I.F.I.P.S. Informatique
Cinquième année
Spécialité Architecture des Réseaux

TRAFFIC SHAPING
SOLUTIONS OPEN SOURCE

Thibault Blaiset
Franck Massé
Aurélien Méré

SOMMAIRE

1. Introduction

1.1. Le traffic shaping

1.1.1. Le besoin

Que ce soit dans le cadre d'un réseau personnel ou d'une infrastructure d'entreprise, la bande passante est une ressource importante.

Vouloir conserver un accès fluide aux pages HTTP tandis que l'on télécharge une distribution Linux par FTP constitue un exemple de besoin personnel de contrôle de la bande passante.

Dans le cas d'une entreprise, les besoins sont plus complexes : on peut souhaiter garantir un accès équitable aux visiteurs d'un site Web, offrir un accès privilégié au même serveur à partir des réseaux de ses partenaires ou bien encore conserver une partie fixe de sa bande passante vers ses serveurs pour garantir diverses opérations de maintenance.

1.1.2. Définition du traffic shaping

L'appellation « traffic shaping » regroupe l'ensemble des techniques permettant de contrôler les quantités de données circulant sur un lien réseau par l'application d'une politique paramétrée.

1.1.3. Fonctionnalités recherchées

Pour généraliser l'expression du besoin, on peut classer les fonctionnalités de traffic shaping dans les catégories suivantes :

- *Limitation de bande passante pour un client ou un groupe de clients*

  L'idéal est de pouvoir utiliser des masques d'adresses, de faire référence à des sous réseaux, etc.

- *Limitation de trafic par protocole / port*

  On peut souhaiter limiter individuellement IP, TCP, UDP, ou des protocoles tels que HTTP, FTP, les échanges Peer-to-Peer etc.

- *Définition de niveaux de priorités*

  Ceci revient à faire en sorte que certains types de données soient transmis de façon prioritaire, indépendamment des limites de bande passante.

- *Garanties de bande passante pour les clients et/ou protocoles spécifiés en cas de congestion du réseau*

  Ainsi, on protège les services critiques au détriment des autres types de flux.

- *Application à tout type de ressource réseau*

  Les besoins peuvent en effet s'appliquer à des ressources locales (réseau interne) ou distantes (réseau externe, Internet).

- *Transparence*

  Idéalement, la solution de traffic shaping devrait s'intégrer au réseau sans générer de modification particulière de l'architecture ou des configurations des équipements. En particulier, il serait préférable de ne pas avoir à ajouter d'autre matériel ni à modifier les configurations réseaux des postes clients ou des serveurs.

1.2. Objectifs de l'étude

1.2.1. Description de différentes solutions

Tout d'abord, cette étude doit présenter un échantillon représentatif des différentes solutions Open Source, de préférence sur plusieurs systèmes d'exploitation. Chaque présentation doit détailler le mode de fonctionnement ainsi que les fonctionnalités offertes.

Nous avons choisi de décrire les solutions existantes sous OpenBSD 3.X, FreeBSD 5.X et Linux 2.2 et suivants. A titre de comparaison, une brève présentation d'outils Windows suivra celles des solutions Open Source.

1.2.2. Déploiement et tests d'une partie des solutions décrites

Il convient d'effectuer le déploiement d'une partie significative des solutions présentées en première partie dans le but de s'assurer de leur validité par des tests d'accès à une ressource.

1.2.2.1 Choix d'une infrastructure

Afin de réaliser les tests, il était nécessaire de mettre en place une ressource réseau à contrôler.

Nous avons décidé que cette ressource serait un serveur FTP auquel des clients pourraient accéder sous le contrôle du traffic shaper.

1.2.2.2 Définition d'un protocole de test

Afin d'évaluer le fonctionnement des solutions testées, il faut mettre en place une suite d'opérations à effectuer de façon identique sur chacune d'entre elles.

Nous avons donc décidé d'une politique de traffic shaping que les solutions retenues devront implémenter. Les tests consisteront à vérifier que la politique est bien appliquée par le traffic shaper.

1.2.2.3 Déploiement des systèmes et tests

Nous avons pu tester deux solutions :

- *PF – ALTQ sur OpenBSD 3.6*
- *IPFW – DUMMYNET sur FreeBSD 5.3*

Une partie de ce document présente donc les étapes du déploiement ainsi que les résultats des tests.

1.2.3. Conclusion

La dernière étape consiste à exploiter les résultats des tests précédents avec les objectifs suivants :

- *Mettre en évidence les caractéristiques principales des solutions présentées.*
- *Valider le fonctionnement des solutions testées.*
- *Comparer les performances de celles – ci.*

2. Les solutions existantes

2.1. Les solutions d'OpenBSD 3.X

2.1.1. Présentation générale

L'équipe d'OpenBSD a choisi de ne plus fournir que PF (Packet Filter) pour toutes les tâches relatives au contrôle des flux. PF remplit donc les rôles de NAT et de firewall pour OpenBSD à partir de la version 3.0.

2.1.2. PF (PacketFilter) + ALTQ

2.1.2.1 Présentation

Précédemment, OpenBSD utilisait le firewall IPF mais un changement dans la licence de ce logiciel a contraint l'équipe d'OpenBSD à inclure un nouveau firewall doté d'une licence compatible avec celle d'OpenBSD.

PF est donc le firewall officiel d'OpenBSD depuis la version 3.0. Dans le même temps, ALTQ (Alternate Queueing) a été intégré à OpenBSD sous la forme d'un programme autonome pour réaliser du contrôle de flux.

C'est depuis la version 3.3 que PF et ALTQ ont fusionné pour fournir une solution firewall / traffic shaper unique. Depuis, il n'est plus possible d'utiliser ALTQ sans PF.

2.1.2.2 Fonctionnement et fonctionnalités

PF permet d'effectuer du traffic shaping en manipulant des *queues* : une *queue* contient les paquets en attente d'émission sur une interface réseau. La manière dont les paquets seront traités dans cette *queue* va influer sur le débit des connexions.

Ce sont les règles de filtrage de PF qui permettent de diriger un paquet vers une *queue*. Ainsi, on peut assigner des *queues*, et donc contrôler les débits dans les configurations suivantes :

- *Par machine*
- *Par protocole*
- *Par sous réseau*
- *Par type de messages (ex : paquets d'acquittement TCP : ACK)*
- *…*

De plus, PF supporte le protocole Explicit Congestion Notification (ECN), qui permet de rendre compte à un client (compatible ECN) d'une congestion sur le lien.

Cette grande flexibilité est l'avantage majeur de la solution PF.

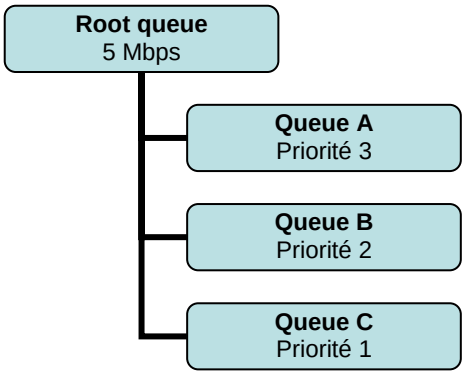
Concernant les configurations, PF permet de définir deux types de *queues* qu'il est possible d'appliquer indépendamment sur chaque interface du shaper :

- *Priority Queueing (PRIQ)*

  La première étape est de définir le trafic maximum autorisé sur l'interface considérée en associant à celle-ci une *root queue*.

  Ensuite, on rattache à chaque *root queue* plusieurs *queues*, qui ont chacune une priorité. Le trafic d'une *queue* de priorité élevée sera *toujours* traité avant celui d'une *queue* de priorité plus faible.

Voici un exemple de configuration PRIQ :



Il n'existe qu'un niveau de *queues*, en d'autres termes, il est impossible de définir de nouvelles priorités pour distinguer différents trafics au sein d'une *queue*.

Dans une *queue* donnée, les paquets sont traités en mode First In First Out.

Une fois que la *queue* de priorité la plus élevée sera vide, PF traitera la *queue* de plus grande priorité parmi les autres.

Puisque le trafic d'une *queue* de priorité élevée sera *toujours* traité avant celui d'une *queue* de priorité plus faible, l'attribution des priorités doit être réalisé avec soin. Dans le cas contraire, une *queue* de priorité faible pourrait voir tous ses paquets retardés, voir perdus si une *queue* de priorité plus élevée recevait un flux constant de paquets.

- *Class Based Queueing (CBQ)*

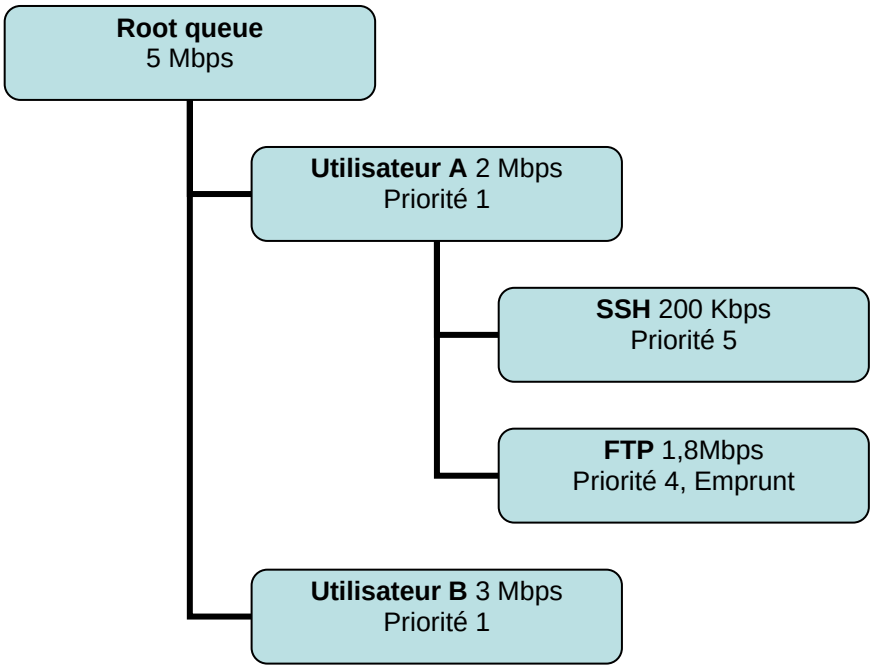
  Dans ce mode de fonctionnement, les *queues* sont organisées de façon arborescente. On assigne un débit à chaque *queue*, ainsi qu'une priorité et enfin on définit la capacité d'une *queue* à emprunter le débit alloué à ses sœurs si elles ne l'utilisent pas.

