2008

UFR Ingénieurs 2000

Vivien Boistuaud
Florence Fraux
Julien Herr

QOS – GESTION DE LA QUALITE DE SERVICE SOUS RESEAUX IP AVEC LINUX

Ce document a été réalisé par V. Boistuaud, F. Fraux et J. Herr dans le cadre des travaux pratiques de Qualité de Service coordonnés par Hakim Badis, au sein de l'UFR Ingénieurs 2000 de l'Université de Marne-la-vallée.

Table des matières

Introduction

Aujourd'hui, les flux transitant sur les réseaux IP sont de types très variables, allant du téléchargement P2P à la téléphonie et la vidéotransmission. De par leurs caractéristiques propres, on imagine facilement que ces flux n'ont pas la même importance pour l'utilisateur, et qu'ils ne devraient donc pas avoir la même priorité de transit et de traitement dans les réseaux IP.

De nombreuses technologies de gestion de la qualité de service ont été développées pour les réseaux IP, comme IntServ et DiffServ, qui servent tous deux à simuler une commutation de cellule sur un chemin garantissant la qualité de service.

Cependant, ces technologies sont souvent utilisables uniquement au sein des réseaux de fournisseurs de service internet (ISP) ou, pour les entreprises de grande taille, au sein de l'entreprise elle-même. Il est rare que ces technologies soient accessibles de bout en bout sur un réseau comme internet.

Le but de cette séance de TP, dirigée par Hakim Badis, était de mettre en œuvre une gestion de la qualité de service au niveau d'un système d'exploitation GNU/Linux. L'intérêt de ce procédé est de permettre de configurer des routeurs logiciels basés sur le système d'exploitation Linux pour qu'ils puissent simuler une gestion de la qualité de service entre un réseau local (LAN) et un réseau internet (WAN).

Ce TP se déroule en deux phases : dans un premier temps, l'objectif était de se familiariser avec les outils de mesure de transit IP tandis que, dans un second temps, l'objectif était de mettre en œuvre une politique de gestion de la qualité de service au niveau des couches réseaux du noyau Linux, afin de pouvoir garantir la priorité de certains flux par rapport à d'autres.

1. Présentation générale du TP

Cette section regroupe les informations utiles au lecteur pour comprendre la démarche suivie lors de ce TP, ainsi que la matériel utilisé afin que les manipulations puissent être reproduites.

1. Matériel utilisé

Pour mettre en œuvre ce TP, nous avons utilisé le matériel suivant :

- 3 ordinateurs équipés de Debian GNU/Linux, distribution Etch (la version la plus stable lors de l'écriture de ce document) ;
- Les logiciels `mgen`, `trpr`, `gnuplot`, `wireshark` installés sur la machine (installation détaillée ci-après),
- Les modules noyaux « *CBQ packet scheduler* », « *Stochastic Fairness Queueing* » et « *Hierarchical Token Bucket* » qui doivent être compilés comme modules noyau. Ils sont dans la rubrique « *QoS and/or fair queuing* » de la configuration réseau du noyau Linux 2.6.
- 1 commutateur Hewlett Packard ProCurve 2626, 16 ports 10/100 Base-TX,
- 3 câbles réseaux RJ-45 Ethernet Classe 3 droits,

Notez que les logiciels nécessaires fonctionnent également sous Windows. Cependant, le but de ce TP étant de mettre en évidence la gestion de la qualité de service sous Linux, il n'est possible d'utiliser des machines Windows pour la seconde partie de ce TP, excepté pour la réception de paquets. Notez toutefois que la première partie peut tout à fait être réalisée sous Microsoft Windows.

Dans la section suivante, nous détaillons le montage et la façon dont l'ensemble de ces matériels interagissent ensemble.

2. Montage initialement utilisé

Le montage initialement réalisé est le suivant :

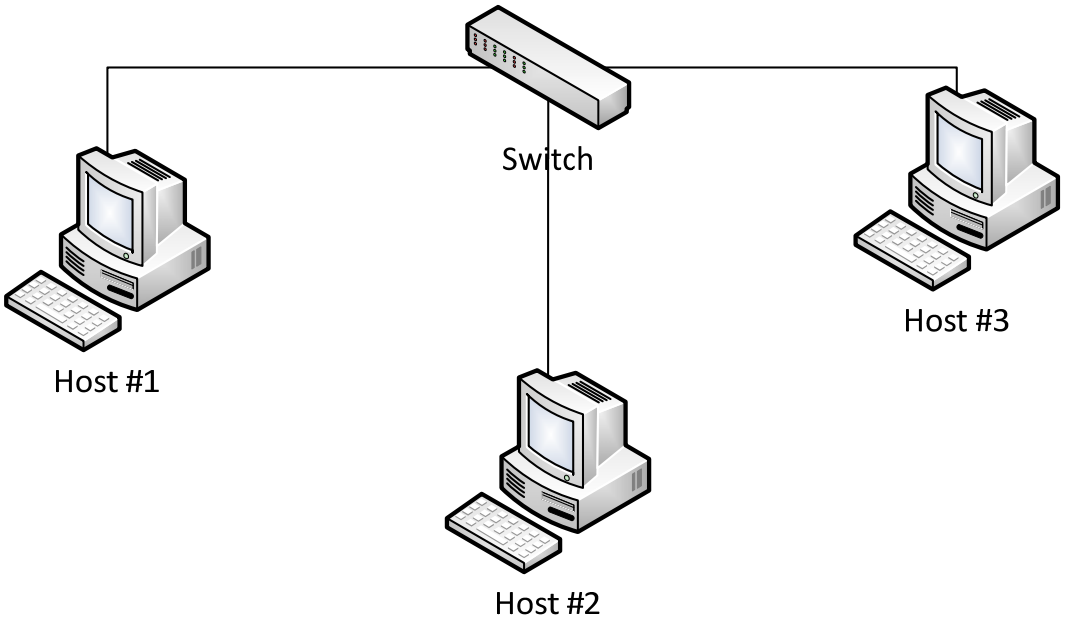

Figure 1 - Configuration initiale du réseau

Le plan d'adressage a peu d'importance dans ce TP. Aussi, nous avons décidé d'utiliser le réseau 192.168.0.0/24 pour notre réseau local, l'octet de poids faible correspondant au numéro d'hôte de chaque machine.

