Partage de bande passante étendu : nouveaux protocoles TCP & *fair queueing*

Résumé :

Dans ce chapitre, nous revisitons la relation fondamentale qu'il existe entre les ressources du lien, la demande en trafic et la performance obtenue. Nous considérons l'introduction de nouveaux protocoles TCP plus efficaces, conjointement à l'utilisation d'un ordonnancement *fair queueing* (PFQ, muni de la politique LQD).

PFQ/LQD permet un partage équitable de la capacité du lien (C) et du buffer (B), l'introduction de nouveaux mécanismes TCP plus efficaces (et moins dépendant des ressources disponibles), ainsi que la différenciation implicite des flots *streaming* et élastiques. La combinaison PFQ/LQD et MBAC permet de garantir leur performance.

Sommaire

Contributions :

- Illustration de la dégradation de performance des flots streaming et élastique sans ordonnancement, et des apports du *fair queueing* ;
- Évaluation des versions les plus courantes de TCP à la lumière de notre compréhension du trafic, dans un environnement équitable ;
- Propositions pour la conception d'algorithmes TCP plus efficaces.

Publications et présentations :

Ce chapitre a fait l'objet des présentations suivantes en *workshop* :

- James Roberts, Jordan Augé, **Fair Queueing and Congestion Control**, *Workshop on Congestion Control, Hamilton Institute, September 2005.*
- Jordan Augé, James Roberts, **Performance of TCP over a Fair Queueing Link**, *Proceedings of Workshop on QoS and Traffic Control, EuroNGI, Paris, December 2005.*

Il intègre également du contenu de la publication citée dans le chapitre précédent :

- Jordan Augé, James Roberts, **Buffer Sizing for Elastic Traffic**, *NGI2006, 2nd Conference on Next Generation Internet Design and Engineering, València, April 3-5 2006*

5.1 Introduction

Les modèles de trafic tels que ceux présentés dans le Chapitre 3 se basent sur une version fluide, idéalisée du trafic, où les mécanismes au niveau paquet n'interviennent pas. En pratique, TCP ne réalise pas un partage équitable, instantané, et efficace des ressources. Comme il a été évoqué plus tôt, les versions classiques de TCP – Reno/NewReno – sont reconnues comme insuffisantes pour exploiter convenablement la bande passante disponible sur des liens à fort "Bandwidth-Delay product", notamment au vu de la lenteur de l'algorithme AIMD. Nous avons vu de plus que la présence de petits buffers ne permet pas à l'algorithme une utilisation efficace des ressources. Les interactions au niveau paquet au sein d'un petit buffer, exacerbées par l'envoi en rafales causé par le contrôle de congestion de bout en bout, entraînent des pertes de paquets "naturelles", non liées à de la congestion, qui obligent TCP à adapter son débit bien en dessous du débit équitable. Dans des conditions réalistes de trafic, il est ainsi difficile de réduire drastiquement la taille des buffers dès lors que l'on souhaite assurer la performance des flots TCP Reno, sur un lien FIFO.

Les évolutions que connaissent les réseaux aujourd'hui, avec l'arrivée des réseaux optiques à très haut débit, ou le fort intérêt suscité par les *datacenters*, sont autant de facteurs motivant l'introduction de nouveaux protocoles ou mécanismes de gestion du trafic plus adaptés. De nombreux travaux montrent aujourd'hui l'intérêt de reconsidérer l'introduction d'ordonnancement (*scheduling*) ou de politiques de gestion des files d'attente (*AQM*), afin d'améliorer la performance des flots (FQ-Codel [70], pFabric [5], etc.).

Une alternative radicale aux solutions FIFO ou avec AQM est d'introduire un ordonnancement équitable, ou *fair queueing* (FQ), entre les flots. FQ permet un découplage entre le contrôle de débit et l'ordonnancement des flots. Nous utilisons l'implémentation PFQ (que nous avons réalisé dans ns-2), avec une politique rejetant les paquets de la file la plus longue (LQD, *Longuest Queue Drop*). Suter *et al.* montrent qu'un tel mécanisme est largement plus efficace que RED pour la gestion du trafic. Nous verrons qu'il permet aux flots *non-bottlenecked* de ne subir aucune perte. Leurs paquets subissent notamment de faibles délais [1], ce qui est une propriété désirable pour les flots *streaming* qui y sont inclus.

Un autre avantage bien connu du *fair queueing* est la possibilité d'introduire et d'expérimenter de nouvelles versions de TCP sans nuire aux utilisateurs de la version standard, puisque l'équité dans le réseau est assurée indépendamment du comportement de l'utilisateur. Le *fair queueing* peut être considéré comme infaisable si le nombre de flots à contrôler est trop important, et s'il croît avec la capacité du lien. En réalité, le nombre de flots à ordonnancer à tout instant est relativement faible, indépendamment de la capacité du lien (voir Chapitre 3, et [95, 93]).

1. Les délais sont d'autant plus réduits avec l'utilisation d'une file prioritaire décrite dans [91, 92].

L'impact sur la performance ne sera bien entendu perceptible que lorsque le lien sera saturé, ce que nous avons qualifié précédemment de régime élastique. Ainsi dans ce chapitre, nous explorons sous une nouvelle perspective la relation introduite dans le Chapitre 2, entre ressources, demande et performance. Nous réexaminons les arguments relatifs à l'introduction de tels protocoles au vu de notre compréhension du trafic, et en considérant l'utilisation complémentaire d'un mécanisme de *fair queueing*.

Nous commençons dans la section 5.2.2 par présenter un ensemble d'évolutions proposées pour TCP, pour ensuite motiver l'introduction dans la section 5.3 de *fair queueing* (FQ), qui assure la performance des flots *streaming* et élastiques, et permet l'introduction de nouveaux protocoles TCP plus efficaces. En considérant désormais un environnement équitable, nous analysons dans la section 5.4 un ensemble représentatif de versions de TCP au vu de notre compréhension du trafic, afin de vérifier dans quelle mesure les résultats établis dans le chapitre précédent sont confirmés (impact de petits buffers en présence de charge *background*). Finalement, la section 5.5 discute les résultats obtenus et un ensemble de pistes. Malgré une performance supérieure, nous constatons que les versions existantes de TCP peuvent encore être améliorées afin de mieux accepter les pertes aléatoires causées par les petits buffers et mieux exploiter la bande passante disponible. Il est possible d'exploiter les bénéfices apportées par le *fair queueing* afin de concevoir de nouveaux protocoles, plus agressifs, tout en gardant un taux de perte de paquets limités. Un complément prometteur, bien que de mise en œuvre complexe, semble être l'utilisation de techniques d'espacement des paquets (*pacing*),

5.2 État de l'art : Nouveaux protocoles TCP et évaluation de leur performance

5.2.1 Faiblesses de TCP Reno pour les transferts à haut débit

TCP a été introduit à une époque où les débits des liens étaient encore relativement faibles. Ce protocole a dû s'adapter aux différentes évolutions que le réseau a connu, et assurer un transfert fiable même sur les capacités actuelles des réseaux. Cependant, on lui reconnaît aujourd'hui certaines faiblesses sur les liens à forts délais et/ou à haut débit.

Le premier problème vient notamment de l'algorithme AIMD, qui régit l'évolution de la taille de la fenêtre de congestion w ainsi :

$$w \leftarrow w + \frac{1}{w}$$

à chaque acquittement reçu en mode congestion avoidance, et

$$w \leftarrow \frac{w}{2}$$

en cas de perte de paquet.

Lors d'une congestion ponctuelle, l'algorithme AIMD entraîne une réduction relativement brutale (*multiplicative decrease*) de la taille de la fenêtre de congestion (elle est divisée par deux), et celle-ci ne retrouvera sa taille d'origine que longtemps après à cause de la faible croissance (*additive increase*). La réalisation de très hauts débits avec TCP Reno nécessite un taux de perte infime, qui peut être difficile en pratique. Ainsi, les pertes subies par un flot TCP entraîneront un débit moyen de transfert bien en deçà de la limite équitable permise par le lien. Cette sensibilité peut être diminuée en ne considérant plus seulement une notion binaire telle qu'une perte de paquets, mais en y adjoignant d'autres estimateurs (TCP Vegas utilise par exemple une estimation du nombre de paquets attente dans le buffer). L'évolution de la fenêtre de congestion est modulée par l'état de congestion du réseau.

5.2.2 Nouveaux protocoles TCP

	Protocole	Caractéristiques
Loss	Reno	Incrément additif, décrément multiplicatif de *cwnd* (AIMD)
	HSTCP [52]	AIMD classique pour de faibles valeur de *cwnd*, version plus agressive au delà
	Scalable [85]	Incréments et décréments multiplicatifs (MIMD)
	Westwood+	Incrément additif, et décrément adaptatifs en fonction d'une estimation de bande passante basées sur le flux d'acquittements avant une perte
	BIC	Une combinaison de fonctions logarithmiques et exponentielles pour la croissance de *cwnd*
	CUBIC	Extension de BIC avec une fonction cubique
Delay	H-TCP	AIMD adaptatif
	Vegas	Incrément et décrément additifs (AIAD)
	Fast [78]	Incrément/décrément basés sur une équation faisant intervenir le délai principalement, ainsi que le signal de perte
Hybrid	Compound	Somme de deux valeurs de *cwnd*, de type Reno et Vegas
	Illinois	AIMD dont les paramètres d'incrément/décrement sont fonction du délai
	Veno	Combinaison Reno/Vegas afin de détecter les pertes non dues à la congestion
	Yeah	Alternance entre un protocole agressif (eg. Scalable TCP) et un contrôle de congestion à la Reno, en fonction d'une estimation de délai.

TABLE 5.1 – Résumé des versions de TCP considérées dans ce chapitre

Il existe un très grand nombre de protocoles TCP. Il ne s'agit pas ici d'en fait un inventaire exhaustif, mais d'en sélectionner un petit nombre représentatif, et couvrant les différentes approches. Nous avons retenu les protocoles présents dans le noyau Linux, disponibles également dans le simulateur NS avec le package TCP Linux. Ils sont résumés dans le tableau 5.1, en fonction de leur signal de congestion principal (pertes, délais ou mixte), et accompagnés d'un bref descriptif de leurs différences.

Nous n'avons pas considéré dans ce chapitre des protocoles nécessitant un marquage ou une signalisation explicite des débits, comme XCP [81], qui est difficilement réalisable dans l'interdomaine et nécessite des modifications importantes au sein des routeurs, ou encore DCTCP [4], qui est proposé dans un contexte de *datacenter* uniquement, et qui n'était pas disponible au moment de ces travaux.

Performance

Il n'est pas surprenant que l'évaluation de la performance des différents algorithmes TCP ait reçu un intérêt considérable de la communauté, et l'on recense un nombre incalculable de publications réalisant des évaluations analytiques, par simulation ou expérimentales de la performance d'une ou plusieurs propositions d'algorithme TCP.

Une des premières études de l'impact de la taille du buffer sur la performance de HSTCP est par exemple proposée par Barman *et al.* [12]. Le débit réalisé atteint respectivement 90% et 98% de la capacité du lien pour des tailles de buffer égales à 10% et 20% du *Bandwith Delay Product*. Pour une trop petite valeur du buffer toutefois, le débit observé est bas. Typiquement, le fonctionnement des protocoles est dégradé en présence de petits buffers, ce qui est consistent avec nos observations. On pourra se référer à [100] pour une évaluation expérimentale systématique d'un grand nombre de variantes TCP. Les métriques généralement considérées sont les suivantes :

équité : plusieurs notions et métriques d'équité ont été définies dans la littérature. L'étude est complexe dès lors que l'on considère plusieurs protocoles car il convient de considérer l'équité intraprotocolaire (entre protocoles similaires), ainsi que l'équité interprotocolaire (en fonction des différentes combinaisons possibles). Le comportement d'un réseau intégrant de nombreuses versions hétérogènes de TCP est peu maitrisé, et à de fortes chances de le rester.

En pratique, les systèmes d'exploitation Linux et Windows offrent déjà des versions différentes, respectivement CUBIC et Compound. Nous ne nous intéresserons pas à cette métrique, puisque nous confierons l'allocation de ressources à l'ordonnanceur FQ. Le rôle de TCP sera alors de maximiser son efficacité et l'utilisation de la bande passante disponible. A notre connaissance, aucune évaluation des protocoles n'a été effectuée dans un tel environnement.

vitesse de convergence : elle représente le temps mis par le protocole à s'adapter aux arrivées et départs de flots pour rejoindre le débit équitable. Lors d'un départ de flot, chaque flot recevra équitablement une partie du débit correspondant qui, s'il n'ajuste pas son débit suffisamment rapidement, sera soit perdue, soit utilisée par un autre flot. Lors de l'arrivée d'un nouveau flot, et lorsqu'il atteindra son débit équitable, cette lenteur se manifestera par des pertes de paquets. En présence de *fair queueing*, une convergence plus lente entraînera soit des pertes de paquets, soit un débit temporairement inférieur à celui alloué.

efficacité : il s'agit du débit moyen atteint par les flots. C'est la métrique que nous allons plus particulièrement développer dans ce chapitre, puisqu'elle reflète les critères de performance que nous considérons pour les flots élastiques (un débit moyen suffisant à garantir un temps de complétion satisfaisant).

débit utile : le débit utile est lié à la notion précédente et correspond au ratio entre le débit d'envoi, et le débit effectif après pertes. La présence de *fair queueing* permettrait en théorie à une source d'émettre au débit maximum, mais cela causerait un gaspillage inutile des ressources lorsque les paquets sont rejetés en masse à l'entrée d'un lien saturé. L'utilisation de codes correcteurs d'erreur comme alternative au mécanisme d'acquittement serait une autre motivation pour conserver un taux de pertes acceptable.

stabilité des buffers : à ne pas confondre avec la stabilité au niveau flot (demande < capacité), ou au niveau paquet (avec un buffer fini, le lien sera stabilisé par les pertes de paquets). Il s'agit de minimiser les oscillations dans le buffer, responsables de gigue pour les paquets, et qui sont liées avec la synchronisation des flots TCP. Généralement, une diminution de la taille des buffers améliorera cette métrique (Voir les travaux de Raina et Wishik [129] introduits dans le Chapitre 4 Section 4.3.2).