  La somme des débits des filles d'une *queue* ne peut pas être supérieure au débit de celle-ci.

  La gestion des priorités lors des congestions est effectuée au sein d'une même branche. Entre deux *queues* ayant la même *queue* mère, celle ayant la plus grande priorité sera avantagée sur l'autre en cas de congestion. Deux *queues* ayant la même priorité seront traitées tour à tour.

  On obtient ainsi une arborescence complète pouvant ressembler à l'exemple ci-dessous :

  - *On a assigné un débit maximum à la root queue, que l'on a subdivisé en deux queues correspondant aux machines de deux utilisateurs.*
  - *Entre les utilisateurs A et B, les paquets sont traités tour à tour car les priorités de leurs queues sont identiques.*
  - *Pour l'utilisateur A, le SSH (priorité 5) sera traité de façon privilégiée par rapport au FTP (priorité 4) en cas de congestion.*



- *La configuration de FTP en mode « Emprunt » (option borrow dans la configuration) permet à cette queue d'exploiter la bande passante de ses sœurs (ici SSH) lorsque le flux FTP a atteint son débit maximum (1,8 Mbps) mais que le débit de l'utilisateur A est inférieur à sa limite (2 Mbps), c'est à dire dans le cas où la queue SSH ne serait pas pleine. Si la queue SSH se remplissait à nouveau, le débit emprunté serait immédiatement restitué.*

De plus, pour prévenir les congestions, OpenBSD offre une fonctionnalité appelée Random Early Detection (RED) qui peut s'appliquer ou non à chaque *queue*. Le principe est d'éliminer des paquets pour prévenir les congestions lorsque le débit approche de sa limite autorisée. Pour cela, on compare la taille moyenne de la *queue* à deux paliers :

- *Si la taille moyenne de la queue est inférieure au pallier le plus bas, tous les paquets arrivant seront traités.*
- *Si la taille moyenne de la queue est supérieure au pallier le plus haut, tous les paquets arrivant seront éliminés.*
- *Entre les deux paliers, PF ignore aléatoirement une partie des paquets arrivant :*

  Plus la taille de la *queue* est proche du palier supérieur, plus PF éliminera de paquets. Les connexions dont les paquets seront éliminés sont choisies aléatoirement, ce qui a pour conséquence de ne pas couper brutalement une connexion alors qu'une autre continuerait de se dérouler normalement. De plus, une connexion envoyant de nombreux paquets a plus de chance de voir une partie de ses paquets être éliminés qu'une connexion au débit plus réduit.

  Ainsi les connexions à fort débit sont réduites, les congestions sont évitées et les pertes violentes de débit ne se produisent pas. Les pics de trafic sont aussi mieux gérés car une *queue* utilisant RED ignorera quelques paquets avant qu'elle ne soit pleine. Lors d'un pic de trafic, il reste suffisamment de ressources pour accueillir une partie des nouveaux paquets.

2.1.2.3 Informations complémentaires

http://www.openbsd.org/faq/pf/queueing.html

2.2. Les solutions de FreeBSD 5.X

2.2.1. Présentation générale

Trois solutions sont prévues sous FreeBSD 5 pour réaliser du traffic shaping : PacketFilter, IPFireWall + Dummynet et enfin IPFilter + ALTQ.

2.2.2. PF (PacketFilter) + ALTQ

2.2.2.1 Présentation

Le firewall PF est issu du monde OpenBSD.

Son intégration à FreeBSD s'est concrétisée en juillet 2003. La première release de FreeBSD qui intégrait PF en tant que partie du système de base est la version 5.3-RELEASE.

PF inclut ALTQ (Alternate Queueing) depuis la version 3.3 de OpenBSD pour réaliser un contrôle de la bande passante.

FreeBSD a choisi d'intégrer des versions de PF équivalentes aux versions disponibles sous OpenBSD 3.4 et plus récentes, incluant donc ALTQ.

2.2.2.2 Fonctionnement et fonctionnalités

Le fonctionnement est identique à celui PF sous OpenBSD (voir 2.1.2.2.). Toutefois son utilisation sur des bridges est encore au stade expérimental; c'est pourquoi nous n'avons pu tester cette solution.

2.2.3. IPFW2 (IPFireWall2) + Dummynet

2.2.3.1 Présentation

IPFW est un outil de filtrage de paquets initialement développé par BSDi. La version 2 est celle qui est incluse à FreeBSD depuis la version 5.0.

Dans ce dossier, nous utiliserons l'appellation IPFW, bien qu'il s'agisse de la version 2 puisque nous avons travaillé avec FreeBSD 5.3.

En plus de sa fonctionnalité de firewall, IPFW sert de socle à Dummynet, un logiciel conçu à l'origine pour tester les protocoles réseaux en simulant des réseaux aux performances différentes. Aujourd'hui, Dummynet est largement utilisé pour effectuer de la gestion de bande passante.

L'intégration de Dummynet à FreeBSD remonte à septembre 1998 mais il a été réécrit entre janvier et juin 2000. La version réécrite est donc disponible à partir de FreeBSD 3.4-STABLE.

2.2.3.2 Fonctionnement et fonctionnalités

IPFW permet de contrôler DUMMYNET de la manière suivante : Les paquets sont divisés en catégories par les règles de IPFW. Les flux de chaque catégorie sont ensuite passés à des objets propres à DUMMYNET qui réalisent le shaping.

Comme dans le cas de PF, l'utilisation des règles de IPFW pour trier les paquets offre une grande flexibilité. On peut ainsi définir des règles de shaping sur des protocles, des machines, des sous-réseaux, etc…

Les objets disponibles sous DUMMYNET permettent d'obtenir deux types de fonctionnements :

- *Les pipes*

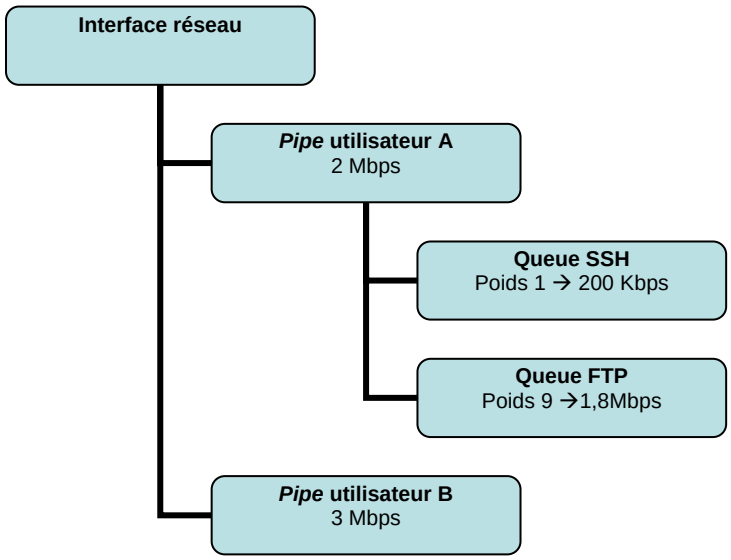
  Ils permettent de donner une limite à la bande passante d'un flux, ainsi que de définir un temps de propagation ou une probabilité de perte des paquets. Ici, la fonctionnalité qui nous intéresse principalement est la limitation de la bande passante.

- *Les queues*

  Elles associent aux flux qui leurs sont passés un poids ainsi qu'un identifiant de *pipe*. Les flux sont ensuite redirigés vers le *pipe* qui leur a été attribué. A l'intérieur de ce *pipe*, ils se partagent la bande passante disponible en fonction du poids qui leur a été attribué. Il ne s'agit donc pas d'une priorité, puisque même les flux ayant un poids faible se verront attribuer une fraction de la bande passante.

  Dans la pratique, les pipes sont utilisées pour limiter la bande passante d'un flux, tandis que les queues permettent de déterminer comment différents flux partageront la bande passante disponible. De cette manière, on peut optimiser le partage de la bande passante entre les flux : il est possible de définir une répartition entre deux flux (1/3 – 2/3 par exemple) qui ne s'appliquera que quand les deux flux seront actifs. Le reste du temps, le seul flux actif bénéficiera de toute la bande passante.

  L'exemple ci-dessous schématise une configuration entre deux utilisateurs proche de celle présentée en 2.1.2.2 dans la présentation de PF :



  Enfin, tout comme PF, IPFW propose l'option Random Early Detection (RED) qui permet de prévenir les congestions (voir 2.1.2.2).

2.2.4. IPF (IPFilter) + ALTQ

Le firewall IPF ne dispose pas de module dédié au traffic shaping mais peut être utilisé en collaboration avec l'application ALTQ, qui devait être disponible sous sa forme indépendante dans l'ensemble des applications portées sous FREEBSD 5. Malheureusement, l'intégration n'est pas encore terminée au 26/01/2005 alors que la version 5.3-RELEASE de FreeBSD a été publiée le 06/11/2004. En l'absence de version finalisée d'ALTQ, nous ne pourrons pas intégrer la solution IPF+ALTQ à notre étude.
http://daemon.rofug.ro/projects/freebsd-altq/
http://www.csl.sony.co.jp/person/kjc/kjc/software.html

2.3. Solutions sous Linux 2.2 (et suivants)

2.3.1. Présentation générale

La solution proposée par Linux a complètement évolué depuis la version 2.2 du noyau. En effet, les noyaux Linux des séries 2.2 et plus ont un sous-système réseau complètement réécrit. La nouvelle structure d'Iproute2 a permis de formuler clairement des fonctionnalités impossibles à implémenter dans le sous-système réseau précédent.

Réaliser du traffic shaping sous Linux post-2.2 revient donc à utiliser les deux outils suivants : Iptables / Netfilter et Iproute2.

2.3.2. Iptables / Netfilter

Iptables / Netfilter permet de configurer le filtrage de paquets, le NAT et beaucoup d'autres fonctionnalités qui ne seront pas détaillées ici. Dans le cadre du traffic shaping, Iptables permet de marquer les paquets circulant sur une interface pour qu'ils soient ensuite routés en fonction de la politique de shapping.

2.3.3. Iproute2

Le package Iproute2 est composé principalement des utilitaires TC (outil de Traffic Control permettant entre autres le traffic shaping) et IP (consultation et modification des paramètres réseau tels que la table de routage). De nombreux modes de gestion des files d'attente sont implémentés. Nous les présentons brièvement ci-dessous.

