Impact des changements de chemins sur les flux TCP

Jérémy OTTO

Encadré par
Cristel PELSSER, Stéphane CATELOIN et Pascal MERINDOL

2015-2016

Contents

1 Remerciements

Avant d'entrer dans le vif du sujet, je voudrais remercier l'Université de Strasbourg pour la qualité de ses enseignements et l'opportunité qui m'a été offerte de travailler au sein d'une équipe de recherche lors de ce stage.

Je voudrais également remercier mes tuteurs, Cristel PELSSER, Stéphane CATELOIN et Pascal MERINDOL pour leurs conseils et pour avoir répondu à mes nombreuses questions.

Je tiens également à remercier ma famille et mes amis pour leur soutien.

2 Contexte

2.1 Présentation du stage

Dans le cadre de mon dernier semestre de Master Réseaux Informatiques et Systèmes Embarqués à l'Université de Strasbourg, j'ai décidé de m'orienter vers un stage au sein d'un laboratoire de recherche en informatique.

Ayant eu un premier aperçu de métier de chercheur en informatique, et plus spécialement dans les réseaux fixes, lors de ma première année de Master, j'ai décidé de réitérer l'expérience. En effet, j'ai été amené à travailler avec un de mes tuteurs, Pascal MERINDOL, sur le *Fingerprinting de routeurs IP de transit*. Cela m'a permis de découvrir plus en profondeur le rôle et le travail d'un chercheur mais aussi de découvrir de nouvelles façons de travailler ainsi que parfaire les connaissances acquises lors de mon cursus. Lors d'un travail de recherche, nous avons une bonne connaissance de l'état actuel des choses mais surtout de certaines problématiques. Le cheminement permettant de trouver une solution à ce problème peut être plus complexe que ce à quoi nous nous attendons. Il n'est pas rare de prouver qu'un problème n'est pas solvable et c'est aussi pour cette raison que j'ai souhaité réaliser ce stage en laboratoire : le déroulement d'un stage recherche étant très différent d'un stage en entreprise.

Durant toute la durée du stage, nous avons été amenés à échanger très régulièrement, par exemple sur la progression de mes recherches.

Dans ce but, et afin de garder une trace écrite de toutes mes actions, j'ai eu accès à un Redmine. Chaque papier que je lisais et chaque outil que je découvrais y était décrit. De plus, chaque présentation, rapport ou code source était partagé via un dépôt Git. De fait, tous mes travaux étaient sécurisés et accessibles par l'ensemble de l'équipe.

Enfin, nous organisions de nombreuses réunions afin d'avoir un suivi constant des découvertes et des discussions sur les voies à explorer. Cela m'a permis d'améliorer mes présentations ainsi que mon expression orale.

2.2 Présentation de la structure et de l'équipe

Le stage s'est déroulé au sein du Laboratoire ICUBE de l'Université de Strasbourg, situé au sud, au cœur du Parc d'Innovations d'Illkirch-Graffenstaden. L'Université, quant à elle, dispose également de plusieurs autres campus comme ceux de Cronenbourg ou de l'Esplanade. Cet emplacement stratégique permet des partenariats avec des entreprises proches, telles qu'Alcatel Lucent dont certains membres interviennent en Master.

Plus que des partenaires locaux, le laboratoire a également créé des liens avec des industriels, tels que la SNCF ou encore Renault, ou encore des groupes internationaux comme Cisco.

Le laboratoire ICUBE, fondé en 2013 et actuelement dirigé par le professeur Michel De Mathelin, compte 14 équipes de recherche. Ces équipes sont regroupées par domaine au sein de 4 départements :

- Informatique Recherche
- Imagerie, Robotique, Télédétection et Santé
- Electronique du solide, Systèmes et Photoniques
- Mécanique

C'est au sein du département Informatique Recherche que s'est déroulé mon stage. Ce département est constitué de 6 équipes. La première, Informatique Géométrique et Graphique, travaille entre autres sur des sujets tels que la spécification, les contraintes mais aussi les preuves en géométrie. La deuxième équipe, Informatique et Calculs Parallèles Scientifiques, travaille sur des grilles ainsi que les optimisations de compilation parmi d'autres sujets. L'équipe Science des Données et des Connaissances est spécialisé dans les connaissances et les technologies sémantiques. La quatrième équipe, Systèmes Complexes, Bioinformatique Translationnelle, travaille sur la résolution de problèmes en se basant sur des méthodes inspirées par la nature, telles que la programmation génétique. La cinquième équipe est spécialisée dans le traitement d'images médicales ou encore astronomiques. Il s'agit de l'équipe Modèles, Images et Vision. Enfin, l'équipe Réseaux, dirigée par le professeur Thomas NOEL se penche sur des problèmes aussi bien filaires que sans-fils. C'est au sein de cette équipe, composée de 14 personnes, que j'ai effectué mon stage. Les problèmes qui y sont traités sont en lien avec le routage multipath, le partage de charge mais aussi les topologies et simulations réseaux ou encore les capteurs sans-fils.

3 Introduction

Dans un souci de fiabilité ou dans le cadre de l'ajout d'un nouveau point d'interconnexion, les réseaux peuvent être amenés à évoluer et voir le nombre de liens les reliant augmenter. Cela étant, le nombre de chemins disponibles pour aller d'une source à une destination peut croître rapidement. Ce phénomène peut s'appliquer aussi bien aux réseaux privés que pour l'ensemble d'internet. Cela peut être facilement vérifié en utilisant un outil tel que Paris-Traceroute (Augustin et al. [2]) dont le but est de mettre en évidence la présence de multi-chemin.

L'augmentation du nombre de ces chemins, dont les caractéristiques de bout-en-bout peuvent varier, peut ne pas apporter que des avantages, comme nous allons le voir dans ce travail.

Pour commencer, il faut savoir que la majeure partie des comunications sur internet se font actuellement en utilisant le protocol TCP et c'est pour cela que nous nous y intéressons. Un impact, aussi faible soit-il, impactera largement les communications.

Pour étudier ce phénomène, nous allons comencer par étudier les principes de bases tels que les types de partage de charge. Nous approfondirons également nos connaissances sur TCP. Par la suite, nous nous intéresserons aux problmes qui peuvent se poser et nous analyserons les études actuelles sur le ujet. Après description et interprétation des résultats de nos simulations, nous tenterons, si nécessaire, de faire varier certains paramètres afin de limiter l'impact des changements de chemins sur les flux TCP. Finalement, nous étudierons nos nouveaux résultats et en tirerons nos conclusions et les perspectives de nos travaux.

4 Etat de l'art

4.1 Partage de charge

4.1.1 Description et intérêt

Nous avons énoncé que pour une même destination, les données peuvent emprunter différents chemins. Ces chemins peuvent de fait avoir des caractéristiques différentes (débits, délais, ...). Certaines applications peuvent avoir des besoins spécifiques concernant ces caractéristiques. En effet, des applications de VoIP ne nécessitent pas beaucoup de bande passante mais peuvent rapidement être impactées par des fluctuations de délais. Ce type de flux est considéré comme un flux souris, comme définit entre autres dans le papier de Guo et al.[6] . A l'inverse, les flux dits éléphants correspondent à ceux nécessitant une bande passante bien plus importante comme lors du téléchargement d'un fichier de taille importante.

De manière générale, au moment où un paquet entre dans un nœud, ce dernier a connaissance de sa destination. Précédemment, il aura échangé sa table de routage avec les nœuds auxquels il est relié. Cette table est une structure mettant en relation une liste de destinations et les moyens de les atteindre. Dans notre cas, il s'agit d'un préfixe réseau et une liste de liens par lesquels les paquets peuvent être redirigés. Disposant d'un ou plusieurs liens pour transférer le paquet vers sa destination, le nœud doit alors choisir vers lequel le paquet sera orienté, en le faisant traverser un ordonnanceur. Différents types d'allocation de chemins, expliqués plus tard dans cette section, peuvent être utilisés par cet ordonnanceur pour prendre sa décision, selon sa configuration.

Pour en revenir aux flux éléphants et souris, notons que s'il prennent le même chemin cela peut alors poser des problèmes. En effet, les flux éléphants peuvent remplir progressivement les files d'attentes, augmentant alors le délai entre l'arrivée d'un paquet dans un routeur et son départ. Dans le cadre d'un flux de VoIP, ce phénomène peut créer des désagréments.

Afin d'éviter ces dégradations, il est possible de transférer une partie des données sur un lien plutôt qu'un autre. C'est entre autres pour ce cas de figure que le partage de charge est intéressant.

Pour mieux comprendre l'intérêt, intéressons-nous à un cas général. Pour ce faire, nous allons utiliserons une topologie typique où nous avons plusieurs liens entre deux mêmes routeurs et plus précisément le cas où nous avons 2 liens, partageant les mêmes caractéristiques, à savoir une bande passante de 10 Mo/s. Les autres caractéristiques n'ont que peu d'importance dans le cas de cet exemple. Ajoutons que nous avons deux flux souhaitant traverser ces routeurs, dans le même sens, au même moment.

Sans partage de charge, deux cas peuvent se poser :

- Chaque flux utilise un lien qui lui est propre
- Les deux flux se partagent un même lien

Dans le premier cas, chaque flux va pouvoir utiliser la bande passante totale du lien qu'il emprunte, à savoir 10Mo/s. Dans le second cas, cette bande passante sera partagée entre les deux flux. Chaque flux pourra alors profiter d'une bande passante allant jusqu'à 10 Mo/s.

Sur la figure 1, dont les axes des absisses et des ordonnées représentent la bande passante utilisée par respectivement le flux A et le flux B, ces deux cas sont représentés par le carré bleu. S'ils partagent le

même lien, la bande passante allouée au flux A n'est plus disponible pour le flux B. Dans le cas d'un chemin par flux, les deux peuvent alors profiter des 10 Mo/s.

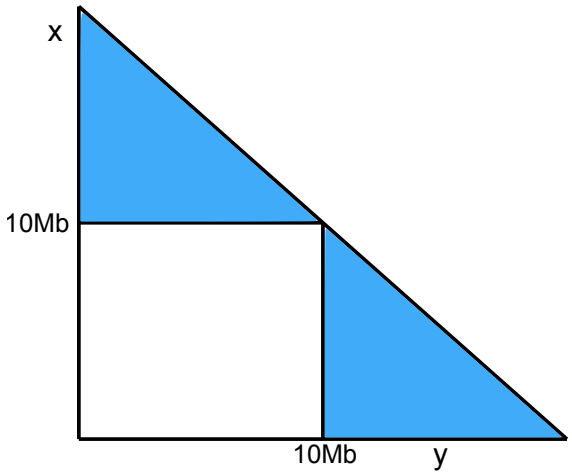

Figure 1: Mise en évidence de l'intérêt du partage de charge

Avec du partage de charge, non seulement les flux vont pouvoir avoir un qui leur est propre, mais ils pourront emprunter le second chemin nécessaire, permettant aux deux flux d'atteindre des débits jusqu'à 20 Mo/s soit la somme des deux liens. Bien évidemment, il n'est pas possible que les deux profitent de tels débits. Dans le cas d'un partage de charge de ce type, la zone blanche sur la figure est ajoutée à la zone bleue, déjà accessible.