Peu d'études considèrent un modèle de trafic avec des arrivées dynamiques de flots finis. La plupart considèrent des flots permanents, et en très grand nombre, ce qui ne correspond pas à un taux de charge réaliste pour un lien opérationnel. Nous avons remarqué dans le Chapitre 4 que la présence de trafic concurrent pouvait fortement dégrader la performance d'un flot TCP. Un phénomène similaire à été observé plus tard dans [150, 151] et [148], mais n'a pas donné lieu à une évaluation des protocoles dans un tel contexte. Une évaluation intéressante de la performance d'un flot TCP en présence d'une charge *background* est faite dans [80], mais le modèle est complexe et ne peut être réutilisé pour notre étude (spécifique à Reno).

5.3 Motivations pour l'intégration de fair queueing

Dans cette section, nous effectuons quelques simulations permettant d'illustrer les bénéfices apportés par l'utilisation d'un ordonnancement *fair queueing*. Ce même environnement de simulation sera utilisé par la suite pour l'évaluation des différents protocoles TCP.

5.3.1 Environnement de simulation

Topologie et modèle de trafic

Nous utilisons un environnement de simulation similaire à celui du chapitre précédent, permettant de réaliser un simple lien goulotté au travers d'une topologie *dumbbell*. Le lien central possède une capacité C (de 50, 100 ou 500Mb/s), et le RTT des flots y est de 100ms. Les simulations mettent en jeu des tailles de buffers B variées, notamment inspirées du chapitre précédent : $B = 20$, 100 ou 625 paquets (BDP). Deux disciplines de services sont considérées : FIFO/DropTail et PFQ/LQD.

Notre modèle de trafic repose sur une hypothèse de quasi-stationnarité qui nous permet de considérer deux ensembles de flots : un trafic *background* de charge ρ_b qui est constitué de l'agrégat des flots *non-bottlenecked* (la majorité des flots), ainsi qu'un ensemble de N flots *bottlenecked*, que l'on peut alors assimiler à des flots permanents.

Flots *bottlenecked* (contraints localement)

Comme nous l'avons vu dans le Chapitre 3, leur nombre est généralement faible avec $Pr[N \geq i] \sim \rho^i$, où ρ désigne la charge relative des flots goulottés. Pour illustrer la performance des flots *bottlenecked* et *non-bottlenecked*, nous limiterons nos simulations aux quelques cas représentatifs tels que $N \in [1,2,4]$. Considérons par exemple un lien avec une charge *background* $\rho_b = E[flowarrivalrate] \times E[flowsize]/C = 50\%$, pour une charge globale de 0.6, on a $Pr[N \geq 5] \approx \rho^5 = 0.2^5 \approx 0.0003$.

Flots *non-bottlenecked* (non-contraints localement)

En régime transparent, nous avons vu dans le chapitre précédent qu'il est possible de garantir la performance des flots (*non-bottlenecked*) avec un dimensionnement correct du buffer. Des petites tailles suffisent tant que la charge n'est pas trop proche de 100%.

Ici, nous considérons le cas réaliste où de tels flots sont acheminés en présence de flots élastiques à plus fort débit, utilisant éventuellement des versions plus agressives de TCP, et saturant le lien (nous

avions nommé ce régime "transparent-élastique"). Ces flots *non-bottlenecked* ne réalisent pas leur débit équitable (*slow start*, débit d'accès limité ou congestion sur un autre lien). Ils ne participent pas activement à l'allocation des ressources réalisée par TCP, et sont éventuellement victimes des connexions à plus fort débit.

Les simulations présentées dans cette section correspondent à une charge *background* $\rho_b = 50\%$, générée par des arrivées Poissonniennes de flots TCP de taille exponentielle de moyenne 200ko et de débit crête 1Mb/s.

5.3.2 Performance des flots *non-bottlenecked*

Impact d'un flot TCP sur les flots *non-bottlenecked*

Dans les figures 5.1 et 5.2, nous représentons les résultats de simulation obtenus avec un flot *bottlenecked*, respectivement TCP Reno et HSTCP, pour B=20, 100 et 625 paquets (de gauche à droite), sur un lien FIFO. De haut en bas, nous avons : l'évolution de *cwnd* pour le flot TCP, l'évolution de la file d'attente, le taux de pertes de paquets pour chaque classe, et une représentation des pertes de chaque flot (le rectangle sur l'axe des abscisses représente les dates de début et de fin du flot, la position verticale est aléatoire, la hauteur représente le débit du flot, l'aire la taille transmise, et enfin le dégradé de couleur le taux de pertes subi (gris : pas de perte, rouge : faible taux de perte, noir : taux élevé).

L'évolution de *cwnd* dans les différentes configurations est caractéristique de l'algorithme AIMD au cœur du contrôle de congestion réalisé par TCP. Ce dernier est responsable de trois phénomènes qui affectent la performance réalisée par les flots *non-bottlenecked* :

- des **pertes de paquets**. Les figures 5.1 et 5.2 montrent les pertes pour chaque classe de flots, causées par la saturation du buffer lorsque la fenêtre de congestion devient trop importante. Ces instants de saturations sont d'autant plus fréquents que le protocole est agressif.
- des **délais** : un paquet sera retardé d'autant plus que la taille de la file d'attente sera importante à son arrivée (FIFO). Un buffer de 625 paquets (le BDP) représente par exemple une latence d'environ 100ms sur un lien à 50Mb/s (en fait, 1 RTT maximum tel que considéré pour le dimensionnement).
- de la **gigue** : elle est causée par les oscillations de la file d'attente.

Ces phénomènes sont aggravés lors des périodes de *slow-start* de TCP (croissance exponentielle de *cwnd*), ou avec l'utilisation de protocoles plus agressifs comme HSTCP.

Les taux de pertes subis par le flot permanent (*bottlenecked*) et par les flots dynamiques (*non-bottlenecked*) sont représentés respectivement en trait rouge plein, et en bleu pointillés. Ce taux peut être relativement élevé, de 10 à 30% pour TCP Reno en mode *congestion avoidance*, jusqu'à plus de 70% lors des périodes de *slow start*. Les figures montrent la répartition de ces pertes : même si elles sont distribuées sur un grand nombre de flots, elles affecteront néanmoins plus ou moins fortement leur performance en fonction des codecs utilisés (dans la figure du bas, la couleur rouge symbolise un faible taux de pertes, et noir un fort taux de pertes).

Les flots *non-bottlenecked* contiennent en partie des flots élastiques, pour lequels la considération du taux de pertes seul n'est pas suffisamment représentatif de la performance. Il s'agit de flots n'atteignant déjà pas le débit équitable, potentiellement interrompus pendant l'établissement de la connexion, la période de *slow-start*, etc. La conséquence pour ces flots sera un allongement de leur temps de transfert. Nous représentons en figure 5.3 la distribution du nombre de flots en cours, qui permet d'illustrer la performance réalisée. La durée moyenne d'un flot τ est associée au nombre moyen de flots N et à leur taux d'arrivée λ par la loi de Little : $\tau = N/\lambda$. L'allongement du temps de transfert d'un flot est reflété par le plus grand nombre de flots en cours. Les figures semblent indiquer que la performance est principalement dégradée par le délai subi par les flots (buffer), et par les pertes dans une moindre mesure (le protocole HSTCP étant plus agressif).

Petites tailles de buffers

L'impact du protocole semble atténué par la présence d'un petit buffer, certainement parce qu'il limite l'évolution de *cwnd*. Cela va dans le sens des propositions d'un réseau *Best Effort* avec de petits buffers, où le contrôle est effectué en bordure. On voit toutefois que des flots de fort débit crête peuvent dégrader la performance, notamment lors du *slow start*, et d'autant plus fortement qu'ils seront agressifs.

Tailles de buffers intermédiaires

Nous avons recommandé une taille de buffer B de l'ordre de 100 paquets comme un compromis raisonnable pour assurer la performance des flots élastiques (*bottlenecked* et *non-bottlenecked*). La perfor-

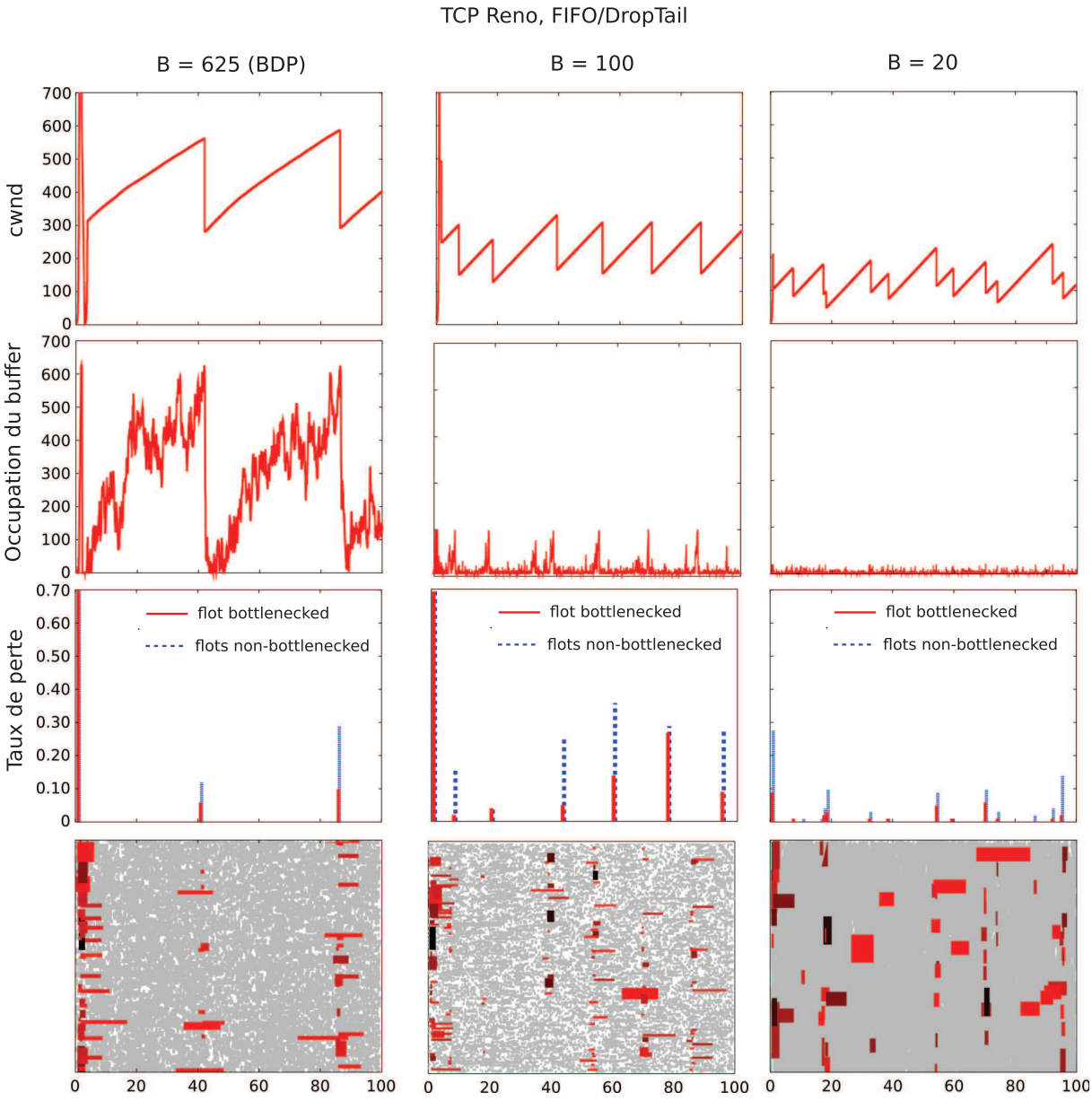


FIGURE 5.1 – Scénario impliquant un flot TCP Reno en compétition avec une charge *background* de 50%, sur un lien FIFO/DropTail, pour différentes tailles de buffer. De haut en bas : *cwnd* du flot TCP, occupation de la file d'attente, taux de pertes subit par les deux classes de flots, et représentation des pertes distribuées sur les flots *non-bottlenecked*

.

