Network Simulator 2 - Topologie et trafic réaliste

Abdellah Elazzam

Decembre 2017

1 Approche du sujet et choix des paramètres :

1.1 Approche du sujet :

Le sujet consiste à générer un trafic pseudo-réaliste le plus possible qui simule un réseau réel. Il s'agit d'un réseau à très haut débit destiné à la recherche et à l'éducation en Europe nommé GÉANT. Pour cela, on se base sur deux fichier, le premier est *topo.top* qui donne la topologie du réseau concernant tous les fournisseurs de la recherche européenne. Le deuxième est $traff.traf$ qui donne le vrai traffic pris par GÉANT par tranche de quart d'heure. Le but du jeu est de stresser GÉANT surdimensionnée pour qu'il fonctionne bien et de le rendre réaliste le plus possible en se basant sur des outils statistiques.

1.2 Programme :

1.2.1 Utilisation :

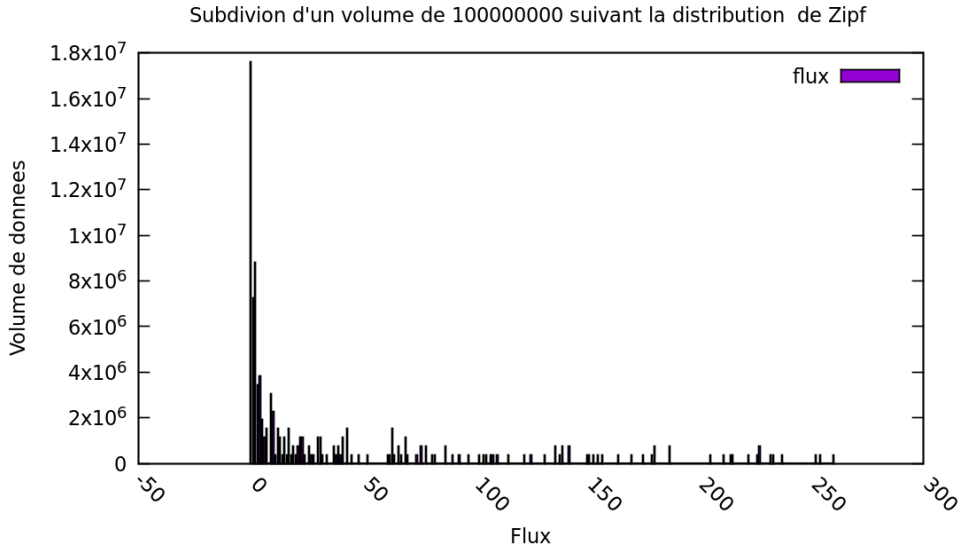
- Se placer dans le dossier contenant le code source
- Lancer le script de simulation : *python generate_GEANT.py*
- Exécuter la simulation : *ns GEANT.tcl*
- Générer la trace des pertes et débit utile : *awk -f awk.awk < traceall.tr*

1.3 Choix des paramètres :

Le programme écrit en python, introduit une fonction pour créer la topologie en parsant le fichier Topp.top. J'ai considéré les liens entre les noeuds en Mégabits pour respecter le rapport entre cette capacité et les quantités injectées qu'on verra après et qui sont moyennement faibles. Ces liens sont aussi duplex pour pouvoir transporter l'information dans les deux sens entre les fournisseurs. Sur chaque lien, Chaque lien à sa propre file d'attente, la taille limite des files est 100. On attache à chaque lien un agent *Monitoring Queue* qui permet d'avoir les statistiques concernant les pertes des paquets de ce lien. J'ai utilisé aussi la méthode de $trace - all$ fournie par NS2 qui donne l'échéancier contenant tous les événements qui se produisent dans le réseau. Ainsi, en parsant ce échéancier grâce au script écrit en AWK, on déduit les pertes et le débit utile voire la charge du réseau. La deuxième fonction du programme concerne le trafic généré par le réseau. Afin de préserver un peu de réalisme, on met en place le modèle ON/OFF pour générer un trafic correspondant la distribution de Pareto. Ce trafic est en faite un flux UDP. En effet, on associe un agent *par-src-des* à chaque *udp-src-des* et ceci correspond à 80% du quantité mise dans le fichier $traff.traf$. Les paquets sont donc envoyés à un débit fixe pendant les périodes ON de durée égale à : $500ms$, et aucun paquet n'est envoyé pendant les périodes

OFF de même durée.On prend un durée moyennement grande puisqu'il s'agit d'un réseau LFN où les temps RTT sont grandes. Les périodes d'activation et de désactivation sont prises à partir d'une distribution Pareto avec des paquets de taille constante égale à $MTU = 1500$ *octets* qui correspond à la fameuse taille pouvant être transmis en une seule fois (sans fragmentation) sur plusieurs interfaces.