De fait, l'intérêt du partage de charge est une possible augmentation de la bande passante globale entre deux nœuds à faible coût. En effet, au lieu de changer le matériel actuellement en place (routeurs, fibre, ...) par du matériel plus puissant et plus cher, l'administrateur pourra opter pour une augmentation du nombre de lien entre les deux nœuds. Cela implique cependant la maintenance de plusieurs liens en même temps. Enfin, il sera alors possible d'allouer les ressources de manière plus fine en transférant, par exemple, une partie des flux éléphants sur un lien dédié. Il faut toute fois noter que les algorithmes utilisés peuvent augmenter, entre autres, la charge CPU du routeur.

4.1.2 Les types d'allocation de chemins

Dans cette partie, nous allons nous intéresser à différents types d'allocation de chemins, leur principe de fonctionnement, leurs avantages mais aussi les risques que peuvent engendrer leur utilisation. Nous nous limiterons dans ce travail aux allocations par flux et par parquet. Le premier est le type d'allocation classique, principalement utilisé dans la qualité de service où l'on attribue un chemin à un type de flux selon ses besoins. Le deuxième type d'allocation, par paquet, est un cas plus atypique car il ne prend pas directement en compte les caractéristiques des liens ou des flux qu'il doit ordonnancer. D'autres types d'allocation, par rafales par exemple, existent mais nécessitent du matériel plus spécifique pour les implémenter.

4.1.2.1 Par flux

Un algorithme d'allocation de chemins par flux applique une *fonction de hachage* sur certains champs des paquets. Si plusieurs paquets obtiennent le même résultat, ils font partie d'un même flux. En effet, cette fonction est généralement appliquée à un 5-tuple (protocol, IP-source, IP-destination, port-source, port-destination). Il est impossible qu'un tel tuple génère une allocation de prochain saut différente. Les paquets correspondant à un même flux empruntent ainsi le même chemin de bout-en-bout.

Une allocation par flux est intéressante dans le cas où l'on veut supprimer les risques de déséquencement des paquets : les paquets suivent un ordre strict, aucun paquet ne peut en doubler un autre. Malheureusement, cela peut amener à un partage sous-optimal de la bande passante. En effet, un tel partage de charge peut amener à faire passer l'ensemble des flux éléphants sur un même chemin et l'ensemble des flux souris sur un autre et donc à des congestions qui auraient pu être évitées.

Un exemple de comportement induit par l'utilisation d'une allocation par chemin est présenté en figure 2. Les paquets entre sur l'ordonnanceur dans l'ordre de leur numéro de séquence et sont transférés dans les files d'attentes (les paquets sont stockés dans les files d'attentes en partant de la droite). Chaque couleur de paquet correspond à un flux précis. En sortie de l'ordonnanceur, l'ensemble des paquets d'un même flux empruntent le même lien et leur ordre est respecté.

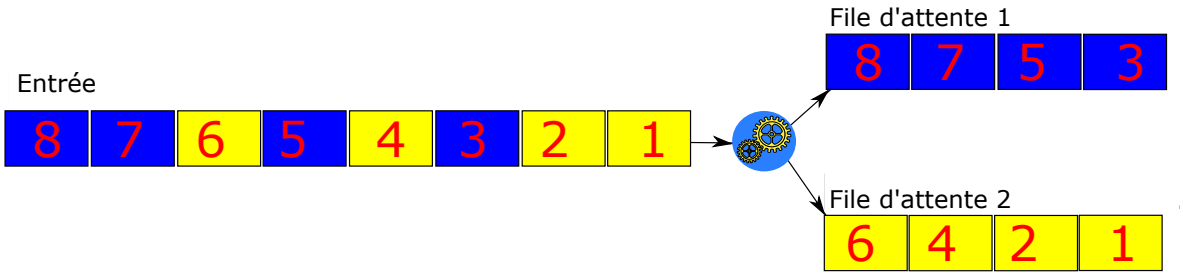

Figure 2: Allocation par flux

Notons tout de même que, suite aux essais que nous avons réalisés sur du matériel Cisco et Juniper, nous avons pu observer qu'en l'absence de normalisation, chaque fabriquant voire modèle pouvait

utiliser une fonction de hachage ou un tuple différent. Dans certains cas, la documentation d'un routeur ne correspondait pas à la fonction implémentée.

4.1.2.2 Par paquet

L'allocation par paquet traite chaque paquet indépendamment et en fonction de l'algorithme d'ordonnancement utilisé, un tourniquet par exemple.

Prenons pour exemple des paquets arrivant sur un routeur utilisant un algorithme de **Round Robin**. Admettons que ce routeur dispose de 3 liens en sortie. Dans ce cas de figure, le premier paquet est envoyé sur le premier lien, le deuxième sur le deuxième et enfin, le troisième sur le dernier chemin. Finalement, l'ensemble des liens disponibles ayant reçu un paquet, le suivant sera envoyé à nouveau sur le premier chemin, tel un *modulo* sur le nombre de chemins. Une version pondérée de cet algorithme peut amener plusieurs paquets consécutifs sur un même lien à chaque itération. Cela permet, par exemple, de favoriser les échanges sur un lien dont la bande passante est supérieure. Les ressources peuvent tendre à être mieux réparties. Néanmoins, le fait que des paquets qui se suivent empruntent des chemins différents peut poser certains problèmes. Comme énoncé précédemment, les paquets disposent d'un *numéro de séquence* afin de savoir dans quel ordre les données ont été transmises. Lors de l'utilisation de ce type d'allocation, les paquets sont transmis aléatoirement sur les liens, sans garantir l'ordre des données à l'arrivée. En effet, les caractéristiques fixes, comme la latence et la capacité, mais aussi dynamiques, comme la bande passante résiduelle, peuvent induire des déséquencements.

Un exemple de comportent d'un tel algorithme est présenté en figure 3. De même que précédemment, les paquets sont stockés en file d'attente en partant de la droite et arrivent suivant un ordre strict sur l'ordonnanceur. En sortie, on peut remarquer que les paquets d'un même flux empruntent des chemins différents.

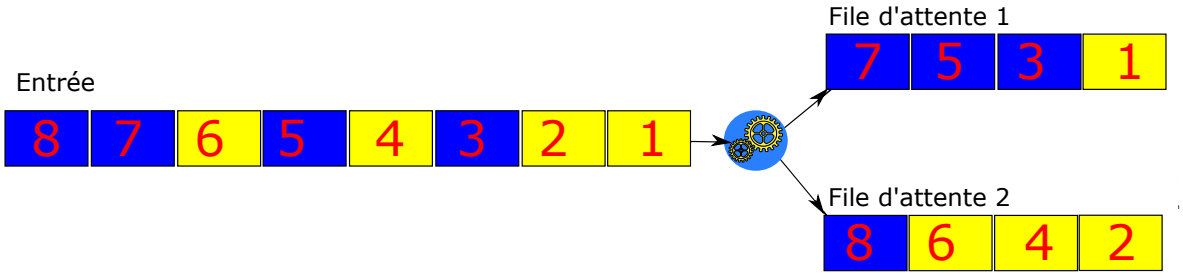

Figure 3: Allocation selon un Round Robin

4.2 Raisons des changements de chemins

En plus des algorithmes de partage de charge, les changements de chemins peuvent survenir sans que nous ayons la possibilité de les contrôler. Nous allons maintenant étudier deux raisons qui peuvent engendrer des changements de chemins.

4.2.1 Liés à la topologie

Lorsqu'un lien n'est plus exploitable (lors d'une panne par exemple) ou, au contraire, lors de l'établissement d'un nouveau lien, l'algorithme de partage de charge doit s'adapter en partageant le trafic sur les liens nouvellement disponibles. Cette répartition dépend des caractéristiques des liens ou encore des paramètres définis par l'administrateur. Sur la figure 4, le lien rouge est celui qui est devenu inexploitable, les routeurs sont représentés par des carrés bleus. Avant la panne, chaque lien se partage 33% du trafic total entre les nœuds S et D. Une fois le premier routeur informé de la panne, la moitié du trafic du lien rendu inexploitable est transféré sur le premier lien, la deuxième moitié sur le troisième lien.

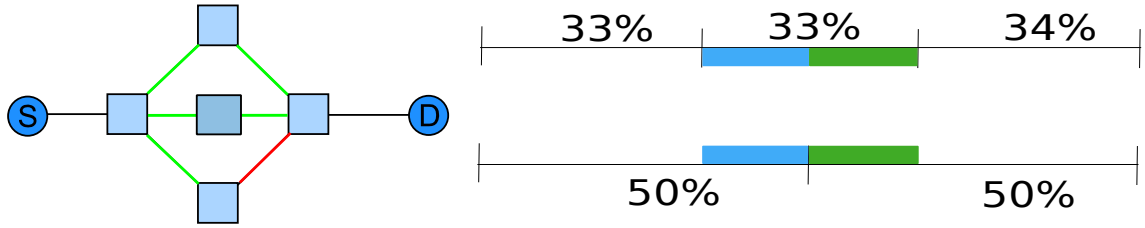

Figure 4: Répartition de la charge après une panne sur un lien

On peut facilement remarquer que les liens restant, se partageant la bande passante transférée, ont plus de risques de congestion. Dans le cas présenté ici, la charge reste acceptable (50%) mais si une nouvelle panne survenait, le dernier lien sera surchargé et cela pourra causer des pertes dues à des files d'attentes pleines engendrant des baisses significatives des performances des flux.

4.2.2 Liés au trafic

Une autre raison d'un changement de chemin est le trafic lui même. Lors d'un début congestion, un routeur peut, en théorie, re-router une partie du trafic sur un autre lien afin d'en limiter les effets, tels que des pertes.

De manière générale, chaque lien dispose d'un poids indiquant la quantité de trafic qui peut y être transféré. Lors d'une congestion, les poids des différents liens sont ré-évalués pouvant impliquer un transfert de charge entre plusieurs liens. En diminuant la charge du lien congestionné au profit d'un autre, la congestion devrait disparaitre.

