Contrôle de congestion pour la transmission multipoints en couches

Ibtissam El Khayat, Guy Leduc

Université de Liège, Research Unit in Networking,
Institut Montefiore, Sart-Tilman B28, B4000 Liège, Belgique
Tel : +32 4 366 26 99, Fax: +32 4 366 29 89
Courriel : {elkhayat, leduc}@montefiore.ulg.ac.be

Synthèse

Le contrôle de congestion en transmission multipoints est rendu difficile par l'hétérogénéité des récepteurs. En effet, pour la transmission vidéo par exemple, il serait peu raisonnable que l'émetteur adapte son débit en fonction du récepteur le moins performant ou de celui qui subit temporairement la congestion la plus sévère. Pour contourner ce problème, l'émetteur peut émettre un flux structuré en couches, de telle sorte que la couche de base donne une qualité minimale et que les couches suivantes améliorent successivement cette qualité. L'algorithme de contrôle de congestion proposé est basé sur ce schéma. Il permet à chaque récepteur de sélectionner dynamiquement un sous-ensemble adéquat de couches en répondant aux objectifs suivants. Premièrement, l'algorithme doit être équitable vis-à-vis de TCP; ce qui signifie que les débits du flux multicouches reçu et celui d'un flux TCP placé dans les mêmes conditions doivent être plus ou moins les mêmes. Deuxièmement, les récepteurs doivent être suffisamment coordonnés pour qu'une congestion résultant de l'ajout d'une couche par l'un d'eux ne puisse être interprétée par un autre récepteur comme une congestion résultant de ses propres décisions ou d'un trafic perturbateur. Enfin, lorsque deux sessions multicouches empruntent un même goulet, nous souhaitons que les récepteurs obtiennent le même débit, ce qui signifiera en général des nombres de couches différents si les débits des couches sont distincts.

Mots-clés : Contrôle de congestion, Multicast, TCP-Friendly, Equité, Vidéo en couches.

1 Introduction

La transmission multimédia multipoints s'avère être aujourd'hui un besoin fondamental pour certaines applications. Nous citerons par exemple la visioconférence, la télévision sur Internet et le télé-enseignement. Le problème majeur avec ce genre d'applications réside dans la nécessité de l'envoi des données en temps réel. L'utilisation du protocole TCP, qui repose sur une bufferisation des paquets avant l'envoi, est inadéquate pour fournir le flux continu dont elles ont besoin. Il a donc fallu penser à utiliser un autre trafic permettant d'abord une transmission Multicast (TCP n'ayant pas cette capacité) ensuite mettre au point des moyens de contrôle de congestion efficaces.

Dans le cas simple d'une transmission point à point unicast, la source envoie à un débit toléré par toutes les lignes la séparant du récepteur. Ce débit ne dépassant pas, bien entendu, la capacité d'exploitation de l'information de ce dernier. Dans le cas multicast, l'adaptation du débit au niveau de la source s'avère être une tâche délicate. L'hétérogénéité du réseau fait que tous les récepteurs ne sont pas sur le même pied d'égalité. Ils ne sont pas tous capables de recevoir de l'information à très haut débit. Certains sont accessibles via des liens à forte congestion ou à faible débit, d'autres disposent de ressources modestes (mémoire limitée, processeur lent...) ne leur permettant pas le traitement en temps réel de toutes les informations qu'ils pourraient pourtant recevoir.

Donc si on optait pour une adaptation du débit au niveau de la source, cette dernière devrait envoyer au débit minimum toléré par ses récepteurs sans quoi il résulterait congestion et perte. Les récepteurs pouvant recevoir un plus haut débit se verraient donc accorder une mauvaise qualité vidéo (ou autres selon l'application) alors qu'ils disposent de ressources leur donnant droit à une meilleure qualité. Dans le cas d'une transmission multipoints, la responsabilité du contrôle de congestion est donc enlevée des mains de la source pour être mise dans celles des récepteurs. Ceux-ci prennent des décisions locales et indépendantes de la source pour adapter le débit d'information véhiculé jusqu'à eux aux contraintes du réseau.

Une approche proposée consiste à exploiter un codage multirésolution, ou en sous-couches de l'information. La source émet de façon indépendante les diverses résolutions, ou sous-couches, qu'elle génère et le récepteur sélectionne parmi les couches générées un flux (dans le cas du codage multirésolution), ou une collection de flux (dans le cas de sous-couches), qui satisfasse les contraintes liées aux contrôles de flux et congestion.

