

Petit guide pour utiliser les nouvelles commandes (iproute2 sur 2.2, idem 2.4, 2.6 ?)

J.Bonneville

January 30, 2004

(ou pour éviter la question : pourquoi on travaille avec les anciennes commandes ?)

1 Sources et objectif

Le document «IP Command Reference» de N.Kuznetsov sert de référence principale et de source pour les exemples. Un autre document (reprenant le texte précédent, entre autre) a largement inspiré ce texte : «Linux Advanced Routing & Traffic Control» publié sous la responsabilité de Bert Hubert (PowerDNS.COM BV) , avec pour auteurs : Thomas Graf, Greg Maxwell, Remco van Mook (Virtu Secure Webservices), Martijn van Oosterhout, Paul B Schroeder, Jasper Spaans, Pedro Larroy. Ce dernier est disponible en français grace à Laurent Foucher (Foucher@gch.iut-tlse3.fr), Philippe.Latu (linux-france.org) et Guillaume Allègre (imag.fr) sous le titre «HOWTO du routage avancé et du contrôle de trafic sous Linux».

Ce document ne traite principalement que des commandes utilisées au cours de la série de TP. à rajouter MGEN et trpr...(juste exemples)

2 Nouvelles commandes

2.1 *ip*, pour presque tout

Une rapide table de conversion pourrait être :

- `ifconfig` : `ip link` et `ip address`,
- `route` : `ip route`,
- `arp` : `ip neighbour`,
- `iptunnel` : `ip tunnel`,

mais cela passerait sous silence quelques fonctionnalités nouvelles comme :

`ip rules` qui permet de définir une politique de routage,

`ip maddress` et `ip mroute` facilitant l'accès aux réseaux *multicast*.

L'évolution des protocoles (ipv6 notamment) conduit à dissocier les problèmes de configuration de voies et ceux d'adressages.

Configuration des interfaces réseaux : `ip link`

visualisation : `ip link show` (`show` peut être remplacé par `list`, `lst`, `ls`, `l`, `sh`); si l'argument `up` est fourni, seules les interfaces actives sont montrées, si un nom de périphérique est fourni, seule cette interface est visualisée. L'option `-s` (ou `-statistics`) permet de visualiser les compteurs de l'interface (`-s -s` donne plus de détail).

Modification des attributs : `ip link set` . Il est recommandé de ne modifier qu'un seul attribut à la fois (pour éviter les états incohérents en cas d'échec). Les arguments sont :

interface,

up ou down,

attributs :

`arp on` ou `arp off` (ne pas modifier sur une interface active).

`multicast on` ou `multicast off`

`dynamic on` ou `dynamic off`

`name NAME` : change lenom (ne pas faire si l'interface active ou a déjà des adresses configurée)

`txqueuelen NUMBER` ou `txqlen NUMBER`

`mtu NUMBER`

`address LLADDRESS` (link layer address)

`broadcast LLADDRESS` ou `brd LLADDRESS` pour modifier l'adresse de diffusion liaison de donnée. Danger !

`peer LLADDRESS` pour liaison point à point.

À noter la disparition des attributs `PROMISCUITOUS` et `ALLMULTI`.

Assignation des adresses : `ip address` , soit ipv4, soit ipv6, à une interface.

Ajout d'une adresse, `ip address add` avec pour arguments :

`local ADDRESS` (`local` n'est pas utile), le format de l'adresse dépendant de son type.

`peer ADDRESS`, dans le cas d'une liaison point à point,

`broadcast ADDRESS`, + ou - peuvent être utilisés lorsque l'adresse de diffusion est calculée à partir du masque. Attention, n'est plus ajoutée automatiquement !

`label NAME`, pour donner un label (compatibilité netgroup)

`scope SCOPE_VALUE` précisant la «zone» de validité de l'adresse qui peut être :

`global`

`site` (domaine ipv6 local)

`link` (juste l'interface)

`host` (juste pour l'hôte)

Pour illustrer, 2 exemples «classiques» :

1. loopback :

```
ip addr add 127.0.0.1/8 dev lo brd + scope host
```

2. une adresse en 10.0.0

```
ip addr add 10.0.0.1/24 dev eth0 brd + label eth0:Alias
```

Pour retirer une adresse, `ip address del` avec les mêmes arguments que `ip address add`, tous optionnels sauf l'interface (*device*). Si l'adresse n'est pas précisée, la première trouvée est détruite.