Ainsi, le plan d'adressage retenu est le suivant :

Machine	Adresse IPv4 et Masque de sous réseau
Hôte #1	Adresse : 192.168.0.1 Sous réseau : 255.255.255.0
Hôte #2	Adresse : 192.168.0.2 Sous réseau : 255.255.255.0
Hôte #3	Adresse : 192.168.0.3 Sous réseau : 255.255.255.0

Figure 2 - Plan d'adressage du réseau

Pour configurer cela, sur chaque machine, il suffit d'utiliser la commande :

```
host1:~# ifconfig eth0 address 192.168.0.1 netmask 255.255.255.0 up

host2:~# ifconfig eth1 address 192.168.0.2 netmask 255.255.255.0 up

host3:~# ifconfig eth1 address 192.168.0.3 netmask 255.255.255.0 up
```

Note: l'hôte 1 utilise son interface `eth0` *dans la mesure où l'interface* `eth1` *n'était pas configurée correctement sur la machine.*

3. Installation des outils nécessaires

Plusieurs outils sont nécessaires à une bonne appréhension de ce TP. Certaines peuvent être installées par le système avancé de gestion de paquets de Debian GNU/Linux.

1. Wireshark (évolution d'Ethereal)

Cet outil permet d'épier (sniffer) les paquets échangés sur un réseau et d'analyser leur contenu de façon détaillée à l'aide d'un certain nombre de dissecteurs. Ce programme peut s'installer à l'aide d'APT :

```
hostX:~# apt-get install wireshark
```

Note : Sous les versions précédentes de Debian (Sarge par exemple), vous devrez installer le paquetage `ethereal` *à la place.*

2. Gnuplot

Cet outil permet de dessiner rapidement des courbes à partir de séries de chiffres. Il est extrêmement simple d'utilisation et permet d'interpréter rapidement des résultats parfois complexes. Ce programme peut s'installer à l'aide d'APT :

```
hostX:~# apt-get install gnuplot
```

3. Mgen

Mgen est un outil de génération de trafic réseau simple d'utilisation. Il permet de générer des émissions constantes de paquets UDP et TCP (mode CBR), mais également de générer des trafics aléatoires variables selon des lois probabilistes, comme les lois exponentielles et de Poisson. Il permet également de créer rapidement un écouteur permettant de recevoir ces trafics.

Il n'est pas disponible sous forme précompilé sous Debian GNU/Linux. Il est donc recommandé de le télécharger et le compiler soit même.

La dernière version (beta) disponible à la date d'écriture de ce rapport est la 4.2 beta 6. On peut l'installer facilement comme suit :

```
# wget http://downloads.pf.itd.nrl.navy.mil/mgen/src-mgen-4.2b6.tgz
# tar xzf src-mgen-4.2b6.tgz
# cd mgen-4.2b6/unix
# make -f Makefile.linux mgen
```

L'exécutable `mgen` est alors stockée directement dans le dossier `mgen-4.2b6/unix`. Pour qu'il soit disponible dans votre path, vous pouvez soit le copier dans le dossier `/usr/local/bin`, soit ajouter ce répertoire à votre path.

Remarqué : lorsque vous créez un fichier d'auto-configuration pour mgen, notamment sous Windows, il est primordial de laisser une ligne vide à la fin du fichier pour que la dernière commande soit exécutée.

4. Trpr

Trpr est un outil qui permet de traiter les données recueillies par Mgen afin de les transformer, entre autre, dans des formats compatibles avec Gnuplot. Il permet également de générer des statistiques intéressantes comme des moyennes de jigue, des moyennes de débit ou des moyennes sur le taux de perte de paquets UDP.

Aucune version précompilée n'est disponible, et son code source tient en un seul fichier C++. Pour le compiler, il suffit de faire comme suit (pour la version 2.0 beta 2) :

```
# wget http://downloads.pf.itd.nrl.navy.mil/proteantools/src-trpr-2.0b2.tgz
# tar xzf src-trpr-2.0b2.tgz
# cd TRPR
# g++ -o trpr trpr.cpp –lm
```

Le switch –lm permet de lancer également le linker, de sorte que l'objet compile soit linké et mis au format standard Linux ELF.

De même que pour `mgen`, vous devez soit copier `trpr` dans votre répertoire `/usr/local/bin`, soit placer le répertoire `TRPR` dans votre path.

Une fois les outils installés, il est possible de réaliser la première partie de ce TP.

5. Activation de la QoS sous Linux

Pour activer la gestion de la qualité de service (QoS – Quality Of Service) sous Linux, il faut que les options CBQ, SFQ et HTB aient été compilés comme module du noyau. Il faut également les charger à l'aide des commandes :

```
debsrv:~# modprobe sch_cbq
debsrv:~# modprobe sch_sfq
debsrv:~# modprobe sch_htb
```

Cette manipulation est essentielle pour la seconde partie du TP (la section 3 de ce document).