La deuxième approche, par sous-couches, étant celle qui cause le moins de congestion est la plus utilisée. Dans ce cas, on associe à chaque sous-couche une adresse multipoints IP particulière. Il suffit donc à un récepteur de souscrire aux adresses multipoints qui correspondent aux sous-flux qu'il désire recevoir. En cas de congestion, le récepteur peut choisir d'abandonner une sous-couche, en signifiant qu'il ne veut plus recevoir l'adresse multipoints correspondante. Une telle approche requiert cependant que soient définis des algorithmes spécifiques permettant aux récepteurs de déterminer la ou les sous-couches acceptables et ce, dynamiquement.

Les protocoles proposés cherchent à éviter les problèmes suivants :

- Le contrôle de congestion ne peut être efficace si des récepteurs derrière le même routeur agissent sans coordination. En effet, si un récepteur teste un niveau i non supporté par un lien qu'il partage avec un récepteur désirant lui tester un niveau $j < i$. Il y aura congestion et le récepteur testant le niveau j pensera qu'il en est la cause et retirera sa couche. Couche qui continuera à passer par l'engorgement car le premier récepteur continuera à la recevoir.

- Un long délai pour quitter une couche nuit à l'efficacité de l'algorithme. En effet, si un récepteur fait une tentative qui échoue , et que l'effet de cet échec dure longtemps, les autres récepteurs pourront mal interpréter leurs pertes et détruiront ainsi leur couche supérieure.

La première génération d'algorithmes n'avait qu'un seul objectif : réussir à coordonner les récepteurs pour éviter les problèmes décrits ci-dessus. La deuxième génération a eu un autre souci qui est d'être TCP-Friendly.

2 Quelques rappels

Dans cette section nous rappelons quelques notions utilisées dans ce document.

2.1 TCP-Friendly

Un protocole est dit TCP-Friendly s'il fournit un débit moyen semblable à celui de TCP. Ce dernier correspond à la formule de Mathis[1] donnée par [4] :

$$B_{tcp} \approx \frac{C.s}{\sqrt{p}\overline{RTT}} \text{ avec } C = \sqrt{\frac{3}{2}} = 1,22 \tag{1}$$

où s est la taille des paquets, RTT le round-trip-time, et p le taux de perte de paquets.

[1] Cette formule est applicable dans le cas de faibles pertes ($p \leq 16\%$) et considère un fonctionnement périodique en phase d'évitement de congestion.

2.1.1 Le cycle de TCP [1]

Le cycle de TCP commence après une perte et se termine par une perte, donc il y a une perte par cycle, ce qui peut se traduire par :

$$p = \frac{s}{B_{tcp}.Cycle},$$

où s est la taille des paquets et $Cycle$ la longueur du cycle tel que décrit ci-dessus. Par ailleurs, la formule (1) nous donne :

$$p = \frac{C^2.s^2}{B_{tcp}^2.\overline{RTT}^2} \tag{2}$$

ce qui donne au cycle une valeur de :

$$Cycle = \frac{1}{C^2}\frac{B_{tcp}.\overline{RTT}^2}{s} \tag{3}$$

2.1.2 L'organisation en couche

On suppose que la source utilise un transcodeur produisant l couches cumulatives. La source offre donc l groupes multicast ayant chacun un débit L_i $(i = 0, \ldots, l-1)$. Le récepteur a ainsi le choix dans une gamme de débit B_i $(i = 0, \ldots, l-1)$ avec:

$$B_0 = L_0 \text{ et } B_i = \sum_{j=0}^{j=i} L_j. \tag{4}$$

Et quand $B_{i+1} = 2.B_i$ on a $B_i = 2^i B_0$.
Dans toute la suite quand on parlera du débit B_i de la couche i on parlera du débit offert par l'ensemble des groupes allant du groupe (couche) de base au groupe i.

3 Protocoles existants

Dans cette section nous parlerons brièvement de deux protocoles de contrôle de congestion pour la transmission multicouches géré par le récepteur. Le premier est le grand classique RLM, né de la première génération, et le second, RLC, un protocole de la deuxième dont nous nous sommes inspirés pour réaliser notre protocole.

3.1 RLM [6]

RLM, pour Receiver-driven Layered Multicast, fait partie des premiers protocoles à adaptation au niveau du récepteur et ses seuls soucis étaient de faire coordonner les récepteurs et éviter les mauvaises interprétations. RLM, comme dit précédemment ne se préoccupait pas d'être TCP-Friendly.