Un exemple à ne pas suivre :

```
ip addr add 127.0.0.1/8 dev lo
```

Pour afficher `ip address show` (ou `list`, `lst`, `sh`, `ls`, `l`) avec comme arguments :

`dev NAME` (dev optionnel)

`scope SCOPE_VALUE`

`to PREFIX` : adresses correspondant au préfixe

`label PATTERN` : adresses dont le label correspond au `PATTERN` (même syntaxe que le shell)

`dynamic` et/ou `permanent` (ipv6 seulement)

`tentative` : adresses ipv6 présumées duppliquées

`deprecated` : adresses ipv6 périmées

`primary` ou `secondary`

Pour détruire un ensemble d'adresses : `ip address flush` , qui a les mêmes arguments que `ip address show`. L'option `-statistics` passe la commande en mode verbeux. Par exemple, pour détruire toutes les adresses dynamiques ipv6 :

```
ip -6 address flush dynamic
```

Pour remplacer arp : `ip neighbour` (ou `neighbor`, `neigh`, `n`).

Pour ajouter, changer une entrée existante, ou créer ou modifier une entrée,

```
ip neigh add (a)
```

```
ip neigh change (chg)
```

```
ip neigh replace (repl)
```

3 commandes possédant les mêmes arguments :

`to ADDRESS` (to optionnel), l'adresse ipv4 ou ipv6 du voisin,

`dev NAME` : nom de l'interface concernée,

`lladdr LLADDRESS` : adresse liaison de données du voisin (qui peut être `null`)

`nud NUD-STATE` : état de l'entrée (`Neighbour Unreachability Detection`), qui peut être,

`permanent` : fixé administrativement,

`noarp` : considéré valide (pas de validation arp), mais avec une durée de vie limitée,

`reachable` : valide jusqu'au prochain *time-out*

`stale` : valide mais suspecte.

Pour voir les entrées : `ip neigh show` (ou `list`, `sh`, `ls`), avec pour arguments,

`to ADDRESS` (to optionnel), préfixe d'adresse pour sélectionner,

`dev NAME`, interface (périphérique)

`unused`, seulement non actuellement utilisé(s)

`nud NUD_STATE`, comme ci-dessus plus `all`.

L'option `-statistics` fournit des informations sur l'utilisation.

Pour vider la table `ip neighbour flush` (ou `f`) selon un critère, avec des arguments identiques à `ip neigh show`. Par défaut, les entrées aux états `permanent` ou `noarp` ne sont pas retirées. L'option `-statistics` passe la commande en mode verbeux.

Pour le *tunneling* : `ip tunnel` avec différentes commandes. 3 types de tunnels sont possibles,

`ipip` : ipv4 dans ipv4

`sit` : ipv6 dans ipv4

`gre` : n'importe quoi en ipv4 (en l'occurrence) (RFC1702 à confirmer)

Ceci remplace `iptunnel`.

Pour ajouter, modifier ou détruire un tunnel,

`ip tunnel add` (ou `a`)

`ip tunnel change` (ou `chg`)

`ip tunnel delete` (ou `del, d`)

3 commandes avec les mêmes arguments :

`name NAME` (`name` optionnel), nom du tunnel

`mode MODE` parmi `ipip`, `sit`, `gre`

`remote ADDRESS`, adresse de l'autre extrémité

`local ADDRESS`, adresse locale du tunnel, qui doit être sur une autre interface,

`ttl N`, avec `N` entre 0 et 255; par défaut, 0 qui signifie que chaque paquet hérite du TTL.

`tos T` ou `dsfield T` : fixe le *tos* à la valeur `T`, par défaut, valeur héritée.

`dev NAME` permet de fixer le tunnel à une interface,

`nopmtudisc` : interdit le *Path MTU Discovery* pour le tunnel (fait par défaut); ne peut être utilisé avec un `ttl` fixe.

`key K`, ou `ikey K`, ou `okey K`, pour tunnel `gre` uniquement; `K` est un nombre ou un quadruplet sous la forme d'une adresse ipv4.

`csum`, `icsum`, `ocsum` (tunnel `gre`) pour calculer les *checksum*, sur tous les paquets, les entrants, les sortants,

`seq`, `iseq`, `oseq` (tunnel `gre`) pour sérialiser les paquets sortant (`seq`, `oseq`), ou vérifier que les entrants le sont (`seq`, `iseq`) (Note de A.N.Kuznetsov : pas testé, à quoi ça sert ?)

Pour visualiser : `ip tunnel show` (ou `list, sh, lst, ls, l`), avec éventuellement un ou plusieurs noms en arguments. L'option `-statistics` permet d'avoir les statistiques de trafic.

