Université de Marne la Vallée - Ingénieurs 2000

# Travaux Pratiques : QoS

## Rapport

Professeur : H. Badis

Ngoné DIOP
Abderrahim ESSAIDI
Alexis MORELLE

Mars 2008

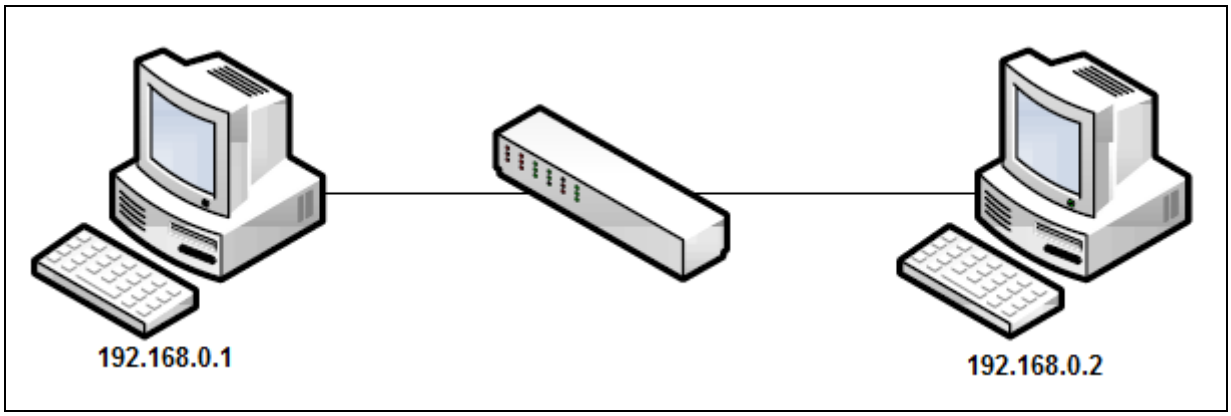
# Sommaire

# Introduction

Ce document est un rapport du TP de Qualité de Service (QoS). Nous étudions dans ce TP la circulation des paquets sur un réseau en fonction du type de trafic, du type de réseau.

# I. Première partie

## 1. Configuration IP




Avec « ifconfig », on configure nos deux stations connectées par l'intermédiaire d'un switch Hp ProCurve.

Sur machine 1 :

```
Ifconfig eth0 192.168.0.1 netmask 255.255.255.0
```

Sur la machine 2 :

```
ifconfig eth0 192.168.0.2 netmask 255.255.255.0
```

## 2. MGEN

MGEN (Multi-Generator) permet de faire des tests de performance sur des réseaux IP. Avec MGEN, on génère du trafic temps réel en UDP ou TCP. On est en mesure de générer des fichiers de traces du trafic qu'on a généré et effectuer des analyses sur ces traces.

### a. Installation

Les sources du logiciel sont disponibles à l'adresse suivante : http://downloads.pf.itd.nrl.navy.mil/mgen/src-mgen-4.2b6.tgz

Pour l'installer, commençons par récupérer l'archive à l'aide de la commande « wget » :

```
# wget http://downloads.pf.itd.nrl.navy.mil/mgen/src-mgen-4.2b6.tgz
--14:31:19--  http://downloads.pf.itd.nrl.navy.mil/mgen/src-mgen-4.2b6.tgz
           => `src-mgen-4.2b6.tgz'
…
Requête HTTP transmise, en attente de la réponse...200 OK
Longueur: 402 323 (393K) [application/x-gzip]
100%[====================================>] 402 323      442.78K/s
…
[402323/402323]
```

On décompresse l'archive qui contient MGEN pour plusieurs makefile correspondants aux différents systèmes d'exploitations. Nous allons installer la distribution linux avec la commande make en précisant le makefile pour la distribution linux. Pour nous simplifier l'utilisation de MGEN, on crée un alias de la commande.

```
# tar xzf src-mgen-4.2b6.tgz
# ls
mgen-4.2b6  src-mgen-4.2b6.tgz
# cd mgen-4.2b6/
# cd unix/
# ls
Makefile.arm-linux  Makefile.hpux    Makefile.mklinux  Makefile.solaris
Makefile.common     Makefile.linux   Makefile.netbsd   Makefile.solx86
Makefile.freebsd    Makefile.macosx  Makefile.sgi      README.TXT
# make -f Makefile.linux mgen
# alias mgen='~/diffserv/mgen-4.2b6/unix/mgen'
```