Pour pouvoir utiliser la commande `tc`, qui permet de gérer les files d'attente au niveau du noyau Linux, il faut installer le paquetage debian `iproute` :

```
debsrv:~# apt-get install iproute
```

Enfin, nous auront besoin de pouvoir réduire temporairement le mode de fonctionnement de notre carte réseau de 100Mbps Full-Duplex vers 10Mbps Full-Duplex. Pour ce faire, on peut utiliser plusieurs applications. Dans notre cas, nous avons décidé d'utiliser `mii-tools` qui est déjà installé en standard sous Debian GNU/Linux.

2. Simulation de trafic sans QoS

L'objectif de la simulation sans qualité de service est d'avoir des valeurs de référence. En effet, de cette façon nous pourrons comparer le gain réel de l'utilisation de qualité de service.

De plus, ces mesures nous permettrons d'étudier le seuil d'engorgement du réseau suivant la vitesse et avec ou sans perturbations.

1. Trafic périodique UDP en 100Base-TX Full Duplex

Nous allons tout d'abord prendre un jeu de mesures d'un trafic UDP en 100Base-TX Full Duplex, ce qui représente la norme que nous trouvons habituellement dans n'importe quel réseau local.

1. Mode opératoire

Afin d'avoir un jeu de mesures larges, nous allons varier le nombre de paquet envoyé par seconde (50, 60, 100 et 100) ainsi que la taille de ce dernier (128, 1024, 8192).

Pour cela, depuis la machine source hôte #2, on configure `mgen` pour des émissions comme suit :

```
0.0 ON 1 UDP DST 192.168.0.1/5000 PERIODIC [50.0 128]
30.0 OFF 1

30.0 ON 1 UDP DST 192.168.0.1/5000 PERIODIC [60.0 128]
60.0 OFF 1

60.0 ON 1 UDP DST 192.168.0.1/5000 PERIODIC [100.0 128]
90.0 OFF 1

90.0 ON 1 UDP DST 192.168.0.1/5000 PERIODIC [1000.0 128]
120.0 OFF 1

120.0 ON 1 UDP DST 192.168.0.1/5000 PERIODIC [50.0 1024]
150.0 OFF 1

150.0 ON 1 UDP DST 192.168.0.1/5000 PERIODIC [60.0 1024]
180.0 OFF 1

180.0 ON 1 UDP DST 192.168.0.1/5000 PERIODIC [100.0 1024]
210.0 OFF 1

210.0 ON 1 UDP DST 192.168.0.1/5000 PERIODIC [1000.0 1024]
240.0 OFF 1

240.0 ON 1 UDP DST 192.168.0.1/5000 PERIODIC [50.0 8192]
270.0 OFF 1

270.0 ON 1 UDP DST 192.168.0.1/5000 PERIODIC [60.0 8192]
300.0 OFF 1

300.0 ON 1 UDP DST 192.168.0.1/5000 PERIODIC [100.0 8192]
330.0 OFF 1

330.0 ON 1 UDP DST 192.168.0.1/5000 PERIODIC [1000.0 8192]
360.0 OFF 1
```

Voici la signification du formatage de ces lignes :

- Pour chaque ligne, la première valeur représente l'instant de démarrage, relatif au moment t=0 de démarrage de la commande mgen.
- La seconde valeur ON x précise sur quel processus on souhaite effectuer l'action ;
- UDP DST x.x.x.x/n défini qu'on va envoyer un flux UDP à destination de l'adresse IP x.x.x.x et du port n. ;
- PERIODIC [N M] indique que le flux UDP transmis est périodique, que le paquet est envoyé N fois par seconde (fréquence d'envoi en Hertz) et que chaque paquet mesure M *octets* ;
- OFF x signifie qu'on met fin au processus x existant, par exemple OFF 1 pour arrêter la transmission démarrée par la commande ON 1.

Depuis la machine cible hôte #1 on configure `mgen` pour une réception comme suit (fichier `destination.mgn`) :

```
0.0 LISTEN UDP 5000,5001,5002
```

Cette ligne indique qu'à partir du lancement de l'application (instant t=0) on écoute sur les ports UDP 5000, 5001 et 5002 les paquets entrant.

Il suffit alors de lancer l'écoute des paquets reçus sur les ports5000, 5001 et 5002 sur l'hôte 1 à l'aide de la commande `mgen input destination.mgn` puis de lancer l'émission de paquets sur l'hôte 2 à l'aide de la commande `mgen input source.mgn`.

En redirigeant la sortie de la commande exécutée sur l'hôte 1 dans un fichier, on peut alors traiter le résultat à l'aide de la commande `trpr` comme suit :

```
trpr input mesures.dat history 800 window 30 mgen real > mesures-rate-data.dat
trpr input mesures.dat history 800 window 30 mgen real loss > mesures-loss-
data.dat
trpr input mesures.dat history 800 window 30 mgen real interarrival > mesures-
gigue-data.dat
```

La première ligne permet de récupérer le débit réel. La seconde ligne permet de récupérer la quantité de pertes des paquets, identifiée à l'aide des numéros de séquence des paquets.

Enfin, la dernière ligne permet de mesurer la gigue, ou plutôt l'intervalle de temps type entre la réception de deux paquets, d'où on peut déduire la gigue comme ceci est expliqué plus loin dans ce document.

2. Résultats de la manipulation

Ci-après le tableau résumant les mesures prises. Notez que la taille des paquets est mesurée en octets, alors que le débit est mesuré en kbps (avec la convention 1kbps = 1024 bps, puissances binaires).

