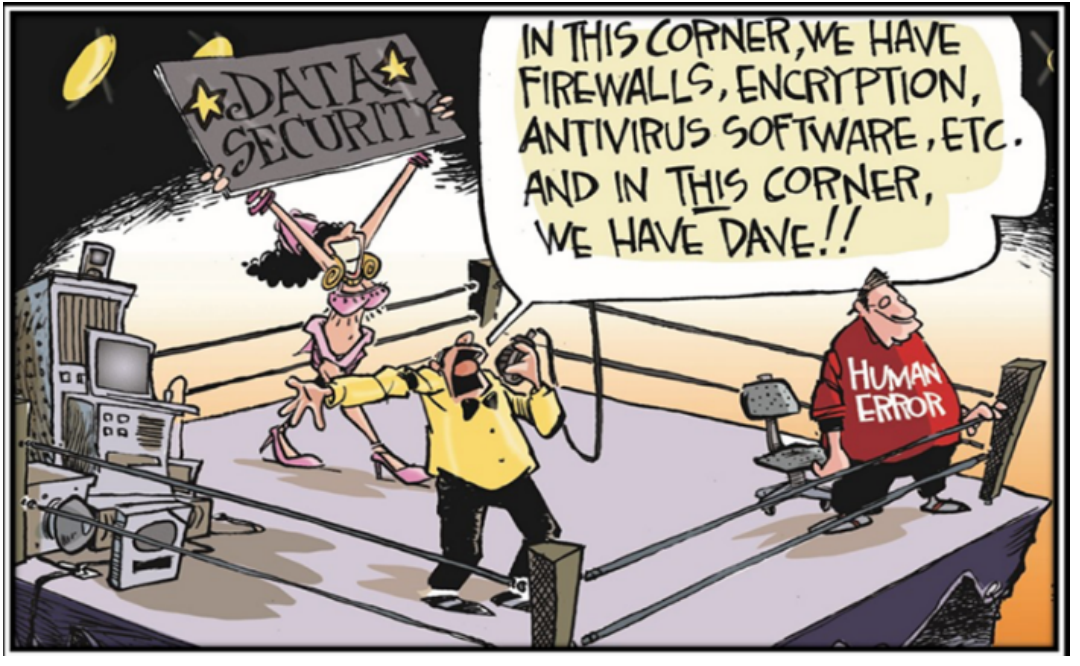
Faculté des Sciences & Techniques

Université de Limoges

Master 1ère année



Administration réseau — P-F. Bonnefoi

Version du 21 octobre 2024

Plan

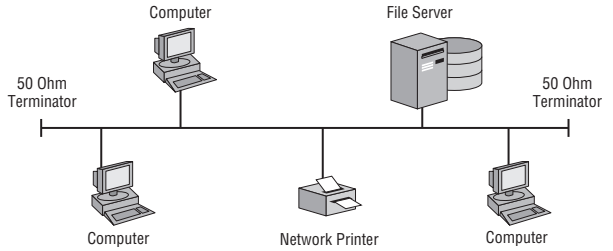
- ☐ Qu'est-ce qu'un réseau local : Domaine de collision *vs* Domaine de diffusion ;
- ☐ Cadre d'apprentissage : la virtualisation sous Linux avec les «*net namespaces*» ;
- ☐ Architecture réseau : la segmentation : CIDR, VLSM et VLANs ;
- ☐ Services dans un réseau local : adressage de groupe et DNS ;
- ☐ Gérer les problèmes : protocole ICMP et risques associés ;
- ☐ Filtrage du trafic réseau : présentation du Firewall NetFilter ;
- ☐ Administration réseau et firewall : les différents usages ;
- ☐ NetFilter, un firewall modulaire ;
- ☐ Administration réseau pour l'arbitrage du trafic des usagers : la QoS, «*Quality of Service*» ;

Domaine de Collision

vs

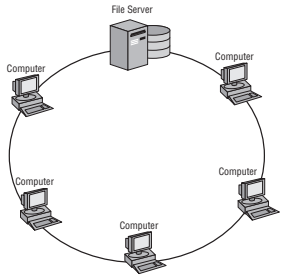
Domaine de diffusion

Le bus



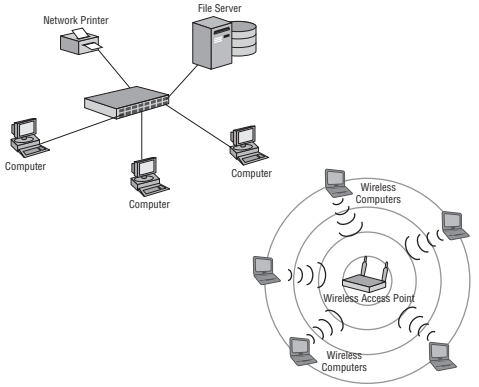
- ancienne technologie : n'existe plus actuellement ;
- représentation «conceptuelle» d'un LAN, «*local area network*» ;
- réseau à diffusion :
  - tout matériel connecté **peut** communiquer avec tous les autres : «*multiple access*» ;
  - un matériel décide de manière autonome **quand** il transmet : il peut y avoir des **collisions d'accès**, et il faut s'en protéger : «*collision detection*», «collision avoidance», *etc.*

Anneau ou «ring»



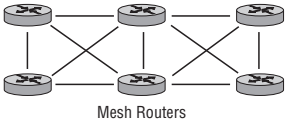
- utilisé chez les fournisseurs d'accès Internet ou ISP, «*Internet Service Provider*», pour des **liaisons rapides** en fibre optique ;
- chaque matériel est connecté à deux autres matériels :
  - les données sont transmises dans un seul sens : elle «tourne» sur l'anneau entre les différents matériels ;
  - un matériel ne peut transmettre que lorsque c'est **son tour** : un **jeton d'autorisation** circule dans l'anneau pour autoriser les transmissions (on parle de «*token ring*») ;
  - un seul matériel communique à la fois, pas de collision : les performances sont meilleures, le débit maximal est atteint.

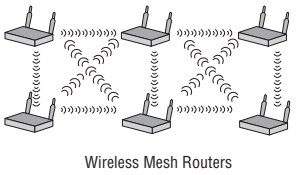
Étoile ou «star»



- la topologie la plus courante pour définir un LAN : plusieurs matériels connectés à un nœud de connexion central :
  - un «hub» : un seul **domaine de diffusion** ;
  - un «switch» ou un point d'accès sans fil : le nœud a la capacité de mettre en relation deux matériels voulant communiquer entre eux (commutation ou «switching») ;
- avantage : si un matériel est en panne, le réseau continue à fonctionner.

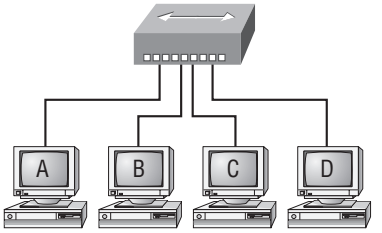
Réseau maillé ou «Mesh»





- chaque matériel possède une ou plusieurs connexions avec les autres matériels ;
- cette topologie offre une meilleur :
  - une meilleure résistance, «resilience», en cas de rupture d'un lien de connexion ou de panne d'un matériel ;
  - une économie de moyens par rapport à une redondance totale (maillage complet) ;
- les liens peuvent être filaire ou sans fil ou un mélange des deux. Dans le cas du WDS, «*Wireless Distribution System*», un AP, «*access point*», du maillage sert d'intermédiaire de connexion vers des APs connectés au réseau filaire «*backbone*».

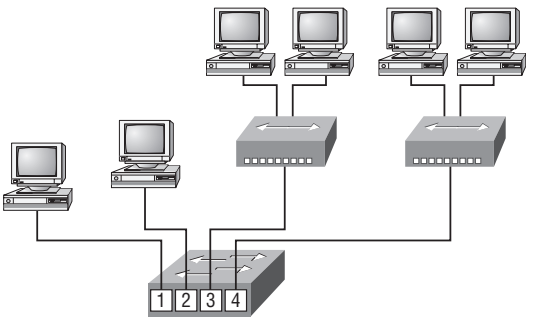
- le **répéteur** ou «*repeater*» :
  - correspond au «*hub*» : un répéteur multiport ;
  - regénére, réamplifie la transmission physique et la relaie vers tous les ports (pour en augmenter la distance, lutter contre les atténuations) …
  - tous les hôtes connectés
    - font parti du **même domaine de collision** ;
    - font parti du **même domaine de diffusion** ;
    - **partagent la capacité** du réseau.



- le **pont** ou «*bridge*» :
  - «divise» le domaine de collision en sous-domaines :
    - **relaie** une trame d'un **sous-domaine de collision à un autre** seulement si l'émetteur et le destinataire sont dans deux sous-domaines différents (un domaine par port, avec un nombre très limité de ports) ;
    - **apprend** le sous-domaine où chaque hôte est localisé grâce à son adresse MAC ;

- le «***switch***» :
  - bridge multi-ports, réalisant le travail en «hardware» :
    - **circuits électronique dédiés** : rapide et intelligent ;
    - **parallélisme** : plusieurs trames relayées simultanément entre des ports différents : débit aggrégé **très élevé**

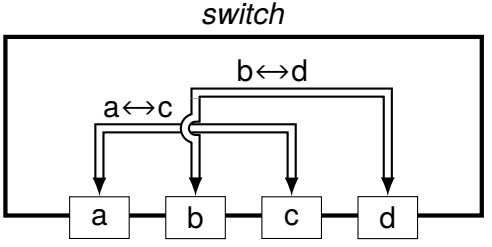
```
Forwarding modes: Store-and-forward
Bandwidth:       48 Gbps for GS724TS,
                 96 Gbps for GS748TS
```



  - chaque segment/port a **son propre domaine de collision** ;
  - tous les segments sont dans le **même domaine de diffusion** ;
  - ne bloque pas les trames en «brodcast» ou en «multicast» ;
  - évite les boucles avec STP, «*Spanning Tree Protocol*».