## b. Génération de trafic

Pour générer du trafic, on crée un fichier texte dans lequel on met les informations sur le type de trafic, la destination, la fréquence. On génère du trafic en utilisant les instructions suivantes, mises dans le fichier **source.mgm** :

```
2.0 ON 1 UDP DST 192.168.0.1/5000 PERIODIC [10.0 1024]
6.0 ON 2 UDP DST 192.168.0.1/5001 POISSON [100.0 8192]
11.0 OFF 2
```

**Explications :**

- `2` secondes après le démarrage, un flux nommé `1` en `UDP` avec comme IP de destination `192.168.9.24` sur le port `5000` un flux `PERIODIC` qui envoie `10` paquets de `1024` octets par seconde.
- `6` secondes après le démarrage, un flux nommé `2` en `UDP` avec comme IP de destination `192.168.9.24` sur le port `5001` un flux `POISSON` qui envoie `100` paquets de `8192` octets par seconde. Ce flux s'arrête au temps `11.0`.

Enfin, nous lançons la génération du trafic sur la machine 1.

```
# mgen input source.mgm
mgen: version 4.2b6
mgen: starting now ...
13:57:45.396624 START
```

Sur la machine 2, on écoute ce trafic mis dans le fichier **destination.mgm**.

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

On lance l'écoute de façon suivante :

```
# mgen input destination.mgm
```

## *3. TRPR*

Trpr (Trace Plot Real-time) est un programme qui analyse les fichiers de trace générés par certains logiciels d'analyse réseau. Il permet de créer des sorties analysable par l'outil GNU plot. TRPR peut travailler avec les traces de MGEN, Nous allons l'utiliser pour cela.

### c. Installation

Les sources du logiciel sont disponibles à l'adresse suivante : http://downloads.pf.itd.nrl.navy.mil/proteantools

On télécharge les sources et on les décompresse. Pour l'installation, la compilation classique en utilisant g++ suffit. Ici aussi, on crée un alias de TRPR pour faciliter son utilisation.

```
# wget http://downloads.pf.itd.nrl.navy.mil/proteantools/src-trpr-2.0b2.tgz
# tar xvf src-trpr-2.0b2.tgz
TRPR/
TRPR/Hcat.dsp
TRPR/README.TXT
TRPR/Trpr.dsp
TRPR/Trpr.dsw
TRPR/Trpr.opt
TRPR/hcat.cpp
TRPR/trpr.cpp
TRPR/trpr.html
# cd TRPR/
# g++ -o trpr trpr.cpp -lm
# alias trpr='~/diffserv/TRPR/trpr'
```

### d. Analyse de trafic

Pour un affichage graphique du trafic en temps réel :

```
mgen input destination.mgn | trpr mgen real | gnuplot
```

Pour un affichage graphique du trafic à partir d'un fichier :

```
trpr input <nom_du_fichier_de_log> mgen real | gnuplot -persist
```

## 4. Expérimentation

Nous allons faire une expérience entre les deux machines connectées au même commutateur. L'expérimentation consiste à envoyer un certain nombre de paramètre : nombre de paquets par seconde (colonne 1), la taille des paquets (colonne 2). Nous allons générer un trafic en utilisant ces paramètres. A partir des paramètres, nous allons calculer le débit théorique :

```
[(Taille du paquet x nombre de paquets x 8) / 1000].
```

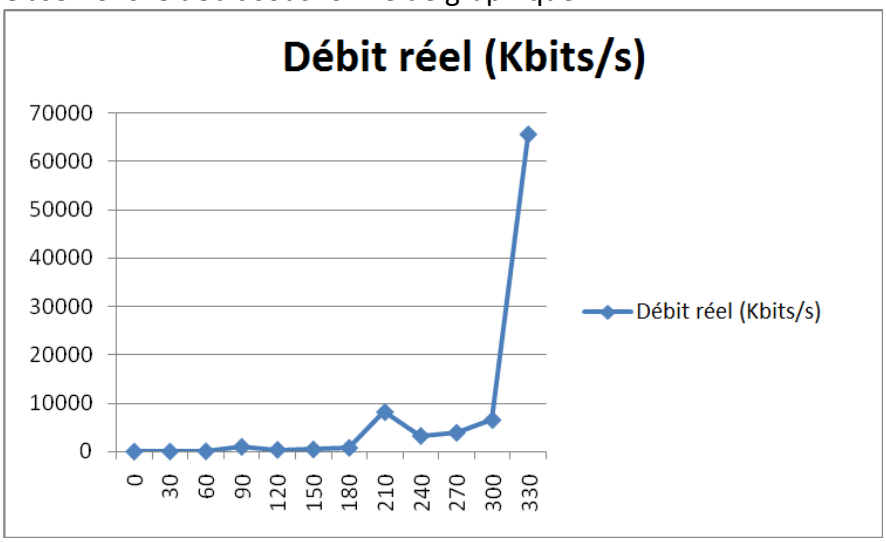
Nous allons ensuite comparer ce débit avec le débit réel obtenu dans les traces obtenus des outils décrits ci-dessus. On verra la perte et la gigue qui vont expliquer la différence entre le débit théorique et pratique.