Il s'agit ici de continuellement analyser l'état du réseau afin d'être le plus réactif possible et ainsi diminuer les risques de baisse de performances des flux empruntant les liens.

Le principal risque est d'envoyer toute la charge sur un lien créant alors une nouvelle congestion. Une fois cette dernière détectée, la charge serait alors transférée sur un nouveau lien qui à son tour serait congestionné. En l'absence de convergence, il y a un risque d'oscillation. Ces changements successifs peuvent avoir des impacts négatifs sur les flux que nous tenterons de découvrir par la suite.

4.3 TCP : Mécanismes de contrôle

4.3.1 Algorithmes

Les premiers travaux sur les mécanismes d'évitements de congestion ont permis la réalisation en 1996 de la RFC2001 (Stevens et al.[10]). Cette RFC définit différents mécanismes de TCP ainsi que la manière dont ils réagissent à certains phénomènes, tels que des pertes ou des déséquencements. Nous allons en définir les principaux avant de nous intéresser aux différentes versions de TCP les implémentant. Ces mécanismes peuvent être classés selon deux catégories : les mécanismes de montée en charge, comme *Slow Start*, et les mécanismes de reprises sur pertes, comme *Fast Retransmit*. Avant de les voir plus en détail, il est nécessaire de connaître certains des termes qui vont être employés.

La fenêtre de congestion (*cwnd*) correspond à la quantité de données d'un flux qui peuvent transiter au même moment sur le réseau. Cette fenêtre évolue en fonction du temps d'aller-retour, le *RTT*. Notons également que chaque paquet émis dispose d'un temporisateur. Si celui-ci expire (*Retransmission TimeOut*), il est considéré comme perdu.

4.3.1.1 Slow Start

Quand on entre en *Slow Start*, la fenêtre de congestion est habituellement positionnée à 1 et le seuil de Slow Start (**ssthresh**), représentant la valeur limite de **cwnd** au-delà de laquelle nous quittons le *Slow Start*, à 65536. Quand l'émetteur reçoit un acquittement, la *cwnd* est incrémentée de 1. Cela implique donc une évolution exponentielle binaire de la fenêtre à chaque **RTT**.

L'entrée en *Congestion Avoidance* se fait lorsque la valeur de **cwnd** est supérieure ou égale à celle de **ssthresh**. Ce seuil est ré-évalué à chaque perte ou déséquencement. En effet, les les conditions de trafic évoluant constamment, l'évolution de la **cwnd** ne doit pas être trop rapide mais doit s'approcher d'une valeur limite. **Ssthresh** peut être interprétée comme une mémoire d'une valeur de **cwnd** pour laquelle il n'y a pas eu de perte ou déséquencement. Au-delà de cette valeur, il est nécessaire d'augmenter plus prudemment la valeur de **cwnd** afin de s'approcher lentement mais sûrement d'une valeur maximale.

4.3.1.2 Congestion avoidance

Une fois que la **cwnd** a atteint la valeur de **ssthresh**, TCP passe en **Congestion Avoidance**. Cela signifie que la progression de la **cwnd** est ralentie afin de retarder au maximum toute congestion. En effet, rappelons que la **cwnd** est incrémentée en fonction du *RTT*. Dès que l'émetteur reçoit 3 **dupacks** ou au déclenchement d'un timeout, on diminue la valeur de **ssthresh** ainsi que la **cwnd**. Ces diminutions dépendent de la version du protocol utilisée. On parle ici d'AIMD : Additive Increase Multiplicative Decrease.

4.3.1.3 Fast Retransmit

Quand l'émetteur reçoit, selon les implémentations mais généralement 3 *acquittements dupliqués* (**dupacks**), il peut s'agir d'une perte ou d'un déséquencement. En effet, prenons 5 paquets, numérotés de 1 à 5. Supposons la perte du paquet 2 : à la réception des paquets 3 à 5, le récepteur acquittera toujours le paquet 3, indiquant qu'il ne l'a pas reçu. Si le paquet 2 arrivait simplement après le 5, le même cas de figure se présenterait. La réception de ces acquittements dupliqués ne permet pas de faire une distinction entre une perte et un déséquencement mais uniquement d'informer qu'un *problème* est survenu.

En l'absence d'information lui permettant de distinguer les deux cas, le segment est renvoyé. S'il s'agissait d'un simple déséquencement, le récepteur peut, grâce à son buffer, remettre les paquets dans l'ordre : le segment ré-envoyé n'est pas pris en compte à sa réception.

La valeur de la **cwnd** est diminuée selon l'aglorithme d'évitement de congestion utilisé. Cette diminution a donc un impact sur les performances.

4.3.1.4 Fast Recovery

Quand l'émetteur reçoit trois *dupacks* (généralement), il entre temporairement en *Fast Recovery*. Dans ce mode, la **cwnd** actuelle est fixée à *ssthresh+3*. A chaque nouveau **dupack**, la **cwnd** est incrémentée. Dès qu'un paquet a été correctement réceptionné, un nouveau est envoyé, permettant ainsi de maintenir le nombre de paquets en vol constant.

Une fois que tous les segments précédemment marqués comme perdus ont été reçus, l'émetteur entre en **Congestion Avoidance**. Notons que pour entrer en *Fast Recovery*, l'émetteur doit d'abord être passé par la phase de *Fast Retransmit*.

4.3.2 Versions de TCP

4.3.2.1 Tahoe

TCP Tahoe est la version de base de TCP, introduite dans la version 4.3 de BSD. Elle intègre les mécanismes les plus élémentaires parmi ceux présentés précédemment. En effet, comme nous pouvons le voir sur la figure 5, en cas de perte ou de déséquencement, la fenêtre de congestion est immédiatement fixée à 1 : *Fast Recovery* n'est pas implémenté dans cette version de TCP. L'algorithme reprend alors par un *Slow Start*.

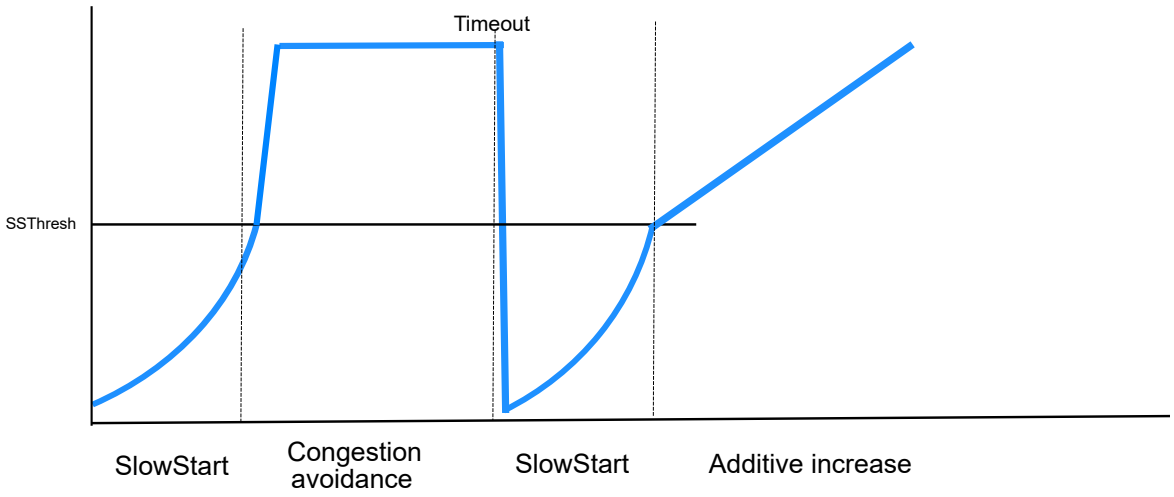

Figure 5: Tahoe

Cette version de TCP n'est plus utilisée sur les systèmes d'exploitation actuels.

4.3.2.2 Reno / NewReno

TCP Reno est une amélioration de TCP Tahoe, décrite dans la RFC3782 (Floyd et al.[4]). Dans cette amélioration, les valeurs de **ssthresh** et de **cwnd**, à la réception de 3 **dupacks**, sont fixées comme suit :

$$ssthresh = \frac{cwnd}{2}$$
$$cwnd = \frac{cwnd}{2} + 3$$

Le comportement de Fast Recovery est le même qu'énoncé précédemment.

Dans cette nouvelle version, le mécanisme de *Fast Recovery* a été amélioré. Dès qu'il détecte la perte de plusieurs segments d'une même rafale, et dès qu'il reçoit un acquittement partiel, l'émetteur va ré-émettre le segment perdu suivant sans sortir du **Congestion Avoidance**. Cette amélioration, bien que minime, a un réel impact positif sur les performances de TCP : en cas de congestion, le fait de ne pas entrer en *Slow Start* et ne pas fixer la **cwnd** à 1 permet de garder des performances plus stables.

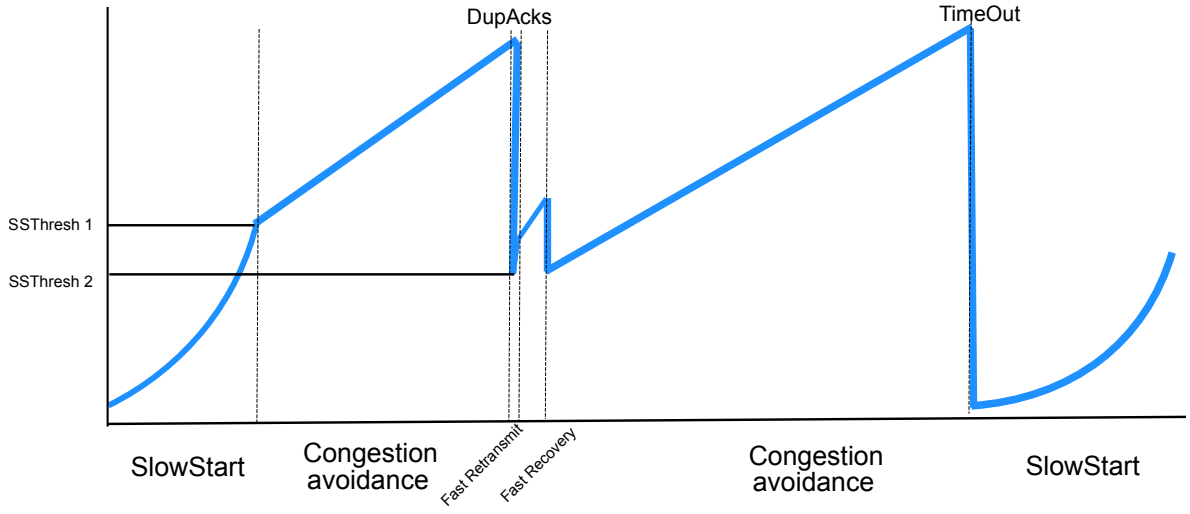

Figure 6: Reno/NewReno

Comme nous pouvons le voir sur la figure 6, en cas de réception d'acquittements dupliqués, la valeur de **cwnd** reste élevée. Ainsi, le débit est moins impacté. Toutefois, en cas de d'expiration de temporisateur, signe d'un plus gros problème, la baisse de **cwnd** implique une importante diminution des performances.