Pour le *monitoring* : `ip monitor` , soit à partir d'un fichier (argument : `file FILE`), soit à partir de `RTNETLINK` pour tout (`all`) ou une liste d'objets qui peuvent être des liaisons, des adresses, des routes, ... Le fichier peut être créer à l'aide de la commande `rtmon` qui permet, à la fois de conserver les changement au format binaire et aussi sous forme d'un fichier de *log* (suggestion : `rtmon file /var/log/rtmon.log` à insérer dans le script de démarrage).

2.2 Le routage, toujours avec ip

Les tables de routages : À partir de la version 2.2, le noyau Linux maintient plusieurs tables de routages indexées de 1 à 255, ou désignées par leur nom défini dans `/etc/iproute2/rt_tables`. Par défaut, la table `main` (ID 254) est utilisée (ancienne commande `route`). Une autre table est définie par défaut (`local`, ID 255) qui contient les adresses locales et celles de diffusion; cette dernière est invisible et, généralement l'administrateur n'y touche pas. Pour un paquet, le choix de la table sera défini par des règles (`ip rules`).

Rappel : Une route utilise comme clé le couple (appelée *prefixe*) (adresse réseau, longueur de masque) et éventuellement la valeur du *TOS*. Un paquet est dirigé vers une «route» si les bits hauts (masque) (*prefixe de la destination*) sont égaux au préfixe de la route **et** si le *TOS* de la route est 0 ou égal à celui du paquet. Si plusieurs routes sont possibles, les règles suivantes s'appliquent :

1. sélection du (ou des) plus long préfixes
2. sélection selon le *TOS* :
 - (a) route(s) ayant le même *TOS* que le paquet,
 - (b) si aucune trouvée, prendre celles avec *TOS* = 0
 - (c) si aucune trouvée, alors échec.
3. sélection selon l'attribut préférence (*preference*) des routes
4. s'il y a encore plusieurs routes, la «première» est choisie (en pratique, difficilement prévisible).

Actuellement, les routes créées à l'aide de `ip` sont uniques sauf pour la route `default` sur les hôtes (*non-forwarding host*). Si plusieurs routeurs sont directement connectés, Linux exécute un algorithme de *dead gateway detection* pour choisir le routeur actif, **mais**, dans ce cas, il est plutôt conseillé de ne pas définir les routes manuellement mais d'utiliser `rdisc` (*Router Discovery Protocol* RFC1122). Actuellement (Linux 2.2, et pour ipv6, pas de définition manuelle de la route par défaut).

Plusieurs types de routes existent :

`unicast` : décrit une route réelle vers une destination (préfixe)

`unreachable` : les destinations ne peuvent être atteintes et le message ICMP adéquat est renvoyé (*host unreachable*).

`blackhole` : idem et aucun message ICMP est renvoyé.

`prohibit` : idem, mais le message ICMP *communication administratively prohibited* est renvoyé.

`local` : destination locale via *loopback*.

`broadcast` : diffusion et envoyé comme tel sur la liaison par diffusion.

`throw` : à voir avec les règles.

`nat` : route nécessitant une translation d'adresse (attribut `via`).

`anycast` : ipv6, non implémenté

`multicast` : pour le routage multicast, non présent dans les tables normales.

Pour ajouter, modifier, ajouter ou modifier une route,

```
ip route add (ou a),  
ip route change (ou chg),  
ip route replace (ou repl),
```

avec les arguments :

to PREFIX ou **to** TYPE PREFIX (**to** optionnel), le type par défaut est **unicast**. Un préfixe **default** équivalent à 0/0 ou ::/0.

tos TOS ou **dsfield** TOS : TOS est un nombre hexadécimal sur 8 bits ou un identificateur défini dans `/etc/iproute2/route2/route2_dsfield`.

metric NUMBER ou **preference** NUMBER : préférence (32 bits)

table TABLEID : identifiant de table (numéro ou nom); par défaut, **main** ou **local** si le type est **broadcast** ou **nat**.

dev NAME

via ADDRESS : Pour **unicast**, prochain routeur, ou si mode compatible BSD, une adresse locale; Pour les routes NAT, la première adresse du bloc d'adresses à traduire.

src ADDRESS : l'adresse source à utiliser de préférence pour la destination de cette route,

realm REALMID : identifiant de «royaume», soit un nombre, soit un nom défini dans `/etc/iproute2/route2/route2_realms` (voir plus loin).

