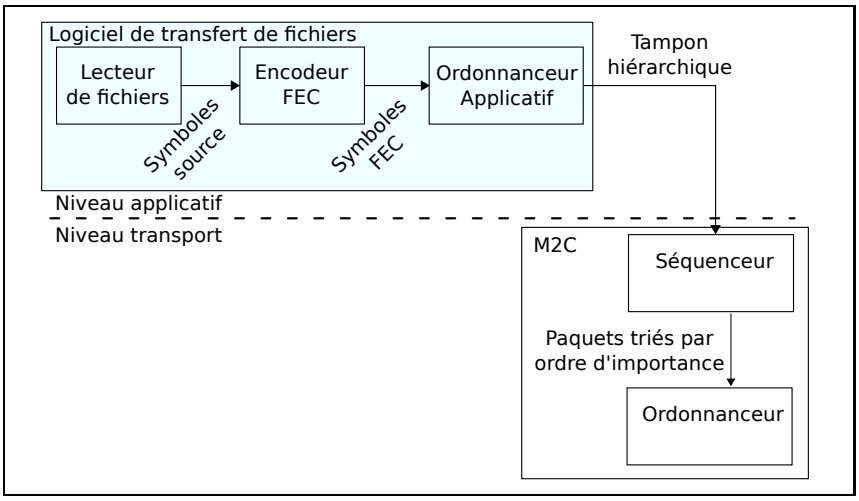


FIGURE 8.1 – Diagramme de flux du transfert de fichiers

nous voulons montrer qu'il est aisé d'utiliser l'*API* du *séquenceur* et que ce dernier permet de créer des applications capables de proposer une grande variété de débits qui permettent à chaque récepteur de s'adapter aux conditions du réseau. Ce logiciel utilise un *carousel* comme méthode cyclique de transmission des données. À chaque itération les données sont émises suivant le diagramme de la figure 8.1 :

- Le *Lecteur de fichiers* découpe un fichier en plusieurs parties appelées symboles sources, puis les fournit à l'*encodeur FEC*.
- L'*encodeur FEC* génère des symboles correcteurs à partir des symboles sources. Les symboles sources et correcteurs forment l'ensemble des symboles *FEC*.
- L'*ordonnanceur applicatif* organise les symboles *FEC* au sein de tampons hiérarchiques et les fournit à l'*API* du *séquenceur*.
- Le *séquenceur* assigne les données du tampon hiérarchique dans les paquets des différents groupes *multicast*.
- L'*ordonnanceur au niveau transport*, qui fait partie du mécanisme de contrôle de congestion *multicast*. Il organise l'émission des différents paquets pour chaque groupe *multicast*.

8.1.1 L'*ordonnanceur applicatif*

Le but ici est de trier les données d'un fichier par importance. Or pour la transmission d'un fichier, toutes les données doivent être reçues et par conséquent ont toutes la même importance.
L'*ordonnanceur applicatif*, dérivé de [Birk 03], trie B symboles dans des tampons composés de N niveaux hiérarchiques (cf. figure 8.2), de manière à maximiser la distance entre deux occurrences d'un symbole et à réduire le nombre de symboles redondant :

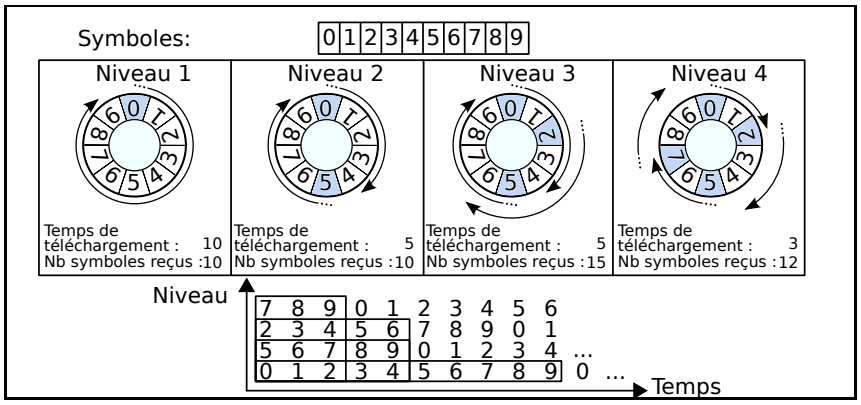


FIGURE 8.2 – Algorithme de l'*ordonnanceur applicatif*

- Au niveau 1, le symbole b est sélectionné pour le $b$ième tampon, avec $b \in [0...B[$.
- Au niveau n, le symbole $b + x$ est sélectionné pour le $b$ième tampon. Avec $n \in [2...N[$ et x le symbole à mi-distance du plus long intervalle entre deux symboles déjà sélectionnés aux niveaux inférieurs. Par exemple la figure 8.2 montre qu'au niveau 3 le symbole 2 est sélectionné, car il se situe à mi-distance entre le symbole 0 et le symbole 5, qui sont à une distance de 5.

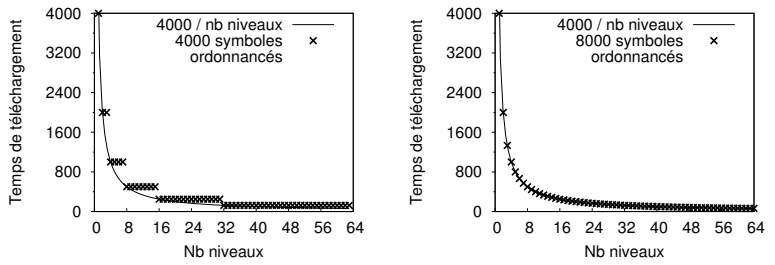
Ainsi, le temps de téléchargement est réduit de moitié chaque fois que le récepteur double le nombre de niveaux reçus (cf. figure 8.3(a)). De plus, cette propriété est indépendante du premier tampon b reçu.
Une granularité fine des débits est fournie en utilisant une taille de symbole correspondant à la taille des données utiles d'un paquet.
Bien entendu, les niveaux utilisés pour l'*ordonnanceur applicatif* sont totalement indépendants de la répartition en couche des groupes *multicast* au sein du protocole de contrôle de congestion.

8.1.1.1 L'*encodeur FEC*

Nous avons vu pour le séquenceur qu'un code *FEC* permet de générer n symboles à partir de k symboles sources. Le principal avantage est que seulement $k + \epsilon$ (avec ϵ petit) symboles distincts suffisent pour retrouver les K symboles sources.
Les codes où $\epsilon = 0$ sont appelés *Maximum Distance Separation (MDS)* codes, comme les codes Reed Solomon [Rizzo 97], ont une complexité et un temps de calcul non linéaire, ce qui limite le débit d'encodage/décodage pour un nombre de symboles important.
Tandis que les codes ou $\epsilon \neq 0$, dont les codes appelés *Low-Density Parity-Check (LDPC)*, repoussent cette limitation sur le nombre de

Niveau	Nombre de blocs envoyés	Nombre de blocs pas encore envoyés
1	N	0
2	$2 * (N/2)$	0
3	$3 * (N/4)$	$(N/4)$
4	$4 * (N/4)$	0
5	$5 * (N/8)$	$3 * (N/8)$
6	$6 * (N/8)$	$2 * (N/8)$
7	$7 * (N/8)$	$(N/8)$
8	$8 * (N/8)$	0
9	$9 * (N/16)$	$7 * (N/16)$
10	etc.	etc.

TABLE 8.1 – Blocs envoyés avant l'émission d'un doublon



(a) Sans code *FEC*

(b) Avec code *FEC* ($k = 4000$, $n = 8000$, $\epsilon = 0$)

FIGURE 8.3 – Ordonnancement de 4000 blocs

symboles utilisables et ont un débit d'encodage/décodage élevé. Notre logiciel de transfert de fichiers utilise un code *LDPC-triangle* [Roca 04] dont l'implémentation provient de la bibliothèque du projet "Planete-bcast" [INRIA 06].
Outre le fait de corriger des pertes de paquets, augmenter le nombre de symboles peut permettre de réduire le temps de téléchargement. En effet, l'analyse de l'*ordonnanceur applicatif* (cf. tableau 8.1) montre que le nombre de symboles qui n'ont pas encore été envoyés avant d'envoyer le premier doublon est $< N/2$. Ainsi, l'*ordonnanceur applicatif* peut-être amélioré avec $n = 2 * k$ symboles et permet de réduire le temps de téléchargement pour chaque nouveau niveau reçu (cf. figure 8.3(b)).

8.1.2 ÉVALUATION

Pour évaluer les performances du logiciel de transfert de fichier, nous allons effectuer deux scenarii :

- 1 transfert de fichiers avec 1 long flux concurrent.
- Plusieurs transferts de fichiers simultanés avec du trafic en bruit de fond.

Pour cela, nous allons utiliser la plateforme locale. Les conditions du réseau sont obtenues selon les combinaisons des paramètres données par le tableau 8.2.

Paramètres	Valeurs
Débit du lien faible (Mb/s)	1 ou 4
Files d'attente (# paquets)	25
TPA dans chaque sens (ms)	0

TABLE 8.2 – Paramètres du scénario "Expérimentations du transfert de fichiers"

De plus, comme les flux utilisant des protocoles de contrôle de congestion *multicast* ont une adaptation du débit relativement lente, ces flux sont plus aptes pour des longs transferts de fichiers. Ainsi, la taille des fichiers à transmettre ne doit pas être trop petite. C'est pourquoi, nous avons choisi 3 fichiers dont les tailles sont :

- 4 Mo pour un transfert d'une durée minimale de respectivement 32 et 8 secondes à 1 et 4 Mb/s. Pour ces durées, les flux *M2C* sont la plupart du temps en train de converger vers le débit équitable, ce qui permet d'étudier le comportement du transfert de fichier avec des variations importantes du débit.
- 48 Mo pour un transfert d'une durée minimale de respectivement 384 et 96 secondes à 1 et 4 Mb/s. Cela permet à *M2C* de passer une partie du test au niveau du débit équitable.
- 99 Mo pour un transfert d'une durée minimale de respectivement 792 et 198 secondes à 1 et 4 Mb/s. Cette taille de fichier est la taille maximale utilisable par nos machines sources et récepteurs, qui sont limitées par leur mémoire vive.

Les études sont faites entre un logiciel de transfert de fichiers en *unicast* utilisant le contrôle de congestion de *TCP new-RENO* avec l'option *SACK* activée, et notre implémentation [Lucas 09a] du logiciel de transfert de fichiers en *multicast*.

8.1.3 CRITÈRES D'ÉVALUATION

Les critères d'évaluation du logiciel de transfert de fichiers sont :

- *time* : le nombre de secondes (s) nécessaires pour télécharger et recalculer les symboles sources du fichier.
- *gput* : le débit utile (goodput) correspondant au débit applicatif pendant le téléchargement (Kb/s).

- $tput$: le débit du transfert utilisé (throughput) pendant le téléchargement (Kb/s).
- $loss$: Le pourcentage de pertes de paquets (%).
- dup : La surcharge de symboles dupliqués (%) entre le nombre total de symboles *FEC* reçus (*received_symbols*) et le nombre de symboles nécessaires par le code *FEC* pour recalculer les symboles source du fichier ($k + \epsilon$) :
  $dup := (received\_symbols/(k + \epsilon) - 1) * 100$.
  Comme le *séquenceur* et l'*ordonnanceur applicatif* fonctionne de façon optimale, en l'absence de perte dup doit être proche de 0.
- sym : La surcharge de symboles *FEC* (%) nécessaire pour le code *FEC*. C'est le rapport entre le nombre de symboles sources (k) et le nombre de symboles nécessaires au code *FEC* pour recalculer les symboles sources du fichier ($k + \epsilon$) :
  $sym := ((k + \epsilon)/k - 1) * 100$.
- *head* : La surcharge due aux en-têtes (%) entre la longueur des datagrammes ou des paquets (*packet_length*) et les données applicatives ($applicative\_data$) contenues par le même paquet :
  $head := (packet\_length/applicative\_data - 1) * 100$.
  Puisque *packet_length* est de 1480 et que les données applicatives *unicast* et *multicast* transportent respectivement 1460 et 1448 octets, *head* est une surcharge constante de :
  - 1.4% pour les flux *unicast*, ou 3.7% en comptant les entêtes Ethernet et *IP*.
  - 2.2% pour les flux *multicast*, ou 4.6% en comptant les entêtes Ethernet et *IP*.
- *net* : La surcharge totale du réseau (%). C'est le rapport entre la taille du fichier ($file\_length$) et le nombre de données reçues au niveau liaison ($link\_nb\_data$) :
  $net := (link\_nb\_data/file\_length - 1) * 100$.
  Ce critère représente une estimation générale de la surcharge totale générée par le réseau et inclut notamment les surcharges dup, sym et *head*.
- *comp* : Le temps de calcul pour retrouver les symboles sources du fichier une fois que le dernier symbole *FEC* nécessaire est reçu (%). C'est le rapport entre le critère *time* et le temps nécessaire au réseau pour recevoir le fichier (*network_time*) :
  $comp := (time/network\_time - 1) * 100$.
  Ce critère représente le temps nécessaire pour recalculer les derniers symboles sources dès qu'un nombre suffisant de symboles a été reçu.

Par la suite, les résultats de tous ces critères sont données en précisant la variation pour un intervalle de confiance de 95% calculé à partir de séries de 20 tests pour chaque expérience effectuée.