Réalise de la commutation

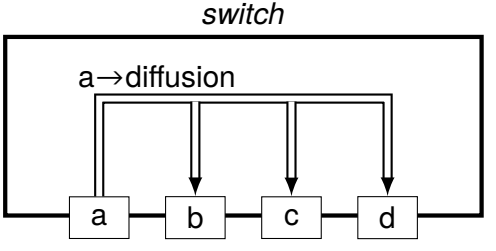
- ▷ **supprime les collisions de paquets** lors de la **compétition** pour l'accès au réseau ;
- ▷ **réalise plusieurs communications simultanées** ce qui améliore le débit global :



Les machines connectées aux ports «*a*» et «*c*» peuvent communiquer en même temps que les machines connectées sur les ports «*b*» et «*d*».
⟹*deux communications simultanées au lieu d'une seule !*

Réalise de la diffusion

Lorsque la machine connectée au port «*a*» envoie une trame en **diffusion**, le switch diffuse cette trame à tous les ports.



- ▷ le switch représente **un seul domaine de diffusion** : c'est le réseau local !
- ▷ le switch **découpe** les **domaines de collision** : seules les machines en communication sont concernées, pas de risque de collisions.

Le lien point à point, «point to point»

- un seul émetteur d'un coté du lien ;
- un seul récepteur de l'autre côté du lien ;
- contrôle des échanges MAC «*Medium Access Control*» : «flow control» ou contrôle de flux ;
- exemples : PPP, tunnels, Ethernet Gigabit, switch full duplex…

Le lien à diffusion, «broadcast link»

- de multiples matériels accèdent en émission et en réception au support de communication, «Multiple Access» ;
- chaque matériel reçoit une copie du message émis ;
- contrôle des échanges MAC «Medium Access Control» : CSMA/CD «*Carrier sense Multiple access with Collision Detection*», CSMA/CA «*...with Collision avoidance*» ;
- exemples : Ethernet 100Mbits/1Gbps, switch, WiFi…

Les différents types de liens sous Linux

xterm

```
root@starfox:~# ip link
2: eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast state UP mode
DEFAULT qlen 1000
    link/ether 00:0c:29:89:01:44 brd ff:ff:ff:ff:ff:ff

8: mon_pt_a_pt: <POINTOPOINT,NOARP> mtu 1480 qdisc noop state DOWN mode DEFAULT
    link/sit 10.1.1.1 peer 192.168.1.1
```

- `BROADCAST` : lien à diffusion avec support du multicast ;
- `POINTOPOINT` : lien point à point sans support du protocol ARP.

Ethernet

- «***Half-Duplex***»
  - à l'origine dans la norme IEEE 802.3 Ethernet ;
  - «CSMA/CD» :
    - éviter les collisions ;
    - retransmettre les trames en cas de collision ;
  - utilise une paire de fils dans le câble pour une transmission **alternée** dans les deux sens ;
  - 30-40% d'efficacité : 30 Mbps à 40 Mbps ;
- «***Full-Duplex***»
  - utilise une liaison «point à point» entre l'émetteur et le récepteur ;
  - plus de collisions possibles ;
  - exige un «switch» et non un «hub», avec un port «dédié» pour l'hôte :
    - un switch et un autre switch ;
    - un switch et un hôte ;
    - un routeur et un autre routeur ;
    - un switch et un routeur ;
    - un hôte et un autre hôte à l'aide d'un câble croisé (le croisement est réalisé automatiquement par une interface gigabit avec l'option «Auto-MDIX») ;
  - utilise deux paires de fils dans le câble pour une transmission **simultanée** dans les deux sens ;
  - 100% d'efficacité possible dans les deux directions :
    - 20 Mbps pour de l'Ethernet 10Mbps :
    - 200 Mbps pour de l'Ethernet 100Mbps ;
- Comment la carte choisit entre les deux modes et le débit ? la «**négociation**» et un mécanisme d'**auto-détection**.

La carte Ethernet et sa configuration

xterm

```
root@starfox:~$ sudo -s
root@starfox:~# ethtool eth0
Settings for eth0:
    Supported ports: [ TP ]
    Supported link modes:   10baseT/Half 10baseT/Full
                            100baseT/Half 100baseT/Full
                            1000baseT/Full
    Supported pause frame use: No
    Supports auto-negotiation: Yes
    Advertised link modes:  10baseT/Half 10baseT/Full
                            100baseT/Half 100baseT/Full
                            1000baseT/Full
    Advertised pause frame use: No
    Advertised auto-negotiation: Yes
    Speed: 1000Mb/s
    Duplex: Full
    Port: Twisted Pair
    PHYAD: 0
    Transceiver: internal
    Auto-negotiation: on
    MDI-X: off
    Supports Wake-on:
    Wake-on: d
    Current message level: 0x00000007 (7)
                drv probe link
    Link detected: yes
```

auto-négociation activée

mode gigabit négocié

permet de «croiser» le câble en cas de connexion directe vers une autre carte réseau

La commande doit être exécutée avec les droits de «root».

On remarque que la commande nous renseigne sur l'interface «eth0» :

- *le gigabit Ethernet uniquement en «Full Duplex» ;*
- *le support du «câble croisé».*

Expérimentation et compréhension

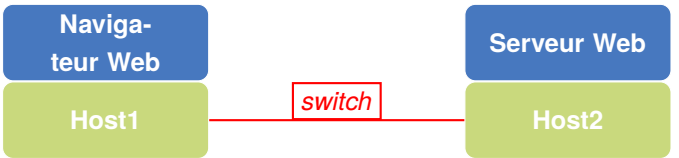
- ▷ «*apprendre par la pratique*» : se rapprocher du réel, expérimenter ;
- ▷ pour la programmation ? Programmer, modifier, compiler *etc.*
- ▷ pour le réseau ? Disposer de plusieurs machines, de switches, de routeurs *etc.*
- ▷ pour l'administration réseaux ? Manipuler les commandes de configuration et de diagnostique.

Plateforme	Avantages	Inconvénients
Matériel	rapide, «réelle» mais impossible de disposer de tous les matériels des différents constructeurs	coûteux, difficile à partager, difficile à reconfigurer et à changer
Simulateur	peu coûteux, flexible, permet d'accèlérer l'expérimentation (temps artificiel)	peut nécessiter des ajustements, ne correspond pas à des opérations normales du système d'exploitation, pas toujours interactif
Émulateur	*peu coûteux, flexible, assez réelle, interactif*	*plus lent qu'une solution matérielle, peut ne pas donner de bons résultats lors du multiplexing de plusieurs communications*

Exemple de réseau

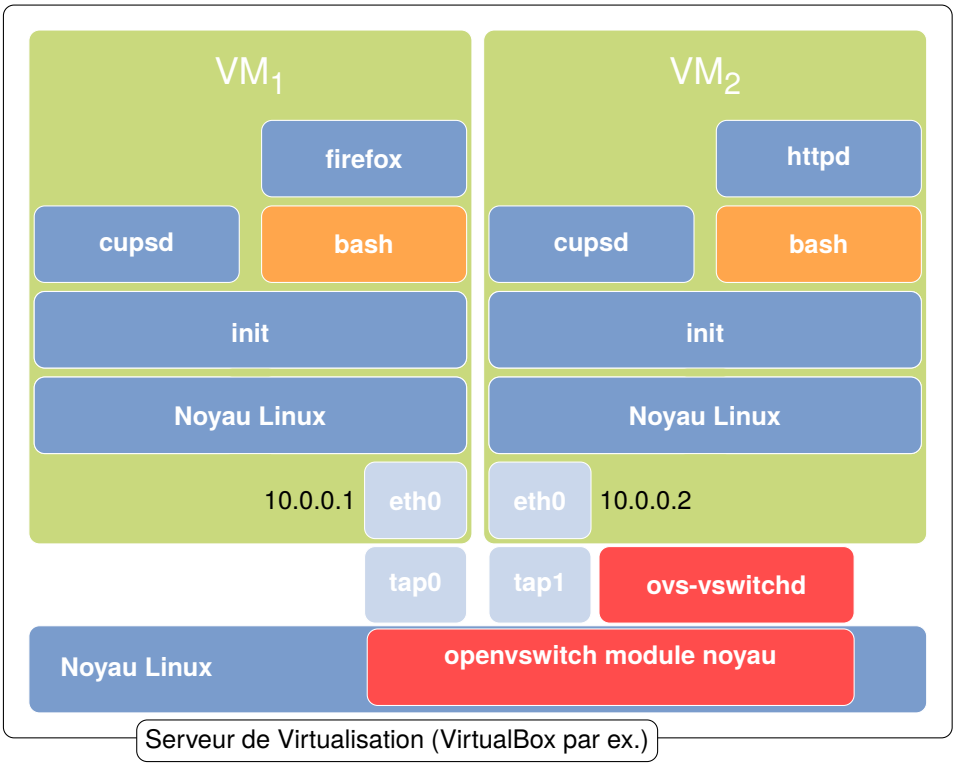
Deux hôtes connectés par un **switch** :

- ☐ le «Host2» héberge un serveur Web, c-à-d un processus «httpd» ;
- ☐ le «Host1» exécute un navigateur Web, qui va se connecter au serveur Web ;
- ☐ les deux machines sont connectées à un switch et appartiennent au même réseau IP.



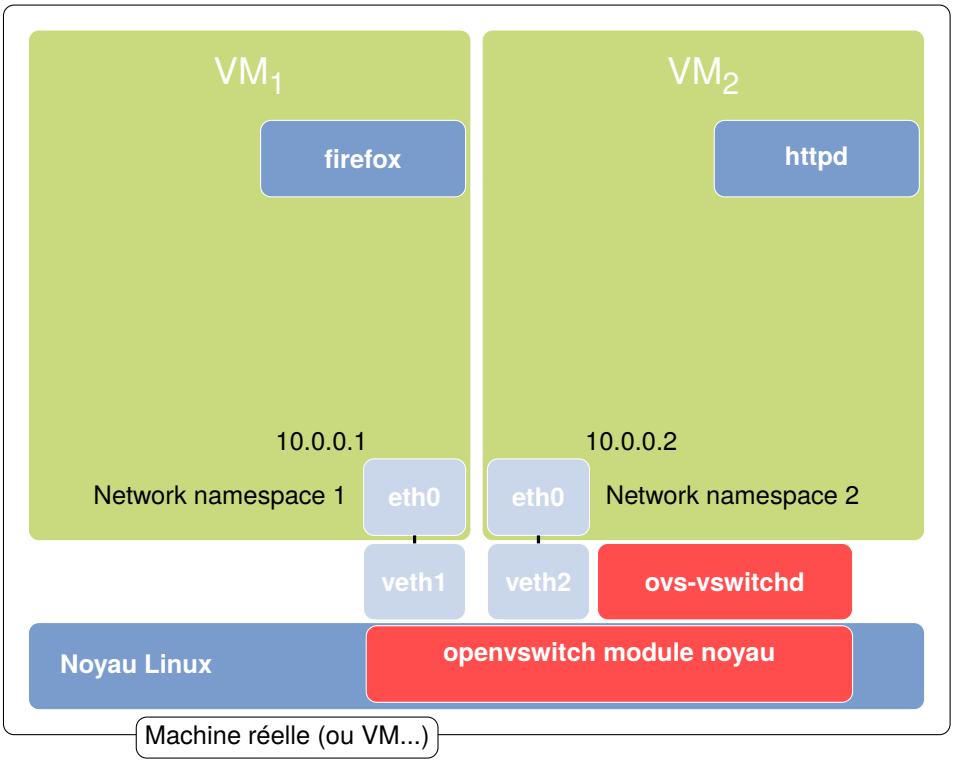
Virtualisation complète : deux VMs, «*Virtual Machine*», et un switch virtuel

- installation de Virtualbox : solution gratuite de virtualisation (permet par ex. de faire tourner un Linux sur un Windows) ;
- création de deux VMs : chaque VM est une machine complète sur laquelle il faut installer Linux (~ 10Go par VM) ;
- configuration du réseau connectant les deux VMs.