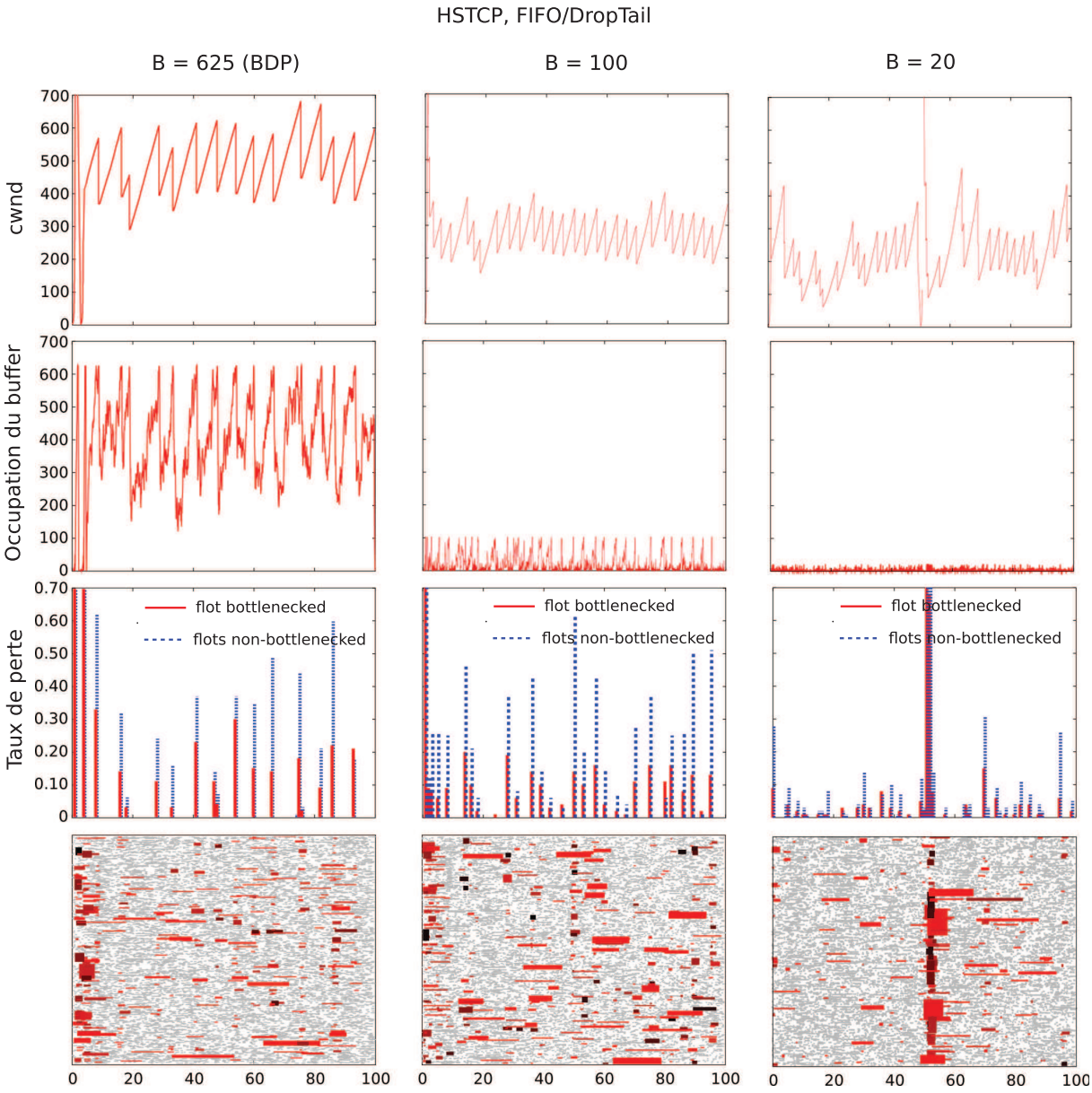


FIGURE 5.2 – Scénario impliquant un flot HSTCP en compétition avec une charge *background* de 50%, sur un lien FIFO/DropTail, pour différentes tailles de buffer. De haut en bas : *cwnd* du flot TCP, occupation de la file d'attente, taux de pertes subit par les deux classes de flots, et représentation des pertes distribuées sur les flots *non-bottlenecked*

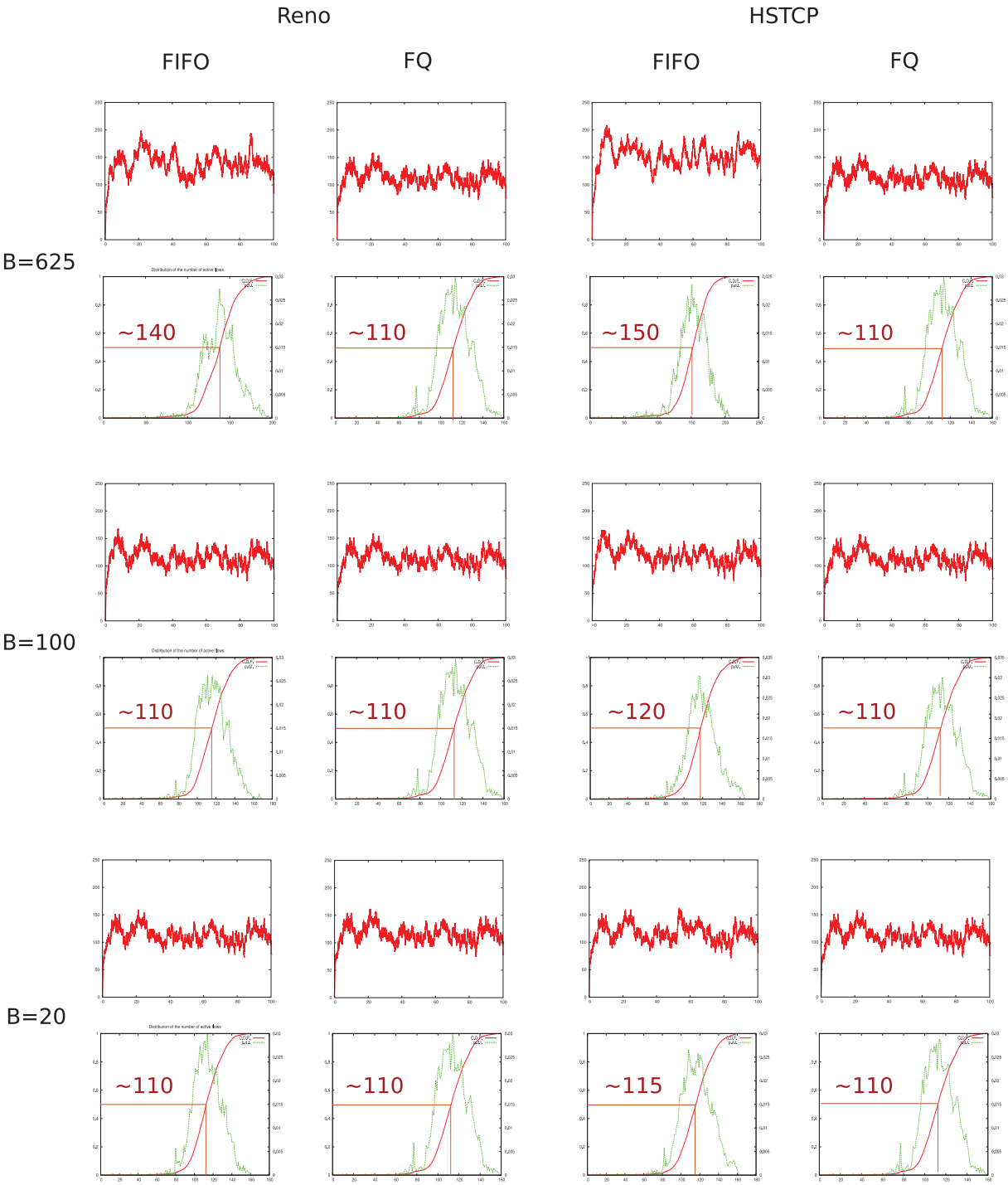


FIGURE 5.3 – Nombre de flots en cours (figure du haut), et densité/distribution/mediane (figure du bas), pour différents scénarios impliquant un flot TCP Reno ou HSTCP en compétition avec une charge *background* de 50%, sur un lien FIFO et FQ, pour différentes tailles de buffers.

mance réalisée par les flots *non-bottlenecked* est similaire au cas $B = 20$ avec l'utilisation de NewReno. Elle reste également acceptable pour HSTCP, malgré la présence de plus nombreux événements de saturation qui causent des pertes importantes.

Utilisation de *fair queueing*

Dans l'ensemble des scénarios FIFO considérés, la performance des flots est dégradée, c'est pourquoi nous recommandons l'utilisation de *fair queueing*. LQD élimine les pertes pour les flots *non-bottlenecked* (et permet de garantir un débit moyen équitable pour les autres), tandis que le rejet d'un paquet en tête de file (FrontDrop) permet une meilleure réactivité des flots TCP [147].

Les résultats utilisant FQ correspondant aux figures 5.1 et 5.2 ne sont pas représentés ici. Ils montrent que son utilisation permet de protéger les flots *non-bottlenecked* des pertes, et leur traitement en priorité assure des délais négligeables et une gigue bornée (conjecture de gigue négligeable), pourvu que la charge prioritaire reste contrôlée. Dans le cas de flots à débit crête limité, mais qui auraient accumulé de la gigue dans une autre partie du réseau, cette dernière sera absorbée puisque ces flots seront alors servis au débit équitable (voir Chapitre 6). Bonald *et al* [25] remarquent que l'ordonnancement joue un rôle plus important que le traitement en priorité. La figure 5.3 montre que FQ rétablit la performance optimale en ce qui concerne la distribution du nombre de flots en cours (ces flots sont alors uniquement limités par le *slow-start*).

Nous remarquons que la réalisation de FQ sur le lien permet ainsi le développement de nouvelles méthodes d'estimations fiables de la bande passante disponible, telles que *packet pair* [87], ainsi que de protéger l'ensemble des connexions établis d'un éventuel comportement agressif lors de cette phase de découverte. L'introduction de nouveaux mécanismes permettant une performance plus satisfaisante que les méthodes actuelles de *slow start* reste une piste ouverte de recherche.

5.3.3 Performance des flots *bottlenecked*

Le but de cette section est d'illustrer l'impact de l'introduction de nouveaux protocoles TCP. Nous présentons un certain nombre de résultats de simulation nous permettant d'établir un ensemble d'observations. Les figures présentées sont toujours sous forme de paire avec en haut l'évolution de *cwnd* (la taille de la fenêtre de congestion) pour chaque flot goulotté, et en bas l'utilisation du lien au fil du temps. Nous considérons généralement trois tailles de buffer : 20, 100 et 625 paquets, ce dernier correspondant à un dimensionnement selon la règle empirique dite du *Bandwidth Delay Product*. Dans l'ensemble de ces scénarios, la charge de trafic *background* offerte est de 50%.

Nous limitons dans cette section les résultats présentés au protocole HSTCP, que nous comparons avec TCP Reno. Les résultats sont qualitativement similaires avec l'utilisation des autres variantes de TCP présentées plus haut. Il conviendra de se référer par exemple à Li *et al.* [100] pour une telle étude. Notons toutefois dans nos scénarios la présence de trafic *background* qui affecte par exemple le débit réalisé par un flot en présence de petits buffers, ou permet l'ajout d'un certain aléa qui parfois aide à une convergence plus rapide.

Scénarios avec 1 flot *bottlenecked*

Nous commençons par un scénario identique aux conditions du chapitre précédent, où un seul flot TCP Reno est en compétition avec un trafic *background* de charge 50%, et faisons varier la taille du buffer.

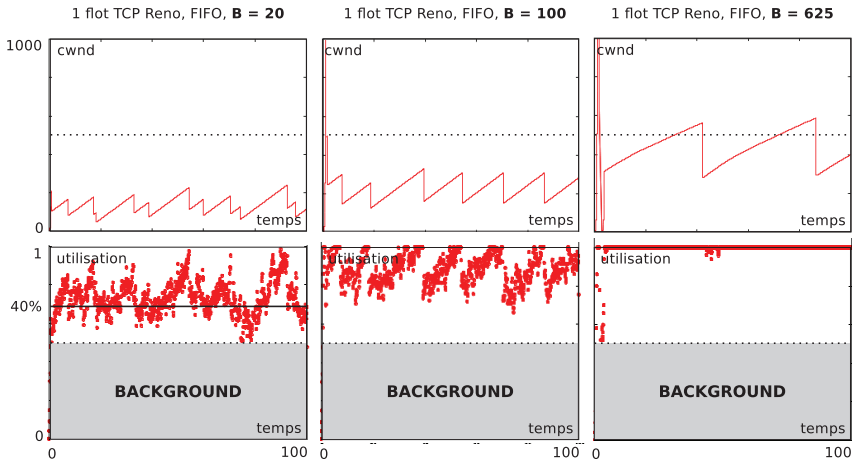


FIGURE 5.4 – Évolution de *cwnd* et utilisation de la bande passante disponible faite par une connexion TCP Reno en compétition avec une charge *background* de 50%, pour des tailles de buffer de 20, 100 et 625 paquets.

Les résultats présentés en figure 5.4 nous permettent de rappeler l'observation faite dans le chapitre précédent, à propos de TCP Reno :

Observation 1. *De très petits buffers de l'ordre de 20 paquets entrainent une perte importante de débit pour TCP Reno. Le flot n'utilise que 40% de la capacité disponible.*

Nous comparons la performance obtenue pour une taille de buffer de 100 paquets, entre TCP Reno et HSTCP (figure 5.5 à gauche).