8.1.3.1 UN TRANSFERT DE FICHIER AVEC UN LONG FLUX CONCURRENT

Ce scénario montre le comportement d'un transfert de fichier (ft) en compétition avec un long flux concurrent :

- 1 *transfert de fichiers utilisant TCP* (TCP_ft) avec 1 flux *TCP* concurrent.
- 1 *transfert de fichiers utilisant M2C* ($M2C\_ft$) avec 1 flux *TCP* concurrent.
- 1 *transfert de fichiers utilisant M2C* ($M2C\_ft$) avec 1 autre flux *M2C* concurrent.

Dans tous ces tests, TCP_ft *net* $< 5\%$, principalement composé de 3.7% de surcharge due aux en-têtes.
La figure 8.4 montre pour le fichier de 99 Mo que $M2C\_ft$ *net* $\simeq$ 10%, principalement composé d'au moins 5.66% de *sym* et 4.6% de surcharge due aux en-têtes. Concernant le fichier de 4 Mo, $M2C\_ft$ *net* est compris entre 13 et 16%. Cette croissance de *net* est liée à l'augmentation des symboles *FEC* supplémentaires nécessaires ($sym \simeq 8\%$). Cela peut s'expliquer du fait que le nombre ϵ de symboles supplémentaires varie en fonction des symboles reçus : tous les symboles n'ont pas la même importance. De plus, pour de petits fichiers la variation de $tput$ est due à l'importance du temps de convergence de *M2C* pour rejoindre le débit équitable. Ce qui confirme que l'utilisation de $M2C\_ft$ n'est pas bien adaptée pour la diffusion de fichiers de petite taille.
En outre, les petites valeurs obtenues pour dup confirment le bon fonctionnement du séquenceur.
Les flux *TCP* et *M2C* utilisent un mécanisme de croissance additive et de réduction multiplicative, ou *Additive Increase and Multiplicative Decrease* (*AIMD*) [Yang oo], qui provoque des congestions de façon cyclique. Mais, les congestions de *M2C* provoquent plus de rafales de pertes qui sont dues aux variations soudaines de débit lors de l'adhésion à un nouveau groupe. Sans tenir compte du débit obtenu, le transfert de fichier n'est pas sensible à la distribution des pertes et est relativement performant avec 3% de pertes indépendantes, comme avec 10% de pertes en rafales. Finalement, la différence de temps de téléchargement est uniquement due au manque d'équité de *M2C* quand ce dernier est en compétition avec un autre flux *M2C*.
Ainsi, par sa conception $M2C\_ft$ propose une granularité fine de débits et les résultats obtenus montrent que son comportement n'est pas sensible aux différents débits du lien faible, capable alors de s'adapter à différentes valeurs de $tput$.

Fichier de 4 Mo			
	TCP_ft/TCP	*M2C_ft/TCP*	*M2C_ft/M2C*
	Lien faible à 1 Mb/s		
time(*s*)	79 (±10)	111 (±21)	125 (±62)
gput(*Kb*/*s*)	446 (±52)	318 (±54)	319 (±235)
tput(*Kb*/*s*)	468 (±54)	365 (±65)	372 (±280)
loss(%)	1.09 (±0.25)	2.84 (±0.61)	7.94 (±1.48)
net(%)	4.0 (±0.41)	13.6 (±4.22)	15.05 (±6.76)
dup(%)	N.A.	0.5 (±1.04)	0.77 (±0.92)
sym(%)	N.A.	7.95 (±3.8)	9.1 (±6.14)
comp(%)	N.A.	1.15 (±0.83)	1.16 (±0.93)
	Lien faible à 4 Mb/s		
time(*s*)	20 (±2)	48 (±15)	62 (±48)
gput(*Kb*/*s*)	1757 (±181)	767 (±290)	913 (±1139)
tput(*Kb*/*s*)	1909 (±194)	907 (±406)	1134 (±1679)
loss(%)	1.1 (±0.23)	2.46 (±0.81)	12.87 (±6.63)
net(%)	4.99 (±1.46)	14.68 (±5.09)	15.64 (±5.3)
dup(%)	N.A.	1.4 (±3.26)	1.8 (±3.06)
sym(%)	N.A.	7.65 (±2.81)	7.95 (±2.81)
comp(%)	N.A.	2.6 (±2.35)	3.79 (±7.1)

Fichier de 99 Mo			
	TCP_ft/TCP	*M2C_ft/TCP*	*M2C_ft/M2C*
	Lien faible à 1 Mb/s		
time(*s*)	1737 (±32)	1760 (±112)	2330 (±366)
gput(*Kb*/*s*)	477 (±9)	472 (±29)	358 (±50)
tput(*Kb*/*s*)	495 (±9)	519 (±16)	391 (±25)
loss(%)	0.81 (±0.04)	3.29 (±0.25)	6.6 (±0.45)
net(%)	3.71 (±0.01)	9.56 (±3.94)	9.48 (±21.41)
dup(%)	N.A.	0.08 (±0.05)	0.69 (±2.63)
sym(%)	N.A.	7.14 (±4.12)	8.18 (±18.45)
comp(%)	N.A.	0.45 (±0.14)	0.32 (±0.15)
	Lien faible à 4 Mb/s		
time(*s*)	434 (±6)	438 (±29)	766 (±55)
gput(*Kb*/*s*)	1910 (±26)	1896 (±122)	1090 (±76)
tput(*Kb*/*s*)	1986 (±26)	2134 (±136)	1223 (±93)
loss(%)	0.67 (±0.09)	3.47 (±0.64)	8.97 (±0.98)
net(%)	3.74 (±0.05)	10.58 (±0.55)	11.0 (±0.89)
dup(%)	N.A.	0.1 (±0.04)	0.52 (±0.71)
sym(%)	N.A.	5.66 (±0.53)	5.95 (±0.7)
comp(%)	N.A.	1.79 (±0.32)	1.05 (±0.26)

FIGURE 8.4 – Résultats du transfert de fichier avec un long flux concurrent

8.1.3.2 TRANSFERT DE FICHIERS AVEC DU BRUIT DE FOND ET DES RÉCEPTEURS MULTIPLES.

Ce scénario montre le bénéfice que peut procurer l'utilisation de $M2C\_ft$ lorsque plusieurs récepteurs téléchargent simultanément le même fichier dans un environnement plus réaliste : 1 ou 2 téléchargements simultanés sont effectués en présence de trafics en bruit de fond.
Les figures 8.5 et 8.6 montrent que pour chaque TCP_ft et $M2C\_ft$, *net* est semblable pour 1 ou 2 récepteurs. Cependant, le temps de téléchargement de TCP_ft double quasiment quand 2 récepteurs participent au transfert, alors que le temps de téléchargement de $M2C\_ft$ reste stable. Ainsi, $M2C\_ft$ est capable de tirer parti de la capacité du *multicast* à passer à l'échelle, alors que les flux *unicast* sont forcés de se partager la bande passante entre eux. Par conséquent, il suffit de 2 récepteurs pour rendre l'utilisation de $M2C\_ft$ plus avantageuse que celle de TCP_ft.
Pour confirmer que le bénéfice obtenu par l'utilisation de $M2C\_ft$ croît avec le nombre de récepteurs du même fichier, nous avons réalisé une autre série de tests avec 5 récepteurs pour le téléchargement du fichier de 48 Mo et toujours en présence de bruit de fond. Les résultats de la figure 8.6 confirment les bonnes performances de $M2C\_ft$. En effet, ces résultats montrent que $M2C\_ft$ est quasiment indépendant du nombre de récepteurs, alors que le temps de téléchargement pour *TCP* croît linéairement avec le nombre de récepteurs.

8.1.4 PROBLÈMES RESTANTS

Malgré les bons résultats obtenus, ce transfert de fichier est toujours limité pas la taille de la mémoire des machines sources et réceptrices. Ainsi, la question de l'utilisation d'un mécanisme d'*interleaving* pour se soustraire à cette limite est posée. En effet, l'utilisation d'un tel mécanisme permet de décomposer un fichier en blocs dont la taille limitée permet de réaliser le calcul des codes *FEC* en mémoire. Cependant, diviser un fichier en plusieurs blocs de redondance distincts peut détériorer les performances observées et notamment rendre les transferts de fichiers plus sensibles à certains schémas de pertes, par exemple :

- Des pertes en rafales doivent se répartir sur les différents blocs et ainsi ne pas réduire l'efficacité du transfert de fichiers.
- Des pertes cycliques peuvent provoquer une concentration importante de pertes répétitives sur un petit ensemble de blocs, ce qui détériorerait les performances du transfert de fichiers. En effet, les blocs ayant perdus le plus de paquets obligent l'application à continuer de recevoir tous les pa-

Fichier de 4 Mo				
	Lien faible à 1 Mb/s		Lien faible à 4 Mb/s	
	TCP_ft	*M2C_ft*	*TCP_ft*	*M2C_ft*
1 récepteur				
time(*s*)	38 (±1)	46 (±3)	10 (±1)	17 (±4)
gput(*Kb*/*s*)	913 (±19)	767 (±45)	3457 (±285)	2118 (±457)
tput(*Kb*/*s*)	972 (±3)	899 (±47)	3948 (±20)	2688 (±744)
loss(%)	0.76 (±0.09)	9.42 (±2.8)	0.65 (±0.03)	3.24 (±2.72)
net(%)	4.12 (±0.47)	13.81 (±1.92)	7.85 (±0.55)	17.52 (±7.23)
dup(%)	N.A.	0.24 (±0.25)	N.A.	2.09 (±5.09)
sym(%)	N.A.	7.82 (±1.65)	N.A.	8.9 (±2.83)
comp(%)	N.A.	3.07 (±1.75)	N.A.	7.81 (±6.33)
2 récepteurs				
time(*s*)	73 (±5)	45 (±3)	19 (±2)	17 (±9)
gput(*Kb*/*s*)	477 (±36)	771 (±59)	1879 (±173)	2083 (±741)
tput(*Kb*/*s*)	502 (±38)	899 (±52)	2029 (±179)	2737 (±1078)
loss(%)	1.16 (±0.23)	8.19 (±2.82)	1.08 (±0.31)	8.89 (±7.81)
net(%)	4.04 (±0.85)	13.16 (±2.37)	4.1 (±2.61)	17.76 (±7.3)
dup(%)	N.A.	0.23 (±0.42)	N.A.	1.64 (±3.88)
sym(%)	N.A.	7.23 (±1.55)	N.A.	9.61 (±3.1)
comp(%)	N.A.	3.04 (±1.66)	N.A.	11.4 (±8.51)

Fichier de 99 Mo				
	Lien faible à 1 Mb/s		Lien faible à 4 Mb/s	
	TCP_ft	*M2C_ft*	*TCP_ft*	*M2C_ft*
1 récepteur				
time(*s*)	876 (±3)	948 (±15)	218 (±1)	253 (±8)
gput(*Kb*/*s*)	947 (±3)	875 (±14)	3808 (±13)	3282 (±106)
tput(*Kb*/*s*)	983 (±3)	971 (±4)	3967 (±5)	3743 (±131)
loss(%)	2.56 (±2.32)	7.25 (±0.48)	0.23 (±0.0)	7.7 (±2.59)
net(%)	3.73 (±0.03)	10.15 (±1.28)	3.79 (±0.02)	10.73 (±0.5)
dup(%)	N.A.	0.2 (±0.04)	N.A.	0.0 (±0.0)
sym(%)	N.A.	5.78 (±1.12)	N.A.	5.78 (±0.36)
comp(%)	N.A.	0.77 (±0.16)	N.A.	2.98 (±0.45)
2 récepteurs				
time(*s*)	1733 (±25)	976 (±66)	432 (±7)	262 (±11)
gput(*Kb*/*s*)	479 (±7)	851 (±55)	1921 (±30)	3172 (±135)
tput(*Kb*/*s*)	496 (±7)	964 (±48)	1996 (±35)	3697 (±213)
loss(%)	0.88 (±0.05)	7.18 (±1.13)	0.82 (±0.04)	8.42 (±1.92)
net(%)	3.71 (±0.04)	10.34 (±1.13)	3.74 (±0.12)	10.81 (±0.48)
dup(%)	N.A.	0.2 (±0.08)	N.A.	0.0 (±0.0)
sym(%)	N.A.	6.02 (±1.01)	N.A.	5.84 (±0.42)
comp(%)	N.A.	2.84 (±5.62)	N.A.	5.18 (±2.28)

FIGURE 8.5 – Résultats du transfert de fichier de 4 Mo et de 99 Mo avec 1 ou 2 récepteurs et du bruit de fond