3.1.1 Le fonctionnement de RLM

Un récepteur RLM, dans sa recherche de l'optimum, fait des souscriptions au niveau supérieur à des moments qu'il juge opportuns. Ces souscriptions spontanées sont appelées *"joint-experiment"*, anglicisme qui se traduire par tentatives d'adhésion.
Une tentative qui a échoué peut causer une longue congestion qui peut détériorer la qualité du signal délivré. Pour éviter la fréquence des tentatives d'adhésion, sans pour autant avoir un impact sur la convergence du protocole vers le bon débit ni sur la capacité du protocole de réagir aux changements de l'état du réseau, chaque récepteur dispose sur chaque niveau d'un temporisateur, appelé *join-timer*. Ainsi, quand la tentative échoue le récepteur en déduit que c'est une couche problématique et multiplie son join-timer par un facteur $\alpha > 1$.

Pour éviter les interférences, un récepteur RLM envoie un message à tous les récepteurs avant de tenter une adhésion ainsi ceux-ci ne risquent pas d'interpréter la tentative comme étant une arrivée de trafic perturbateur. Ce message d'annonce permet aussi aux récepteurs de reconnaître leurs couches problématiques. En effet, quand un récepteur reçoit un message annonçant la tentative à un niveau et qu'il sent une congestion juste après, il en déduit que ce niveau est problématique et augmente son join-timer en conséquence. C'est ce qu'on appelle l'apprentissage partagé.
Dans le cas d'un grand nombre de récepteurs, si chaque récepteur devrait attendre son tour pour faire une tentative d'adhésion la convergence au bon débit serait lente. Pour éviter ce genre de désagrément, RLM permet aux récepteurs prêts à adhérer à un niveau, de faire leur tentative à la réception d'un message d'annonce à condition que le niveau testé soit supérieur ou égal au niveau auquel ils veulent souscrire.
Dans sa recherche de l'optimum, RLM ne se soucie pas d'avoir un comportement semblable à celui de TCP.

3.2 RLC [7], [8]

RLC, "Receiver-driven Layered Congestion control", a été développé avec le souci d'être TCP-Friendly. Il a réussi à procurer un débit inversement proportionnel à $\sqrt{p}$ comme TCP. Ce protocole se distingue par deux comportements : les *points de synchronisation* et la *période d'investigation*. Vu que le fonctionnement de notre protocole ne se base que sur le premier élément, nous n'avons pas jugé nécessaire de parler des périodes d'investigation.

3.2.1 Points de synchronisation

Les points de synchronisation (SP) sont utilisés comme protocole implicite de coordination des récepteurs. Ils ne sont autres que des paquets marqués dans le flux des données qui à leur réception autorisent un récepteur à faire une adhésion. RLC est conçu pour fonctionner idéalement sur des couches à débits exponentiels, c'est-à-dire $B_{i+1} = \alpha B_i$. Les développements faits dans le cadre de ce protocole prennent $\alpha = 2$ comme fourni par le transcodeur LVT (décrit dans [2]). Il est suggéré que quand un récepteur fait une tentative d'adhésion, tous les récepteurs dont le niveau est inférieur ou égal au sien en fassent une. Ainsi, quand une congestion résulte d'une tentative d'adhésion à un niveau non supporté par un goulot, les récepteurs (se trouvant derrière ce dernier) non responsables de cet engorgement, suppriment la couche récemment ajoutée, sauvegardant ainsi toutes les couches acquises avant la dernière tentative. Pour mettre en oeuvre cette approche, il faut que les SP d'un niveau soit un sous-ensemble de ceux du niveau hiérarchiquement inférieur.

La souscription à un niveau supérieur se fait donc uniquement aux points de synchronisation, alors que la résiliation peut se faire à n'importe quel moment (détection de perte).

3.2.2 Modèle analytique

Pour calculer la relation entre le débit et le taux de perte, les développeurs de RLC ont utilisé un modèle quasi stable, dans lequel le récepteur oscille entre deux niveaux de souscription i et $i+1$, et pour lequel la destruction de la couche $i+1$ se fait au premier paquet perdu . Une nouvelle souscription à ce niveau ne pourra se faire qu'après le premier point de synchronisation utile, c'est-à-dire après un intervalle de temps complet t_i (distance entre 2 SPs du niveau i) sans qu'il y ait de perte.

Soient :

- s la taille des paquets,
- $W = \frac{B_0 t_0}{s}$ le nombre de paquets du niveau 0 envoyés entre deux SPs de ce même niveau.
- $\alpha.t_i$ le temps passé sur le niveau $i+1$.