On peut noter qu'ici aussi la fonction RED est disponible.

2.3.3.1 Pfifo_fast

Pfifo_fast correspond à une version légèrement complexifiée du principe FIFO :

Trois files First In First Out sont gérées en parallèle. Les données arrivant sur l'interface sont automatiquement aiguillées vers la file qui leur correspond, en fonction des paramètres indiqués dans leurs en-têtes.

La gestion des priorités est effectuée de la manière suivante : Tant qu'il y a un paquet en attente dans la bande 0, la bande 1 ne sera pas traitée. Il en va de même pour la bande 1 et la bande 2.

Le fait que le tri se fasse à l'aide des en-têtes des paquets simplifie grandement la gestion du point de vue de l'administrateur.

2.3.3.2 Token Bucket Filter

L'implémentation TBF consiste en un tampon (seau), constamment rempli par des éléments virtuels d'information appelés jetons, avec un débit spécifique (débit de jeton).

Chaque jeton entrant laisse sortir un paquet de données de la file d'attente de données et ce jeton est alors supprimé du seau. L'association de cet algorithme avec les deux flux de jetons et de données, nous conduit à trois scénarios possibles :

- *Les données arrivent dans TBF avec un débit égal au débit des jetons entrants.*

  Dans ce cas, chaque paquet entrant a son jeton correspondant et passe la file d'attente sans délai.

- *Les données arrivent dans TBF avec un débit plus petit que le débit des jetons.*

Seule une partie des jetons est supprimée au moment où les paquets de données sortent de la file d'attente, de sorte que les jetons s'accumulent jusqu'à atteindre la taille du tampon. Les jetons libres peuvent être utilisés pour envoyer des données avec un débit supérieur au débit des jetons standard, si de courtes rafales de données arrivent.

- *Les données arrivent dans TBF avec un débit plus grand que le débit des jetons.*

  Dans ce cas le seau sera bientôt dépourvu de jetons, ce qui provoquera l'arrêt de TBF pendant un moment. Ceci s'appelle « une situation de dépassement de limite » (*overlimit situation*). Si les paquets continuent à arriver, ils commenceront à être éliminés.

2.3.3.3 Stochastic Fairness Queueing

Le principe-clé dans SFQ est la conversation (ou flux), qui correspond principalement à une session TCP ou un flux UDP. Le trafic est divisé en un grand nombre de files d'attente FIFO : une par conversation. Le trafic est alors envoyé dans un tourniquet, donnant une chance à chaque session d'envoyer leurs données tour à tour.

Ceci conduit à un comportement très équitable et empêche qu'une seule conversation étouffe les autres. SFQ est appelé « Stochastic » car il n'alloue pas vraiment une file d'attente par session, mais possède un algorithme qui divise le trafic à travers un nombre limité de files d'attente en utilisant un algorithme de hachage.

A cause de ce hachage, plusieurs sessions peuvent finir dans le même seau, ce qui peut réduire de moitié les chances d'une session d'envoyer un paquet et donc réduire de moitié la vitesse effective disponible. Pour empêcher que cette situation ne devienne importante, SFQ change très souvent son algorithme de hachage pour que deux sessions entrantes en collision ne le fassent que pendant un nombre réduit de secondes.

2.3.3.4 Priority Queueing (PRIQ)

Ici, TC implémente le principe de priorités déjà abordé dans la section 2.1.2.2 : après avoir trié les flux par priorité, les paquets seront traités en commençant par les plus importantes, puis en descendant vers les priorités moins importantes quand les queues se vident.

La principale différence est qu'il est ici possible de bâtir une arborescence de priorités. Dans cette arborescence, seules les feuilles représenteront des queues, les nœuds intermédiaires n'ont pas d'autre rôle que de structurer les queues.

2.3.3.5 Class Based Queueing (CBQ)

On retrouve ici le concept CBQ présenté en 2.1.2.2 : le trafic est fragmenté en classes et l'on affecte à chacune des paramètres tels que la bande passante utilisable et la possibilité d'échanger de la bande passante avec d'autres classes.

2.3.4. Solution complète

Les nombreuses techniques de gestions des queues présentées précédemment pouvant être utilisées de façon collaborative (ex : définition de priorité à l'intérieur d'une classe), les possibilités offertes sont extrêmement variées. Cependant, il semble qu'en pratique, les seuls modes de gestion de queues utilisés soient PRIQ et CBQ.

Ainsi, la solution de traffic shaping la plus répandue consiste à faire usage de l'outil Iptables pour sélectionner et marquer les paquets et leur attribuer une classe qui serait traitée en Class Based Queueing par TC de Iproute2.

2.4. Solutions sous Windows

Il paraissait intéressant de rechercher rapidement, à titre de comparaison, les solutions disponibles sous le système d'exploitation le plus répandu, même s'il ne semble pas être la plateforme idéale pour des tâches de type trafic shaping. Nous avons donc trouvé les deux produits suivants, le premier étant un outil professionnel coûteux et probablement performant, et le deuxième étant une solution un peu moins « sérieuse », mais plus accessible et présentant des fonctionnalités néanmoins intéressantes.

2.4.1. InJoy Firewall 3.0

Injoy Firewall est un Firewall complet avec fonctionnalité de Traffic Shapping. Quatre classes de priorité (de 1 à 4) sont définies (1:high/2:normal/3:low/4:idle). La configuration permet d'associer chaque port à une classe de priorité. Un pourcentage de bande passante maximale attribué à chaque classe est configurable.

2.4.1.1 Configuration par défaut

- Le trafic sur les ports 1-1024 sont placés dans la classe normal priority(2).
- Le trafic interactif pour les protocoles tels que: telnet, ssh, www, icq, dns, ftp port 21, irc et ICMP, sont placés en classe high priority (1).
- Les flux sortants HTTP et FTP sont placés dans la classe low priority (3).
- Le trafic Peer-to-Peer est placé dans la classe idle (4).
- Le reste du trafic est placé dans la classe low priority.
- Par défaut, aucune limitation de bande passante n'est associée aux différentes classes, mais cela peut être fait facilement via l'interface de configuration.

2.4.1.2 Compatibilité

InJoy Firewall 3.0 est disponible pour Windows 2000 Professional, Windows 2000 Server, Windows XP Home, Windows XP Professional, Windows 2003 Server ainsi que pour Linux et OS2.

2.4.1.3 Prix

Il en coûte de 30$ pour un seul utilisateur réseau à 3000$ pour 1000 utilisateurs.

2.4.1.4 Informations complémentaires

www.fx.dk

http://www.fx.dk/firewall/windows.html

http://www.fx.dk/firewall/shaping.html

http://www.fx.dk/firewall/purchase.html

2.4.2. Bandwidth Controller

Bandwidth Controller est une application uniquement dédiée au traffic shaping et ne joue pas le rôle d'un firewall contrairement aux solutions vues précédemment. Le déploiement est rapide et son prix en fait un produit très accessible.

2.4.2.1 Fonctionnalités

- Limitation de bande passante pour un client ou un groupe de clients.

- Limitation de trafic par protocole / port, on peut limiter individuellement IP, TCP, UDP, HTTP, FTP, Peer-to-Peer et d'autres types de trafic.

- Garanties de bande passante pour les clients et/ou protocoles spécifiés en cas de congestion du réseau.

- Niveaux de priorités par protocole.

- Allocation de bande passante en pourcentage.

- Allocation de bande passante en débit fixe au bps près donc très précis (0 bps à 1 Gbps)

2.4.2.2 Prix

$49.95 US, Version d'évaluation (30j) gratuite

Informations complémentaires :

http://bandwidthcontroller.com/

3. Déploiement et tests

3.1. Choix d'une infrastructure

Pour tester différentes solutions de traffic shaping, nous avons mis en place une plateforme de tests. La ressource dont on souhaite contrôler l'accès sera ici un serveur qui hébergera un service FTP.

3.1.1. Matériel

- *Commutateur Cisco Catalyst 2924 (24 ports Ethernet 100 Mbps).*
- *Machine Bi-Céléron 400 pour héberger les services à contrôler, interface Ethernet 100 Mbps 3Com. Note : par la suite, cette machine sera désignée comme « le serveur ».*
- *Machine Duron 800, 512 Mo de Ram pour héberger les solutions de traffic shaping avec deux interfaces Ethernet 100 Mbps identiques (chipsets Realtek). Note : par la suite, cette machine sera désignée comme « le shaper ».*
- *4 machines clientes :*
    - *Client A : Windows 2000 server (chipset Broadcom 1Gbps)*
    - *Client B : Windows 2000 server (chipset 3Com 100 Mbps)*
    - *Client C : Linux 2.4 (chipset Realtek 100 Mbps)*
    - *Client D : FreeBSD 5.3 (chipset Intel 1Gb)*

3.1.2. Infrastructure réseau

L'infrastructure de base que nous avons choisie est composée d'un ensemble de clients et d'un serveur, tous connectés au même commutateur. (***figure 1***)

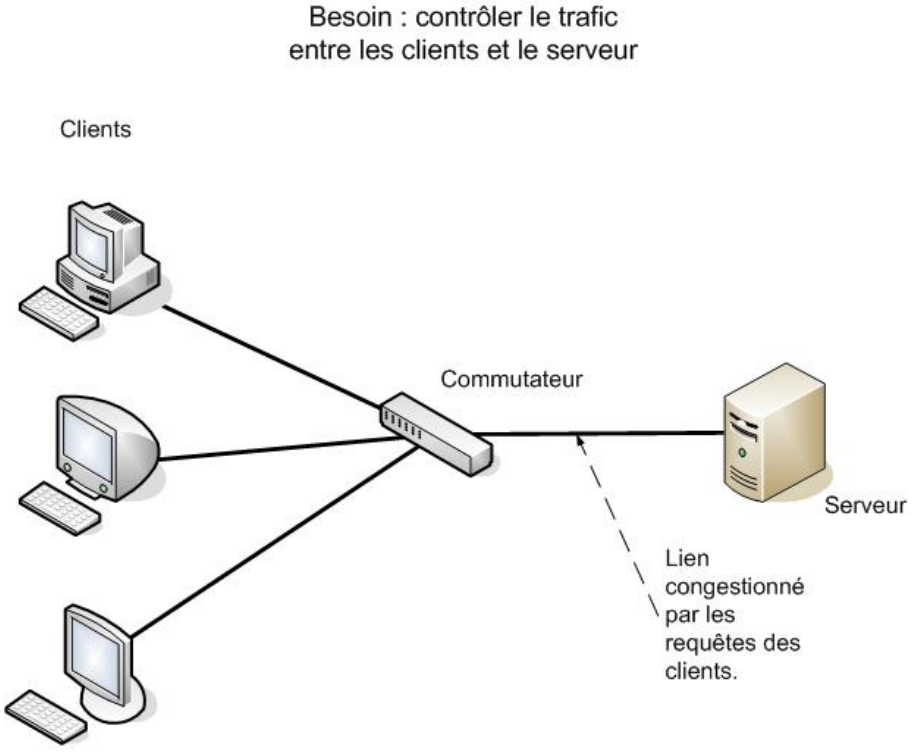


Figure 1

Pour contrôler le trafic, le shaper doit se trouver entre le réseau contenant les clients et le serveur. Une première approche aurait consisté à relier physiquement une des interfaces du shaper au serveur (qui n'est donc pas connecté directement au réseau), et l'autre interface au réseau sur lequel clients sont connectés. (***figure 2***)