4.3.2.3 Vegas

Le but de TCP Vegas est de prévenir les congestions en analysant continuellement les délais d'acheminement des paquets. Si le délai vient à augmenter, il peut s'agir de la mise en buffer des paquets en transit donc un signe qu'une congestion est imminente. Vegas va alors décrémenter le nombre de paquets en transit afin de diminuer les risques de congestion.

En analysant les délais et en réagissant de manière pro-active, Vegas est un des algorithmes les plus équitables. Malheureusement, en présence d'autres versions de TCP, Vegas a tendance à perdre tout son intérêt. Là où Vegas réagit avant la congestion, les autres algorithmes attendent une information quant à une congestion en cours avant de réagir. Le fait que Vegas diminue sa **cwnd** avant n'est pas perçu par les autres algorithmes et la bande passante disponible alors est immédiatement récupérée par les autres versions de TCP.

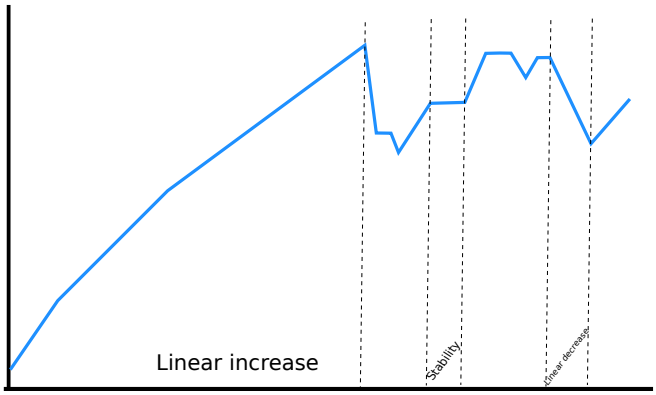

Figure 7: TCP Vegas

D'un point de vue des performances, Vegas est à même de proposer l'une des meilleures utilisations de la bande passante. S'il n'y a qu'un flux utilisant Vegas, la bande passante disponible sera utilisée au maximum. Si plusieurs flux concurrents utilisant Vegas partagent une même bande passante, celle-ci sera partagée équitablement.

4.3.2.4 CTCP

Compound TCP est utilisé dans les dernières versions de Windows. Comme présenté dans le papier de Song et al.[9], il utilise 2 fenêtres différentes, une basée sur les congestions (**cwnd**), la deuxième sur les délais (**dwnd**). La première évolue de la même manière que celle de New Reno. Concernant la **dwnd**, si les paquets en transit sont mis dans les buffers des routeurs, CTCP va la diminuer. Le but étant de garder constante la somme des fenêtres à une valeur estimée du produit-délai-bande passante.

En **Congestion Avoidance**, CTCP incrémente sa **cwnd** de 1/(cwnd+dwnd).

Sa relation avec New Reno les rendent difficiles à distinguer en analysant uniquement l'évolution de la **cwnd**.

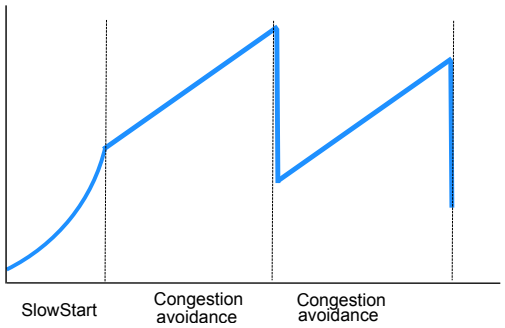

Figure 8: Compound TCP

Le fait d'utiliser une fenêtre basée sur les délais permet à Compound TCP de réagir rapidement aux variations pouvant indiquer un début de congestion.

4.3.2.5 CUBIC

CUBIC est la version actuellement utilisée dans les distributions récentes de Linux. Il s'agit d'une extension de BIC (Binary Increase Congestion) qui a été présentée par Ha et al.[7].

Avant toute chose, il est intéressant de savoir que, là où BIC se base sur le **RTT** pour augmenter sa **cwnd**, CUBIC utilise une fonction du temps. Il s'agit de le rend plus équitable en présence d'autres versions de TCP, de manière indépendante du chemin emprunté par les flux. BIC recherche une valeur de **cwnd** pour laquelle nous ne sommes pas censés avoir de pertes. Pour ce faire, il effectue une recherche dichotomique entre la valeur maximale de **cwnd** atteinte et cette même valeur, multipliée par un certain coefficient ayant pour but de diminuer la **cwnd**.

Lors de l'utilisation de CUBIC, la première partie concerne l'augmentation de la **cwnd**. Si l'augmentation de la **cwnd** est trop lente, la valeur retenue est la plus grande entre les valeurs de **slow start** de CUBIC et New Reno. Nous obtenons ainsi une augmentation au moins aussi importante que celle de New Reno.

La seconde partie est dite de *stabilisation* à une valeur appelée *Wmax*. Finalement, après cette phase, la **cwnd** est incrémentée afin de *sonder* le réseau et obtenir une meilleure valeur.

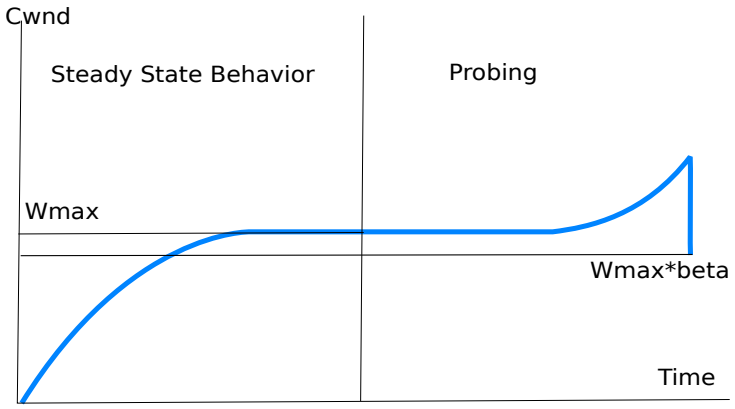

Figure 9: Cubic TCP

Lors de l'entrée en congestion, la **cwnd** est multipliée par une valeur *beta*, propre à la configuration du système d'exploitation. Cela permet alors de continuellement sonder le réseau à la recherche d'un début de congestion et adapter la fenêtre ou ne pas en observer et ainsi gagner en débit moyen sur la communication.

Il est également à noter que l'utilisation d'une fonction cubique rend cette version de TCP facilement reconnaissable en analysant uniquement l'évolution de la **cwnd**.

Comme nous le verrons plus tard, en raison de différences entre la version de CUBIC présentée et celle implémentée dans le simulateur utilisé, nous avons décidé de ne pas l'utiliser.

4.3.2.6 Récapitulatif

Le tableau suivant reprend les principaux points que nous venons de voir.

Version	Slow Start	Fast Retransmit	Fast Recovery
Tahoe	X	X	
NewReno	X	X	X
Vegas	X	Version revue	Version revue
CTCP	X		
CUBIC	X		

Notons également que dans leur papier, Abdeljaouad et al.[1] ont mis en évidence le fait que les versions CUBIC et CTCP ont des performances sensiblement similaires.

4.4 Les effets du partage de charge sur TCP

Lors d'un changement de chemin, les nouvelles caractéristiques du chemin emprunté peuvent avoir un impact sur les performances. Comme nous l'avons vu précédemment, des déséquencements peuvent être induits, pouvant impliquer une diminution de la **cwnd** et donc des débits. Dans cette section, nous nous intéresserons à une différence de temps d'aller-retour (RTT) induit par un changement de délai et expliquerons les impacts prévisibles.

4.4.1 Transfert vers un chemin de RTT supérieur

Dans le cas d'un transfert des paquets sur un chemin de RTT supérieur, les paquets empruntant le nouveau chemin auront accumulé un certain retard. Pour rappel, chaque paquet envoyé dispose d'un temporisateur, lui donnant un certain délai pour être acheminé et acquitté. S'il n'est pas acquitté à l'expiration de ce temporisateur (*Retransmission TimeOut*), il est considéré comme perdu. Les retards provoqués par l'augmentation du RTT impliquent une mauvaise estimation de ce *RTO*. S'il est estimé trop bas, un paquet arrivé convenablement ou acquitté mais toujours en transit peut arriver après expiration du temporisateur. Cela implique une réduction de la fenêtre de congestion donc du débit.

4.4.2 Transfert vers un chemin de RTT inférieur

Lors d'un passage à un chemin de RTT inférieur, certains paquets peuvent mettre moins de temps à atteindre leur destination que d'autres : il y a déséquencement.

La figure 10 correspond à un chronogramme des échanges entre la source et la destination d'un flux TCP dans deux cas. Le premier, sur la gauche, correspond à une faible différence entre les deux chemins. Sur la partie de droite, la différence est nettement supérieure. Les envois de paquets entre la source (barre verticale de gauche) et la destination (barre verticale de droite) sont représentés par les flèches vertes (paquets ayant emprunté un chemin avec un délai faible) et les flèches rouges (paquets ayant emprunté un chemin de délai supérieur). Les acquittements sont représentés par les flèches retour, en noir. Ceux-ci empruntent un unique chemin, n'ayant pas de variation de délai. Les nombres en noir indiquent le numéro de séquence du paquet émis ou reçu tandis que les nombres en bleu indiquent le numéro acquitté par le récepteur. Notons également que le temps évolue de bas en haut.

On peut ainsi remarquer que si la différence est suffisament importante (partie de droite), le récepteur peut envoyer des acquittement dupliqués, demandant le paquet déséquencé, avant de le recevoir. Si l'émetteur reçoit un nombre suffisant de **dupAcks** (généralement 3), il peut alors entrer en *Fast Retransmit*, réduire sa fenêtre de congestion et donc son débit.

