1. Indicateurs de performance

Iperf est un logiciel de mesure de performance d'un lien IP. Il permet d'apprécier la qualité d'un réseau que l'on peut définir par 4 valeurs :

- La bande passante en TCP
- La latence ou le temps aller/retour d'un paquet.
- La gigue (jitter) est la variation de latence. Les paquets arrivent de manière irrégulière en fonction du traffic réseau.
- le taux de perte en UDP

La **bande passante** détermine la 'grosseur du tuyau' et donc le débit maximum théorique de celui-ci.

La **latence** défini le délais d'aller retour d'une information transmise sur le réseau informatique. Elle peut etre mesurée à l'aide de la commande **ping**.

La **gigue** est la variation de la latence. Les paquets arrivent de manière irrégulière en fonction du traffic réseau. Elle est donc déterminante dans le cas de la VOIP, plus la gigue augmente plus la conversation devient hachée.

Le **taux de perte** oblige à la retransmission des données, elle affecte considérablement donc la qualité du lien réseau.

Les valeurs à respectées pour considérer un réseau de qualité:

- Sur un réseau local: Perte < 0.5%, délais < 10 ms, Gigue < 5 ms
- Sur un réseau Wan: Perte < 1%, délais < 40 ms, Gigue < 10 ms
- Sur Internet (ou VPN sur Internet): Perte < 2%, délais < 100 ms, Gigue < 30 ms

2. Installation

Le programme Iperf existe sous forme de sources ou de packages binaire pour divers plateformes. Dans ce chapitre nous étudierons l'installation sous Linux et windows.

2.1. Installation sous Linux

Sur la distribution Debian l'installation de iperf se fait très simplement:

```
apt-get install iperf
```

2.2. Installation sous Windows

Télécharger le binaire de Iperf : http://dast.nlanr.net/projects/Iperf/

Exécuter le fichier iperf...exe, celui-ci se décompresse et nous obtenons un exécutable iperf.exe.

3. Utilisation

Nous devons disposer d'au moins deux machines comportant le logiciel *iperf* pour effectuer nos premières mesures. L'une se comportant en serveur (à l'écoute du port 5001 par défaut) et l'autre en client.

Par la suite j'utiliserai une machine serveur dont l'adresse IP sera 10.33.100.1 et cliente en

10.33.102.12.

Iperf comporte de nombreuses options détaillées ci-dessous:

```
-f, --format [bkmaBKMA] : une lettre spécifie le format de la bande passante à
afficher
    'b' = bits/sec            'B' = Bytes/sec
    'k' = Kbits/sec           'K' = KBytes/sec
    'm' = Mbits/sec           'M' = MBytes/sec
    'g' = Gbits/sec           'G' = GBytes/sec
    'a' = adaptive bits/sec   'A' = adaptive Bytes/sec

-i, --interval          : Temps en secondes entre chaque affichage
-m, --print_mss         : Affichage de la taille MSS (MSS= MTU -40 bytes)
-p, --port              : Port du serveur
-u, --udp               : Utiliser UDP plutôt que TCP (voir aussi l'option -b)
-w, --window            : Pour TCP = Fenêtre TCP et pour UDP = taille du tampon
recevant les datagrammes
-M, --mss               : Taille du Maximum Segment size (MTU - 40 bytes)
-o                      : Ecriture du résultat dans un fichier (uniquement pour
Window)
-c, --client            : fonction client
-l, --len               : The length of buffers to read or write (Default is 8
KB for TCP, 1470 bytes for UDP.)
```

3.1. Mesure de la bande passante

3.1.1. Mesure de la bande passante unidirectionnelle

Iperf est lancé en mode serveur par la commande suivante :

```
# iperf -s
------------------------------------------------------------
Server listening on TCP port 5001
TCP window size: 85.3 KByte (default)
------------------------------------------------------------
```

Et sur la seconde machine en mode client :

```
# iperf -c 10.33.100.1
------------------------------------------------------------
Client connecting to 10.33.100.1, TCP port 5001
TCP window size: 16.0 KByte (default)
------------------------------------------------------------
[  5] local 10.33.102.12 port 37551 connected with 10.33.100.1 port 5001
[  5]  0.0-10.0 sec    371 MBytes    311 Mbits/sec
```

La bande passante mesurée est donc de 311Mbits par seconde.