En optant pour un facteur multiplicateur valant 2 ($B_i = 2^i B_0$ et $t_i = 2^i t_0$) et en supposant qu'il y a une seule perte par cycle, on obtient un débit moyen R vallant [7] :

$$R = \frac{s}{t_0}\sqrt{\frac{W}{p}}c'(\alpha) \text{ où } c'(\alpha) = \frac{\sqrt{2+\alpha+\lfloor\alpha\rfloor}}{\lfloor 2+\alpha\rfloor} \tag{5}$$

qui est inversement proportionnel à $\sqrt{p}$.

La réponse de RLC peut être ajustée en agissant sur les paramètres apparaissant dans l'équation (5). t_0 influence directement la vitesse de réponse aux changements de conditions de RLC, mais il ne peut pas être très faible car un trop faible t_0 ferait de l'algorithme un algorithme étouffant, chose qu'on ne cherche pas à avoir.

Les développeurs de RLC considèrent qu'un t_0 de l'ordre d'une seconde est une bonne valeur, car, les valeurs communes des round-trip-times de TCP appartiennent à l'intervalle [0.1, 1s]. (1 seconde représente le maximum "commun" du RTT de TCP). Ainsi RLC est développé pour être équitable vis-à-vis de TCP, quand celui-ci a un RTT valant 1 seconde.

3.2.3 L'équité fournie par RLC

RLC a été développé pour avoir le même comportement qu'une source TCP ayant un RTT égal à une seconde. Ce qui fait de lui un protocole vulnérable dans le cas de petits RTT (cas de petites dizaines de millisecondes), et étouffant quand le RTT est supérieur à 1 seconde. Un autre point faible de RLC est qu'il opère sur des cycles de longueurs fixes en fonction du niveau d'appartenance quelle que soit la session. Ceci implique que dans le cas de plusieurs sessions le point d'équilibre est un point où les différentes sessions opèrent sur des cycles égaux. Ce qui se traduit par le même nombre de niveaux pour les différentes sessions, mais pas par des débits égaux lorsque les couches de base ont des débits différents.

4 Le protocole proposé

Nous procéderons en deux étapes. Un premier protocole visera essentiellement à être TCP-Friendly. Nous montrerons qu'il n'atteint toutefois pas l'objectif d'équité entre récepteurs. Nous atteindrons celui-ci en ajoutant des points de synchronisation de façon appropriée.

4.1 La longueur des cycles

On s'est proposé d'étudier le comportement d'un protocole multipoints dont les récepteurs opèrent sur des cycles de longueur équivalente à $Cycle$ (équation 3). Pour ce faire, chaque récepteur doit, selon (3) garder son niveau i pendant un temps :

$$t_i = \gamma \frac{B_i}{s}.\overline{RTT}^2 \tag{6}$$

avant d'ajouter une couche (B_i est défini dans la formule (4) et s est la taille des paquets), et de détruire une couche dès que le taux de perte est supérieur au seuil p de la formule (2).

Un des cas les plus défavorables est le cas de petits RTT, la source TCP a tendance à étouffer le trafic contrôlé par le récepteur. Une expérience consistant en la concurrence de TCP avec un trafic contrôlé par le récepteur opérant sur des cycles de longueur t_i[2] avec $\gamma = 1$ sur un lien de délai 20 ms, donne le partage illustré sur la figure 1. Le récepteur réussit à avoir exactement le même partage de la ligne que TCP.

4.2 La nécessité de coordination

Avec un protocole de la sorte nous avons réussi à être TCP-Friendly, mais avons-nous réussi à surmonter les problèmes d'interférence ?
A priori non, les récepteurs adhèrent aux couches ou les quittent sans aucune coordination, ce qui, dans le cas de plusieurs récepteurs derrière un même goulet, peut nuire à la convergence de ceux-ci.

Il a donc fallut penser à un protocole qui opère sur des cycles de longueur t_i (de l'équation 6) en coordonnant les différents récepteurs. Nous nous sommes inspirés de RLC pour mettre au point un protocole se basant sur les points de synchronisation pour la coordination des récepteurs.

[2] En choisissant $B_0 = 8$ kbps et $B_{i+1} = 2.B_i$ et des paquets de taille 1000 bytes on obtient $t_0 = 1,6$ ms et $t_i = 2^i.t_0$.