A priori, le réseau GEANT est sur-dimensionné par rapport au trafic à injecter par le modèle ON/OFF. On le stresse d'une façon intensifié grâce au volume TCP qui correspond à 20% restante du quantité totale du trafic. Ce volume est ensuite subdivisé en sous flux TCP. On opte pour une loi de puissance qui fait de sorte qu'il y'a beacoup de flux et très peu de grand flux. Ainsi, la loi de Zipf était la bonne solution. En effet, plutôt Zipf que pareto, puisqu'il s'agit d'un univers discret, et donc on assure le tirage d'une valeur entière. On choisit comme nombre maximale de flux : $300 flux$ puisque on multiplie les quantités par 1000 ce qui fait peu de données (En réalité il y'a des millions voire des milliards flux qui circulent). Et pour faire qu'il y'a 20% des grands flux contient 80% du trafic vue sur internet, on choisit un rayon de courbure très faible de 1.01. De plus la taille du plus petit flux est égale à 10 paquets ce qui fait 15000. La figure suivante montre la subdivision d'une quantité de 100000000(peu importe l'unité) :



On voit bien qu'on a subdivisé notre volume en 260 flux,et qu'il s'agit bien d'une distribution de Zipf respctant la règle de 80%-20% avec 15000 comme taille du plus petit flux. J'ai remarqué qu'on finit par tirer des valeurs qui permet de dépasser le volume totale. Comme solution, on tranche cette marge de dépassement du premier flux qui contient beaucoup de données comme le montre la figure. Ainsi on ne déforme pas notre trafic. On fait cette procédure, c'est à dire casser chaque grand volume C en petits volumes c ($C = (c1, c2....., cn)$, ensuite, on envoie chaque petit volume entre les fournisseurs suivant la topologie constituant donc plusieurs sous-flux TCP, au lieu de considérer chaque quantité un seul flux ce qui est pas du tout réaliste. On note aussi que les dates de départ sont temporellement équiparties sur l'intervalle [50 :250]. Ainsi on tire les instants de départs uniformément sur cette intervalle.

2 Étude approfondie des pertes de paquets :

J'ai commencé à regardé les pertes générées dans mon réseaux en calculant le taux de pertes correspondant à chaque lien. Le taux de perte local d'un lien correspond au ratio des paquets perdues par rapport au paquets totales traités par ce lien.

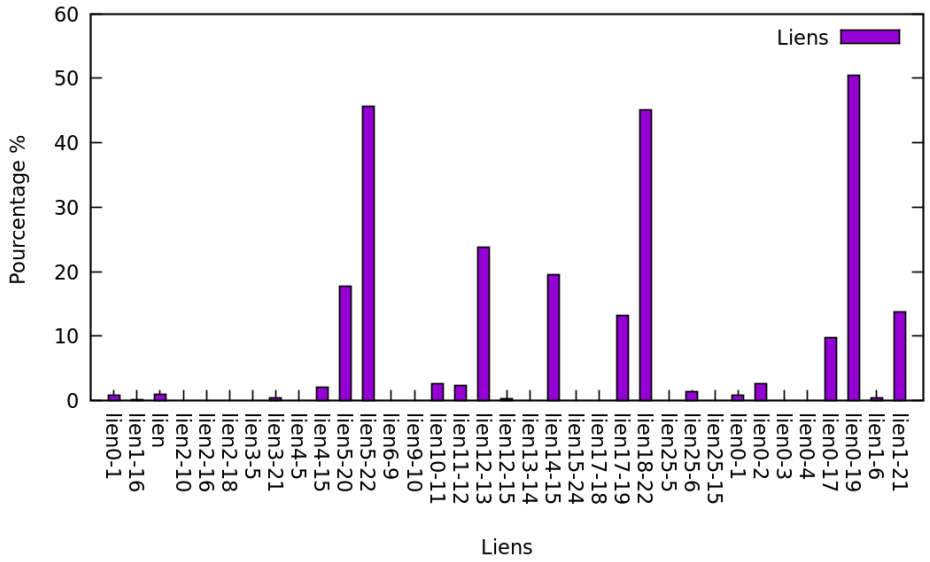


FIGURE 1 – Le taux de pertes pour tous les liens de la topologie GEANT

On remarque bien que les trois liens qui représentent le plus de pertes sont : *lien5-22 , lien18-22 , et lien0-19* avec un taux de pertes à peu près le même et proche de 50%. Afin de comprendre la cause de pertes, on analyse l'"évolution des ces pertes au cours du temps. On prend deux liens pour ceci, un avec un grand taux de pertes, par exemple *lien5-22* et un avec faible taux de pertes *lien0-1* par exemple tout en considérant la taille des files d'attente pour ces deux liens.