mtu MTU ou **mtu lock** MTU : l'utilisation de **lock** interdit la mise à jour du *MTU (Path MTU Discovery)*, les paquets ipv4 sont envoyés avec **DF = 0**, ou ajusté au MTU pour ipv6 (?)

window NUMBER : limite la taille de la fenêtre de prévision utilisée à l'autre extrémité,

rtt NUMBER : *Round Trip Time* initial estimé (attendre 2.4)

rttvar NUMBER : variance estimée du *RTT* (attendre 2.4)

ssthresh NUMBER (attendre 2.4)

cwnd NUMBER (idem)

advms NUMBER : *MSS* annoncée pour cette destination pour TCP (attendre 2.4)

nexthop NEXTHOP : pour les route multi-chemins,

via ADDRESS : prochain routeur

dev NAME : interface de sortie

weight NUMBER poids de cet élément reflétant sa bande passante ou sa qualité relative

scope SCOPE_VAL

protocol RTPROTO : d'où provient l'information; RT-PROTO est soit un nombre, soit un nom défini dans `/etc/iproute2/route2/route2_protos`. Les valeurs prédéfinies sont :

redirect : redirection *ICMP*

kernel : noyau pendant l'autoconfiguration,

boot : installée durant le *boot*; si un démon de routage est activé, ces routes disparaissent.

static : installée par l'administrateur; prend le pas sur les autres.

ra : installée par *Router Discovery Protocol*

D'autres valeurs peuvent être définies par l'administrateur; attention aux démons de routage qui doivent utiliser une unique valeur

onlink : pour dire que le routeur est directement connecté à la liaison, même si ce n'est pas vrai (utilisation pour le *tunneling*),

equalize : permet la «randomisation» paquet par paquet sur les routes multichemins (patch noyau nécessaire), sinon la répartition se fait par flot.

Quelques exemples :

Une route vers 10.0.0 par le routeur 193.233.7.65 :

```
ip route add 10.0.0/24 via 193.233.7.65
```

route par défaut via 2 liaisons *ppp* :

```
ip route add default scope global nexthop dev ppp0 \
                                nexthop dev ppp1
```

mais il vaudrait mieux connaître l'adresse des extrémités.

pour dire que 192.203.80.144 doit être traduite avant que le paquet ne soit réémis :

```
ip route add nat 192.203.80.144 via 193.233.7.83
```

Ne pas confondre *NAT* et *masquerading*

Pour retirer une route, `ip route delete` (ou `del`, `d`) avec les même argument que `add`.

Pour voir le contenu des tables : `ip route show` (ou `list`, `sh`, `ls`, `l`) avec les critères suivants :

to SELECTOR (to optionnel); SELECTOR est constitué d'un modificateur optionnel et d'un préfixe; les valeurs du modificateur sont :

root : adresses pas plus courtes que le préfixe,

match : pas plus longues,

exact : exactement le préfixe.

par défaut : `root 0/0`

tos TOS

table TABLEID, par défaut `main`; valeurs spéciales : `all` et `cache`

cloned ou **cached** (celles créées dynamiquement à partir d'une route existante)

from SELECTOR (seulement avec `cloned`)

protocol RTPROTO,

scope SCOPE_VAL,

type TYPE,

dev DEV,

via PREFIX, routes dont le prochain routeur correspond au préfixe,

src PREFIX, celles dont l'adresse source préférée correspond au préfixe,

realm REALMID ou **realms** FROMREALM/TOREALM.

Pour éliminer des routes : `ip route flush` avec les mêmes arguments que `ip route show`; avec l'option `-sstatistics`, la commande s'exécute en mode verbeux.

Pour voir comment cela réagit : `ip route get` simule la résolution pour un paquet avec comme arguments :

`to ADDRESS` (to optionnel),
`from ADDRESS`,
`tos TOS` ou `dsfield TOS`,
`iif NAME`, interface d'entrée,
`oif NAME`, pour forcer l'interface de sortie,
`connected`, si pas de sources, alors utilise l'adresse préférée.

Base de données des politiques de routages : `ip rules` Ceci permet, selon certain critères, de choisir la table de routage; cette implémentation ne prend en compte que des informations de l'entête ip et l'interface d'entrée (meta information sur le paquet). Pour des règles concernant le contenu du paquet, celui-ci devra être marqué (label de flot ou `fwmark`) par d'autres outils (i.e. `ipchains`). Les règles sont rangées dans une liste ordonnées par une priorité fixée par l'administrateur et est parcourue dans l'ordre croissant. Chaque règle est composé d'une partie sélecteur et d'une partie action.

