Rapport Simulation NS2

SIMEONI
Lorenzo

Utilisation du programme :

-Se placer d'abord dans le dossier avec le code source

-Taper ns Exo1Q6.tcl pour voir le nam de la question 6 de l'exo 1 (simulation lourde, risque de faire planter nam)

-Taper ns Exo2.tcl pour avoir accès à la simulation nam, et à quelques courbes

-Taper gnuplot plot_congestion_window.p pour créer des courbes supplémentaires sur les fenêtres de congestion dans le fichier congestion, en .pdf

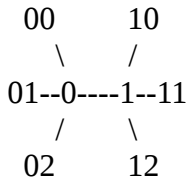
-Taper gnuplot plot_queue.p pour avoir la courbe de la file d'attente entre les 2 points centraux en .pdf

-Taper ns Robuste.tcl pour la simulation «Robuste»

Simulation :

Ce que le programme fait :

Ce programme construit un réseau de topologie :



Il créé ensuite 6 flux TCP : 3 flux de 00 à 10, 01 à 11, 02 à 12 et dans le sens inverse.
Les flux tcp sont notés avec 4 indices, tcp-0-0-1-0 par exemple, ce qui correspond à «un flux TCP allant de 00 à 10. Ceci me permet d'avoir un code générique qui permet de faire marcher le code pour autant de nœuds que voulu, et de toujours savoir qui est l'émetteur et qui est le recepteur.

Il récupère les valeurs de la file d'attente du lien central entre 0 et 1.
Il récupère aussi les valeurs des fenêtres de congestions de tous les flux TCP du graphe.
Il trace les différentes courbes.

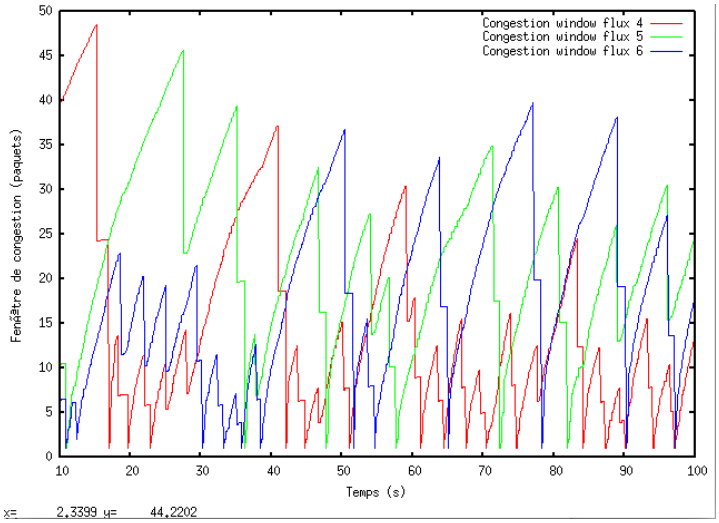
Ce que j'ai étudié :

J'ai décider d'axer ma simulation sur 2 études :
- Tout d'abord, je vais regarder l'influence de la taille de la file d'attente sur les flux TCP
- Ensuite, je regarderai le comportement des flux TCP entre eux, si certains algorithmes se font complètement écraser par d'autres, et donc si il y a des problèmes d'équités, etc.

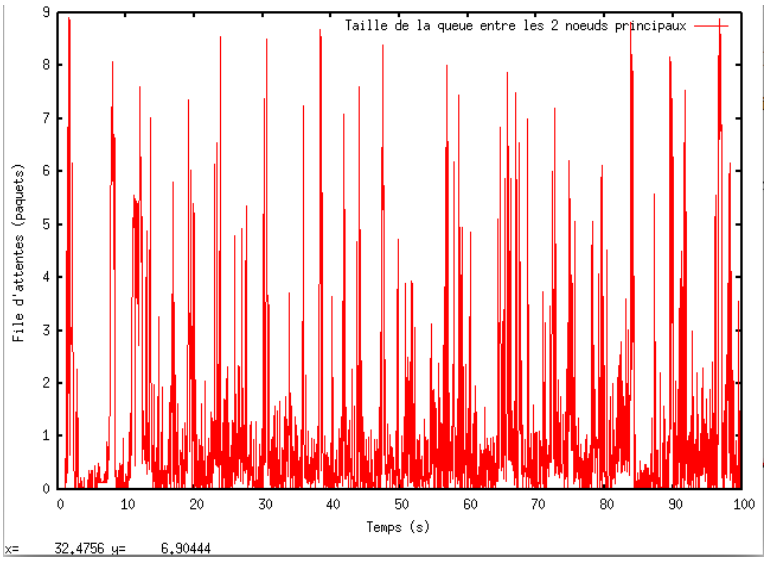
1) Influence de la taille de la file d'attente :