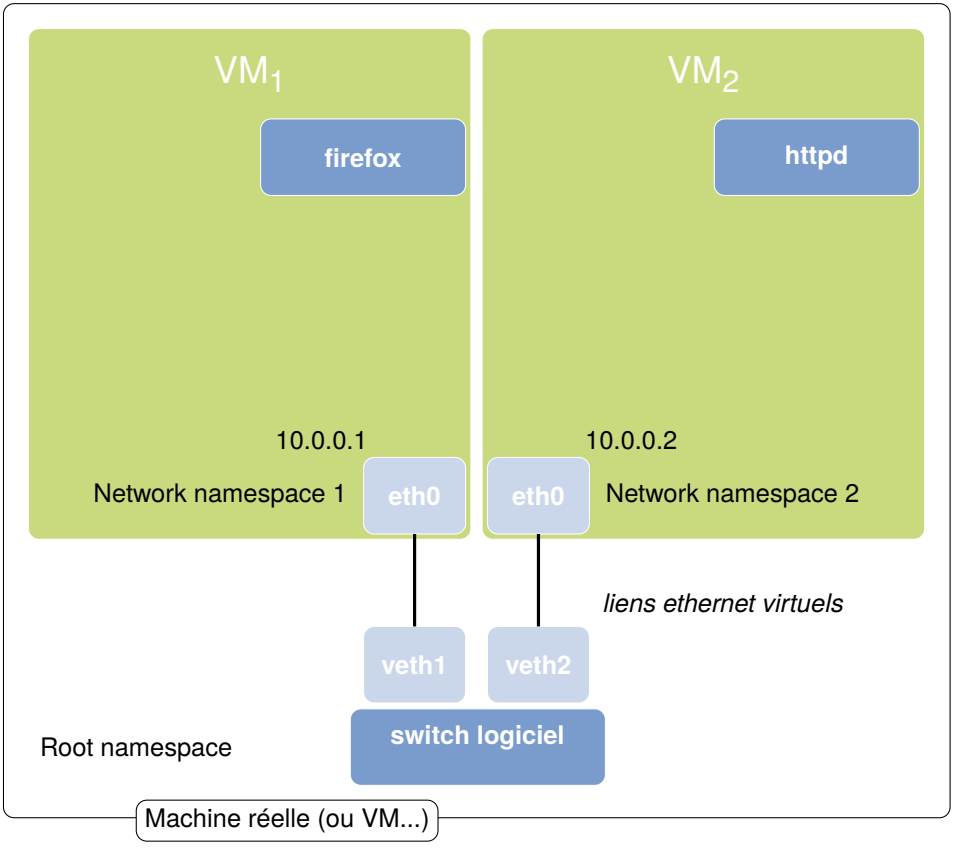
Virtualisation légère : deux espaces de nom réseaux et un switch virtuel

- une seule machine sous Linux ou une seule VM sous Windows ;
- création d'espace de nom réseau, «network namespace» ;
- un switch logiciel ;



Espaces de nom réseaux et liens Ethernet virtuels

- l'espace de nom «*root*» : celui correspondant à la machine, c-à-d sa pile réseau TCP/IP ;
- deux espaces de noms en plus, 1 & 2 : deux piles TCP/IP supplémentaires (soient 3 au total liés par un switch logiciel).

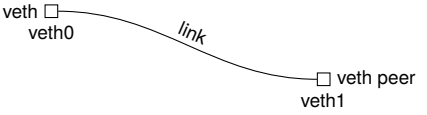


- les **interfaces** :

```
xterm
pef@cube:~$ sudo ip link add toto type dummy
pef@cube:~$ ip link
10: toto: <BROADCAST,NOARP> mtu 1500 qdisc noop state DOWN mode DEFAULT group default qlen 1000
    link/ether 5a:01:c1:2f:d9:81 brd ff:ff:ff:ff:ff:ff
```

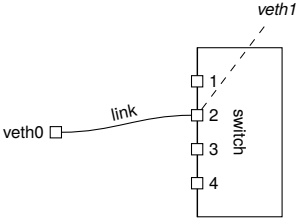
- les **liens**, «*virtual ethernet*», : un lien dispose de deux extrémités, «veth» et «veth peer» :

```
xterm
pef@cube:~$ sudo ip link add veth0 type veth peer name veth1
```



- les **switches** :

```
xterm
pef@cube:~$ sudo ovs-vsctl add-br mon_switch
pef@cube:~$ sudo ovs-vsctl add-port mon_switch veth1
```



- le **routeur** :

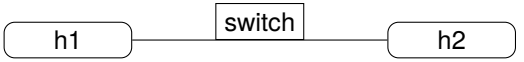
```
xterm
pef@cube:~$ sudo sysctl -w net.ipv4.conf.all.forwarding=1
```

On active la fonction de routage du noyau Linux.

- le **Firewall** pour activer le NAT :

```
xterm
pef@cube:~$ sudo iptables -t nat -A POSTROUTING -s 10.0.0.0/24 -j MASQUERADE
```

Création des deux «hôtes», du switch et câblage



xterm

```
# passe en administrateur
sudo -s
# créer les namespaces pour les hôtes
ip netns add h1
ip netns add h2
# créer le switch
ovs-vsctl add-br s1 ❶
# créer les liens
ip link add h1-eth0 type veth peer name s1-h1
ip link add h2-eth0 type veth peer name s1-h2
ip link show ❸
# accrocher les liens aux namespaces
ip link set h1-eth0 netns h1
ip link set h2-eth0 netns h2
# afficher le lien depuis le namespace
ip netns exec h1 ip link show ❹
ip netns exec h2 ip link show
# connecter les liens au switch
ovs-vsctl add-port s1 s1-h1 ❷
ovs-vsctl add-port s1 s1-h2
ovs-vsctl show
# activer les interfaces du namespace root ❺
ip link set dev s1-h1 up
ip link set dev s1-h2 up
# activer les interfaces des namespaces h1 et h2 ❻
ip netns exec h1 ip link set dev h1-eth0 up
ip netns exec h2 ip link set dev h2-eth0 up
```

Remarques

❶⇒La création du switch est conservé lors d'un redémarrage du système.

❷⇒L'ajout d'un lien est également conservé **même** si le lien n'existe plus (il sera automatiquement rajouté lors de la réutilisation du même nom pour le lien `veth`).

❸⇒On affiche la liste des interfaces présentes.

❹⇒Pour exécuter une commande dans le namespace `h1: ip netns exec h1 <commande>`

❺⇒le «netnamespace root» correspond à la pile réseau racine, c-à-d celle initiale de l'hôte.

❻⇒l'interface de chaque netnamespace doit être activé depuis le netns correspondant.

Si on veut changer l'adresse MAC d'une interface :

xterm

```
ip link set dev eth0 address 01:02:03:04:05:06
```

Vérification de la configuration du switch : commandes `ovs-ofctl` et `ovs-vsctl`

xterm

```
pef@cube:~$ sudo ovs-ofctl show s1
OFPT_FEATURES_REPLY (xid=0x2): dpid:00004e9931535a4a
n_tables:254, n_buffers:256
capabilities: FLOW_STATS TABLE_STATS PORT_STATS QUEUE_STATS ARP_MATCH_IP
actions: output enqueue set_vlan_vid set_vlan_pcp strip_vlan mod_dl_src mod_dl_dst mod_nw_src
mod_nw_dst mod_nw_tos mod_tp_src mod_tp_dst
 1(s1-h1): addr:f2:68:72:19:45:e4
     config:     0
     state:      0
     current:    10GB-FD COPPER
     speed: 10000 Mbps now, 0 Mbps max
 2(s1-h2): addr:d2:c7:cd:08:a6:71
     config:     0
     state:      0
     current:    10GB-FD COPPER
     speed: 10000 Mbps now, 0 Mbps max
 LOCAL(s1): addr:4e:99:31:53:5a:4a
     config:     PORT_DOWN
     state:      LINK_DOWN
     speed: 0 Mbps now, 0 Mbps max
OFPT_GET_CONFIG_REPLY (xid=0x4): frags=normal miss_send_len=0
```

xterm

```
pef@cube:~$ sudo ovs-vsctl show
c5589fdd-97a4-454c-a459-15e200562b5a
    Bridge "s1"
        Port "s1-h1"
            Interface "s1-h1"
        Port "s1-h2"
            Interface "s1-h2"
        Port "s1"
            Interface "s1"
                type: internal
    ovs_version: "2.6.1"
```

Configuration d'une adresse IP pour h1 et h2 appartenant au même réseau

xterm
```
ip netns exec h1 ip a add dev h1-eth0 10.0.0.1/24
ip netns exec h2 ip a add dev h2-eth0 10.0.0.2/24
```

Test de la connectivité

xterm
```
pef@cube:~$ sudo ip netns exec h2 ping -c 1 10.0.0.1
PING 10.0.0.1 (10.0.0.1) 56(84) bytes of data.
64 bytes from 10.0.0.1: icmp_seq=1 ttl=64 time=0.967 ms

--- 10.0.0.1 ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 0.967/0.967/0.967/0.000 ms
```

