2012

Agrégation de liens xDSL sur un réseau radio

Rapport TX

Suiveur : Stéphane Crozat | Commanditaire : Tetaneutral.net/Laurent Guerby

Jocelyn Delalande;Yanick Delarbre
30/01/2012

Sommaire

Chapitre 1 : introduction

I. Introduction

A. Le besoin

Un opérateur utilisant des technologies radio dans son réseau va devoir collecter le trafic de et vers Internet sur un ou plusieurs sites utilisant des technologies filaires comme l'ADSL, le SDSL ou fibre optique. Sur un site de collecte du réseau, pour des raisons de coûts et d'efficacité, plusieurs lignes ADSL peuvent être ouvertes en utilisant les multiples paires de cuivre présentes dans tous les logements de France, mais ces lignes peuvent avoir des caractéristiques de débit et de latences différentes. Les raisons principales proviennent de l'historique des travaux, ou plus simplement du fait que chaque ligne sera ouverte avec un opérateur différent. Du côté radio, les équipements actuels offrent des débits très largement supérieurs à l'ADSL, de l'ordre de 100 Mbit/s pour un lien de qualité jusqu'à quelques kilomètres.

L'objet de la TX est de mettre au point une ou plusieurs solutions pour faciliter l'exploitation d'un tel réseau radio collecté sur ou plusieurs sites en filaire en permettant l'agrégation de lignes en vue d'augmenter le débit disponible et la redondance de sites en vue d'augmenter la fiabilité du réseau. Le cadre théorique général est celui du routage multipath[1], mais ce projet vise une solution spécifique pour des cas de topologie relativement simples mais présents dans la pratique, et non une solution générique.

B. Précisions sur le sujet

Une partie du sujet a en fait été définie plus clairement au cours de nos échanges avec Laurent Gureby. On retiendra les points suivants :

Polyvalence

Nous voulons pouvoir passer à travers non seulement des connexions opérées par tetaneutral.net mais également à travers nimporte quelle connexion, y compris une connexion 3G ou un wifi ambiant (ex: neuf wifi/FreeWifi), aussi, il faut que le système puisse fonctionner derrière un NAT côté client.

Adressage public

Nous voulons pouvoir assigner des adresses IPs publiques, nous ne pouvons donc pas baser notre solution sur un NAT.

Hétérogénéité

Nous voulons pouvoir agréger des liens de nature et de capacité très différentes et variantes au cours du temps.

[1] Utilisation de plusieurs métriques afin de déterminer le meilleur chemin

C. Schéma général de fonctionnement

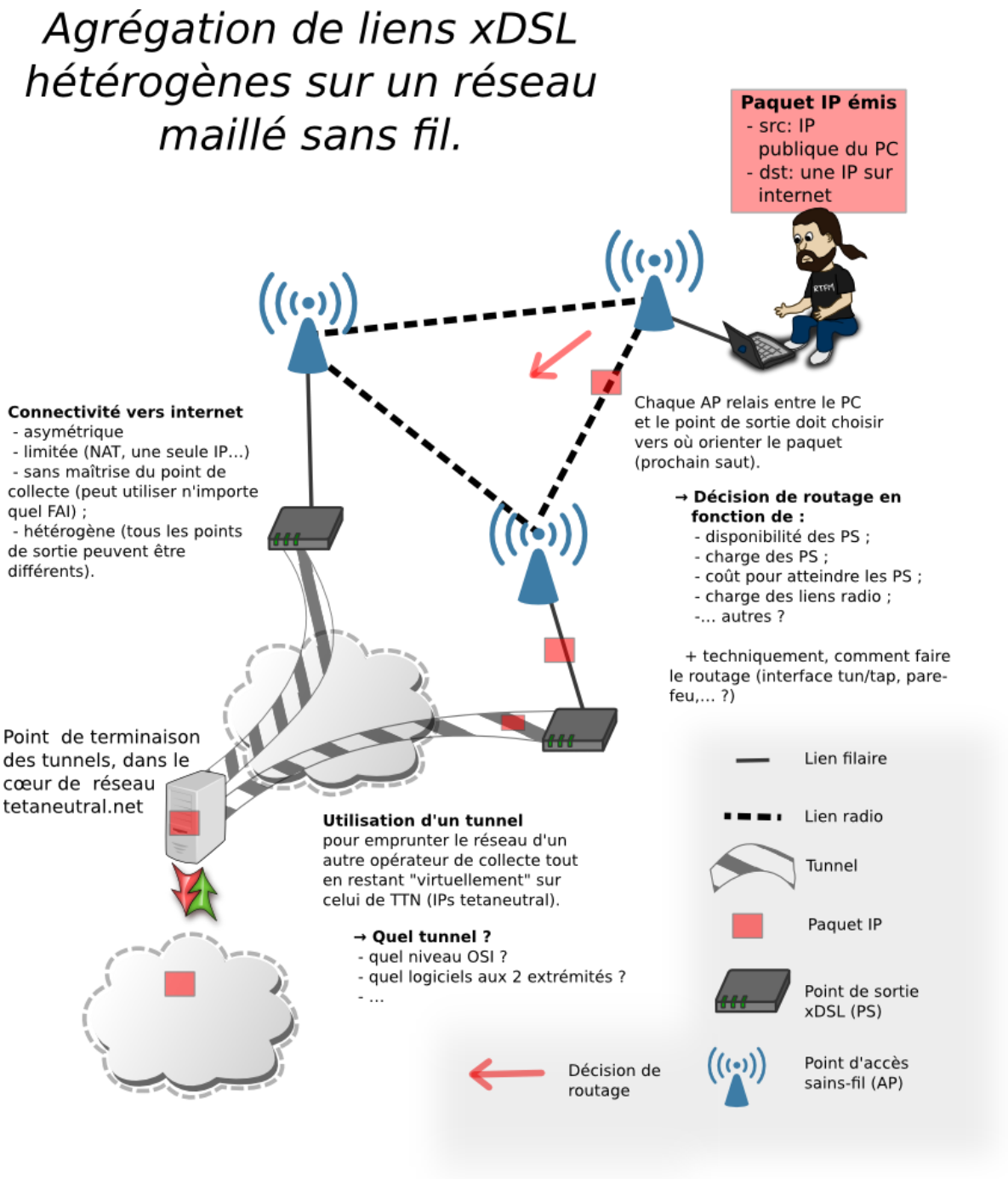


Figure 1 : Agrégation de liens xDSL hétérogènes sur un réseau maillé sans fil

D. Les acteurs du projet

Les acteurs du projet :

Tetaneutral.net – association Toulousaine exerçant les fonctions de F.A.I, d'hébergeur internet et d'opérateur sous forme associative et sans but lucratif ;

Toulouse sans fil – association Toulousaine possédant un réseau alternatif, libre, haut débit et gratuit sans le support d'opérateurs commerciaux ;

Rhizome – association étudiante ayant pour but de fournir un accès Internet aux étudiants ;

Laurent Guerby – président de tetaneutral.net et membre de Toulouse sans fil, et commanditaire pour la TX ;

Stéphane Crozat – suiveur de l'UV TX au sein de l'UTC ;

Fernando Alves – créateur de linkagreg et président du FAI *Sames Wireless* (wifi+ADSL en zone blanche);

Enseignants – conseils techniques ;

Bénévoles tetaneutral.net – conseils techniques ;

Trébons Haut Débits – conseils techniques ;

Sames Wireless – conseils techniques ;

Autres associations « WiFiste » - conseils techniques.

E. Les données d'entrées et les ressources

Les données dont nous disposons pour le projet sont : l'analyse de l'étude de faisabilité, l'expertise de tetaneutral.net et de Toulouse sans fil. Tout au long du semestre, nous avons utilisés le matériel WiFi de l'association Rhizome, une machine virtuelle prêtée par tetaneutral.net nos machines personnelles.

Chapitre 2 : les solutions existantes

II. Les solutions existantes

A. Agrégation en général

L'agrégation a deux objectifs principaux: augmenter la bande passante et la redondance. Les enjeux sont donc de répartir efficacement le trafic réseau et d'avoir une résilience rapide et transparente en cas de panne.

La répartition du trafic doit utiliser au mieux la capacité totale disponible. La capacité totale étant la somme de la capacité des liens disponibles. La notion de capacité d'un lien réunit deux métriques : la bande passante (montante et descendante), ainsi que la latence.

L'agrégation locale fonctionne selon le mode master/slaves. Les slaves sont les liens physiques, plus précisément les différents points de sortie, et le master décide de la répartition du trafic entre les slaves. Le fonctionnement peut s'opérer à plusieurs niveaux par rapport au modèle OSI :

- Couche 1 : physique
  - MiMo : utilisation de plusieurs canaux WiFi
- Couche 2 : liaison
  - MLPP [1][2]: combinaison de plusieurs liaisons physiques munies de tunnels PPP (ex : ligne ADSL) en une seule voie logique
  - Channel bonding [2]: utilisation d'une interface virtuelle utilisant N interfaces physiques

B. MLPPP

MLPPP, MultiLink Point to Point Protocol va permettre d'augmenter le débit entre deux points A et B lors d'une communication par la mise en œuvre de la coordination de plusieurs liens afin d'obtenir une bande passante qui est la somme de celles des liens coordonnés pour le Mulitlink. La force de MLPPP est de combiner des liens de technologies différentes comme les lignes téléphoniques, le RNIS, X25, Frame Relay, tant qu'ils véhiculent leur trafic via un tunnel PPP.

Cependant, le protocole MLPPP doit être mis en œuvre par l'opérateur de son côté du tunnel PPP (cœur de réseau) comme OVH ou Free. Typiquement, MLPPP permet d'agréger plusieurs lignes ADSL identiques d'un même opérateur à destination des professionnels. De plus, l'implémentation se révèle compliqué en pratique. Par exemple, l'agrégation se fera sur lien le plus lent. Plus concrètement, si on a 3 liens, chacun avec une bande passante réel de 3mbps, 5mbps, 10mbs, l'agrégation de ces trois liens sera : 3+3+3 = 9mbps et non 3+5+10=18 mbps. [3]

[2] Cette notation signifie qu'il s'agit d'une référence bibliographique.

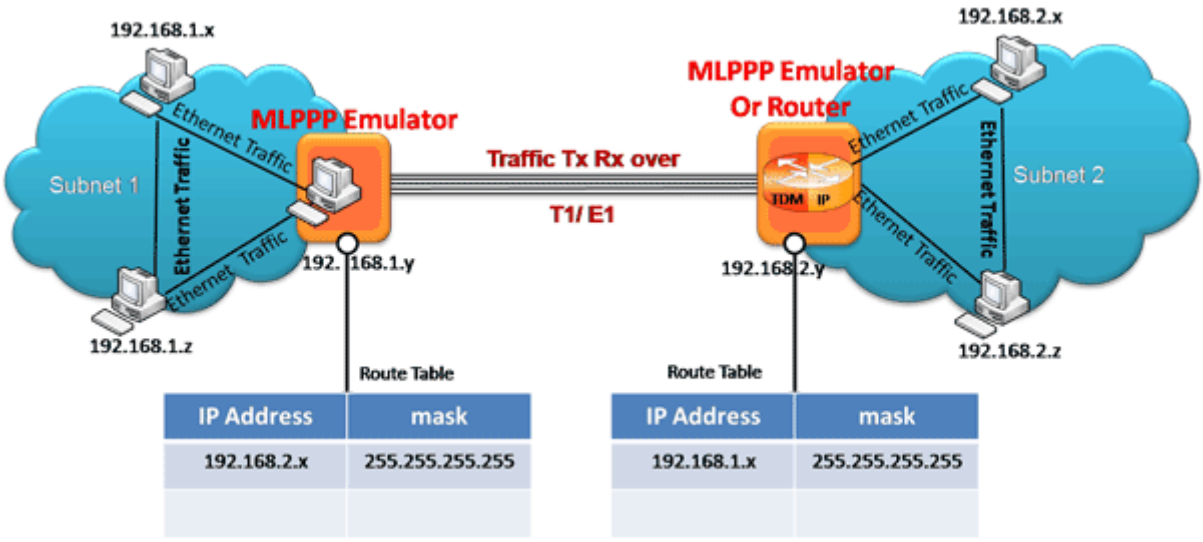
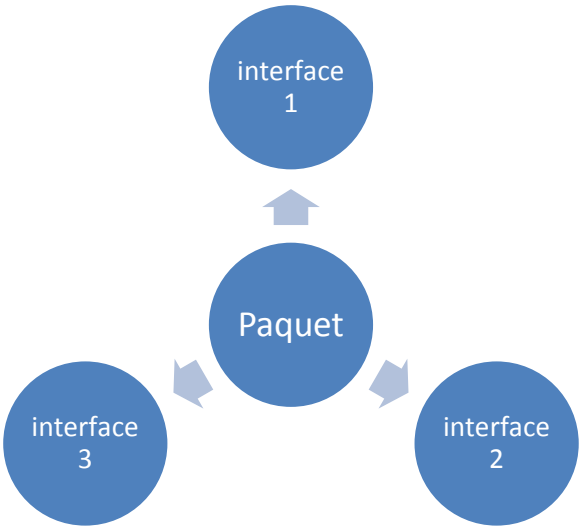


Figure 2 : Illustration du MLPPP [4]

C. Le chanel bonding

Le principe de fonctionnement est d'utiliser une interface virtuelle qui agrège N interfaces physiques. L'implémentation, sous linux, est le module bonding [5]. Une interface virtuelle est présentée au système, elle utilise de manière sous-jacente N interfaces physiques, soit en load-balancing, soit en fallback. Ce mode d'agrégation doit-être mise en place des deux côtés de la communication. Les différents modes de fonctionnements sont :

- mode 0 : round-robin
  - round-robin est un algorithme d'ordonnancement de répartition de charge. La répartition de charge se fait selon le principe du tourniquet :



- mode 1 : fallback simple
  - Si jamais un lien de communication devient indisponible, un autre lien est utilisé à la place
- mode 2 : fallback

    - Toute communication avec un hôte pair (adresse MAC du correspondant) utilisera une et une seule interface (choisie de manière déterministe par XOR).
- mode 3 : broadcast
- mode 4 : 802.3ad, load-balancing
    - Prise en compte dynamiquement de la capacité de chaque lien
- mode 5 : round-robin en utilisant une seule adresse MAC
- mode 6 : round-robin en utilisant deux adresses MAC (une par interface physique)

La détection de panne peut s'effectuer de deux manières différentes : vérification que la liaison de niveau 1 est « branchée », ou vérification que la passerelle répond via une requête ARP. La première vérification est passive (c'est-à-dire qu'elle ne génère pas de trafic réseau), la deuxième vérification est active (donc elle consomme de la bande passante). En mode 0 ou 2, les réponses ARP n'arrivent que sur un des esclaves si le commutateur ne prend pas soin de distribuer les réponses ARP sur tous les liens.

Dans le cas où le mode de bonding prend en compte la capacité d'un lien, un protocole est nécessaire pour communiquer entre eux l'état des liens, leurs débits. Par exemple LACP, protocole IEEE, ou PaGP, Cisco, sont les protocoles permettant cette communication d'informations. Cependant cette approche ne convient pas pour nos besoins. En effet, ces deux protocoles ne fonctionnent que sur des réseaux LAN et non sur des réseaux WAN.

Chapitre 3 : Etude expérimentale de la saturation des liens ADSL et de leur détection

III. Etude expérimentale de la saturation des liens ADSL et de leur détection.

Un des points sur lesquels nous nous penchons particulièrement est la détection de la capacité d'un lien et de son évolution, ceci pour utiliser au mieux des liens de capacités différentes et éventuellement changeantes. Le but est de pouvoir hiérarchiser les routes vers Internet en fonction de leur capacité. Cependant une mesure ultra-précise n'a pas forcément d'intérêt.