3.1.2. Mesure de la bande passante bidirectionnelle (-r)

La commande suivante mesure la bande passante dans le sens client/serveur puis serveur/client

```
# iperf -s
------------------------------------------------------------
Server listening on TCP port 5001
TCP window size: 85.3 KByte (default)
```

```
------------------------------------------------------------
[  4] local 10.33.100.1 port 5001 connected with 10.33.102.12 port 37557
[  4]  0.0-10.0 sec    374 MBytes    313 Mbits/sec
------------------------------------------------------------
Client connecting to 10.33.102.12, TCP port 5001
TCP window size: 30.1 KByte (default)
------------------------------------------------------------
[  4] local 10.33.100.1 port 52127 connected with 10.33.102.12 port 5001
[  4]  0.0-10.0 sec    353 MBytes    297 Mbits/sec

iperf -c 10.33.100.1 -r
```

Et donc 313Mb/s dans le sens client/serveur et 297Mb/s dans l'autre sens.

3.1.3. Mesure de la bande passante bidirectionnelle simultannée (-r -d)

```
# iperf -c 10.33.100.1 -r -d
------------------------------------------------------------
Server listening on TCP port 5001
TCP window size: 85.3 KByte (default)
------------------------------------------------------------
------------------------------------------------------------
Client connecting to 10.33.100.1, TCP port 5001
TCP window size: 26.2 KByte (default)
------------------------------------------------------------
[  6] local 10.33.102.12 port 37558 connected with 10.33.100.1 port 5001
[  7] local 10.33.102.12 port 5001 connected with 10.33.100.1 port 43997
[  7]  0.0-10.0 sec    261 MBytes                              220 Mbits/sec

[  6]  0.0-10.0 sec    211 MBytes                              177 Mbits/sec
```

200 Mbits/s dans le sens client/serveur et 177Mb/s dans l'autre

3.2. Mesure de la gigue et perte de paquet

Pour évaluer la gigue nous allons transmettre les paquets en UDP de manière à saturer notre lien réseau.

```
# iperf  -s -u -i 1
```

Notre gigue est donc de 0.029s sans perte de paquet. plutot bien ça :)

Et en bidirectionnelle simultanné

```
# iperf -c 10.33.100.1 -d -u -b 10m
------------------------------------------------------------
Server listening on UDP port 5001
Receiving 1470 byte datagrams
UDP buffer size:   107 KByte (default)
------------------------------------------------------------
------------------------------------------------------------
Client connecting to 10.33.100.1, UDP port 5001
Sending 1470 byte datagrams
UDP buffer size:   107 KByte (default)
------------------------------------------------------------
[  6] local 10.33.102.12 port 32790 connected with 10.33.100.1 port 5001
[  5] local 10.33.102.12 port 5001 connected with 10.33.100.1 port 33711
[  6]  0.0-10.0 sec  11.9 MBytes  10.0 Mbits/sec
[  6] Sent 8505 datagrams
[  6] Server Report:
```

```
[  6]  0.0-10.0 sec  11.9 MBytes  10.0 Mbits/sec  0.003 ms    0/ 8504 (0%)
[  6]  0.0-10.0 sec  1 datagrams received out-of-order
[  5]  0.0-10.0 sec  11.6 MBytes  9.74 Mbits/sec  0.165 ms  243/ 8505 (2.9%)
```

Houps 2.9% de perte :(Bon il faut dire que je fais les tests sur des machines virtuelles.

3.3. Influence des paramètres TCP

3.3.1. Influence de la taille de la fenêtre TCP (-w)

La fenêtre TCP correspond au nombre d'octets que le recepteur souhaite recevoir sans accusé de reception. On parle aussi de fenêtre glissante car cette fenêtre peut varier en fonction de la qualité du réseau. Cette mesure va nous permettre de vérifier son influence sur le débit transmis.

Par défaut Iperf fixe la fenêtre à 16ko

```
# iperf  -s -w 2000
```

Houps plus que 74.9Mbits/s

3.3.2. Influence de la taille de segment maximale (MSS / MTU / -m)

MSS : Il s'agit de la taille maximale de données (moins les entêtes TCP/IP soit 40 octets) transférées en une seule trame. Pour ethernet elle est généralement fixée à 1500.