J'ai choisi une taille de paquet usuelle, 1500 octets, et une fenêtre d'émission assez grande (8000), pour que la fenêtre de congestion puisse bien augmenter.

J'ai pris le lien central entre N0 et N1 à 2MB et tous les autres liens à 1 MB, pour que le lien central ai une file d'attente qui se remplie assez vite.
Les graphes présentés seront fait avec l'algorithme Reno, mais la simulation marche aussi avec Tahoe ou NewReno.
Tout d'abord j'ai choisi une file d'attente assez petite, pouvant contenir 10 messages.



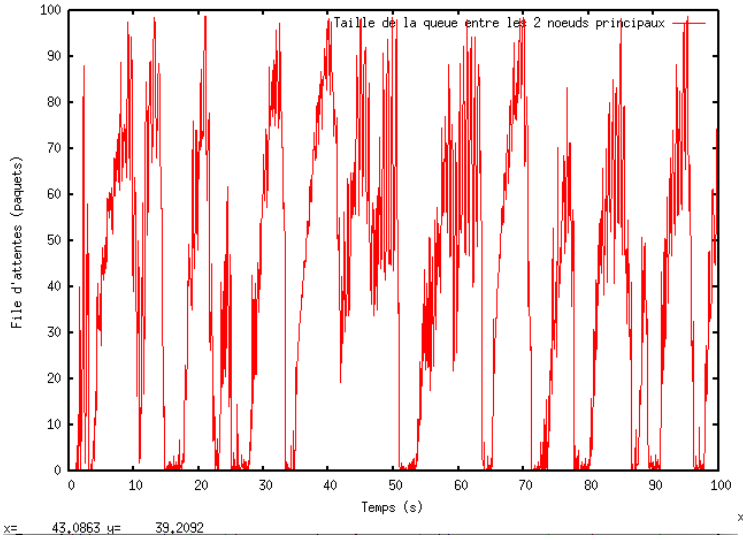
Fenêtres de congestion de la grappe 1 à 0