Nous avons donc besoin de mesurer deux éléments :

- mesurer la capacité d'un lien (la bande passante du lien) ;
- mesurer la charge du lien (la bande passante utilisée à un instant t).

Tout le challenge est de détecter (passivement) plutôt que de mesurer (activement) la capacité d'un lien, sans induire de trafic supplémentaire. [6]

On pourrait, par exemple, utiliser le SNR d'une liaison pour estimer passivement la capacité d'un lien. [7]

La solution que nous avons retenu est de surveiller le temps de réponse d'un ping initié par le tunnel lui-même, puis de trouver la relation qui la saturation du tunnel avec le ping qui s'envole. A terme, on regarde l'évolution du ping, et on arrive à détecter que le lien est saturer quand le ping dépasse du seuil fixé empiriquement.

Nous avons fait des mesures pratiques sur deux liens ADSL différents.

A. Tunnel

Un tunnel est une encapsulation de données d'un protocole réseau dans un autre, situé dans la même couche du modèle en couches, ou dans une couche de niveau supérieur.

Par exemple, pour faire passer le protocole IPv6 dans l'Internet actuel (qui est presque entièrement en IPv4) on va créer un tunnel entre deux machines IPv4; ce tunnel, pour le protocole IPv6, semblera un simple lien point-à-point (un logiciel comme traceroute ne verra donc pas le tunnel).

En l'occurence, on encapsule notre trafic au-dessus d'UDP (niveau 4), on fait passer dans le tunnel:

- soit les niveaux 2 et supérieurs (tunnel tap)
- soit les niveaux 3 et supérieurs (tunnel tun)

i. Le fonctionnement simplifié d'un tunnel

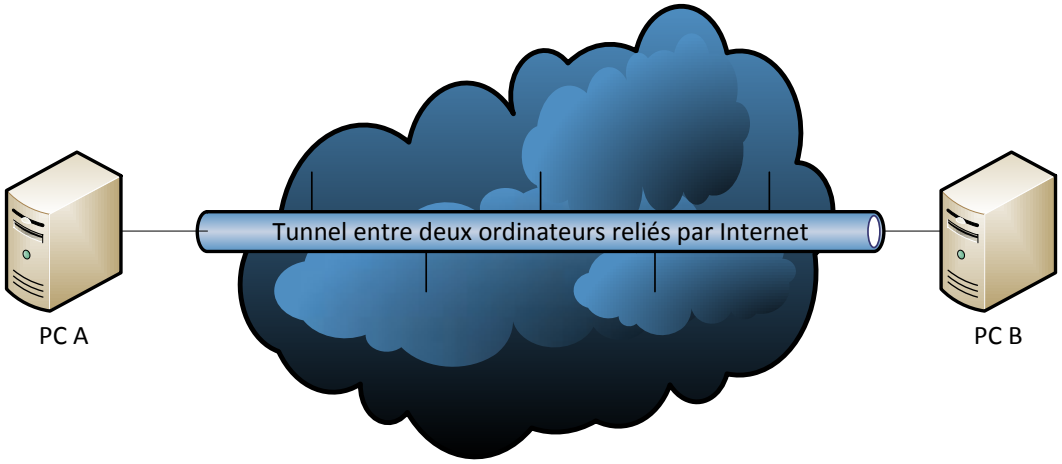


Figure 3 : Le fonctionnement d'un tunnel

Solutions à base de tunnel : estimation de l'overhead

L'utilisation d'un tunnel implique une surcharge due à l'encapsulation pour chaque paquet transmis (overhead). Afin de pouvoir estimer les performances de notre tunnel, nous avons estimé l'overhead induit par le tunnel afin de distinguer la perte de performance due à l'overhead (inévitable) de celle due au fonctionnement de notre tunnel.

Notre test porte sur le téléchargement d'un fichier de 42Mio, ce qui représente 48206 paquets TCP pour un total de 45 132 020 octets. Cela correspond peu ou prou à nos tests ultérieurs (connexion TCP et transfert unidirectionnel).

Selon le type de tunnel employé, nous avons les overheads suivants par paquet en ajoutant les en-têtes d'encapsulation :

- en tap : $ip + udp + mac = 20 + 8 + 14 = 42\ octets$
- en tun : $ip + udp = 20 + 8 = 28\ octets$

Soit un overhead théorique total de:

- tun : $48206 * 28 = 1349768\ (3{,}0\%)$
- tap : $48206 * 42 = 2024652\ (4{,}4\%)$

B. Comportement d'un lien ADSL sous la charge

i. Mesure ADSL Free

Il semble que de la QoS soit appliquée… l'effet de bufferbloat n'est pas vraiment visible : on passe d'un ping de 40 à 70/80ms… Tous les paquets de ping arrivent, même lorsque le lien est saturé.

ii. Mesure ADSL OVH

L'effet de la saturation se fait clairement ressentir sur le ping : on passe de 70 à plus de 300ms de ping lorsque le lien est saturé.

iii. Mesure ADSL FDN

Les résultats ont comparables à la ligne OVH

iv. Les conditions des tests

Le tunnel est fondé sur tunproxy.py, un script python permettant de réaliser un tunnel entre deux sites distants.

Ce tunnel est le seul à utiliser la connexion. Les données sont collectées toutes les secondes, et chaque nœud enregistre les informations suivantes :

- le timestamp ;
- les stats des paquets entrants (bande passante) ;
- le ping

La connexion est testée au repos, en la saturant par moments avec iperf, en TCP et en UDP en indiquant l'option –b à une valeur supérieur à la capacité d'uplink.

> Note : Le comptage du volume sortant n'est pas pertinent puisque la moitié des paquets peuvent-être droppés en cours de route. Il faut compter les paquets arrivant réellement du point de vue du récepteur.

v. Les résultats en TCP de la ligne OVH

Les graphiques ci-dessous correspondent à l'évolution du délai du ping selon la charge du lien en TCP. Pour ces tests, il y a deux charges réalisées grâce à l'outil load_uplink.py à deux instants différents.

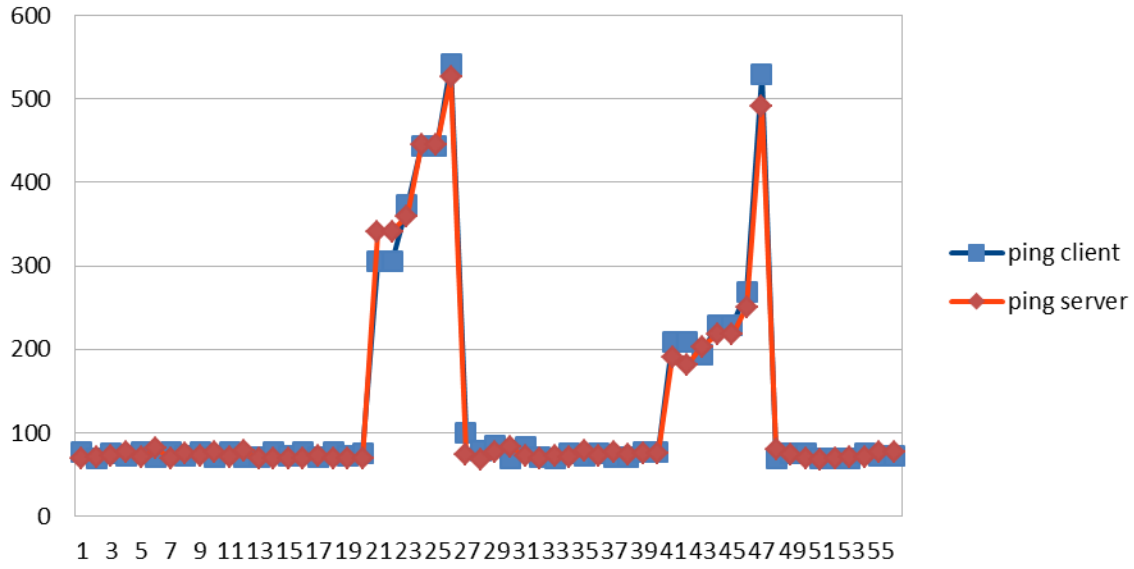


Figure 4: le temps en ms des pings

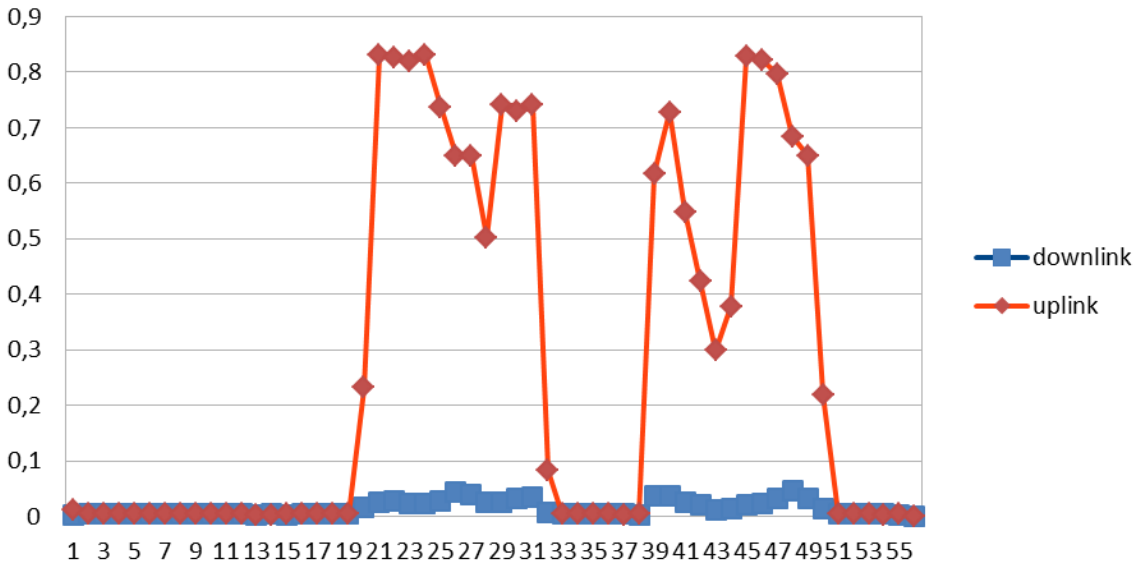


Figure 5: la bande passante utilisée (Mbit/s)

vi. Les résultats en UDP de la ligne OVH

Il s'agit du même scénario que celui en TCP. La seule différence est le protocole utilisée : UDP, qui ne fait pas de contrôle de flux.

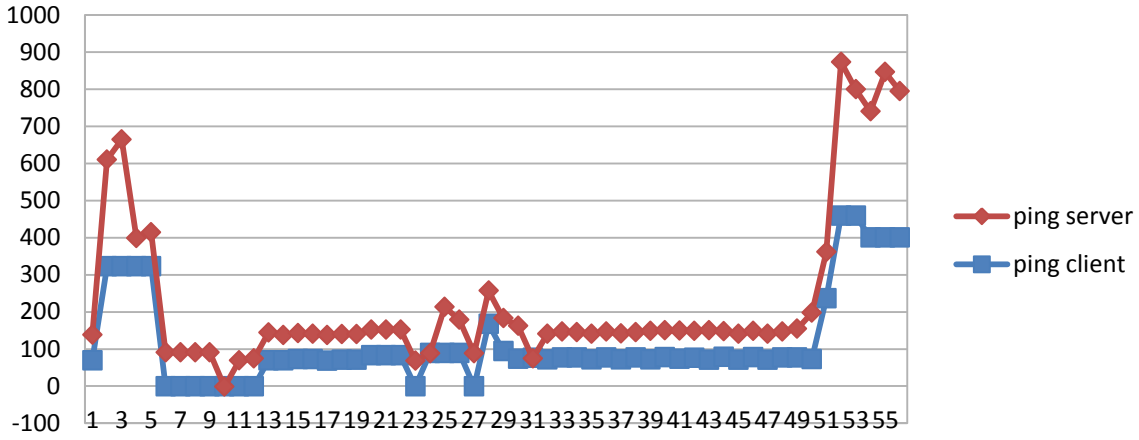


Figure 6 : Ping sous charge, test UDP (ms)

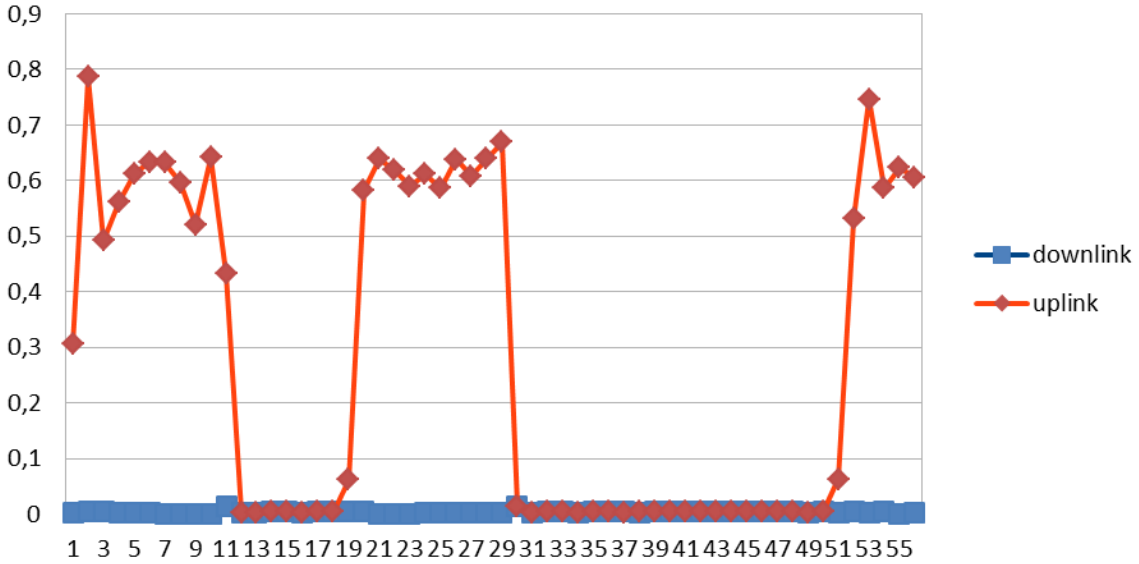


Figure 7 : la bande passante utilisée par le test UDP (Mbit/s)

vii. Analyse des résultats

On note systématiquement une corrélation forte entre le lien saturé et l'augmentation du ping, quelques soit le protocole utilisé (UDP ou TCP).

En UDP, on peut complétement saturer le lien. Le résultat est qu'une partie des pings se perd. On peut donc non seulement prendre en compte le RTT, mais également le taux de paquets perdus (loss rate).