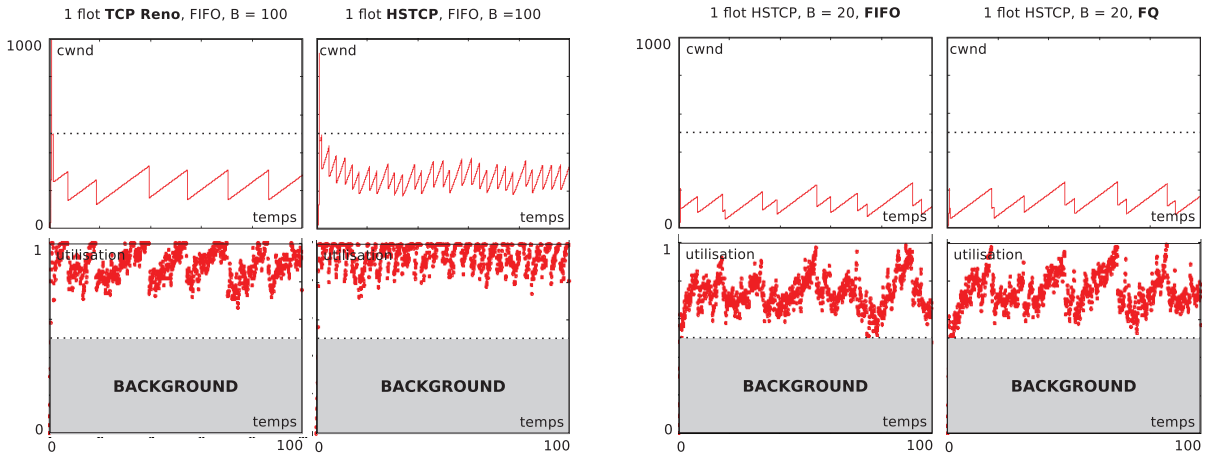


FIGURE 5.5 – Évolution de *cwnd* et utilisation de la bande passante disponible faite par une connexion TCP Reno et une connexion HSTCP en compétition avec une charge *background* de 50%, dans des scénarios FIFO et FQ impliquant plusieurs tailles de buffer.

Observation 2. *HSTCP augmente l'utilisation du lien, au détriment de pertes plus importantes pour les flots* background *(lors des instants de saturation du buffer).*

Nous considérons maintenant le comportement d'un flot HSTCP avec un buffer de 100 paquets, avec un lien FIFO et FQ (figure 5.5 à droite) :

Observation 3. *L'utilisation de fair queueing permet de protéger les flots* background *des pertes, avec un très faible impact sur le flot goulotté (qui semble même positif).*

Scénarios avec 2 flots *non-bottlenecked*

Nous considérons désormais des scénarios avec 2 flots TCP.

Équité intra-protocolaire : Nous comparons en figure 5.6, les comportements respectifs de deux flots TCP Reno (à gauche) et deux flots HSTCP (à droite), pour des tailles de buffer de 20 et 625 paquets, sur un lien FIFO.

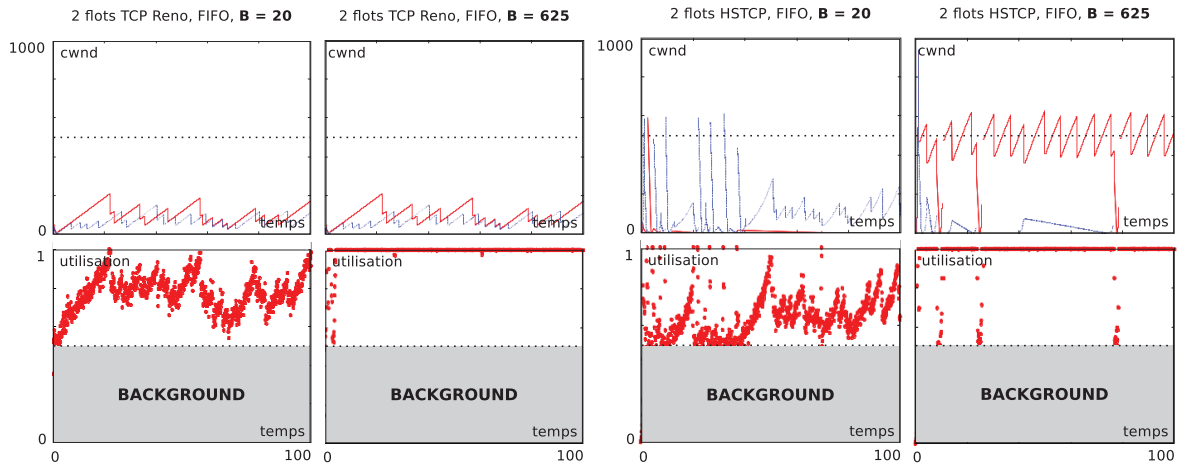


FIGURE 5.6 – Évolution de *cwnd* et utilisation de la bande passante disponible faite par deux connextions TCP Reno (à gauche) et deux connexions HSTCP (à droite), en compétition avec une charge *background* de 50%. Le lien est FIFO et les tailles de buffer de 20 et 625 paquets.

Observation 4. *TCP Reno offre une équité approximative.*

Observation 5. *HSTCP est très inéquitable même dans un contexte homogènes où tous les flots utilisent HSTCP.*

Equité inter-protocolaire : Dans la figure 5.7, nous représentons maintenant la performance réalisée par un flot TCP Reno en compétition avec un flot HSTCP, pour des tailles de buffer de 20 et 625 paquets sur un lien FIFO (les deux figures de gauche), et FQ (les deux figures de droite).

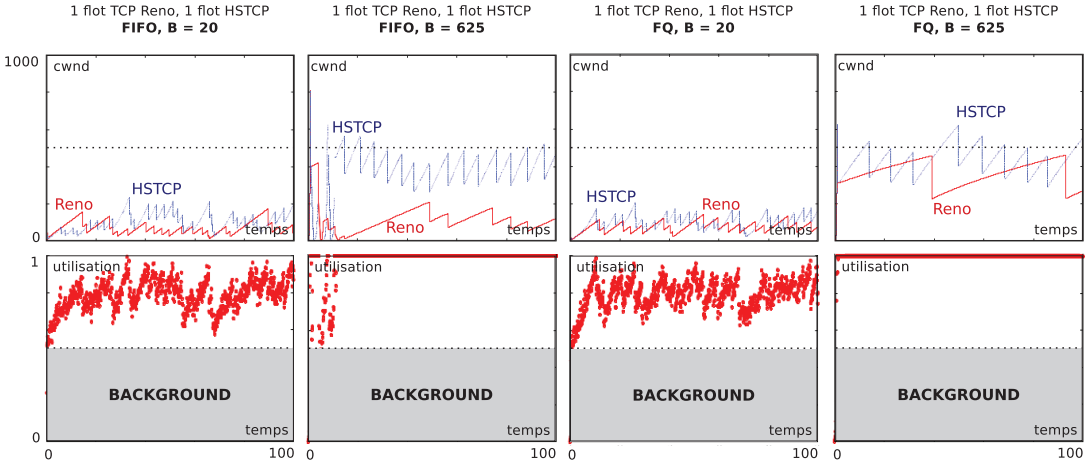


FIGURE 5.7 – Évolution de *cwnd* et utilisation de la bande passante disponible faite par un flot TCP Reno en compétition avec un flot HSTCP, en compétition avec une charge *background* de 50%. Le lien est respectivement FIFO (les deux figures de gauche) et FQ (les deux figures de droite), et les tailles de buffer considérées sont de 20 et 625 paquets.

Observation 6. *HSTCP est très inéquitable envers TCP Reno.*

Observation 7. *L'ajout de* fair queueing *est très efficace pour améliorer l'équité, bien que HSTCP soit plus agressif et obtiennent plus de débit. Il revient au protocole de transport d'utiliser au mieux la capacité qui lui est offerte. Ici AIMD ne récupère que très lentement son débit après une perte.*

Scénarios avec 4 flots goulottés

Nous comparons maintenant, en figure 5.8, les comportements respectifs de 1 et 4 flots TCP Reno, pour des tailles de buffer de 20 paquets (les deux figures de gauche), et 625 paquets (les deux figures de droite).

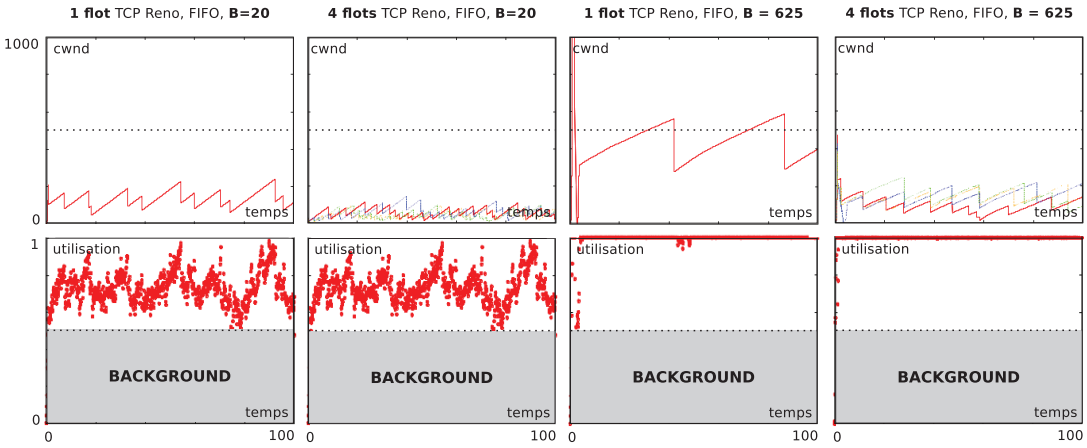


FIGURE 5.8 – Évolution de *cwnd* et utilisation de la bande passante disponible faite par 1 et 4 flots TCP Reno en compétition avec une charge *background* de 50%. Le lien est FIFO, et les tailles de buffer considérées sont de 20 paquets (les deux figures de gauche) et 625 paquets (les deux figures de droite).

Observation 8. *Pour TCP Reno, l'équité est généralement approximative avec de petits ou de grands buffers. Elle est meilleure lorsque le nombre de flots concurrents augmente.*

Nous effectuons la même simulation en figure 5.9 avec 1 flot (figure de gauche) et 4 flots HSTCP (figure du milieu), pour une taille de buffer de 20 paquets, et un lien FIFO.

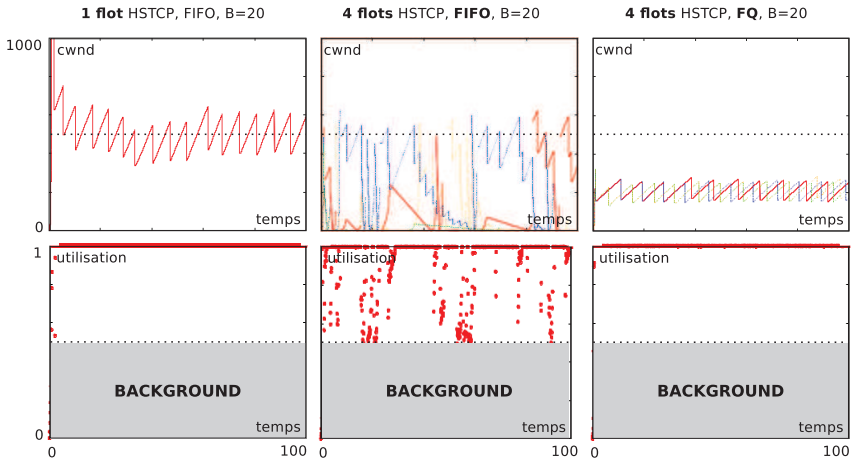


FIGURE 5.9 – Évolution de *cwnd* et utilisation de la bande passante disponible par 1 flot (à gauche) et 4 flots HSTCP (au milieu) sur un lien FIFO, et par 4 flots HSTCP sur un lien FQ (à droite). Les flots sont en compétition avec une charge *background* de 50%, et la taille de buffer considérée est de 20 paquets.

Observation 9. *En présence de 4 flots HSTCP, on observe un comportement assez chaotique des flots, où chaque flot domine temporairement l'un après l'autre. Aussitôt qu'un flot possède une grande taille de fenêtre cwnd, il tend à être plus agressif et présence son avantage relatif jusqu'à ce qu'il soit éventuellement détrôné par un autre. Ce genre de comportement à été observé précédemment par les flots un flot après l'autre. Le manque d'équité ainsi que la perte de débit ont également été observés par Li* et al. *[100].*

Enfin, la partie la plus à droite de la figure 5.9 présente un scénario impliquant 4 flots HSTCP, toujours pour un buffer de 20 paquets, mais fois-ci en présence de FQ.

Observation 10. *L'introduction de* fair queueing *rétablit l'équité et permet de préserver le débit des flots et une utilisation optimale du lien. Il atténue considérablement l'impact de l'arrivée de nouveaux flots sur ceux déjà en place, pour lesquels le comportement agressif du* slow start *conduirait à des événements de perte importants.*

Observation 11. *Deux avantages supplémentaires du* fair queueing *sont des événements de pertes désynchronisés en raison de l'isolation entre les flots, et le fait que les émissions de paquets se retrouvent espacées au débit équitable courant. Ce dernier effet signifie que les acquittement sont également lissés, ce qui conduit à moins de rafales dans le prochain cycle d'émissions de paquets. Ces deux phénomènes semblent améliorer la performance du partage de bande passante, notamment pour de petites tailles de buffers.*

5.4 Comportement de TCP dans un environnement équitable

5.4.1 Motivations

Les résultats préliminaires que nous venons d'établir mettent en évidence les problèmes liés à la présence de flots TCP à fort débit. D'une part, le manque d'isolation entre les flots entraîne des perturbations pour les flots streaming, qui devraient être protégés (et qui ont déjà un faible débit). D'autre part, contrairement aux hypothèses souvent faites, les différentes versions de TCP ne permettent pas un partage équitable, efficace et instantané de la capacité disponible.