La partie sélecteur porte sur un ou plusieurs champs suivant : adresse source, adresse destination, interface d'entrée, `tos`, `fwmark`.

Si un paquet correspond aux critères d'une règle, la partie action est déclenchée; celle-ci consiste à trouver une route dans la table pointée par la règle. Le résultat est soit une route trouvée, soit un échec.

Trois règle sont présentes au démarrage :

1. priorité 0, aucun critère, table `local` (ID 255). Cette règle ne peut être détruite ou modifiée.
2. priorité 32766, aucun critère, table `main` (ID 254).
3. priorité 32767, aucun critère, table `default` (ID 253) initialement vide. En réserve, si aucune règle trouvée.

Les règles sont typées :

`unicast` : retourner la route trouvée

`blackhole` : détruire silencieusement le paquet,

`unreachable` : détruire et répondre via ICMP : *Network unreachable*

`prohibit` : détruire et répondre via ICMP : *Communication is administratively prohibited*

`nat` : changer l'adresse source du paquet.

Pour ajouter ou détruire une règle,

`ip rule add` (ou `a`)

`ip rule delete` (ou `del`, `d`)

avec comme arguments :

`type TYPE` (type optionnel),

`from PREFIX,`
`to PREFIX,`
`iif NAME,`
`tos TOS` ou `dsfield TOS,`
`fwmark MARK` (rappel : MARK entier 32 bits),
`priority PREFERENCE` (rappel : PREFERENCE doit être unique, sinon surprises possibles),
`table TABLEID` : table de routage à utiliser,
`realms FROM/TO` : royaume à choisir si la règle est déclenchée et une route trouvée (TO n'est utilisé que si la route trouvée ne précise pas le royaume).
`nat ADDRESS`; deux cas :

1. base du bloc des adresses translatées par la route NAT
2. adresse locale du routeur ou 0, auquel cas, le routeur doit faire du *masquerading*.

Attention, après modifications, celles-ci ne sont pas prises en compte immédiatement; pour ce faire, il faut vidé le cache par `ip route flush cache`.

Quelques exemples :

`ip rule add from 192.203.80.0/24 table inr.ruhep prio 220`

destruction de la règle par défaut,

`ip rule del prio 32767`

2.3 Le *multicast*, toujours avec ip

Assignment d'une adresse liaison de donnée multicast avec ip address .

Pour visualiser celle déjà assignées `ip address show` (ou `sh`, `list`, `ls`, `l`) avec comme argument `dev NAME` (dev optionnel).

Pour assigner ou retirer une adresse , `ip address add` (ou `a` et `ip address delete` (ou `del`, `d`), avec pour arguments :

`address LLADDRESS` (address optionnel), adresse multicast de liaison de données,

`dev NAME`, l'interface concernée.

`ip mroute show` **pour voir le cache** , seule action possible pour l'instant; les démons peuvent être `pimd` ou `mrouded`. Les arguments sont :

`to PREFIX` : sélection sur adresse destination (multicast),

`iif NAME` : sur interface de réception,

`from PREFIX` : sur l'adresse IP de la source.

L'option `-statistics` fournit les informations sur les volumes.

2.4 Les royaumes (*realms*), et `rtacct`

Les tables de routages utilisées par OSPF ou BGP peuvent rapidement devenir énormes. Classer ou faire une comptabilité des paquets par route devient vite impossible (classement par source et destination).

Les routes peuvent être regroupées sur la base de leurs attributs : les routeurs BGP connaissent leur communauté (ASPATH), les routeurs OSPF gèrent des *tag* ou leur zone, et les administrateurs connaissent aussi la nature des routes qu'ils ajoutent. Le nombre de tels agrégats, appelés **realms**, est alors plus petit. Ainsi, chaque route peut être assignée à un royaume par un démon de routage, mais les routes statiques peuvent être prises en compte avec `ip route`. Pour faciliter la construction, les «royaumes» non précisés peuvent être fournis par les règles de politiques de routage. Ainsi, pour chaque paquet, le noyau calcule les couples de royaumes (source, destination) ainsi :

1. Si la route a un *realm*, la destination reçoit le même,
2. Si la règle a un *realm* source, la source du paquet prend le même, si la route ne fournit pas de *realm*, alors prendre celui de la règle,
3. en cas d'échec, le noyau tente avec la route inverse ...

Si un *realm* n'est toujours pas trouvé, il prend la valeur 0 (*unknown realm*).