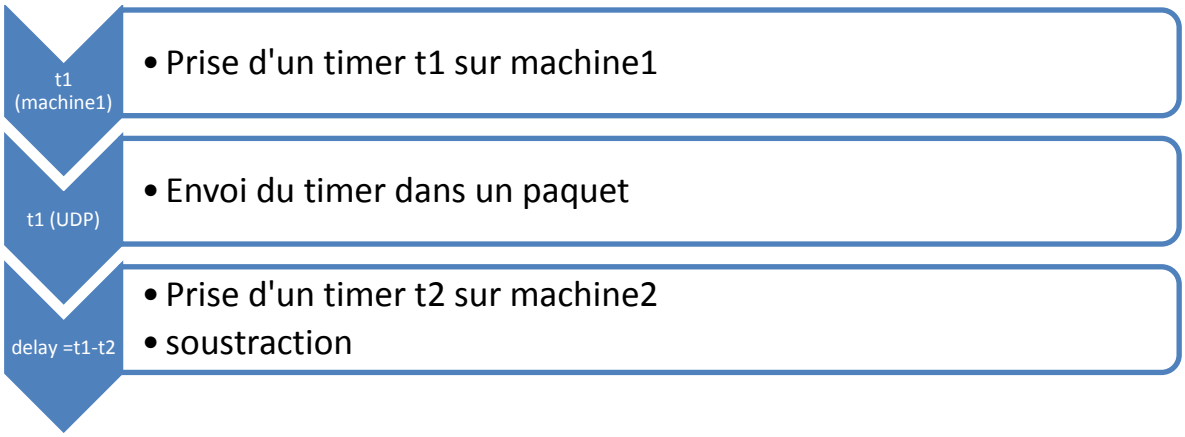
En TCP, on observe aussi une montée du ping significative, mais jamais de ping perdus. On constate par ailleurs, que TCP se rend compte qu'il sature el lien et réduit sa fenêtre d'émission (il s'agit du « trou » sur le graphique correspondant). Le pic de débit est cependant suffisant pour saturer la connexion et en permettre la détection.

Nous pouvons donc corréler une saturation du lien en observant une augmentation du ping, qu'elle soit provoquée par un protocole qui gère la congestion (TCP) ou non (UDP). Dans la réalité, la saturation sera causée par un ensemble de flux de types différents, gérant ou non la congestion.

Cependant, il reste un autre problème. En effet, nous voulons détecter dans quel sens a lieu la saturation. Or un ping nous donne le temps d'aller-retour (RTT : Round Trip Time). Une augmentation du ping signifie alors qu'un des deux sens est probablement saturé sans moyen de savoir lequel.

C. Détection de saturation par demi-délai

Naturellement, on imagine qu'il serait souhaitable de faire un « demi-ping » dont les fluctuations ne reflètent l'état que de la voie montante ou descendante. Mesurer seulement l'aller plutôt que l'aller-retour.



Il n'est en outre pas possible de mesurer la duréede transit absolue d'une trame entre deux sites, les horloges n'étant pas synchronisées. Nos pings sont de l'ordre de 50ms. Il faudrait donc avoir un décalage de moins de 5ms (10%) pour que les résultats aient du sens, c'est-à-dire que les horloges soient synchronisées à 5ms près. Deux approches sont donc envisagées pour résoudre ce problème : la synchronisation par NTP ou par l'évolution du délai relatif.

i. Synchronisation par NTP

NTP [8] est un protocole permettant de synchroniser via le réseau, les horloges de machines distantes. Si NTP fournit une précision suffisante, il serait intéressant pour pouvoir effectuer des demi-ping. Voici l'idée générale :

- On maintient les horloges synchronisées grâce à NTP entre le client et le serveur.
- Le client envoie un paquet au serveur contenant un timestamp.
- Le serveur peut connaître le temps de trajet client -> serveur en comparant ce timestamp avec sa propre horloge.

On mesure des ping entre 20 et 100ms en général, soit des demi-ping entre 10 et 50ms. Or, les [9] montrent qu'à travers un réseau WAN (ex: l'ADSL que nous utilisons), l'erreur de NTP est autour de **10ms**. Soit une erreur relative entre 10% et 50%, ce qui n'est pas acceptable. La seule solution viable, selon l'étude mentionnée, pour synchroniser réellement des équipements serait d'avoir une source GPS qui permet d'avoir une erreur en-dessous de la milliseconde. Cela nécessite de l'équipement supplémentaire et n'est souhaitable.

ii. Par l'évolution du délai relatif

Une autre approche discutée avec Laurent Guerby est de mesurer non pas le délai absolu mais la variation de celui-ci. On mesure le *timestamp_envoi_site_1 – timestamp_reception_site2* pour chaque paquet, la valeur absolue n'a aucun sens car on utilise deux horloges locales différentes et non synchronisées.

En revanche, nous pouvons mesurer l'évolution relative de cette valeur, à condition que les deux horloges ne se décalent pas trop entre elles, Cette idée de délai relatif a d'ailleurs été utilisée dans le protocole UTP de bittorrent. [10].

Cela nous amène au problème de savoir si la dérive relative des horloges gène nos mesures. Une exemple donné dans l'article de [10] est une dérive de 17ms en 10mins. Pour notre application, nous souhaitons comparer la valeur du demi-délai au minimum des 10 dernières minutes (qui constitue notre référence).

Pour évaluer cette dérive relative, on mesure la dérive des horloges entre deux machines géographiquement éloignées et matériellement différentes sur 40 minutes. L'enjeu est de savoir si il est nécessaire, dans notre cas, de mettre en place un mécanisme pour détecter et prendre en compte la dérive des horloges qui rendraient la comparaison de deux délais relatifs peu pertinents si elle était trop importante.

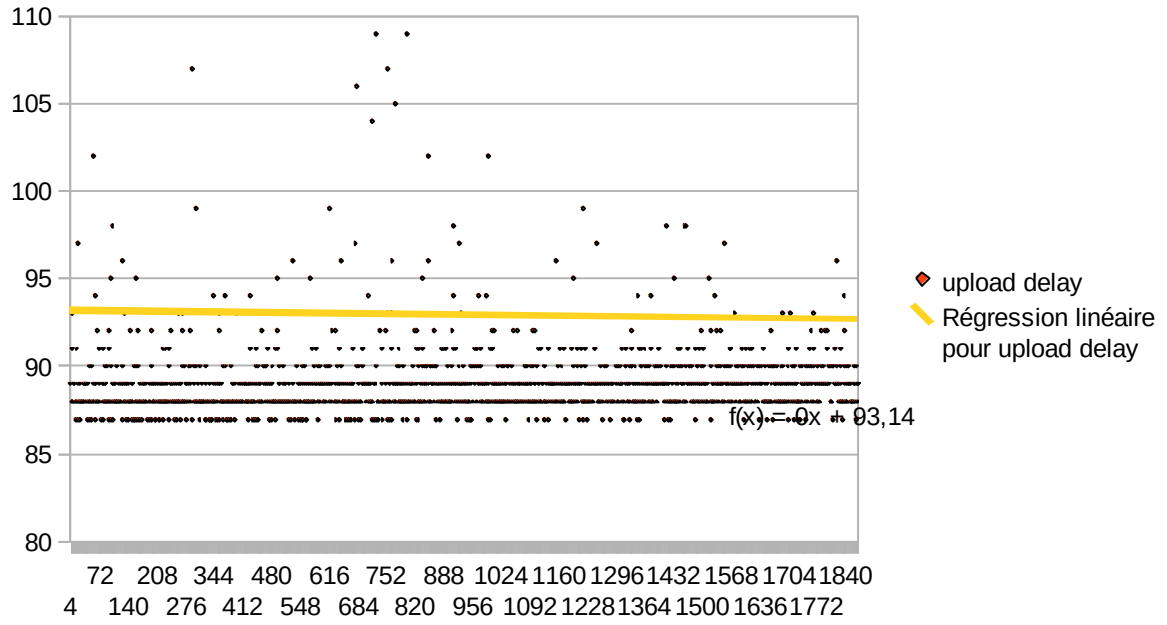


Figure 8: Dérive relative des horloges entre deux machines distantes via le lien ADSL

Sur les 40 minutes d'échantillon, on ne constate pas de dérive particulière. La régression linéaire nous donne une dérive de -0.494804806 ms sur les 40 minutes.

Bien que l'expérience ne porte que sur un cas et ne fasse pas loi, elle nous expose une dérive de 0.5ms sur 40 minutes d'observation (dérive relative de ~1.4%). Ne souhaitant garder pour nos mesures de capacité de lien qu'une fenêtre glissante que de quelques minutes ou dizaines de minutes tout au plus, il n'apparaît pas nécessaire de prendre en compte cette dérive.

a. Résultat : délai avec la saturation TCP

Nous avons donc implémenté un script qui surveille le demi-délai relatif selon le principe décrit précédemment : delta_half_trip_time.py

On souhaite observer l'évolution du délai quand on sature le lien avec l'outil *iperf* en TCP pendant 10s dans un sens (upload) puis dans l'autre (download).

Note : Ces mesures correspondent peut être au cas le plus difficile à détecter (une unique connexion TCP qui sature le lien) étant donné que le backoff de TCP va essayer d'éviter de saturer le lien en permanence.

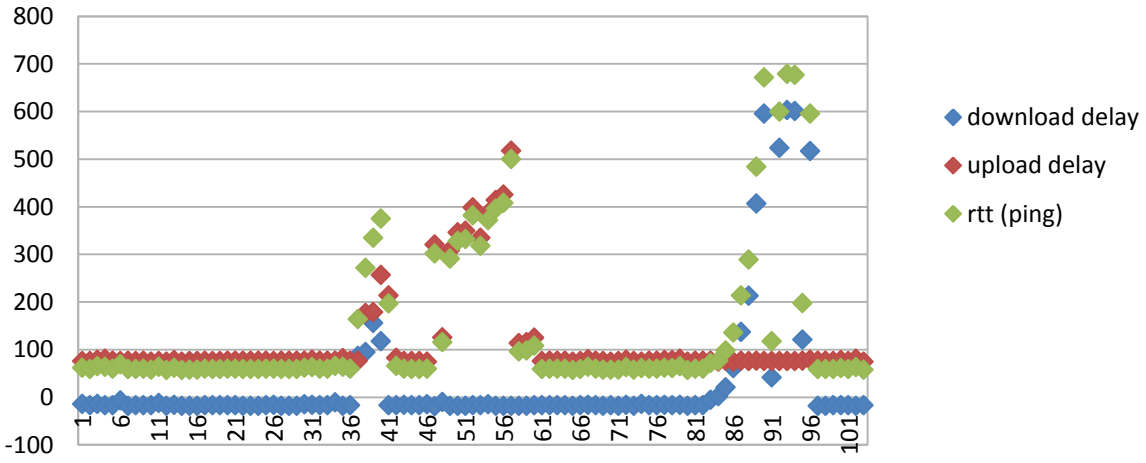


Figure 9 : La saturation TCP

Le premier pic correspond à une saturation de l'upload, le second à uns saturation du download. Dans les deux cas, le RTT (ping) s'envole mais on peut distinguer dans quel sens est la saturation grâce aux mesures de délai dans chaque sens.

On note une première envolée des délais vers 34. Cela est sûrement dû à un usage autre que celle du scénario prévu ; le réseau de test n'étant malheureusement exempt de perturbations. (cf D.)

b. Résultat : délai avec la saturation UDP

Ce jeu de mesure permet d'étudier la saturation sur des périodes plus petites (5 sec).

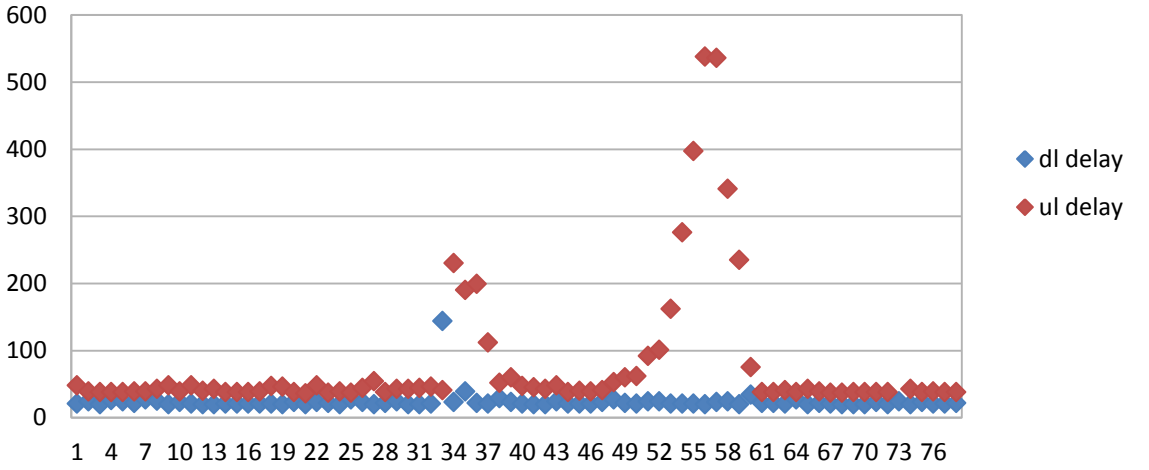


Figure 10 : La saturation UDP

On observe qu'il n'y a pas de saturation progressive. Soit le lien est saturé et en quelques secondes, le delay s'envole, soit il ne l'est pas et le délai reste stable. Plus précisément, le premier pic est une occupation du réseau non souhaitée, et le deuxième pic correspond à une charge progressive obtenue grâce à l'outil load_uplink.py. On note comme précédemment un artefact autour de la seconde 34.

c. *Déduction d'une formule*

A partir des résultats précédents, l'idée était de trouver la formule qui à partir des n derniers delays et du délai minimum des 10 dernières minutes est capable de dire si oui ou non le lien est saturé.

Le but d'une telle détection est de « capturer » l'instant où la saturation est effective, de regarder quelle est la bande passante utilisée à ce moment et de l'enregistrer comme « poids » du lien. La répartition de la charge entre les liens agrégés s'effectuant ensuite en fonction de ces poids.

Il n'est pas impératif de détecter tous les pics mais on veut éviter à tout prix les faux positifs. Des variations de délais ne signifiant pas forcément une saturation du lien. Un faux positif aurait pour effet de sous-noter temporairement un lien, ce qui réduirait l'efficacité de l'agrégation… Une bonne formule doit donc permettre de ne pas détecter les pics « parasites » comme une saturation. En général, les pics liés à la saturation ne sont pas isolés mais regroupent une ou plusieurs valeurs de pic plus quelques autres intermédiaires. Il convient donc de prendre en compte plusieurs échantillons successifs pour déterminer s'il y a oui ou non saturation.

On arrive à une formule de ce genre :

```
Soit "l_derniers" les <hist_size> derniers échantillons de délai


   SI <peak_size> échantillons sont supérieurs à TRIGGER ET
<fraction_size> autres échantillons de l_derniers au moins sont
supérieurs à <fraction>*max(l_derniers), alors SATURATION DETECTEE
```

Afin de trouver les valeurs des constantes de cette formule, nous avons testé différents paramètres sur des scénarios divers (cf en annexe feuille de calcul) dans un tableur en nous appuyant sur des cas réels (valeurs enregistrées par delta_half_delay.py puis exportées en CSV vers le tableur). Un tableur nous a ensuite permis d'essayer sur ces scenarios différentes valeurs de constantes en regardant à chaque fois si les moments de saturations (observés lors de la mesure) étaient bien détectés et surtout si aucun faux positif n'était produit.

Les valeurs que nous avons trouvées sont :

- *TRIGGER*=300 (en ms, le seuil par rapport au minimum sur les 10 dernières minutes)
- *peak_size*=1 (nombre d'échantillon supérieur à TRIGGER)
- *hist_size*=6 (échantillons de 1s pour l'historique)
- *fraction_size* = 3 (échantillons supérieur à TRIGGER*fraction)

- *fraction* = 0.25 (coefficient minorant TRIGGER)

Enfin, nous avons modifié une partie de la formule : ce à quoi s'applique la fraction n'est plus la taille du pic mais une moyenne pondérée entre la taille du pic (pondéré à 2) et le TRIGGER (pondéré à 1). Cette modification a été opérée car dans certain cas, au milieu d'une saturation on trouve un pic seul et très haut.