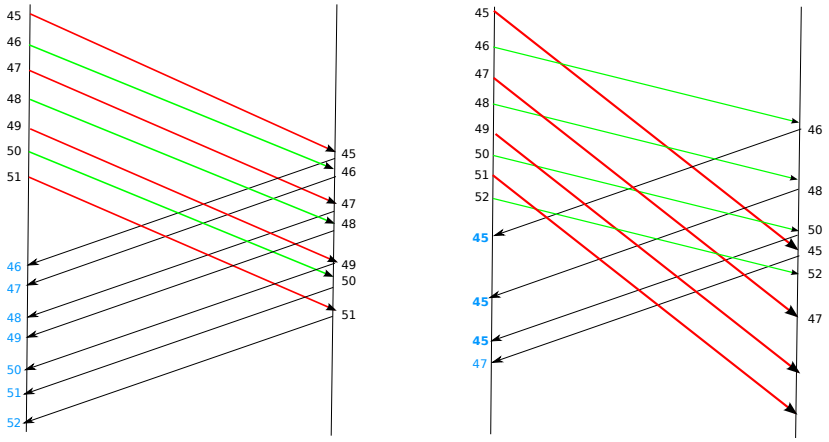

Figure 10: Chronogrammes des échanges selon deux différences de délais

4.5 Solutions envisagées dans la littérature

Certaines études ont déjà été menées sur l'impact des changements de chemins sur les flux TCP. Nous allons commencer par évoquer des méthodes de prédiction du nombre de paquets déséquencés puis nous présenterons quelques propositions d'amélioration de TCP afin de le rendre plus robuste face à ces déséquencements.

Gao et al.[5] ont montré une relation entre le *One-Way Delay* (délai d'acheminement d'un bout à l'autre), le nombre de paquets par seconde et le nombre de paquets qui pourraient être déséquencés :

$$N = K(OWD_2 - OWD_1)$$

Ici, N est le nombre de paquets déséquencés impliquant un probable problème (habituellement 3). K est le nombre de paquets transmis par seconde, OWD 1 et 2 sont respectivement les *One-Way Delay* du premier et du nouveau chemin. Ainsi, si l'on connait le nombre de paquets par seconde, le seuil de tolérance ainsi que le délai d'un chemin, on peut estimer le delta à ne pas dépasser lors de l'établissement d'un lien/chemin de secours ce, afin d'éviter trop de déséquencements.

Ce test est également applicable à un mécanisme du tourniquet, moyennant de très petites modifications, telles que la prise en compte du délai de mise sur le lien.

Dans leur papier, Blanton et al.[3] analysent l'impact du réordonnencement des paquets sur les performances de TCP et proposent différentes solutions afin de le rendre plus robuste.

Une autre possibilité évoquée est l'augmentation de la limite des **dupacks** en fonction du nombre de retrasmissions inutiles détectées (réception de l'acquittement juste après le renvoi du paquet). Le but étant ici de trouver la meilleure valeur pour cette communication, à un instant précis. Cependant, afin de trouver la bonne valeur, plusieurs retransmissions inutiles doivent avoir lieu. Lorsque la durée totale de communication est petite, ces quelques retransmissions, impliquant une diminution de la **cwnd** auront un impact plus important sur le débit moyen. Au contraire, lors de communications plus longue, le temps d'adaptation de l'algorithme, pour un même nombre de diminution de **cwnd** n'auront que peu d'effet sur le débit moyen sur toute la communication. Il a également été proposé de faire varier le seuil de **dupacks** en fonction du temps. Tant qu'aucun problème n'est rencontré, le seuil est augmenté progressivement. Cependant, si aucun problème n'est détecté pendant un longue durée, le seuil peut être trop élevé et des déséquencements pourraient ne pas être détectés.

De leur côté, Karlsson et al.[8] ont tenté de mettre en avant que le fait de privilégier un même chemin pour les acquittements, sur les paquets de données ou les deux peut améliorer les performances. Toutefois, le gain en privilégiant les données ET les acquittements par rapport aux données uniquement n'est pas important et peut même engendrer des performances moindres selon les cas. Le fait de privilégier uniquement les acquittements ne permet pas un gain suffisamment important pour mettre en production un tel changement.

5 Contributions et simulations

Maintenant que nous avons définit les mécanismes mis en jeu par TCP, les prévisions des impacts sur les performances ainsi que les propositions d'améliorations formulées dans la littérature, nous allons réaliser nos propres simulations afin de proposer de nouvelles améliorations quant à la gestion des déséquencements ainsi qu'une nouvelle vision des impacts selon la configuration, aussi bien physique qu'algorithmique, de notre topologie. Les expérimentations sur du matériel réel n'ayant pu donner de résultats significatifs dû à des limitations matérielles, nous nous sommes tournés vers le simulateur NS-2, présenté par exemple dans le papier de Wei et al.[11].

5.1 Scénarii et paramètres de simulation

Pour l'ensemble des simulations, nous utiliserons la topologie de la figure 11. Nous avons ici, une source S, une destination D, deux routeurs R1 et R2 et enfin une zone en bleu pouvant contenir de nombreux routeurs, augmentant de la même manière le nombre de chemins entre S et D. Chaque routeur de la zone bleue ne pourra avoir qu'un lien vers R1 et un lien vers R2. Aucun lien vers un autre routeur ne pourra être créé. Le débit entre S et R1 ainsi qu'entre R2 et D est au moins égal à la somme des débits disponibles entre R1 et R2, de manière à pouvoir utiliser le maximum de bande passante disponible entre R1 et R2. Lors de nos simulations, les débits étaient tous de 1 Mo/s en zone bleue et 5 Mo/s à l'extérieur (ne dépassant pas, généralement, les 5 chemins).

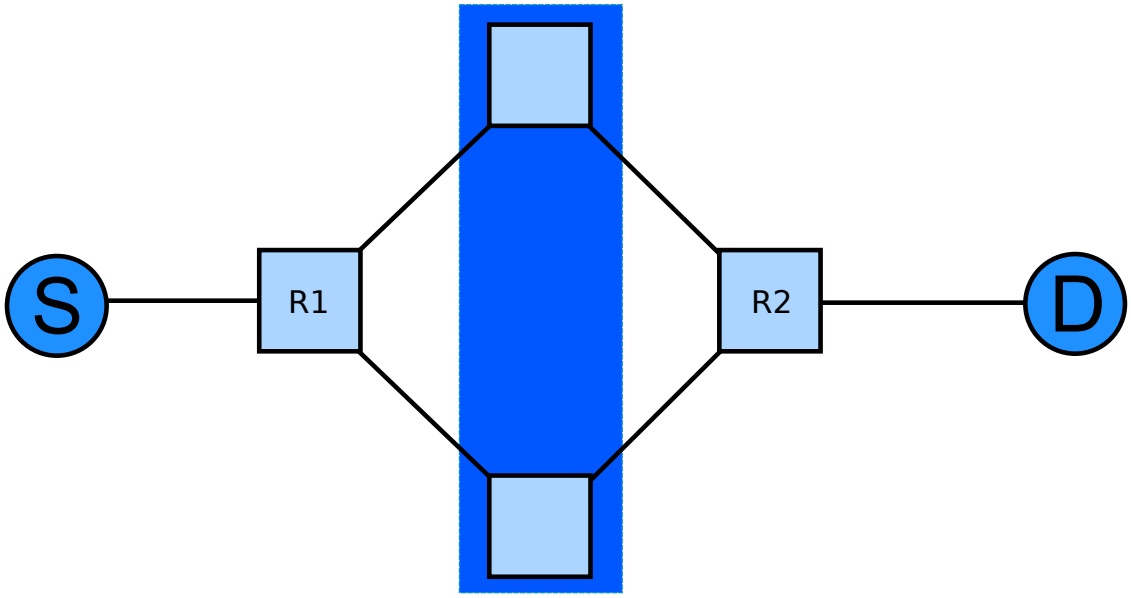

Figure 11: Topologie de référence

Les simulations ont été réalisées avec NS-2 v2.35. Des modifications ont été apportées au code afin

de s'adapter à certains de nos besoins : ces cas seront cités explicitement. Les communications entre la source et la destination se font en utilisant un agent NewReno, avec une application FTP. Cette application nécessite l'établissement d'une connexion TCP entre les hôtes. Lors de nos premières simulations, nous avons pu remarquer des différences entre les prévisions d'évolution de la **cwnd** et les résultats obtenus. Nous avons ainsi découvert que certaines versions de TCP, basées sur un agent Linux, utilisaient des améliorations modifiant légèrement leur comportement. Afin de ne pas obtenir de résultats biaisés par ces modifications que nous retrouvons uniquement sur des systèmes Linux„ nous avons décidé de nous limiter à un agent NewReno.

Afin de pouvoir étudier convenablement les résultats, plusieurs outils ont été développés :

- un générateur de chronogramme permettant de manière visuelle d'observer les échanges entre une source et une destination et plus précisément les déséquencements qui peuvent avoir lieu
- un simulateur d'échanges entre deux nœuds offrant la possibilité de définir le nombre et les caractéristiques des chemins entre ces nœuds tout en faisant abstraction de certaines caractéristiques

Le plus intéressant concernant la compréhension des déséquencements est le générateur chronogramme expliqué précédemment. A partir d'une trace NS-2, le programme génère un fichier contenant les dates de départ et d'arrivée de chacun des paquets ainsi que les numéros de séquence et d'acquittement entre la source et la destination. Ce nouveau fichier, simplifié par rapport à la trace initial, est alors traité via *R*. Chaque paquet et acquittement fait alors l'objet d'un tracé mettant en avant le décalage entre son moment de départ et d'arrivée comme nous pouvons le voir sur la figure 12. De plus, les segments et leur acquittement partagent une même couleur permettant d'avoir un aperçu direct des acquittements et envois (ou renvois) mais surtout des délais d'acheminement et du nombre de paquets déséquencés.

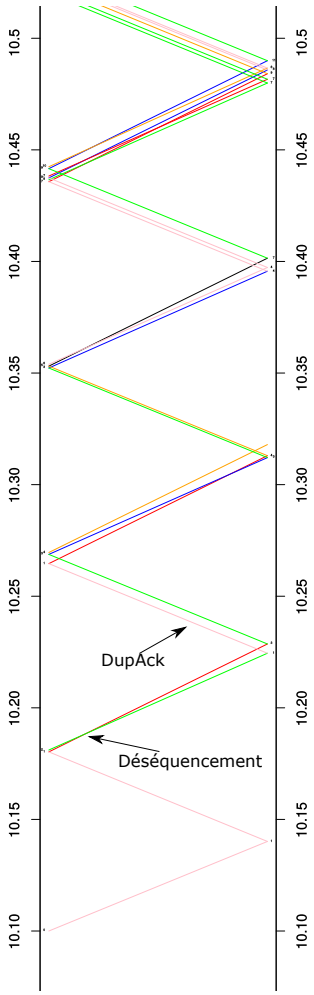

Figure 12: Chronogramme du début d'une communication TCP (incluant la connexion) avec renvoi dès le premier dupack reçu

L'axe des abscisses représente le temps écoulé depuis le début de cette simulation. L'axe des ordonnées de gauche représente la source et celui de droite la destination. Le but d'un tel outil est de nous aider à comprendre visuellement et rapidement les échanges et déséquencements entre les hôtes sans avoir à analyser une trace NS-2 pouvant comporter plusieurs milliers voire millions de lignes.