Ces *realms* sont utiles pour `tc` (Traffic Control) et le comptage qui peut être visualisé avec la commande `rtacct`.

2.5 Traffic Controller : `tc`

Ce programme permet de gérer la bande passante et la mise en file d'attente des paquets transitants par ce noeud.

Plusieurs aspects peuvent être abordés :

- Contrôle de trafic sur une liaison en découpant la bande passante en plusieurs canaux et en associant des flux de paquets à ces canaux,
- Mise en forme du trafic (*traffic shaping*) sur un interface de sortie ou en utilisant des files internes (files IMQ) pour des opérations plus complexes,
- Gestion du flux entrant (*traffic policing*), généralement pour anticiper les congestions et les rafales de réémissions.

2003/2004 : le noyau 2.4.23 utilisé n'a pas été modifié pour gérer de façon complexe le trafic entrant ou mise en forme du trafic, c'est à dire les files IMQ.

Un peu de terminologie : Ce paragraphe reprend en partie le «HOWTO...» (9.4) qui dit s'inspirer du *draft-ietf-diffserv-model-06.txt* (devenu **RFC3290**).

1. Gestionnaire de mise en file d'attente (**qdisc** - *queuing discipline*), algorithme qui gère la file d'attente d'un périphérique (éventuellement virtuel),
ingress : soit pour les données entrantes,
egress : soit pour les données sortantes.
2. Gestionnaire de mise en file d'attente sans classes (*classless queuing*), gestionnaire de file d'attente qui n'a pas de subdivisions internes **configurables** (bfifo, pfifo, pfifo_fast, tbq, sfq,...),
3. Gestionnaire de mise en file d'attente avec classes (*classful qdisc*), gestionnaire basé sur la notion de classes, ces dernières pouvant elle-mêmes être redécoupées ou être associées à un gestionnaire de file d'attente (par défaut fifo pour toutes les classes feuilles). Citons PRIO, CBQ (*class based queuing*), HTB (*Hierarchical Token Bucket*),...

4. Classificateur (*classifier*), permet d'affecter la bonne classe à un paquet (en principe, en restant à IP); basé sur des filtres (ensemble de conditions à vérifier),
5. Mise en forme du trafic (*traffic shaping*), processus consistant à retarder un paquet sortant pour respecter un débit maximum configuré (le rejet de paquet pour le ralentir un trafic est aussi appelé mise en forme).
6. Règlementation (*policing*), retarder ou jeter un paquet entrant pour rester sous une bande passante donnée;
7. *Work-Conserving* : ce dit d'un gestionnaire qui envoie toujours un paquet si un est disponible (pas de retard à l'envoi si le périphérique est disponible);
8. *non-Work-Conserving* : peut conserver un paquet, même si les ressources sont disponibles (par exemple tbf).

Un peu de syntaxe et de pratique `tc` permet :

de mettre en oeuvre différents types de file et de leurs associer différentes méthodes de découpages de la bande passante,

d'utiliser des filtres utilisant la table de routage ou les *classifiers* `u32`, `tcindex` ou `RSVP` pour affecter chaque paquet à la file appropriée.

La syntaxe est la suivantes :

```
tc [OPTIONS] OBJECT {COMMAND | help}
```

avec :

```
OBJECT : { qdisc | class | filter }
```

```
OPTIONS : {-s[statistics] | -d[details] | -r[raw]}
```

Suivant `OBJECT`, les commandes sont différentes (ainsi que les arguments) et la liste ne semble pas encore close (de nouveaux gestionnaires sont implantables - et implantées).

Ne seront détaillées que les commandes susceptibles d'être utilisées en TP (mais ce qui ne dispense pas de consulter les références). D'abord les gestionnaire *egress* sans classe, ceux avec; pour les filtres, et les gestionnaires *ingress*, voir références.

Un premier gestionnaire simple : `bfifo`/`pfifo`, implémentant une fifo simple de taille fixée soit donnée en octets (`bfifo`), soit en paquets (`pfifo`).

```
tc qdisc {bfifo|pfifo} [ limit <nombre> ]
```

Par défaut la taille en paquet est égale à `txqueuelen`, taille de la file de l'interface (en paquets), soit `txqueuelen * mtu` en octets.

Ce gestionnaire est utile pour faire des statistiques sur le trafic sans ordonnancement spécifique (pour référence).