Paquet/s	Taille paquet	Débit mesuré (kbps)	Débit attendu (kbps)	Perte mesurée	Gigue mesurée (ms)
50	128	50,00	50,00	0,00 %	0,00
60	128	60,00	60,00	0,00 %	0,00
100	128	99,97	100,00	0,00 %	0,00
1000	128	998,97	1 000,00	0,00 %	0,00
50	1024	400,00	400,00	0,00 %	0,00
60	1024	480,00	480,00	0,00 %	0,00
100	1024	799,47	800,00	0,00 %	0,00
1000	1024	7 896,53	8 000,00	1,10 %	0,42
50	8192	3 200,00	3 200,00	0,00 %	0,01
60	8192	3 840,00	3840,00	0,00 %	0,01
100	8192	6 393,60	6 400,00	0,00 %	0,01
1000	8192	62 730,67	64 000,00	1,76 %	0,47

Figure 3 - Moyennes de débit/perte/gigue pour du 100Base-TX FD sans perturbation

3. Analyses et Explications

On remarque, d'emblée, certaines variations de débit sans pour autant que des pertes ne soient mesurées : il s'agit à priori d'erreurs d'approximation liées à `trpr` et celles-ci ne seront donc pas analysées outre mesure : il aurait fallut garantir l'isolation des mesures en effectuant une capture par flux.

Débit

Tout d'abord, plus on envoie de paquet, et plus le débit est important. Chose logique, car on envoie plus en envoie de paquet par seconde et donc de bits.

Nous avons ensuite effectué les mêmes mesures avec des paquets contenant plus d'informations, notamment des paquets de 1024 et 8192 octets. La constatation précédente se vérifie également dans ces exemples. On distinguera également que plus la taille des paquets est importante, plus le débit sera important, pour la même raison que précédemment.

Il est à noter que la mesure effectuée avec des paquets de 8192 octets en envoyant 1000 paquets par seconde est intéressante. En effet, bien que le tableau donne une valeur moyenne, lors de l'expérimentation, nous avons observé que le débit variait

entre 60 000 et 65 000. Cette variation est liée à la file d'attente du switch qui se remplie. Une fois que cette dernière est pleine, le débit diminue. Celui-ci pourra à nouveau augmenter lorsqu'il y aura de la place dans la file.

Perte

Dans l'ensemble des premières mesures, on ne constate aucune perte. En effet, le débit est supporté donc aucun paquet n'est perdu. Par contre, les mesures effectuées avec des paquets de 1024 et 8192 en envoyant 1000 paquets par seconde, montrent des pertes. Tout comme pour le débit, cela s'explique en observant les files d'attentes. En effet, lorsque ces dernières sont pleines, le switch détruit les autres paquets qui arrivent. Ce mécanisme explique donc les pertes détectées.

Toutefois, les commutateurs étant des commutateurs Ethernet Gigabit et notre machine l'exploitant en 100Mbps Full-Duplex, ceci n'aurait pas dû se produire. Il se peut aussi que le problème vienne des files d'attente des matériels des machines (car une surcharge réseau étant donnée la puissance des machines serait plausible).

Gigue

La gigue correspond à une variation de la fréquence d'un signal par rapport à ce qui est attendu. Par exemple, si on émet 50 paquets par secondes, on s'attend à recevoir un paquet toutes les 20 ms en moyenne. Si on reçoit en moyenne un paquet toutes les 22ms, alors on a une gigue de 2 ms. En anglais, gigue se dit *jitter*.

Le paramètre qu'est la gigue est très important dans les applications qui ne nécessitent pas de retard comme, par exemple, la VoIP. En effet, si la gigue est trop importante, le service est dégradé. On estime qu'une gigue supérieure à 100ms dégrade trop le service et n'est pas acceptable.

A partir des captures de paquets que nous avons réalisées, on remarque que deux éléments peuvent faire varier la gigue : la perte de paquet augmente nécessairement la valeur de la gigue, et la taille du paquet peut également la faire augmenter car les paquets sont, potentiellement, plus longs à traiter à la fois par le commutateur et par les machines. Ce phénomène ne nous est en effet apparu qu'avec des paquets de taille de 8092 octets.

2. Trafic périodique UDP en 10Base-T Full Duplex

Nous allons maintenant procéder aux mêmes études sur un support physique plus faible, à savoir 10Base-T Full Duplex.

1. Mode opératoire

Le mode opératoire est le même que pour le trafic périodique UDP en 100Base-TX Full Duplex. Toutefois, nous avons forcé la carte réseau à réduire son débit. Pour cela, nous nous servons de l'outil `mii-tool`.

Une fois installé, il suffit de spécifier le média que l'on souhaite avoir. Dans notre cas, pour 10Base-T Full Duplex, il faut taper la commande :

```
mii-tool -F 10baseT-FD eth0
```

2. Résultats de la manipulation

Le résultat le plus intéressant de cette manipulation est le comportement du commutateur et de la machine en cas de saturation, donc uniquement dans le cas où on envoit 1000 paquets de 8192 octets à chaque seconde.

Nous obtenons les résultats suivants :

- Débit : 7485,87 kbps
- Pertes : 88,8 %
- Gigue : 7,55 ms (8,55ms au lieu des 1ms attendus)

3. Explications

Comme cela était prévisible, le commutateur sature lors de la tentative d'envoi de paquets au client. En effet, le commutateur reçoit des données à transmettre à une vitesse de 64 000 kbps, qu'il doit faire transiter sur un lien ayant un débit maximal théorique de 10 240 kbps. Ceci se traduit donc par d'importantes pertes, car la file d'attente du commutateur sature et il jette les paquets non transmis.

De plus, on remarque que le débit réel du lien en10 base-T Full Duplex est en fait de 7485 kbps, logiquement inférieur au débit maximal théorique. On peut en conclure que ce lien ne peut supporter une charge supérieure à 75% de sa capacité théorique.

Enfin, le taux de perte est logique, puisque le débit est réduit de 88% par rapport à ce qui était attendu. De même, la jigue est en adéquation avec ces pertes.

3. Trafic périodique UDP en 100Base-TX Full Duplex avec perturbation

Nous allons maintenant continuer l'étude en générant des perturbations sur le réseau.