On a donc au final détection si, parmi les 6 derniers échantillons :

- 1 échantillon est d'une valeur *peak* supérieure à 300ms
- 3 autres échantillons sont supérieurs à 0.25*MOYENNE(*peak, peak, 300*)

Quelques remarques concernant cette formule :

- Elle évite les faux positifs
- Elle ne peut pas être active dès les premières secondes de fonctionnement du programme (nécessité d'un historique)
- On détecte les pics à posteriori, il faut donc garder un historique des débits seconde par seconde pour retrouver à quel débit correspondait le pic.
- Elle ne permet pas de détecter les pics très courts… De toutes façons, les saturations courtes (<= 3s) ne se traduisent pas par une augmentation de latence puisqu'elles ne parviennent pas à saturer les buffers.

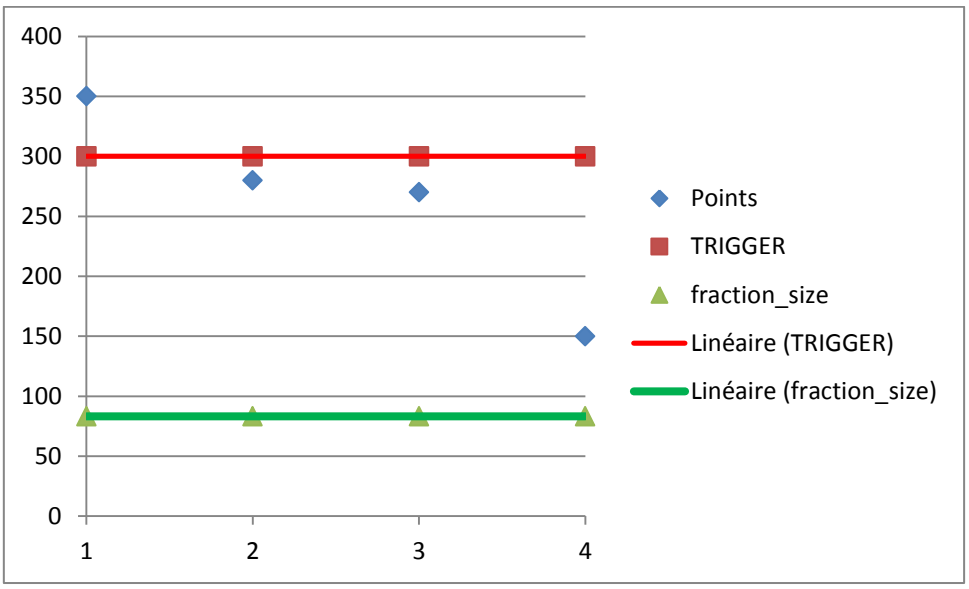


Figure 11 : Comprendre la formule de détection des saturations dans le cas d'une saturation

Les points correspondent par exemple à une série de valeur. Linaire (TRIGGER) correspond au seuil de trigger et Linéaire (fraction_size) correspond à 0.25*MOYENNE(*peak, peak, TRIGGER*).

Ici, on constate bien que le point 1 est au-dessus de TRIGGER (donc il s'agit de peak), et qu'au moins 3 points autre que le peak sont au-dessus de 0.25*MOYENNE(*peak, peak, TRIGGER).* On a donc ici une détection de saturation.

D. Influence du réseau radio

Nous avons souhaitez mesurer si la qualité d'un lien radio influait ou non sur le délai d'un ping.

En l'absence de deux connexions internet reliées par lien radio jusqu'à un stade avancé du projet, nous avons fait les tests présentés dans les parties précédentes sur une seule connexion et via un réseau wifi « classique » (wifi de box, bande des 2.4GHz).

À quelques semaines de la fin du projet, nous avons pu obtenir un cas plus réaliste par rapport à notre sujet, c'est-à-dire une connexion en local (filaire) et une autre amenée sur le même lieu que la première grâce à un réseau radio à 5GHz via du matériel de qualité opérateur de la marque Ubiquiti, ce qui correspond exactement à la situation de l'opérateur tetaneutral.net.

Comme décrit lors de l'introduction, le réseau radio est censé peu influer sur le débit de l'ADSL, qui se trouve être le goulot d'étranglement. Nous avons cependant voulu voir si la présence de ce réseau radio influait sur les délais, qui sont nos principaux paramètres de mesure. En considérant vraiment un cas « extrême» :

- 3 sauts radio en 5GHz (4 équipements radio traversés)
- dont un équipement[3] en relais
- un kilomètre parcouru au total, pas de vue directe (1 lien avec des toitures, un avec un morceau de bâtiment et un avec un bâtiment à traverser).
- puissance du signal sur les différents équipements : -78dBm, -76dBm, -78dBm, -82dBm (signal faible).

Les deux mesures présentées sur les graphiques ci-après montrent le délai d'un bout à l'autre du lien réseau présenté, d'une machine raccordée au nœud tout à droite jusqu'au nœud en bas à gauche. Dans le premier test, la machine est raccordée au nœud par du wifi 2.4GHz (10m), dans le second directement par un câble réseau.

[3] Le fait d'utiliser un équipement radio en « relais », c'est-à-dire à la fois pour recevoir et émettre le même datagramme vers un autre nœud utilise divise le débit par deux et peut introduit une latence supplémentaire (les interfaces sans-fil sont half-duplex).

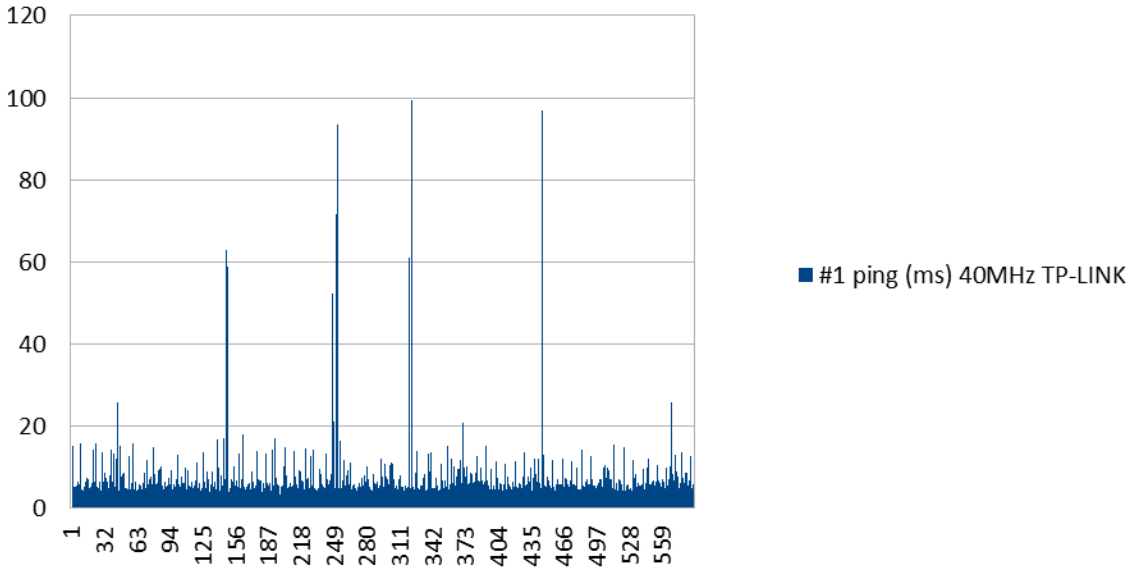


Figure 12 : L'évolution du délai, 3 sauts 5GHz + 1 saut 2.4GHz (10 minutes)

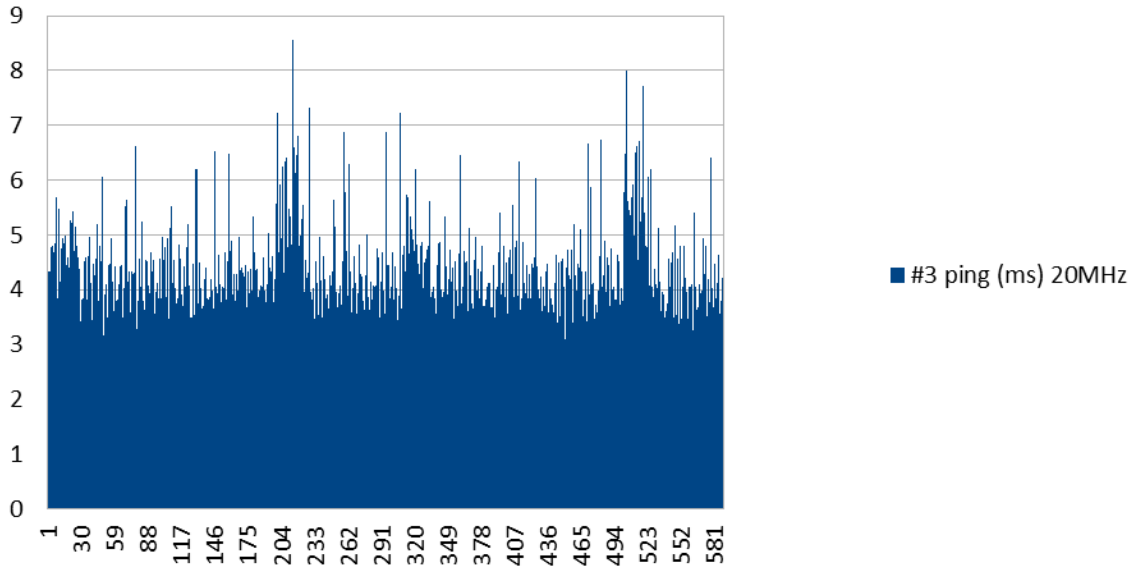


Figure 13 : L'évolution du ping : 3 sauts 5GHz (10 minutes)

Sur ces deux graphiques, on constate un écart type de 8.66 pour le premier et un écart type de 0.79 pour le deuxième graphique. La dispersion du délai autour de la moyenne est beaucoup plus importante pour le premier graphique, que le second.

La conclusion est que les liens radio en 5GHz, même dans des conditions difficiles rajoutent environ 4ms de délai, délai que varie peu et ne faussera donc pas notre détection (cf Figure 13).

En revanche, nous nous rendons compte que nous avons fait une erreur en voulant approximer le réseau 5GHz longue distance par une connexion en 2.4GHz courte distance. Même à quelques mètres du point d'accès, la bande des 2.4GHz est si bruitée (toutes les «box» des

FAI l'utilisent) que, comme on le voit sur la Figure 12, on observe des pics aléatoires de ping, de l'ordre de 100ms pour certains. Cela peut fausser nos mesures et explique les quelques « pics » aléatoires que nous avons pu observer jusqu'à-présent.

Les tests présentés dans la suite de ce rapport passent par un réseau 5GHz mais pas par un réseau 2.4GHz pour éviter de fausser les mesures.

E. Conclusion

Pour résumer cette partie, nous retiendrons quelques points :

- le ping permet de détecter une saturation sur un lien ADSL, sans connaître son sens ;
- le demi-délai relatif nous permet de détecter une saturation et son sens (voie montante ou descendante), une formule évitant les faux positifs a été trouvée ;
- le fait que les points de sortie vers internet soient reliés par un réseau radio 5GHz ne devrait pas gêner la détection par variation du demi-délai relatif ;
- si les liens entre les points de sortie étaient en 2.4GHz, cela pourrait poser problème.

Chapitre 3 : Approches abandonnées

IV. Approches abandonnées

A. Ajout de métriques à B.A.T.M.A.N[4][5]

Pour la première approche, nous souhaitions partir sur du routage dynamique pour effectuer de l'agrégation de liens distribués. L'idée principale était que chaque site radio avait une route par défaut dépendant de la charge des liens de sortie. La route par défaut étant le prochain saut emprunté par les paquets à destination d'Internet. Par exemple B.A.T.M.A.N, protocole de routage dynamique permet d'attribuer plusieurs routes par défaut. Nous souhaitions modifier dynamiquement la route par défaut selon les métriques du réseau. Les métriques sont la bande passante utilisée.

i. B.A.T.M.A.N

B.A.T.M.A.N. [11], comme Better Approach To Mobile Adhoc Networking est donc un protocole de routage dynamique, dont l'idée centrale réside dans le fait de partager les informations sur les meilleures connexions entre tous les nœuds du réseau complet. Chacun des nœuds regarde uniquement d'où viennent les données reçues par leur partenaire de communication, et renvoient les données correspondantes via le même chemin. Les données sont transmises de cette manière de proche en proche. Grâce à cela, la nécessité d'informer l'ensemble des nœuds B.A.T.M.A.N. à chaque modification des routes disparaît.

ii. Fonctionnement de B.A.T.M.A.N.

La tâche principale de B.A.T.M.A.N. est la même que pour les protocoles de routage classiques. Il s'agit de découvrir les autres nœuds B.A.T.M.A.N. et de calculer la meilleure route vers ces nœuds. En plus de cela, il informe ses voisins sur les nouveaux nœuds et les routes vers ceux-ci.

Chaque nœud informe régulièrement ses voisins à travers un message de broadcast de son existence. Chaque voisin répond par son message d'existence, ce qui permet aux voisins des voisins d'apprendre également l'existence de ce nœud. C'est comme ça que l'information sur chacun des nœuds est distribuée dans le réseau complet. Pour trouver le meilleur chemin vers tous les voisins, B.A.T.M.A.N. compte les messages reçus d'une même origine, et mémorise quel voisin lui a transmis. En opposition aux solutions existantes, le protocole n'essaye pas de définir le chemin complet vers un autre nœud, mais utilise les origines des messages collectés pour évaluer le premier saut dans la bonne direction. Ces données seront alors transmises uniquement au meilleur voisin pour cette direction, lequel appliquera le même principe. Ce processus se répète jusqu'à ce que les données arrivent au destinataire.

iii. La gestion de l'agrégation de liens

Dans la dernière version de B.A.T.M.A.N. [12], chaque nœud peut annoncer au réseau, qu'il offre un accès à Internet. Les utilisateurs sont alors capables de savoir qu'une connexion Internet est disponible à proximité et quelle bande passante est disponible. Ils peuvent alors choisir une

[4] Il ne s'agit pas d'un protocole de transport immeuble à immeuble en mode volant sous forme de chauve-souris.

[5] Note de note de B.A.T-page

passerelle spécifique ou alors laisser cette décision à B.A.T.M.A.N. (se basant sur des critères comme par exemple la connexion Internet la plus rapide).

De plus, il est désormais possible d'annoncer un périphérique dans un réseau B.A.T.M.A.N., même lorsque celui-ci n'est pas configuré avec B.A.T.M.A.N. C'est habituellement par ce biais que les réseaux domestiques peuvent se connecter au réseau maillé. Par exemple, l'installation sur le toit d'une antenne permet via B.A.T.M.A.N. de communiquer avec le reste du réseau maillé, le reste de la maison ayant accès au réseau maillé via cette installation.

B.A.T.M.A.N choisit ses routes en fonction de la qualité des liens radio**. Notre idée** était d'ajouter, pour la route vers internet des métriques concernant la capacité de chaque lien de sortie vers internet. Chaque paquet étant ensuite routé de nœud du réseau radio en nœud du réseau radio en fonction de ces deux paramètres.

iv. Abandon de cette approche

Devant la difficulté des protocoles de routage dynamique, nous avons préféré abandonner cette approche. En effet, le protocole de routage dynamique sur un réseau WiFi est un sujet de recherche en cours loin d'être trivial. De plus les expérimentations pratiques effectuées par Tetaneutral.net ont montré une faiblesse de l'implémentation. Le protocole de routage B.A.T.M.A.N est fonctionnelle sur de petit réseau WiFi, mais à l'échelle d'une ville, ce protocole se révèle instable et non fonctionnel. Les communications induites par cette implémentation se révèlent trop importantes en bande passante par rapport aux données utiles. Enfin, l'association tetaneutral.net, qui utilisait du routage dynamique de niveau 3 sur son réseau souhaite, pour les raisons évoquées précédemment, l'abandonner au profit d'un réseau transparent au niveau 2. Il n'y a donc plus de raison de vouloir baser notre solution sur du routage dynamique.