TCP

Une partie de ces imperfections provient de la multiplicité des rôles que doit assurer le protocole. En plus de garantir une connexion fiable entre les hôtes (grâce aux mécanisme d'acquittement et de retransmission), TCP a pour objectif (au travers de l'algorithme AIMD pour sa version classique) la réalisation :

- d'une utilisation optimale de la bande passante disponible sur le lien (nous avons vu précédemment que TCP s'appuie sur la présence de buffers suffisamment dimensionnés) ;
- de l'allocation équitable entre les différentes connexions de cette capacité disponible (et de l'espace buffer).

Cette dernière tâche repose généralement sur la détection par TCP de la congestion d'un lien, par des heuristiques basées sur les pertes de paquets ou les délais constatés. Elle nécessite de plus la coopération des utilisateurs, et ne tient pas compte de protocoles non-réactifs tels qu'UDP. En particulier, une propriété souvent recherchée dans la conception de nouveaux protocoles – et potentiellement limitante – est la rétro-compatibilité avec TCP Reno (TCP *friendliness*) qui garantit qu'un tel protocole en compétition avec un flux TCP Reno sera le plus équitable possible envers lui. HSTCP par exemple possède deux régimes opérationnels, l'un équivalent à TCP Reno, l'autre plus agressif. Les résultats présentés dans le Chapitre 3, d'insensibilité notamment, reposent sur un partage équitable des ressources.

Fair Queueing

La discipline FIFO ne permet pas d'éviter les pertes de paquets dues à la saturation du buffer. Nous avons vu dans le Chapitre 3 que la présence de *fair queueing* au sein du réseau permettrait de mieux répondre à ces problèmes, sans nécessiter de mécanisme complexe de contrôle de trafic, et qu'un tel ordonnanceur est réaliste dans un contexte de cœur de réseau. Les simulations précédentes illustrent les bénéfices d'une telle proposition. Les flots sont isolés entre eux, ce qui assure notamment l'absence de perte et des délais négligeables pour les flots *streaming*. De plus, des flux de nature très hétérogènes peuvent coexister et se partager équitablement la bande passante.

Nous avons également vu qu'un tel mécanisme est fondamental afin de dimensionner et d'opérer un réseau de manière robuste et prévisible, en suivant des modèles simples de trafic. Il permet de rendre réalistes les hypothèses d'équité entre les flots et de convergence rapide vers l'état d'équilibre. Dans la suite, nous allons explorer la faculté des différents algorithmes à exploiter efficacement la bande passante disponible. Notre objectif ici est de démontrer que l'introduction de *fair queueing* permet le développement de nouveaux protocoles TCP à la fois simples et efficaces, sans nuire à l'existant.

5.4.2 Environnement de simulation

Nous considérons l'environnement de simulation introduit dans la section précédente, avec les modifications suivantes :

Approximation Poisson pour les flots *non-bottlenecked*

Comme dans le chapitre précédent, section 4.5.1, nous considérons des arrivées Poisson de paquets générant une charge *background* $\rho_b \in [0,1]$. Ce trafic modélise l'agrégat de flots à débit limité p. Tant que le ratio C/p n'est pas trop élevé, le lien voit un paquet de chaque flot de temps en temps et tous seront traités en priorité par l'algorithme de *fair queueing* PFQ.

Flots TCP

En plus des algorithmes TCP présentés en début de ce chapitre, nous considèrerons des généralisations des algorithmes AIMD et MIMD que nous avons développées pour lesquels il nous sera possible de faire varier les coefficients d'incrément et de décrément.

Bien que les différents algorithmes évalués soient paramétrés pour être *TCP-friendly*, leur diversité nous permet de balayer une large gamme de mécanismes de contrôle de congestion (voir Section 5.2.2). Notre évaluation porte notamment sur l'évolution de la performance du protocole en fonction de la capacité C du lien et du RTT des flots au travers de leur produit $C \times RTT$ (ou *Bandwidth Delay Product*), la taille B du buffer, ainsi que la charge ρ_b des flots *non-bottlenecked* concurrents.

Nous ne considérons pas des flots de RTT différents dans cette section ; leur instant de démarrage sera tiré aléatoirement afin d'éviter des phénomènes artificiels de synchronisation. La présence de *fair queueing* suffira ensuite à prévenir de tels cas.

5.4.3 Modèle d'étude

Débit instantané des flots

Nous avons introduit le paramètre γ, représentant l'espérance du débit instantané, dans le Chapitre 3, Section 3.3.2. Sa formulation requérait toutefois l'équité des débits des flots, qui n'était qu'approximative pour TCP Reno. Cette hypothèse sera ici assurée par la présence de *fair queueing*, pourvu que les protocoles soient suffisamment efficaces pour exploiter la bande passante qui leur est allouée.

Nous rappelons ici l'expression de γ :

$$\gamma = \frac{\rho}{E[X](\rho)} = \frac{\sum \rho^i / \Phi(i)}{\sum i \rho^{i-1} / \Phi(i)}. \tag{5.1}$$

avec $\Phi(i) = \prod_1^j \phi(j)$, et $\phi(i)C$ le taux de service du lien lorsque i flots sont en cours.

γ est une fonction rationnelle n'admettant pas 0 pour pôle, elle est donc développable en série entière en 0 telle que : $\sum_{k=0}^{\infty} \beta_k \rho^k$. On a en particulier : $\gamma = \phi_1 + (1 - 2\phi_1/\phi_2)\rho + o(\rho^2)$.

Il est possible d'en déduire le temps moyen de complétion d'un flot, $R(s)$:

$$R(s) = \frac{s}{\gamma(\rho)}$$

.

Dans les simulations mettant en œuvre TCP Reno, la courbe gamma était approximativement linéaire avec pour ordonnée à l'origine $\phi(1)$. Ce n'est généralement pas le cas. L'expression de γ n'est pas surprenante dans la mesure où à faible charge, la probabilité d'avoir 1 ou 2 flots est largement dominante.

Remarques

Nous ferons, comme dans le chapitre précédent, l'hypothèse simplificatrice qui consiste à négliger le temps de convergence des protocoles vers un état équitable. Ce modèle nous permet toutefois d'obtenir une estimation qualitative de la performance réalisée par les différents protocoles TCP, plus juste que certains modèles considérant uniquement des flots permanents et sans charge *background*.

Nous remarquons qu'à faible charge, on a très peu de flots et le changement de débit important dû à une arrivée ou un départ est balancé par un taux d'arrivée faible, et un taux de départ réduit (faible efficacité). A forte charge, les taux de départ et d'arrivée sont importants, mais le grand nombre de flots en présence induit de faibles variations de débit. Il sera intéressant de voir dans quelle mesure ces phénomènes ont un impact sur la performance effectivement réalisée.

5.4.4 Résultats

Dans cette discussion, nous allons présenter les principales observations faites à partir de ces simulations.

Performance des différentes versions de TCP avec de petits buffers

La figure 5.10 représente les valeurs de ϕ et γ pour notre scénario de référence ($B = 20$ paquets, et $\rho = 0.5$). Comme dans le chapitre précédent, la performance théorique du modèle PS est représentée en noir.

Pour l'utilisation de la capacité disponible $\phi(.)$, nous avons représenté l'axe des abscisses – le nombre de flots en cours – selon une échelle logarithmique. Un nombre faible de flots correspond en effet aux cas les plus probables pour des charges pas trop élevées. Nous nous limitons uniquement à une valeur indicative de la performance, et n'avons pas représenté les intervalles de confiance sur la figure à des fins de clarté.

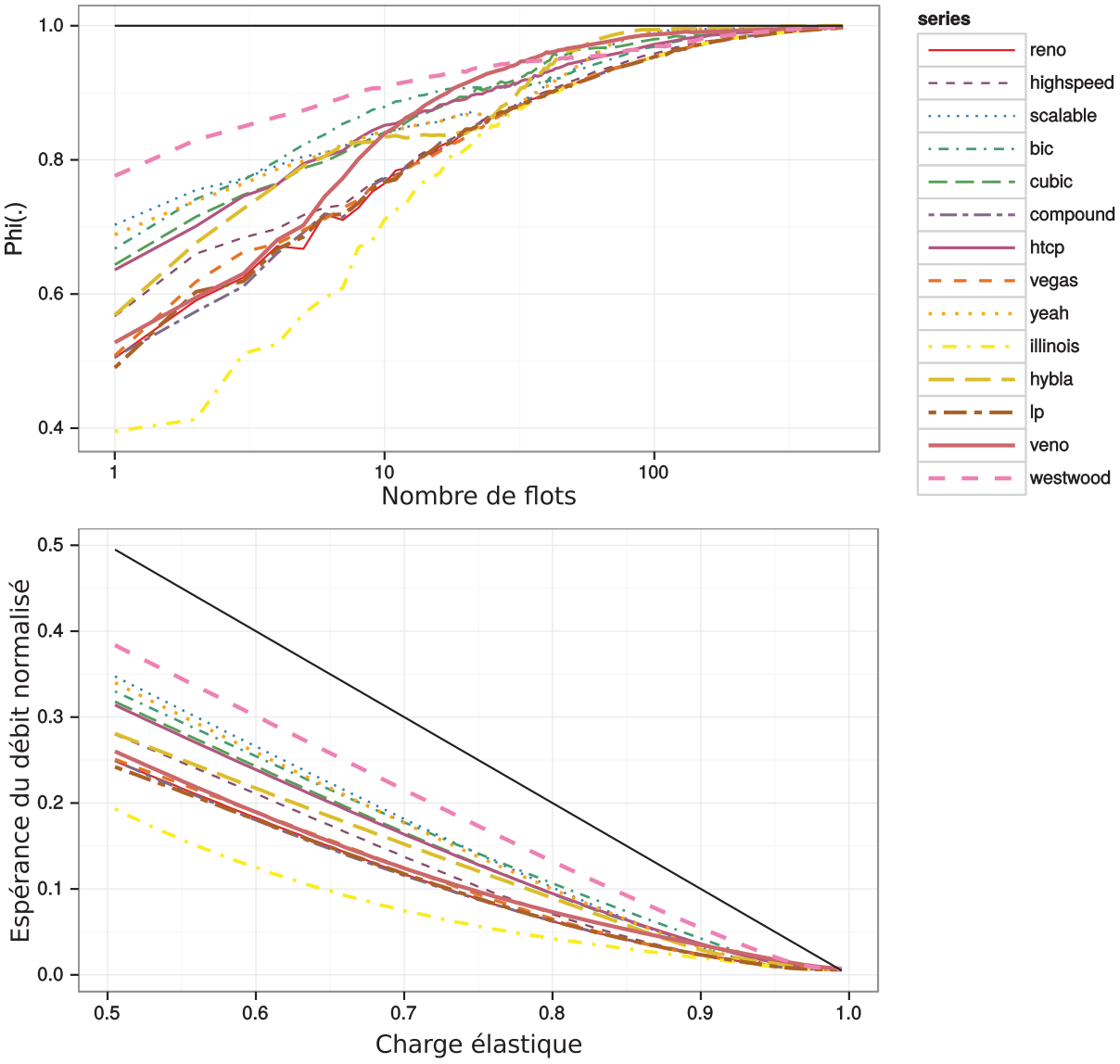


FIGURE 5.10 – Performance des différentes versions de TCP, $B = 20$, $\rho = 0.5$. La valeur théorique pour le modèle PS est représentée en noir.

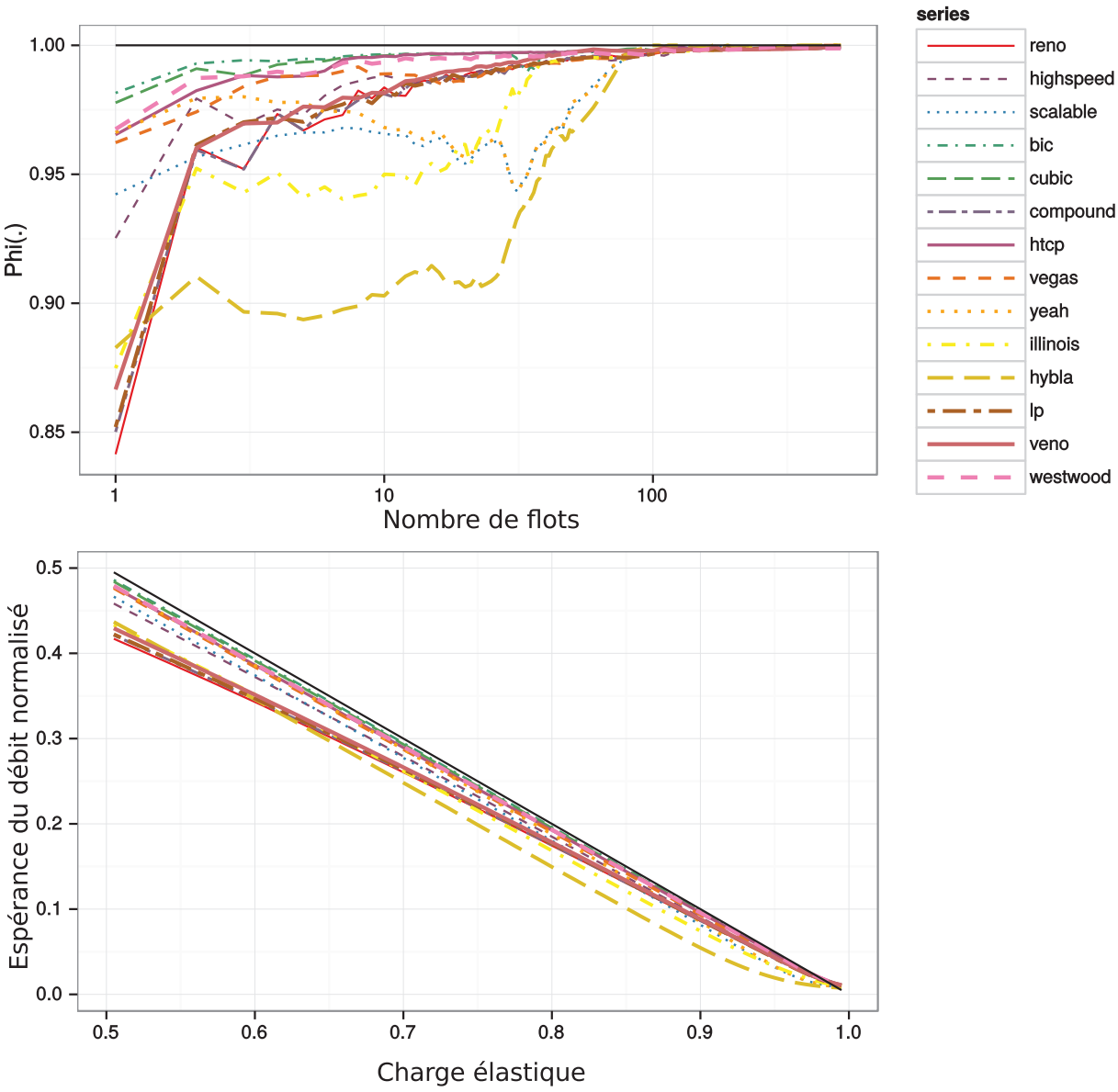


FIGURE 5.11 – Performance des différentes versions de TCP, $B = 100$, $\rho = 0.5$. La valeur théorique pour le modèle PS est représentée en noir.