Fichier de 48 Mo				
	Lien faible à 1 Mb/s		Lien faible à 4 Mb/s	
	TCP_ft	$M2C\_ft$	TCP_ft	$M2C\_ft$
1 récepteur				
$time(s)$	419 (±2)	457 (±8)	104 (±1)	124 (±5)
$gput(Kb/s)$	941 (±4)	863 (±15)	3793 (±35)	3195 (±139)
$tput(Kb/s)$	978 (±3)	964 (±8)	3967 (±6)	3708 (±154)
$loss(\%)$	0.29 (±0.03)	7.76 (±1.65)	0.26 (±0.01)	8.38 (±6.08)
$net(\%)$	3.76 (±0.05)	10.69 (±1.65)	3.96 (±0.17)	12.35 (±1.88)
$dup(\%)$	N.A.	0.22 (±0.22)	N.A.	0.06 (±0.06)
$sym(\%)$	N.A.	5.84 (±1.6)	N.A.	7.14 (±1.51)
$comp(\%)$	N.A.	0.91 (±0.21)	N.A.	3.45 (±0.71)
2 récepteurs				
$time(s)$	827 (±16)	462 (±23)	206 (±4)	127 (±7)
$gput(Kb/s)$	477 (±9)	853 (±40)	1913 (±39)	3108 (±165)
$tput(Kb/s)$	495 (±10)	960 (±45)	1992 (±44)	3643 (±243)
$loss(\%)$	0.92 (±0.06)	7.75 (±1.28)	0.85 (±0.07)	13.26 (±52.51)
$net(\%)$	3.73 (±0.08)	10.82 (±1.06)	3.78 (±0.28)	11.33 (±0.93)
$dup(\%)$	N.A.	0.2 (±0.1)	N.A.	0.0 (±0.0)
$sym(\%)$	N.A.	6.01 (±1.11)	N.A.	6.21 (±0.82)
$comp(\%)$	N.A.	1.55 (±0.68)	N.A.	5.28 (±2.45)
5 récepteurs				
$time(s)$	2055 (±30)	495 (±65)	512 (±8)	146 (±28)
$gput(Kb/s)$	192 (±3)	801 (±97)	770 (±13)	2733 (±467)
$tput(Kb/s)$	199 (±3)	921 (±129)	799 (±13)	3360 (±754)
$loss(\%)$	5.86 (±1.85)	9.86 (±9.5)	4.03 (±0.11)	8.17 (±3.36)
$net(\%)$	3.71 (±0.06)	11.64 (±7.27)	3.68 (±0.26)	11.38 (±1.41)
$dup(\%)$	N.A.	0.51 (±1.29)	N.A.	0.21 (±1.08)
$sym(\%)$	N.A.	6.49 (±1.79)	N.A.	6.06 (±0.93)
$comp(\%)$	N.A.	3.04 (±2.67)	N.A.	10.17 (±7.61)

FIGURE 8.6 – Résultats du transfert de fichier avec 1, 2 et 5 récepteurs et du bruit de fond pour le fichier de 48 Mo

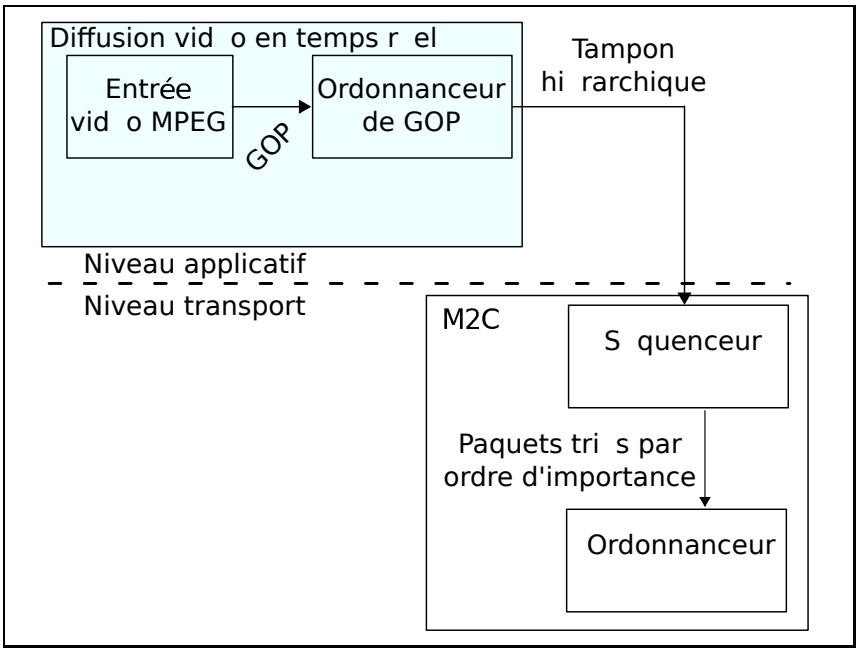


FIGURE 8.7 – Diagramme de flux pour la vidéo en différé

quets envoyés par la source y compris ceux des blocs déjà complets.

8.2 EXEMPLE D'APPLICATION : DIFFUSION VIDÉO

Le second type d'utilisation concerne la diffusion de vidéos, qui est une application sensible aux délais, mais ne nécessitant pas forcément de recevoir l'intégralité des données. Ainsi, il ne s'agit plus de trouver un moyen de transférer toutes les données dans un temps minimal, mais de transférer un maximum de données dans un temps limité. En effet, comme une donnée ne reste valide que pendant un temps limité, la priorité des données ne peut donc plus être en fonction du temps. Cependant, pour une vidéo les données peuvent être triées par importance de coefficients : par exemple en calculant les valeurs représentant un grand nombre de pixels avec une proximité spatiale (au sein de l'image) ou temporelle (entre plusieurs images successives).
Dans cette partie, nous allons nous intéresser à deux types d'encodage vidéo adaptés à être utiliser avec le séquenceur.

8.2.1 UTILISATION POUR LES TRANSFERTS DE VIDÉO EN DIFFÉRÉ

La diffusion de vidéo en différé permet d'illustrer comment utiliser le *séquenceur* sans pour autant demander à l'application de créer un système complexe pour hiérarchiser ses données. Cette

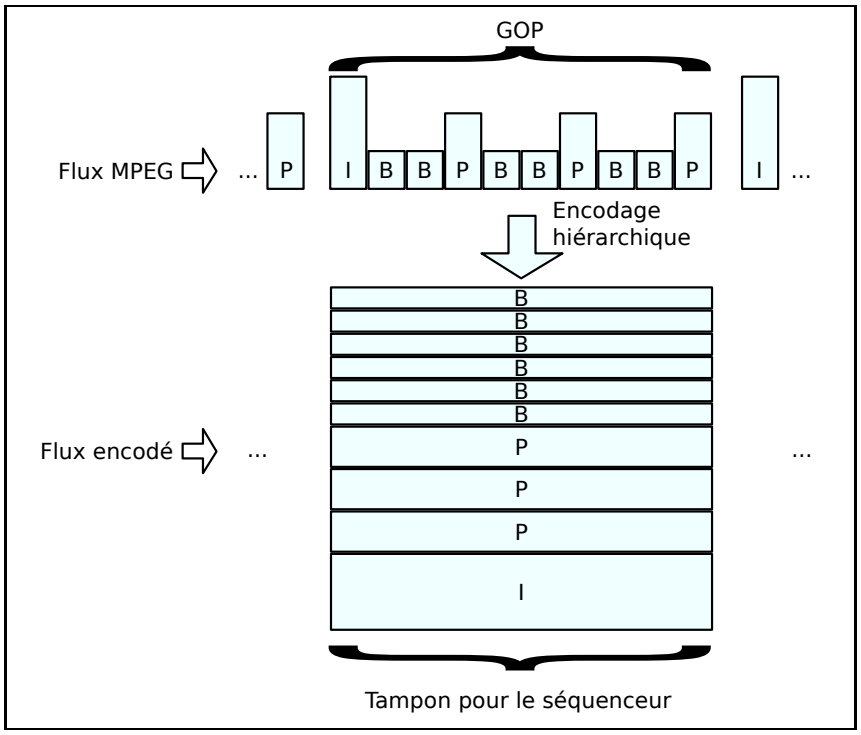


FIGURE 8.8 – Mécanisme de l'ordonnanceur de *GOP*

application (cf. figure 8.7) permet de diffuser un flux vidéo avec pour seule contrainte de retarder de quelques secondes les images envoyer. Ce type de logiciel est adapté pour des flux qui ne sont pas interactifs tels que les flux *IPTV*.
En différant la diffusion de quelques secondes, nous pouvons encoder la vidéo en rassemblant les images en groupes ou *Group Of Pictures (GOP)*. Un *GOP* contient généralement une image de référence (image de type *I*) ainsi que plusieurs images relatives (de type *B* ou *P*).
Ainsi, un *GOP* peut-être vu comme étant une hiérarchie temporelle de la vidéo. De cette constatation découle directement l'algorithme suivant pour encoder un *GOP* donné :

- Les informations de l'image de type *I* sont placées en priorité dans le tampon et sont ensuite suivies par ordre d'importance par les différentes images *P* puis des images *B*.
- Le tampon ainsi constitué peut alors être envoyé par le séquenceur pendant toute la durée du *GOP* originel. Bien sûr, il faut s'assurer au moment de créer la session que le débit du groupe de base est suffisant pour envoyer une trame *I* entière pendant la durée d'un *GOP*.

Ainsi, cette implémentation simple consiste à envoyer les données de façon hiérarchique sans avoir à recalculer un encodage spécifique pour la vidéo. En effet, dans cet algorithme la compression des données n'est pas modifiée et seule la présentation des dif-

férentes images a été changée. Ainsi, le taux de compression de la vidéo n'est pas altéré et de plus le débit qui reste variable est néanmoins lissé car la différence entre 2 *GOP*s est moindre que la différence entre une image *I* et une image *B* ou *P*. Bien entendu, la restriction de cette méthode est que l'envoi des données est alors différé d'une durée minimale de 2 *GOP*s entiers.
Par exemple pour un *DVD*, un *GOP* peut contenir jusqu'à 15 images en *PAL* ou 18 en *NTSC*. Ce qui fait au moins 15 résolutions différentes correspondant à 15 débits différents utilisables par les récepteurs. On peut constater que contrairement au transfert de fichiers, ce type d'encodage ne permet pas aux récepteurs d'avoir une granularité fine en terme de débit applicatif. Si cette hiérarchie n'est pas assez fine, il est toujours possible de combiner 2 *GOP*s (ou plus) dans le même tampon à fournir au séquenceur, ce qui permet de doubler le nombre de résolution différentes en différant 2 fois plus l'envoi de la vidéo. Sans connaître les capacités des différents récepteurs, le choix du nombre de couches à transmettre peut se révéler être un problème complexe. Par contre, une connaissance même partielle des capacités des récepteurs peut suffire pour utiliser des techniques d'optimisation des débits pour les couches applicatives [Soldani 09].
D'un autre point de vue si l'objectif est de créer un logiciel de diffusion en temps réel avec un débit de 30 images par seconde (soit 33 ms par image), il faudrait alors limiter le *GOP* à 2 images pour conserver une dynamique temps réel (délai de $2 * GOP < 150$ ms) et seulement 1 image dans le cas d'une vidéo à 25 images par secondes. Cela limiterait le nombre de hiérarchie à respectivement 2 ou 1 niveaux et rendrait l'utilisation du protocole de contrôle de congestion inutile.

8.2.2 UTILISATION POUR LES TRANSFERTS DE VIDÉO TEMPS-RÉEL

Nous avons vu que le premier logiciel de vidéo n'est pas adaptée pour effectuer de la diffusion en temps réel. Par conséquent, nous proposons d'utiliser un second type d'encodage plus complexe qui permet une utilisation en temps-réel, pour par exemple réaliser des vidéos-conférences. Ainsi, le temps d'encodage et de transmission des données doit être inférieur au seuil humainement tolérable ($\simeq$ 150 ms) pour pouvoir effectuer une communication sans ressentir ce délai comme une gêne trop importante.
Ainsi, une solution proposée est d'utiliser un codage hiérarchique pour chaque image envoyée (cf. figure 8.9). Pour cela, nous allons nous appuyer sur l'algorithme du codec *Progressive Video with Hybrid transform* (*PVH*) [McCanne 97] [McCanne 96a] qui a été conçu à l'origine pour être utilisé avec le protocole de contrôle de con-

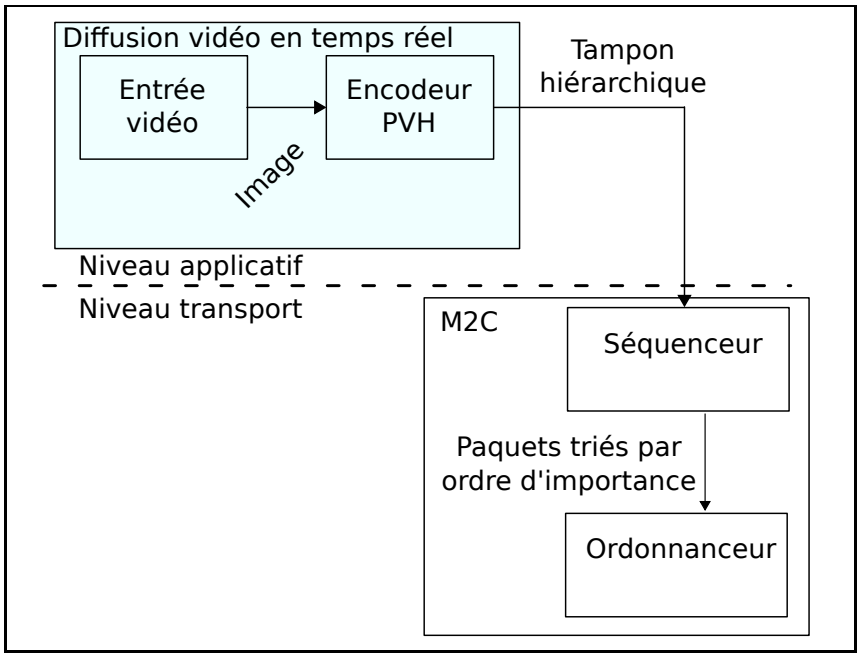


FIGURE 8.9 – Diagramme de flux pour la vidéo en temps réel

gestion à débit statique *RLM* et a été implémenté pour le logiciel *Videoconferencing Tool* (*VIC*). Le processus d'encodage de chaque image suit les étapes suivantes :

1. *Sélection des blocs à encoder.*
   Chaque image est divisée en blocs de $16 * 16$ pixels. Pour chaque bloc, 2 lignes de la composante de luminance sont comparées avec les dernières valeurs sélectionnées pour ce bloc. Si la différence entre ces lignes est trop importante, ou si la date de rafraîchissement du bloc est trop ancienne, alors le bloc est sélectionné pour être encodé et envoyé aux récepteurs.
   Les informations sur les chrominances des blocs sélectionnés sont alors simplement sous échantillonnées de moitié et seules les informations de luminance sont compressées par l'algorithme suivant.
2. *Compression par ondelettes.*
   La luminance pour chaque bloc sélectionné est alors décomposée par des filtres passe-haut et passe-bas en 2 passes : l'une horizontale et l'autre verticale. Ainsi, 4 décompositions fréquentielles sont obtenues :
   - la plus grossière mais contenant la plus forte densité d'information (nommée LL),
   - celle contenant les détails horizontaux (HL),
   - les détails verticaux (LH)
   - et les détails diagonaux (HH) qui ne seront pas envoyés aux récepteurs.