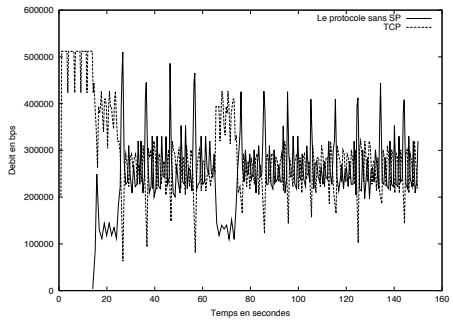

Figure 1: Un récepteur du protocole sans points de synchronisation et une source TCP

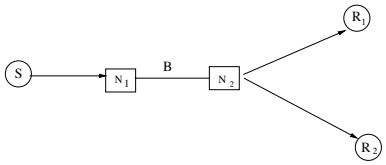

Figure 2: 2 récepteurs et une source

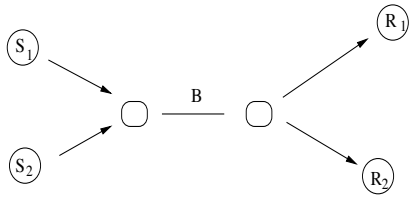

Figure 3: 2 sources 2 récepteurs

4.3 Le choix de la distance entre les points de synchronisation

Un récepteur ne tente d'adhérer au niveau suivant qu'à la réception d'un point de synchronisation à condition que le temps passé sur un niveau i soit d'au moins t_i. Donc, ces points de synchronisation ne doivent pas être trop espacés sinon le temps de convergence du récepteur risque d'être long, et pire encore si le trafic est en compétition avec TCP, celui-ci montera plus vite et étouffera notre trafic. Le choix d'une petite distance entre les points causerait des périodes de non réaction (voir section 4.4) trop fréquentes et conduirait à l'effet inverse qui est loin d'être souhaité. Donc il a fallut trouver un juste milieu. On pourrait donner à cette distance une valeur fixe T quelle que soit la session, comme le fait RLC, et le récepteur à l'arrivée d'un point de synchronisation adhérera au niveau suivant s'il a passé un temps supérieur à t_i. Ceci risque de conduire, dans le cas de très faibles RTT, deux sessions différentes [3] à opérer sur des cycles de même longueur (car comme $t_i \ll T$ les adhésions dans les deux cas se feront après T). Ce qui fournirait une équité en terme de niveaux et non en terme de débit. C'est pour cette raison que nous avons choisi un intervalle fonction du débit de la couche de base. Soit

$$T = \gamma' \frac{B_0}{s}$$

cet intervalle. La figure 4 montre deux cas de montées dans les niveaux en mettant en évidence la longueur de l'intervalle T et celle du cycle t_i. Le premier cas est celui d'un cycle relativement court par rapport à T, le second celui d'un cycle plus long.

Les points de synchronisation dans RLC sont conçus de telle manière que la distance entre eux grandisse en fonction du niveau de leur appartenance, ce qui n'est pas le cas dans le protocole que nous proposons. La distance est la même quel que soit le niveau. Ainsi, quand il s'agit de petits RTT le récepteur pourra monter dans les niveaux à chaque T, et dans le cas inverse, celui-ci ne pourra atteindre rapidement le niveau optimal, car sa montée dans les niveaux est conditionnée par l'écoulement d'un temps minimum de t_i.

[3] On dit de deux sessions qu'elles sont différentes si le débit de leur couche de base est différent.

4.4 La machine d'états

Le protocole proposé, RLS Receiver-driven Layered multicast with Synchronisation points, peut être présenté sous forme d'une machine à trois états (figure 5) : (S) l'état stable, (A) l'état d'ajout et (D) l'état de destruction.

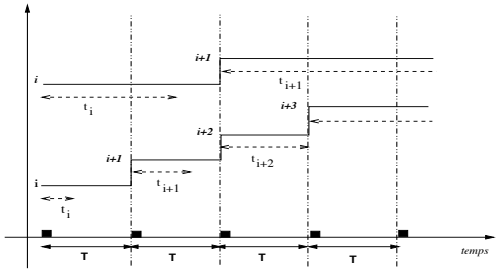

Figure 4: Illustration des intervalles t_i et T

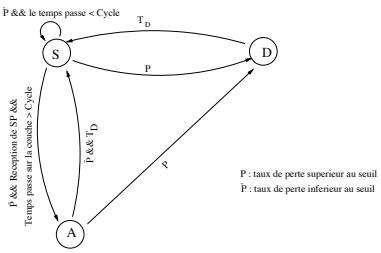

Figure 5: La machine d'états de RLS

Le récepteur recevant un point de synchronisation devra calculer le temps qu'il a passé sur son niveau i. Si ce temps est supérieur ou égal à t_i de la formule (3), le récepteur adhère au niveau supérieur, sinon il reste sur son niveau et rentre dans une période de non réaction durant laquelle le récepteur ne réagit pas aux pertes. Cet intervalle de temps est nécessaire à l'évitement de mauvaises interprétations. En effet, sans la présence de cette période, le scénario suivant est susceptible de se produire :