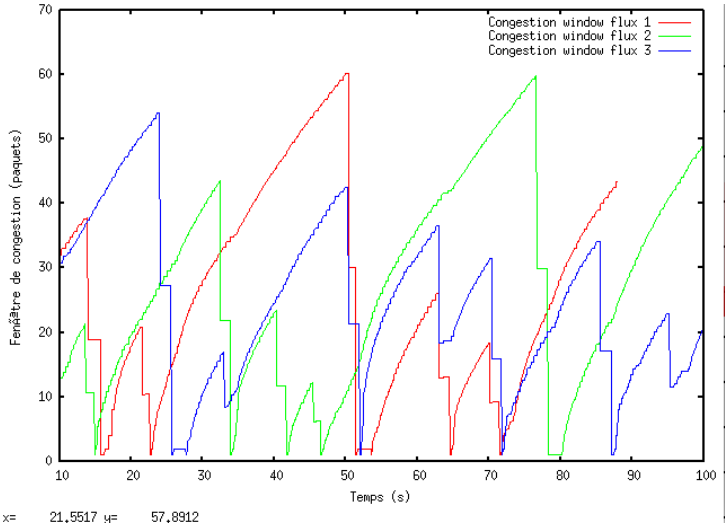
File d'attente entre 0 et 1

On voit qu'avec une petite file d'attente, Reno fonctionne normalement, il commence en slow start, et quand il y a congestion, Reno n'attends pas son timeout pour redémarrer. En effet on voit bien sur les courbe que Reno utilise le Fast Recovery, en divisant sa fenêtre de congestion par deux et en le Fast Recovery. On observe d'ailleurs que plusieurs FastRecovery ne se terminent pas forcément par un timeout, on a donc un algorithmes Reno qui fait plutôt bien son travail étant donné qu'il arrive à faire ses FastRecovery sans avoir forcément un timeout et retomber en slow-start, auquel cas Reno est encore plus lent que Tahoe qui lui retourne directement en slow-start. Du côté de la file d'attente, on voit qu'elle est souvent pleine ou presque, et qu'il y a donc potentiellement un grand nombre de congestion.
On peut aussi remarquer que TCP/Reno est bien équitable avec d'autres Reno, en effet on voit que le flux rouge occupe d'abord beaucoup de bande passante, mais c'est ensuite le flux vert qui domine, puis le bleu. Avec un lien central de plus grande capacité, on voit bien les courbes se coller au début.

On met ensuite la file d'attente à 100, en théorie, le flux va atteindre une fenêtre de congestion plus grande car ça sera plus long pour avoir une congestion, on pourrait donc penser dans un premier temps qu'allonger les files d'attentes est bénéfique pour les flux et pour le Réseau. En effet même si il y a congestion, en Reno, le routeur aura un cwnd/2 plus grand pour une file d'attente de 100 que pour une file d'attente de 10, et pourra donc avoir un FastRecovery plus efficace. Seulement on va voir que allonger les files d'attentes pose un problème.



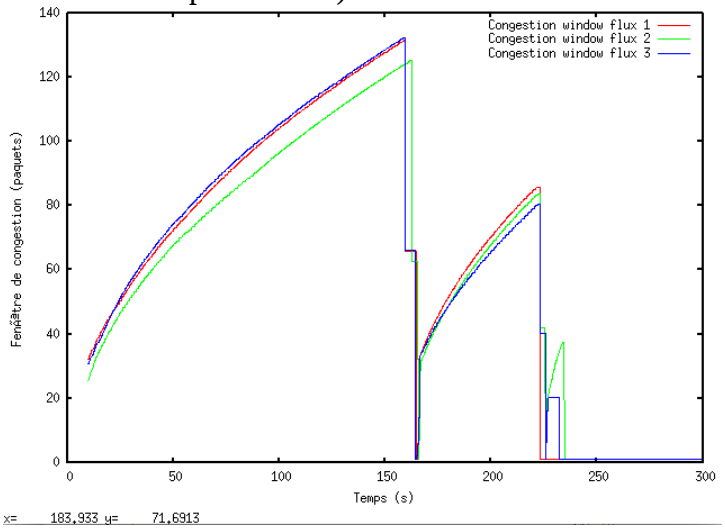
File d'attente de 100



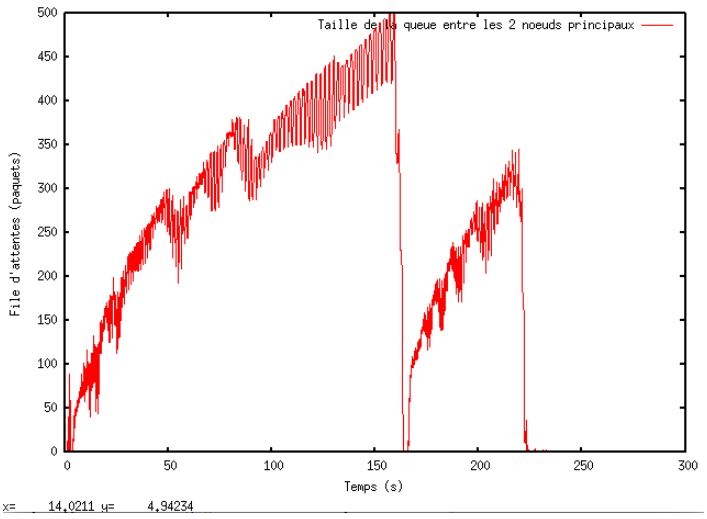
Fenêtres de congestion de la grappe 0 à 1

On voit que comme on s'y attendait, la fenêtre de congestion est un petit peu plus grande que pour 10, seulement on peut aussi remarquer que de moins en moins FastRecovery marchent. On a donc plus de slow start et Reno perd donc plus de temps que si il arrivait à faire ses fastRecovery.