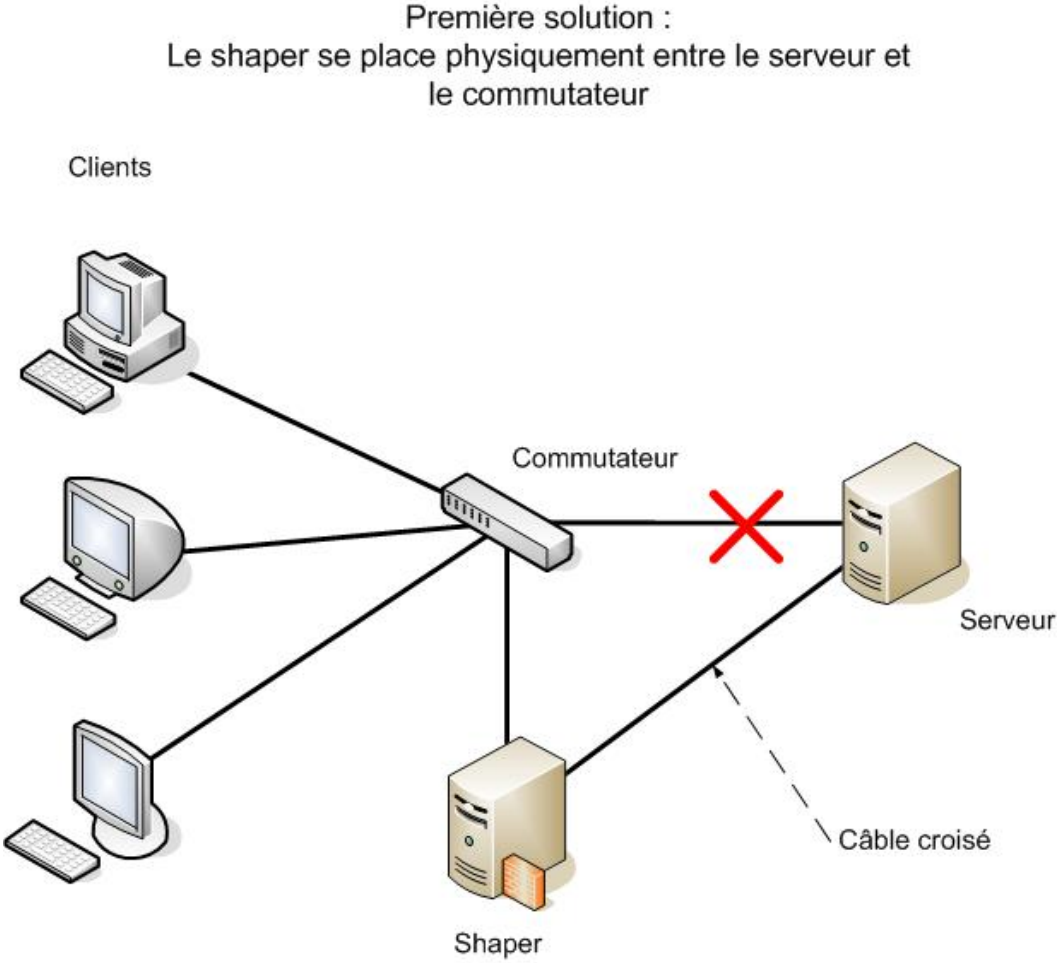


Figure 2

Cependant cette approche semble présenter l'inconvénient majeur suivant : la mise en place du shaper nécessite une modification physique de la connexion du serveur.

Ceci est d'autant plus contraignant si on veut contrôler le trafic de plusieurs serveurs (***figure 3***) avec le même shaper, il est nécessaire de mettre en place un réseau particulier qui isole ces serveurs. Cette intervention nécessite l'ajout d'un ou plusieurs commutateurs, modifications des liens etc… (***figure 4***)

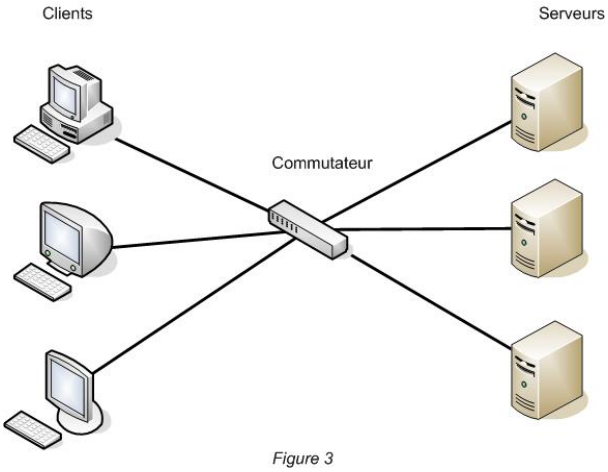


Figure 3

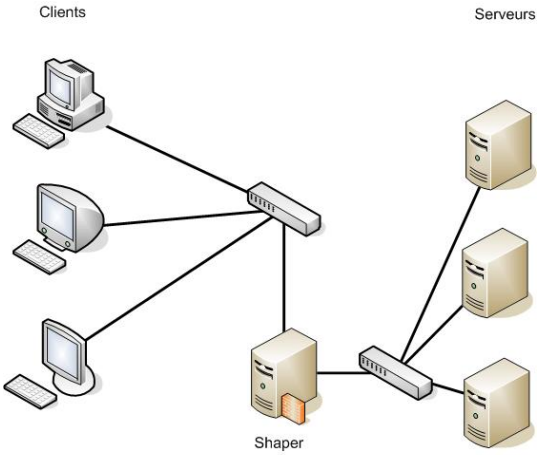


Figure 4

Nous avons donc décidé d'exploiter la possibilité offerte par le commutateur de configurer des VLANs afin d'isoler le réseau des serveurs du réseau des clients. Le shaper, grâce à ses deux interfaces réseau, est naturellement connecté à ces deux VLANs. Ainsi, tout trafic entre un client et un serveur doit passer par les deux interfaces du shaper.

Pour que les trames transitent entre les deux VLANs, le shaper doit être configuré comme un pont (voir 3.2.X.1. Installation et configuration préliminaire des plateformes).

Une fois le shaper connecté à deux ports du commutateur, il n'y a donc aucune modification physique du réseau à faire, puisque la configuration des VLANs est faite à distance et peut être adaptée aux ports du commutateurs sur lesquels les serveurs à contrôler sont connectés. (***figure 5***)

Deuxième solution :
Le serveur est isolé des clients par la mise en place de réseaux logiques.

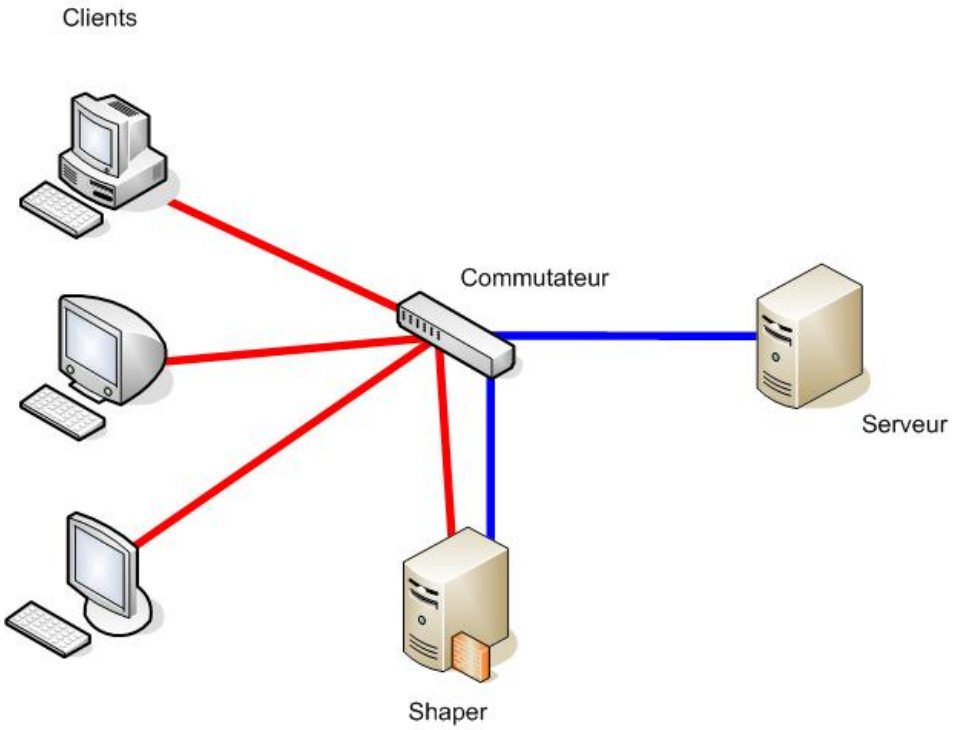


Figure 5

Dans cette configuration, contrôler plusieurs serveurs avec le même shaper ne demande aucune intervention physique. (***figure 6***)