Iperf nous permet d'en connaitre la valeur sur un lien. Celle-ci peut être limitée par un routeur qui serait entre nos deux machines.

```
# iperf -c 10.33.100.1 -m
------------------------------------------------------------
Client connecting to 10.33.100.1, TCP port 5001
TCP window size: 16.0 KByte (default)
------------------------------------------------------------
[  5] local 10.33.102.12 port 37571 connected with 10.33.100.1 port 5001
[  5]  0.0-10.0 sec    365 MBytes    306 Mbits/sec
[  5] MSS size 1448 bytes (MTU 1500 bytes, ethernet)
```

Il est aussi possible d'en vérifier l'influence en modifiant le MTU des trames émises:

```
# iperf -c 10.33.100.1 -m -M 500
WARNING: attempt to set TCP maximum segment size to 500, but got 536
------------------------------------------------------------
Client connecting to 10.33.100.1, TCP port 5001
TCP window size: 16.0 KByte (default)
------------------------------------------------------------
[  5] local 10.33.102.12 port 37572 connected with 10.33.100.1 port 5001
[  5]  0.0-10.0 sec  67.6 MBytes  56.7 Mbits/sec
[  5] MSS size 488 bytes (MTU 528 bytes, unknown interface)
```

On peut constater la lourde influence de ce paramètre.

3.3.3. Influence de la taille des buffers (-l)

La taille des buffers permet d'agir sur la taille des datagrammes, fixée par défaut à 8k pour TCP et 1470 pout UDP.

Testons avec un buffer de 40 en UDP, ce qui correspond à la taille des paquets de VOIP.

```
# iperf  -s -u -l 40

------------------------------------------------------------
Server listening on UDP port 5001
Receiving 40 byte datagrams
UDP buffer size:   108 KByte (default)
------------------------------------------------------------
[  3] local 10.33.100.1 port 5001 connected with 10.33.102.12 port 32790
[  3]  0.0-10.0 sec  1.24 MBytes  1.04 Mbits/sec  0.003 ms  178/32772 (0.54%)
[  3]  0.0-10.0 sec  1 datagrams received out-of-order

# iperf -c 10.33.100.1 -u -l 40
------------------------------------------------------------
Client connecting to 10.33.100.1, UDP port 5001
Sending 40 byte datagrams
UDP buffer size:   107 KByte (default)
------------------------------------------------------------
[  5] local 10.33.102.12 port 32790 connected with 10.33.100.1 port 5001
[  5]  0.0-10.0 sec  1.25 MBytes  1.05 Mbits/sec
[  5] Sent 32773 datagrams
```

OK, notre réseau est compatible VOIP :)

3.3.4. Influence du DSCP (-S)

Cette option n'est pas documentée dans le 'help' de la commande *iperf* mais elle l'est dans sa documentation.

Le champ DSCP (Differentiated Services Field) d'un paquet est un moyen de différencier divers services de catégories séparées, et de leur donner différentes priorités. Il est ainsi possible de prioriser les flux interactif (Telnet, SSH, Voip ...). Faut-il encore que l'OS et les routeurs intermédiaires prennent en charge ce mécanisme.

Pour mémoire, le DSCP est intégré dans le champ TOS des entêtes IPV4 :

TOS= 8bits on les deux de gauche inutilisés. On a donc 6 bits pour le DSCP :

| 6bits DSCP | 2 bits à 0 |

Pour pouvoir effectuer ce test encore faut-il connaitre les valeurs des dscp mis en place.

Par exemple, le réseau equant de FT fixe le dscp 42 pour la voip. **42** en décimale => **101010** en binaire

Et donc le champ TOS sera **10101000** soit **0xA8** en hexa et donc la commande *iperf* sera :

```
# iperf -c 10.33.100.1 -u -S0xA8
```

4. Représentation graphique

Après avoir vu les modes de fonctionnement d'iperf, il serait intéressant de pouvoir en grapher les résultats.

Dans cet exemple nous allons grapher la gigue et la perte de paquets en UDP. Sur le serveur exécutons la commande suivante:

```
# iperf -s -u
```

Et sur le client :

```
# iperf -c 10.33.100.1 -d -u -b16K
------------------------------------------------------------
Server listening on UDP port 5001
Receiving 1470 byte datagrams
UDP buffer size:   107 KByte (default)
------------------------------------------------------------
------------------------------------------------------------
Client connecting to 10.33.100.1, UDP port 5001
Sending 1470 byte datagrams
UDP buffer size:   107 KByte (default)
------------------------------------------------------------
[  6] local 10.33.102.12 port 32790 connected with 10.33.100.1 port 5001
[  5] local 10.33.102.12 port 5001 connected with 10.33.100.1 port 56768
[  5]  0.0-11.0 sec  21.5 KBytes  16.1 Kbits/sec  9.224 ms    0/   15 (0%)
[  6]  0.0-11.0 sec  21.5 KBytes  16.0 Kbits/sec
[  6] Sent 15 datagrams
[  6] Server Report:
[  6]  0.0-11.0 sec  21.5 KBytes  16.0 Kbits/sec  0.860 ms    0/   15 (0%)
```