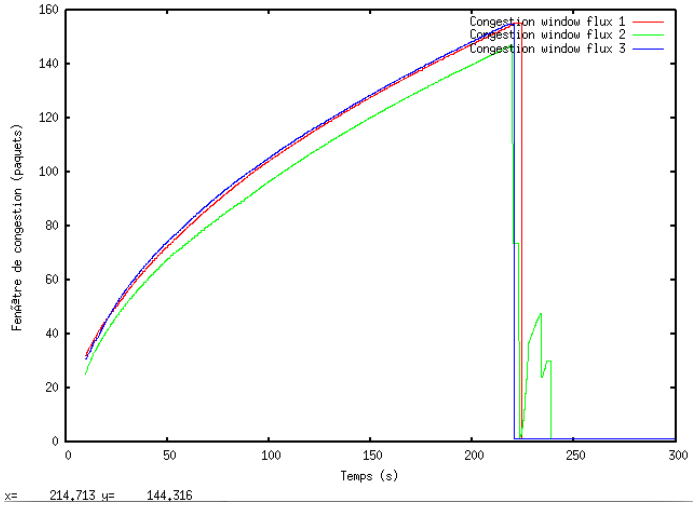
Pour des files d'attentes de 500 puis de 1000 (J'ai multiplié le temps de simulation par 3 pour bien voir le phénomène):



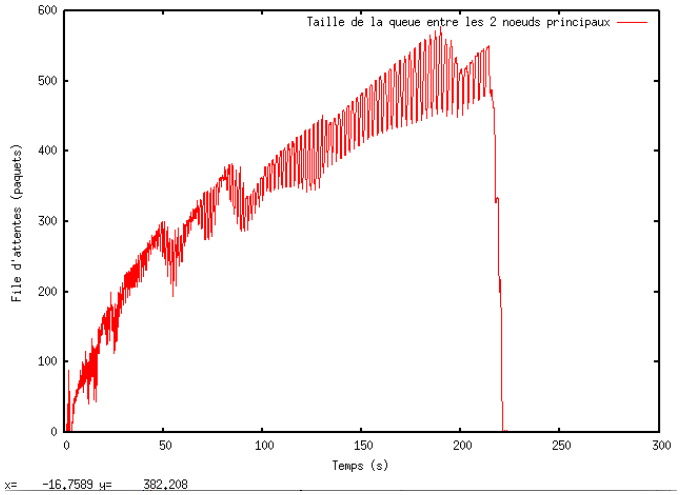
Fenêtre de congestion de la grappe 0 à 1