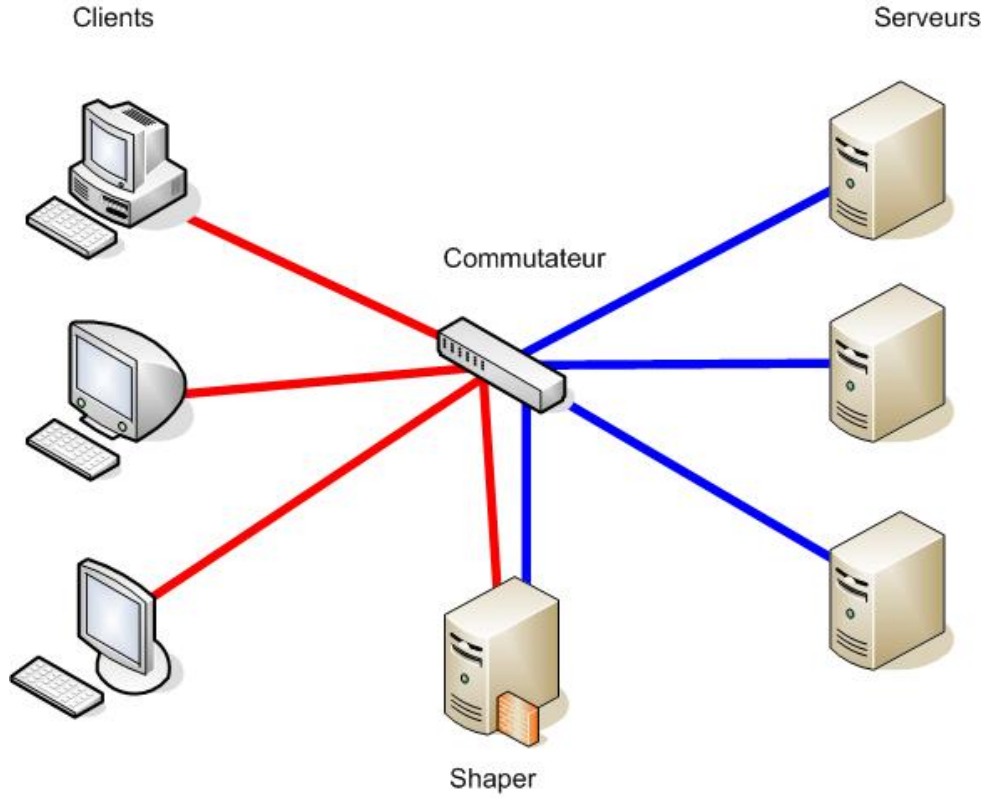


Figure 6

3.1.3. La ressource à contrôler : le lien vers le serveur

Dans notre étude, le lien réseau du serveur joue le rôle de la ressource dont on souhaite contrôler l'accès.

Pour les tests, ce serveur FreeBSD 5.3 hébergeait le service FTP suivant :

- *FTP server Version 6.00LS*

Ce service permet de générer des flux importants qui saturent le lien réseau du serveur. De plus, FTP permet de générer des débits constants sur une longue période. Les flux générés par ce protocole sont asymétriques : les commandes et leurs réponses ont des tailles faibles tandis que les données échangées représentent des volumes importants.

3.1.4. Le shaper

Nous avons décidé de tester plusieurs produits qui tournent sur deux plateformes différentes : FreeBSD 5.3 et OpenBSD 3.6.

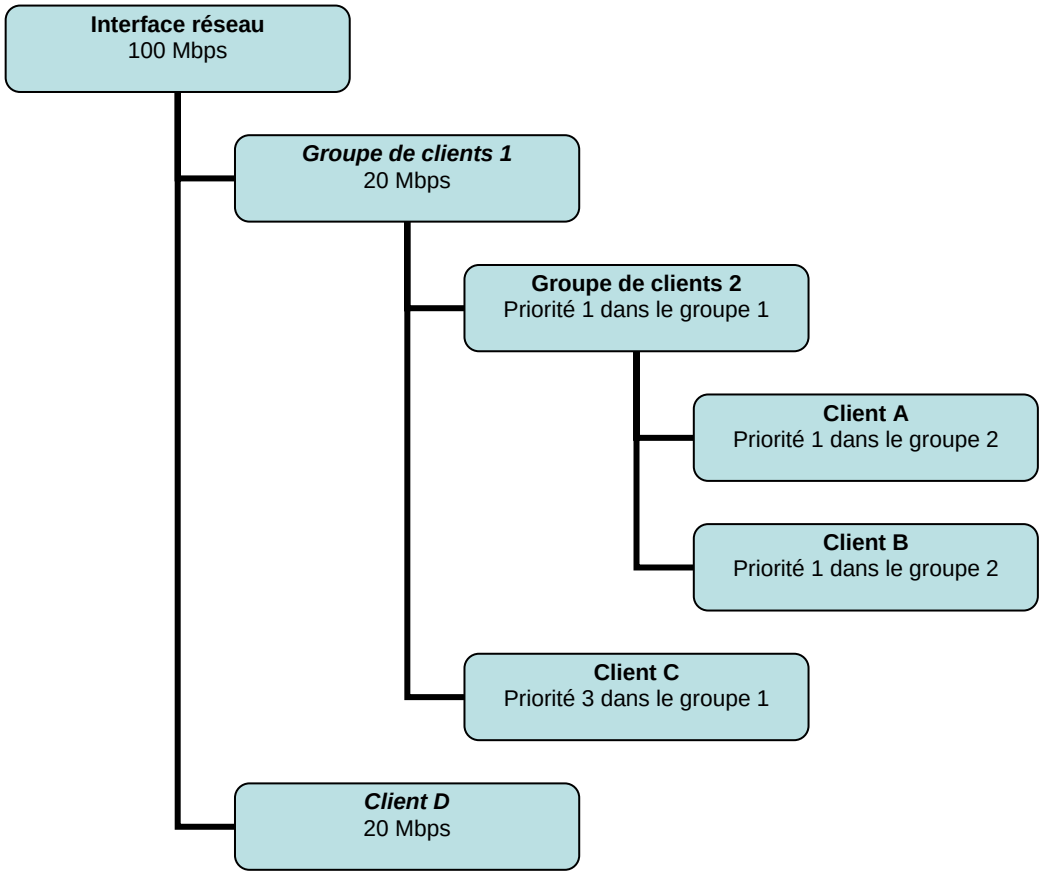
- *FreeBSD 5.3 permettra de tester IPFW-DUMMYNET*
- *OpenBSD 3.6 servira de plateforme à PF-ALTQ.*

Afin que les tests soient les plus comparables possibles (processeur, mémoire, cartes réseaux), la machine reste la même dans tous les cas. Les systèmes d'exploitation sont installés sur des disques durs différents et sont installés avec les fonctionnalités minimales.

3.2. Définition d'un protocole de test

3.2.1. La politique retenue

Nous avons choisi de partager la bande passante vers le serveur, dans le sens serveur→clients de la manière suivante :



Ainsi, le premier niveau d'arborescence définit des limites fixes pour les débits du groupe 1 et du client D : 20 Mbps pour chacun de ces ensembles.

Dans le groupe 1, on cherche à définir des priorités entre le groupe 2 (clients A et B), sans imposer de limite fixe (dans le but d'exploiter au maximum le débit du groupe).

- *Si le client C est actif et qu'une partie du groupe 2 l'est aussi, alors le client C doit bénéficier de trois fois plus de débit que le groupe 2 (priorité 3 pour C contre 1 pour le groupe 2). On doit donc obtenir 5 Mbps pour le groupe 2 et 15 Mbps pour le client C.*
- *Si le client C ou le groupe 2 est inactif, alors celui qui est actif doit bénéficier de la totalité de la bande passante du groupe 1. On obtient donc 20 Mpbs pour celui qui est actif.*

Dans le groupe 2, nous avons aussi défini un partage, cette fois ci équitable, toujours sans imposer de limite fixe par client :

- *Si A et B sont actifs, ils se partagent équitablement le débit du groupe 2. On doit obtenir 2,5 Mbps ou 10 Mbps par client selon que le client C est actif ou non. Si A ou B est inactif alors celui qui est actif doit bénéficier de la totalité du débit du groupe 2.*

3.2.2. Le protocole de test

Afin de vérifier que la politique définie ci-dessus est bien respectée, nous avons défini un ensemble de tests.

Nous avons choisi de regrouper le plus de vérifications possibles par test dans un objectif de concision.

3.2.2.1 Test n° 1 : Client C

Extraits de la politique :

- *« limites fixes pour les débits du groupe 1 […] 20 Mbps »*
- *« Si le client C ou le groupe 2 est inactif, alors celui qui est actif doit bénéficier de la totalité de la bande passante du groupe 1.»*

On cherche à vérifier :

- *Que la limite du groupe 1 est respectée.*
- *Que l'utilisation du débit dans le groupe 1 est maximale quand le groupe 2 est inactif.*

Résultat attendu :

- *Débit pour C = 20 Mbps*

3.2.2.2 Test n° 2 : Clients A et C

Extrait de la politique :

- *« limites fixes pour les débits du groupe 1 […] 20 Mbps »*
- *« Si le client C est actif et qu'une partie du groupe 2 l'est aussi, alors le client C doit bénéficier de trois fois plus de débit que le groupe 2 »*
- *« Si A ou B est inactif alors celui qui est actif doit bénéficier de la totalité du débit du groupe 2. »*

On cherche à vérifier :

- *Que la limite du groupe 1 est toujours respectée quand le groupe 2 est actif.*
- *Que les priorités à l'intérieur du groupe 1 sont respectées.*
- *Que l'utilisation du débit dans le groupe 2 est maximale quand l'un des clients est inactif.*

Résultats attendus :

- *Débit pour A = 5 Mbps*
- *Débit pour C = 15 Mbps*

3.2.2.3 Test n° 3 : Machines A, B, C et D

Extrait de la politique :

- *« limites fixes pour les débits du groupe 1 et du client D : 20 Mbps pour chacun »*
- *« Si A et B sont actifs, ils se partagent équitablement le débit du groupe 2 »*

On cherche à vérifier :

- *Que les limites du client D et du groupe 1 sont respectées.*
- *Que les priorités à l'intérieur du groupe 2 sont respectées.*

Résultats attendus :

- *Débit pour A = 2,5 Mbps*
- *Débit pour B = 2,5 Mbps*
- *Débit pour C = 15 Mbps*
- *Débit pour D = 20 Mbps*

3.2.2.4 Test n° 4 : Machines A et B

Extrait de la politique :

- *« limites fixes pour les débits du groupe 1 […] : 20 Mbps»*
- *« Si A et B sont actifs, ils se partagent équitablement le débit du groupe 2 »*

On cherche à vérifier :

- *Que la limite du groupe 1 est respectée.*
- *Que les priorités à l'intérieur du groupe 2 sont respectées.*

Résultats attendus :

- *Débit pour A = 10 Mbps*
- *Débit pour B = 10 Mbps*

3.2.2.5 Test n° 5 : Machine A

Extrait de la politique :

- *« limites fixes pour les débits du groupe 1 […] : 20 Mbps»*
- *« Si le client C ou le groupe 2 est inactif, alors celui qui est actif doit bénéficier de la totalité de la bande passante du groupe 1. »*
- *« Si A ou B est inactif alors celui qui est actif doit bénéficier de la totalité du débit du groupe 2. »*

On cherche à vérifier :

- *Que la limite du groupe 1 est respectée.*
- *Que l'utilisation du débit dans le groupe 1 est maximale quand le groupe 2 est inactif.*
- *Que l'utilisation du débit dans le groupe 2 est maximale quand l'un des clients est inactif.*

Résultats attendus :

- *Débit pour A = 20 Mbps*

3.2.3. Les outils de test

Les mesures de débit sont réalisées avec différents outils sur les différentes plateformes impliquées dans les tests.