Voici le tableau qui récapitule les résultats obtenus :

| Temps | Nb Paquets | Taille paquet (Bits/s) | Débit calculé (Kbits/s) | Débit réel (Kbits/s) | Perte | Gigue |
|---|---|---|---|---|---|---|
| 0 | 50 | 128 | 51,2 | **51,302** | -0,20% | 0,01 |
| 30 | 60 | 128 | 61,44 | **61,406** | 0,06% | 0,01 |
| 60 | 100 | 128 | 102,4 | **102,45** | -0,05% | 0,009 |
| 90 | 1000 | 128 | 1024 | **1024,31** | -0,03% | 0,009 |
| 120 | 50 | 1024 | 409,6 | **409,89** | -0,07% | 0,001 |
| 150 | 60 | 1024 | 491,52 | **490,98** | 0,11% | 0,0199 |
| 180 | 100 | 1024 | 819,2 | **831,36** | -1,48% | 0,0166 |
| 210 | 1000 | 1024 | 8192 | **8098,89** | 1,14% | 0,009 |
| 240 | 50 | 8192 | 3276,8 | **3284,76** | -0,24% | 0,001 |
| 270 | 60 | 8192 | 3932,16 | **3941,12** | -0,23% | 0,0199 |
| 300 | 100 | 8192 | 6553,6 | **6753,97** | -3,06% | 0,0166 |
| 330 | 1000 | 8192 | 65536 | **62901,04** | 4,02% | 0,009 |

Remarquons la forte augmentation du débit lorsque le nombre de paquets envoyés augmente. Nous remarquons par ailleurs que le taux de perte est grand pour les flux qui sont plus conséquents. Enfin, la gigue varie en fonction de la taille des paquets traités puisqu'il y a plus de temps de traitement au niveau des équipements.

Observons le débit sous forme de graphique :



## 5. Expérimentation avec carte réseau à 10Mo/s

Nous allons, dans cette partie, contraindre la machine 2 à recevoir au maximum avec un débit de 10Mo/s. Nous utilisons pour cela l'outil mii-tools, comme ceci :

```
# mii-tool –F 10baseT-FD
```

Prenons en compte cette modification en recalculant le débit et en reprenant les mesures pour la dernière ligne du tableau :

| Temps | Nb Paquets | Taille paquet (Bits/s) | Débit calculé (Kbits/s) | Débit réel (Kbits/s) | Perte | Gigue |
|---|---|---|---|---|---|---|
| 330 | 1000 | 8192 | 65536 | **7691,74** | -80% | 0,008 |

Avec cette basse du débit, nous observons une perte de 80%. Cette chute est due au fait que nous envoyons des données à un débit (65536 KBits/s soit 65Mbits/s) supérieur au débit maximal du lien (10Mbps).

## 6. Expérimentation avec un flux supplémentaire

L'ajout d'un second flux vient nécessairement perturber la communication monolithique qu'il y avait jusque là.

Recalculons les valeurs.

| Temps | Nb Paquets | Taille paquet (Bits/s) | Débit calculé (Kbits/s) | Débit réel (Kbits/s) | Perte | Gigue |
|---|---|---|---|---|---|---|
| 330 | 1000 | 8192 | 65536 | **3676,5** | -39% | 0,02 |

Avec ces conditions, nous voyons que le débit mesuré différent du débit calculé, cela est du aux deux flux en parallèle, il y a saturation du lien. On observe que la perte est proche de 50%, on estime donc que la perte au niveau de l'autre envoi est quasiment similaire, donc qu'il y a une répartition équivalente.

# a. Deuxième partie

Après notre étude du trafic selon son type réalisé dans la partie 1 et nos conclusions, nous allons maintenant essayer de lui imposer des restrictions pour améliorer la qualité de service. Pour ce faire, nous utiliserons les même équipements et limiterons nos débits sortants à 1Mb / s.

Un schéma de restrictions ou plutôt de contrôle de flux a été proposé dans le sujet. Nous allons réaliser un script afin de la mettre en œuvre.