Par conséquent, après cette étape la luminance est déjà décomposée en 3 couches hiérarchiques.

3. *Compression par Discrete Consinus Transform (DCT)*
   Ensuite la couche LL et les informations de chrominances qui ont alors une taille de $8 * 8$ pixels subissent une nouvelle transformation par cosinus discret permettant d'obtenir 64 coefficients dont le premier est la moyenne des 64 pixels et les suivants contiennent les informations de façon hiérarchique sur les détails de la couche compressée.
   Avec les couches LH et HL, ces transformations ont permis de créer 66 couches hiérarchiques pour la luminance et 64 pour chaque composante de chrominance dont chaque nouvelle couche améliore la finesse de définition de l'image.
4. *Quantification*
   Une fois encodée, les données obtenues après la DCT sont quantifiées avec une matrice approximant d'avantages les coefficients les moins importants. C'est à ce moment là que les données sont perdues, mais c'est également cette étape qui permet de réellement compresser les données.
5. *Codage par Huffman et arbre quaternaire (quad-tree)*
   Finalement, les données sont encodées avec l'algorithme de *Huffman* pour les données de chrominance et pour LL. Alors que les données pour LH et HL, qui contiennent une majorité de zéros, sont encodées par arbre quaternaire.

Il est à noter que ce codec ne comprend pas de réelle compression temporelle, ce qui permet une réelle utilisation en temps réel mais limite son taux de compression comparativement à des codecs comme ceux utilisés dans la norme *MPEG*.

8.2.3 Conclusion et perspectives

Nous avons implémenté l'algorithme de PVH au sein d'un logiciel de diffusion vidéo [Lucas 09a], mais malheureusement l'évaluation des performances de ce logiciel n'a pas encore été réalisée. Cependant, il est certain que cette évaluation va devoir répondre à plusieurs interrogations :

- *Quelles sont les métriques d"évaluation ?*
  Généralement pour évaluer une vidéo on utilise la rapport signal sur bruit, ou *Peak Signal-to-Noise Ratio* ($PSNR$). Cependant, cette métrique qui donne une estimation moyenne de la qualité reçue n'est pas forcément représentative de la perception de l'oeil humain. Ainsi, une autre métrique basée sur la similarité structurelle, ou *Structural Similarity* ($SSIM$) [Wang 04] a été proposée. Cette méthode part du principe que l'oeil humain est plus sensible aux variations

qui perturbent la définition du contour d'un objet. Ainsi, les bordures d'objets sont considérées comme étant les zones pour lesquelles la variation de luminance est importante.

- *Quel est le taux de codage de redondance à utiliser ?*
  Contrairement au transfert de fichier, la transmission vidéo implique que l'encodage de la redondance crée un surplus de données reçues s'il n'y a pas de pertes. Ainsi le choix du taux de redondance va avoir un impact certain sur l'efficacité de l'application de diffusion de vidéos.
  Un premier choix peut se limiter à ne pas ajouter d'informations de redondance, en se basant sur l'idée qu'une perte n'affecte qu'une image et seulement les couches supérieures du paquet perdu. Cependant, l'évaluation du séquenceur à montrer que le CBL peut être très variant car les pertes dues aux flux *multicast* sont souvent provoquées en rafales. Ce qui dans notre cas impacterait plusieurs images successives et avec de fortes probabilités de perte dès la couche de base de la vidéo.
  Une redondance est nécessaire, mais le choix du taux ne doit pas forcément correspondre aux 10% évaluées pour le séquenceur. Ainsi, il est envisageable de définir au niveau applicatif un taux de redondance pour chaque couche hiérarchique de façon à mieux protéger les données importantes tout en limitant le taux de redondance pour les récepteurs à débit élevé.

CONCLUSION GÉNÉRALE ET PERSPECTIVES

CONCLUSION GÉNÉRALE

L'architecture *multicast IP* est bien définie au niveau routage, mais l'absence de mécanismes de contrôle de congestion au niveau transport la rend encore difficilement utilisable dans l'*Internet* général. Cette thèse montre que les communications de groupe sont possibles à grande échelle tout en permettant à chaque récepteur d'obtenir un débit optimal et équitable avec les autres flux *multicast* et *TCP*.
Pour cela, nous avons défini une méthodologie d'évaluation, expérimenté, conçu et amélioré des protocoles de contrôle des congestion pour le *multicast*, avant de montrer comment des applications peuvent tirer partie de tels mécanismes. Les nombreux résultats présentés ont été obtenus par des expérimentations reposant sur une implémentation effective de plusieurs protocoles de contrôle de congestion.

MÉTHODOLOGIE POUR L'ÉVALUATION DES PROTOCOLES DE CONTRÔLE DE CONGESTION *multicast*

Nous avons tout d'abord mis en place une méthodologie permettant de comparer et d'évaluer les comportements de différents protocoles de contrôle de congestion pour le *multicast*. Ainsi, cette méthodologie nécessite en premier lieu la définition de métriques afin de pouvoir évaluer les différentes caractéristiques d'un mécanisme de contrôle de congestion. La plupart de ces métriques sont issues de réflexions effectuées à l'*IETF* sur les méthodes d'évaluation des protocoles de congestion en *unicast*. Cependant, nous les avons complétées en prenant en compte les spécificités des flux *multicast*. Puis, nous avons défini une liste de scenarii permettant :

- de mettre en valeur une métrique particulière,
- d'obtenir des résultats dans un environnement réaliste et éprouvant pour les protocoles testés.

Enfin, différentes plateformes ont été mises en place afin :

- de supprimer les différents biais qui peuvent être causés par des flux ou des événements extérieurs,
- ou au contraire de passer par des routeurs utilisés en production dans *Internet* qui sont traversés par des flux concurrents non contrôlés, ce qui accentue le réalisme des expériences.

ANALYSE ET CONCEPTION D'UN PROTOCOLE ÉQUITABLE ET ROBUSTE

Après avoir étudié l'état de l'art des protocoles de contrôle de congestion pour le *multicast IP*, nous avons analysé en détail les comportements des différents protocoles proposés et en particulier celui de *WEBRC*. Le premier résultat de cette étude est la mise en évidence que le temps d'adhésion à un groupe *multicast* est très variable et n'est pas négligeable. Cette spécificité influe fortement sur le comportement des protocoles utilisant des groupes à débit dynamique. Ainsi les protocoles comme *FLID-DL* ou *WEBRC* sont fortement impactés par :

- L'impossibilité de s'abonner au premier groupe dynamique si le temps d'abonnement est supérieur au *TSD*.
- L'utilisation inadéquate du temps d'adhésion comme estimation du *RTT*.

En utilisant des environnements exempts de ces problèmes, nous avons continué notre analyse du comportement de *WEBRC* et ainsi montré que ce dernier est équitable dans des environnements stables, mais devient rapidement fluctuant quand le nombre de flux concurrents augmente ou quand des flux apparaissent de façon aléatoire.
Fort de ce constat, nous avons proposé une série de protocoles se basant sur la source de *WEBRC* avec cependant une adaptation permettant d'accélérer la réduction dynamique du débit des groupes *multicast* et améliorant de façon incrémentale le récepteur pour, au fur et à mesure, apporter au protocole de l'équité, une rapidité de convergence, de la stabilité et un démarrage accéléré. Ce protocole appelé *M2C* se base principalement sur les mécanismes de *TCP*. *M2C* a également été testé et validé dans de multiples situations qui ont mis en évidence sa robustesse et son comportement équitable envers les autres flux *M2C* et les flux *TCP*. Bien sûr ce protocole n'est pas parfait car, tout comme *TCP*, il est par exemple inéquitable envers des flux avec une grande différence de *RTT*. Cependant, l'équité de *M2C* sur le long terme peut être considérée comme ayant des performances comparables à celles de *TCP*. Enfin *M2C* a une convergence beaucoup plus lente que *TCP*, ce qui réserve son utilisation à des flux de longue durée.

INTERFACE AVEC LE NIVEAU APPLICATIF

Indubitablement, il ne suffit pas à un protocole d'être efficace pour être utilisable et, qui plus est, utilisé. En effet, la manière d'émettre les données au niveau de la source pose un double problème pour l'application utilisant un protocole de contrôle de congestion pour le *multicast* :

- Le premier problème vient du fait qu'avec un flux de données unique émis par la source, il est difficile d'organiser les données de ce flux pour que chaque récepteur puisse utiliser un maximum voir toutes les données reçues, alors que chaque récepteur reçoit un débit qui lui est propre et ce indépendemment des autres récepteurs.
- Le second problème vient de l'utilisation des groupes à débit dynamique, car l'application source doit connaître tout le processus d'ordonnancement effectué par le protocole de contrôle de congestion pour pouvoir calculer le débit instantané de chaque groupe et ainsi pouvoir décider comment répartir les données entre les groupes.

Pour répondre à ces problématiques, nous avons proposé une solution permettant à la fois que les données reçues lors de ces communications soient réellement utiles et utilisables et que l'application source puisse utiliser les protocoles de contrôle de congestion sans avoir à en connaître les mécanismes. Cette solution couple à la fois un séquenceur et une *API* qui nécessite de la part de l'application le passage en paramètre d'un tampon de données trié par ordre d'importance et ceci sans autre contrainte. Ainsi, l'application n'a pas besoin de savoir comment fonctionnent les groupes à débit dynamique et inversement le séquenceur n'a pas besoin de connaître la hiérarchie utilisée pour le tampon de données de l'application source.
Quant au séquenceur, nous avons montré qu'il distribue les données de l'application dans les différents groupes *multicast* de façon optimale. Ainsi, un récepteur abonné à N% du débit de la source, reçoit alors les N% les plus importants des données applicatives.
Plusieurs types d'applications peuvent en tirer partie. En effet, nous avons montré que :

- d'une part, les applications comme celles utilisant des encodages vidéos hiérarchiques sont capables d'utiliser l'*API* proposée sans nécessiter d'adaptation.
- d'autre part, cette *API* est également utilisable par des applications de transfert de fichiers, où il n'existe pas de hiérarchie intuitive permettant de donner une importance aux données. Dans ce cas, nous avons montré qu'il existe une priorité relative au temps d'attente pour obtenir la donnée. Par exemple, réduire le temps d'un transfert de fichiers si-

gnifie réduire le temps nécessaire pour recevoir toutes les données. Ainsi, la donnée dont la date d'émission est la plus lointaine jusqu'à un certain niveau de la hiérarchie devient la donnée la plus importante pour le niveau suivant de la hiérarchie.

IMPLÉMENTATION DES PROTOCOLES ET APPLICATIONS

Enfin, une contribution importante de cette thèse est l'implémentation des derniers protocoles de l'état de l'art (*FLID-SL*, *FLID-DL* et *WEBRC*) ainsi que *M2C*. Cette implémentation est téléchargeable sous forme d'une bibliothèque [Lucas 10] dont les sources sont disponibles sous licence libre. Cette bibliothèque est à notre connaissance la seule actuellement disponible publiquement permettant d'utiliser des protocoles *multicast* à canaux dynamiques.
De plus, un logiciel expérimental de diffusion de vidéos [Lucas 09a] et un logiciel expérimental de transfert de fichiers [Lucas 09b] sont également disponibles sous licence libre permettant d'étudier et de prouver par leurs réalisations la faisabilité du *multicast* avec des mécanismes permettant un contrôle de congestion et une efficacité applicative supportant le passage à l'échelle.

PERSPECTIVES

Ce travail ouvre des perspectives sur plusieurs domaines dont l'évaluation des performances au niveau applicatif, l'utilisation dans un environnement sans fil et l'étude des mécanismes de routage en cœur de réseau.

UTILISATION AU NIVEAU APPLICATIF

Les premières perspectives sont d'évaluer les performances des applications proposées qui utilisent des mécanismes de contrôle de congestion pour le *multicast*. Ainsi, à court terme nos recherches permettront l'évaluation :

- Des 2 types d'applications vidéos proposées. Cette évaluation portera entre autre sur le rapport entre la qualité de l'image et le débit utilisé, la rapidité de zapping et la capacité du flux à maintenir une résolution donnée face aux variations du débit équitable.
- Du transfert de fichiers en utilisant un mécanisme d'interleaving afin de supprimer la dépendance entre la taille de la mémoire vive de l'ordinateur et la limite de la taille des fichiers transférables.

COMMUNICATIONS SANS FIL

Dans cette thèse, nous nous sommes intéressés aux échanges sur des réseaux filaires. Or, la propriété du *multicast* à réduire le nombre de copies inutiles d'un même paquet peut s'avérer intéressante pour des équipements dont l'énergie est limitée. Ces équipements communiquent généralement au moyen de technologies sans fil et il est donc intéressant d'étudier les performances de *M2C* pour ce type de communications où :

- Les caractéristiques des réseaux sans fil peuvent poser des problèmes notamment en terme de fiabilité tels que les problèmes rencontrés pour *TCP* [Mascolo 01] et ainsi perturber le comportement de *M2C*.
- Des problèmes pour des usages mobiles [Sarikaya 09] apparaissent, où la lenteur d'abonnement pose des problèmes de discontinuité de services et de variations importantes de débit, comme par exemple pour des flux audios ou vidéos.