3.2.3.1 Clients Windows (A et B)

Le système d'exploitation utilisé sur les clients Windows permet la mesure du débit sur les interfaces réseau grâce à l'analyseur graphique de performances inclut dans le système.

C'est cet outil que nous avons choisi d'utiliser pour observer les débits, observation réalisée en temps réel pour obtenir une précision plus importante.

3.2.3.2 Clients Unix (C et D)

Ne disposant pas d'outil particulier sur les clients Unix, nous allons utiliser les informations fournies par le client qui effectue le transfert FTP (lftp).

Nous prenons la précaution d'attendre que le trafic soit stabilisé avant de le relever.

Remarque : le débit indiqué par le client FTP est celui des données reçues ; Ce débit ne tient pas compte des en-têtes TCP/IP et est, par conséquent, inférieur au débit réel de l'interface. Ceci doit être pris en compte dans l'interprétation des données collectées.

3.2.3.3 Serveur FreeBSD

Pour le serveur, nous avons utilisé un outil développé par un des membres du groupe. Celui-ci génère des graphiques en interrogeant le commutateur sur le débit de chaque port et en mettant en forme ces données. Cet outil utilise RDD Tool.

3.3. Déploiement et tests des différents produits

3.3.1. Solution sous OpenBSD

3.3.1.1 Installation de OpenBSD 3.6 et configuration préliminaire

Après une installation minimale de OpenBSD 3.6, nous avons procédé à la mise en place du bridge :

La ligne suivante est décommentée dans */etc/sysctl.conf* :

```
net.inet.ip.forwarding=1
```

Les commandes suivantes sont utilisées pour créer le bridge :

```
echo 'up' > /etc/hostname.rl0
echo 'up' > /etc/hostname.rl1
echo 'add rl0 add rl1 up' > /etc/bridgename.bridge0
```

Ces changements seront pris en compte à chaque futur démarrage.

3.3.1.2 PF - ALTQ

3.3.1.2.1 Configuration

Afin d'activer PF à chaque démarrage, on modifie la ligne suivante dans */etc/rc.conf* :

```
pf=YES
```

Le fichier de configuration utilisé par défaut est : /etc/pf.conf. Voici le contenu de celui-ci conformément aux tests que nous avons réalisés :

```
altq on $ext_if cbq bandwidth 100Mb queue { default,client_D,groupe_1 }
queue default bandwidth 60Mb cbq(default) # 60 + 2x20 = 100
queue client_D bandwidth 20Mb
queue groupe_1  bandwidth 20Mb { groupe_2, client_C }
 queue groupe_2 bandwidth 25% cbq(borrow)
 queue client_C bandwidth 75% cbq(borrow)
# Redirection du trafic, pf lit les règles de bas en haut !
pass in quick on $ext_if all
pass out quick on $ext_if all
pass in on $int_if inet proto tcp from $serveur port 20 to any keep state queue groupe_2
pass in on $int_if inet proto tcp from $serveur port 20 to $client_D keep state queue client_D
pass in on $int_if inet proto tcp from $serveur port 20 to $client_C keep state queue client_C
```

3.3.1.3 Tests

Les résultats des tests sont présentés en annexe 1.

3.3.1.3.1 Test n° 1

Résultat attendu :

- *Débit pour C = 20 Mbps*

Résultat constaté :

- *Débit pour C = 15,28 Mbps, en-têtes TCP/IP non comprises*

Indicateur :

- *Respect de la politique : La limite semble être respectée mais le débit disponible n'est pas utilisé en totalité.*

3.3.1.3.2 Test n° 2

Résultats attendus :

- *Débit pour A = 5 Mbps*
- *Débit pour C = 15 Mbps*

Résultats constatés :

- *Débit pour A = 4,8 Mbps*
- *Débit pour C = 13,2 Mbps, en-têtes TCP/IP non comprises*

Indicateur :

- *Respect de la politique: Les limites semblent être respectées.*

3.3.1.3.3 Test n° 3

Résultats attendus :

- *Débit pour A = 2,5 Mbps*
- *Débit pour B = 2,5 Mbps*

- *Débit pour C = 15 Mbps*
- *Débit pour D = 20 Mbps*

Résultats constatés :

- *Débit pour A = 1,9 Mbps*
- *Débit pour B = 3,2 Mbps*
- *Débit pour C = 13,4 Mbps, en-têtes TCP/IP non comprises*
- *Débit pour D = 16,8 Mbps, en-têtes TCP/IP non comprises*

Indicateur :

- *Respect de la politique: Les limites fixes semblent être respectées mais le partage du débit dans le groupe 2 n'est pas parfaitement équilibré. Le client D n'utilise pas tout le débit disponible.*

3.3.1.3.4 Test n° 4

Résultats attendus :

- *Débit pour A = 10 Mbps*
- *Débit pour B = 10 Mbps*

Résultats constatés :

- *Débit pour A = 1,7 Mbps*
- *Débit pour B = 4,6 Mbps*
- *Débit total = 6,3 Mbps, ce qui correspond à 31% du débit attendu.*

Indicateur :

- *Respect de la politique: Il apparaît clairement que la bande passante inutilisée est très peu exploitée par l'option borrow dans notre configuration.*

3.3.1.3.5 Test n° 5

Résultats attendus :

- *Débit pour A = 20 Mbps*

Résultats constatés :

- *Débit pour A = 6,9 Mbps ce qui correspond à 34% du débit attendu.*

Indicateur :

Respect de la politique: Ici encore, la bande passante libre dans les groupes 1 et 2 est très faiblement récupérée pour A.

3.3.1.4 Interprétation des résultats

Le problème le plus significatif semble être la faible efficacité de l'option borrow, qui fait que la bande passante inutilisée est très lentement réaffectée aux clients actifs. Le problème est particulièrement visible sur les tests 4 et 5.

De plus, le partage du débit entre les clients A et B n'est pas parfaitement équilibré. Toutefois, PF propose des options qui permettraient probablement de corriger cette situation.

3.3.2. Solution sous FreeBSD

3.3.2.1 Installation de FreeBSD 5.3 et configuration préliminaire

Après une installation minimale de FreeBSD 5.3, nous avons procédé aux étapes suivantes dans la configuration de la machine :

3.3.2.1.1 Recompilation du noyau

L'objectif est d'obtenir un noyau minimal, ainsi que d'inclure IPFW et le BRIDGE au noyau. Bien que ces opérations ne soient pas absolument nécessaires (IPFW et le BRIDGE peuvent être lancés sous la forme de modules chargeables au démarrage du système), cette pratique est relativement courante et c'est pour cette raison que nous avons choisi de vous la présenter.

Les lignes à ajouter au noyau dépendent sont les suivantes :

- *IPFW – DUMMYNET*

  La configuration du noyau FreeBSD pour IPFW-DUMMYNET consiste à ajouter les options suivantes :

```
# Ajout d'IPFW au noyau
options         IPFIREWALL
# Activation de dummynet
options         DUMMYNET
# DUMMYNET : delai de propagation, en ms
options         HZ=1000
```

- *BRIDGE*

  Une ligne suffit pour inclure le BRIDGE au noyau :

```
options         BRIDGE
```

3.3.2.1.2 Configuration du bridge

Il reste à activer et à configurer le BRIDGE entre les deux interfaces. Sous FreeBSD, le bridge est géré par des clusters : le pont est effectif uniquement sur les interfaces déclarées dans le même cluster.

Dans notre cas, il faut ajouter les lignes suivantes à */etc/sysctl.conf* :

```
# Activation du bridge
net.ether.link.bridge.enable=1
# Specification des interfaces sur le cluster 1 : rl0 et rl1
net.ether.link.bridge.config='rl0,rl1:1'
```

3.3.2.2 IPFW- DUMMYNET

3.3.2.2.1 Configuration

Après la configuration préliminaire, il faut ajouter la ligne suivante à */etc/sysctl.conf* :

```
# Activation de IPFW sur le bridge
net.link.ether.bridge.ipfw=1
```

Bien que le module IPFW soit inclus dans le noyau, il est nécessaire de l'activer au démarrage de la machine et de préciser l'emplacement de son fichier de configuration.

Pour cela, on ajoute les lignes suivantes à */etc/rc.conf* :

```
# Activation d'IPFW
firewall_enable="YES"
# Emplacement du fichier de configuration
firewall_script="/etc/firewall.conf"
```

Voici le contenu du fichier de configuration :

```
# Configuration des pipes et des queues
ipfw pipe 1 config bw 20Mbit/s
ipfw pipe 2 config bw 20Mbit/s
ipfw queue 1 config pipe 2 weight 1
ipfw queue 2 config pipe 2 weight 3
# Redirection du trafic
ipfw add pipe 1 tcp from $serveur 20 to $client_D via rl1
ipfw add queue 1 tcp from $serveur 20 to $client_C via rl1
ipfw add queue 2 tcp from $serveur 20 to any via rl1
ipfw add allow ip from any to any
```

3.3.2.3 Tests

Les résultats des tests sont présentés en annexe 2.