Soient deux récepteurs R_1 et R_2 derrière le même goulot, ayant respectivement un niveau i et j, avec, $i < j$ et j le niveau maximal toléré par le lien partagé. Supposons qu'à la réception d'un SP le récepteur R_2 ait déjà passé un temps supérieur ou égal à t_j^2 [4] sur son niveau, et que le récepteur R_1 soit sur son niveau depuis un temps inférieur à t_i^1, alors le récepteur R_2 fera sa tentative qui congestionnera la ligne. Le récepteur R_1 sentant la congestion, réagira en détruisant sa couche i. Cette destruction est inutile dans la mesure où cette couche, toujours reçue par R_2, continuera à passer par le lien partagé.

Si en dehors de la période de non réaction, le récepteur sent une perte, il calcule le taux de cette dernière, qui si il est supérieur au seuil défini dans l'équation (2) détruit sa couche.
Pour éviter les chutes en cascade, après une destruction de couche le récepteur ne réagit plus aux pertes pendant un certain temps. Ceci a pour but de prendre en considération le temps nécessaire au routeur le plus proche pour réagir à la demande d'élagage.

On pourrait se dire que dans le cas d'une congestion sévère, le récepteur aurait tout intérêt à détruire plusieurs couches à la fois au lieu d'une seule. Mais, nous pensons que vu la quantité de trafic TCP dans les réseaux, un lien qui connaît une congestion sévère a de fortes chances d'être traversé par plusieurs sessions TCP qui diminueraient leur débit simultanément, ce qui libérerait la ligne et qui rendrait le retrait de plus d'une couche inutile.

Pour calculer le round-trip-time, nous avions pensé en un premier temps à faire un ping au début de la session et l'utiliser tout au long de l'échange de données. Cette idée a vite été abandonnée car les résultats dépendraient énormément du moment de commencement. Nous avons donc opté pour faire un ping du récepteur à la source aux transitions A-S et D-S, et faire un lissage exponentiel. Ainsi le RTT vaut :

$$RTT = \alpha.ping + (1-\alpha)RTT \text{ avec } \alpha = 0,25.$$

[4] $t_u^k = \gamma\overline{RTT_k}^2.\frac{B_u}{s}$ où RTT_k est le round-trip-time du récepteur k

5 Simulations

Pour toutes nos expériences nous utilisons le simulateur de réseau NS [5].
Un récepteur reste sur son niveau i pendant :

$$j.\gamma' \frac{B_0}{s} \geq \gamma \frac{B_i}{s} RTT^2 \text{où } j \text{ est un entier}$$

ce qui revient à dire qu'il y reste pendant :

$$\gamma' \frac{B_0}{s} \lceil \frac{\gamma}{\gamma'} 2^i RTT^2 \rceil.$$

En ajustant le γ' et le γ on peut obtenir d'assez bons résultats mais il faut décider de privilégier le cas de petits délais ou grands délais. Suite à plusieurs expériences nous avons trouvé que la mise des paramètres à 1 ($\gamma = \gamma' = 1$) donne une assez bonne moyenne pour tous les cas. Les résultats des simulations donnés ci-après sont les résultats faits avec les valeurs données ci-dessus.