MÉCANISMES DE ROUTAGE EN CŒUR DE RÉSEAU

Parallèlement à ces études, il est important de développer les mécanismes de routage et de signalisation pour le cœur du réseau. Ainsi, le développement de l'utilisation du *multicast* peut s'appuyer sur la création de mécanisme de routage *multicast* hybride entre implémentation native dans les routeurs et applicative sur les machines clientes pour interconnecter les réseaux multicast natifs et les zones qui en sont dépourvues.
De plus, pour permettre l'essor des protocoles comme *M2C* consommant une grande quantité de groupe *multicast*, il est important d'étudier le passage à l'échelle du nombre d'états et des modifications engendrées par la signalisation *IGMP* et *Protocol Independent Multicast (PIM)*. Ainsi, une évolution possible est de proposer une signalisation spécifique permettant de s'abonner alternativement aux différents groupes *multicast* d'une même session sans pour autant nécessiter de créer et de détruire à chaque fois des états dans les tables de routage *multicast*. C'est-à-dire différencier le désabonnement d'un groupe et l'arrêt de la propagation des données qui pourra être réactivée lors du prochain cycle de la source *M2C*.
Enfin, nous avons montré que les protocoles de contrôle de congestions à débit dynamique sont insensibles aux pertes de paquets de désabonnement d'un groupe *multicast*. Cependant, la fiabilité de la signalisation concernant les abonnements peut influencer les performances de *M2C*. Il est donc important de mesurer et d'évaluer cette influence de la fiabilité du protocole *PIM* et de la proposition de l'utiliser avec *TCP* ou *SCTP* [Stewart 07].

BIBLIOGRAPHIE

[Albanna 01] Z. Albanna, K. Almeroth, D. Meyer & M. Schipper. *IANA Guidelines for IPv4 Multicast Address Assignments*. RFC 3171, IETF, August 2001.

[Birk 03] Y. Birk & D. Crupnicoff. *A Multicast Transmission Schedule for Scalable Multi-Rate Distribution of Bulk Data using Non-Scalable Erasure-Correcting Codes*. In INFOCOM, San Francisco, March 2003.

[Brakmo 95] L.S. Brakmo & L.L. Peterson. *TCP Vegas : end to end congestion avoidance on a global Internet*. IEEE JSAC, vol. 13, no. 8, pages 1465–1480, 1995.

[Byers 02] J.W. Byers, G. Horn, M. Luby, M. Mitzenmacher & W. Shaver. *FLID-DL : congestion control for layered multicast*. IEEE JSAC, vol. 20, no. 8, pages 1558–1570, 2002.

[Cain 02] B. Cain, S. Deering, I. Kouvelas, B. Fenner & A. Thyagarajan. *Internet Group Management Protocol, Version 3*. RFC 3376, IETF, October 2002.

[Chaintreau 01] A. Chaintreau, F. Baccelli & C. Diot. *Impact of Network Delay Variations on Multicast Sessions with TCP-Like Congestion Control*. In IEEE INFOCOM, volume 2, pages 1133–1142, 2001.

[Chiu 89] D.-M. Chiu & R. Jain. *Analysis of the Increase and Decrease Algorithms for Congestion Avoidance in Computer Networks*. Computer Networks and ISDN Systems, vol. 17, no. 1, page 114, 1989.

[Deering 91] S. Deering. *Multicast Routing in a Datagram Internetwork*. PhD thesis, Standford, December 1991.

[Farinacci 10] D. Farinacci, S. Venaas & M. Napierala. *A Reliable Transport Mechanism for PIM*. draft 03, IETF, March 2010.

[Fenner 06] B. Fenner, M. Handley, H. Holbrook & I. Kouvelas. *Protocol Independent Multicast - Sparse Mode (PIM-SM) : Protocol Specification (Revised)*. RFC 4601, IETF, August 2006.

[Floyd 93] S. Floyd & V. Jacobson. *Random early detection gateways for congestion avoidance*. IEEE/ACM Trans. Netw., vol. 1, no. 4, pages 397–413, 1993.

[Floyd 94] S. Floyd. *TCP and explicit congestion notification*. SIGCOMM Comput. Commun. Rev., vol. 24, no. 5, pages 8–23, 1994.

[Floyd 00a] S. Floyd, M. Handley & J. Padhye. *A Comparison of Equation-Based and AIMD Congestion Control*. Rapport technique, May 2000.

[Floyd 00b] S. Floyd, M. Handley, J. Padhye & J. Widmer. *Equation-based congestion control for unicast applications*. In Proceedings of the conference on Applications, Technologies, Architectures, and Protocols for Computer Communication, pages 43–56, Stockholm, Sweden, 2000. ACM.

[Floyd 08] S. Floyd. *Metrics for the Evaluation of Congestion Control Mechanisms*. RFC 5166, IETF, March 2008.

[Ha 08] S. Ha, I. Rhee & L. Xu. *CUBIC : a new TCP-friendly high-speed TCP variant*. SIGOPS Oper. Syst. Rev., vol. 42, no. 5, pages 64–74, 2008.

[Handley 03] M. Handley, S. Floyd, J. Padhye & J. Widmer. *TCP Friendly Rate Control (TFRC) : Protocol Specification*. RFC 3448, IETF, January 2003.

[Hinden 06] R. Hinden & S. Deering. *IP Version 6 Addressing Architecture*. RFC 4291, IETF, February 2006.

[Hoerdt 06] M. Hoerdt, B. Owens, S. Venaas, H. Asaeda, A. Gall, N. Bhaskar & P. Savola. *[m6bone] Join delay tree setup time, Why is it so long on Cisco/Juniper ?*, 2006. Published : IPv6 Multicast mailing-list – m6bone@ml.renater.fr https ://listes.renater.fr/sympa/arc/m6bone/2006-01/msg00041.html.

[Holbrook 99] H. W. Holbrook & D. R. Cheriton. *IP multicast channels : EXPRESS support for large-scale single-source applications*. SIGCOMM Comput. Commun. Rev., vol. 29, no. 4, pages 65–78, 1999.

[Holbrook 01] H. W. Holbrook. *A channel model for multicast*. PhD thesis, Stanford University, 2001.

[INRIA 06] Project Planète INRIA. *Planete-bcast : Tools for Large Scale Content Distribution*. http ://planete-bcast.inrialpes.fr/, 2006.

[Jacobson 88] V. Jacobson. *Congestion avoidance and control*. In Symposium proceedings on Communications architectures and protocols, pages 314–329, Stanford, California, United States, 1988. ACM.

[Jain 98] R. Jain, D. Chiu & W. Hawe. *A Quantitative Measure Of Fairness And Discrimination For Resource Allocation In Shared Computer Systems*. cs/9809099, September 1998.

[Jain 99] R. Jain, A. Durresi & G. Babic. *Throughput Fairness Index : An Explanation*. In ATM Forum Contribution 99, volume 45, 1999.

[Legout 00a] A. Legout. *Contrôle de congestion multipoint pour les réseaux best effort*. PhD thesis, Université de Nice-Sophia Antipolis, 2000.

[Legout 00b] A. Legout & E. W. Biersack. *PLM : fast convergence for cumulative layered multicast transmisson schemes*. In Proceedings of the 2000 ACM SIGMETRICS international conference on Measurement and modeling of computer systems, pages 13–22, Santa Clara, California, United States, 2000. ACM.

[Legout 02] A. Legout & E. W. Biersack. *Revisiting the fair queuing paradigm for end-to-end congestion control*. IEEE network, vol. 16, no. 5, page 38–46, 2002.

[Li 07] Y.-T. Li, D. Leith & R. N. Shorten. *Experimental evaluation of TCP protocols for high-speed networks*. IEEE/ACM Trans. Netw., vol. 15, no. 5, pages 1109–1122, 2007.

[Lopez-Aguilera 08] E. Lopez-Aguilera, M. Heusse, Y. Grunenberger, F. Rousseau, A. Duda & J. Casademont. *An Asymmetric Access Point for Solving the Unfairness Problem in WLANs*. IEEE Transactions

on Mobile Computing, vol. 7, no. 10, pages 1213–1227, 2008.

[Luby 00] M. Luby & A. Haken. *Reliable Multicast Transport Building Block : Layered Congestion Control*. draft 00, IETF, November 2000.

[Luby 02] M. Luby, V. Goyal, S. Skaria & G. B. Horn. *Wave and equation based rate control using multicast round trip time*. SIGCOMM Comput. Commun. Rev., vol. 32, no. 4, pages 191–204, 2002.

[Luby 04a] M. Luby & V. Goyal. *Wave and Equation Based Rate Control (WEBRC) Building Block*. RFC 3738, IETF, April 2004.

[Luby 04b] M. Luby, R. Lehtonen, V. Roca & R. Walsh. *FLUTE - File Delivery over Unidirectional Transport*. RFC 3926, IETF, October 2004.

[Mascolo 01] S. Mascolo, C. Casetti, M. Gerla, M. Y. Sanadidi & Ren Wang. *TCP westwood : Bandwidth estimation for enhanced transport over wireless links*. In Proceedings of the 7th annual international conference on Mobile computing and networking, pages 287–297, Rome, Italy, 2001. ACM.

[Mathis 97] M. Mathis, J. Semke, J. Mahdavi & T. Ott. *The macroscopic behavior of the TCP congestion avoidance algorithm*. SIGCOMM Comput. Commun. Rev., vol. 27, no. 3, pages 67–82, 1997.

[McCanne 96a] S. McCanne. *Scalable Compression and Transmission of Internet Multicast Video*. PhD thesis, EECS Department, University of California, Berkeley, December 1996.

[McCanne 96b] S. McCanne, V. Jacobson & M. Vetterli. *Receiver-driven layered multicast*. In Conference proceedings on Applications, technologies, architectures, and protocols for computer communications, pages 117–130, Palo Alto, California, United States, 1996. ACM.

[McCanne 97] S. McCanne, M. Vetterli & V. Jacobson. *Low-complexity video coding for receiver-driven layered multicast*. IEEE JSAC, vol. 15, no. 6, pages 983–1001, 1997.

[Munir 07] K. Munir, M. Welzl & D. Damjanovic. *Linux beats windows !–or the worrying evolution*

of TCP in common operating systems. In PFLDnet Workshop, 2007.

[Namburi 04] P. Namburi, K. Sarac & K. C Almeroth. *Ssmping : A ping utility for source specific multicast*. In IASTED CIIT, volume 11, page 63–68, St. Thomas, US Virgin Islands, USA, 2004.

[Padhye 98] J. Padhye, V. Firoiu, D. Towsley & J. Kurose. *Modeling TCP throughput : a simple model and its empirical validation*. SIGCOMM Comput. Commun. Rev., vol. 28, no. 4, pages 303–314, 1998.

[Peltotalo 07] J. Peltotalo, S. Peltotalo, J. Harju & R. Walsh. *Performance analysis of a file delivery system based on the FLUTE protocol*. International Journal of Communication Systems, vol. 20, no. 6, pages 633–659, 2007.

[Puangpronpitag 03] S. Puangpronpitag. *Design and Performance Evaluation of Multicast Congestion Control for the Internet*. PhD thesis, University of Leeds, November 2003.

[Quinn 01] B. Quinn & K. Almeroth. *IP Multicast Applications : Challenges and Solutions*. RFC 3170, IETF, 2001.

[Rhee 07] I. Rhee & L. Xu. *Limitations of equation-based congestion control*. IEEE/ACM Trans. Netw., vol. 15, no. 4, pages 852–865, 2007.

[Rizzo 97] L. Rizzo. *Effective erasure codes for reliable computer communication protocols*. SIGCOMM Comput. Commun. Rev., vol. 27, no. 2, pages 24–36, 1997.

[Rizzo 00] L. Rizzo. *pgmcc : a TCP-friendly single-rate multicast congestion control scheme*. In Proceedings of the conference on Applications, Technologies, Architectures, and Protocols for Computer Communication, pages 17–28, Stockholm, Sweden, 2000. ACM.

[Roca 01] V. Roca & A. El-Sayed. *A Host-Based Multicast (HBM) Solution for Group Communications*. In Networking — ICN 2001, LNCS, pages 610–619. 2001.

[Roca 04] V. Roca & C. Neumann. *Design, Evaluation and Comparison of Four Large Block FEC Codecs, LDPC, LDGM, LDGM Staircase and LDGM*

Triangle, plus a Reed-Solomon Small Block FEC Codec. Rapport technique, 2004.

[Sarikaya 09] B. Sarikaya & S. Venaas. *Multicast Mobility (multimob) - Charter*. http ://datatracker.ietf.org/wg/multimob/charter/, 2009.

[Soldani 09] Cyril Soldani & Guy Leduc. *Optimisation des débits des couches d'une transmission vidéo multipoint avec une meilleure prise en compte du surcoût d'encodage*. In CFIP, October 2009.

[Stevens 97] W. Stevens. *TCP Slow Start, Congestion Avoidance, Fast Retransmit, and Fast Recovery Algorithms*. RFC 2001, IETF, 1997.

[Stewart 07] R Stewart. *Stream Control Transmission Protocol*. RFC 4960, IETF, September 2007.

[Thaler 10] D. Thaler, M. Talwar, A. Aggarwal, L. Vicisano & T. Pusateri. *Automatic IP Multicast Without Explicit Tunnels (AMT)*. draft 10, IETF, March 2010.