3.3.2.3.1 Test n° 1

Résultat attendu :

- *Débit pour C = 20 Mbps*

Résultat constaté :

- *Débit pour C = 17,76 Mbps, en-têtes TCP/IP non comprises*

Indicateur :

- *Respect de la politique : La limite semble être respectée.*

3.3.2.3.2 Test n° 2

Résultats attendus :

- *Débit pour A = 5 Mbps*
- *Débit pour C = 15 Mbps*

Résultats constatés :

- *Débit pour A = 4,8 Mbps*
- *Débit pour C = 13,6 Mbps, en-têtes TCP/IP non comprises*

Indicateur :

- *Respect de la politique : Les limites semblent être respectées.*

3.3.2.3.3 Test n° 3

Résultats attendus :

- *Débit pour A = 2,5 Mbps*
- *Débit pour B = 2,5 Mbps*

- *Débit pour C = 15 Mbps*
- *Débit pour D = 20 Mbps*

Résultats constatés :

- *Débit pour A = 2 Mbps*
- *Débit pour B = 3 Mbps*
- *Débit pour C = 13,6 Mbps, en-têtes TCP/IP non comprises*
- *Débit pour D = 17,76 Mbps, en-têtes TCP/IP non comprises*

Indicateur :

- *Respect de la politique : Les limites fixes semblent être respectées mais le partage du débit dans le groupe 2 n'est pas parfaitement équilibré.*

3.3.2.3.4 Test n° 4

Résultats attendus :

- *Débit pour A = 10 Mbps*
- *Débit pour B = 10 Mbps*

Résultats constatés :

- *Débit pour A = 6,88 Mbps*
- *Débit pour B = 12,8 Mbps*

Indicateur :

- *Respect de la politique : La limite du débit du groupe 1 est respectée mais le partage du débit dans le groupe 2 n'est pas équilibré.*

3.3.2.3.5 Test n° 5

Résultats attendus :

- *Débit pour A = 20 Mbps*

Résultats constatés :

- *Débit pour A = 18,4 Mbps*

Indicateur :

Respect de la politique : La limite du groupe 1 est respectée, le débit est relativement bien utilisé dans les groupes 1et 2 lorsque certains clients sont inactifs.

3.3.2.4 Interprétation des résultats

Les limites de débits fixés sur les *pipes* sont dans l'ensemble bien respectées et la bande passante généralement bien utilisée.

Toutefois, il apparaît que, dans les tests, les *queues* n'ont pas assuré un partage très équitable du débit (cf. tests n° 3 et 4). Néanmoins, un paramétrage plus affiné de celles-ci, notamment par l'utilisation des délais, permettrait probablement de corriger ce problème.

4. Conclusion

4.1. Caractéristiques principales des solutions présentées

L'étude des caractéristiques des différentes solutions a révélé les éléments suivants :

- *Toutes les solutions Open Source étudiées tirent une grande flexibilité de l'utilisation des règles de filtrage des firewalls impliqués dans le traffic shaping.*

  Ceux-ci permettent en effet de classer les paquets en prenant en compte, non seulement des paramètres classiques tels que les adresses d'émission et de réception, le type de protocole utilisé (comme le proposent les solutions Windows), mais aussi des paramètres bien plus pointus tels que les flags présents dans les en-têtes TCP/IP.

- *Les possibilités de gestion des queues offertes par la solution Linux sont particulièrement nombreuses. Cependant, il semble que seule une partie d'entre elles répondent au besoin du traffic shaping : PRIQ et CBQ.*

  On utilisera donc généralement les mêmes types de stratégies de gestion des queues sous Linux et sous BSD.

- *Les trois solutions Open Source détaillées dans ce document prennent en charge l'option Random Early Detection (RED).*

  Cette option peut s'avérer particulièrement utile pour éviter les congestions brutales.

4.2. Fonctionnement des solutions testées

Les tests ont permis de révéler les éléments suivants concernant PF+ALTQ et IPFW+DUMMYNET :

- *Les limites de débit n'ont jamais été dépassées.*
- *Le partage équitable du débit entre plusieurs clients d'une même queue nécessite un paramétrage plus fin que celui que nous avons réalisé.*

4.3. Comparaison des performances

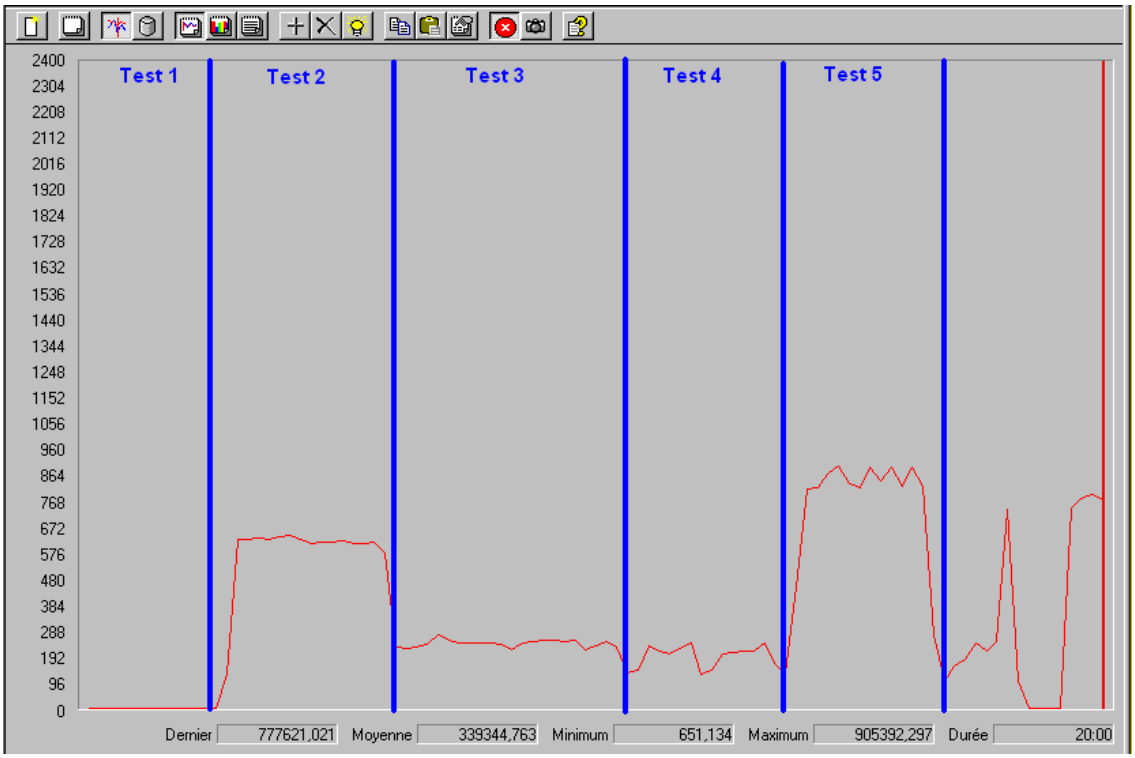
Les tests ont permis de révéler les éléments suivants concernant les performances respectives de PF+ALTQ et d'IPFW+DUMMYNET :

- *Lorsqu'une queue est inutilisée, le débit qui lui est attribué, s'il peut être réutilisé par une autre queue, est moins bien exploité avec PF + ALTQ qu'avec IPFW+DUMMYNET.*
- *De manière générale, nous avons constaté lors de nos tests que les débits étaient globalement plus soutenus avec IPFW + DUMMYNET qu'avec PF + ALTQ.*