4. Mode opératoire

Le mode opératoire reste le même que dans la manipulation précéent. Nous avons juste besoin de rajouter un poste émetteur, ayant pour adresse IP 192.168.0.3, qui viendra perturber l'émission de paquets du premier émetteur en envoyant des paquets vers la machine qui mesure la réception des paquets.

Le script `mgen` d'émission pour le second émetteur est le suivant :

```
180.0 ON 3 UDP DST 192.168.0.1/5004 PERIODIC [1800.0 4096]
210.0 OFF 3
```

Remarque: nous avons envoyé les parquets vers le port 5004 car le but est uniquement de saturer la liaison et non de faire des statistiques sur les paquets causant la perturbation.

5. Résultats de la manipulation

Nous avons obtenus les résultats suivant selon les données fournies par mgen :

- Débit : 49 306 Kbps
- Pertes : 22,3 %
- Gigue : 0,3 ms

6. Explications

On constate une perturbation qui se traduit en une chute du débit. En effet, le débit de la liaison est limité à 100Mbps or les flux que nous envoyons sature ce dernier. La machine ne peut pas recevoir les deux flux simultanément avec le débit défini. La saturation s'effectue, plus précisément, au niveau du commutateur qui détecte que le lien est à 100Mbps et que tous les paquets ne peuvent être envoyés.

A partir de ces informations on peut déduire que la file d'attente du commutateur, qui est ici saturée, est équipée d'un algorithme équitable de gestion de files, type SFQ ou un algorithme équivalent. En effet, les pertes semblent réparties équitablement entre le flux auquel nous nous sommes intéressés et le flux de perturbation, les deux ayant la même importance pour le commutateur.

3. Mise en place d'une stratégie de QoS sous Linux

La gestion de la qualité de service permet de différencier les types de trafic (audio, vidéo, voix, types de données…) afin de leur assigner des règles de traitement spécifiques. Par exemple, dans le cadre d'un réseau exploité pour du trafic Web et pour de la VoIP, on veut pouvoir favoriser le trafic vocal qui possède des contraintes de temps réel et de débit minimum, par rapport au trafic web moins important.

Dans notre cas, nous allons différencier les trafics UDP sortants de la machine à destination des ports 5000, 5001 et 5002.

1. Principes de la stratégie de QoS

Pour atteindre cet objectif, il faut tout d'abord expliquer la façon dont la qualité de service est gérée sous Linux.

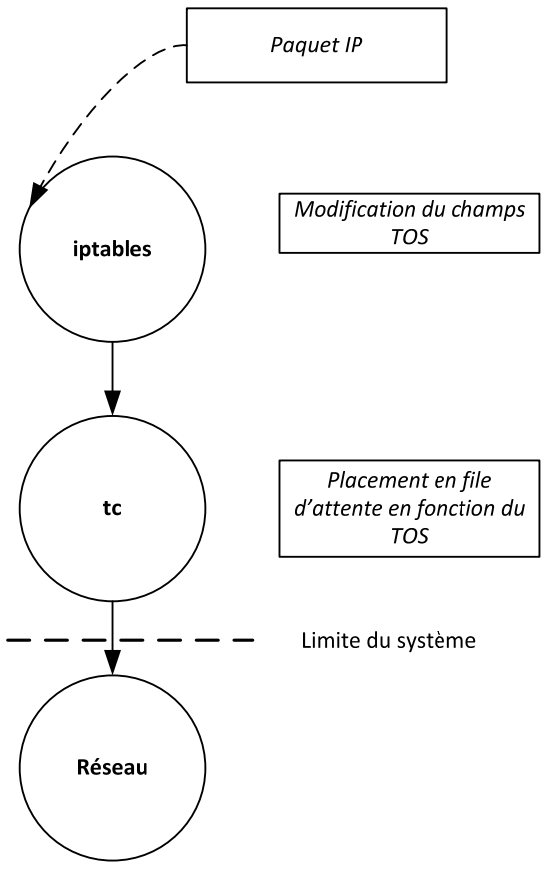

Figure 4 - Traitement d'un paquet pour la QoS

Lorsqu'un paquet IP est envoyé depuis une application, il transite par la table MANGLE d'`iptables`. A ce moment, il est possible d'effectuer des modifications sur le paquet IP, comme une réécriture d'adresse, ou la modification d'un champ IPv4. Pour la gestion de la QoS, on modifie la valeur du champ TOS du paquet IPv4.

Dans notre cas, nous affecterons le TOS 1 aux paquets destinés au port 5000, le TOS 2 pour le port 5001, le TOS 3 pour le port 5002.

Lorsque les modules de QoS sont chargés dans le noyau Linux (voir section 1 du présent document), le paquet est ensuite traité par le module de gestion de la QoS.

Ce module permet alors de gérer les paquets dans des séries de file d'attente personnalisables, en fonction de la valeur de leur champ TOS.

