Rapport Simulation NS2 :
Réseau Géant

Utilisation :

- Se placer dans le dossier avec le code source
- Générer le code tcl avec la commande python Geant.py
- Exécuter le code tcl avec ns Partie2Projet.tcl
- Pour avoir les statistiques des 3 pires nœuds : python statistiques.py

Simulation :

Ce que le programme fait :

Ce programme créé d'abord la topologie donnée dans topo.top. Les liens entre les nœuds sont en Gb pour ressembler à ce qui existe vraiment(et ils sont en duplex, pou pouvoir communiquer de ville en ville sans allonger le RTT) . Chaque nœuds a une file d'attente de 20, car l'ordre de grandeur en réalité est aussi de quelques dizaines. Par la suite on pourra observer le comportements des fenêtres de congestions pour des files d'attentes plus ou moins grandes. La queue est géré soit par DropTail, soit par RED, on étudieras aussi par la suite les différences observées avec ces deux types de protocols. Sur chaque liens, est attaché un monitor queue, permettant d'avoir accès aux valeurs de pertes de paquets ainsi que du taux de pertes, données grâce au fichier statistiques.py. Cette première partie de code permet aussi de générer(si l'on veut) le code présent dans statistiques.py. En effet, il fallait ouvrir énormément de fichiers, et je ne pouvais pas faire de boucle, allant de 0 à 25 car par exemple queue-0-6 n'existe pas. Je me suis donc servi de Geant.py pour ne pas à avoir à tout taper à la main.

La deuxième partie du code, la plus importante, met d'abord en place un trafic de fond ON/OFF, basé sur une loi de Pareto, pour créer un flux constant de données. Il correspond à 80 % du flux présent dans traff.traf. Ce trafic ON/OFF est en fait un flux UDP, que l'on ne va pas devoir subdiviser en plusieurs flux, comme on va le faire après pour les flux TCP, ce qui augmente grandement le temps de simulation. Pour ce trafic ON/OFF j'ai choisi un MTU basique qui fait 1500 octets. De plus, vu qu'on se trouve dans ce qu'on peut considérer être un LFN, j'ai choisi un temps pour les pèriodes ON/OFF assez élevé, 500ms, pour simuler le fait qu'il y a un grand RTT dans ce type de réseau.
Ensuite on prend 20% de ce qu'on nous donne dans traff.traf et on le transforme en flux TCP. Pour ça, on utilise une loi Zipf, avec un shape de 1.2. J'ai fais au préalable quelques test dans un fichier test.py, pour voir quelle valeur correspond le mieux au 90 % souris, 10 % d'éléphant et c'est la valeur 1.2 qui me semblait la meilleure. J'avais d'abord testé avec une loi uniforme, mais le trafic généré était surtout du gros trafic et ne correspondait donc pas à la réalité, je suis donc passé par la suite à une simulation avec une loi Zipf. Pour ne pas que le tirage soit plus grand que le chiffre donné dans traff.traf, j'ai rajouté une boucle while, qui permet de relancer ce tirage. Tous les petits volumes de data, sont ensuite par passé en Go (* 1000000000) , ce qui fait qu'il y a au minimum des flux de 1Go, ce qui est assez conséquent , mais qui permet grandement de réduire la simulation, et de pouvoir tracer quelque chose d'assez uniforme, en effet, avec des flux de quelques octets, on ne verrait les flux qu'un centième de seconde.
Pour une soucis de réalisme, les trafics ne sont pas tous lancés en même temps. J'ai tout d'abord voulu utiliser une loi exponentielle, qui permet de mettre en place des périodes de burst et de warm

up, pour observer des congestions. Mais il m'a paru ensuite plus réaliste que les départ suivent une loi uniforme, car en moyenne sur 5min il y a autant d'utilisateur au début qu'à la fin. De plus les périodes de burst et warm up sont quand même observables sans loi exponentielle, vu que les trafic sont plus ou moins grands. J'ai donc opté au final pour une loi uniforme.
Enfin, cette partie de code fait appelle à la fonction récursive plotWindow, qui enregistre à un instant t, la taille de la fenêtre de congestion d'un flux. Etant donné qu'il y a beaucoup de petits flux créé à partir du volume de base, et aussi beaucoup de flux entre chaques nœuds, et que je voulais tracer tous les flux d'un nœud sur un seul graphe, j'ai suivi la fenêtre de congestion du premier petit volume de flux à chaque fois.