B. Extension de l'outil linkagreg

La deuxième approche a été d'utiliser un programme, linkagreg, développé par Fernando Alves.

i. Fonctionnement de linkagreg

Linkagreg ouvre plusieurs connexions UDP, par l'intermédiaire d'un socket UDP, entre le client et le serveur. Ces différents sockets sont « bindés » sur chaque interface physique. Le tunnel est réalisé grâce à une interface virtuelle tun (ou tap), et cette interface virtuelle sert juste d'interface de communication avec le « kernel network ».

ii. Utilisation de linkagreg

Le serveur de linkagreg se trouve sur le serveur virtualisé chez tetaneutral.net. Le client possède par exemple, deux liaisons ADSL.

```
# Création de l'interface virtuelle
# Source : http://www.inetdoc.net/guides/vm/vm.network.tun-tap.html
apt-get install openvpn
openvpn --mktun --dev tun10

 ##############################
 # Sur le serveur eth0 = 192.168.1.1
 ##############################
 ./linkagreg -s -d -i tun10 &
 ip addr add 10.0.1.1/24 dev tun10
 ip link set up dev tun10

 ##############################
 # Sur le Client eth0 = 192.168.1.2 et eth1 = 192.168.1.3
 # ouverture de 2 tunnel UDP:
 # 192.168.1.2 <--> 192.168.1.1
 # 192.168.1.3 <--> 192.168.1.1
 ##############################
 ./linkagreg -c 192.168.1.1 -f 192.168.1.2,192.168.1.3 -d -i tun10 &
 ip addr add 10.0.1.2/24 dev tun10
 ip link set up dev tun10
```

iii. Les tests effectués

Plusieurs tests de linkagreg ont été effectués :

- Utilisation d'une seule interface sur le client : fonctionne
- Utilisation de deux interfaces réseaux distincts
  - Ne fonctionne pas, car la deuxième connexion utilisée était une connexion WiFi domestique avec trop de perte de paquets (Aux alentours de 20% de paquets perdus)
- Utilisation de plusieurs interfaces virtuelles
  - Ne fonctionne pas, car la QoS [13] ne peut pas s'appliquer sur une interface virtuelle

iv. Abandon de cette approche

Plusieurs facteurs ont contribués à l'abandon de cette approche. Le premier a été le manque de documentation (pas de site officiel, la documentation sous la forme du code source peu commenté). Le deuxième a été le langage utilisé dans le code source. En effet, ce projet a été développé en C, or aucun de nous ne maîtrisait suffisamment la programmation système et réseau en C pour être capable d'expérimenter sur cette base de code dans un temps raisonnable. Par conséquence, l'adaptation de ce projet pour avoir un algorithme d'ordonnancement plus fin aurait été difficile. Enfin, il est à noter que ce projet était à l'état expérimental à l'époque et que nous

aurions certainement dû commencer par devoir le débuguer avant de pouvoir y ajouter nos fonctionnalités.

Chapitre 5 : notre solution : scripts python pour l'agrégation

Cette approche s'articule autour de trois axes. Le premier axe est le tunnel entre deux points A et B (ici, le client qui agrège les liens et le serveur). Le deuxième axe est d'agréger les différentes connexions disponibles selon un algorithme d'ordonnancement. L'algorithme d'ordonnancement va dépendre des métriques des connexions. Ce point sera développé dans la partie suivante. Le troisième axe est de faire du routage prenant en compte la spécifité du tunnel et de l'agrégation

V. Notre solution : scripts python pour l'agrégation

La troisième approche a été d'utiliser plusieurs outils et scripts pour réaliser toutes les fonctionnalités attendues de l'agrégation.

Notre projet peut être téléchargé à l'adresse suivante :

```
git clone git://rhizome-fai.tetaneutral.net/agregation.git
```

A. Base python « multi.py »

Le script initial fait une trentaine de lignes et offrait le fonctionnement suivant :

- création d'interfaces virtuelles via openvpn
- distribution de la charge sur plusieurs tunnels UDP (en utilisant le même lien) par round-robbin ((principe du tourniquet, un paquet est envoyé sur la première interface, un sur la suivante… etc.))
- configuration en dur
- démarrage à la main

Le but de ce script écrit par Laurent Guerby était d'étudier la QoS[6] en place sur les connections ADSL d'opérateurs tiers : voir si dispatcher le trafic sur plusieurs tunnels UDP augmentait la bande passante par rapport à un seul, il était suspecté qu'une QoS par socket UDP était appliquée Cela constituait un point de départ autour duquel nous avons construit notre solution.

L'utilisation de ce script :

```
server# openvpn --dev tun --ifconfig 10.1.0.1 10.1.0.2  --auth none --cipher none --
port 65404 --local127.0.0.1
server# python multi.py -s

client# python multi.py -c
client# openvpn --dev tun --ifconfig 10.1.0.2 10.1.0.1 --auth none --cipher none --
remote localhost 65404 --keepalive 10 30
```

Voici le schéma de fonctionnement du script :

[6] Qualité de service, règles pouvant-être appliquées à du trafic réseau par exemple pour prioriser certains services ou limiter certains autres

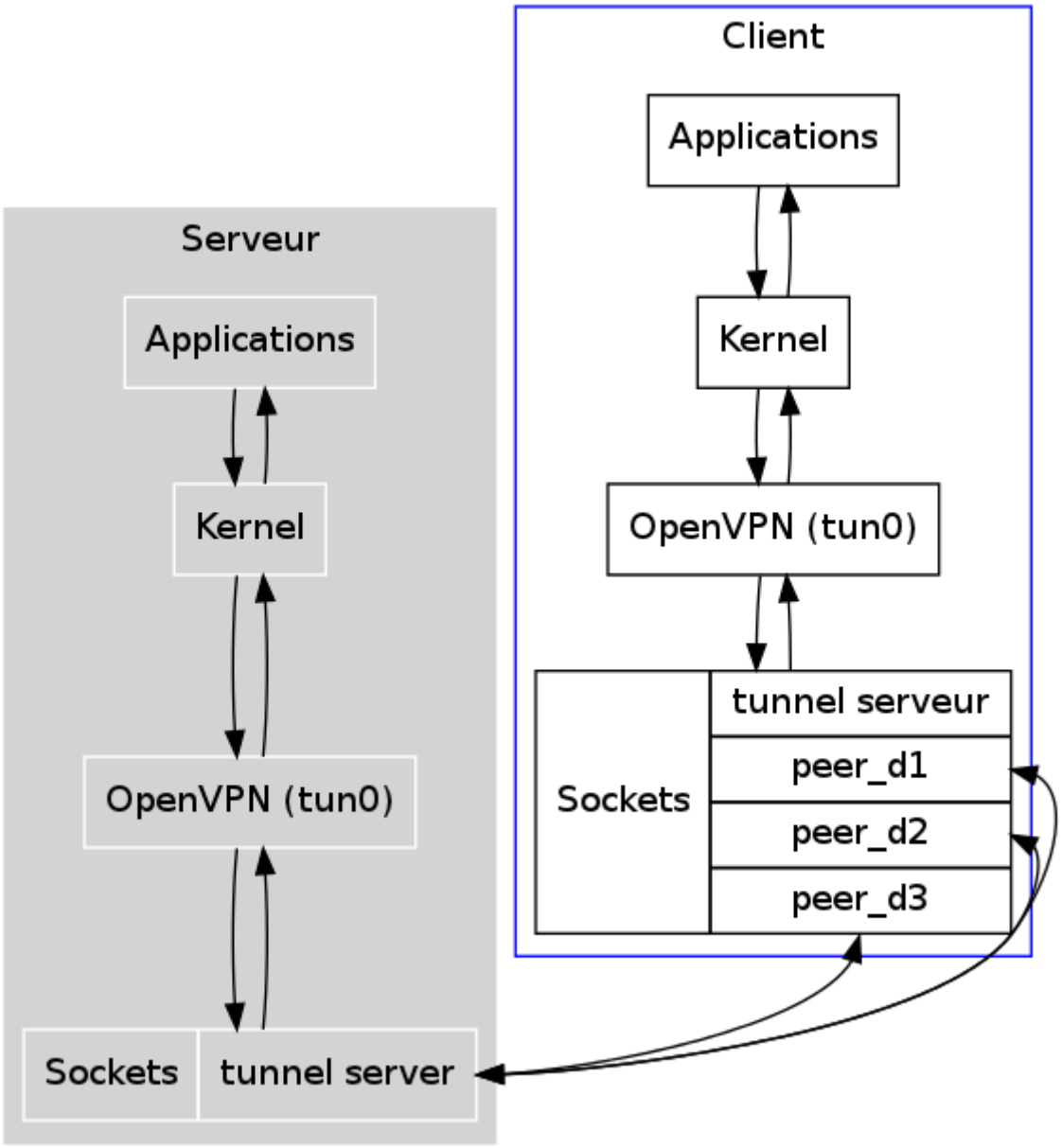


Figure 14 : Schéma de fonctionnement du script multi.py

B. Objectif redéfini

Là où notre but initial était de faire une solution fonctionnelle basique, nous nous sommes rendus compte que des solutions de ce genre existaient déjà, à des stades plus ou moins poussés d'avancement (Nous avons déjà parlé de linkagreg dans les approches utilisées, on peut aussi citer agrego [14] ou MLVPN [15]). Aucune ne semble cependant pour l'instant se détacher du lot et être massivement utilisée. Développer une solution basique supplémentaire du même type que les autres (en C, permettant de faire de la répartition de charge pondérée statique) nous est apparu inutile et long. Nous avons préféré nous orienter vers une solution certes fonctionnelle mais surtout permettant l'expérimentation aisée, quitte à y sacrifier la consommation de ressources.

En l'occurrence, aucune des solutions actuelle ne permet, à notre connaissance, l'ajustement dynamique de la pondération de chaque lien. Or, sur des connexions xDSL, les débits varient, par exemple selon le moment de la journée[7]. Nous nous sommes donc focalisés sur ce point en espérant, si nos résultats sont intéressants, qu'ils puissent-être utilisés dans d'autres projets.

C. Fonctionnalités

Notre solution présente donc les fonctionnalités suivantes:

- dispatchage sur plusieurs connections (1 socket/connexion) ;
- détection de la capacité des liens et de la variation opportuniste (via surveillance de la saturation par demi-délai) ;
- dispatchage du traffic réseau par sélection aléatoire pondérée (par les capacités des liens) entre les connexions ;
- fichier de configuration ;
- script d'initialisation pour gérer tous les processus locaux, leur démarrage, leur synchronisation et leur arrêt.
- Routage par l'IP source (évite d'avoir à définir plusieurs IP publiques côté serveur).
- Peut fonctionner derrière un NAT[8].

D. Architecture

Le script est décomposé en plusieurs processus qui communiquent entre eux via des sockets locaux. Pourquoi ce fonctionnement plutôt qu'un seul programme ? Une modularité telle est intéressante puisque le script python n'est peut-être pas le plus adapté à terme pour ce qui est des performances. Ainsi, il sera possible par exemple de passer multi.py (qui est la partie critique en performances/consommation de ressources) en C tout en laissant le reste en Python.

En outre, le fait d'utiliser le processus OpenVPN pour gérer l'interface virtuelle réduit la quantité de code à écrire… Et surtout permet d'utiliser si besoin l'authentification, le chiffrement et le mécanisme de keepalive (qui permet la reconnexion en cas d'inactivité suspecte détectée) inclus dans OpenVPN sans ajouter de code. OpenVPN est une solution robuste et portée sur de nombreuses plateformes, y-compris embarquées (ex: OpenWRT).

[7] Nous avons pu observer au cours de nos expériences des variations de 400kbps à 700kbps de lien descendant, le lien montant fluctuant moins en général

[8] Network Address Translation, fait de cacher plusieurs machines possédant des adresses privées [RFC 1918] derrière une unique adresse publique

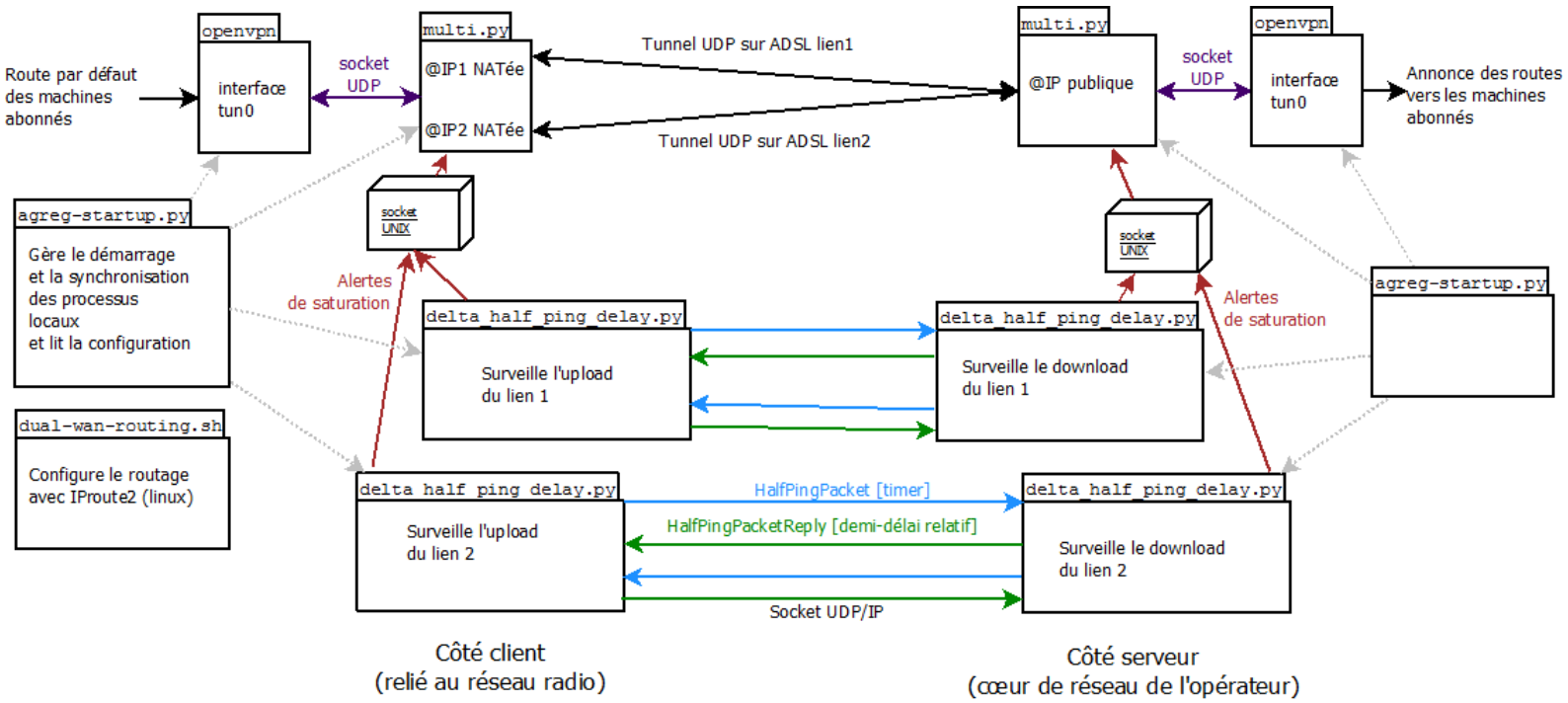


Figure 15 : Le schéma du fonctionnement de l'agrégation

E. Routage

Pour que le script multy.py présenté juste avant fonctionne, il faut faire du routage selon la source. Par exemple, l'outil iproute2 permet de le faire. [16] [17]. En effet, il n'y a qu'une IP de serveur, mais N IPs de client. Or les décisions de routage d'une table gérée avec la commande route sont de la forme suivante (simplifiée) :

```
<Destination> <passerelle> <interface>
```

On constate que le routage est fait seulement en fonction de la destination. On utilise donc l'outil *ip route*. (Voir en annexe son utilisation et sa mise en œuvre).