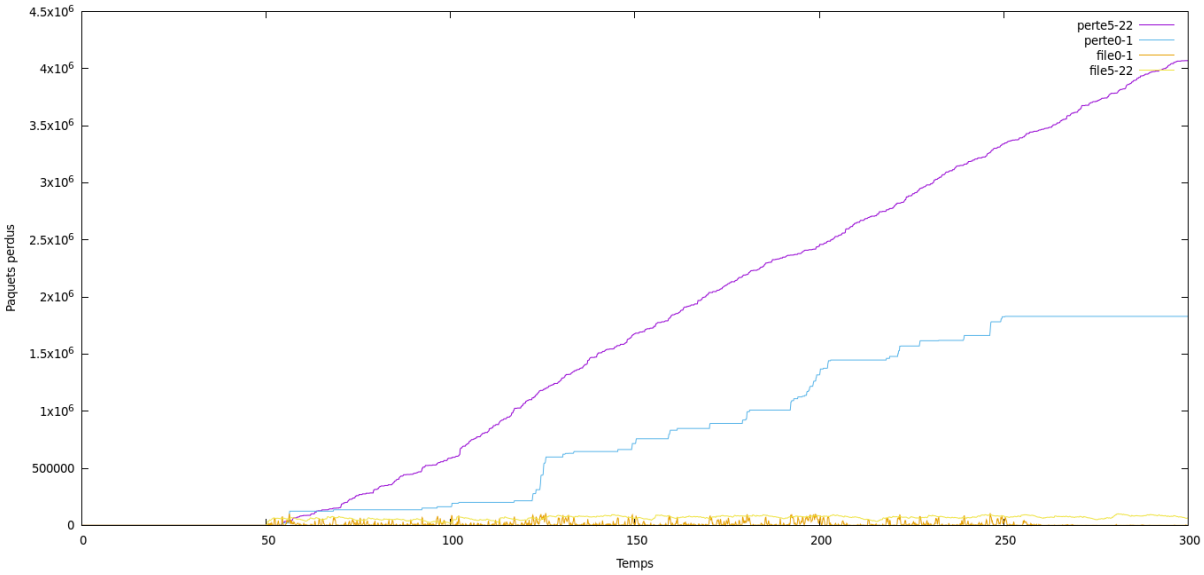


FIGURE 2 – L'évolution des pertes et la file d'attente pour les deux liens 5-22 et 0-1

On remarque bien que les pertes croit linéairement et rapidement pour le lien 5-22 à cause de la file d'attente qui est toujours pleine comme la montre la figure puisqu'il reste tout le temps remplie (constante égale à 100 limite de file d'attente),et donc il y'a des paquets qui sont détruits à cause de la saturation. Par conséquence, la mauvaise gestion des files d'attentes permet une grosse congestion au niveau de notre réseau puisqu'il n y'a pas d'équité des files(**Fair Queueing**. D'autre part, pour le lien 0-1, on voit bien que la file d'attente se vide de temps en temps, ce qui permet d'empêcher plus de paquets d'être perdus. Comme solution à ce problème, il faut augmenter le débit de ce lien qui est de $2.5GB$ à une valeur très grande pour

permettre un envoi des paquets plus vite avant saturation de la file d'attente ou bien augmenter la limite de la file d'attente. On remarque que le noeud 22 est centrale dans notre réseau puisque les deux liens 5-22 et 18-22 qui le concerne subissent plus de pertes.