Ce que j'ai regardé :

J'ai tout d'abord regardé les pertes présentes dans mon réseau, pour voir si il y en avait, et donc que des congestions étaient présentes, ou si il fallait que je génère des phases de congestions. Ayant assez de pertes, j'ai donc regardé les trois pires liens.

Tout d'abord pour le temps T = 5 min :
J'ai donc trouvé que les liens entre 12-13, 11-12 et 0-2 étaient souvent les pires en thermes de pertes. En effet on trouvait en général(pas moyenne car je n'ai pas assez de valeurs, mais on retrouvait souvent ces ordres de grandeur) une quantité de pertes de valeur : 1632616795 pour le premier, 1302857562 pour le deuxième et 1218642632 pour le troisième. (en paquet)

On a aussi sur ces trois liens, un volume d'arrivée de 124887915115, 124213736918 et 321603658848 paquets.
Soit en octet, pour le premier, 187331872700000 octets, ce qui est dans l'ordre de grandeur des 20 % de data donné dans le fichier traff.traf.

On arrive donc à des valeurs de taux pertes de 1,3% pour le premier lien, de 1 % pour le deuxième et de 0,38 % pour le troisième. On constate qu'il y a en moyenne pas mal de pertes sur ce réseau car les trois pires liens ont un pourcentages de pertes très grand. En effet avoir 1 pertes sur 100 est beaucoup trop !!

Sur une échelle de temps de 100sec :

On retrouve aussi en général le lien 12-13, 11-12, et 0-17.
avec par exemple des valeurs de pertes de 141705772, 140165784 et 138657071.
Et des volume d'arrivé de 10531627222 11788331790 et 14345197737.
On voit qu'en général les volume d'arrivée dont bien moins grand que sur 3 min, c'est du au fait que les flux n'ont pas pu finir la fin de leur envoi de donné, avant la fin du programme.

Quoiqu'il en soit on retrouve des taux de pertes de 1,3 %, 1,2 % et 0,96 %. On retrouve des taux de pertes assez proches que ceux d'avant, mais quand même plus grand. On peut donc se demander pourquoi, en effet, si l'on regarde le volume de donné arrivé, pour 100sec on voit qu'il est bien inférieur à celui qui est envoyé pour 300sec /3. Le volume de data en général n'est donc pas le problème, c'est plutôt le temps auxquels sont envoyés les « gros » flux. En effet vu que les flux commencent à un temps entre 0 et 100 avec autant de chance, si en même temps plusieurs gros flux commencent, on va avoir congestion, et donc plus de pertes.

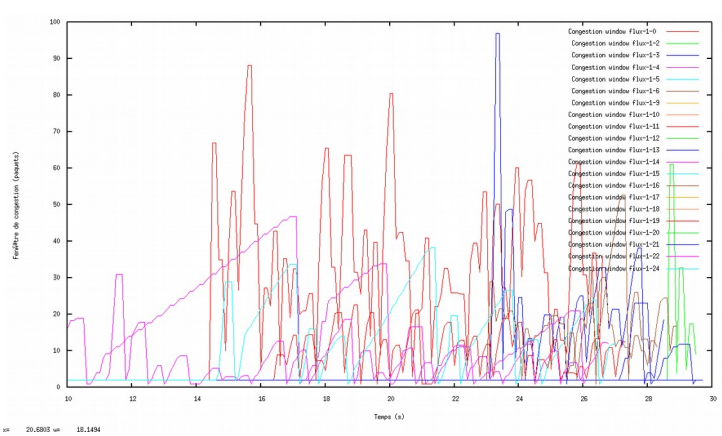
Pour 30sec :
J'ai trouvé ici le lien entre 12-13, 0-17 et 0-2
Avec les valeurs de pertes : 45338070, 44918805 et 43445085 .
Les volumes d'arrivée : 3405498604, 2868171649 et 8338586369.