Un gestionnaire intermédiaire : `pfifo_fast`, qui est dit sans classe bien qu'il découpe le trafic en trois bandes (car rien n'est modifiable) traitées par ordre de priorité (la bande 1 n'est exploitée que si la bande 0 est vide, la 2 que si la 0 et la 1 sont vides).

Selon la valeur de son champs `TOS`, le paquet est placé dans l'une des 3 bandes. Ceci est configurable avec le paramètre `priomap`; par contre, le paramètre `txqueuelen` est la longueur de la file de l'interface et n'est pas configurable avec `tc`.

```
tc qdisc pfifo_fast [ priomap <map> ] [ txqueuelen <nombre> ]
```

L'argument `<map>` est une suite de 16 valeurs de bandes, chacune correspondant à l'une des 16 valeurs de `TOS` possibles prises dans l'ordre croissant, et donc pour chaque `TOS`, donne son affectation.

Par défaut `<map>` = 1,2,2,2,1,2,0,0,1,1,1,1,1,1,1,1. Pour plus de détails sur l'utilisation du champs `TOS` voir références et RFC1349.

Pour information, sont disponibles aussi **tb**f (*token bucket filter*) utiles pour maîtriser les tampons en sorties et **sf**q (*Stochastic Fair Queuing*) conseillé pour les liaisons lentes surchargées.

Un premier gestionnaire avec classe : prio. Celui-ci permet de découper le trafic en plusieurs bandes (par défaut 3) en se basant sur le champs TOS des paquet selon une carte (par défaut la même que **pfifo_fast**); simplement, chaque bande, gérée par défaut en FIFO, peut se voir affectée n'importe quel gestionnaire. Il ne faut pas oublier cependant qu'une bande ne sera exploitée que si toutes les bandes de numéro inférieur sont vides (la numérotation commence à :1).

```
tc qdisc prio [ bands <nombre> ] [ priomap <map> ]
```

Bien évidemment, si le nombre de bandes change, la table d'affectation doit être fournie (**priomap**). Pour un exemple voir HOWTO page 42.

Un mot sur l'imbrication des gestionnaires avec classes : L'ensemble des classes est organisé en arbre, la racine étant attachée à un périphérique. Le noyau ne connaît que la classe racine avec laquelle il dialogue (ajout ou retrait de paquet). C'est alors le gestionnaire associé qui se charge d'interroger ses sous-classes dans l'ordre défini par son algorithme d'ordonnancement, et ainsi de suite jusqu'aux feuilles. La conséquence, c'est qu'un gestionnaire ne peut pas retirer de paquets plus vite que ne l'autorise son gestionnaire parent (conservation des contraintes).

Class Based Queuing (cbq) est un gestionnaire permettant de décrire des classes, mais aussi pratiquant une mise en forme du trafic (sur ce point, différents auteurs émettent des réserves). En effet, lors de la définition d'une classe, la bande passante maximum affectée au trafic de cette classe doit être fournie. Pour garantir cette limitation, CBQ tente de mesurer le pourcentage de temps pendant lequel le périphérique réseaux est inactif. Cette estimateur n'est pas très précis, dépend de la configuration matérielle et supporte mal les périphériques virtuels. (pas détaillé ici, trop de paramètres:).

Hierarchical Token Bucket (htb), une version adapté aux classes du seau à jetons. Pour les explications, voir <http://luxik.cdi.cz/devik/qos/htb/>.

2.6 MGEN et trpr

Pour générer des trafics avec des TOS différents, plusieurs outils sont disponibles. MGEN permet de générer des flux UDP et de collecter les mesures. La possibilité de faire du TCP est annoncé. (Note: Si les outils ne sont pas installés, les sources se trouvent dans le répertoire /root: src-XXX.tgz)

Si vous devez absolument faire des flux TCP, utilisez Iperf (permet TCP, UDP avec TOS), mais il n'est pas installé.

Nous conseillons l'utilisation systématique de script et la syntaxe ainsi simplifiée est la suivante :

```
mgen input <scriptFile> [ output <logfile> ]
```

Comme lors de la plupart des mesures effectuées, nous utilisons un flux uni-directionnel, nous distinguerons le script émetteur et le script récepteur. Ceux-ci sont constitués d'événements constitués d'une date (donnée sous la forme d'un nombre en virgule flottante, l'unité étant la seconde, et relative à l'instant de démarrage de MGEN), d'une commandes ainsi que de ses paramètres.