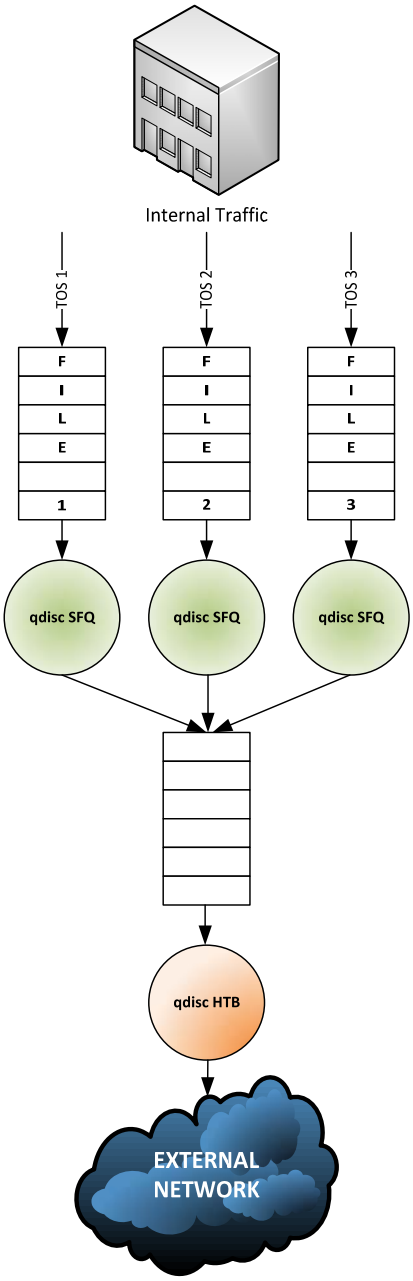

Figure 5 - Files d'attente à mettre en place

Dans notre cas, nous allons mettre en place 4 files d'attente configurées selon les modalités suivantes :

- Pour le TOS 1, une file de discipline SFQ[1] avec un trafic limité à 10 kb/s et un trafic normal à 10 kb/s.
- Pour le TOS 2, une file de discipline SFQ avec un trafic limité à 1 Mb/s et un trafic normal à 200 kb/s.
- Pour le TOS 3, une file de discipline SFQ avec un trafic limité à 1 Mb/s et un trafic normal à 700 kb/s.
- L'ensemble de ces files sont limitées à 1 Mb/s au total, toutes files confondues La discipline de cette file de sortie est HTB[2].

Par conséquent, le trafic sortant de la machine sera limité, sur les ports 5000, 5001 et 5002 à 1 Mb/s au total, et le traitement par files d'attentes séparées permettra de répartir le trafic selon son importance.

Afin de vérifier le fonctionnement de ces séries de files d'attente, et la façon dont les paquets sont priorisés en cas d'engorgement, il faut tout d'abord créer un script de configuration automatique pour configurer `iptables` et `tc`. C'est ce que la section suivante présente plus en avant.

[1] SFQ : Stochastic Fairness Queuing
[2] HTB : Hierarchical Token Buckets

2. Script de mise en œuvre automatique

Le script que nous avons rédigé pour mettre en place nos files d'attente est le suivant :

```
#!/bin/bash

# Setup classes

tc qdisc del dev eth1 root
tc qdisc add dev eth1 root handle 10: htb
tc class add dev eth1 parent 10: classid 10:1 htb rate 1048576
tc class add dev eth1 parent 10:1 classid 10:10 htb rate 10240 ceil 10240
tc class add dev eth1 parent 10:1 classid 10:20 htb rate 204800 ceil 1048576
tc class add dev eth1 parent 10:1 classid 10:30 htb rate 716800 ceil 1048576

# Enable equity for queues

tc qdisc add dev eth1 parent 10:10 handle 40: sfq
tc qdisc add dev eth1 parent 10:20 handle 50: sfq
tc qdisc add dev eth1 parent 10:30 handle 60: sfq

# Reset the mangle table type

iptables -t mangle -F

iptables -t mangle -A OUTPUT -o eth1 -p udp --dport 5000 -j MARK --set-mark 1
iptables -t mangle -A OUTPUT -o eth1 -p udp --dport 5001 -j MARK --set-mark 2
iptables -t mangle -A OUTPUT -o eth1 -p udp --dport 5002 -j MARK --set-mark 3

tc filter add dev eth1 protocol ip handle 1 fw flowid 10:10
tc filter add dev eth1 protocol ip handle 2 fw flowid 10:20
tc filter add dev eth1 protocol ip handle 3 fw flowid 10:30

# Display the result of the tc operations
tc -s -d class ls dev eth1

# Display the results of the iptables operations
iptables -t mangle -L
```

Voici tout d'abord le descriptif des commandes « tc » utilisées. Cette commande est utilisée pour configurer le contrôle de trafic dans le noyau Linux (kernel).

« qdisc » signifie « queueing discipline ». Lorsque le noyau doit envoyer un paquet à une interface (eth1 par exemple), les paquets sont empilés dans le « qdisc » configuré pour l'interface. De suite après le noyau essaye de récupérer autant de paquets que possible, pour les donner au driver d'adaptation réseau. Un « qdisc » simple est une pure file « FIFO ».

Certains « qdisc » peuvent contenir des classes (mot clé « class » au lieu de « qdisc »), qui contiennent davantage de « qdisc ». Le trafic peut de ce fait être empilé dans l'un de ces « qdisc » internes.

« filter » est utilisé par un « qdisc » contenant des classes afin de déterminer dans quelle classe doit être empilé le paquet. Chaque fois qu'un trafic arrive dans une classe possédant des sous-classes, cela nécessite une classification. Il est important de noter que les filtres résident dans les « qdisc ». Les filtres ne sont pas maître de ce qui arrive.

Le mot clé « add » signifie qu'un « qdisc », une « class » ou un « filter » est ajouté à un nœud. Pour ce qui est des « qdisc », le mot clé « del » sert à supprimer l'entrée répondant aux caractéristiques qui suivent.

Les « qdisc » peuvent être attachés à la racine (« root ») d'une interface (« dev ») que ce soit avec « del » ou « add ».

Un « qdisc » qui peut avoir des fils peut leur affecter un identifiant appelé « handle ». L'identifiant qui suit le mot clé « handle » est l'identifiant qui sera affecté aux fils. Dans notre cas il s'agit de « 10 : » qui est la valeur en général choisi lorsque le « qdisc » est attaché à la racine.