3 Gestion des files d'attente :

Les files d'attente que nous avons traité auparavant sont en *DropTail*, c'est à dire l'ensemble des files d'attente des liens sont des FIFO. Ce sont les premiers paquets acceptés qui sont transmis. Afin d'améliorer l'équité des pertes entre les flux, nous avons ensuite étudié l'influence de la discipline de file d'attente mise en œuvre sur les liens. Nous avons donc remplacé la discipline de file d'attente des tous les liens par une SFQ (Stochastic Fairness Queueing) ,qui est supposé être un algorithme de répartition équitable. Malheureusement, il s'avère que c'est pas le cas, puisque le taux de perte pour les trois pire liens a augmenté de 50% à 60%. La raison pour ceci est que SFQ subdivise le trafic en plusieurs file d'attentes FIFO, ensuite le trafic est envoyé à tour de rôle donnant pour chaque session la chance d'envoyer. Par contre dans notre cas, on veut que les trois pires liens aient plus de priorité . En plus, un autre inconvénient est que la bande passante utilisé sur les liens a faibles pertes est perdue inutilement de presque 50% puisqu'on les empêche d'envoyer à cause du processus de tour de rôle qui est supposé équitable. On constate donc que l'équité ici mène à des résultats catastrophiques.

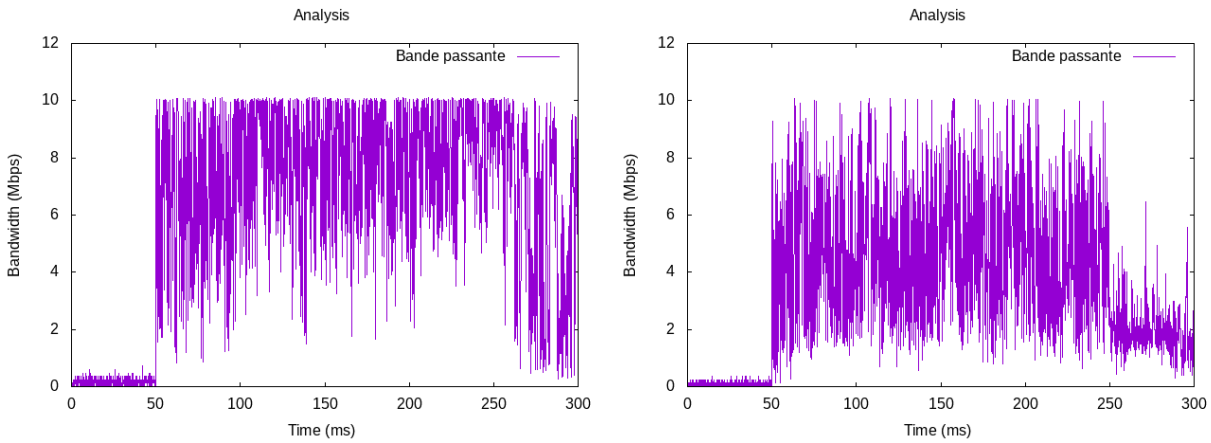


Figure 3 – l'évolution de la bande passante du lien0-1 au cours du temps avec le mécanisme Droptail(à gauche) et SFQ(à droite)

Le script awk permettait de calculer le débit utile. On trouve que : $Average\_throughput$: $6.081(Mbits)$ pour Droptail et $Average\_throughput$: $3.467(Mbits)$ pour SFQ. On remarque bien qu'il y'a consommation inutile de la bande passante pour une file d'attente utilisant SFQ. Afin d'améliorer la situation,j'ai tenté d'utiliser l'algorithme de discipline de file d'attente RED (Random Early Detection) . Mais,en vain, le taux de pertes est encore pire qui s'approche de 90% pour les trois pires liens. Par contre, j'ai remarqué que le débit augmente un peu pour le lien 0-1 : $Average\_throughput$: $6.422(Mbits)$

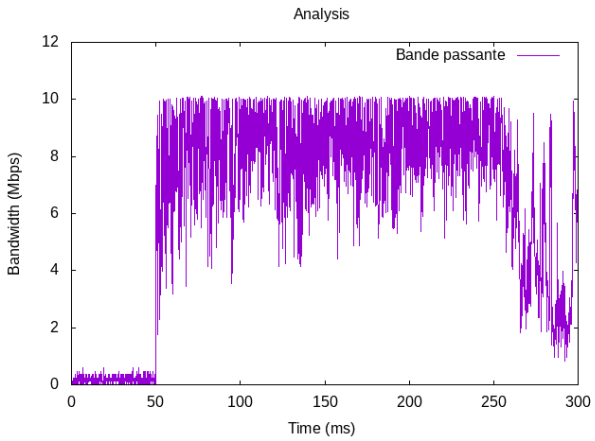


FIGURE 4 – l'évolution de la bande passante du lien0-1 au cours du temps avec le mécanisme RED

On constate donc que RED est plus juste que Taildrop, dans le sens où il ne possède pas de biais contre le trafic en rafale qui n'utilise qu'une petite partie de la bande passante.