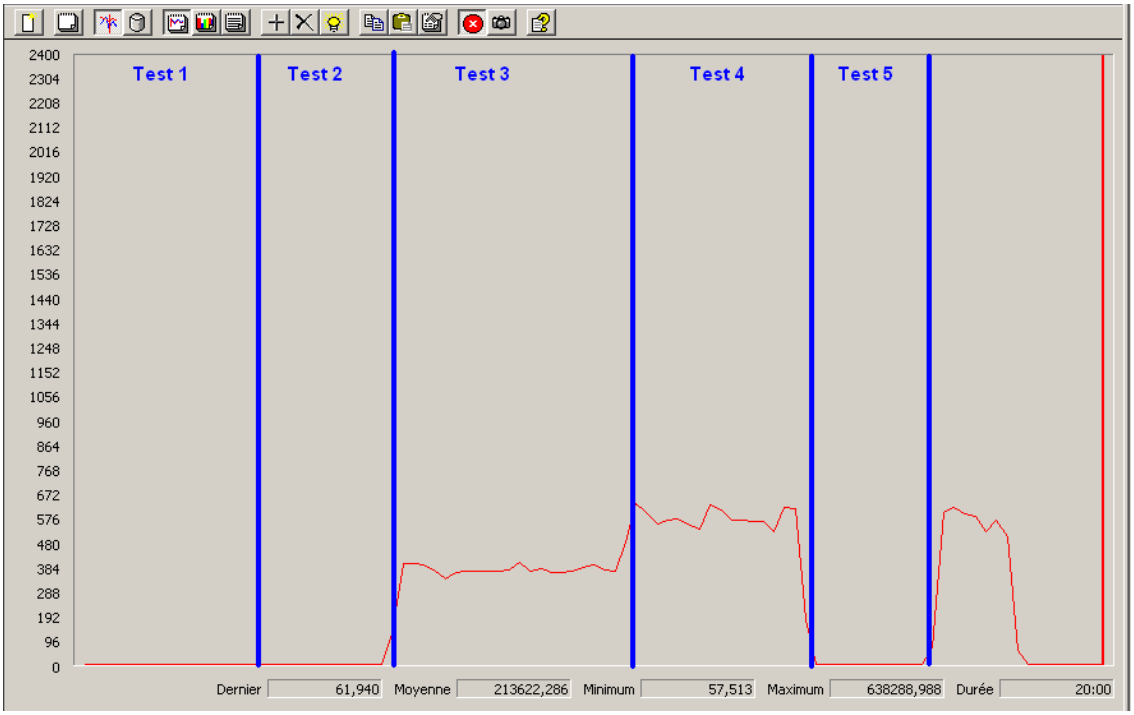
5. Annexes

5.1. Annexe 1 : résultats des tests sous OpenBSD

5.1.1. Machine A



5.1.2. Machine B



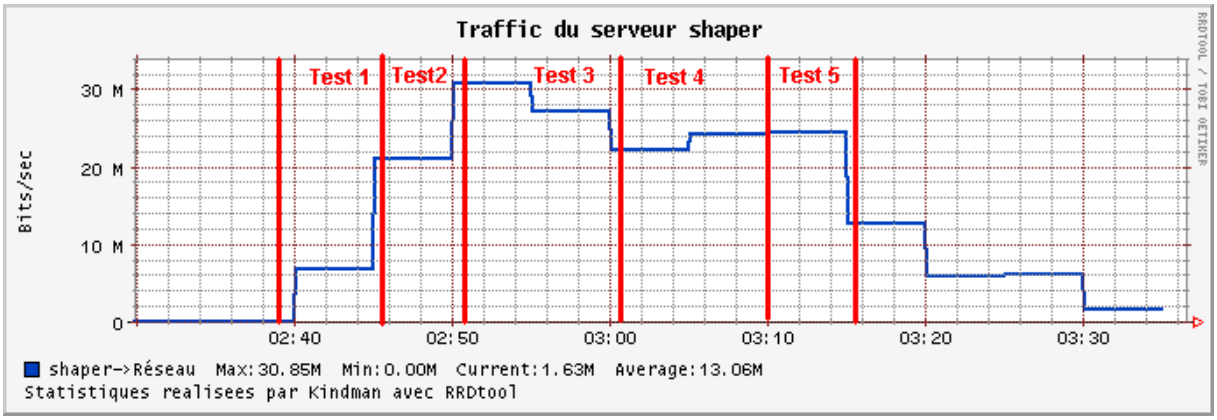
5.1.3. Machine C

```
Test 1
'Fichier' at 63011168 (2%) 1.91M/s eta:17m [Receiving data]

Test 2
'Fichier' at 406866280 (18%) 1.65M/s eta:15m [Receiving data]

Test 3
'Fichier' at 893201696 (41%) 1.67M/s eta:12m [Receiving data]
```

5.1.4. Machine D

```
Test 3
FICHIER: 7%          |   244 MB    2.18 MB/s    25:07 ETA
FICHIER: 36%         |  1294 MB    2.05 MB/s    18:12 ETA
```

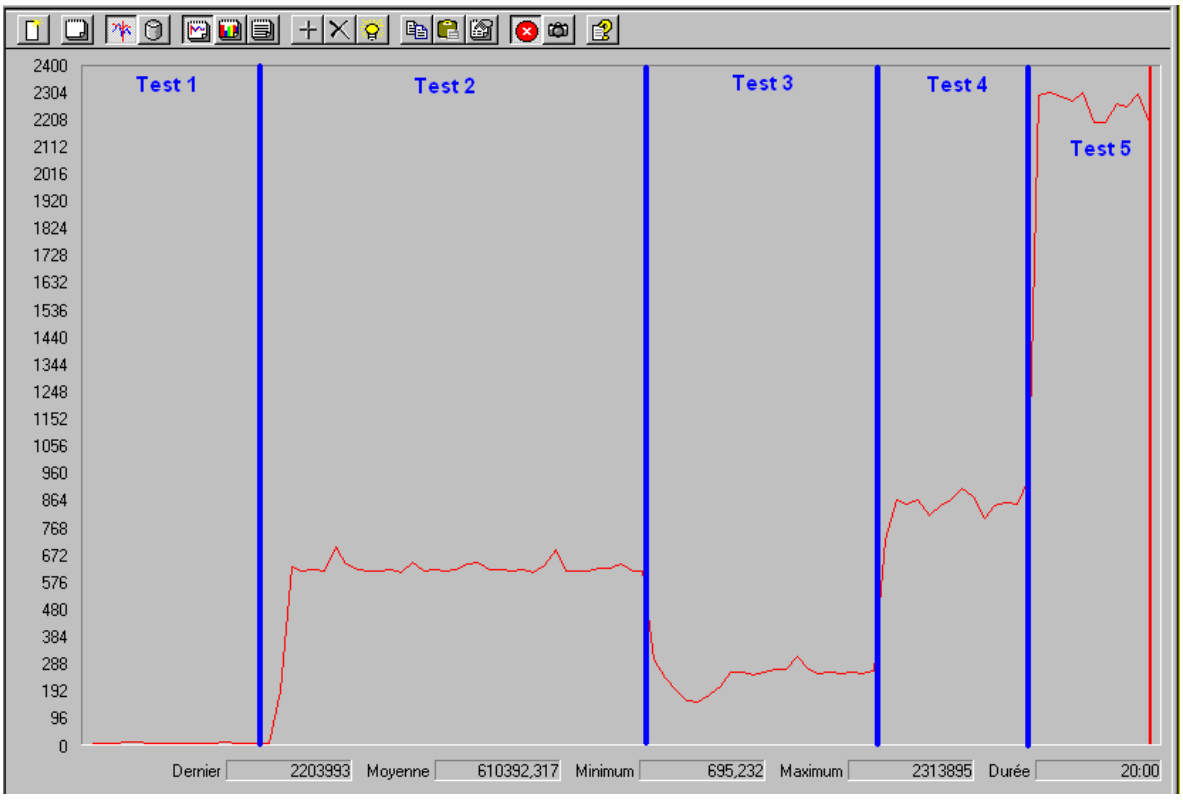
5.1.5. Serveur



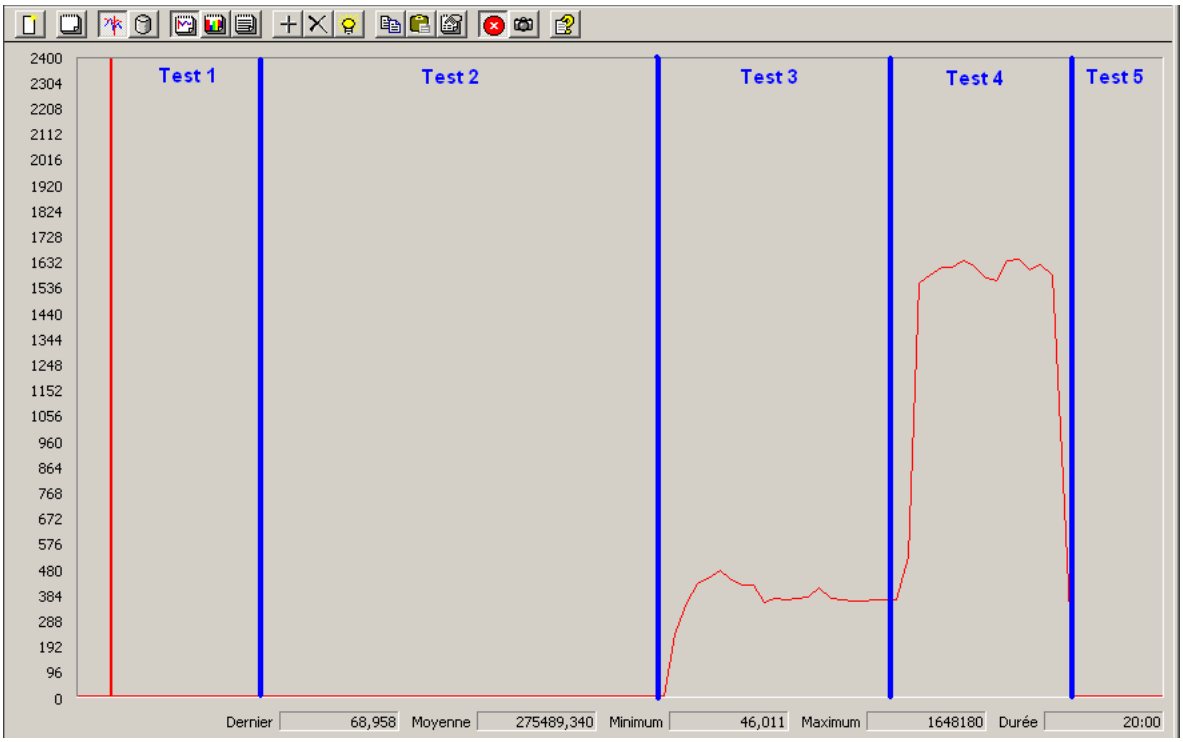
Nb : Ce graphique est généré à partir du trafic moyen sur des intervalles de cinq minutes, ce qui explique sa faible précision.

5.2. Annexe 2 : résultats des tests sous FreeBSD

5.2.1. Machine A



5.2.2. Machine B



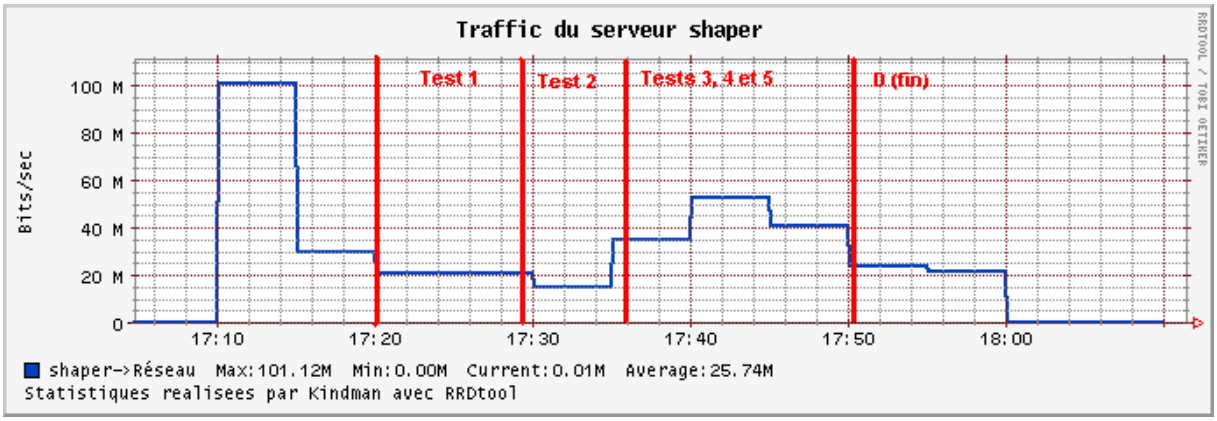
5.2.3. Machine C

```
Test 1
'Fichier' at 289439272 (13%) 2.22M/s eta:13m [Receiving data]
Test 2
'Fichier' at 1070072000 (49%) 1.70M/s eta:9m [Receiving data]

Test 3
'Fichier' at 1768894530 (72%) 1.70M/s eta:3m [Receiving data]
```

5.2.4. Machine D

```
Test 3
Fichier: 19%   675 MB    2.22 MB/s    21:25 ETA
```

5.2.5. Serveur



Nb : Ce graphique est généré à partir du trafic moyen sur des intervalles de cinq minutes, ce qui explique sa faible précision.