Quelque soit le type (« qdisc », « class » ou « filter ») un « parent » peut être précisé an passant sont identifiant ou en attachant directement la racine d'une interface, comme précisé précédemment. Lors de la création d'un « qdisc » ou d'un « filter », il peut être nommé avec le paramètre « handle ». S'il s'agit d'un « qdisc » avec classes, il sera nommé avec « classid ». Les « parent » sont numérotés avec l'identifiant qui suit le mot clé, on précise ensuite le type de nommage (« classid » ou « handle »). L'identifiant est précisé avec celui saisi après le mot clé.

Il est possible de préciser la bande passante et le débit à l'aide des paramètres « rate » et « ceil ». L'unité de mesure est précisée en suivant la valeur (exemple : 10kbps, 1Mbps, …).

Après le mot clé « protocol » suit le type de protocole qui est employé, dans notre cas il s'agit de l'Internet Protocol.

Enfin « flowid » permet d'affecter un identifiant au flux. Cet identifiant correspond à un des « parents » qui ont été précisés dans les commandes précédentes.

Pour afficher les résultats des opérations de la commande tc la ligne est :

```
tc -s -d class ls dev eth1
```

Cela signifie que les opérations effectuées sur l'interface « eth1 » vont être affichées.

Dans ce fichier se trouve aussi des commandes « MANGLE iptables », en voici le descriptif. « iptables » est un outil d'administration pour le filtrage de paquet IPv4 et NAT.

L'option « -t » permet de spécifier le paquet correspondant à la table sur laquelle la commande doit s'appliquer. La table, dans notre cas est « mangle ». Cette table est utilisée pour des altérations de paquets spécialisés.

L'option « -A » ou « --append » introduit une chaîne de spécification de règles. La ou les règles seront ajoutées à la fin de la chaîne sélectionnée. Ici se sera donc en sortie (OUTPUT).

L'option « -o » ou « --out-interface » introduit l'interface vers laquelle le paquet va être envoyé.

« -p » ou « --protocol » est l'option permettant de spécifier le protocole employé, tel que « udp ».

C'est ensuite le port de destination qui est saisie à la suite de « --dport ».

L'option « -j » ou « --jump » spécifie la cible de la règle.

« --set-mark » précise un marquage interne au noyau Linux du paquet, il ne s'agit pas d'une modification du champ TOS mais d'un champ interne utilisé pour les règles de routage e de filtrage élaborées.

Les TOS1, TOS2 et TOS3 sont respectivement destinés aux ports 5000, 5001 et 5002. C'est pourquoi il est besoin d'y avoir pour chaque port une ligne de commande « iptables ».

La dernière commande « iptables » sert à afficher toutes les règles de la table « manlgle ».

3. Tentative de saturation des files d'attente

Afin de vérifier les principes de fonctionnement des différentes files d'attente mises en place, nous avons créé un script `mgen` qui vise à simuler différents trafics au cours du temps. Nous allons, pour simuler un débit de 1Mbps, envoyer 512 paquets de 256 octets chaque seconde durant la simulation. La taille des paquets n'est pas choisie au hasard : 256 octets font 2048 bits, ce qui est un multiple de la taille des files d'attentes que nous avons utilisées.

Le premier script, pour les communications UDP à destination du port 5000, est le suivant :

```
0.0 ON 1 UDP DST 192.168.0.1/5000 PERIODIC [512.0 256]
30.0 OFF 1

90.0 ON 1 UDP DST 192.168.0.1/5000 PERIODIC [512.0 256]
120.0 OFF 1

150.0 ON 1 UDP DST 192.168.0.1/5000 PERIODIC [512.0 256]
180.0 OFF 1

180.0 ON 1 UDP DST 192.168.0.1/5000 PERIODIC [512.0 256]
210.0 OFF 1
```

Le second script (à exécuter simultanément, dans un processus séparé) génère les communications UDP vers le port 5001 et est le suivant :

```
30.0 ON 1 UDP DST 192.168.0.1/5001 PERIODIC [512.0 256]
60.0 OFF 1

90.0 ON 1 UDP DST 192.168.0.1/5001 PERIODIC [512.0 256]
120.0 OFF 1

120.0 ON 1 UDP DST 192.168.0.1/5001 PERIODIC [512.0 256]
150.0 OFF 1

180.0 ON 1 UDP DST 192.168.0.1/5001 PERIODIC [512.0 256]
210.0 OFF 1
```

Le dernier script (à exécuter simultanément, dans un processus séparé) génère les communications UDP vers le port 5002 est le suivant :

```
60.0 ON 1 UDP DST 192.168.0.1/5002 PERIODIC [512.0 256]
90.0 OFF 1

120.0 ON 1 UDP DST 192.168.0.1/5002 PERIODIC [512.0 256]
150.0 OFF 1

150.0 ON 1 UDP DST 192.168.0.1/5002 PERIODIC [512.0 256]
180.0 OFF 1

180.0 ON 1 UDP DST 192.168.0.1/5002 PERIODIC [512.0 256]
210.0 OFF 1
```

En admettant que ces scripts s'appellent generate-packets-n.mgn où n représente le numéro du script, on peut les démarrer automatiquement à l'aide du script suivant :

```
#!/bin/bash

mgen ipv4 input generate-packets-1.mgn &
mgen ipv4 input generate-packets-2.mgn &
mgen ipv4 input generate-packets-3.mgn &
```

Remarque : nous forçons ici le mode IPv4 car les adresses que nous utilisons sont en IPv4 et si votre version de mgen est en IPv6 alors le port UDP sera bindé sur votre adresse IPv6 (si disponible) et non sur votre adresse IPv4. Vous pouvez, naturellement, ajuster cette valeur en fonction du réseau IP que vous utilisez. Cette précision est indispensable sous Windows Vista, qui active l'IPv6 par défaut.