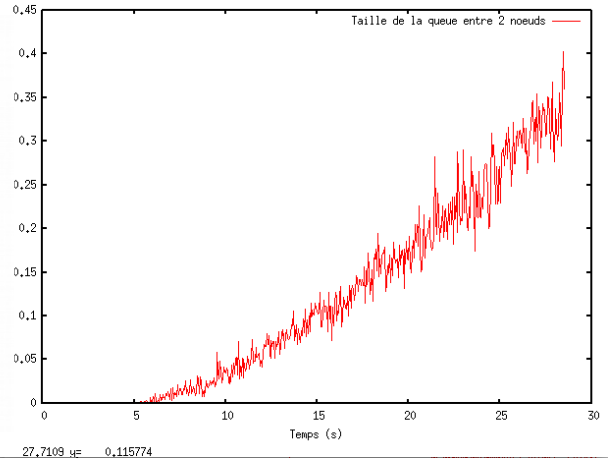
On a donc un taux de perte de 1,3 % , 1,56 % et 0,5 %
Les taux restent similaires..

Le réseau étant bien assez mis à l'épreuve vu le nombre de pertes, je n'ai pas augmenté les capacités des liens etc..

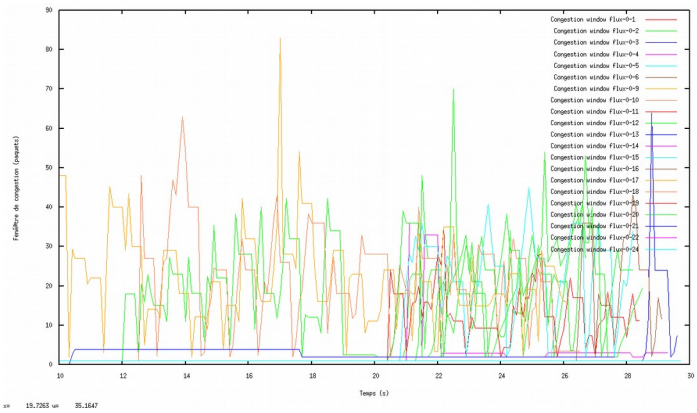
Enfin, j'ai étudié le comportement du réseau en changeant le type de queue, le temps de simulation, et le type de protocol TCP utilisé. J'ai aussi changé par la suite la grandeur des files d'attentes.

Tout d'abord j'ai commencé avec un temps de 30sec, une file d'attente de 20 avec Droptail, et un Linux en protocol de transmission.



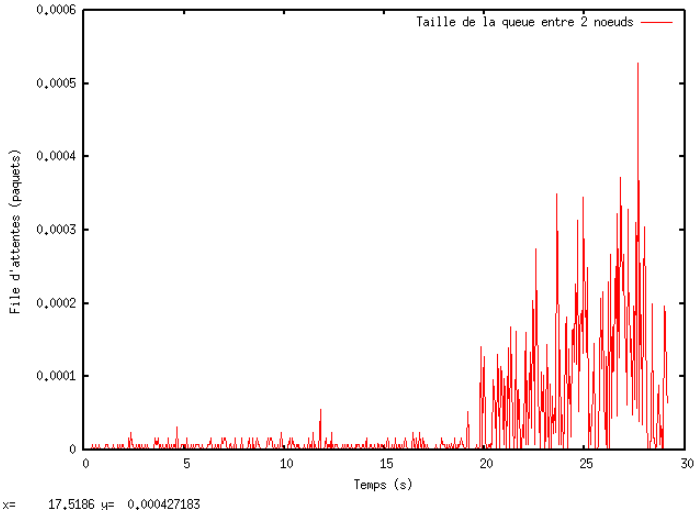


Je pensais que Linux était le protocole TCP utilisé par Linux, donc CUBIC, mais on voit que le protocole arrive difficilement à garder un flux avec une haute fenêtre de congestion, et ressemble un peu à du RENO, j'ai donc par la suite changé pour RENO et j'ai retrouvé les même valeurs de fenêtre de congestion. On peut voir qu'il y a beaucoup de pertes, contrairement à ce qu'on a pu trouver dans la partie 2. En effet les 1 % et 0,3 % nous semblait peu, mais ici les flux n'arrivent pas vraiment à garder un état stationnaire, ou a augmenter leur fenêtre de connexion. Par la suite j'ai essayé avec d'autre protocoles, clairement différents, comme Vegas, qui fonctionne très différemment de Reno.