File d'attente de 500



Fenêtre de congestion de la grappe 0 à 1

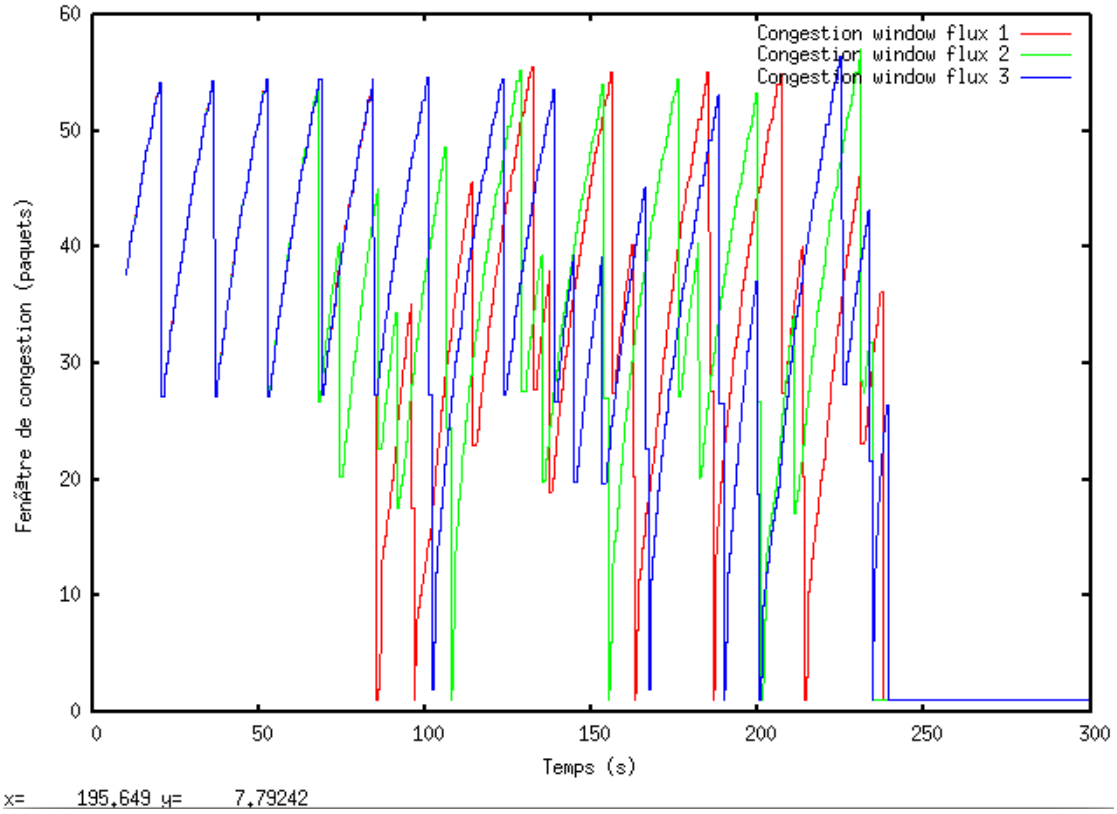


File d'attente de 1000

Pour les deux cas ci dessus, on voit que Reno a une fenêtre de congestion bien plus grande, seulement cela pose problème car il n'y a plus de FastRecovery possible, car les paquets dans la fin de la file mettent tellement de temps à passer qu'on dépasse le timeout, et que Reno retombe en Slow Start. Dans le cas avec une file d'attente à 1000 on a carrément directement un timeout pour le flux bleu et rouge, car les paquets mettent trop de temps à passer. Ainsi on voit bien que les algorithmes TCP, ne se comportent plus correctement quand la file d'attente augmente, et que même si ça semble être une bonne idée pour augmenter la fenêtre de congestion au départ, c'est en fait anti productif.

2) Comparaison entre algorithmes :

J'ai choisi un lien centrale de 5 MB et les autres de 1MB et une file d'attente de 50, pour ne pas qu'il y ai trop de congestion et qu'on voit bien l'équité entre les flux.
J'ai choisi de mettre tous les liens en NewReno tout d'abord. Car c'est un algorithmes assez agressif car souvent en slow-start.

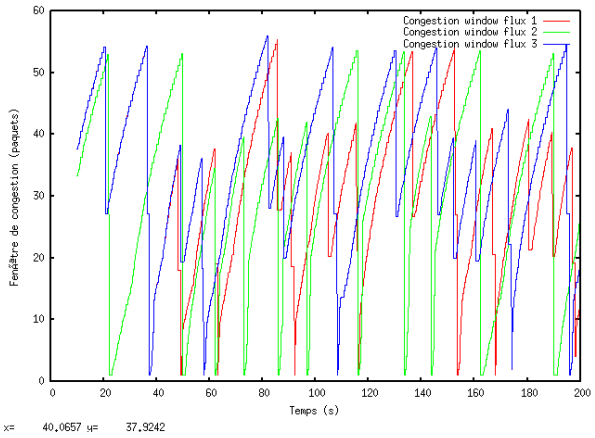


Graphe des Flux de la grappe 0 vers la 1

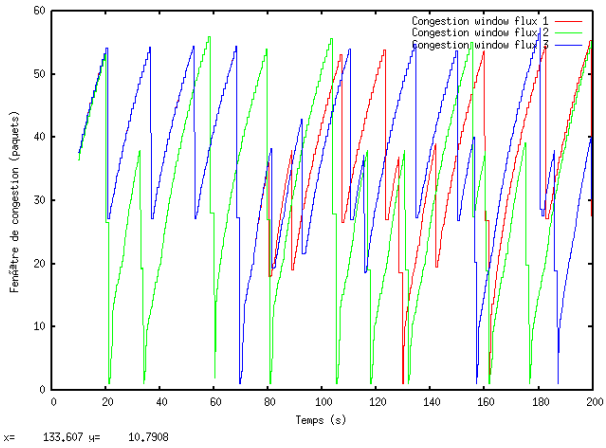
On voit bien l'équité entre les flux, au début car les signaux sont collés et utilisent autant de bande passante.
On voit ensuite qu'il y a plus de congestions, timeout etc, donc les signaux ne restent pas collés, mais alternent successivement à la place de celui qui a la plus grande bande passante, un flux ne domine pas tous les autres.