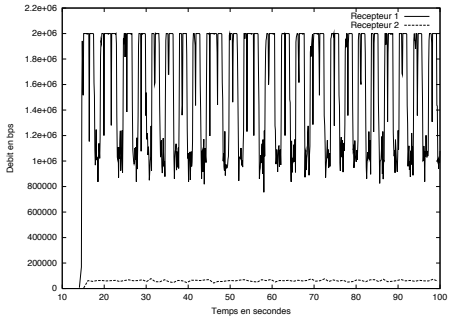

Figure 6: 2 récepteurs du protocole sans points de synchronisation

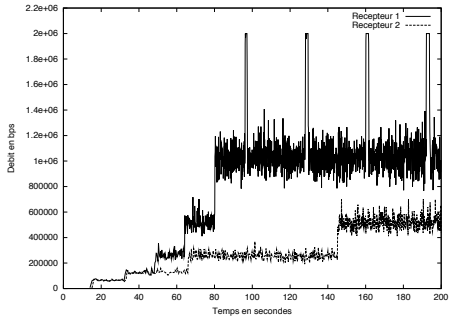

Figure 7: 2 récepteurs RLS

5.1 Cas de plusieurs récepteurs

Pour faire apparaître la nécessité des points de synchronisation dont nous avons parlé dans la section 4.2, nous avons fait une petite expérience où on comparait les deux protocoles, avec et sans points de synchronisation. Nous avons utilisé la topologie simple de la figure 2, où deux récepteurs de différents RTT étaient derrière le même goulet. Sans point de synchronisation le récepteur ayant le plus petit RTT piétine le second. Le résultat est celui illustré à la figure 6. Celui de la figure 7 est celui du protocole avec points de synchronisation; chaque récepteur y atteint son niveau optimal sans gêner l'autre dans sa quête.

5.2 RLS face à TCP

Nous utilisons la topologie de la figure 3 (page 6) pour tester les comportements des deux trafics, avec une source TCP et une source RLS. Nous n'illustrons ici que les cas extrêmes[5] qui testés avec RLM et RLC donnaient de mauvais résultats. La figure 8 montre le partage de la ligne dans le cas d'un petit délai de propagation et dans le cas d'un grand délai. Le partage de la ligne dans les deux cas est acceptable, aucun trafic n'est étouffé. Donc la relation maître-esclave n'existe pas entre RLS et TCP. Dans nos simulations, nous avons utilisé une couche de base de débit 8 Kbps, avec des paquets de 1000 bytes. Ceci donnait à T la valeur de 1 seconde. Pour t_i la valeur dépend du RTT[6]. Dans le premier cas, RTT vaut 40 ms et par suite $t_0 = 1,6\,\text{ms}$. Dans le second, RTT vaut 2 secondes et $t_0 = 4\,\text{s}$. La relation $t_i = 2^i t_0$ reste la relation liant t_i à t_0.

[5] Par cas extrêmes, nous entendons le cas où TCP commence avant l'autre trafic sur un lien à petits délais de propagation, et le cas où TCP commence après que le récepteur ait atteint son optimal sur un lien à grand délai.

[6] En omettant le temps de transmission et de bufferisation dans les files d'attente, nous pouvons dire que RTT = 2 * (le délai de propagation).

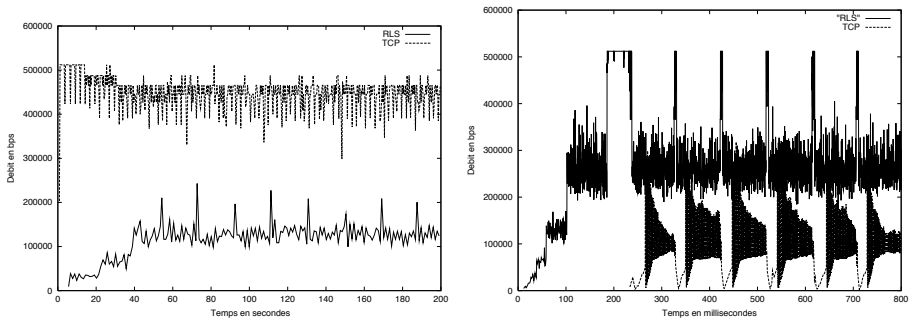

Figure 8: A gauche le cas d'un délai de 20ms; à droite le cas 1000ms

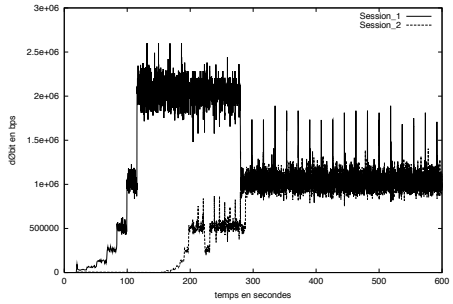

Figure 9: 2 sessions différentes de RLS

Pour faire apparaître la dépendance en fonction du délai de propagation nous avons effectué plusieurs expériences où on ne changeait que le délai. Nous l'avons fait pour RLM et RLC aussi (pour plus de résultats concernant RLM et RLC se référer à [3]). Le cycle T choisi ici vaut 8 secondes. Le résultat est illustré sur la figure 10. La courbe de RLS est celle qui ne tend pas vers zéro dans le cas des grands délais, ce qui signifie que RLS n'est pas agressif envers TCP dans ce cas, où RLM et RLC écrasent TCP. Dans le cas inverse, c'est-à-dire le cas des petits délais, RLS ne se soumet pas à TCP.