Nous remarquons que l'utilisation varie du simple au double, avec $\phi(1) \sim 0.4$ pour TCP Hybla, et $\phi(1) \sim 0.8$ pour TCP Westwood ; et qu'elle croît dans tous les cas lentement, en fonction du nombre de flots en présence.

La plupart des protocoles permettent d'améliorer la performance obtenue par TCP Reno. Celle-ci est d'ailleurs légèrement supérieure à celle obtenue dans le chapitre précédent, sans l'utilisation de *fair queueing*. Ceci peut s'expliquer par un lissage des rafales de paquets par l'ordonnanceur, ce qui permet à la fenêtre de congestion de croître plus longuement avant de causer des pertes de paquets.

L'utilisation d'une taille plus importante de buffer ($B = 100$ paquets) est représentée en figure 5.11. Cette fois-ci, l'utilisation dépasse les 85%, voire 95% pour la plupart des protocoles. Nous remarquons que ce sont cette fois-ci les protocoles BIC et CUBIC qui obtiennent une meilleure performance, et qu'ils semblent ainsi plus sensibles à de petits buffers. Avec un choix correct de protocoles et une taille de buffers de 100 paquets, on obtient ainsi une performance satisfaisante (elle était déjà correcte pour TCP Reno).

Alors que ce n'était pas le cas pour $B = 20$ paquets, γ décroit ici de manière aproximativement linéaire à la charge.

Le cas d'un buffer dimensionné au *Bandwith Delay Product* n'est pas représenté ici mais il permet une performance optimale pour l'ensemble des protocoles.

Nous avons enfin représenté, en figures 5.12 et 5.13, la relation entre entre l'espérance du débit normalisé pour 1 flot (en abscisse) et le taux de pertes subit (en ordonnée), pour des tailles de buffer

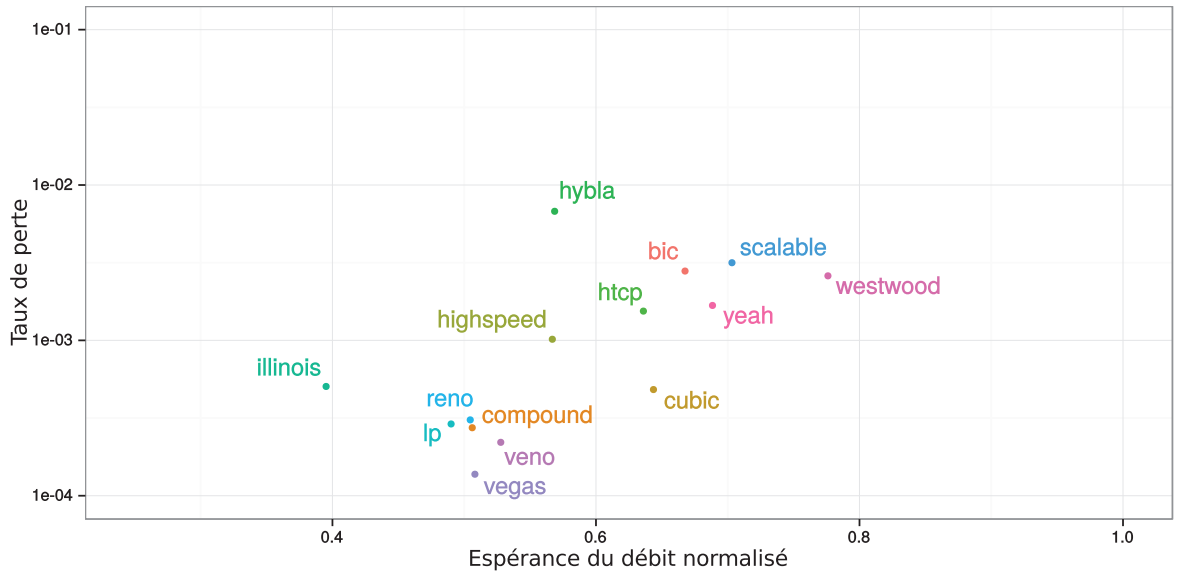


FIGURE 5.12 – Performance réalisée par les différents protocoles TCP pour une taille de buffer $B = 20$ paquets : compromis entre l'espérance du débit normalisée, pour 1 flot (en abscisse) et le taux de pertes subit (en ordonnée).

respectives de 20 et 100 paquets. Le fonctionnement interne de chaque protocole aboutit à un couple débit/taux de perte dépendant de l'environnement considéré, certains étant plus adaptés que d'autres à de petits buffer. Cette figure illustre en outre le compromis nécessaire entre le comportement du protocole – son agressivité/opportunisme pour émettre des paquets – et le débit réalisé. Nous reviendrons plus tard sur ce compromis, sachant qu'un taux de perte cible minimum semble inévitable dans un tel contexte.

Impact de la charge *background*

Comme attendu, on constate une forte sensibilité à la charge *background* de l'ensemble des protocoles pour $B = 20$ paquets. L'ordre entre les protocoles reste sensiblement le même pour $\rho_b = 0.25$, $\rho_b = 0.50$ et $\rho_b = 0.75$.

Avec une taille $B = 100$ paquets, on obtient une performance acceptable pour tous les protocoles. L'ordre est majoritairement conservé, mais on constate que les protocoles comme BIC, CUBIC, H-TCP et Scalable TCP arrivent cette fois-ci en tête en même temps que Westwood.

La performance de Westwood avec de faibles buffers n'est pas surprenante puisque ce protocole est une adaptation de Reno qui estime la bande passante disponible à partir du flux des ACKs, et ajuste la diminution de *cwnd* en fonction. Ce mécanisme lui permet d'être moins sensibles aux pertes aléatoires liées aux petits buffers, qui n'ont aucun rapport avec la congestion. Nous verrons dans la suite l'importance du paramètre de réduction de la fenêtre.

Sensibilité des protocoles au dimensionnement

Nous comparons la performance de Reno avec celle obtenue pour Westwood, qui est l'algorithme offrant le meilleur compromis pour différentes configurations de charge *background* et de buffer. La figure 5.14 présente le débit atteint par une connexion TCP Reno (à gauche) et Westwood (à droite), en fonction de ρ_b (en abscisse) et B (en ordonnée).

Nous considérons d'abord le cas $C = 50$Mb/s. Pour une taille $B = 20$ paquets, Westwood offre une performance acceptable jusqu'à des charges d'environ 50%, alors que le débit d'un flot TCP Reno est fortement dégradé. Les besoins en buffer du protocole sont ainsi beaucoup plus réduits, de par sa résistances aux pertes aléatoires. Un buffer d'une centaine de paquets convient, pour tous les seuils de charge *background*.

A plus haute capacité ($C = 100$Mb/s), Westwood réalise toujours une bien meilleure performance que Reno. La performance, moyenne pour $B = 20$, devient rapidement bonne au-delà.

Étant basé sur TCP Reno, Westwood souffre également de la lenteur de la croissance additive de *cwnd*, qui n'est pas trop visible dans ces simulations avec un seul flot stationnaire. Ceci explique

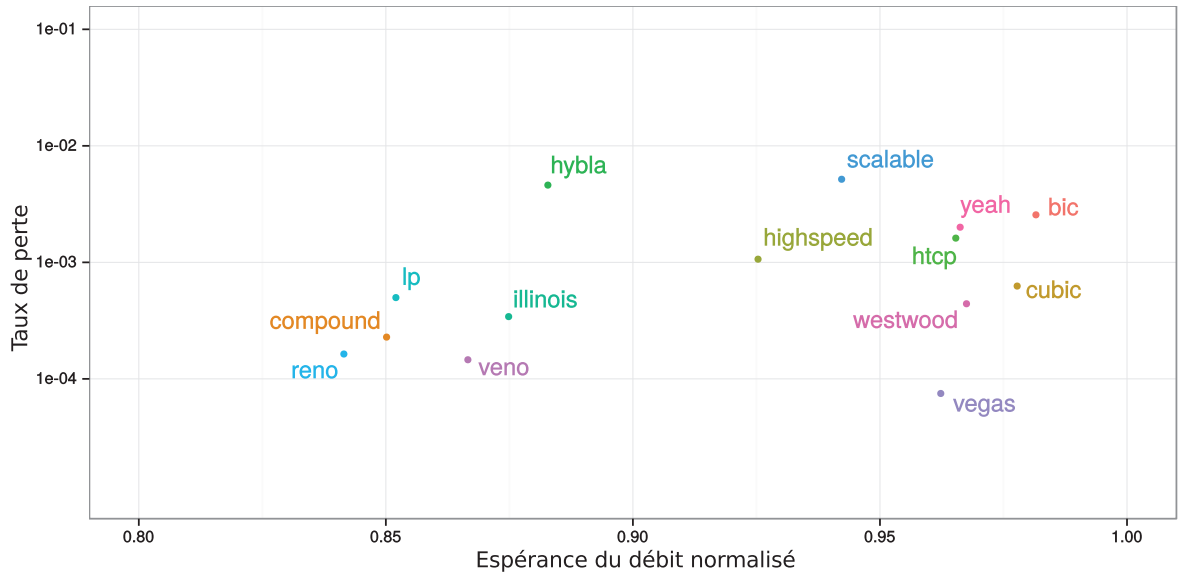
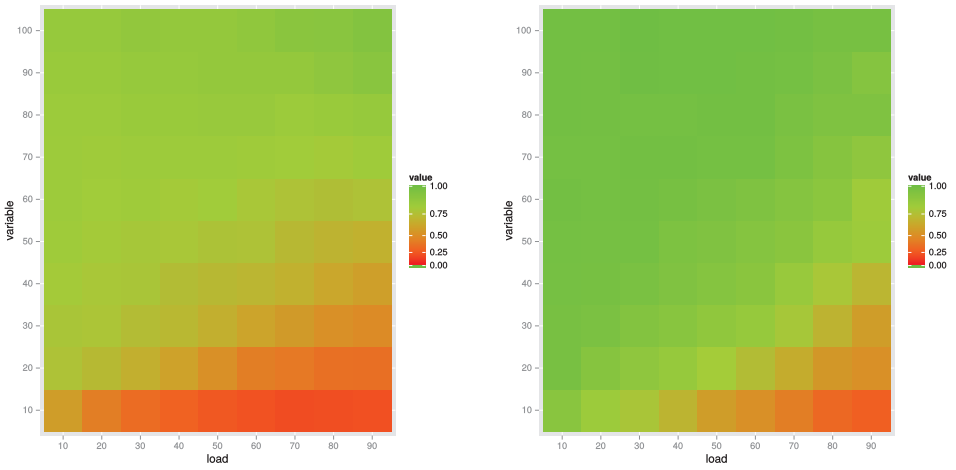
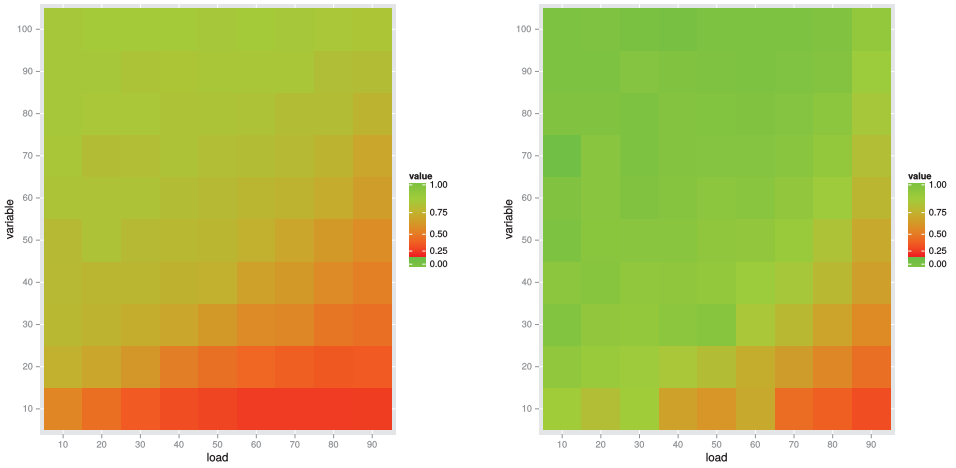


FIGURE 5.13 – Performance réalisée par les différents protocoles TCP pour une taille de buffer $B = 100$ paquets : compromis entre l'espérance du débit normalisé, pour 1 flot (en abscisse) et le taux de pertes subit (en ordonnée).