Écoute du traffic directement sur l'interface d'un netns

xterm
```
root@cube:~# tcpdump -l -i s1-h1
20:38:51.697615 ARP, Request who-has 10.0.0.2 tell 10.0.0.1, length 28
20:38:51.698127 ARP, Request who-has 10.0.0.1 tell 10.0.0.2, length 28
20:38:51.698189 ARP, Reply 10.0.0.1 is-at d2:a8:f5:cc:9d:8b (oui Unknown), length 28
20:38:51.698289 ARP, Reply 10.0.0.2 is-at 7a:68:37:b4:47:0e (oui Unknown), length 28
20:39:01.781418 IP 10.0.0.2 > 10.0.0.1: ICMP echo request, id 3410, seq 1, length 64
20:39:01.781492 IP 10.0.0.1 > 10.0.0.2: ICMP echo reply, id 3410, seq 1, length 64
```

Pour changer le prompt et indiquer le netns du shell courant

xterm
```
root@cube:~# cat change_prompt
INTERFACE=$(ip l | awk -F"[ -]" '/eth0/ { print "[" $2 "]" }')
PS1="$INTERFACE$PS1"
#PS1="h1:\w "
root@cube:~# source change_prompt
[h1]root@cube:~#
```

Pour remplacer un fichier de configuration dans un netns : le fichier resolv.conf pour la résolution DNS

xterm
```
root@ishtar:~# mount --bind /home/rezo/resolv.conf /etc/resolv.conf
root@ishtar:~# more /etc/resolv.conf
nameserver 8.8.8.8
nameserver 8.8.4.4
```

Attention

Ne marche que pour le shell qui a exécuté la commande : une fois le shell fermé le point de montage n'existe plus.

Pour utiliser un shell dans un netns

xterm
```
pef@cube:~# ip netns exec h2 sudo -u pef bash
pef@cube:~$ ip link
1: lo: <LOOPBACK> mtu 65536 qdisc noop state DOWN mode DEFAULT group default qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
8: h2-eth0@if7: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue state UP mode DEFAULT
group default qlen 1000
    link/ether 7a:68:37:b4:47:0e brd ff:ff:ff:ff:ff:ff link-netnsid 0
```

En lançant simultanément un shell dans le netns «h1» et «h2»

xterm
```
[h2]root@cube:~# python -m SimpleHTTPServer
Serving HTTP on 0.0.0.0 port 8000 ...
```

On lance un serveur Web basé sur le module Python «SimpleHTTP-Server» sur `h2` *et on lance la commande nmap sur* `h1` *à destination de* `h2`.

xterm
```
[h1]root@cube:~# nmap 10.0.0.2

Starting Nmap 7.40 ( https://nmap.org ) at
2017-09-07 22:05 CEST
Nmap scan report for 10.0.0.2
Host is up (0.000015s latency).
Not shown: 999 closed ports
PORT     STATE SERVICE
8000/tcp open  http-alt
MAC Address: BA:E5:AF:FC:22:AC (Unknown)

Nmap done: 1 IP address (1 host up) scanned
in 14.49 seconds
```

Sandboxing ou isolation

Limiter les capacités d'un processus :
- ▷ utilisation des ressources : CPU, mémoire, etc.
- ▷ lire/écrire de fichiers utilisateurs ;
- ▷ lire/écrire la mémoire des autres processus ;
- ▷ lire la configuration de l'hôte et accéder au réseau.

Utilisation :
- ▷ pour la sécurité : limiter les actions d'un programme malveillant, isoler des serveurs logiciels ;
- ▷ pour le «*cloud computing*» : partager des ressources entre différents clients, simulation réseau.

Les moyens :
- □ **gestion de ressources** :
  - ◇ nofile : nombre maximal de fichiers ouverts ;
  - ◇ nproc : nombre maximal de processus
  - ◇ locks : nombre maximal de verrous sur des fichiers.
- □ **cgroup**, «*control group*» : allouer, gérer les priorités, empêcher, surveiller l'usage des ressources système, comme en particulier :
  - ◇ le processeur : control CPU usage ;
  - ◇ la mémoire : control memory usage ;
  - ◇ les périphériques : blkio, «control block device», «devices» : control device access ;
  - ◇ l'ordonnancement : control process status
- □ **namespace** :
  - ◇ «*mounts namespace*» : point de montage du système de fichier ;
  - ◇ «*PID namespace*» : les identifiants de processus ;
  - ◇ «*IPC namespace*» : les communications inter-processus ;
  - ◇ «*UTS namespace*» : nom de l'hôte et nom de domaine ;

Lors de la conception d'un réseau pour une organisation (une société, un campus universitaire *etc.*) répartie entre un ou plusieurs sites (salles, bureaux, bâtiments, *etc.*), il est possible d'utiliser :

- l'**interconnexion** de différents réseaux locaux par routage ;
- la **commutation** au sein d'un même réseau local.

Interconnexion de différents réseaux

Elle utilise les capacités de la couche 3, «la couche réseau», pour :

- ⋆ définir des @IPs de sous-réseaux (en utilisant VLSM et CIDR) ;
- ⋆ affecter ces @IPs à différents réseaux locaux mis en œuvre dans l'organisation ;
- ⋆ interconnecter ces réseaux à l'aide de routeur(s) ;
- ⋆ définir des tables de routage :
  - ◇ pour un poste de travail : au minimum, la route par défaut, ou passerelle ;
  - ◇ pour un routeur : au minimum, la connaissance des différents réseaux interconnectés joignables à un saut. *Plus tard, on verra l'utilisation de protocole de construction de table de routage comme IGP, OSPF ou RIP*.

Commutation au sein d'un même réseau local

Elle utilise les capacités de la couche 2, «la couche liaison de données», pour :

- ▷ décomposer un réseau local en plusieurs parties indépendantes ;
- ▷ créer différents **domaines de collisions** dans un même **domaine de diffusion**.

Croissance du réseau Internet

Le protocole IP a été utilisé intensivement depuis des dizaines d'années. Son succès s'est accompagné d'une croissance exponentielle du réseau Internet.

Problème du manque d'adresses

Le réseau basé sur IP ne peut plus s'agrandir par épuisement du nombre d'adresses disponibles :

- 1987 : 100 000 réseaux prévues après plusieurs décennies ;
- 1996 : 100 000 réseaux effectifs ;
- 2012 : épuisement des adresses IPv4 disponibles.

L'adressage sur 32 bits permet de définir **2 milliards d'adresses** potentielles.

Mais l'**organisation de l'espace d'adressage** en classes en consomme des millions :

- *les réseaux de classe A sont beaucoup trop grands (16 millions d'adresses) ;*
- *les réseaux de classe C sont trop petits (256 adresses) ;*
- *les réseaux de classe B sont trop grands (65536 machines).*

Problème de la taille des tables de routage pour aller d'un réseau à l'autre

L'espace des adresses IP est une hiérarchie à deux niveaux :

```
<identifiant réseaux>.<identifiant machine>
```

Les routeurs doivent connaître l'identifiant de tous les réseaux !

La taille des tables de routage devient **énorme** et **difficile à gérer** :

- occupation mémoire trop importante dans les routeurs ;
- complexité des algorithmes de gestion de ces tables de routage ;
- routeur conçu dans la perspective d'un réseau Internet contenant 10 000 réseaux, et où les 100 000 réseaux étaient un avenir lointain (limitation du protocole de routage RIP) ;
- transmission des tables de routage entre routeurs propice aux erreurs, et aux pertes…

Le «subnetting»

Où comment partitionner un réseau d'une classe donnée en différents sous-réseaux.

Fonctionnement : Une structure obtient un seul réseau de classe A, B ou C :

- elle décompose ce réseau en différents sous-réseaux qu'elle peut répartir géographiquement ;
- elle peut faire du routage entre ces différents sous-réseau ;
- *Pas de différence vu de l'extérieur du réseau !*

Le «supernetting»

Où comment supprimer le concept de classe réseau et optimiser l'usage des adresses restantes.

Fonctionnement : Une structure obtient des «tranches de réseau» :

- par exemple : pour disposer d'au plus 500 machines elle obtient deux réseaux de classe C (2*256) ;
- pour simplifier le routage, on considère que cette tranche **ne désigne qu'un seul et même réseau**, ce qui oblige à dispose de certaines propriétés sur les adresses de ces réseaux ;
- *Pas de différence vu de l'extérieur du réseau !*

Le NAT, Network Address Translation

Où comment partager une ou plusieurs adresses entre différentes machines.

Fonctionnement : Une structure obtient une ou plusieurs adresses fixes :

- ces adresses sont **partagées** entre différentes machines connectées dans un **réseau privé** ou Intranet (`192.168.x.Y`, `172.16.x.y` ou `10.x.y.z`).
- **Attention** *: différence vu de l'extérieur du réseau !*

IPv6

Où comment passer à Internet NextGen ! *Avec des adresses sur 128 bits entre autre…*

En utilisant la notion de classe

Il suffisait de comparer les `<id. réseau>` des deux @IP suivant la **répartition donnée par la classe**, mais…

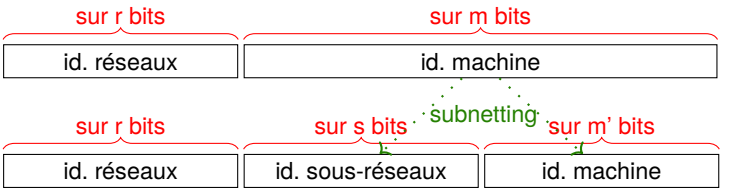
La notion de sous-réseau ou *Subnetting*

But Partition d'un réseau en **différents sous-réseaux**.

Avantages Utiliser le réseau global fourni par le NIC à l'entité pour disposer de **réseaux autonomes** au sein de cette même entité

Exemple : l'université de Limoges avec les machines du site de Condorcet, du campus de Vanteaux, de La Borie…

Mise en œuvre Définition de sous-réseaux en découpant `<id. machine>` en deux parties :



avec r + s + m' = 32.

Le découpage par multiple de 8bits facilite le travail des routeurs, mais n'est pas obligatoire.

Une machine connectée à un sous-réseau doit connaître :

- son @IP ;
- le nombre de bits attribués à l'`<id. réseau + sous-réseau>` ;

Remarque : l'ensemble des sous-réseaux d'un même réseau est vu de l'extérieur comme un **unique réseau** (routage, courrier…).