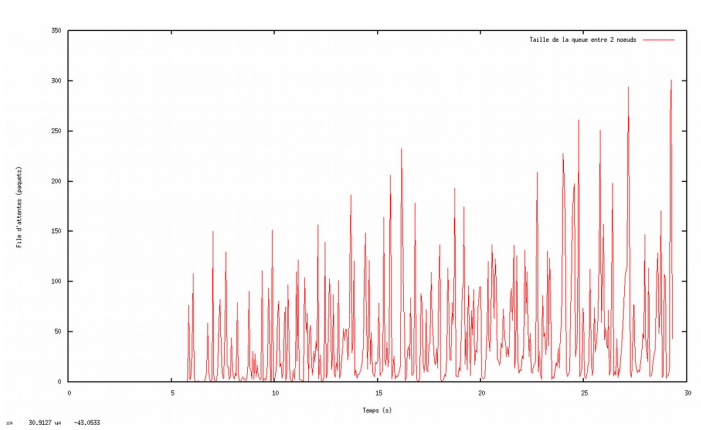
Avec Vegas on voit (pas très bien), que pareil, nous n'arrivons ni à garder un état stationnaire, ni à avoir une grande fenêtre de congestion. De plus les courbes ne ressemblent même pas à du Vegas, car je pense que dès la sortie du slow-start, on a directement une perte/timeout, et donc un retour au slow-start. c'est assez étrange, mais je pense que les protocoles sont juste pas fait du tout pour un LFN comme Geant qui a un très grand RTT.

J'ai ensuite changé de type de file, et j'ai mis RED à la place de DropTail :

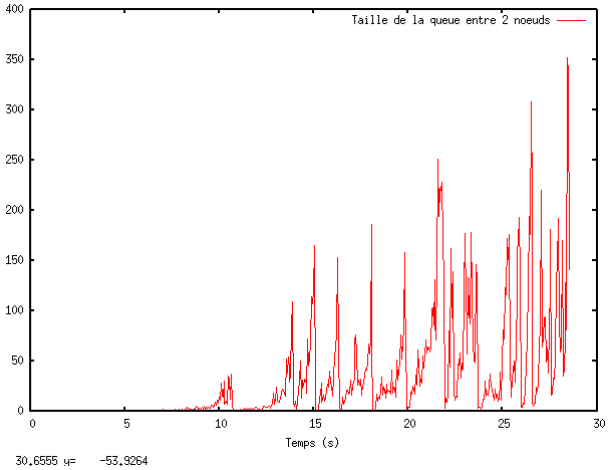


On voit qu'avec RED, la file d'attente se rempli bien moins vite, en effet, RED permet de supprimer des paquets même si la file n'est pas pleine. On a même l'impression que RED arrive à garder la file d'attente dans un état qui reviens à 0, puisque à 28sec on voit que la file d'attente se vide presque complètement.

Pour voir si Red permettait de garder des files plus ou moins vide au cours du temps avec une une grande file d'attente, j'ai mis la grandeur des files d'attentes à 2000 :