(a) TCP Reno (b) TCP Westwood

FIGURE 5.14 – Performance de TCP pour différentes valeurs de B et ρ_b, pour $C = 50$Mb/s



(a) TCP Reno (b) TCP Westwood

FIGURE 5.15 – Performance de TCP pour différentes valeurs de B et ρ_b, pour $C = 100\text{Mb/s}$

probablement la dégradation lors de l'augmentation de la capacité.

Versions plus agressives AIMD et MIMD

Les nouvelles versions de TCP sont généralement conçues avec le double objectif de respecter l'équité avec TCP Reno, et avec d'autres connexions utilisant le même protocole : le protocole doit permettre à un nouvel entrant de récupérer de la bande passante, tandis que ce dernier ne doit pas dégrader la performance des connexions en cours. Cela limite nécessairement l'efficacité du protocole, qui ne peut être suffisamment agressif dans les phases *slow start* et *congestion avoidance*. Ces limites n'ont plus de raison d'être si le réseau réalise une allocation équitable.

Nous avons évalué 2 versions d'un protocole TCP plus agressif, basées respectivement sur des algorithmes AIMD – inspiré de TCP Reno – et MIMD – inspiré de Scalable TCP, pour lesquels il nous est possible de faire varier les paramètres de croissance (α) et de décroissance (β) de la fenêtre. Les figures 5.18 et 5.19 représentent respectivement le taux d'utilisation et le taux de pertes de paquets obtenus pour différentes configuration de α et β, le débit cible étant de 25Mb/s.

AIMD : Nous remarquons que les performances peuvent être sensiblement améliorées avec une plus grande valeur de α et une plus faible valeur de β. Nous passons par exemple d'environ 12Mb/s avec les valeurs par défaut de Reno ($\alpha = 1$, $\beta = 2$) à plus de 19Mb/s pour $\alpha = 5$ et $\beta = 10$.

C'est le paramètre de décroissance qui semble jouer le rôle le plus important : il permet à l'algorithme d'être **moins sensible aux pertes** de paquets, qui sont "normales" pour de petites tailles de buffers. L'amélioration de la performance se fait au prix d'une augmentation du taux de pertes, qui reste cependant raisonnable pour les configurations que nous avons évaluées. On peut cependant s'attendre à ce qu'un tel algorithme soit dépendant de la capacité du lien (*additive increase*).

Les valeurs trop importantes de α semblent causer un taux important de pertes, et ne permettent pas à la connexion d'utiliser plus de bande passante. Ces pertes sont probablement dues à l'émission de **rafales** de paquets d'autant plus importantes que le coefficient α est élevé.

MIMD : L'impact de MIMD est plus important : on retrouve un débit de 17Mb/s pour les valeurs par défaut de Scalable TCP ($\alpha = 50$ et $\beta = 3$), qui peut aller jusqu'à 19Mb/s pour des protocoles plus agressifs. On remarque cependant que le taux de pertes croît rapidement avec des valeurs plus agressives de α et β, pour dépasser le taux généralement accepté de 1%. On retrouve le même phénomène de sensibilité aux rafales que décrit précédemment.

Il convient de remarquer que l'augmentation de l'agressivité du protocole, en diminuant la réduction de la fenêtre lors de la détection d'une perte, entraînera de plus forts taux de pertes lorsqu'un nouveau flot entrera en compétition.

La conception d'un protocole adapté à de petites tailles de buffer ($B = 20$) semble cependant un objectif réaliste. Le besoin d'un protocole agressif dans un tel contexte implique un mécanisme de lissage des envois de paquets, que nous considérons ci-après.

Pour de petites valeurs de buffers, il semble difficile de faire beaucoup mieux avec MIMD que Scalable TCP, sauf à accroître au delà de 1% le taux de pertes accepté. Nous soulignons toutefois que les auteurs recommendent l'utilisation de *pacing* avec un algorithme MIMD, pour éviter la formation de rafales.

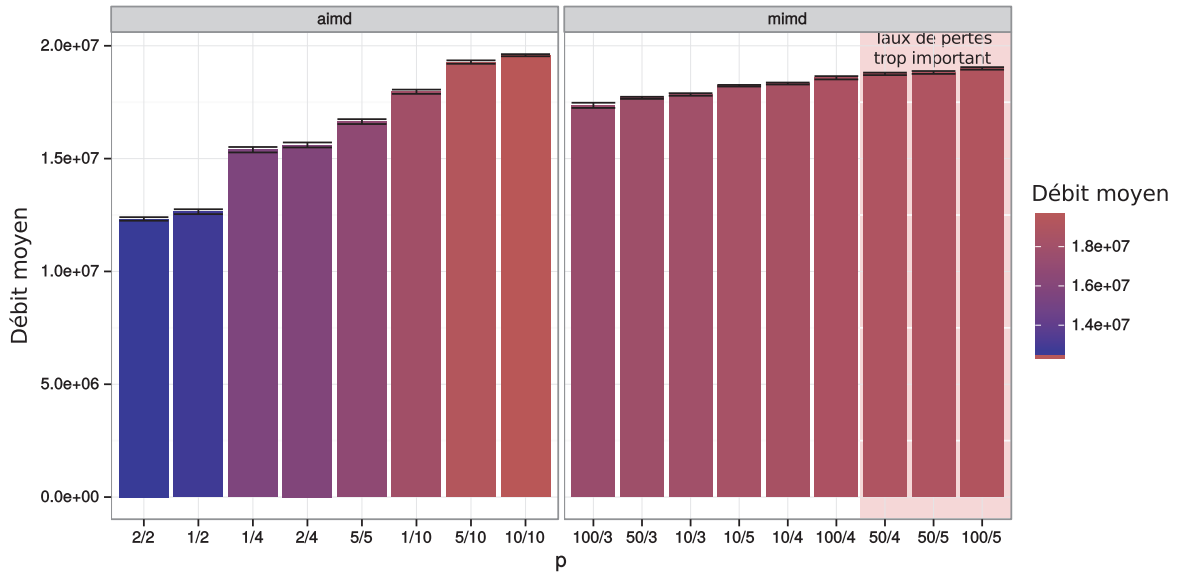


FIGURE 5.16 – Débit réalisé par différents paramétrages des algorithmes AIMD (sur la gauche) et MIMD (sur la droite) pour une taille de buffer $B = 20$ paquets : les coefficients d'incrément (α) et de décrément (β) sont représentés sous la forme α/β.

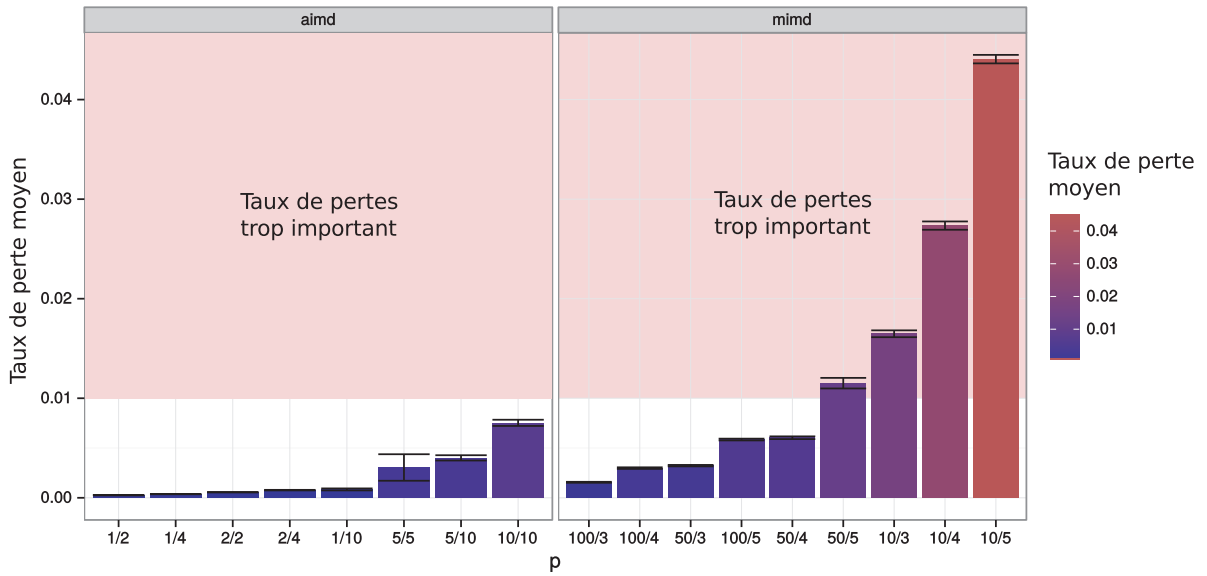


FIGURE 5.17 – Taux de pertes réalisé par différents paramétrages des algorithmes AIMD (sur la gauche) et MIMD (sur la droite) pour une taille de buffer $B = 20$ paquets : les coefficients d'incrément (α) et de décrément (β) sont représentés sous la forme α/β.

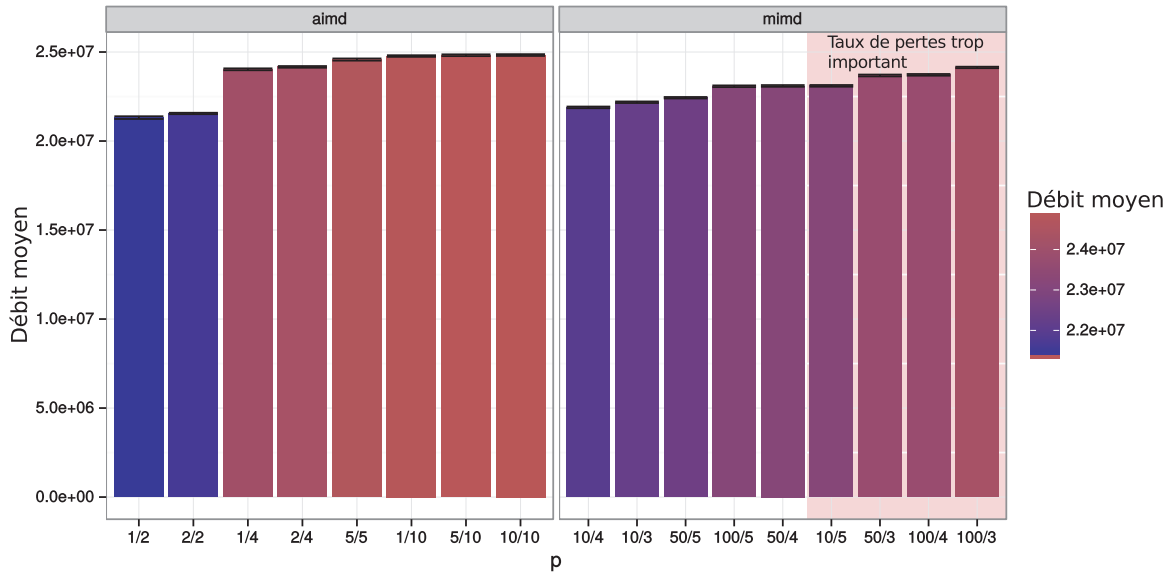


FIGURE 5.18 – Débit réalisé par différents paramétrages des algorithmes AIMD (sur la gauche) et MIMD (sur la droite) pour une taille de buffer $B = 100$ paquets : les coefficients d'incrément (α) et de décrément (β) sont représentés sous la forme α/β.

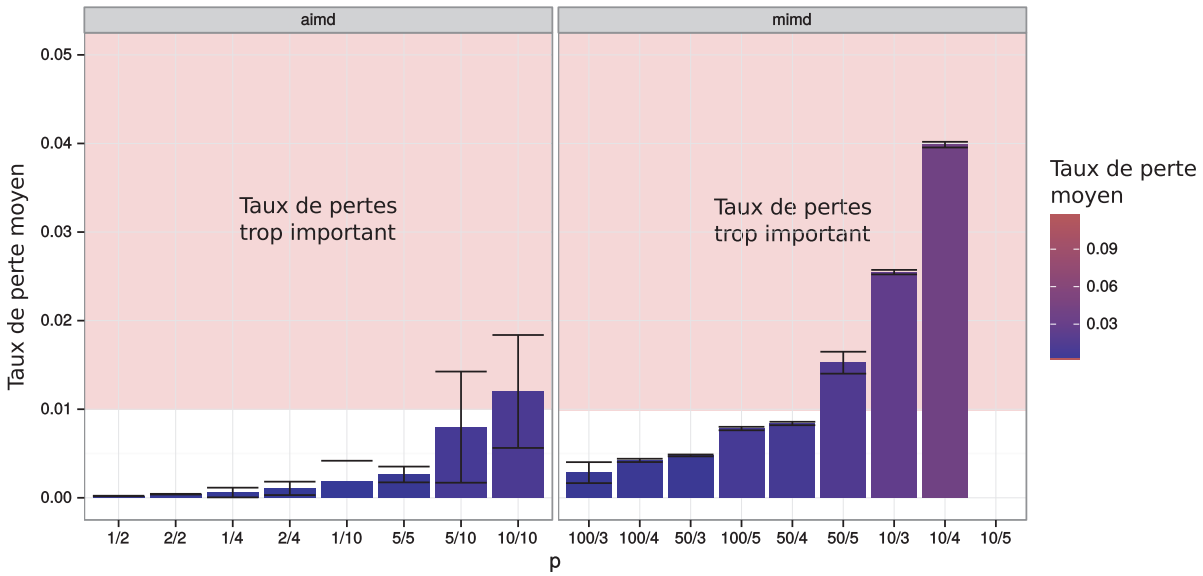


FIGURE 5.19 – Taux de pertes réalisé par différents paramétrages des algorithmes AIMD (sur la gauche) et MIMD (sur la droite) pour une taille de buffer $B = 100$ paquets : les coefficients d'incrément (α) et de décrément (β) sont représentés sous la forme α/β.