Masque de sous-réseau (subnet mask)

c'est un mot de 32 bits contenant :

- ⋆ des bits à 1 à la place de l'identifiant réseau/sous-réseau,
- ⋆ des bits à zéro en lieu et place de l'identifiant machine.

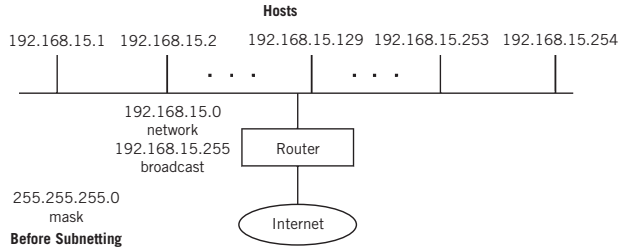
Ainsi, `255.255.255.0` indique que les premiers 24bits désignent le sous-réseau.

Nouvelle notation CIDR : `@IP / nombre de bits pour identifiant réseau`

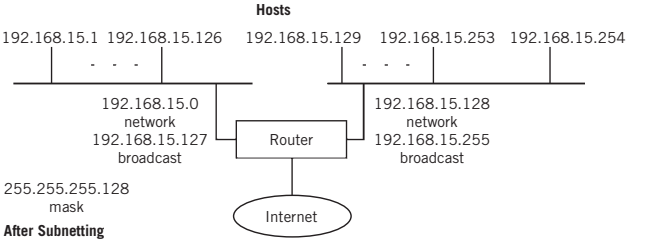
Exemple : `164.81.55.0/24` pour désigner un réseau ou bien `164.81.55.12/24` pour une machine.

Subnetting : RFC 950

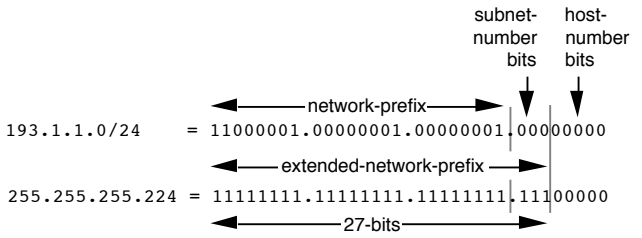
Avant



Après



Exemple



Découpage du réseau `193.1.1.0/24` en 8 sous-réseaux :

* `193.1.1.0/27`
* `193.1.1.32/27`
* …
* `193.1.1.224/27`

Pour décider du nombre de bits à affecter aux sous-réseaux

- de combien de réseaux l'organisation a besoin et dans le futur ?
- combien de machines au maximum dans un des sous-réseaux, dans le futur ?

Attention

- La RFC 950 autorise un **seul masque de sous-réseau** pour le *subnetting* dans une même structure.
- Sur de vieux routeurs, *classful*, on évite d'utiliser le sous réseau «tout à zéro» et «tout à 1».

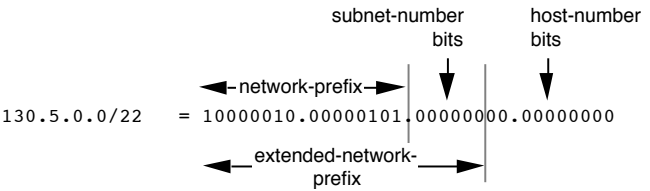
But : Permettre d'utiliser plusieurs masques de sous-réseaux pour le subnetting, *d'où la notion de «variable length»*.

Exemple

On dispose du réseau `130.5.0.0/16` à décomposer :

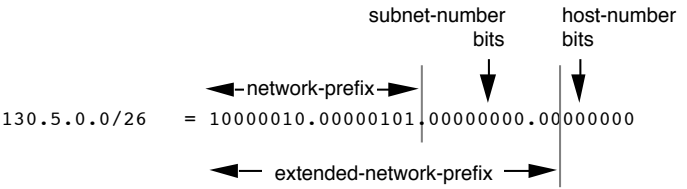
En `/22`, ce qui donne 64 sous-réseaux, pour 1022 machines chacun :

Si un réseau ne contient que 30 machines, on perd énormément d'adresses !



En `/26`, ce qui donne 1024 sous-réseaux, pour 62 machines chacun :

C'est mieux, mais l'idéal serait de disposer du `/22` et du `/26` simultanément.



Contraintes

- ▷ plusieurs masques de sous-réseau compatibles **hiérarchiquement** ;
- ▷ l'utilisation de **routeurs** qui emploient :
  - ◇ des protocoles de construction de table de routage qui transmettent le masque de sous-réseau pour chaque réseau destination ;
  - ◇ un algorithme de «*forwarding*» qui utilise le «*longest matching*» ;
  - ◇ une affectation des sous-réseaux qui soit topologiquement adapté.

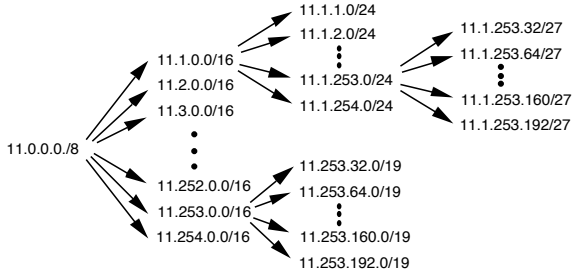
Algorithme de forwarding basé sur le «longest match»

Pour un datagramme à destination de l'adresse `11.1.2.5`, on sélectionne la route qui correspond avec le plus grand nombre de bits d'un des sous-réseaux, ici, `11.1.2.0/24` ★

```
Destination    11.1.2.5     = 00001011.00000001.00000010.00000101
Route #1       11.1.2.0/24  = 00001011.00000001.00000010.00000000
Route #2       11.1.0.0/16  = 00001011.00000001.00000000.00000000
Route #3       11.0.0.0/8   = 00001011.00000000.00000000.00000000
```

Attention : la machine `11.1.2.5` doit bien être connectée dans le réseau `11.1.2.0/24`. *Dans le réseau `11.1.0.0/16` elle ne serait pas accessible !*

Aggrégation de routes

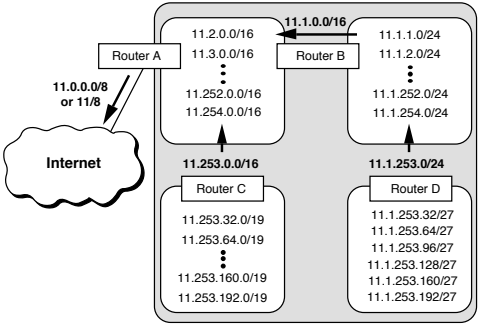


L'aggrégation permet de simplifier le routage : le réseaux `11.0.0.0/8` est décomposé en `/16`, puis le sous-réseau `11.1.0.0/16` est décomposé en `/24`.

Le processus est récursif, mais **n'impose pas** *que la taille du préfixe soit la même à chaque niveau de l'arbre.*

On diminue la taille des tables de routage nécessaires :

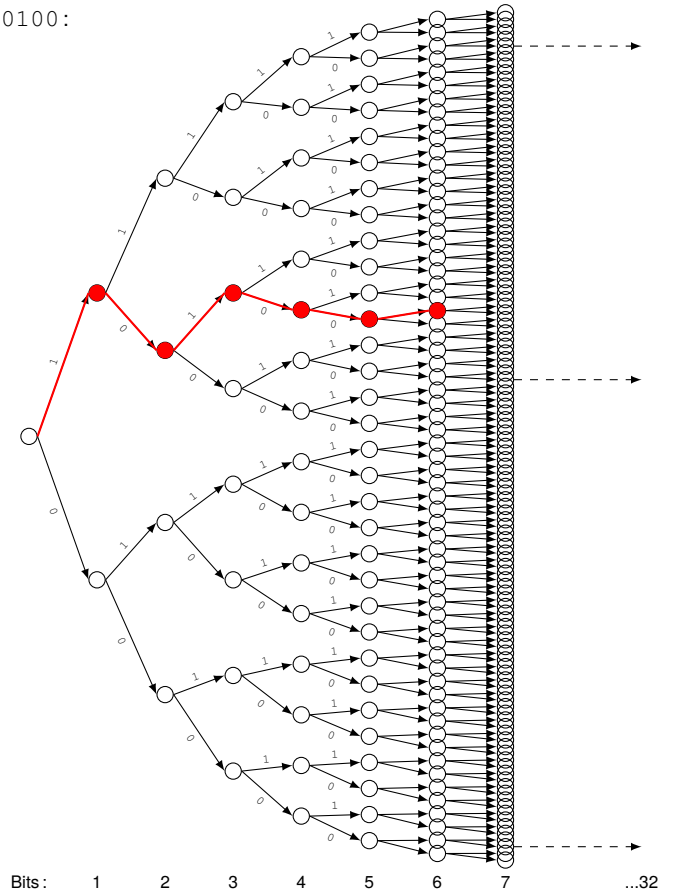
- le routeur D «cache» 6 sous-réseaux derrière `11.1.253.0/24` ;
- le routeur C cache 6 sous-réseaux avec `11.253.0.0/16` ;
- la structure des sous-réseaux est cachée de l'extérieur avec `11.0.0.0/8`.



Préfixe et routage

Soit l'adresse IP `164.x.y.z`, $164 \Longrightarrow$ `10100100` :

Chaque bit de l'adresse définit un chemin différent et permet d'atteindre la destination...



CIDR : routage sans classe, RFC 1519

La technique du CIDR “Classless InterDomain Routing” RFC 1519

Allouer des adresses de classe C (2 millions disponibles) sous forme de blocs de taille variable.

Exemple :

- ▷ *si un site a besoin de 2000 adresses, on lui alloue un bloc de 2048 adresses (8 réseaux de classe C contigus).*
- ▷ *si un site a besoin de 8000 adresses on lui alloue un bloc de 8192 adresses (32 réseaux de classe C contigus).*

Ce principe est étendu vers les réseaux de classe A et B également.