Le deuxième outil permet de simuler un réseau multi-chemins en faisant abstraction de certaines caractéristiques telles que la bande passante des liens. Son fonctionnement est expliqué dans la prochaine section.

Les valeurs par défaut utilisées lors de nos simulations sont répertoriées dans le tableau 5.1. Bien que la durée totale de simulation soit supérieure, la durée réelle d'émission depuis l'agent source est de 700 secondes. Cette différence est essentiellement présente afin de laisser aux différents routeurs le temps d'échanger leurs routes. En effet, lors des premiers instants de nos simulations, les routeurs doivent

découvrir le réseau et échanger leurs routes. Si l'ensemble des routes n'est pas échangé, il est possible qu'un lien soit privilégié par rapport à un autre.

Nom du paramètre	Valeur
Taille des files*	4 paquets
Seuil SlowStart	30000 paquets
Taille de fenêtre	10000 paquets
Taille de paquet	1000 (1050) octets
Durée de simulation	700 secondes

Tableau 5.1 : Autres paramètres définis
* Ce choix est expliqué en section 5.2

5.2 Résultats de simulations (sans améliorations)

5.2.1 Analyse des résultats

La première étape était de vérifier la formule proposée par Gao et al. (section 4.5) concernant le délai maximum accepté lors d'un changement de chemin. Cette limite serait alors un point de repère pour l'ensemble de nos simulations.

Pour ce faire, j'ai réalisé différentes simulations sur 2 chemins, avec du *Round Robin*, en utilisant exactement les mêmes paramètres, à un point près : le délai du deuxième chemin. Ce dernier varie de 10 à 150ms.

Afin de mieux comprendre les résultats, deux nouveaux termes doivent être introduits :

- Throughput : il s'agit de la quantité totale de données transmises entre les hôtes
- Goodput : il s'agit de la quantité utile de données transmises, à savoir les données uniques qui ont transité (hors ré-émission, ...)

Sur la figure 13, le délai maximum accepté est représenté par une barre verticale bleue. L'axe des abscisses représente le délai du deuxième chemin (le premier étant fixé à 10ms), tandis que les ordonnées représentent le débit. Les courbes, représentées par les points verts et rouges, représentent respectivement le *goodput* et le *throughput* de notre simulation sur une durée de 700s.

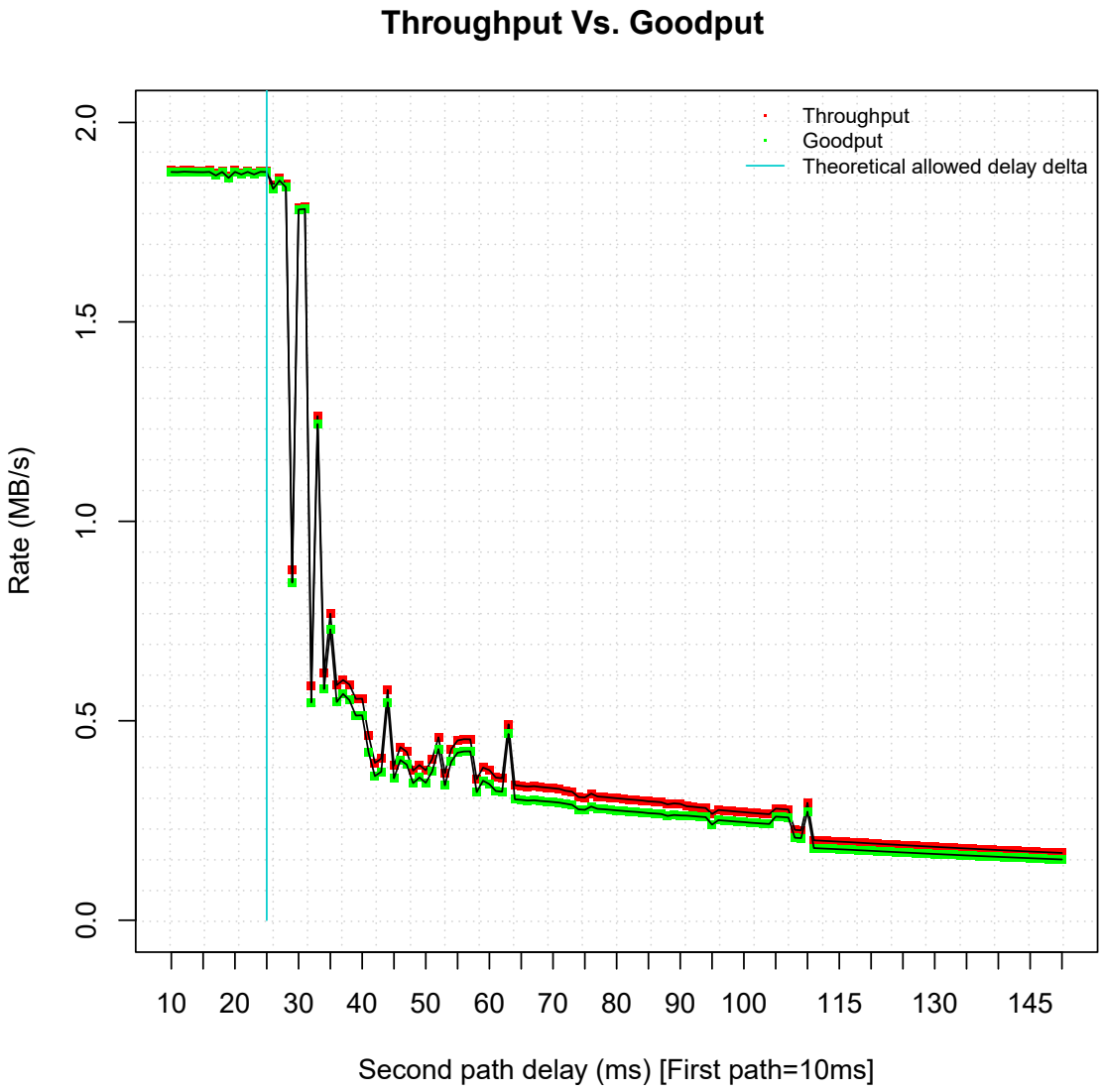

Figure 13: Evolution du *goodput* et du *throughput* sur deux chemins en fonction du délai appliqué au deuxième lien

Comme nous nous y attendions, le *goodput* ainsi que le *throughput* diminue nettement au-delà de la limite des 13ms de différence entre les deux chemins (délai de 23ms appliqué au deuxième lien). Lors de nos différentes simulations, cette limite était parfois amenée à varier légèrement selon la taille des files d'attentes. Pour des files d'attentes de 4 à 13 paquets, le délai à partir duquel les débits diminuent est toujours identique. Au-delà de 13 paquets, les files se remplissent progressivement lorsque des paquets sont envoyés en rafale. Cela a pour effet d'augmenter le temps de transit des paquets et pouvant conduire à un déclenchement de **RTO**, impactant les débits. Afin d'éviter tout effet de bord, nous avons décidé d'utiliser une taille de file de 4 paquets.

5.2.2 Réflexion générale

Maintenant que nous avons vérifié la limite lors de l'utilisation de deux chemins, nous souhaitons vérifier si cette règle s'applique à une configuration avecà plus de chemins. Cependant, les résultats obtenus dépendent de nombreux points. La moindre modification du délai voire même de l'ordre des chemins impactait considérablement les résultats, empêchant de définir une règle précise. Nous décidions alors de nous concentrer sur 3 chemins.

L'étape suivante est de repérer quel moment de la simulation est le plus critique. On peut imaginer deux phases :

- La phase de montée en charge, pendant laquelle la **cwnd** est progressivement incrémentée
- La phase stabilisée, pendant laquelle la **cwnd** est aux alentours de sa valeur maximale

Afin de différencier les deux phases, nous avons exécuté deux simulations : l'une avec l'utilisation d'une très grande fenêtre dès le début de la simulation, l'autre sans. Dans le premier cas, cette moyenne est passée à près de 2.9 Mo/s soit l'utilisation de près de l'ensemble de la bande passante disponible sur les trois chemins. Dans le deuxième cas, la moyenne du *goodput* était de 0.5 Mo/s soit la moitié de la bande passante d'un des deux liens disponibles.

Il semble donc que la partie la plus touchée par la mise en place de partage de charge sur des chemins dont les caractéristiques diffèrent soit la phase de montée en charge mais cela peut être compensé par une fenêtre initiale plus grande.

Maintenant que nous avons mis la main sur un point essentiel de la phase de montée en charge, à savoir l'envoi d'une grande fenêtre initiale, intéressons-nous à la phase stabilisée. Pour ce faire, j'ai développé un outil permettant de créer une simulation d'ordonnancement Round Robin et calculer les déséquencements mais surtout, le nombre de fois que nous passons au-delà de 3. Ce simple programme utilise des paramètres entrés par l'utilisateur :

- nombre de chemins
- délais des liens
- durée de la simulation

En fonction de ces paramètres la simulation, très sommaire, part du principe que les paquets arrivent en continu dans chacune des files. Il est également considéré que les temps de mise sur le médium et de

sortie des files d'attente sont négligés. On considère également que les délais sont des entiers et que chaque paquet fait la même taille.

Ce outil, déterministe, a été utilisé sur des délais différents et nous avons commencé à repérer des faits similaires entre les simulations.