L'algorithme TCP Westwood ressort de nos simulations comme le plus adapté aux différentes conditions envisagées, notamment avec des capacités élevées et de petites tailles de buffers. Son équité avec TCP [64] en fait un protocole de choix pour diverses configurations de réseaux, réalisant ou non un ordonnancement *fair queueing*.

Espacement des paquets (*pacing*)

Enachescu *et al.* [50] proposent que les flots qui n'ont pas un débit crête limité par leur lien d'accès utilisent du *pacing*. Un tel mécanisme atténuerait en effet la perte de débit pour de petites tailles de buffers, qui se produisent lorsque le contenu d'une fenêtre de congestion est émis en rafale au début de chaque intervalle de RTT.

Bien que recommandé, aucune implémentation efficace de *pacing* n'a été disponible jusqu'à très récemment [48]. Les implémentations précédentes utilisaient généralement soit des *timers* coûteux pour chaque paquet, soit des spécificités de la couche Ethernet (datagramme PAUSE), perdant ainsi l'accès aux flots individuels. La réalisation faite dans [48] utilise un algorithme proche d'un ordonnanceur *fair queueing*. Elle offre à la couche applicative une limitation optionnelle des débits des flots utilisant un seul *timer*, mais malheureusement n'est pas intégrée avec les versions de TCP.

D'autres résultats discutent de l'intérêt du *pacing* [152], qui peut entraîner une synchronisation des flots, voire une certaine iniquité due à un nombre accru de pertes indépendantes. Nous pensons que l'utilisation de *fair queueing* permet d'éviter de telles situations, et d'exploiter les avantages du *pacing*.

Faute d'implémentation disponible (à la fois en simulation ou dans le noyau Linux), nous nous contentons d'un simple scénario illustratif montrant le gain en débit que peut obtenir un flot TCP *paced*. Il serait intéressant d'étendre plus en détail ces résultats avec les algorithmes AIMD et MIMD plus agressifs introduits précédemment, ainsi que différentes variantes de *pacing*.

Les figures 5.20 illustrent l'évolution de *cwnd* pour un seul flot *bottlenecked* avec[2] ou sans *pacing*, avec $B = 20$ et $B = 100$. Les résultats confirment que le *pacing* améliore sensiblement les performances obtenues pour de petites tailles de buffers puisque le débit obtenu est en gros deux fois plus élevé. La différence est négligeable pour une taille de buffer plus grande, $B = 100$.

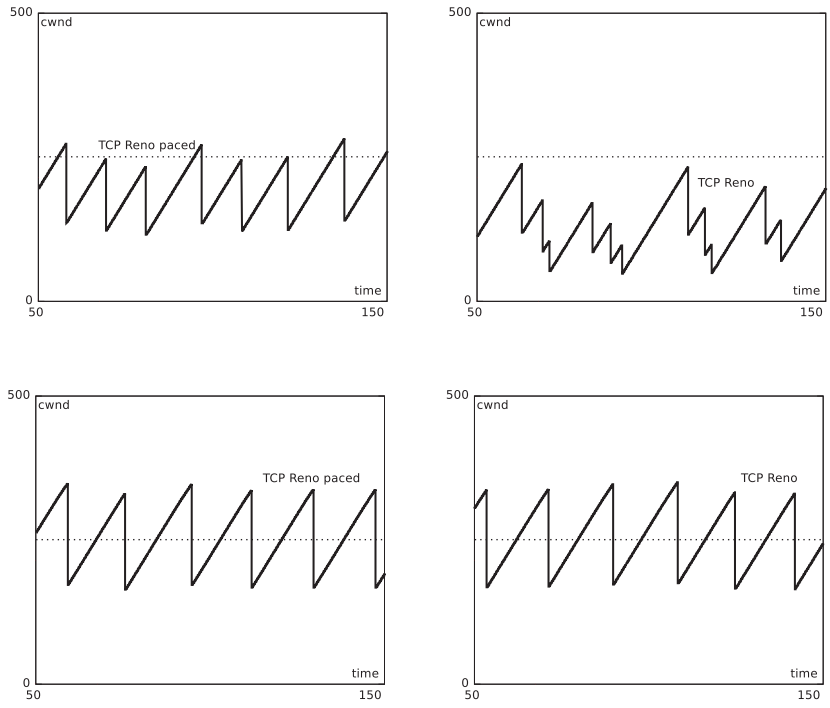


FIGURE 5.20 – Évolution de `cwnd` pour un flot Reno avec *pacing* (à gauche) et un flot Reno sans *pacing* (à droite), pour des tailles de buffers $B = 20$ paquets (en haut) et $B = 100$ paquets (en bas), pour $C = 50$ Mb/s, $\rho_b = 0.5$

L'utilisation de *fair queueing* au sein des routeurs permet également de réguler les arrivées de paquets au sein des flots. Nous en verrons un exemple dans le Chapitre 6, Section 6.5.6. Il en résulte

2. Nous avons utilisé le code TCP ns2 mis à disposition par D.X. Wei : *A TCP pacing implementation for NS2*, disponible à l'adresse `http://www.cs.caltech.edu/~weixl/technical/ns2pacing/index.html`, avec l'option *traditional pacing*.

généralement une probabilité de débordement plus faible, ce qui permet au flot de maintenir une fenêtre de congestion plus importante et ainsi un débit plus élevé.

5.5 Discussion

5.5.1 Enjeux

L'introduction de *fair queueing* présente ainsi des enjeux importants pour un opérateur, afin de pouvoir garantir la performance des applications de ses utilisateurs. Il ne requiert pour cela pas de configuration complexe au sein des routeurs, et permet une évolution saine des protocoles en bordure de réseau.

Son importance nous semble d'autant plus cruciale avec l'arrivée prochaine de réseaux tout-optique, où il n'est techniquement pas possible de réaliser des buffers de plus de quelques dizaines de paquets. Avec la montée des débits, la solution TCP Reno/FIFO/DropTail présente des insuffisances qu'il serait possible de surmonter au moins partiellement grâce à des protocoles plus efficaces, et un meilleur contrôle de l'ordonnancement des flots. Nous avons présenté une méthode mixte mélant un modèle simple à des résultats de simulation permettant d'avoir une estimation qualitative de la performance obtenue par différents protocoles.

Un autre domaine ou l'intégration de *fair queueing* semble décisive est celui des *datacenters*. Ses capacités de différenciation implicite et d'isolation permettront de protéger efficacement les flots court et de leur offrir un temps de réponse satisfaisant, tout en partageant équitablement les ressources pour les autres. Les problèmes d'équité intra-protocolaire sont d'autant plus importants que le besoin de performance motive l'introduction de nouvelles versions de TCP a des degrés de maturité différents (DCTCP, etc.). De plus, l'émergence des *datacenters* partagés, où des machines (virtuelles) sont loués à différents clients, nous pousse à considérer l'interaction de versions TCP hétérogènes comme un cas très probable.

5.5.2 Indicateurs de congestion

La plupart des mécanismes TCP que nous avons considérés utilisent les délais ou les pertes de paquets comme indicateurs de congestion afin d'adapter leur fenêtre de congestion. Avec de petits buffers, les délais d'attente sont faibles et difficiles à estimer, et les pertes de paquets ne sont pas représentatives d'une congestion.

Il convient sans doute de considérer d'autres signaux, prenant en compte non plus les événements, mais plutôt leur distribution ou bien leur moyenne. Par exemple des statistiques sur une certaine échelle de temps du taux de pertes de paquets ou du débit peuvent offrir une bonne indication de l'état du réseau. FQ permet de supprimer l'excédent de trafic par rapport au *fair rate*.

Il serait intéressant de voir dans quelle mesure il est possible de définir un protocole de contrôle plus adapté à de petites tailles de buffers, et permettant un meilleur compromis entre le taux de pertes réalisé et le débit atteint. Idéalement, l'objectif d'un tel algorithme serait d'émettre un paquet au sein de chaque *busy period* de la file d'attente (dont l'évolution est aléatoire), afin d'être servi au débit équitable. Cela nécessite un algorithme suffisamment opportuniste pour profiter de la capacité non occupée par les autres flots et de l'espace disponible dans le buffer, mais sans toutefois risquer de perdre trop de paquets. Ce protocole exploiterait avantageusement le compromis entre son agressivité, et le taux de perte subit.

Un protocole ayant un taux de pertes de paquets cible de 1% par exemple ne nous semble pas irréaliste, afin de pouvoir estimer convenablement cette valeur. Il convient de garder ce taux tout de même relativement restreint afin de ne pas consommer des ressources inutilement sur les liens, même si le débit gaspillé fait partie de l'allocation équitable d'un utilisateur (phénomène dit de *dead packet*). Au choix du protocole, ces pertes pourront être compensées par un mécanisme de retransmission, comme dans TCP, ou par l'utilisation de codes correcteur d'erreur.

5.5.3 Apports du *fair queueing*

La présence de *fair queueing* dans le réseau permettrait l'utilisation de mécanismes de contrôle de congestion alternatifs, comme la proposition *packet pair* de Keshav [86], qui permet de démarrer plus rapidement les connexions (typiquement en mode *slow start*) : le débit équitable est estimé au bout de quelques paquets. Cela peut nous permettre de réduire sensiblement le temps de réponse des flots courts. Une méthode encore plus simple et radicale serait de commencer au débit maximum, pour ensuite réguler le débit au moyen d'un algorithme AIMD/MIMD suffisamment efficace par exemple, ou

tout autre proposition satisfaisante. Les quelques simulations que nous avons effectuées nous laissent présager qu'il s'agit d'une première piste intéressante, en attendant le développement de mécanismes plus sophistiqués.

Il est licite de penser que le traitement équitable apporté par le *fair queuing* entre les flots permettra des estimateurs moins bruités et plus robustes.

Enfin, la comparaison d'une telle approche avec celles plus explicites telles que XCP [81] ou p-Fabric [5] nous semble intéressante à explorer. Il est possible que la complexité que ces solutions ajoutent ne soit pas fondamentalement nécessaire.

5.6 Conclusion

Les insuffisances de TCP Reno pour des liens à fort *Bandwith Delay Product*, et les contraintes fortes qu'il impose sur le dimensionnement des buffers nous conduisent à considérer l'introduction de nouveaux protocoles, plus adaptés à ces nouvelles conditions. Bien que conçus avec l'objectif d'être équitables avec TCP Reno, nous avons mis en évidence des problèmes d'équité inter- et intraprotocolaires, ainsi que les dégradations causées sur les flots *non-bottlenecked* en compétition, incluant des flots *streaming* (pertes, délais, gigue).

L'introduction de *fair queueing*, est proposée comme une alternative plus robustes aux différentes solutions d'AQM, permettant l'isolation des flots. Cette isolation permet d'une part de protéger les flux *streaming* à débit limité, leur assurant de faibles délais (grâce à la différenciation) et un taux de pertes négligeable, et d'autre part d'assurer l'équité entre les flux TCP en compétition pour la bande passante sur le lien. Nous rappelons que dans des conditions réalistes de trafic, le *fair queueing* est à la fois faisable et extensible puisque le nombre de flots à ordonnancer à chaque instant reste de l'ordre de quelques centaines indépendamment de la taille du lien. FQ impose une équité max-min, qui est nécessaire afin de pouvoir offrir des garanties prévisibles et robustes pour le trafic, sans nécessiter la collaboration des différentes sources de données.

Dans un contexte ou une allocation équitable est imposée par le réseau, il n'est plus nécessaire que les algorithmes TCP se restreignent à être équitables avec TCP Reno. Une évaluation de la performance des principales propositions TCP, notamment celles conçues pour des liens à fort *Bandwidth Delay Product* montrent qu'elles sont généralement insuffisantes pour exploiter la bande passante disponible en présence de petits buffers causant des pertes aléatoires (vraisemblable dans un contexte de *datacenters* ou de réseaux optiques, ...).

Grâce à la présence de *fair queueing* dans le réseau, il est possible aux sources de trafic d'utiliser un algorithme très simple suffisamment agressif basé sur AIMD ou MIMD pour obtenir une performance satisfaisante. L'étude suggère qu'un bon compromis entre utilisation et taux de pertes pourrait être atteint par l'utilisation d'un protocole TCP avec un incrément multiplicatif de la fenêtre de congestion comme dans Scalable TCP, utilisant du *pacing*, et un décrément basé sur une estimation de la bande passante disponible comme dans Westwood. La conception d'un algorithme optimal exploitant l'information implicitement transmise par le *fair queueing*, si tant est qu'il est nécessaire, reste un problème ouvert.

Enfin, dans un réseau optique, la réalisation d'un algorithme de *fair queueing* tel que nous l'avons proposé doit être reconsidérée, car il est techniquement difficile de réordonnancer les paquets. Il serait alors intéressant de considérer des solutions alternatives qui rejetteraient les paquets entrants en fonction de l'estimation du débit équitable, à la manière de la proposition *Approximate Fair Dropping* (AFD) [122]. Une piste d'amélioration serait de paramétrer la discipline de service au vu de notre compréhension du trafic, afin d'améliorer les propositions de règles empiriques proposées par les auteurs.