Les valeurs qui nous intéressent sont mises en évidence :

Dans le sens client-serveur : 9.224ms de gigue et 0% de perte

sens serveur-client : 0.860ms et 0% de perte

Ces données seront enregistrées en base RRD, ce qui nous faciletera bien les choses lors de la création des graphes. En effet RRDTools founis tous les outils pour nous aider dans cette tâche. Nous devrons donc disposer d'une base RRD dans laquelle ces 4 données seront enregistrées. Les données seront mises à jour toutes les 5 minutes et stockées sur une durée d'un an. (soit 12 données/heure x 24 x 365 = 105120 enregistrements/an)

Débutons par la création de la tables RRD nommée `gigue_perte.rrd`:

```
rrdtool create --start N --step 300 gigue_perte.rrd \
           DS:gigue_vers_serveur:GAUGE:300:U:U \
           DS:gigue_vers_client:GAUGE:300:U:U  \
           DS:perte_vers_serveur:GAUGE:300:U:U \
           DS:perte_vers_client:GAUGE:300:U:U  \
           \
           RRA:AVERAGE:0.5:12:105120 \
           RRA:AVERAGE:0.5:12:105120 \
           RRA:AVERAGE:0.5:12:105120 \
           RRA:AVERAGE:0.5:12:105120
```

Je n'entre pas dans les détails de la création de la base à plat RRD, ce n'est pas l'objectif de ce document.

Et enfin les données devront être chargées sous cette forme :

```
rrdtool update gigue_perte.rrd N:9.244:0:0.866:0
```

N: pour 'Now' suivi des données spérées par ':'

Après avoir fourni un certain nombre de valeurs il est possible de créer le graphe avec la commande suivante:

```
rrdtool graph gigue_perte.png -a PNG  -w 600 -h 250 --title="Gigue et perte" \
          'DEF:probe1=gigue_perte.rrd:gigue_vers_serveur:AVERAGE' \
          'DEF:probe2=gigue_perte.rrd:perte_vers_serveur:AVERAGE' \
          'DEF:probe3=gigue_perte.rrd:gigue_vers_client:AVERAGE' \
```

```
'DEF:probe4=gigue_perte.rrd:perte_vers_client:AVERAGE' \
'LINE1:probe1#ff0000:gigue_vers_serveur' \
'LINE1:probe2#00ff00:perte_vers_serveur' \
'LINE1:probe3#ff00ff:gigue_vers_client'  \
'LINE1:probe4#0000ff:perte_vers_client'
```

J'ai fais un petit script Perl permettant d'automatiser tout celà:

Modifier la variable $serveur du script pour l'adapter à l'adresse ip du serveur qui devra être lancé avec la comamnde:

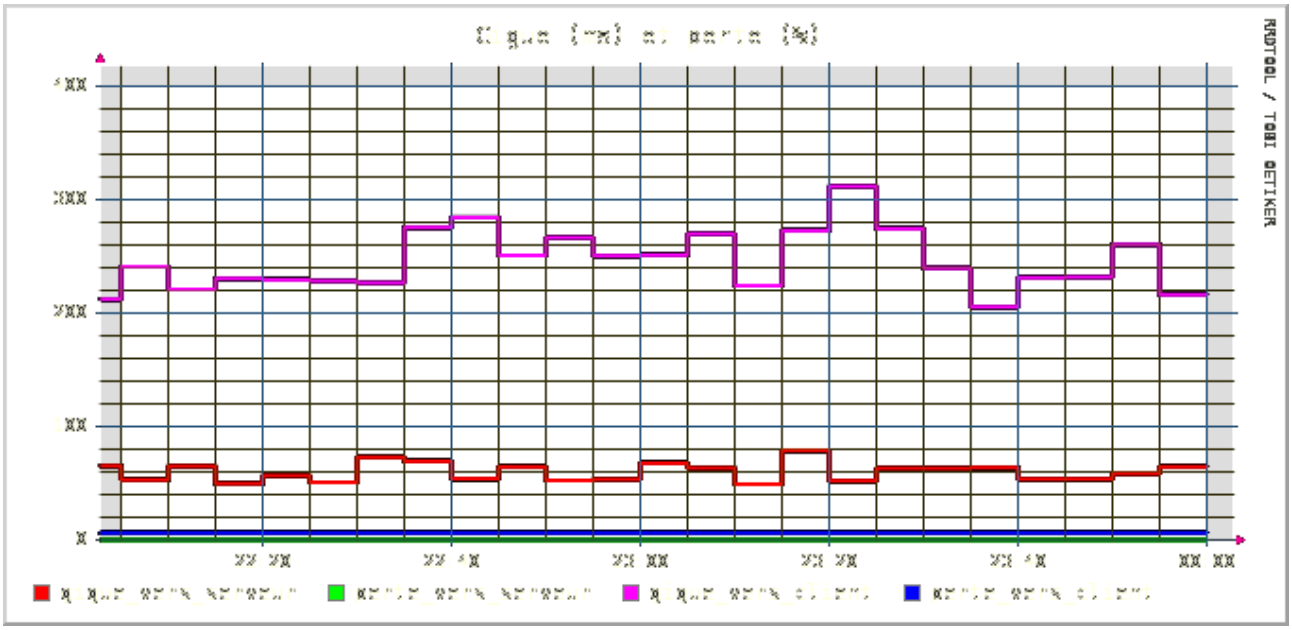
```
# iperf -s -u
```

Sur le poste client exécuter `./iperf_rrd.pl` . Celui-si lance un client '*iperf*', recupère les données retournées, créer une base rrd si elle est absente et y insère les données récoltées.

Lancé dans un cron son exécution pourra être automatisée.

Pour finir la commande suivante construit le graphe des deux dernières heures..

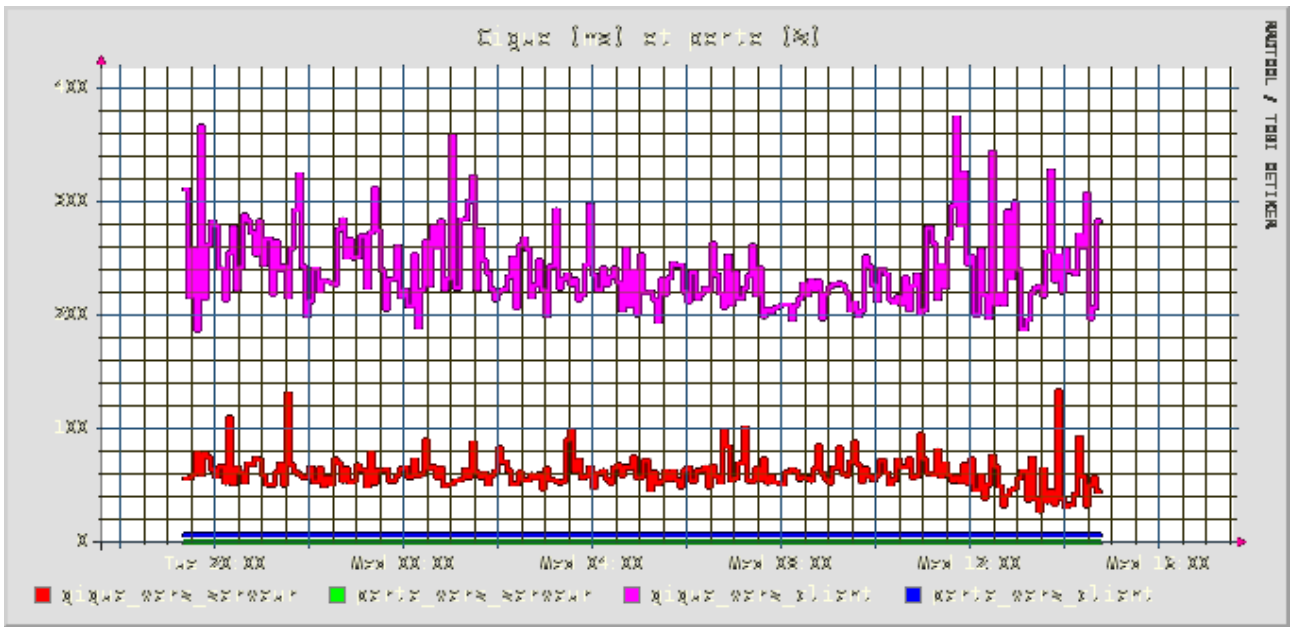
```
# ./perf_rrd.pl png 2h
```



Il est aussi possible de créer les graphes suivants :

- Deux dernières heures : 2h
- quotidien : jour
- Hebdomadaire : semaine
- Mensuel : mois
- Annuel : annee

Attention j'affiche les moyennes dans le temps (AVERAGE)


RRDTOOL / TOBI OETIKER