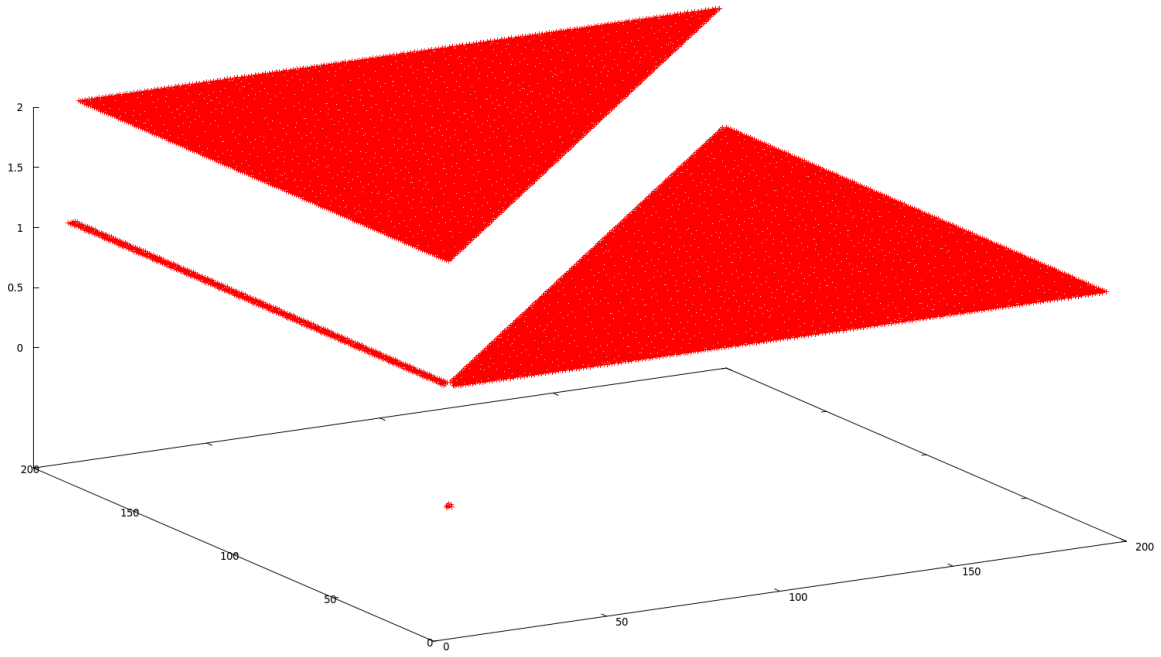

Figure 14: Nombre de déséquencements en fonction du délai appliqué à deux chemins sur 3

La figure 14 représente le nombre de fois que le nombre de déséquencements dépasse la limite de 3 paquets. Pour cette simulation, nous avons fixé le délai du premier lien à 10ms et avons fait varier le délai des deux autres de 0 à 200ms, comme nous pouvons le voir sur l'axe des abscisse et des ordonnées. Le nombre de fois que nous dépassons les 3 paquets déséquencés est représenté sur l'axe des cotes. Nous pouvons ici observer 3 plateaux. Lorsque les délais sont proches de 0ms, nous pouvons remarquer qu'il n'y a aucun impact sur les déséquencements. Lorsque les délais augmentent, nous pouvons observer un nouveau plateau. L'apparition d'une ligne est due à l'ordre dans lequel les paquets sont envoyés sur le lien. Enfin, le troisième plateau correspond aux plus grands écarts entre les délais des trois liens.

Peu importe le nombre de simulations, les délais appliqués aux liens ou la durée de simulation, nous pouvons observer que, jusqu'à trois chemins, le nombre de **dupacks** est toujours compris entre 0 et 2.

De plus, à partir de 4 chemins, nous pouvons observer un lien direct avec la durée de la simulation. Plus celle-ci est grande, plus il y aura de **dupacks**. Enfin, même en augmentant le nombre de chemins, nous n'observions plus de changement majeur dans le nombre de **dupacks**.

Finalement, le schéma qui en ressort est comparable à celui de la figure 15 dont voici le descriptif :

- Chaque ligne correspond à un chemin
- Chaque colonne correspond à un instant T
- Les valeurs indiquées dans chaque case correpondent au numéro de séquence du paquet
- Un fond vert correspond à un paquet réceptionné alors qu'il était attendu
- Un fond rouge correspond à un paquet réceptionné alors qu'un paquet le précédent n'est pas encore arrivé
- Les valeurs indiquées au dessus sont les instants T, utilisés comme indicateurs du début et fin de phase lors du calcul expliqué plus tard dans ce travail

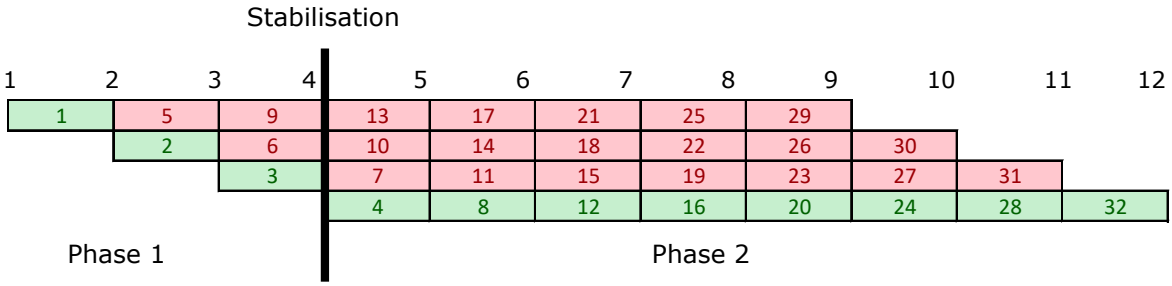

Figure 15: Représentation des paquets réceptionnés et mise en évidence des déséquencements par notre outil

Sachant que les problèmes surviennent au moment où nous atteignons les 3 **dupacks**, peu importe le nombre total, la diminution de la **cwnd** ne se posera que la première fois où l'on dépasse ce seuil. Le délai entre les deux premiers chemins est donc un premier indicateur de problème : dès lors que le délai est suffisamment élevé pour laisser passer 3 paquets devant, nous savons d'avance qu'il y aura un problème. Néanmoins, cette même règle ne s'applique pas lors d'un passage à 3 chemins. En effet, à chaque milliseconde, deux paquets passent devant le troisième. C'est pourquoi, pour la suite, il est intéressant de travailler sur deux zones distinctes :

- La phase 1 : avant stabilisation, c'est à dire avant que le premier paquet du chemin le plus ne soit arrivé
- La phase 2 : phase stabilisée, c'est à dire quand le récepteur reçoit des paquets en continu de chaque chemin encore actifs

Pour les deux phases, soient *nDupAcks* et *nChemins* respectivement le nombre de fois où nous avons plus de 3 déséquencements et le nombre total de chemins. *delay(i;i-1)* correspond au délai entre le chemin i et le chemin qui le précède.

Premièrement, analysons la phase 1. La formule suivante prédit le nombre de fois que l'on obtient au moins 3 déséquencements :

$$\forall i \in [2; nChemins], \quad \text{si} \quad \lfloor \frac{3}{2^{i-2}} \rfloor < delay(i; i-1), \quad \text{alors} \quad nDupAcks = nDupacks + 1$$

Cette formule prend en compte le nombre de chemins qui précèdent celui sur lequel le paquet vient d'arriver ainsi que le nombre de paquets qui peuvent arriver avant lui.

Etant donné que les paquets arrivent précisément à chaque instant T et sont traités dans l'ordre en commençant par le premier lien, à l'arrivée d'un paquet sur le lien N, les liens 1 à N-1 ont déjà réceptionné un paquet. C'est pourquoi nous avons utilisé un tel diviseur. Il s'agit en fait d'une astuce nous permettant d'obtenir, aux trois premières itérations, les valeurs 1, 2 puis 4.. Lors la première itération, il faut que 3 paquets au maximum soient réceptionnés avant l'arrivée du paquet sur le deuxième lien. Ainsi, les paquets arrivant toutes les millisecondes, la différence de délai ne doit pas être de plus de 3ms. A la deuxième itération, comme nous avons déjà 2 liens actifs, cette différence est divisée par 2. Comme nous utilisons des délais entiers, il n'est pas possible d'obtenir une différence de 1,5ms. L'utilisation de la partie entière nous permet de nous contraindre à 1ms de différence. Au-delà de 3 chemins, la différence de délai ne doit plus être positive. En effet, il y a forcément 3 paquets qui précèdent (1 par lien).

Concernant la phase 2, la formule est plus simple mais nécessite plusieurs applications successives.

$$Si \quad nbChemins > 3 \quad alors \quad nDupAcks = nDupAcks + (nFin - nDébut)$$

Ici, *nbChemins* correspond au nombre de chemins toujours actifs. De plus, *nDébut* et *nFin* sont les temps de début et de fin de chaque segment ayant un même nombre de chemins. Ces deux valeurs sont plus facilement compréhensibles en se référant à la figure 15. Dans les deux formules précédentes, 3 est le nombre de paquets dupliqués reçus avant de procéder à une diminution de la **cwnd**.

Cette formule est facilement compréhensible dès lors que l'on sait que le problème principal est le passage, ici, de 3 paquets devant un envoyé avant. Dans le cadre d'un *Round Robin*, et avec les hypothèses posées précédemment, si l'on a plus de 3 chemins en *activité*, il y a forcément au moins 3 paquets qui passeront devant un plus ancien. Ainsi, si on segmente la deuxième phase en plus petits blocs dont la fin se place à chaque fois qu'un nouveau chemin arrête de recevoir des paquets, et si on itère à chaque fois la formule, on obtient le nombre de fois que 3 paquets sont passés devant. On comprend par la même occasion le fait que, dès que l'on a plus de 3 chemins, le temps de simulation devient une des variables à prendre en compte pour la prédiction du nombre de *problèmes* encourus.

5.3 Pistes d'améliorations envisagées

Comme énoncé précédemment, la phase de montée en charge est la plus sensible aux différences de délai entre les chemins. Le fait d'augmenter la fenêtre d'émission dès le premier échange permettrait de palier à ces problèmes. Néanmoins, celà en amène un nouveau : trouver la valeur idéale.

En effet, une valeur trop petite ne résoudrait pas le problème entièrement. De même, une valeur trop grande causerait un probable remplissage des files d'attentes, ce qui amènerait à des pertes. Trouver une valeur idéale pourrait se faire en analysant les chemins caractéristiques dès le début de la communication. Cette recherche des caractéristiques peut se faire, par exemple, de deux manières différentes :

- utiliser les premiers échanges
- utiliser des *sondes*, envoyées en parallèle

Dans le premier cas, ces échanges peuvent subir les mêmes problèmes que ceux que l'ont tente d'éviter. En effet, ces échanges peuvent subir des déséquencements. A la réception des **dupacks**, l'émetteur pourrait alors choisir une valeur non idéale impliquant des effets indésirables sur la suite des échanges. Dans le deuxième cas, plus de données sont envoyées dans le même laps de temps. Si plus de données transitent sur les liens, cela augmente les risques de congestion. Nous avons de plus énoncé que certains routeurs pouvaient modifier les chemins empruntés lorsqu'ils détectaient une congestion. Ces risques et modifications impliquent que l'utilisation de sondes n'est pas envisageable tel quel. Finalement, la mise en place de ce type d'analyse et d'évolution dynamique seraient trop coûteux, en temps et en bande passante, pour trop peu de bénéfice.