[Venaas 06] S. Venaas. *ssmping*. http ://www.venaas.no/multicast/ssmping/, 2006.

[Vicisano 98] L. Vicisano, J. Crowcroft & L. Rizzo. *TCP-like congestion control for layered multicast data transfer*. In INFOCOM, volume 3, pages 996–1003 vol.3, 1998.

[Vida 04] R. Vida & L. Costa. *Multicast Listener Discovery Version 2 (MLDv2) for IPv6*. RFC 3810, IETF, June 2004.

[Vojnovic 02] M. Vojnovic & J.-Y. Le Boudec. *On the long-run behavior of equation-based rate control*. SIGCOMM Comput. Commun. Rev., vol. 32, no. 4, pages 103–116, 2002.

[Wang 04] Z. Wang, A.C. Bovik, H.R. Sheikh & E.P. Simoncelli. *Image Quality Assessment : From Error Visibility to Structural Similarity*. IEEE Transactions on Image Processing, vol. 13, no. 4, pages 600–612, 2004.

[Xu 04] L. Xu, K. Harfoush & I. Rhee. *Binary increase congestion control (BIC) for fast long-distance networks*. In INFOCOM, volume 4, 2004.

[Yang 00] Y.R. Yang & S.S. Lam. *General AIMD congestion control*. In ICNP, pages 187–198, 2000.

LISTES DES PUBLICATIONS

CONFÉRENCES INTERNATIONALES

[Lucas 09c] V. Lucas, J.-J. Pansiot, D. Grad. & B. Hilt *Fair Multicast Congestion Control (M2C)*. In IEEE Global Internet Symposium, Rio De Janeiro, Brazil, April 2009.

CONFÉRENCES NATIONALES

[Lucas 08] V. Lucas, J.-J. Pansiot, D. Grad & B. Hilt. *Vers un mécanisme de contrôle de congestion dirigé par les récepteurs pour le multicast*. In CFIP, Les Arcs, FRANCE, March 2008.

[Lucas 06] V. Lucas, J.-J. Pansiot & M. Hoerdt. *Influence du temps d'adhésion sur le contrôle de congestion multicast*. In CFIP, Tozeur (Tunisie), October 2006.

COMMUNICATIONS ET PUBLICATIONS DIVERSES

[Lucas 10b] V. Lucas, J.-J. Pansiot, D. Grad & B. Hilt. *Efficient multicast data transfer with congestion control using dynamic source channels*. arXiv.org 1004.4809, April 2010.

[Lucas 07] V. Lucas, J.-J. Pansiot & D. Grad. *Contrôle de congestion pour le multicast*. Poster - RESCOM, June 2007.

[Pansiot 05] J.-J. Pansiot, M. Hoerdt, V. Lucas & S. Venaas. *D5.10 : Report on using multipoint applications over SSM*. Project deliverable, June 2005.

LOGICIELS

[Lucas 09a] V. Lucas. *must : multicast streaming application*. http://sourceforge.net/projects/must/, 2009.

[Lucas 09b] V. Lucas. *mutual : multicast file sharing*. http://sourceforge.net/projects/mutual/, 2009.

[Lucas 10a] V. Lucas. *mcc : FLID-SL, FLID-DL, WEBRC and M2C implementation*. http://svnet.unistra.fr/mcc/, 2010.

A

SPÉCIFICATIONS COMPLÈTES DE *M2C*

Reliable Multicast Transport V. Lucas
Internet-Draft J-J. Pansiot
Expires: February 2, 2010 D. Grad
LSIIT
B. Hilt
MIPS
August 2009

Multicast Congestion Control (M2C) Building Block
draft_m2c_lucas_pansiot_grad_hilt

Status of this Memo

By submitting this Internet-Draft, each author represents that any applicable patent or other IPR claims of which he or she is aware have been or will be disclosed, and any of which he or she becomes aware will be disclosed, in accordance with Section 6 of BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF), its areas, and its working groups. Note that other groups may also distribute working documents as Internet-Drafts.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

The list of current Internet-Drafts can be accessed at http://www.ietf.org/ietf/1id-abstracts.txt.

The list of Internet-Draft Shadow Directories can be accessed at http://www.ietf.org/shadow.html.

This Internet-Draft will expire on February 2, 2010.

Abstract

This document specifies the Multicast Congestion Control (M2C) protocol. M2C is an highly scalable receiver driven protocol providing multiple-rates such as each receiver can independently choose its received rate. This received rate corresponds to the fair rate that each receiver computes depending on the network conditions it experiments. The fair rate is computed by each receiver without sending any feedback to the source. This property ables M2C to be highly scalable to a large number of receivers for the same session. Moreover, since each receiver adjusts individually its reception rate

according to the network conditions it experiments, M2C provides to each receiver the highest rate it can afford.

M2C follows the same goal as WEBRC and can be seen as an improvement of it. Specifically, M2C solves the problem of join time latency which leads WEBRC to compute a non-representative Average multicast Round Trip Time (ARTT). Moreover, experiments show that M2C fair rate computation is more robust than WEBRC one.

Table of Contents

1. Introduction

This document specifies Multicast Congestion Control (M2C) building block. M2C is designed to reach the same goals as WEBRC [refs.RFC3738] and is based upon it.

M2C is designed to compete fairly with TCP and similar congestion-controlled sessions. M2C can be used as a congestion control protocol for any type of data delivery, including reliable content delivery and streaming delivery.

Compared to WEBRC, M2C source only differs by a modified dynamic rate slope. This leads M2C to improve its reactivity when a congestion event occurs, which can cause a burst of losses.

M2C receiver uses several states similar to those used by TCP sender [refs.RFC2001]. In particular, each M2C receiver uses Slow Start (SS), Congestion Avoidance (CA) and a new Fast Start (FS) state to compute its fair rate. The received rate is adjusted to this fair rate, by joining a set of layered multicast groups provided by the source.

As WEBRC, M2C fair rate computation depends on an estimation of the RTT and on loss experiments. Each M2C receiver Estimates its Round Trip Time (ERTT) using the One Way Delay (OWD) corresponding for each packet to the delay between the source timestamp and its reception local time. Of course, this requires that all sources and receivers MUST be synchronized by a mechanism such as NTP or GPS. The ERTT is measured individually by each receiver and averaged as a substitute for conventional unicast Round-Trip Time (RTT). As the throughput of a TCP session depends strongly on RTT, an accurate estimation of the RTT is essential for a congestion control protocol to be "TCP-friendly". Finally, all bottleneck downstream receivers of a multicast group may experience similar losses and compute similar ERTT converging to the same fair rate.

M2C is designed to provide a long-term fairness when competing with others M2C or TCP streams. Indeed, M2C mechanisms make throughput variations far smoother than TCP ones, which makes it more suitable for long lived applications such as video streaming or large file transfer. This smooth variation behavior of the throughput implies a less responsive behavior than TCP to congestion events and to the variations of the available bandwidth.

The paper [refs.GIS09] provides much of the rationale and intuition for the M2C design and presents a set of experiments to evaluate M2C and WEBRC performances.

2. Rationale

M2C is based on WEBRC and focus on the same goals:

- Provide an highly scalable multicast congestion control.
- Be TCP-friendly.

This draft proposes WEBRC enhancements concerning two remaining problems:

- WEBRC uses the join time to approximate the RTT. But, PIM join is a signalization message, which is treated in the control plane and is scheduled by most routers. This has been verified experimentally [refs.CFIP06] showing that a router adds an average delay of 150ms to forward a PIM join. Thereby, the join time can take several seconds whereas the RTT is less than one hundred milliseconds.
- Even with a network where the join time is an accurate estimation of the RTT, the estimation of the fair rate of WEBRC is not robust enough notably when there are multiple WEBRC competitor streams [refs.GIS09].

3. Basic principles

M2C is a receiver-driven congestion control protocol and shares its basic principles with WEBRC. M2C is divided into two parts: the source and the receiver, which is the main part of the protocol.

The source splits the application stream into several layers, each sent to a different multicast group. Individually, each receiver joins more or less multicast groups to manage its reception rate, according to its estimation of the fair rate based on the experimented network conditions.

More details about source and receiver design are respectively given in Section 4 and Section 5.

4. Source part

M2C source part sends out application data into several multicast groups, each identified by a Group Number (GN). Given a multicast group, each packet sent is identified by a Packet Sequence Number (PSN). The source divides the time into time slots of Time Slot Duration (TSD) each and identified by a Time Slots Index (TSI).

As WEBRC, M2C uses the principle of dynamic rate for each multicast group. This means that each multicast group begins to send at its highest rate and decreases progressively until becoming quiescent.

Only the base group does not become quiescent. Each receiver MUST join this group when starting a session.

4.1. Sender inputs and initialization

The primary inputs to the sender of the session are :

- R_max and R_min, which are respectively the aggregate rate of all groups at each TSI beginning and the rate of the base group at each TSI ending in Kbits per second.
- The multicast address for the base group. Other multicast group addresses are automatically computed by adding the GN to this base address.
- The UDP port used for the base group. Other group ports are automatically computed by adding the GN to this port.

The secondary inputs to the sender are listed below. These inputs are secondary because their values will generally be fixed to the RECOMMENDED values:

- TSD is the time slot duration in seconds. The RECOMMENDED value for TSD is 10 seconds.
- QD is the minimum quiescent period duration in seconds. QD MUST be a multiple of TSD and MUST be > 260 seconds, which corresponds to ((Robustness_Variable * Query_Interval) + Query_Response_Interval) for which IGMPv3 [refs.RFC3376] keeps the multicast group record whithout any other join report: i.e. when the leave report has been lost. 260 seconds are sufficient as they are > 210 seconds, which is the PIM Keepalive_Period [refs.RFC4601]. The RECOMMENDED value for QD is 300 seconds.
- K is the multiplicative drop speed-up factor, which corresponds to the number of groups becoming quiescent at each new TSI. The RECOMMENDED value for K is 4. K MUST be >= 1.
- P is the multiplicative drop in every group rate over each (TSD / K) seconds. The RECOMMENDED value for P is 0.75. P is also the ratio between the aggregate rate of (n+1) and (n) groups. P must be > 0 and < 1.

Note that, since multicast join time can last several seconds, it is necessary to keep a TSD >= 10 seconds in order that a receiver has the time to join the next group before this one becomes quiescent. That is why having K > 1 ables to verify this constraint and at the same time to improve protocol reactivity to congestion event with the same inter-group rate granularity of P.

Each group alternates between an active and a quiescent period. An active period followed by a quiescent period constitutes a cycle, which is repeated indefinitely. From these inputs the following fixed sender parameters can be derived:

- o N the number of active groups including the base group.
  N := floor(log_P(R_min / R_max)).
- o NA the number of active time slots per cycle for a group.
  NA := ceil(N / K).
- o NQ the number of quiescent time slots per cycle for a group.
  NQ := QD / TSD.
- o TN the total number of time slots of a cycle.
  TN := NA + NQ.
- o NQG the number of simultaneous quiescent groups.
  NQG := NQ * K.
- o TNG the total number of groups.
  TNG := N + NQG.

4.2. Sending packets to the session

Each time the current TSI changes, K multicast groups become quiescent and K new multicast groups begin to send at their highest rates.

The base group is a particular multicast group that does never become quiescent. Packets of this group are sent at a rate (R_min / (P^K)) at the beginning of a time slot and decreases gradually to reach R_min at the end of the time slot. This base group pattern is repeated over each time slot.

The other multicast groups are dynamic and are composed of 2 separate periods:

- o An active period. A group GN is active when TSI is from ((ceil(GN / K) + TN - NA) modulo TN) to (ceil(GN / K)) included.
- o A quiescent period. A group GN is quiescent during the NQ time slots where the group is not active. When quiescent, no packet are sent via this group.

Each new TSI, K groups become active at high rate ((R_min * (1 - P)) / (P^(N + K - x))) with x in [0...K[. For example if K := 4, 4 groups start at the same time with rates:

- o (R_min * (1 - P)) / (P^(N + K - 3))
- o (R_min * (1 - P)) / (P^(N + K - 2))
- o (R_min * (1 - P)) / (P^(N + K - 1))
- o (R_min * (1 - P)) / (P^(N + K - 0))

Then each group rate decreases in function of time by a fixed fraction (P^K) per time slot. This decrease lasts NA time slots. After NA time slots of activity, a multicast group becomes quiescent, during which the multicast group does not send any packet anymore. The quiescent period lasts NQ time slots. After, this quiescent period the cycle loops and the group becomes active once again.

Let us call R_start the rate of a dynamic group when it becomes active or the rate of the base group at the beginning of a new TSI.

For all groups, the current rate can be computed by (R_start * P^((t * K) / TSD)) with t the time since the group is active for dynamic groups or since the last change of TSI for the base group. The current rate of a group has to be recomputed for each new packet sent.

The sender keeps track of the current time slot index TSI. The value of TSI is incremented by 1 modulo TN each TSD seconds. The value of TSI is placed into each packet header in the format described in Section 7.

For each packet sent to the session, the sender also places the group number GN of the group into the packet header in the format described in Section 7. GN = 0 for the base group and GN = 1,2,...,(TN-1) for the dynamic groups.

For each packet sent to the session, the sender calculates a Packet Sequence Number (PSN) and places it also into the packet header in the format described in Section 7. The value of PSN starts at 0 for the first packet of a group leaving a quiescent period. For each new packet sent PSN is incremented by 1 modulo 2^16.

5. Receiver part

M2C receiver starts to join the base group and increases its reception rate by joining a consecutive set of dynamic groups. In opposition to WEBRC, M2C ables the receivers to join several multicast groups simultaneously. It remains connected to the base group until the end of the session.