Quand on change un des flux pour passer en Reno ou en Tahoe, on observe que les 2 algorithmes sont moins efficaces car reviennent plus souvent en slow start, mais que ils partagent bien la bandes passante avec NewReno, car on a un flux qui est dominant à tour de rôle.

Dans les graphes suivant, le flux vert sera celui qui sera changé par rapport aux autres.

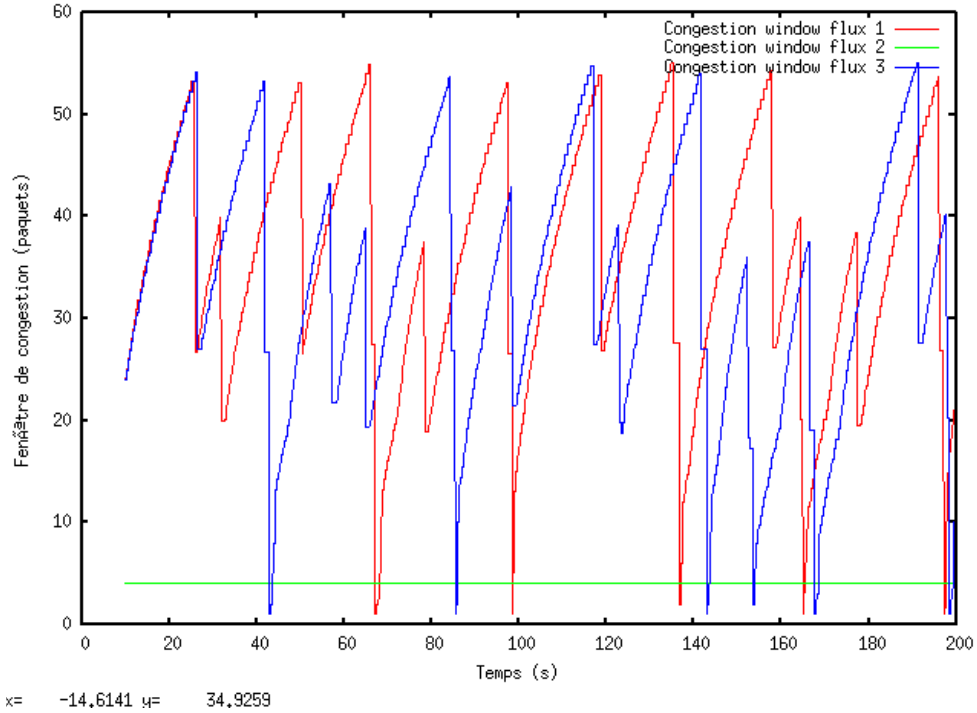


Congestion avec Tahoe(flux vert)



Congestion avec Reno(flux vert)

Avec Vegas, on a un cas plus intéressant :



Dans le cas avec Vegas on voit que le flux TCP vert s'écrase complètement devant les autres qui monopolisent la bande passante. Les deux autres n'ont même pas de fenêtre de congestion plus grande que dans le cas précédent avec 3 NewReno, ils ne profitent donc même pas du fait que le flux vert s'écrase devant eux. On a donc un gros problème d'équité et d'efficacité quand on utilise un Vegas avec d'autres algorithmes.

Congestion avec Vegas (flux vert)

Dans l'ensemble les algorithmes TCP sont quand même assez équitables.

Pour rendre le réseau plus robuste j'ai rajouter 3 liens entre entre des routeurs de grappes différentes, on pouvait aussi rajouter un routeur et des liens…
La perte d'un lien ou routeur va entraîner des messages pour pouvoir mettre à jour la table de routage. La perte d'un lien est souvent peu grave car on s'assure souvent d'avoir un réseau plus ou moins complexe, mais la perte d'un routeur peut priver plusieurs utilisateurs d'avoir accès à internet si leur switch était directement relié à ce routeur.