F. Détection de saturation du lien

La détection de saturation de capacité du lien est faite par le script delta_half_trip_time.py par surveillance de l'évolution du demi-délai. Un processus de ce type est lancé par lien à surveiller. Des messages de notification de saturation sont émis à destination de multi.py via un socket UNIX. La formule permettant de détecter la saturation d'un lien est détaillée dans l'étude expérimentale de la saturation des liens ADSL.

G. Répartition de charge pondérée

Notre programme prend pour chaque paquet la décision du lien par lequel l'envoyer. Pour mémoire, dans le script tunnel.py, un simple round-robin était utilisé. Nous souhaitons faire un peu mieux… Nous voulons envoyer le nombre juste de paquets sur chaque interface en fonction de sa capacité.

multi.py conserve l'historique du volume de données envoyées via chaque lien toutes les secondes. Cela permet de garder un historique des débits.

Lorsque multi.py reçoit une notification de saturation de la part de delta_half_trip_time.py, il essaye de retrouver le pic de débit pour le lien concerné dans son historique récent (10 dernières secondes). Cette valeur de pic constitue alors la pondération du lien. Ainsi, les pondérations de chaque lien sont ajustées à chaque fois que l'un d'entre eux est saturé.

i. Comment initialiser les pondérations ?

On pourrait initialiser les pondérations dans la configuration manuellement ou bien à des valeurs moyennes. Cependant, nous voulons si possible éviter un paramètre inutile dans la configuration et surtout, nous voulons détecter au plus vite les valeurs réelles. Garder longtemps des valeurs fausses pouvant réduire les performances et des pondérations trop basses pouvant conduire à ne jamais détecter de saturation sur un des liens.

Aussi, nous avons choisi d'initialiser les pondérations des n liens de la manière suivante :

```
p = 100 000 000
pour n de 1 à nombre_de_liens
   pondération_n <- p
   p <- p*1000
```

La valeur initiale de p est bien trop élevée (correspond à (100Mbps)), le but est d'être certains qu'elle sera réduite rapidement.

Le point crucial est d'introduire une grosse dissymétrie (chaque pondération étant 1000 fois supérieure à la précédente). Ainsi, au lancement du programme, par exemple pour deux liens on aura :

- pondération_1 = 10^8
- pondération_2 = 10^11

1. Seul le lien 2 est utilisé (à peu de choses près).
2. Le lien 2 est rapidement saturé.
3. La saturation est détectée et la pondération du lien 2 est ajustée à sa valeur réelle.
4. le lien 1 devient donc largement le maximum et c'est à son tour d'être utilisé exclusivement, donc saturé, détecté et ajusté.

Après un court temps d'utilisation, nous avons donc saturé nos deux liens et initialisé correctement leurs valeurs.

Exemple :

```
New weights are [100000000, 100000000]
 received report : SAT TX OVH_ADSL, bw was: 1181kb/s
 New weights are [100000000, 151200]
 received report : SAT TX FDN_ADSL, bw was: 823kb/s
 New weights are [105352, 151200]
...
```

H. Mesures de performances

i. Round-trip-time (ping)

Un simple test de ping nous donne les résultats suivants de ping moyen sur 20 échantillons (cf résultats complets en annexe):

- Sans tunnel (lien OVH) : 60.815ms
- Sans tunnel (lien FDN) : 58.671ms
- Dans le tunnel : 62.325ms

On ajoute très peu de temps de parcours (4%). C'est une valeur acceptable qui ne sera pas ressentie par l'utilisateur.

ii. Débit

Les mesures de débit ont été prises pour l'upload et le download séparément, en utilisant un simple round-robin puis en utilisant une répartition de charge aléatoire pondérée avec ajustement sur saturation.

Les tests sont utilisés avec la commande "iperf -P7 -c <host>" (7 sessions TCP en parallèle) On ne considère ici que les médianes des valeurs des séries de tests. Les résultats complets sont disponibles en annexe.

OVH correspond à la ligne OVH, sans tunnel
FDN correspond à la ligne ADSL FDN, sans tunnel, via le réseau radio 5GHz
TUN(RR) correspond au tunnel avec une round-robbin sans pondération
TUN(BAL) correspond au tunnel avec répartition pondérée par les pics de débit en saturation.

a. Upload

- OVH: 855kbps
- FDN: 711kbps
- OVH+FDN: 1,53Mbps
- TUN(RR): 1,23Mbps
- TUN(BAL) : 1,33Mbps

On note un usage du processeur de 3% à 5% durant ces tests de charge avec le tunnel[9].

On constate également sur ces tests (cf annexes) que nous ne déclenchons d'alerte de saturation que sur un des deux liens que nous utilisons (le plus petit en capacité). Logiquement, on sature en revanche les deux liens en utilisant la pondération de lien avec ajustement sur saturation.

Nous avons donc une perte de performance par rapport à la performance agrégée idéale de -19% avec le tunnel en round-robbin et -12% avec le tunnel avec répartition pondérée. N'oublions que parmis cette perte est inclue la perte « inévitable » liée à l'overhead de 3%.

On peut dire que ces résultats sont satisfaisants, avec la sélection pondérée et intéressants pour l'opérateur.

b. Download

- OVH: 6,93Mbps
- FDN: 7,11Mbps
- OVH+FDN: 14,04Mbps
- TUN(RR): 8,90Mbps
- TUN(BAL):10,3Mbps

On note un usage du processeur de 10% à 20% durant ces tests de charge avec le tunnel.

Les résultats sont beaucoup plus décevants qu'en upload : -27% de performances avec la répartition pondérée et -36% avec le round-robin. Ils peuvent cependant rester intéressants mais demandent d'investiguer : il doit être possible de faire mieux. La forte consommation de CPU nous laisse croire qu'il peut y avoir un goulot d'étranglement à ce niveau.

I. Perspectives

i. Fonctionnalités

Notre programme a été une bonne base pour mener des tests et expérimenter différentes formes de load-balancing. Un programme en C ne nous aurait pas permis cette souplesse d'expérimentation.

Il serait intéressant d'y ajouter les fonctionnalités suivantes :

- Détection de la coupure/rétablissement d'un lien (aisé à implémenter).
- Prise en compte du taux de paquets perdus (loss-rate) en plus du demi-délai pour détecter les saturations.
- Possibilité d'utiliser plusieurs tunnels par lien (pour contourner une QoS par ex, cf A.).
- Chiffrement et authentification[10].
- Meilleure gestion des faux-positifs de saturation

[9] Core 2 Duo CPU T7300 @ 2.00GHz

[10] Ces fonctionnalités sont en fait fournies "gratuitement" par l'usage d'OpenVPN que nous faisons, il y aurait juste à les rajouter dans la configuration

Concernant le dernier point, notre algorithme souffre d'une faiblesse. Dans le cas où, malgré notre formule "prudente" de détection de saturation (cf III.D), on détecte un faux-positif (ndlr: on considère que la bande passante actuelle est une saturation alors que ça n'est pas le cas), on va valuer trop faiblement un lien. Par conséquent, il sera moins utilisé que les autres… Et il est peu probable qu'il devienne saturé à nouveau et qu'on détecte sa valeur max. réelle. Le lien est alors bloqué à une valeur trop faible.

Il faudrait prévoir des mécanismes correctifs pour sortir de cette impasse.

ii. Performances

Clairement, lorsqu'on sature le lien descendant, la consommation CPU est trop importante[11], et il est possible que le goulot d'étranglement soit pour partie le traitement CPU.

Le point qui nous fait penser que le traitement CPU est en cause est le fait que les débits d'upload (10x inférieurs) sont biens gérés par l'agrégateur alors que les débits descendants possèdent des rendements largement inférieurs.

Il est avéré que les solutions de type OpenVPN (sans agrégation) peuvent monter à des débits proches du Gigabit [18].

Optimiser les traitements effectués dans multi.py, et éventuellement ré-écrire des parties de ce programme en C permettraient éventuellement de gagner en performances.

Cependant, il n'est pas certain que la consommation CPU soit la seule cause des performances relativement faibles en download. On peut penser également que le fait que les paquets d'une même session TCP arrivent via deux canaux différents demande à TCP de réordonner ses paquets systématiquement. Il serait intéressant d'observer le nombre de paquets désordonnés qui arrivent, avec et sans le tunnel.

L'utilisation d'un algorithme de sélection aléatoire (cf V.G) peut causer des dés-ordonnancement de paquets également, il serait intéressant de tester avec un algorithme à base de jetons qui garantiraient une équité « parfaite » localement.

Des tests de débit en UDP pour comparer permettraient également de progresser sur ce point.

Enfin, certains travaux montrent qu'il peut être judicieux d'augmenter largement la taille du MTU sur les interfaces de tunnel. Cela occasionne moins de paquets et donc moins de traitements en espace utilisateur [18], un traitement étant déclenché par paquet.

[11] entre 10 et 20% du CPU sur un Core 2 Duo CPU T7300 @ 2.00GHz

Définitions

Le phénomène de bufferbloat est lié à une saturation de buffers. Le chemin entre deux nœuds sur un réseau passe par plusieurs routeurs, que ça soient des routeurs domestiques (ex: box) ou des routeurs d'opérateurs. Lorsqu'on possède deux liens de capacité différente (ex: réseau local puis sortie vers internet via ADSL), les données sont mises en buffer sur le routeur, qui les accepte jusqu'à ce que son buffer soit plein (il ne peut pas les écouler aussi vite vers internet qu'il ne les reçoit sur le LAN). Une fois que le buffer est plein, le routeur va rejetter une partie des paquets arrivant sur son interface LAN. Du point de vue de l'émetteur d'une grande quantité de données, le débit vers internet va être d'abord très rapide puis ralenti subitement à la saturation. Tous les autres clients du LAN vont voir une partie de leurs paquets rejettés et le round-trip-time vers internet augmenter. Paradoxalement, plus le constructeur du buffer prévoit un buffer important, plus le problème de bufferbloat se fera ressentir. Le bufferbloat et les manières de l'éviter sont des sujets de recherche actifs en réseau. 11

VI. Bibliographie

[1] «Qu'est ce que le MLPP,» [En ligne]. Available: http://wapiti.telecom-lille1.eu/commun/ens/peda/options/ST/RIO/pub/exposes/exposesrio2003ttnfa04/garcia-preuilh/html%5Cd%C3%A9finition.htm.

[2] «802.1AX,» [En ligne]. Available: http://en.wikipedia.org/wiki/802.1AX-2008.

[3] Limitation de MLPPP, [En ligne]. Available: http://searchtelecom.techtarget.com/definition/multilink-PPP.

[4] «MC-MLPPP Emulation using Client-Server,» [En ligne]. Available: http://www.gl.com/mlppptxrxinwcs.html.

[5] «Linux Channel Bonding,» [En ligne]. Available: http://sourceforge.net/projects/bonding/files/Documentation/12%20November%202007/bonding.txt/download?use_mirror=heanet.

[6] «Comparaison de l'efficacité des différentes méthodes,» [En ligne]. Available: http://www.thlab.net/~thsalon/papers/wons09.pdf.

[7] «Article pour déterminer avec le SNR la capacité d'un lien,» [En ligne]. Available: http://citeseer.ist.psu.edu/viewdoc/download;jsessionid=D8D74970F81D4957F250F4357BB6CCC9?doi=10.1.1.15.5673&rep=rep1&type=pdf.

[8] «NTP,» [En ligne]. Available: http://www.frameip.com/ntp/.

[9] «Etude sur NTP,» [En ligne]. Available: http://www.eecis.udel.edu/~mills/database/brief/perf/perf.pdf.

[10] «Etude du protocole UTP,» [En ligne]. Available: http://www.rasterbar.com/products/libtorrent/utp.html.

[11] «B.A.T.M.A.N.,» [En ligne]. Available: http://fr.wikipedia.org/wiki/B.A.T.M.A.N..

[12] «B.A.T.M.A.N,» [En ligne]. Available: http://www.open-mesh.org/.

«QoS,» [En ligne]. Available: http://blog.nicolargo.com/2009/03/simuler-un-lien-wan-sous-linux.html.
[13]

«The AgreGo project,» [En ligne]. Available: http://code.google.com/p/agrego/.
[14]

«Github du projet MLVPN,» [En ligne]. Available: https://github.com/zehome/MLVPN.
[15]

«Guide IPRoute 2,» [En ligne]. Available: http://www.inetdoc.net/guides/lartc/lartc.iproute2.html.
[16]

«Notion avancée de routage,» [En ligne]. Available: http://www.rjsystems.nl/en/2100-adv-routing.php.
[17]

M. MOEKSTRA, «Comparing TCP performance of tunneled and non-tunneled traffic using OpenVPN,» *http://staff.science.uva.nl/~delaat/rp/2010-2011/p09/report.pdf,* 2011.
[18]

«Rendre un dépot public,» [En ligne]. Available: http://doc.ubuntu-fr.org/gitolite#rendre_un_depot_public.
[19]

«Configuration de git-daemon,» [En ligne]. Available: http://computercamp.cdwilson.us/git-gitolite-git-daemon-gitweb-setup-on-ubunt.
[20]

«Agrego,» [En ligne]. Available: http://code.google.com/p/agrego/.
[21]

VII. Table des illustrations

Annexe

I. Les outils utilisés

A. Gitolite

Pour le développement des outils et des scripts utilisés, le gestionnaire de version GIT a été utilisé. Pour gérer et rendre public ce gestionnaire de version, gitolite et git-daemon ont été installés sur un serveur prêté par tetaneutral.net.

i. Installation du serveur gitolite

```
#La clé de l'admin
client$ .ssh/id_rsa.pub root@rhizome-fai.net:/tmp/ydelarbr.pub
server# apt-get install gitolite -y
server# su - gitolite
#Initialisation du repository admin
gitolite@server$ gl-setup /tmp/ydelarbr.pub
```

ii. Configuration de gitolite

```
client$ git clone gitolite@rhizome-fai.tetaneutral.net:gitolite-admin.git
client$ cd gitolite-admin
#Les clés des clients
client$ mv cle.pub key/
#La configuration de chaque repository
client$ vim conf/gitolite.conf
        repo    gitolite-admin
                RW+     =   user
        repo    testing
                RW+     =   @all
        repo    agregation
                RW+     = user ydelarbr jocelyn fernando
                R       = daemon
client$ git add conf/gitolite.conf key/*
client$ git commit -m "add repo agregation + git-daemon in R"
#Note: toutes la configuration s'applique grâce à un hook. Donc il suffit de "pusher" la
configuration vers le serveur
client$ git push
```

iii. Rendre le repository public

```
server# apt-get install git-daemon-run -y
server# vim /etc/init.d/git-daemon
# Voir la conf sur les sources

# Automatiser le service
server# update-rc.d git-daemon default
```

iv. Cloner le repository

```
# Le repository en lecture seule (public)
 client_git_public$ git clone git://rhizome-fai.tetaneutral.net/agregation.git

# Le repository en lecture écriture pour un utilisateur dont la clé a été déjà ajouté
 client_git_user$ git clone gitolite@rhizome-fai.tetaneutral.net:agregation.git
```

v. Sources

[19] [20]