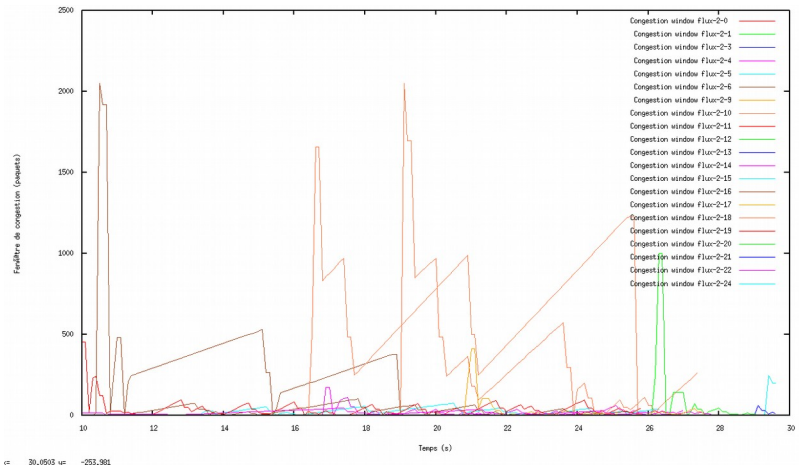


queue avec RED



queue avec DropTail

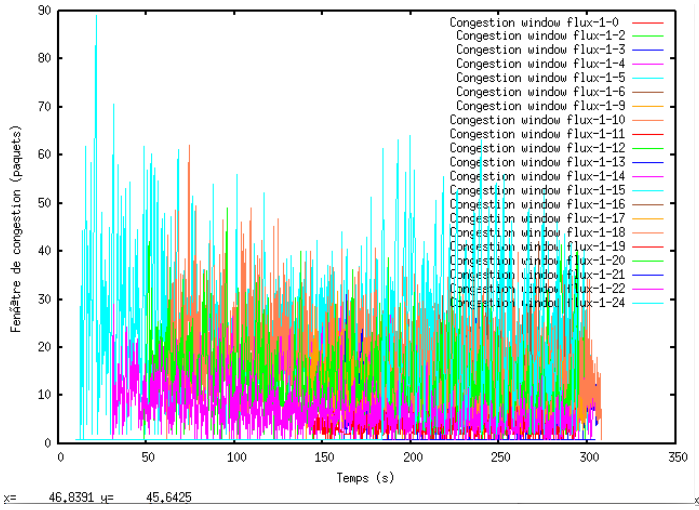
On voit ici que la queue se rempli bien plus que précédemment mais on peut remarquer qu'elle retombe souvent à 0 avec RED. Avec DropTail, la file d'attente se rempli plus qu'avec RED, et la file d'attente semble plus souvent pleine, ou en tout cas un peu rempli. Cela peut causer des timeout, vu que les paquets vont rester plus longtemps dans les files d'attentes.

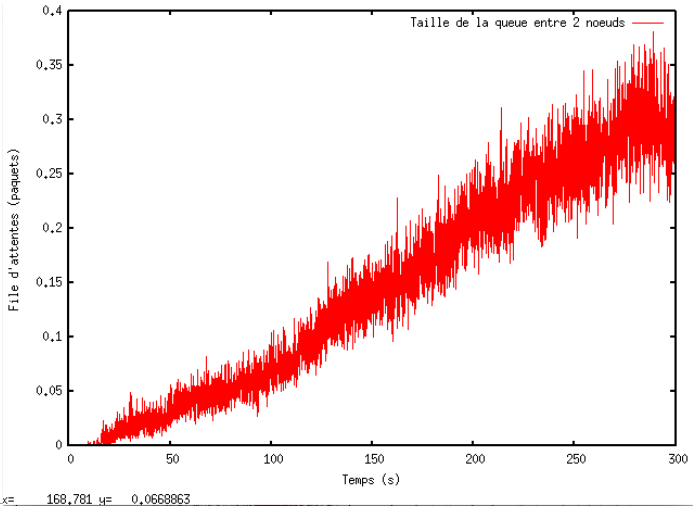


Avec ces queues et Reno on obtient ces courbes de fenêtre de congestion. On voit que les valeur de fenêtres de congestion montent bien (bien) plus haut, avec des valeurs à presque 2000paquets, et on peut encore observer les fast recovery. On n'arrive donc pas forcément au timeout, vu que les files d'attentes arrivent à se désemplir. Cela semble bien meilleur qu'avec la file d'attente de 20. Seulement on fait cette simulation sur seulement 30 sec.

J'ai donc ensuite regardé sur 300 sec les résultats avec une file d'attente de 20 puis de 2000 :