La deuxième amélioration est donc la modification du nombre d'acquittements dupliqués avant une diminution de la **cwnd**. Une nouvelle fois, nous devons trouver la valeur idéale :

- une valeur trop basse ne changerait rien à notre problème
- une valeur trop haute résoudrait les problèmes de déséquencements mais en cas de perte, le temps de réaction serait alors augmenté.

Habituellement, le nombre de chemins disponibles entre deux points ne dépasse pas les 4 à 5 chemins. L'idée serait par exemple de fixer la limite de **dupacks** à 5.

5.4 Résultats de simulations (avec améliorations)

Dans la section précédente, nous avons proposé de fixer la valeur du nombre de **dupacks** avant diminution de la **cwnd** à 5. Nous allons maintenant analyser l'impact de cette modification.

Dans un premier temps, analysons l'impact sur les déséquencements et les diminutions de la **cwnd**. Pour cela, nous avons relancé les simulations en fixant la valeur maximum de **dupacks** acceptés à 5.

La différence entre les deux simulations (avec 3 et 5 déséquencements acceptés) est visualisable sur la figure 16. Le délai du premier chemin est toujours fixé à 10ms, tandis que le deuxième est fixé à 21ms (représenté par la barre verticale rouge). Enfin, la valeur du troisième est représentée par l'axe des abscisses. Comme on peut le remarquer, le débit moyen sur la même simulation est sensiblement meilleur pour des délais faibles (inférieurs à 40ms). Le pic que nous pouvons observer est dû au fait que les deux chemins, dont les délais sont les plus longs, ont finalement une différence suffisament faible pour ne pas subir de déséquencements.

En appliquant un seuil de 5 paquets déséquencés, le *goodput* passe ainsi, sur les valeurs inférieures à 40ms, d'une moyenne de 0,6Mo/s à 2.1Mo/s, soit un gain de plus de 300%. L'utilisation des liens peut alors atteindre près de 70%.

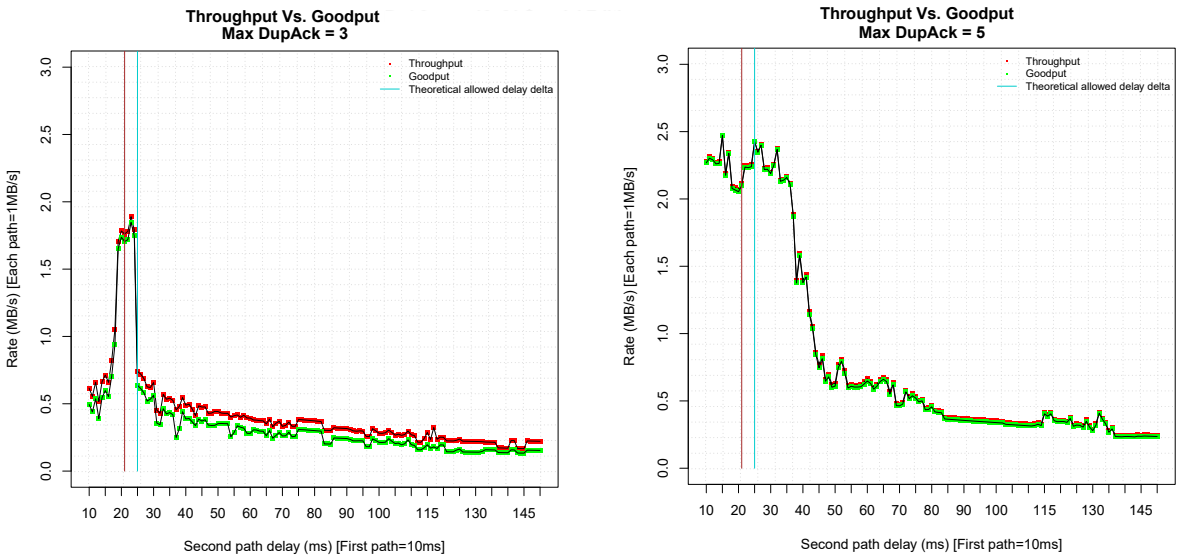

Figure 16: Evolution des *goodput* et *throughput* selon une limite de **dupacks** à 3 ou 5 paquets

Dans un deuxième temps, nous avons analysé le débit moyen lorsque l'on obtient une perte. En effet, lors d'une perte, plus d'acquittements dupliqués sont reçus avant de procéder à une diminution de la fenêtre d'émission et la réémission du paquet perdu. Cela a donc un impact sur le débit moyen car aucun paquet n'est envoyé pendant un plus long laps de temps et la nouvelle montée en charge est décalée dans le temps. Ce passage à 5 acquittements dupliqués ne crée une diminution du débit moyen que de 0,1Mb/s soit de 4%.

Finalement, le gain, lors d'un passage à 5 **dupacks** est nettement supérieur à la perte engendrée. L'utilisation de cette valeur est envisageable dans la plupart des scenarii.

6 Conclusions et perspectives

Lors de ce travail, nous avons analysé l'impact des changements de chemins sur les flux TCP. Bien qu'ayant appris les bases de ces protocoles et algorithmes, nous avons dû approfondir les différents principes de TCP, les types de partages de charges, ... Renforcer ces bases permet de mieux préparer les différentes simulations que nous avons mis en place mais aussi avoir une première idée des résultats que nous souhaitions. A partir des différentes simulations et analyses, nous avons mis en évidence différents points importants quant à la baisse de performances dans certaines situations.

Le code source des outils étant disponibles sur le Git de l'équipe, il peut être intéressant de les améliorer, leur ajouter des fonctionnalités. Par exemple, pour le simulateur, l'ajout de délais non-entiers permettrait d'en faire un outil plus complet permettant ainsi de prédire plus facilement les déséquencements et donc les baisses de performances en fonction de divers paramètres.

Les modifications que nous avons apporté nous ont permis d'améliorer les débits moyens des communications sur des chemins utilisant du partage de charge. Néanmoins, ces modifications n'ont pas pu être vérifiées sur du véritable matériel. A titre de perspective, il peut être intéressant de vérifier nos propositions.

A partir de nos simulations, nous avons mis en évidence que deux légères modifications peuvent significativement améliorer les performances de certaines communications tout en ayant un faible impact en cas de perte d'un ou plusieurs paquets.

Nous avons de plus montré que, dans le cas d'un Round Robin, bien que nous puissions parfois améliorer les performances des flux TCP, il y a trop de variables aléatoires pour en définir un modèle à suivre. Toutefois, une adaptation des formules présentées dans ce mémoire pourrait permettre de définir des règles de bons usages lors de la mise en place d'algorithmes de partage de charge existants ou à venir.

7 Bilan personnel

Le fort intérêt que j'ai eu pour ce sujet et cette équipe vient de ma première *expérience* en recherche lors du Travail d'Etude et de Recherche. En plus de découvrir plus en profondeur le métier de chercheur, j'ai pu parfaire mes connaissances sur de nombreux points comme TCP. De plus, bien que n'ayant pu travailler sur de réelles topologies, les configurations et recherchers que j'ai pu réaliser dans le cadre de la préparation des routeurs m'ont permis de mieux connaître les possibilités, les éléments de configuration mais surtout les interfaces de chaque marque.

De plus, cela m'a permis de développer mon esprit critique mais aussi mes capacités à rendre compte et à préparer des présentations.

Peu importe mes choix futurs, ce stage m'aura énormément appris.

Références

1. Abdeljaouad, I, H Rachidi, S Fernandes, and A Karmouch. 2010. "Performance Analysis of Modern TCP Variants: A Comparison of Cubic, Compound and New Reno." In *Communications (QBSC), 2010 25th Biennial Symposium on*, 80–83. IEEE.

2. Augustin, Brice, Xavier Cuvelier, Benjamin Orgogozo, Fabien Viger, Timur Friedman, Matthieu Latapy, Clémence Magnien, and Renata Teixeira. 2006. "Avoiding Traceroute Anomalies with Paris Traceroute." In *Proceedings of the 6th ACM SIGCOMM Conference on Internet Measurement*, 153–58. ACM.

3. Blanton, Ethan, and Mark Allman. 2002. "On Making TCP More Robust to Packet Reordering." *ACM SIGCOMM Computer Communication Review* 32 (1). ACM: 20–30.

4. Floyd, S, and A Gurtov. 2004. "The NewReno Modification to TCP's Fast Recovery Algorithm", RFC3782." Citeseer.

5. Gao, Ruomei, Dana Blair, Constantine Dovrolis, Monique Morrow, and Ellen Zegura. 2007. "Interactions of Intelligent Route Control with TCP Congestion Control." In *NETWORKING 2007. Ad Hoc and Sensor Networks, Wireless Networks, Next Generation Internet*, 1014–25. Springer.

6. Guo, Liang, and Ibrahim Matta. 2001. "The War Between Mice and Elephants." In *Network Protocols 2001, Ninth International Conference on*, 180–88. IEEE.

7. Ha, Sangtae, Injong Rhee, and Lisong Xu. 2008. "CUBIC: a New TCP-Friendly High-Speed TCP Variant." *ACM SIGOPS Operating Systems Review* 42 (5). ACM: 64–74.

8. Karlsson, Jonas, Per Hurtig, Anna Brunstrom, Andreas Kassler, and Giovanni Di Stasi. 2012. "Impact of Multi-Path Routing on TCP Performance." In *World of Wireless, Mobile and Multimedia Networks (WoWMoM), 2012 IEEE International Symposium on a*, 1–3. IEEE.

9. Song, Kun Tan Jingmin, Q Zhang, and M Sridharan. 2006. "Compound TCP: A Scalable and TCP-Friendly Congestion Control for High-Speed Networks." *Proceedings of PFLDnet 2006*.

10. Stevens, Wright. 1996. *TCP Sow Start, Congestion Avoidance, Fast Retransmit, and Fast Recovery Algorithms*.

11. Wei, David X, and Pei Cao. 2006. "NS-2 TCP-Linux: an NS-2 TCP Implementation with Congestion Control Algorithms from Linux." In *Proceeding from the 2006 Workshop on Ns-2: the IP Network Simulator*, 9. ACM.