B. Linkagreg : troubleshooting

```
./linkagreg
bash: ./linkagreg: Aucun fichier ou dossier de ce type
#Il faut recompiler le projet ! Ou avoir les bonnes librairies.

#Des aides pour déterminer le problème
##Pour voir si il y a des problèmes de caractères
ls -lb linkagreg

##Plus d'infos sur le fichier
file linkagreg

##Les librairies nécessaires (dynamique
ldd linkagreg

##Voir la trace de l'exécution
strace linkagreg

##Débugger une application C
gdb linkagreg

##Voir l'en-tête:
apt-get install binutils
readelf -a linkagreg | grep Requesting
# [Requesting program interpreter: /lib/ld-linux.so.2]

ls -l /lib/ld-linux.so.2
# lrwxrwxrwx 1 root root 20 2011-04-28 17:32 /lib/ld-linux.so.2 -> /lib32/ld-linux.so.2

##La solution:
apt-get install libc6-dev-i386
```

C. Quelques outils réseaux

```
#tcpdump | http://openmaniak.com/fr/tcpdump.php
tcpdump -D #Interfaces réseaux disponibles pour la capture
tcpdump port 80 -i eth0 -w capture.log #Enregistre le trafic Web vers le fichier capture.log pouvant être ouvert avec Wireshark
tcpdump icmp #Affiche tout le trafic associé au protocole icmp

#ping | http://www.bortzmeyer.org/ping-taille-compte.html
ping -s 1600 192.168.0.1 # Permet de tester un problème de MTU grâce à l'option -s de ping permettant de fixer une taille de paquet

#hping3 | Un ping évolué
hping --syn -p 80 --data 1200 10.0.0.1 #Envoie de paquet tcp syn sur le port 80 de taille 1200
```

i. Utilisation de l'outil ip route

```
#Montrer la table de routage:
ip route show

#Montrer la route pour une IP source et dest données :
ip route get 0.0.0.0 from 192.168.1.71

# Initialement, on a deux interfaces (eth0 et wlan0) avec une passerelle vers internet sur chaque (2 lignes ADSL), on est NATé sur les deux. Mais une seule est déclarée comme route par défaut.
jocelyn@sensitive:~$ ip route get 8.8.8.8 from 192.168.2.33
8.8.8.8 from 192.168.2.33 via 192.168.1.254 dev wlan0
    cache
jocelyn@sensitive:~$ ip route get 8.8.8.8 from 192.168.1.71
8.8.8.8 from 192.168.1.71 via 192.168.1.254 dev wlan0
    cache
 s
# Par défaut, on n'a que la table « main » et « default ». (on peut voir les tables avec ip rule).
# Nos deux ip locales sont 192.168.1.71 (wlan0) et 192.168.2.33 (eth0)
# On y ajoute notre table : fdn_rhizome avec nos règles :
# Ajout d'une nouvelle table
echo "1000 rhizome_fdn" >> /etc/iproute2/rt_tables
# et sa route par défaut
ip route add default via 192.168.2.1 dev eth0 table rhizome_fdn
# On dit au système de ne regarder notre table que pour les requêtes venant de 192.168.2.33
ip rule add from 192.168.2.33 lookup rhizome_fdn prio 1000

# On vérifie qu'on passe par une interface différente en fonction de l'IP source :
jocelyn@sensitive:~$ ip route get 8.8.8.8 from 192.168.2.33
    8.8.8.8 from 192.168.2.33 via 192.168.2.1 dev eth0
    8.8.8.8 from 192.168.1.71 via 192.168.1.254 dev wlan0

# Pour un test pratique, on peut utiliser ping avec l'option -I pour spécifier l'interface de sortie, puis vérifier dans wireshark que les trames sortent bien par une interface différente (header MAC, il suffit de regarder l'addr. MAC source).
```

D. Le script de mesure par délai relatif

Un outil permettant de mesurer l'évolution du délai relatif a été développé. Voici les commandes de son utilisation :

```
# Côté serveur :
./delta_half_trip_time.py -s 2244

# Côté client:
./delta_half_trip_time.py -s <ip_serv>:2244

# Le script mesure en permanence les délais toutes les secondes. Il ne prend pas en compte la dérive d'horloge pour l'heure. La sortie est du CSV contenant les délais dans les deux sens (de chaque côté). Le format est :
## pkt_type,sequence number,delay ##
# pkt_type vaut 't' (comme timer) pour les mesures entrantes (download) et 'd' (comme delay) pour les réponses aux paquets sortants (upload).
```

II. Résultats des mesures

A. Tests de deux liens agrégés : première version

```
[  4] local 91.224.149.199 port 5001 connected with 80.67.177.5 port 5001
[  4]  0.0-123.9 sec  10.3 MBytes    697 Kbits/sec
[  5] local 91.224.149.199 port 5001 connected with 109.190.12.102 port 64916
[  5]  0.0-122.1 sec  10.9 MBytes    750 Kbits/sec
[  4] local 10.1.0.2 port 5001 connected with 10.1.0.1 port 56190
[  4]  0.0-120.5 sec  14.5 MBytes  1.01 Mbits/sec10

jocelyn@rover:~/divers/agregation$ wget http://10.1.0.2/bigfile.bin
--2012-01-28 22:16:28--  http://10.1.0.2/bigfile.bin
Connexion vers 10.1.0.2:80...connecté.
requête HTTP transmise, en attente de la réponse...200 OK
Longueur: 41984000 (40M) [application/octet-stream]
Sauvegarde en : «bigfile.bin»

100%[================================================================================
===>] 41 984000   335K/s   ds 92s

2012-01-28 22:18:00 (447 KB/s) - «bigfile.bin» sauvegardé [41984000/41984000]



jocelyn@rover:~/divers/agregation$ wget --bind-address=192.168.2.107 http://rhizome-
fai.tetaneutral.net/bigfile.bin
--2012-01-28 22:19:34--  http://rhizome-fai.tetaneutral.net/bigfile.bin
Résolution de rhizome-fai.tetaneutral.net... 91.224.149.199
Connexion vers rhizome-fai.tetaneutral.net|91.224.149.199|:80...connecté.
requête HTTP transmise, en attente de la réponse...200 OK
Longueur: 41984000 (40M) [application/octet-stream]
Sauvegarde en : «bigfile.bin.1»

100%[================================================================================
===>] 41 984000   198K/s   ds 2m 2s

2012-01-28 22:21:35 (337 KB/s) - «bigfile.bin.1» sauvegardé [41984000/41984000]


jocelyn@rover:~/divers/agregation$ wget --bind-address=192.168.1.112 http://rhizome-
fai.tetaneutral.net/bigfile.bin
--2012-01-28 22:22:24--  http://rhizome-fai.tetaneutral.net/bigfile.bin
Résolution de rhizome-fai.tetaneutral.net... 91.224.149.199
Connexion vers rhizome-fai.tetaneutral.net|91.224.149.199|:80...connecté.
requête HTTP transmise, en attente de la réponse...200 OK
Longueur: 41984000 (40M) [application/octet-stream]
Sauvegarde en : «bigfile.bin.2»

100%[================================================================================
===>] 41 984000   839K/s   ds 50s

2012-01-28 22:23:14 (818 KB/s) - «bigfile.bin.2» sauvegardé [41984000/41984000]


CONSO CPU ENTRE 3 et 10% (MOYENNE=5%)

jocelyn@rover:~$ ping -c20 10.1.0.2
PING 10.1.0.2 (10.1.0.2) 56(84) bytes of data.
64 bytes from 10.1.0.2: icmp_req=1 ttl=64 time=59.3 ms
```

```
…

--- 10.1.0.2 ping statistics ---
20 packets transmitted, 20 received, 0% packet loss, time 19021ms
rtt min/avg/max/mdev = 58.559/62.325/72.464/4.513 ms

jocelyn@rover:~$ ping -Ieth0  91.224.149.199 -c20
PING 91.224.149.199 (91.224.149.199) from 192.168.2.107 eth0: 56(84) bytes of data.

--- 91.224.149.199 ping statistics ---
20 packets transmitted, 20 received, 0% packet loss, time 19029ms
rtt min/avg/max/mdev = 57.694/58.671/60.225/0.682 ms
jocelyn@rover:~$

jocelyn@rover:~$ ping -Ieth0.1  91.224.149.199 -c20
PING 91.224.149.199 (91.224.149.199) from 192.168.2.107 eth0.1: 56(84) bytes of data.

--- 91.224.149.199 ping statistics ---
20 packets transmitted, 20 received, 0% packet loss, time 19026ms
rtt min/avg/max/mdev = 54.433/60.815/81.957/5.692 ms


#Réseau radio Client->modem
jocelyn@rover:~$ ping -c20 10.42.1.1
PING 10.42.1.1 (10.42.1.1) 56(84) bytes of data.

--- 10.42.1.1 ping statistics ---
20 packets transmitted, 20 received, 0% packet loss, time 19027ms
rtt min/avg/max/mdev = 1.379/1.974/5.669/0.892 ms
```

B. Avec un seul lien agrégé

```
jocelyn@rover:~/divers/agregation$ iperf  -c 10.1.0.2
------------------------------------------------------------
Client connecting to 10.1.0.2, TCP port 5001
TCP window size: 16.0 KByte (default)
------------------------------------------------------------
[  3] local 10.1.0.1 port 56220 connected with 10.1.0.2 port 5001
[ ID] Interval       Transfer     Bandwidth
[  3]  0.0-10.5 sec  1.14 MBytes    911 Kbits/sec

jocelyn@rover:~/divers/agregation$ iperf  -B 192.168.1.112 -c  91.224.149.199
------------------------------------------------------------
Client connecting to 91.224.149.199, TCP port 5001
Binding to local address 192.168.1.112
TCP window size: 16.0 KByte (default)
------------------------------------------------------------
[  3] local 192.168.1.112 port 5001 connected with 91.224.149.199 port 5001
[ ID] Interval       Transfer     Bandwidth
[  3]  0.0-11.4 sec  1.28 MBytes    946 Kbits/sec

jocelyn@rover:~/divers/agregation$ wget http://10.1.0.2/bigfile.bin
--2012-01-28 22:59:35--  http://10.1.0.2/bigfile.bin
Connexion vers 10.1.0.2:80...connecté.
requête HTTP transmise, en attente de la réponse...200 OK
Longueur: 41984000 (40M) [application/octet-stream]
Sauvegarde en : «bigfile.bin»

100%[======================================>] 41 984 000   816K/s   ds 52s

2012-01-28 23:00:27 (782 KB/s) - «bigfile.bin» sauvegardé [41984000/41984000]

jocelyn@rover:~/divers/agregation$ wget --bind-address=192.168.1.112 http://rhizome-
fai.tetaneutral.net/bigfile.bin
--2012-01-28 23:00:49--  http://rhizome-fai.tetaneutral.net/bigfile.bin
Résolution de rhizome-fai.tetaneutral.net... 91.224.149.199
Connexion vers rhizome-fai.tetaneutral.net|91.224.149.199|:80...connecté.
requête HTTP transmise, en attente de la réponse...200 OK
Longueur: 41984000 (40M) [application/octet-stream]
Sauvegarde en : «bigfile.bin.1»

100%[======================================>] 41 984 000   837K/s   ds 50s

2012-01-28 23:01:39 (820 KB/s) - «bigfile.bin.1» sauvegardé [41984000/41984000]
```

C. Estimation de l'overhead

Téléchargement d'un fichier de 42Mio 48206 paquets/45 132 020 octets :

- en tap : $ip + udp + mac = 20 + 8 + 14 = 42$
- en tun : $ip + udp = 20 + 8 = 28$

Overhead théorique :

- tun : $48206 * 28 = 1349768$ (3,0%)
- tap : $48206 * 42 = 2024652$ (4,4%)

D. Comparaison par rapport à la perte de débit

Sans modification de la MTU (tun0 mtu=1500) :

- sans tunnel : 820kB/s
- avec tunnel : 782kB/s

On a donc une perte de débit de : 4.5% (en tun donc 1,5% de perte de performance liée au traitement).

i. Modification de la MTU tun0 avec une mtu=1400

```
jocelyn@rover:~/divers/agregation$ wget --bind-address=192.168.1.112 http://rhizome-
fai.tetaneutral.net/bigfile.bin
--2012-01-28 23:42:19--  http://rhizome-fai.tetaneutral.net/bigfile.bin
Résolution de rhizome-fai.tetaneutral.net... 91.224.149.199
Connexion vers rhizome-fai.tetaneutral.net|91.224.149.199|:80...connecté.
requête HTTP transmise, en attente de la réponse...200 OK
Longueur: 41984000 (40M) [application/octet-stream]
Sauvegarde en : «bigfile.bin.9»

100%[======================================>] 41 984 000   838K/s   ds 50s

2012-01-28 23:43:09 (815 KB/s) - «bigfile.bin.9» sauvegardé [41984000/41984000]

jocelyn@rover:~/divers/agregation$ wget http://10.1.0.2/bigfile.bin
--2012-01-28 23:43:15--  http://10.1.0.2/bigfile.bin
Connexion vers 10.1.0.2:80...connecté.
requête HTTP transmise, en attente de la réponse...200 OK
Longueur: 41984000 (40M) [application/octet-stream]
Sauvegarde en : «bigfile.bin.10»

100%[======================================>] 41 984 000   675K/s   ds 51s

2012-01-28 23:44:06 (798 KB/s) - «bigfile.bin.10» sauvegardé [41984000/41984000]
```

On passe de 815kB/s -> 798kB/s (-2%) Ce qui est étrangement en-dessous de la valeur théorique (3%). (conditions de test ?) En tout cas, l'ajustement de la MTU permet de gagne un peu en évitant la présence d'en-têtes inutiles.

L'ajustement optimal de la MTU du TUN correspond au minimum du PMTU sur les deux interfaces moins l'overhead. Ici : $min(1492,1400) - 42 \approx 1350$.

E. Influence de la MTU en download avec de l'agrégation

En kB/s : $FDN/OVH/TUN(MTU = 1500)/TUN(MTU = 1350)$

Avec la modification de la MTU (1350) : 826/820/818/863

Moyenne de 3 mesures (valeurs *très* fluctuantes, à prendre avec des pincettes *mais* on voit que la MTU apporte une très légère amélioration mais n'est pas le seul problème…

F. Problème de mesure

Solution partielle : iperf -c -P7

De cette manière, on sature systématiquement l'un de nos deux liens.

En upload: $OVH/FDN/TUN = 721/700/1300$

Ce qui conduit à : -8.5% (ndlr, on considère -3% d'overhead inévitable), on a donc une perte de -5.5%, ce qui est acceptable !

En download (*: médiane):

- OVH: 6,94*/6,93/6,83/8,42/6,89
- FDN: 8,37/8,03/8,44/8,02/8,11*
- TUN: 11,9/12/9,49/11*/10,9
- OVH+FDN = 15.05

Perte tunnel : 21% (- overhead : -18%)

G. Round-robbin intelligent

i. En upload

Avant RR intelligente : (upload, Mbps)

- 1,28/1,18/1,22/1,17/1,36

received report : SAT TX OVH_ADSL, bw was: 1139kb/s
received report : SAT TX FDN_ADSL, bw was: 1076kb/s
received report : SAT TX FDN_ADSL, bw was: 896kb/s
received report : SAT TX OVH_ADSL, bw was: 991kb/s
received report : SAT TX FDN_ADSL, bw was: 812kb/s
received report : SAT TX FDN_ADSL, bw was: 875kb/s
received report : SAT TX FDN_ADSL, bw was: 896kb/s

=> Le lien FDN est beaucoup plus saturé que le lien OVH qu'on exploite pas à fond.