Au niveau du client, qui recevra les paquets et mesurera le débit réel, nouslançons3 processus `mgen` séparés qui exécutent respectivement chacune des lignes suivantes :

```
0.0 LISTEN UDP 5000
0.0 LISTEN UDP 5001
0.0 LISTEN UDP 5002
```

Dans les scripts d'envoi précédents, nous simulons tour à tour les envois de données du tableau suivant :

Id.	Port 5000 (class 10:10)		Port 5001 (class 10:20)		Port 5002 (class 10:30)	
	généré	envoyé	généré	envoyé	généré	envoyé
1	1 Mbps	9,11 kb/s				
2			1 Mbps	878kb/s		
3					1 Mbps	877,3 kb/s
4	1 Mbps	9,11 kb/s	1 Mbps	878kb/s		
5			1 Mbps	190,4 kb/s	1 Mbps	681,7 kb/s
6	1 Mbps	8,93 kb/s			1 Mbps	861,7 kb/s
7	1 Mbps	4,6 kb/s	1 Mbps	195,2 kb/s	1 Mbps	675,6 kb/s

Figure 6 - Mise en oeuvre de la tentative de saturation des files de QoS

Le système est constamment en saturation et ne permet à aucun moment d'avoir le trafic généré au départ sur chaque port. Lorsque le trafic est généré sur plusieurs ports on remarque que la diminution de débit en sortie de la machine est fonction des valeurs de qualité de service définies : la file ayant un plus grand débit usuel sera favorisée.

La valeur plafond de chaque file, ni celle de la file parent, n'est jamais atteinte : au mieux on peut bénéficier de 90% du débit maximum autorisé. En revanche, les trafics de classe 10:20 et 10:30 sont capables d'utiliser un burst supérieur à leur débit classique lorsqu'ils sont utilisés seuls ou avec des débits faisant en sorte que la file parente délivre au plus 90% de sa capacité.

Ces mesures mettent en évidence le fonctionnement des files de discipline SFQ (*Stochastic Fair Queueing*) car on observe que SFQ se base sur un algorithme de hachage qui approxime un fonctionnement équitable en fonction du hachage obtenu. Par conséquent, il se peut que deux types de paquets soient hachés de la même façon, ce qui a pour conséquence de limiter le débit en dessous de sa valeur réelle, ici de 10% en moyenne.

4. Influence de la politique de gestion des files

Reprenons le script de mise en œuvre des files et rectifions de façon à avoir une file Fast FIFO. Il suffit de remplacer les lignes suivantes :

```
# Enable equity for queues

tc qdisc add dev eth1 parent 10:10 handle 40: sfq
tc qdisc add dev eth1 parent 10:20 handle 50: sfq
tc qdisc add dev eth1 parent 10:30 handle 60: sfq
```

Par les lignes :

```
# Enable equity for queues

tc qdisc add dev eth1 parent 10:10 handle 40: pfifo_fast
tc qdisc add dev eth1 parent 10:20 handle 50: sfq
tc qdisc add dev eth1 parent 10:30 handle 60: sfq
```

Une seule file a été modifiée, comme il a été demandé de le faire durant le TP.

Les scripts de tests de saturation n'ont pas besoin d'être modifié puisque le type de file ne rentre pas en ligne de compte dans ces fichiers, cela n'est pas spécifié.

Nous simulons donc tour à tour :

Id.	Port 5000 (class 10:10)		Port 5001 (class 10:20)		Port 5002 (class 10:30)	
	généré	envoyé	généré	envoyé	généré	envoyé
1	1 Mbps	9,13 kb/s				
2			1 Mbps	878kb/s		
3					1 Mbps	873,1 kb/s
4	1 Mbps	9,13 kb/s	1 Mbps	868,1kb/s		
5			1 Mbps	180,2 kb/s	1 Mbps	681,6 kb/s
6	1 Mbps	9,13 kb/s			1 Mbps	856,66 kb/s
7	1 Mbps	8,80 kb/s	1 Mbps	195,2 kb/s	1 Mbps	675,2 kb/s

On remarque que lorsque l'on génère du trafic sur plusieurs files et que l'une d'elle est la pile « Fast FIFO » le trafic de cette file est plus important qu'à la question précédente. En effet, la file 10:10 arrive presque à la valeur de son plafond de débit fixé par la commande tc. Ceci est surtout vrai dans le cas de grandes perturbations.

On peut donc en déduire que d'avoir une pile « Fast FIFO » ne résout pas totalement le problème de saturations, mais elle permet d'avoir un trafic plus important que si les files sont toutes des « SFQ », dans la mesure où les autres débits des files de type SFQ sont sensiblement inchangés.

Conclusion

Grâce à ce TP, nous avons découvert des outils Linux/Windows pratiques pour réaliser des tests de débit de transmission sur un réseau. Ces outils peuvent s'avérer utiles pour diagnostiquer d'éventuels problèmes sur un réseau personnel comme d'entreprise.

De plus, nous avons vu comme intégrer, sur une machine Linux, une stratégie de gestion de la qualité de services (QoS) et les différences entre les deux principaux types de files disponible (SFQ et FastFIFO). Ceci peut nous être utile en entreprise, pour pouvoir configurer une machine Linux utilisée comme routeur pour qu'elle favorise les types de trafic importants (comme la VoIP) par rapport aux trafics non prioritaires (comme le trafic web).

En effet, à l'aide d'`iptables`, de `tc` et des modules de gestion de la QoS intégrables au noyau Linux, il est possible de réaliser n'importe quelles limitations en fonction des besoins de l'entreprise.

Ces manipulations nous ont donc été utiles, à la fois pour notre culture personnelle, et pour nos besoins professionnels.