Distribution des adresses de manière régionale, RFC 1174

Les règles d’attribution d’adresses de classe C sont changées, le monde est partagé en 4 zones :

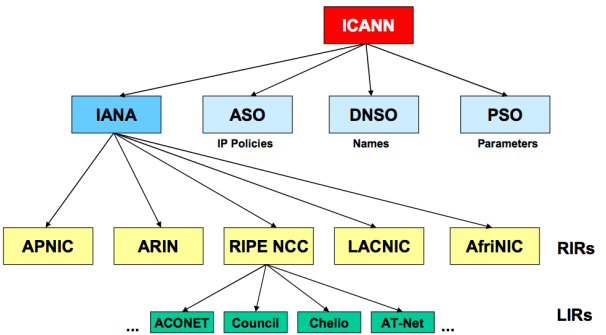
- ⋆ 194.0.0.0 à 195.255.255.255 sont attribuées à l’Europe, **RIPE NCC** et à l’Afrique, **AfriNIC** ;
- ⋆ 198.0.0.0 à 199.255.255.255 sont attribuées à l’Amérique du Nord, **ARIN** ;
- ⋆ 200.0.0.0 à 201.255.255.255 sont attribuées à l’Amérique du Sud & Centrale, **LACNIC** ;
- ⋆ 202.0.0.0 à 203.255.255.255 sont attribuées à l’Asie et au Pacifique, **APNIC** ;
- ⋆ 204.0.0.0 à 223.255.255.255 sont conservées en réserve.
- ⋆ 64.0.0.0 à 127.0.0.0 sont conservés pour la «fin d’IPv4»…

Chaque région dispose ainsi de 32 millions d’adresses de classe C.

Les organismes chargés d’allouer les adresses sont des RIRs, *Regional Internet Registries*.

Simplification du routage

Quand un routeur hors d’Europe reçoit une adresse à destination de 194.x.y.z ou 195.x.y.z, il doit l’expédier vers un routeur européen.



Les différents préfixes possibles

Prefix Length	Dotted Decimal Netmask	Number of Classful Networks	Number of Usable IPv4 Addresses
/1	128.0.0.0	128 Class A's	2,147,483,646
/2	192.0.0.0	64 Class A's	1,073,741,822
/3	224.0.0.0	32 Class A's	536,870,910
/4	240.0.0.0	16 Class A's	268,435,454
/5	248.0.0.0	8 Class A's	134,217,726
/6	252.0.0.0	4 Class A's	67,108,862
/7	254.0.0.0	2 Class A's	33,554,430
/8	255.0.0.0	1 Class A or 256 Class B's	16,777,214
/9	255.128.0.0	128 Class B's	8,388,606
/10	255.192.0.0	64 Class B's	4,194,302
/11	255.224.0.0	32 Class B's	2,097,150
/12	255.240.0.0	16 Class B's	1,048,574
/13	255.248.0.0	8 Class B's	524,286
/14	255.252.0.0	4 Class B's	262,142
/15	255.254.0.0	2 Class B's	131,070
/16	255.255.0.0	1 Class B or 256 Class C's	65,534
/17	255.255.128.0	128 Class C's	32,766
/18	255.255.192.0	64 Class C's	16,382
/19	255.255.224.0	32 Class C's	8,190
/20	255.255.240	16 Class C's	4,094
/21	255.255.248.0	8 Class C's	2,046
/22	255.255.252.0	4 Class C's	1,022
/23	255.255.254.0	2 Class C's	510
/24	255.255.255.0	1 Class C	254
/25	255.255.255.128	1/2 Class C	126
/26	255.255.255.192	1/4 Class C	62
/27	255.255.255.224	1/8 Class C	30
/28	255.255.255.240	1/16 Class C	14
/29	255.255.255.248	1/32 Class C	6
/30	255.255.255.252	1/64 Class C	2
/31	255.255.255.254	1/128 Class C	0
/32	255.255.255.255	1/256 Class C (1 host)	– (1 host route)

Attention

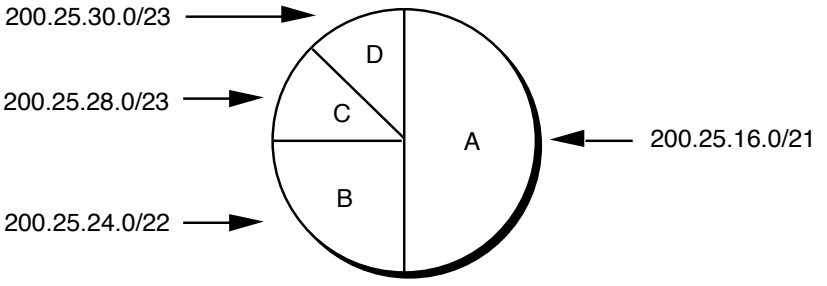
Le préfixe `\31` est réservé aux connexions «point à point» entre routeurs : avec un identifiant machine réduit à 0 ou à 1.

Avec des routeurs CISCO, il est également possible d'utiliser des «unnumbered interfaces» :

```
interface Serial0
 ip unnumbered Ethernet0
```


Gestion de l'affectation de sous-réseau par un FAI

Soit le réseau `200.25.0.0/16` géré par un FAI, «Fournisseur d'accès internet» ou ISP, «Internet service provider».
Il veut décomposer le réseau `200.25.16.0/20` suivant :




```
ISP's Block  200.25.16.0/20
Org A:200.25.16.0/21
Reserved:  200.25.24.0/21
```

Puis :

```
Reserved 200.25.24.0/21
Org B:200.25.24.0/22
Reserved 200.25.28.0/22
```

et enfin :

```
Reserved 200.25.28.0/22
Org C:200.25.28.0/23
Org D: 200.25.30.0/23
```

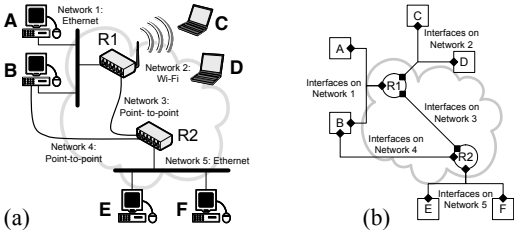
Le CIDR est similaire au VLSM

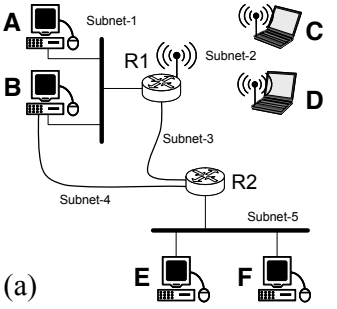
- le VLSM est réalisé à l'intérieur du réseau de l'organisation et il est caché de l'extérieur ;
- le CIDR est réalisé par les différents organismes, FAI, et il est visible de l'extérieur.

La configuration des tables de routages n'est pas la même, car le CIDR soit être communiqué à toutes les autres organisations.

Une abstraction du réseau et sa configuration en CIDR

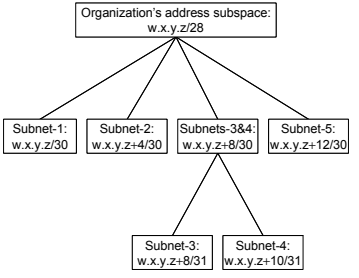
Une première abstraction du réseau indépendamment de l'adresse réseau de l'organisation :

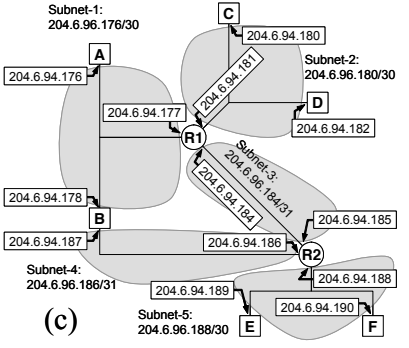




Une seconde abstraction du réseau avec la notion de «sub-net» et l'adresse réseau : `204.6.96.176/28`

Subnet	Subnet mask	Network prefix	Interface addresses
1	204.6.94.176/30	11001100 00000110 01011110 101100--	A: 204.6.94.176
			R1-1: 204.6.94.177
			B-1: 204.6.94.178
2	204.6.94.180/30	11001100 00000110 01011110 101101--	C: 204.6.94.180
			R1-2: 204.6.94.181
			D: 204.6.94.182
3	204.6.94.184/31	11001100 00000110 01011110 1011100-	R1-3: 204.6.94.184
			R2-1: 204.6.94.185
4	204.6.94.186/31	11001100 00000110 01011110 1011101-	R2-2: 204.6.94.186
			B-2: 204.6.94.187
5	204.6.94.188/30	11001100 00000110 01011110 101111--	R2-3: 204.6.94.188
			E: 204.6.94.189
			F: 204.6.94.190





Pour configurer une interface réseau sous Gnu/Linux

```
xterm
$ ifconfig -a
```

permet d'afficher toutes les interfaces réseaux présentes sur la machine qu'elle soit ou non activée.

```
xterm
$ sudo ifconfig eth0 up
```

Active l'interface réseau (`down` pour la désactiver).

```
xterm
$ sudo ifconfig eth0 192.168.1.1/24
```

permet de définir l'@IP d'une interface réseau.

```
xterm
$ sudo ifconfig eth0 0.0.0.0/0
```

supprime l'@IP associée à l'interface.

Pour configurer une interface avec DHCP, «Dynamic Host Configuration Protocol»

```
xterm
$ sudo dhclient eth0
```

lance la configuration de l'interface par DHCP (@IP + @IP passerelle + @IP DNS + *etc.*).

```
xterm
$ sudo dhclient eth0 - r
```

dé-configure l'interface en «libérant» la configuration IP

Pour configurer la table de routage

```
xterm
$ sudo route
```

Affiche la table de routage du système.

```
xterm
$ sudo route add default gw 192.168.1.254 eth0
```

ajoute la passerelle «par défaut» vers le routeur `192.168.1.254`.

```
xterm
$ sudo route add -net 172.16.10.10/24 gw 192.168.1.254
```

ajoute un chemin de routage vers le réseau indiqué (l'option `del` au lieu de `add` permet la suppression de la route).

```
xterm
sudo route add -host 98.76.54.32 gw 12.34.56.1
```

ajoute un chemin pour un «host» donné.

Il **vaut mieux** utiliser la commande «`ip`», du package `iproute2`.

Afficher le configuration réseau

```
xterm
$ ip addr
$ ip link
```

afficher la liste des interfaces qu'elles soient ou non activées.

```
xterm
$ ip route
```

affiche la table de routage.

Configuration d'une interface

```
xterm
$ sudo ip addr add 172.16.0.1/24 dev tap0
$ sudo ip link set tap0 up
```

configurer l'interface et l'activer.

```
xterm
$ ip link set eth0 promisc on
```

passer l'interface en mode «promiscuous» pour laisser passer les trames qui ne sont pas à destination de cette interface.

```
xterm
$ ip link set eth0 mtu 1500
```

configure la MTU de l'interface.