Les sites relatifs à ces 2 logiciels sont :

<http://manimac.itd.nrl.navy.mil/MGEN/>

<http://proteantools.pf.itd.nrl.navy.mil/trpr.html>

Evénements définissant la transmission

Création d'un flot de paquets `<eventTime> ON UDP DST <flowId> <adresseIP>/<port>`
`<pattern [ param ]> { <options> }`

`<flowId>` est un entier permettant d'identifier un flot de paquets. À un instant onnée, un identifiant de flot doit être unique.

`<pattern [ param ]>` permet de choisir la forme du trafic :

periodique : `PERIODIC [ rate size ]` génère des message de taille `size` à la vitesse `rate` paquets par secondes (`rate * size = débit en octets/s`);

Poisson : `POISSON [ aveRate size ]` pour un flux de paquets de taille fixée `size` à une vitesse moyenne de `aveRate` messages par secondes, les intervalles entre paquets respectant un processus poissonien;

en rafale : `BURST [ { REGULAR | RANDOM } <aveInterval> <pattern [ param ]> { FIXED | EXPONANTIAL } <aveDuration> ]` Pour générer des raffales de paquets à intervalles, exprimés en secondes, `<aveInterval>` fixes ou aléatoirement tirés selon une loi exponentielle de moyenne `<aveInterval>`, et de durée `<aveDuration>` exprimé aussi en secondes, la durée étant fixe ou tirée selon une loi exponentielle (la valeur est donc la durée moyenne dans ce cas). Durant les créneaux ainsi spécifiés, les paquets sont produits selon une des 3 méthodes décrites. Ce type de trafic permet de simuler du trafic de type «voix sur IP».

Un flot peut être modifié par un événement `MOD` avec exactement les même paramètres que la commande `ON`.

Pour les options, nous n'utiliserons que l'option : `TOS <valeur>` ; la valeur est souvent donnée en hexadécimal.

Attention, pour avoir des flux avec différentes valeurs de `TOS`, il faut utiliser des ports différents destinations différents (constaté en TP).

Arrêt d'un flot `<eventTime> OFF <flowId>`

Pour la réception Bien que nous transmettions des paquets UDP, un script pour récupérer les paquets envoyés est nécessaire, d'une part pour éviter en retour les message ICMP, et d'autre part, pour collecter les informations sur le trafic.

`<eventTime> LISTEN UDP <portList>`

reprise de l'exemple `0.0 LISTEN UDP 5000,5003-5005,5009`

qui met en écoute sur les ports 5000, 5003, 5004, 5005 et 5009. Seuls les paquets ayant ses ports en destinations seront dans le fichier de log.

Pour stopper l'écoute `<eventTime> IGNORE UDP <portList>`

Exploitation des résultats Les fichiers de log peuvent être étudiée à l'aide de `trpr`, soit à la volée, soit à partir des log, soit les 2 en même temps. La visualisation est réalisée avec `gnuplot`. À noter que les flux sortant de `tcpdump` peuvent être filtrer par le même outil (juste débit).

Selon les options, `trpr` permet d'extraire le débit (par défaut), le temps de latence, la gigue ou le nombre de paquets perdus.

À noter que du fait que les horloges ne sont pas synchronisées (pas de serveur NTP installé pour l'instant), les temps de latence mesurés sont fortement biaisés, mais, a-priori, sur une mesure, le biais peut être considéré comme constant.

exemple `mgen -input scriptReception.mgen | trpr real auto X | gnuplot -noraise -persist`

affichera en temps réel (compter quelques secondes pour le démarrage) les débits par flux (les flux MGEN sont nécessairement identifiés - `<flowID>`).

Si l'on veut tracer plusieurs courbes simultanément :

```
#!/bin/sh
mgen -input scriptReception.mgen output log
# debit
tail -f log | trpr mgen real auto X | gnuplot -noraise -persist
## on peut avoir la meme chose avec tcpdump
tcpdump -x ip | trpr real auto X | gnuplot -noraise -persist
# gigue
tail -f log | trpr mgen real interarrival auto X | gnuplot -noraise -persist
#latence
tail -f log | trpr mgen real latency auto X | gnuplot -noraise -persist
#perte
tail -f log | trpr mgen real loss auto X | gnuplot -noraise -persist
```

La sortie de `trpr` peut aussi être stockée dans un fichier; ce fichier contient entre autre, des commandes `gnuplot` facilement éditables (par exemple pour avoir un format directement imprimable).

2.7 Futur

Apparemment un patch du démon SNMP (`snmpd`) et quelques outils permettent la configuration et le monitoring à distance (avec ou sans MRTG mais avec `rdd`). À ne pas oublier pour l'an prochain.