Dans les graphes suivant, on utilise RED et Reno

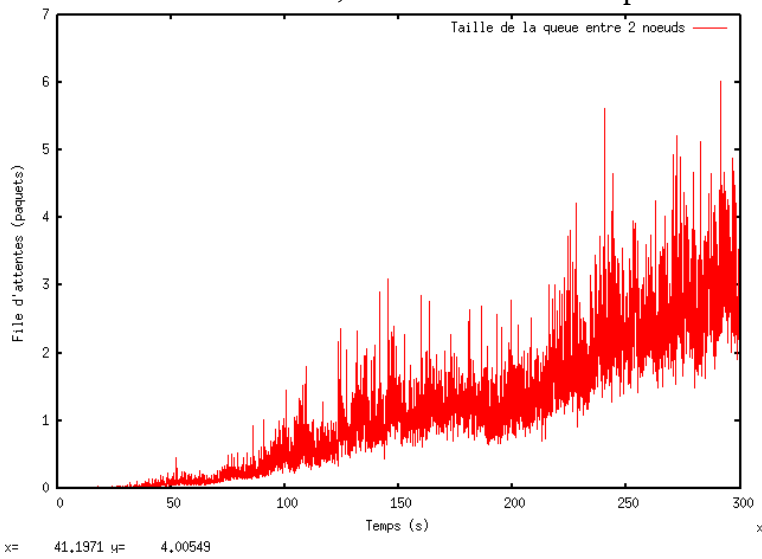


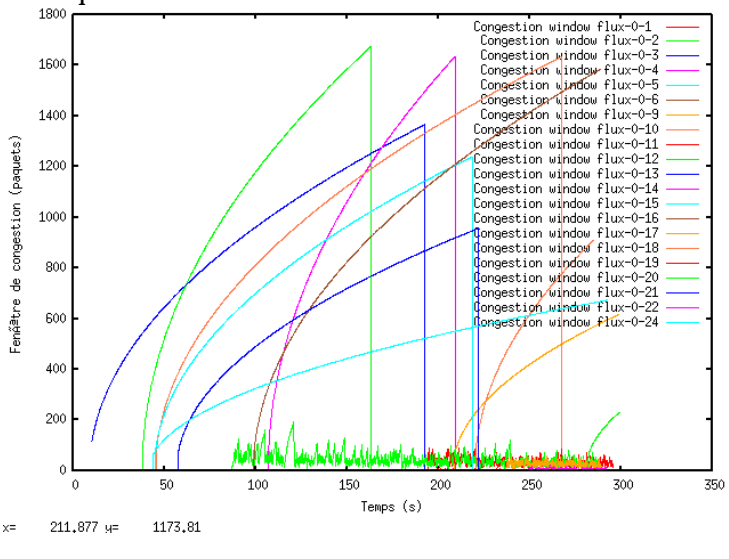


Avec une file d'attente de 20 :
Tout ce qu'on peut dire de ce graphique est que la taille de la fenêtre de congestion varie très vite, c'est pourquoi on ne peut distinguer que des « pics ». De plus la fenêtre de congestion ne grandi pas beaucoup, alors que nous somme sur un LFN et que la bande passante disponible est très important. On a donc un protocole qui n'arrive pas à pleinement utiliser la bande passante du réseau sur lequel elle est. On s'y attendait car Reno n'est pas efficace en période de Congestion avoidance pour des réseaux à grande BP. Seulement il semble y avoir beaucoup trop de pertes.

A droite, on voit la File d'attente, géré par RED, qui se remplie petit à petit. On pensait que RED pouvait garder la file d'attente stable sur 30 sec, mais on voit la sur 300 sec, qu'on ne peut pas.

Pour bien voir l'influence des la taille de la file d'attente, j'ai après changé la taille des files d'attentes à 2000, en laissant les autres paramètres tels quels.





On voit que les files d'attentes se remplissent petit à petits, bien que RED permettent de prévenir un peu ce remplissage. De l'autre côté on voit que certains flux, on pu atteindre une fenêtre de congestion très grande grâce à cette file d'attente démesurée, mais on ensuite reçu un timeout(on le sait car il n'y a pas la phase de Fast retransmit). On pourrait penser que ce n'est pas grave car on atteint quand même une haute Bande passante, et donc que c'est déjà ça, seulement le fait de repartir en slow start est très mauvais pour un réseau avec une très haute BP, et quand les files

d’attentes seront remplies(si la simulation dure plus longtemps), on va avoir constamment des timeout, et donc constamment des slow start. Ce qu’on peut dire sur RED, c’est qu’il ne permet pas, comme on le pensait sur 30 sec, de garder la file d’attente vide à certains moment. Elle se remplie, comme pour DropTail, mais plus lentement.
Avec cette étude je ne pourrai pas affirmer si les files d’attentes grandes ou les files d’attentes petites sont plus efficaces. En effet avec des petites files, on peut avoir du fast recovery, seulement la fenêtre de congestion ne décole pas du tout et reste en dessous de 100.
Avec une grande file d’attente, on n’a plus accès au Fast Recovery, mais on a des Fentres de congest 10 fois plus grandes, ce qui n’est pas négligeable.
D’autre part RED ne sert pas à grand-chose au final, il ne fait que retarder le remplissage des file d’attente, on peut faire une analogie avec Vegas et Reno, ou Vegas ne fait que retarder les pertes qu’à Reno. Je pense donc qu’utiliser la plupart du temps DropTail, est bien plus simple, et pas si mauvais par rapport à RED.