Configuration de la table de routage

```
xterm
$ ip route add 10.0.0.0/24 via 193.233.7.65
```

ajouter une route vers un réseau en passant par un routeur.
On utilisera `del` pour enlever cette route.

```
xterm
$ ip route add default via 192.168.0.254
```

ajouter la route par défaut.

Exemples

xterm
```
$ ip link list
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 16436 qdisc noqueue state UNKNOWN
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
2: eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast state UNKNOWN qlen 1000
    link/ether 00:0c:29:9d:ea:19 brd ff:ff:ff:ff:ff:ff
```

xterm
```
$ ip address show
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 16436 qdisc noqueue state UNKNOWN
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
    inet6 ::1/128 scope host valid_lft forever preferred_lft forever
2: eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast state UNKNOWN qlen 1000
    link/ether 00:0c:29:9d:ea:19 brd ff:ff:ff:ff:ff:ff
    inet 192.168.127.133/24 brd 192.168.127.255 scope global eth0
    inet6 fe80::20c:29ff:fe9d:ea19/64 scope link valid_lft forever preferred_lft forever
```

xterm
```
$ ip route show
192.168.127.0/24 dev eth0  proto kernel  scope link  src 192.168.127.131  metric 1
default via 192.168.127.2 dev eth0  proto static
```


Déclencher les fonctions de routage sous GNU/Linux

À l'aide de la commande `sysctl` :

```
sudo sysctl net.ipv4.ip_forward=1
sysctl net.ipv4.ip_forward
```

La ligne 1 active le «forwarding» et la ligne 2 vérifie l'activation.

Pour le rendre permanent, c-à-d activé lors du prochain redémarrage, il faut ajouter la ligne suivante dans le fichier `/etc/sysctl.conf`

```
net.ipv4.ip_forward = 1
```

Pour prendre en compte la modification du fichier :

```
sudo sysctl -p /etc/sysctl.conf
```

Il est également possible d'utiliser les fichiers spéciaux du répertoire `/proc` :

```
echo 1 > /proc/sys/net/ipv4/ip_forward
```

Le routage aura lieu entre les différentes interfaces présentes sur le système, en accord avec la table de routage du système.

Domaine de diffusion

Pour un réseau à diffusion, on définit le «domaine de diffusion» comme étant la **zone de taille maximale** où il est possible de recevoir un message envoyé dans la zone.

Par exemple dans Ethernet, un domaine de diffusion définie la limite de la transmission d'une trame (qu'elle soit transmise vers l'@MAC d'une machine, ou bien l'@MAC de diffusion FF:FF:FF:FF:FF:FF) : en dehors de cette zone de diffusion, il n'est pas possible de recevoir la trame envoyée.

La taille de la zone de diffusion est limitée :

- par des contraintes techniques : limitation de la transmission physique (WiFi, câbles Ethernet trop long), choix technologiques, *etc.*
- pour des décisions administratives :
  - limitation des transmissions à un ensemble choisi de matériels,
  - augmentation le nombre de machines connectées à un même réseau ;
  - amélioration les débits (moins de partage du réseau), application de QoS, «Quality of Service» ;
  - prise en compte de la sécurité (isoler les postes) ;
  - *etc.*

Domaine de collision

Dans un réseau à diffusion, lorsqu'au moins deux messages sont diffusés par deux entités **autonomes** sans contrôle centralisé d'une entité d'arbitrage, il existe un risque de **collision**.
Cela est vrai dans Ethernet 10/100 Mbps, faux dans Ethernet Gigabit, WiFi par exemple.

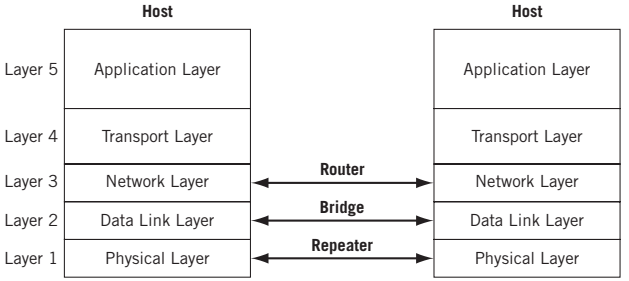
Un **domaine de collision** est une zone du réseau où il existe une **compétition** pour l'accès à cette zone.
Le résultat de cette compétition peut être la création de collision et une baisse de débit par rapport à la capacité du réseau.

La commutation : une affaire de matériel

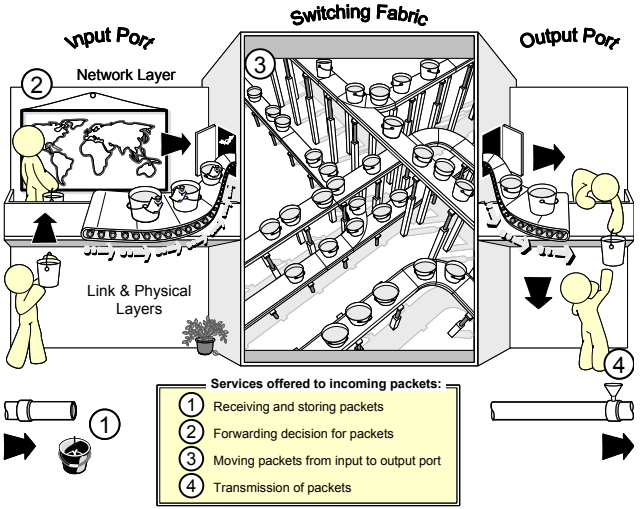
Le matériel utilisé pour réaliser cette commutation :

- le répéteur, «repeater», pour allonger les distances physiques de transmission ;
- le pont, «bridge», pour connecter des domaines de diffusion avec intelligence ;
- le «concentrateur commuté» ou «switch», pour connecter et **isoler** *surtout*.

Couches, commutation & routage



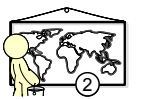
- Le routeur intervient sur la couche 3.
  Il retransmet, «forward», des **datagrammes** ;
  Il réalise du **routage**.
- Le pont intervient sur la couche 2.
  Il retransmet, «forward», des **trames** ;
  Il réalise de la **commutation**.
- Le répéteur intervient sur la couche 1.
  Il ne **commute pas** mais allonge les distances.
  Il est équivalent à un **«hub»**.
  Il retransmet des **bits** à 1 ou à 0.



1. récupérer les paquets (datagramme ou circuit virtuel) ;
2. décider de la manière de «faire suivre» le paquet : *forwarding decision* ;
3. commuter un circuit d'acheminement interne reliant l'entrée à la sortie choisie ;
4. transmettre le paquet sur ce circuit interne.

- une ligne de transmission correspond à un tuyau dans lequel s'écoule les données (des bits) ;
- un paquet correspond à une «portion» de ces données : il est matérialisé par un seau.
- un paquet entre dans le routeur par l'«input» et sort par l'«output» ;
- au milieu se trouve un «switch», un commutateur sélectionnant la sortie du paquet sur une ligne de transmission.

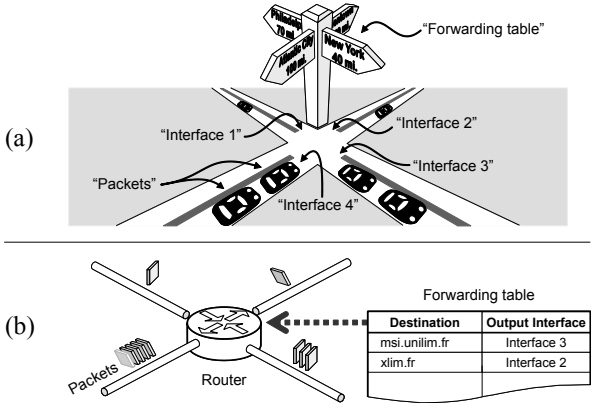
Receiving and storing packets



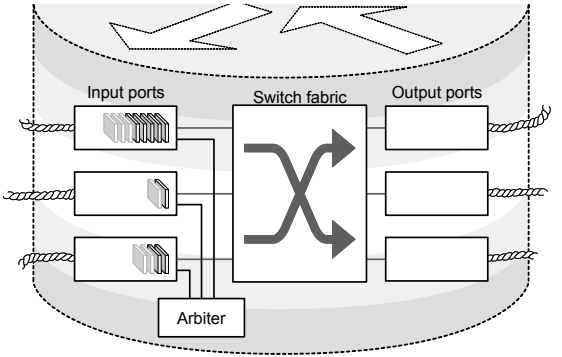
Forwarding decision for packets

Moving packets from input to output port

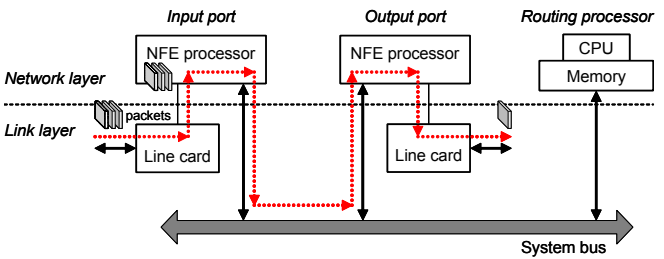
Transmission of packets

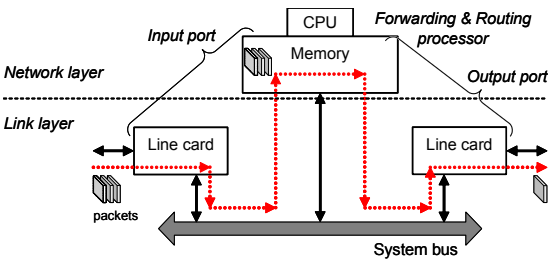


Le routeur décide en fonction du contenu de sa table de routage, «forwarding table», de l'aiguillage des paquets.