- qdisc HTB : Racine
  - class HTB : limitation de trafic à 1Mb/s
    - class HTB : Limitation de trafic à 10kb/s avec un trafic normal à 10kb/s
      - qdisc SFQ : partage équitable de la bande passante entre les connexions.
      - filter : le port destination 5000
    - class HTB : Limitation de trafic à 1Mb/s avec un trafic normal à 200kb/s
      - qdisc SFQ : partage équitable de la bande passante entre les connexions.
      - filter : le port destination 5001
    - class HTB : Limitation de trafic à 1Mb/s avec un trafic normal à 700kb/s
      - qdisc SFQ : partage équitable de la bande passante entre les connexions.
      - filter : le port destination 5002

Cet arbre représente les choix possibles lors de l'envoi de paquets. Un envoi se décidera en partant de la racine de l'arbre et sera redirigé jusqu'aux feuilles selon l'algorithme choisi. Les feuilles sont des files d'attente de paquets.

Le script de configuration utilise la commande tc (traffic control) qui permet justement de contrôler le trafic en définissant des règles. Un fichier d'exemple nous a été donné ainsi qu'une documentation. Nous avons utilisé les options permettant la création des classes de trafic, la gestion des qdisc (entité de base permettant toute gestion du trafic – queuing disciple) et la création des filtres. Nous avons enfin utilisé *iptables* pour le routage des paquets en fonction de nos critères de qualité de service.

Création des classes et des qdisc :

```
tc qdisc del dev eth0 root
tc qdisc add dev eth0 root handle 10: htb
tc class add dev eth0 parent 10: classid 10:1 htb rate 1Mbps
tc class add dev eth0 parent 10:1 classid 10:10 htb rate 10kbit ceil 10kbps tc
class add dev eth0 parent 10:1 classid 10:20 htb rate 200kbit ceil 1Mbps
tc class add dev eth0 parent 10:1 classid 10:30 htb rate 700kbit ceil 1Mbps
tc qdisc add dev eth0 parent 10:10 handle 10:40 sfq
tc qdisc add dev eth0 parent 10:20 handle 10:50 sfq
tc qdisc add dev eth0 parent 10:30 handle 10:60 sfq
```

Configuration du routage (vider puis reconfigurer) :

```
iptables -t mangle -F
iptables -t mangle -A OUTPUT -o eth0 -p tcp --dport 5000 -j MARK --set-mark 1
iptables -t mangle -A OUTPUT -o eth0 -p tcp --dport 5001 -j MARK --set-mark 2
iptables -t mangle -A OUTPUT -o eth0 -p tcp --dport 5002 -j MARK --set-mark 3
```

Création des filtres :

```
tc filter add dev eth0 protocol ip handle 1 fw flowid 10:10
tc filter add dev eth0 protocol ip handle 2 fw flowid 10:20
tc filter add dev eth0 protocol ip handle 3 fw flowid 10:30
```

Grâce à mgen, nous allons générer du trafic et ainsi relever les débits. Premièrement nous envoyons séparément les flux puis petit à petit nous essayerons de les superposer pour voir la qualité de service rendu.

| Port 5000 | | Port 5001 | | Port 5002 | |
|---|---|---|---|---|---|
| Flux généré | Flux relevé | Flux généré | Flux relevé | Flux généré | Flux relevé |
| 1Mb / s | 9,85 | | | | |
| | | 1Mb / s | 979,41 | | |
| | | | | 1Mb / s | 907,72 |
| 1Mb / s | 5,51 | 1Mb / s | 9,38 | | |
| | | 1Mb / s | 342,08 | 1Mb / s | 649,88 |
| 1Mb / s | 5,87 | | | 1Mb / s | 9,81 |
| 1Mb / s | 9,82 | 1Mb / s | 5,902 | 1Mb / s | 2,88 |

Dans l'exercice il nous est demandé de saturer le réseau ce qui explique que certaines mesures laisse transparaître des pertes. De grands paquets ont été utilisés et ne sont donc pas passés correctement.

Nous observons donc que les priorités définies en terme de débit sont respectées et que la queue joue bien sont rôle dans le contrôle du trafic.

# Conclusion

Ce TP faisant suite au cours de QoS dans lequel nous avons abordé de manière théorique la qualité de service, nous nous sommes ainsi initié de manière pratique à la qualité de service. Nous avons vu quelques aspects de celle-ci mais d'autres possibilités restent à approfondir. Nous avons utilisé les outils de contrôle mais aussi de génération de trafic. Nous avons également vu comment donner des priorités à certains types de flux. Tout cela peut être directement exploité pour une application privée et sera un plus dans nos connaissances sur les réseaux.