Avec la répartition intelligente:

received report : SAT TX OVH_ADSL, bw was: 1181kb/s
New weights are [108000, 151200]
received report : SAT TX FDN_ADSL, bw was: 823kb/s
New weights are [105352, 151200]
received report : SAT TX OVH_ADSL, bw was: 1181kb/s
New weights are [105352, 151200]
received report : SAT TX FDN_ADSL, bw was: 823kb/s
New weights are [105352, 151200]
received report : SAT TX OVH_ADSL, bw was: 1181kb/s
New weights are [105352, 151200]
received report : SAT TX OVH_ADSL, bw was: 1023kb/s
New weights are [105352, 131002]
received report : SAT TX OVH_ADSL, bw was: 1023kb/s
New weights are [105352, 131002]
received report : SAT TX OVH_ADSL, bw was: 917kb/s
New weights are [105352, 117450]
received report : SAT TX FDN_ADSL, bw was: 864kb/s
New weights are [110700, 117450]
received report : SAT TX OVH_ADSL, bw was: 917kb/s
New weights are [110700, 117450]
received report : SAT TX FDN_ADSL, bw was: 864kb/s
New weights are [110700, 117450]

On sature les deux liens

Résultats comparatifs

- OVH: 845/855*/836/894/862
- FDN: 719/711*/715/645/710
- TUN (RR 1-1): 1,07/1,21/1,30,1,23*/1,27
- TUN (détection) : 1,41/1,33*/1,24/1,35/1,28

Somme des médianes OVH+FDN: 1,53

TUN (RR) : -19% (dont 3% d'overhead)
TUN (RR) : -12% (dont 3% d'overhead)

ii. En download

```
iperf -c <ip> -P10
```

En Mbps, de goodput (débit utile) TCP, 5 mesures dans les mêmes conditions pour chaque test, l'étoile marque la médiane.

FDN : 7,11*/7,28/6,65/7,97/5,71
OVH : 6,94/6,55/6,94/6,93/6,93*
TUN (RR 1-1) : 10,3/11,5/8,55/7,94/8.90*
TUN (détection sat): 13,0/10.3*/7,98/10,2/11,5

H. Test de deux liens agrégés, deuxièmes versions

```
[  4] local 91.224.149.199 port 5001 connected with 80.67.177.5 port 5001
[  4]  0.0-123.9 sec  10.3 MBytes    697 Kbits/sec
[  5] local 91.224.149.199 port 5001 connected with 109.190.12.102 port 64916
[  5]  0.0-122.1 sec  10.9 MBytes    750 Kbits/sec
[  4] local 10.1.0.2 port 5001 connected with 10.1.0.1 port 56190
[  4]  0.0-120.5 sec  14.5 MBytes  1.01 Mbits/sec10

jocelyn@rover:~/divers/agregation$ wget http://10.1.0.2/bigfile.bin
--2012-01-28 22:16:28--  http://10.1.0.2/bigfile.bin
Connexion vers 10.1.0.2:80...connecté.
requête HTTP transmise, en attente de la réponse...200 OK
Longueur: 41984000 (40M) [application/octet-stream]
Sauvegarde en : «bigfile.bin»

100%[================================================================================
===>] 41 984000   335K/s   ds 92s

2012-01-28 22:18:00 (447 KB/s) - «bigfile.bin» sauvegardé [41984000/41984000]



jocelyn@rover:~/divers/agregation$ wget --bind-address=192.168.2.107 http://rhizome-
fai.tetaneutral.net/bigfile.bin
--2012-01-28 22:19:34--  http://rhizome-fai.tetaneutral.net/bigfile.bin
Résolution de rhizome-fai.tetaneutral.net... 91.224.149.199
Connexion vers rhizome-fai.tetaneutral.net|91.224.149.199|:80...connecté.
requête HTTP transmise, en attente de la réponse...200 OK
Longueur: 41984000 (40M) [application/octet-stream]
Sauvegarde en : «bigfile.bin.1»

100%[================================================================================
===>] 41 984000   198K/s   ds 2m 2s

2012-01-28 22:21:35 (337 KB/s) - «bigfile.bin.1» sauvegardé [41984000/41984000]


jocelyn@rover:~/divers/agregation$ wget --bind-address=192.168.1.112 http://rhizome-
fai.tetaneutral.net/bigfile.bin
--2012-01-28 22:22:24--  http://rhizome-fai.tetaneutral.net/bigfile.bin
Résolution de rhizome-fai.tetaneutral.net... 91.224.149.199
Connexion vers rhizome-fai.tetaneutral.net|91.224.149.199|:80...connecté.
requête HTTP transmise, en attente de la réponse...200 OK
Longueur: 41984000 (40M) [application/octet-stream]
Sauvegarde en : «bigfile.bin.2»

100%[================================================================================
===>] 41 984000   839K/s   ds 50s

2012-01-28 22:23:14 (818 KB/s) - «bigfile.bin.2» sauvegardé [41984000/41984000]


#CONSO CPU ENTRE 3 et 10% (MOYENNE=5%)


jocelyn@rover:~$ ping -c20 10.1.0.2
PING 10.1.0.2 (10.1.0.2) 56(84) bytes of data.

--- 10.1.0.2 ping statistics ---
```

```
20 packets transmitted, 20 received, 0% packet loss, time 19021ms
rtt min/avg/max/mdev = 58.559/62.325/72.464/4.513 ms

jocelyn@rover:~$ ping -Ieth0  91.224.149.199 -c20
PING 91.224.149.199 (91.224.149.199) from 192.168.2.107 eth0: 56(84) bytes of data.

--- 91.224.149.199 ping statistics ---
20 packets transmitted, 20 received, 0% packet loss, time 19029ms
rtt min/avg/max/mdev = 57.694/58.671/60.225/0.682 ms
jocelyn@rover:~$

jocelyn@rover:~$ ping -Ieth0.1  91.224.149.199 -c20
PING 91.224.149.199 (91.224.149.199) from 192.168.2.107 eth0.1: 56(84) bytes of data.

--- 91.224.149.199 ping statistics ---
20 packets transmitted, 20 received, 0% packet loss, time 19026ms
rtt min/avg/max/mdev = 54.433/60.815/81.957/5.692 ms


#Réseau radio Client->modem
jocelyn@rover:~$ ping -c20 10.42.1.1

--- 10.42.1.1 ping statistics ---
20 packets transmitted, 20 received, 0% packet loss, time 19027ms
rtt min/avg/max/mdev = 1.379/1.974/5.669/0.892 ms
```

I. Avec un seul lien agrégé

```
jocelyn@rover:~/divers/agregation$ iperf  -c 10.1.0.2
------------------------------------------------------------
Client connecting to 10.1.0.2, TCP port 5001
TCP window size: 16.0 KByte (default)
------------------------------------------------------------
[  3] local 10.1.0.1 port 56220 connected with 10.1.0.2 port 5001
[ ID] Interval       Transfer     Bandwidth
[  3]  0.0-10.5 sec  1.14 MBytes    911 Kbits/sec

jocelyn@rover:~/divers/agregation$ iperf  -B 192.168.1.112 -c  91.224.149.199
------------------------------------------------------------
Client connecting to 91.224.149.199, TCP port 5001
Binding to local address 192.168.1.112
TCP window size: 16.0 KByte (default)
------------------------------------------------------------
[  3] local 192.168.1.112 port 5001 connected with 91.224.149.199 port 5001
[ ID] Interval       Transfer     Bandwidth
[  3]  0.0-11.4 sec  1.28 MBytes    946 Kbits/sec

jocelyn@rover:~/divers/agregation$ wget http://10.1.0.2/bigfile.bin
--2012-01-28 22:59:35--  http://10.1.0.2/bigfile.bin
Connexion vers 10.1.0.2:80...connecté.
requête HTTP transmise, en attente de la réponse...200 OK
Longueur: 41984000 (40M) [application/octet-stream]
Sauvegarde en : «bigfile.bin»

100%[======================================>] 41 984 000   816K/s   ds 52s

2012-01-28 23:00:27 (782 KB/s) - «bigfile.bin» sauvegardé [41984000/41984000]

jocelyn@rover:~/divers/agregation$ wget --bind-address=192.168.1.112 http://rhizome-
fai.tetaneutral.net/bigfile.bin
--2012-01-28 23:00:49--  http://rhizome-fai.tetaneutral.net/bigfile.bin
Résolution de rhizome-fai.tetaneutral.net... 91.224.149.199
Connexion vers rhizome-fai.tetaneutral.net|91.224.149.199|:80...connecté.
requête HTTP transmise, en attente de la réponse...200 OK
Longueur: 41984000 (40M) [application/octet-stream]
Sauvegarde en : «bigfile.bin.1»

100%[======================================>] 41 984 000   837K/s   ds 50s

2012-01-28 23:01:39 (820 KB/s) - «bigfile.bin.1» sauvegardé [41984000/41984000]
```

J. Comparaison par rapport à la perte de débit

Sans modification de la MTU (tun0 mtu=1500)
Sans tunnel : 820kB/s
Avec tunnel : 782kB/s

-> 4.5% (en tun donc 1,5% de perte de performance liée au traitement)

i. Modification de la MTU tun0 mtu=1400

```
jocelyn@rover:~/divers/agregation$ wget --bind-address=192.168.1.112 http://rhizome-
fai.tetaneutral.net/bigfile.bin
--2012-01-28 23:42:19--  http://rhizome-fai.tetaneutral.net/bigfile.bin
Résolution de rhizome-fai.tetaneutral.net... 91.224.149.199
Connexion vers rhizome-fai.tetaneutral.net|91.224.149.199|:80...connecté.
requête HTTP transmise, en attente de la réponse...200 OK
Longueur: 41984000 (40M) [application/octet-stream]
Sauvegarde en : «bigfile.bin.9»

100%[======================================>] 41 984 000   838K/s   ds 50s

2012-01-28 23:43:09 (815 KB/s) - «bigfile.bin.9» sauvegardé [41984000/41984000]

jocelyn@rover:~/divers/agregation$ wget http://10.1.0.2/bigfile.bin
--2012-01-28 23:43:15--  http://10.1.0.2/bigfile.bin
Connexion vers 10.1.0.2:80...connecté.
requête HTTP transmise, en attente de la réponse...200 OK
Longueur: 41984000 (40M) [application/octet-stream]
Sauvegarde en : «bigfile.bin.10»

100%[======================================>] 41 984 000   675K/s   ds 51s

2012-01-28 23:44:06 (798 KB/s) - «bigfile.bin.10» sauvegardé [41984000/41984000
```

K. Influence MTU download multi-agreg

En kB/s FDN/OVH/TUN(MTU=1500)/TUN(MTU=1350)

Avec modif de la MTU (1350) : 826/820/818/863

Moyenne de 3 mesures (valeurs *très* fluctuantes, à prendre avec des pincettes *mais* on voit que la MTU apporte une très légère amélioration mais n'est pas le seul problème…

III. Le journal d'activité

A. du 23 au 30 Janvier

- Implémentation de l'agrégation avec ajustement dynamique des pondérations
- Mesures sur une vraie agrégation de deux liens (latence, download, upload…)
- Analyse des résultats
- Mesures de l'influence du réseau radio
- création d'un script basique d'initialisation
- Rapport

B. 22 Janvier

- Configuration du routage avec IProute2
  - Script dual-wan.sh de configuration du routage (attention, flush les tables de routage…)

C. 16 Janvier

- Détection de saturation
  - Le tableur détecte à la fois les saturations en upload et download
  - Le tableur prend maintenant des paramètres au lieu de valeurs en dur pour ajuster la formule…
  - Bugfixé le script qui nettoie les CSV.
    TODO: reproduire et vérifier l'histoire de délais dans les 2 sens, appliquer le tableau paramétré détectant l'UP et down aux mesures précédentes

D. 8 janvier.

- Détection de saturation:
  - Évolution de delta_half_trip_time.py pour enregistrer un historique des délais (dans les 2 sens)
  - Ajout d'une détection de saturation (… Mais à améliorer, trop de faux positifs)
  - Création de l'outil de test load_uplink.py pour charger progressivement un lien jusqu'à la saturation et pouvoir ainsi observer le comportement du ping.
- Compréhension du script multy.py de Laurent Guerby
  - Commentaire du script multy.py
  - Schéma graphique du fonctionnement de multy.py

E. 28/29 déc.

- Détection de saturation : nouvel outil pour mesurer les délais dans un sens
  - Création de l'outil, qui fonctionne de manière bidirectionelle et rapporte les informations aux deux pairs
  - Première mesure rapide sur un iperf en TCP, dans un sens puis dans l'autre, simplement pour valider la détection.

F. 5 déc.

- Détection de saturation :
- Output CSV en direct vers le fichier plutôt que statiquement au bout de 3 minutes…
- Écriture d'un outil de script de logs CSV
- Collecte de mesures sur l'effet sur le ping de la saturation d'un lien en UDP et TCP

- Analyse basique des résultats

G. 27 nov.

- Lecture et utilisation de linkagreg (outil d'agrégation de Fernando)
- Faire fonctionner linkagreg sur une architecture 64bits
- Faire fonctionner linkagreg avec une connection sur le client //Fonctionnel
- Faire fonctionner linkagre avec n connection sur le client //Non fonctionnel
    - Test avec une connection filaire et WiFi //Non fonctionnel car perte (important) de paquet sur le lien WiFi
    - Test avec des connections virtuelles //Non fonctionnel car QoS inapplicable sur des interfaces virtuelles
    - Test avec deux interfaces physiques //Non fonctionnel car QoS déficiente

- Ajout de la collecte de données sur les temps de réponse (ping) périodiquement.
- Export des données en CSV (pour exploitation/graphe… etc.)
- Premier jeu de mesure (mauvais) sur une ligne adsl. *

H. 11 nov.

- Debuggage du problème de MTU (c'est honteux mais c'est bêtement la taille des buffers qui n'était pas assez grande dans le programme. Notamment dû aux pseudo en-têtes, cf plus bas).
- Configuration auto des adresses IP de chaque côté du tunnel (plus besoin d'ifconfig à la main)
- Ajout sur tunproxy.py de compteurs de débit * mémorise le traffic sur les x dernières tranches de n secondes (défaut 10 tranches de 1 seconde) * Affiche les moyennes et les max.
- Compréhension de ce qui passe dans TUN : bien qu'étant un tunnel de niveau 3, il y a une pseudo-en-tête de L2, cf doc officielle (merci Laurent!)
- discussion avec Laurent sur les intérêts de faire un tunnel L2 (qui rajoute pourtant l'overhead de l'en-tête L2), en bref : * évite de gérer les soucis spécifiques du niveau IP * TUN ne supporte pas IPV6 par exemple …

I. 5 nov.

- Mise en place d'un dépôt git (gitolite) pour partager du code avec Fernando Alves de Sames Wireless :
    1. Dépot public : (lecture-seule)
       git clone git://rhizome-fai.tetaneutral.net/agregation.git

J. 2 nov

- Modification de la MTU pour éviter la fragmentation de paquet

K. 28 oct.

- Initiation python (découverte pour Yanick)
- Commentaire intégral du tunproxy.py et premiers tests de ce dernier
    - ping ok (+1ms)
    - iperf à travers le tunnel : BP ~= celle de l'uplink ADSL. Le dernier datagrame ne reçoit pas d'ACK

```
[  3] local 10.0.0.2 port 50191 connected with 10.0.0.1 port 5001
[ ID] Interval       Transfer     Bandwidth
[  3]  0.0-10.0 sec  1.25 MBytes  1.05 Mbits/sec
[  3] Sent 893 datagrams
[  3] WARNING: did not receive ack of last datagram after 10 tries.
```