Remarque : En choisissant les valeurs du débit de la couche de base et de la taille des paquets de manière à ce que le rapport $\frac{B_0}{s}$ soit petit, nous pouvons nettement améliorer les résultats obtenus, pour le cas des petits délais. En effet, quand $T \gg t_i$, la montée dans les niveaux est ralentie par la longueur du cycle T.

5.3 Cas multi-sessions

RLC ne parvenait pas à procurer l'équité dans le cas multi-sessions pour les raisons évoquées dans la section 3.2.3. Le protocole que nous proposons, à savoir RLS, arrive à procurer l'équité telle qu'on la souhaite. La figure 9 montre le partage de la ligne par deux sessions dont l'une a une couche de base quatre fois plus grande que celle de l'autre. Pour être sûr du comportement, nous n'avons pas lancé les deux sessions simultanément, mais avons attendu que celle à grande couche de base ait atteint sa couche optimale. La figure montre la destruction de la couche supérieure pour fournir le meilleur partage.

6 Conclusion

Dans cet article, nous avons proposé un protocole de contrôle de congestion, pour une transmission multi-points en couches ayant pour but d'assurer

(a) une équité vis-à-vis de TCP,

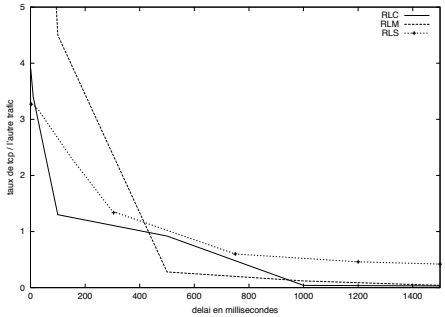

Figure 10: RLM, RLC et RLS en fonction des délais de propagation

(b) une bonne coordination entre récepteurs par un placement judicieux de points de synchronisation,

(c) et une équité entre sessions en terme de débit (plutôt qu'en terme de couches).

Nous avons simulé notre protocole dans différentes situations, et les résultats montraient que les équités intra et inter-sessions étaient garanties et que l'équité envers TCP était garantie à un petit facteur près.

Nous envisageons cependant d'améliorer encore les points suivants :

- Le calcul du RTT est fait par des pings fréquents sur la source, ce qui, dans le cas d'un grand groupe, risque d'inonder la source. Nous pensons pouvoir utiliser les estampilles des paquets pour estimer le RTT, ce qui réduirait le nombre de pings faits par le récepteur sur la source.
- Nous chercherons à rendre le ratio de TCP sur RLS plus proche de 1, indépendemment des débits de la couche de base et de la taille des paquets.
- La machine d'états peut être améliorée de manière à réduire les tentatives d'adhésion infructueuses en ne se basant plus uniquement sur les pertes, mais aussi sur les délais de bufferisation.

Références

[1] Omar Aït-Hellal, Lidia Yamamoto, and Guy Leduc. Cycle-based tcp-friendly algorithm. In *Proceedings of IEEE Globecom'99*, Rio de Janeiro, December 1999. IEEE Press.

[2] Gianluca Iannaccone and Luigi Rizzo. A layered video transcoder for videoconference applications. Technical report, Université de Pise, Italy, July 1999.

[3] A Legout and W Biersack. Pathological behaviors for rlm and rlc. In *Proceedings of NOSSDAV'2000*, Chapel Hill, North Carolina, USA, June 2000.

[4] M. Mathis, J. Semke, Mahdavi, and T. Ott. The macroscopic behavior of the tcp congestion avoidance algorithm. *Computer Communication Review*, 27(3), July 1997.

[5] S. McCanne and S. Floyd. *The LBNL Network Simulator.* Lawrence Berkeley Laboratory, Software on-line (`http://www-nrg.ee.lbl.gov/ns/`).

[6] S. McCanne, V. Jacobson, and M. Vetterli. Receiver-driven layered multicast. In *Proceedings of ACM SIGCOMM'95*, pages 117–130, Palo Alto, California, 1995.

[7] Luigi Rizzo, Lorenzo Vicisano, and Jon Crowcroft. The rlc multicast congestion control algorithm. Submitted to IEEE Network - special issue on multicast.

[8] Lorenzo Vicisano, Jon Crowcroft, and Luigi Rizzo. Tcp-like congestion control for layered multicast data transfer. In *Proceedings of IEEE INFOCOM'98*, San Francisco, CA, March 1998.