A receiver decreases its reception rate by waiting for the currently joined groups to decrease their rate by a factor of P^K each TSD seconds. A M2C receiver only leaves a group when it becomes quiescent, corresponding to K groups per TSD seconds.

Each receiver computes its own fair rate and joins dynamic groups to receive the closest rate below this fair rate.

The fair rate computation is based upon TCP-like Slow Start (SS), Congestion Avoidance (CA) and a new Fast Start (FS) state.

5.1. Receiver inputs and initialization

Before joining a session, a receiver must obtain the session description. The session description is determined by the sender and is typically communicated to the receivers out of band. How receivers obtain the session description is outside the scope of this

document. In order to join a session a receiver only has to know:
- The source address (only for SSM session).
- The multicast address of the base group.
- The destination port of the base group.
- R_min and R_max in order to compute N as well as the current sending rate of the source.

The secondary inputs to the receiver are listed below. For those shared with the source part, they must be set to the same value :
- QD is the minimum quiescent period duration in seconds. QD MUST be a multiple of TSD and MUST be > 210 seconds, which is the PIM Keepalive_Period [refs.RFC4601]. The RECOMMENDED value for QD is 300 seconds.
- K is the multiplicative drop speed-up factor and represents the number of groups becoming quiescent at each new TSI. The RECOMMENDED value for K is 4. K MUST be >= 1.
- P is the multiplicative drop in every group rate over each (TSD / K) seconds. The RECOMMENDED value for P is 0.75. P is as well the ratio between the aggregate rate of (n+1) and (n) groups. P must be > 0 and < 1.
- TSD is the time slot duration measured in seconds. The RECOMMENDED value for TSD is 10 seconds.
- LR_NB is the number of packets used for the linear regression in the Fast Start (FS) algorithm. The default value for LR_NB is 10.

5.2. Multiplicative reduction factor

The multiplicative reduction factor beta is used in most computation functions of the receiver part. This important function can be computed by:
beta(t) := P^((t * K) / TSD)
with t the time of the reduction.
This reduction corresponds to the source rate slope of the dynamic multicast groups.

5.3. Receiver computation

For each new received packet, the receiver looks into the packet header to get:
- The current TSI.
- The GN of the packet.
- The PSN of the packet.
- The packet timestamp.

Then, the receiver:
- Computes the Current Expected Rate (CER).

- Computes the new Estimation of the Fair Rate (EFR).
- Joins new groups if needed.
- Leaves K groups if the TSI has changed.
- Leaves the session if a TimeOut (TO) occurs.

5.3.1. Current Expected Rate (CER) computation

Each receiver can compute its actual expected rate sent:
CER := R_min * (1 / P) ^ NB_G * beta(t)
With :
- t the time since the beginning of the current TSI.
- NB_G the number of groups actually joined or currently joining (the multicast join has been sent but no packet of this group has been received yet) including the base group.

5.3.2. Estimation of the Fair Rate (EFR)

The Estimation of the Fair Rate (EFR) is the main part of the M2C protocol. The EFR computation is divided into 9 parts:
- Initialization of the EFR.
- Estimation of the RTT (ERTT).
- Fast Start (FS) state.
- Slow Start (SS) state.
- Congestion Avoidance (CA) state.
- Congestion Event (CE) transition.
- Filled Buffer (FB) transition.
- Far below Fair Rate (FFR) transition.
- Buffer Filling In (BFI) transition.

5.3.2.1. Initialization of the EFR

Once the first packet received, the receiver must set the initial values for the following parameters:
- EFR is set to R_min
- The current TSI to the received one.
- The PSN of the received GN.
- The current state is set to FS.
- ERTT is set to the OWD obtained by the first packet.
- max_ERTT is set to ERTT.
- Minimal One Way Delay (OWD_min) is set to ERTT.
- The Estimated Buffer Time (EBT) is set to 0.
- max_EBT is set to EBT.
- The current Variation of ERTT (VERTT) is set to 0.
- The last Buffer Filled Event (last_BFE) is set to the current date.

5.3.2.2. Estimation of the RTT (ERTT)

ERTT is an Exponentially Weighted Moving Average (EWMA) based on One Way Delay (OWD) measurements. Note that, ERTT is only updated if the packet is not desequenced (cf. Section 5.3.2.6).

First, the One Way Delay (OWD) is computed as the difference between the packet reception time and the packet timestamp.

Then, the Estimated Buffer Time (EBT) is computed via the last_jitter which is the difference between the last two OWDs:
EBT := max(0, EBT + last_jitter).

Then, the Estimated Propagation Time (EPT) is computed as:
EPT := max(0, OWD - EBT).

The last ERTT (last_ERTT) is computed as:
last_ERTT := (2 * EPT) + EBT.

And finally ERTT is computed with:
ERTT := (0.75 * ERTT) + (0.25 * last_ERTT).

5.3.2.3. Fast Start (FS) state

The Fast Start (FS) state is used at the beginning of each session. The objective of FS is to quickly use the initially available bandwidth, which is the minimum unused bandwidth along the path. FS updates the EFR as explained for the Slow Start (SS) state and each receiver joins new groups according to the algorithm described in Section 5.3.3. But, even if this algorithm does not decide to join a new group, FS forces a receiver to join a new group only if there is no currently joining group.

During FS, the receiver maintains a list of the following parameters for the LR_NB last packets received:
- The delay between the last received packet and the beginning of the session.
- The variation of the OWD which can be calculated by the difference between the last OWD and the minimal OWD.
- The size of the received packet.

This list of LR_NB is used by the Buffer Filling In (BFI) transition.

FS state stops:
- If a Congestion Event (CE) occurs. Then, the protocol switches to the CA state.
- If the Buffer Filling In (BFI) event occurs. Then, the protocol switches to the SS state.

Once FS is stopped, the EFR is updated by the rate received for the last LR_NB received packets, only if this one is over the current EFR.

5.3.2.4. Slow Start (SS) state

The Slow Start (SS) state consists in a multiplicative increase in order to quickly reach a bandwidth close to the fair rate. Each time a new packet is received, the receiver must multiply its EFR by (1 / beta(t)) with t the delay between the reception of the two last packets.

SS state stops if a Congestion Event (CE) occurs. Then, the protocol switches to the CA state.

5.3.2.5. Congestion Avoidance (CA) state

In congestion avoidance (CA) state an additive increase of the EFR is used to share fairly the bandwidth:
EFR := EFR + (((alpha * packet_size^2) / window_size) / ERTT)
With:

- window_size: the size of an emulated TCP window in function of the current ERTT and EFR. The window_size is computed as EFR * ERTT.
- packet_size: the size of the last received packet.
- alpha: the increase factor corresponding to an AIMD TCP-friendly behavior depending on beta(t), with t the delay between the two last received packets. alpha is computed as:
  alpha := 3 * ((1 - beta(t)) / (1 + beta(t)))

CA state stops, if a Far from Fair Rate (FFR) event occurs. Then, the protocol switches to the SS state.
If a Congestion Event (CE) or a Filled Buffer (FB) event occurs in CA state, the protocol follows the corresponding transition behavior, but remains in the CA state.

5.3.2.6. Congestion Event (CE) transition

The Congestion Event (CE) transition may happen in every state: FS, SS or CA. It checks if there is a loss by comparing the PSN of the current packet with the previous received PSN for the same group.
A packet loss is detected if there is at least one packet previously received for the group GN and if for the same GN, the PSN_current_packet differs from the (PSN_previous_packet + 1) modulo (2^16).
Each time a loss is detected, the receiver must reduce its EFR by a multiplicative factor. The reduction factor is beta(t) (cf. Section 5.2) with t the delay between the reception of the two last packets.

When a CE occurs, the protocol switches to CA state and updates the last_BFE to the current date.

Besides, the current packet is considered to be:

- Duplicated, if PSN_current_packet is equal to PSN_previous_packet.
- Desequenced, if (((2^16) + PSN_current_packet - PSN_previous_packet) modulo (2^16)) > (2^15).

5.3.2.7. Filled Buffer (FB) transition

The Filled Buffer (FB) transition is only while in CA state. For each new packet, FB detects if a competing stream may experiment a congestion event. This situation is considered as occurring if the buffer is almost filled:
EBT > max_EBT - VERTT.
With:

- max_EBT: the maximal of the computed EBT since the last CE.
- The Variation of ERTT (VERTT) is computed for each packet received as:
  last_VERTT := |last_ERTT - ERTT|
  VERTT := (0.75 * VERTT) + (0.25 * last_VERTT) Note that, VERTT is only updated if the packet is not desequenced (cf. Section 5.3.2.6).

When a FB occurs, the protocol remains in CA state but updates the last_BFE to the current date.

5.3.2.8. Far below Fair Rate (FFR) transition

The Far below Fair Rate (FFR) transition is only used in CA state. FFR detection is done to re-activate the SS state when the receiver detects that its EFR is far below the fair rate. For example, this may occur when a competing flow has stopped. This detection is the result of the comparison of the time elapsed since the last_BFE and the expected delay between two losses that a TCP stream may experiment for the same reception rate called TCP Loss Cycle Estimation (TCP_LCE). TCP_LCE is computed as:
TCP_LCE := (NB_RTT * max_ERTT) + last_JT
With:

- NB_RTT: the number of RTT for an equivalent TCP stream.
  NB_RTT := 0.5 * (window_size / packet_size)
- max_ERTT: the maximal of the computed ERTT since the last CE.
- last_JT: the last join time is the time between the receiver sends an IGMP/MLD report to join a group and when the first packet of this group is received. This time is added to TCP_LCE since a long join time will delay the moment when the bottleneck may be congestioned due to the amount of data of the new group, provoking a CE.

When a FFR occurs, the protocol switches to SS state.

5.3.2.9. Buffer Filling In (BFI) transition

The Buffer Filling In (BFI) transition is only used in FS state. For each new packet, BFI checks if there is no more free bandwidth available and that the buffer of the bottleneck router begins to fill in.
BFI is only activated once LR_NB packets have been received. BFI detection is done via a linear regression using the variation of the OWD in abscissa and the time from the beginning of the session in ordinate.
If the gradient of the line obtained is over 0.25, then the protocol switches to the CA state.

5.3.3. Join new multicast groups

Each time a new packet arrives, the receiver may have to join new multicast groups, if the difference between the EFR and CER is sufficient. The number nb of new joins can be computed as:
nb = floor(log_1/P (EFR / CER))

The GN of the next dynamic group to join is :
next_group_to_join := (TSI + NB_G).

5.3.4. Leave quiescent multicast group

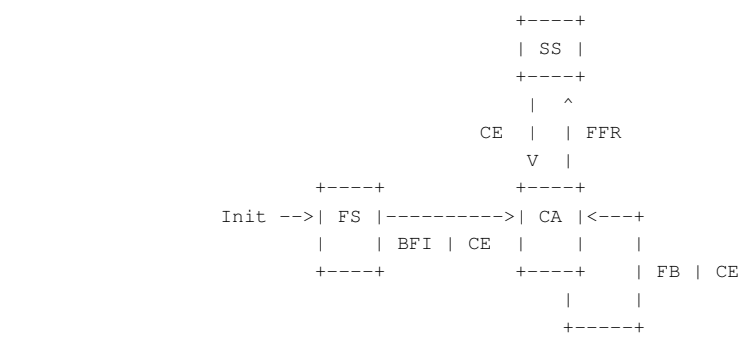
Each new TSI, K multicast groups become quiescent and must be left. A new TSI is only detected if the packet TSI corresponds to the current (TSI + 1) modulo 2^TN.

5.3.5. Leave the current session

If no packet is received since Timeout Out (TO) seconds the receiver must leave all the multicast groups and quit the session. The RECOMMENDED value for TO is 90 seconds which corresponds to (QD - PIM_Keepalive_Period) seconds.

6. Receiver EFR computation state machine

Figure 1 summarizes the receiver EFR computation state machine detailed in previous sections.

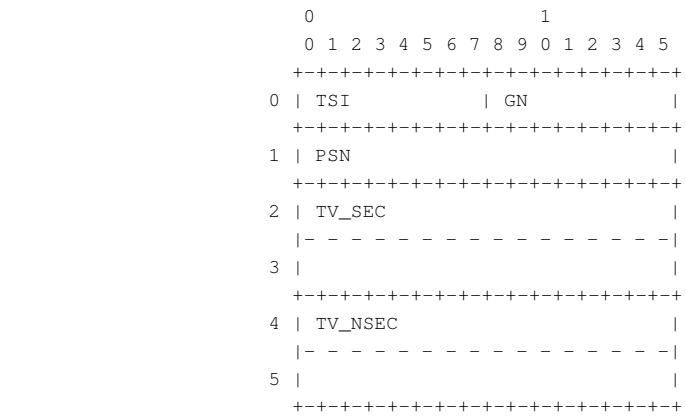


The receiver EFR computation state machine

Figure 1

7. Header description

M2C packets sent to a session MUST use the specific header described below. When using M2C in conjunction with LCT [refs.RFC3451], M2C header MUST be carried in the CCI field of the LCT header.



The M2C message header

Figure 2

The Time Slot Index TSI (8 bits) indicates the current time slot. It must be incremented each TSD seconds modulo TN. TN is the number of time slots of a cycle associated with the session and must be < 2^8.

The Group Number GN (8 bits) that this packet belongs to. Base group is identified with number 0 and dynamic groups are numbered from 1 to TNG.

The Packet Sequence Number PSN (16 bits) is the packet sequence number relative to the current group. PSN is initialized to 0 and is incremented modulo 2^16 for each packet sent.