4 Changement des instants de départ :

On change les instant de départs selon une loi normale de $moyenne = 150$ et $variance = 75$.

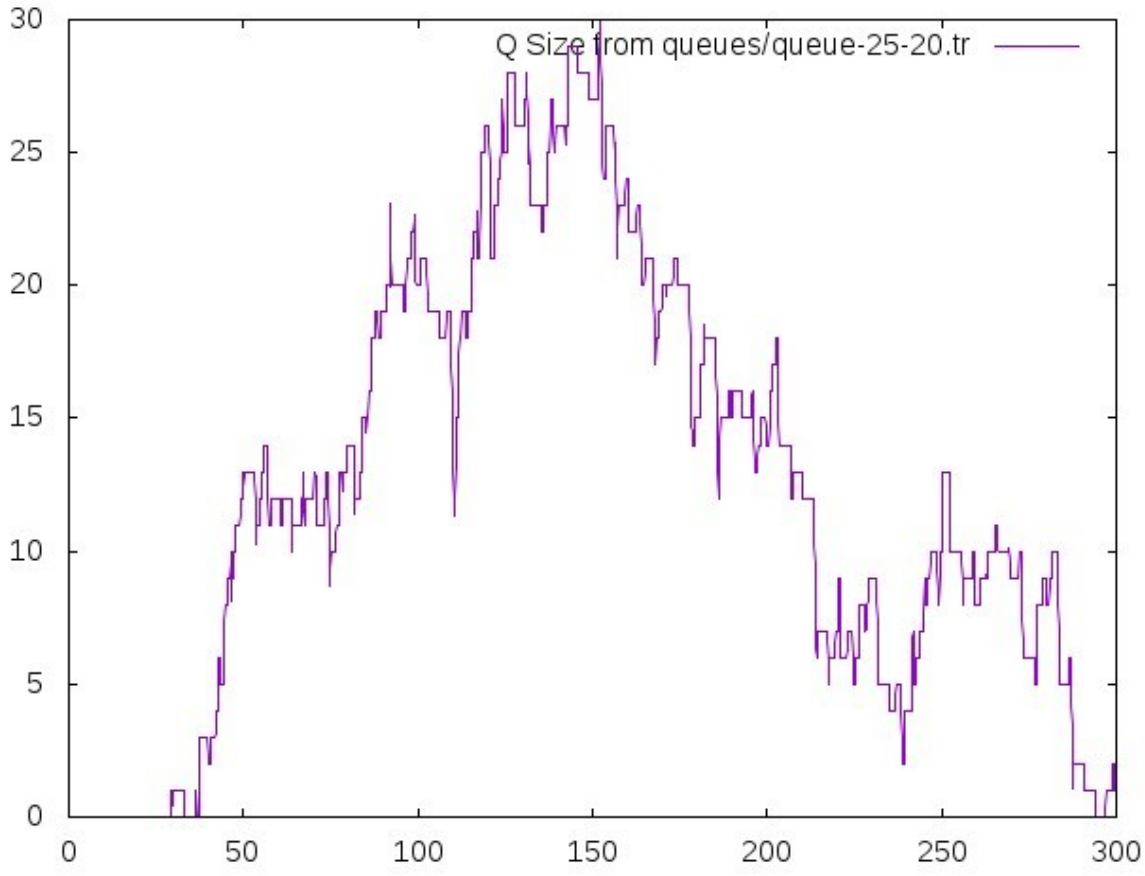


FIGURE 5 – l'évolution de la taille du file d'attente au cours du temps selon un départ gaussien

On remarque bien que la taille maximale est atteinte au alentours de 150 puisqu'il y'a plus

de probabilité de tirer un instant de départ à ce moment là. Ce départ gaussien est plus réaliste puisque dans la journée il y'a des moment ou il y'a plus de trafic qui circule(Beaucoup d'activité au milieu de la journée et faible activité aprés minuit) et donc les instants de départs suivent une loi normale de moyenne égale au temps du milieu de la journée où il y'a plus de trafic. C'est à ces instants qu'on détecte plus de congestion puisque les files d'attente saturent.

Par souci de temps j'ai pas étudié les émetteurs/récepteurs TCP selon plusieurs variantes. Mais d'après le dernières projet dans la matière de Transport et service réseau, on a pu remarquer que CUBIC et WESTWOOD ont une meilleure gestion des LFN (Long FatNetwork).