The Time Value in SEConds TV_SEC (32 bits) and the Time Value in NanoSEConds TV_NSEC (32 bits) represent the clock time when the packet was transmitted. This value is used in OWD and jitter computation and requires that the source and the receivers have their clocks well synchronized via methods like NTP, GPS, etc.

8. Security Considerations

As M2C is based upon WEBRC, M2C is subject to the same security considerations as expressed in [refs.RFC3738].

9. References

[refs.RFC3738]
Luby, M. and V. Goyal, "Wave and Equation Based Rate Control (WEBRC) Building Block", RFC 3738, April 2004.

[refs.RFC2001]
Stevens, W., "TCP Slow Start, Congestion Avoidance, Fast Retransmit, and Fast Recovery Algorithms", RFC 2001, January 1997.

[refs.GIS09]
Lucas, V., Pansiot, J., Grad, D., and B. Hilt, "Fair Multicast Congestion Control (M2C)", 12th IEEE Global Internet Symposium 2009, April 2009.

[refs.CFIP06]
Lucas, V., Pansiot, J., and M. Hoerdt, "Influence du temps d'adhesion sur le controle de congestion multicast", Colloque Francophone sur l'Ingenierie des Protocoles (CFIP) 2006, October 2006.

[refs.RFC3451]
Luby, M., Gemmell, J., Vicisano, L., Rizzo, L., Handley, M., and J. Crowcroft, "Layered Coding Transport (LCT) Building Block", RFC 3451, December 2002.

[refs.RFC4601]
Fenner, B., Handley, M., Holbrook, H., and I. Kouvelas, "Protocol Independent Multicast - Sparse Mode (PIM-SM): Protocol Specification (Revised)", RFC 4601, August 2006.

[refs.RFC3376]
Cain, B., Deering, S., Kouvelas, I., Fenner, B., and A. Thyagarajan, "Internet Group Management Protocol, Version 3", RFC 3376, October 2002.

Appendix A. Linear regression for BFI transition

A way to detect if that the Buffer begins to Fill In (BFI) is to compute the gradient of the linear regression using the time from the beginning of the session in abscissa and the variation of the OWD in ordinate.
BFI is only activated once LR_NB packets have been received. This linear regression is done with the LR_NB data and their images produced by an origin symmetry in order to force the y-intercept of the line to match the origin.
If the gradient of the line obtained is over 0.25, then the BFI occurs and protocol switches to the SS state.

Authors' Addresses

Vincent Lucas
LSIIT, Universite de Strasbourg - CNRS
Pole API, Boulevard Sebastien Brant
Illkirch 67400
FRANCE

Phone: +33 3 90 24 45 87
Email: lucas@unistra.fr
URI: http://clarinet.u-strasbg.fr/~lucas

Jean-Jacques Pansiot
LSIIT, Universite de Strasbourg - CNRS
Pole API, Boulevard Sebastien Brant
Illkirch 67400
FRANCE

Phone: +33 3 90 24 45 63
Email: pansiot@unistra.fr
URI: http://clarinet.u-strasbg.fr/~pansiot

Dominique Grad
LSIIT, Universite de Strasbourg - CNRS
Pole API, Boulevard Sebastien Brant
Illkirch 67400
FRANCE

Phone: +33 3 90 24 45 86
Email: dominique.grad@unistra.fr
URI: https://tetras.u-strasbg.fr/~grad

Benoit Hilt
MIPS/GRTC
IUT de Colmar
BP 50568
Colmar Cedex 68008
FRANCE

Phone: +33 3 89 20 23 64
Email: benoit.hilt@uha.fr

TABLE DES FIGURES

LISTE DES TABLEAUX

GLOSSAIRE

Symbols

K

Le facteur d'accélération de réduction du débit des groupes *multicast*. Correspond également au nombre de groupes qui deviennent muets par TSD.
. 38, 46, 76, 77, 80–82, 113, 123, 124

P

Facteur principal de réduction du débit des groupes *mulitcast* pour la source $WEBRC$ et $M2C$.
. 37, 76, 77, 124

α

Facteur d'accroissement additif de la fenêtre de congestion.
. 21, 22, 84, 116

β

Facteur de réduction multiplicative de la fenêtre de congestion.
. 21, 22, 77, 83, 84, 90, 116

$channel\_rate_{MAX}$

Débit maximal d'un groupe multicast à débit dynamique.
. 77

A

ANYCAST

Modèle de communication où plusieurs entités ayant le même rôle disposent du même identifiant.
. 16

C

CAROUSEL

Mécanisme de retransmission cyclique.
. 137, 138

CUBIC

Protocole de contrôle de congestion à convergence accélérée.
. 23, 24, 26, 89

E

encodeur *FEC*
: Mécanisme permettant de générer des symboles de redondance, permettant ainsi de corriger un certain nombre de paquets perdus.
. 137, 138

Ethernet
: Protocole de la couche liaison, utilisé au sein de réseaux locaux à commutation de paquets.
. 10

M

multicast
: Modèle de communication de groupe.
. 1–6, 9–11, 13–17, 19, 29, 31–34, 36–38, 43–45, 48, 50–56, 72, 75, 77, 79, 81, 87–89, 94, 98, 102, 103, 105, 110, 113–117, 121, 123, 125–127, 130, 131, 133, 134, 137–139, 141, 142, 145, 153, 155–159

O

ordonnanceur applicatif
: Mécanisme utilisé par l'application de transfert de fichiers permettant de trier les symboles applicatifs à envoyés.
. 138–140, 142

ordonnanceur au niveau transport
: Mécanisme utilisé par les protocoles de contrôle de congestion à débit dynamique permettant d'envoyer les paquets de chaque groupe multicast au débit actuel du groupe.
. 137, 138

P

packet_size
: Taille d'un paquet.
. 98

S

séquenceur
: Mécanisme au niveau transport qui prend en entrée les données applicatives triées hiérarchiquement et les envoit de façon optimale sur les différents groupes multicast du protocole de contrôle de congestion à débit dynamique.
. 6, 137, 138, 142, 148

U

UNICAST
: Modèle de communication de 1 vers 1.
. 6, 9–11, 13, 16, 17, 19, 29, 43, 45, 52, 72, 81, 141, 155

V

VEGAS
: Protocole de contrôle de congestion utilisant le temps de propagation pour estimer les évènements de congestion.
. 25

W

WINDOW_SIZE
: Taille de la fênetre de congestion.
. 98

Acronyms

A

ACK
: Acknowledgment. 20, 21, 24, 31, 45

ADSL
: Asymmetric Digital Subscriber Line. 2, 52, 70, 102

AIMD
: Additive Increase and Multiplicative Decrease. 21, 22, 24, 81, 82, 84, 116, 143

AMT
: Automatic IP Multicast Without Explicit Tunnels. 16

API
: Application Programming Interface. 6, 125–127, 129, 137, 138, 157

ASM
: Any-Source Multicast. 11, 13–15

ATM
: Asynchronous Transfer Mode. 10

B

BFI
: Buffer Filling In. 106

BGP
: Border Gateway Protocol. 10

BIC
: Binary Increase Congestion control. 23, 89

BWU
: Bandwidth Usage. 27, 43, 58, 59, 67, 70, 111

C

CA
: Congestion Avoidance. 20–23, 29, 45, 81, 82, 84, 87, 91, 93–97, 104, 106, 107, 111, 116

CE
: Congestion Event. 21, 22, 84

CT
: Convergence Time. 28, 44, 53, 65, 70, 72, 111

CW
: Congestion Window. 20–25, 31, 32, 115, 116

D

DSLAM
: Digital Subscriber Line Access Multiplexer. 52

E

EBT
: Estimated Buffering Time. 96, 99, 116

ECN
: Explicit Congestion Notification. 26

EFR
: Estimated Fair Rate. 82–84, 91, 93, 95, 99, 104–106, 116

ERA
: Explicit Rate Adjustment. 35, 39

ERTT
: Estimated Round Trip Time. 81–84, 90, 93, 96, 98, 116

F

F
: Fairness. 28, 60, 61, 67, 70, 85, 86, 92, 99, 100, 102, 111, 113

FB
: Filled Buffer. 99

FEC
: Forward Error Correction. 133, 138, 139, 142, 143, 145

FFR
: Far from Fair Rate. 95, 97, 102–104, 113

FIFO
: First In - First Out. 25

FLID-DL
: Fair Layered Increase/Decrease with Dynamic Layering. 3, 34, 37, 39, 48, 49, 57–60, 75, 156, 158

FLID-SL
: Fair Layered Increase/Decrease with Static Layering. 3, 34, 37, 39, 48, 57–59, 158

FLT
: First Loss Time. 67, 70

FLUTE
: File Delivery over Unidirectional Transport. 3

FS
Fast Start. 104–107, 111, 116

G

GOP
Group Of Pictures. 149, 150

GPS
Global Positioning System. 77, 83, 96

H

HBM
Host-Based Multicast. 16

I

IETF
Internet Engineering Task Force. 43, 72, 155

IGMP
Internet Group Management Protocol. 10, 11, 13, 14, 36, 48, 55, 98, 114, 115, 159

IIF
Incoming InterFace. 11

IP
Internet Protocol. 1, 9–11, 13, 15–17, 26, 36, 50, 55–57, 116, 142, 155, 156

IPTV
IP TeleVision. 1, 2, 10, 31, 95, 149

IS-IS
Intermediate System to Intermediate System. 10

L

LAST_ERTT
last Estimated Round Trip Time. 98

LAST_JT
last Join Time. 98

LAST_VERTT
last Variation of Estimated Round Trip Time. 98

LDPC
Low-Density Parity-Check. 139, 140

LR
Loss Rate. 25, 27, 43, 58–60

M

M2C
: Multicast Congestion Control. 4, 81, 82, 89, 94, 95, 97, 99, 107, 113–117, 121–126, 131, 132, 141, 143, 156, 158, 159

M2Cv1
: Multicast Congestion Control version 1. 82, 84, 86, 87, 89, 91–93

M2Cv2
: Multicast Congestion Control version 2. 89–95, 97, 99

M2Cv3
: Multicast Congestion Control version 3. 95, 97–99, 102–104

M2Cv4
: Multicast Congestion Control version 4. 104, 106, 110, 111, 113

MAX_EBT
: maximal Estimated Buffering Time. 99

MAX_ERTT
: maximal Estimated Round Trip Time. 98

MDS
: Maximum Distance Separation. 139

MLD
: Multicast Listener Discovery. 10, 11, 13, 14, 36, 55, 98

N

NACK
: Negative ACKnowledgement. 2, 31

NB_RTT
: NumBer of Round Trip Time. 98

NTP
: Network Time Protocol. 36, 77, 83, 96

O

OIF
: Outgoing InterFace. 10, 11, 13

OSPF
: Open Shortest Path First. 9

OWD
: One Way Delay. 62, 63, 81–83, 87, 91, 93, 96, 98, 106, 116

P

P2P
: Peer to Peer. 70

PGMCC
: Pragmatic General Multicast Congestion Control. 2, 31, 39

PIM
Protocol Independent Multicast. 159

PIM-SM
Protocol Independent Multicast - Sparse-Mode. 10, 13, 15, 36, 48, 50, 114, 115

PLM
Pair receiver-driven Layered Multicast. 35, 36, 39

R

RED
Random Early Detection. 25, 26

RFC
Request For Comments. 43, 45, 72

RIP
Routing Information Protocol. 9

RLC
Receiver driven, Layered Congestion control. 3, 33, 34, 39

RLM
Receiver-driven Layered Multicast. 33, 39, 75

RP
Rendez-vous Point. 13–16

RR
Récepteur Réactif. 78–81, 117

RT
Reaction Time. 29, 45

RTO
Retransmission TimeOut. 25

RTT
Round Trip Time. 21–25, 27, 29, 36–39, 50, 52, 57–60, 65, 67, 72, 81–83, 87–93, 95–99, 103, 104, 111, 113, 116, 156

S

SCTP
Stream Control Transmission Protocol. 115, 159

SegSize
Segment Size. 20, 21

SO
Signaling Overhead. 45, 60, 113

SS
Slow Start. 20, 23, 45, 81–83, 87, 89, 91–99, 102–107, 111, 116

SSM
Source-Specific Multicast. 11, 13, 14, 50

T

TCP
: Transmission Control Protocol. 1, 2, 4, 15, 17, 20–26, 28, 29, 31, 35–39, 43–45, 49, 51–54, 60, 62, 63, 65, 67, 69, 70, 73, 75, 80–84, 86–91, 94–99, 102–104, 111, 113, 115, 116, 132, 133, 137, 141, 143, 145, 155, 156, 159

TCP_ERTT
: Transmission Control Protocol Estimated Round Trip Time. 98

TCP_LCE
: Transmission Control Protocol Loss Cycle Estimation. 97–99

TFRC
: TCP-Friendly Rate Control. 24, 35, 38, 39, 65, 73

TIP
: Temps Inter-Paquets. 83

TPA
: Temps de Propagation Ajouté. 49, 51–54, 61, 85, 92, 93, 100, 132, 141

TSD
: Time Slot Duration. 34, 37, 38, 46, 59, 75–77, 81, 113, 123, 124, 127, 156

TSI
: Time Slot Interval. 34, 37, 59, 77, 79, 123, 124

U

UDP
: User Datagram Protocol. 2, 31, 130

V

VERTT
: Variation of Estimated Round Trip Time. 98, 99

W

WEBRC
: Wave and Equation Based Rate Control. 3, 37–39, 48, 49, 57–60, 62, 63, 65, 67, 69, 70, 72, 73, 75–77, 80, 81, 87, 93, 114, 115, 117, 122, 123, 156, 158

WLAN
: Wireless Local Area Network. 45