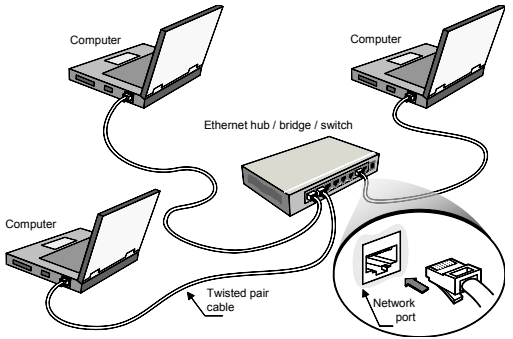
Il possède des **circuits électroniques** spécialisés, des NFE, «Network Flow Engine» :





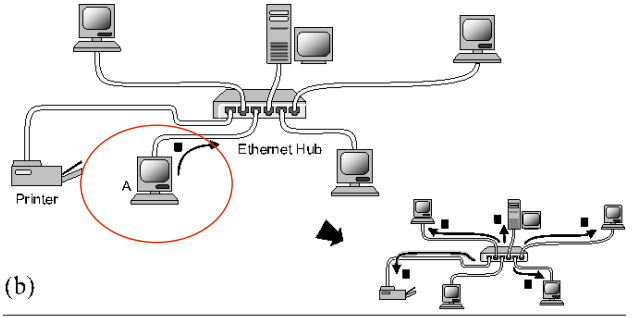
Cette architecture matérielle :

- ▷ permet d'atteindre des débits très élevés : délais de traversé des paquets, gestion de table de routage importante, gestion de la QoS, gestion de «circuit-virtuel», gestion de **Firewall** *etc* ;
- ▷ fait la différence avec des architectures non dédiées comme un matériel générique sous GNU/Linux : plus de port de connexion pour ligne de transmission, support de la fibre optique, alimentation protégée *etc.*
- ▷ quelques sociétés sont présentes : CISCO System, Juniper Networks, Hewlett-Packard, Huawei.

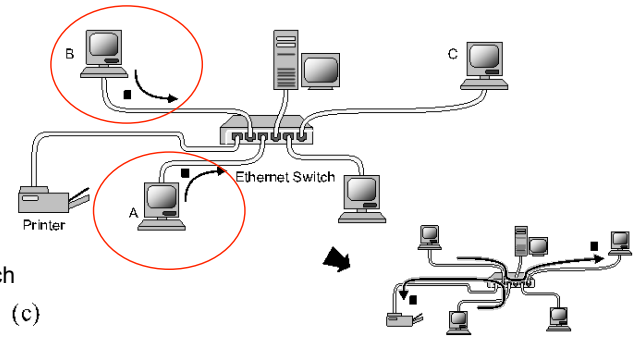


- un hub ou répéteur transmet la trame sur une sortie pendant qu'elle est reçue sur une entrée : on parle de «*cut-through switching*» ;
- un switch ou bridge reçoit d'abord toute la trame et la stocke ; il attend alors que le réseau auquel est connecté le port de sortie soit libre pour la transmettre : on parle de «*store-and-forward switching*».

Ethernet Hub



Ethernet Switch

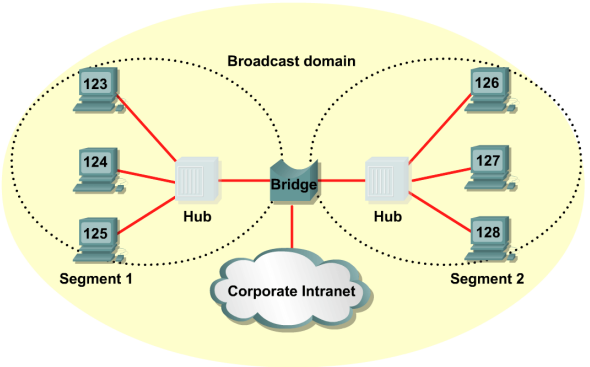


Sur le schéma, on constate que différents ports peuvent communiquer simultanément (indiqués en rouge) dans le cas d'un switch : on comprend qu'il y a une forme de commutation et d'établissement de «circuit» physique.

Le pont

- connecte au moins deux **domaines de collision** ;
- ne crée qu'**un seul** domaine de diffusion ;
- **commute** une trame d'une ligne de transmission à une autre.

Fonctionnement :



Pour chaque trame, suivant l'@MAC de destination :

- «*forwarded*» : la trame est envoyée **uniquement** dans le segment où se trouve la destination ;
- «*filtered*» : la trame est détruite par le pont sans en informer la source ;
- «*flooded*» : la trame est envoyée à chaque segment attaché au pont (cas d'une @MAC de destination de broadcast ou de multicast).

Pour connaitre les domaines de collision, où les matériels entrent en compétition pour l'accès au support de transmission, le pont écoute les trames qui circulent et apprennent les @MAC utilisées dans ces trames.

Il peut y avoir des problèmes :

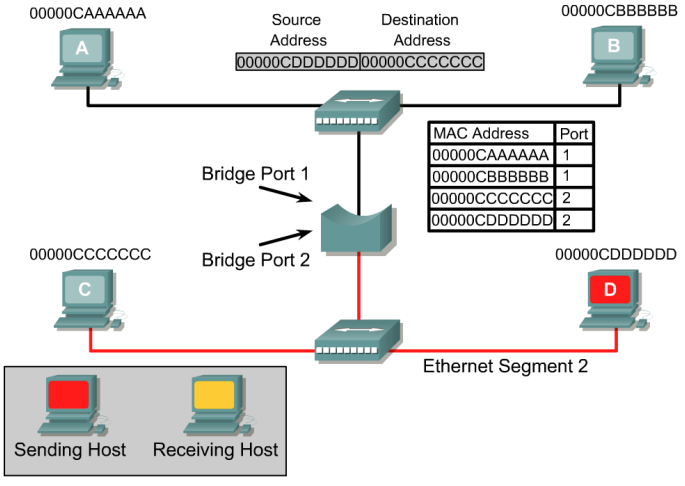
- ⋆ le trafic multicast peut encombrer inutilement le réseau (vidéoconférence par ex.) ;
- ⋆ la présence de plusieurs ponts peut créer des boucles.
  Pour découvrir ces ponts et désactiver ceux inutiles de manière dynamique, on utilise le SPT, «*Spanning Tree Protocol*» : en cas de panne de l'un d'entre eux, un pont désactivé peut redevenir actif automatiquement (il ne découvre plus l'autre).

Le pont

Le pont :

- dispose de deux «ports» de connexion réseau (prise Ethernet) ;
- apprend les @MAC des matériels connectés sur chacun de ces ports (au travers d'un «hub»), en surveillant la **destination** des trames qui y circulent.

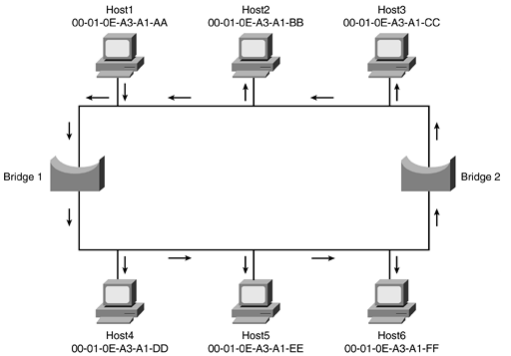
Exemple :



1. le pont établit une **table** où les entrées associent un port avec une @MAC (les entrées ont une durée de vie limitée) ;
2. une trame est envoyée :
   - ◇ de l'@MAC `00:00:0C:DD:DD:DD` ;
   - ◇ vers l'@MAC `00:00:0C:CC:CC:CC` ;
3. le pont :
   - ◇ vérifie les entrées de sa table : les deux @MAC sont sur le port 2 ;
   - ◇ l'@MAC de destination n'est pas une adresse de diffusion ni de multicast ;
   - ◇ ne «*forward*» pas la trame sur le port 1.
4. on évite de diffuser la trame sur une partie du LAN.

Sur l'exemple, les 4 machines sont configurées dans le même réseau IP.

Le STP, «Spanning Tree Protocol»

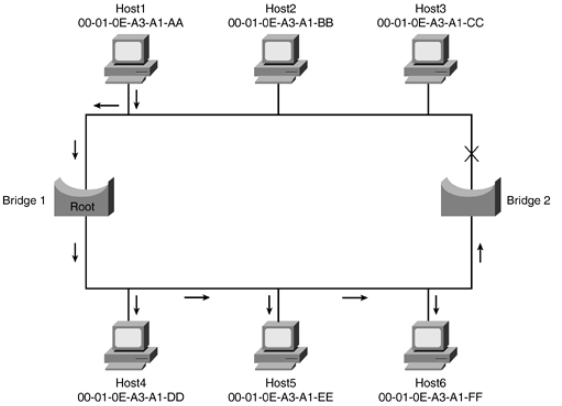


La présence de deux ponts crée une boucle dans le LAN :

a. les deux ponts reçoivent une trame émise par le Host1 à destination du Host6 dans le segment du haut ;
b. bridge1 retransmet la trame vers le segment du bas ;
c. bridge2 reçoit du trafic dans le segment du bas de Host1 à Host6 et modifie sa table pour mettre Host1 dans le segment du bas ;
d. bridge2 retransmet la trame dans le segment du haut ;
e. *etc.*

Le STP permet de **désactiver** les ponts pour éviter les boucles :

- les ponts étant accessibles les uns des autres par un «port», un pont identifie un autre pont par le port qu'il doit employer pour l'atteindre ;
- chaque pont existant conserve un unique pont qui doit lui permettre d'atteindre l'intégralité du réseau : le **pont racine**, ou «*root port*» parce qu'on l'atteint uniquement par un certain port.
- le choix du pont racine à conserver pour, le «*root port*», se fait à l'aide de trame diffusée au format STP, suivant :
  - la sélection du chemin le plus court pour l'atteindre, en nombre de pont à franchir ;
  - un identifiant numérique qui peut être choisi, le «*port ID*» lorsqu'il y a plusieurs chemins possibles ;
  - des intervalles de temps régulier afin d'éviter la rupture du réseau en cas de panne d'un pont.



Le pont : les risques

Attention !

Un pont sert d'intermédiaire de communication, c'est une sorte de «routeur» de niveau 2, il faut donc faire attention à des attaques de type MiTM, «*Man in The Middle*», où :

- un attaquant se fait désigner comme pont racine par les autres ponts et peut ainsi intercepter le trafic réseau. On parle «d'attaque sur le STP» ;
- un attaquant transmets des trames sous l'identité, adresse MAC, d'un autre poste pour se faire retransmettre les trames à destination de cet autre poste en corrompant les tables d'association (MAC spoofing) ;
- un attaquant transmets «énormément» de trame sous des identités différentes pour saturer la mémoire cache de la table associative du pont associée au port.
  Il arrive que dans ce cas le switch passe en mode concentrateur uniquement, «hub», et diffuse toute trame sur tous les ports (il arrête de travailler !).

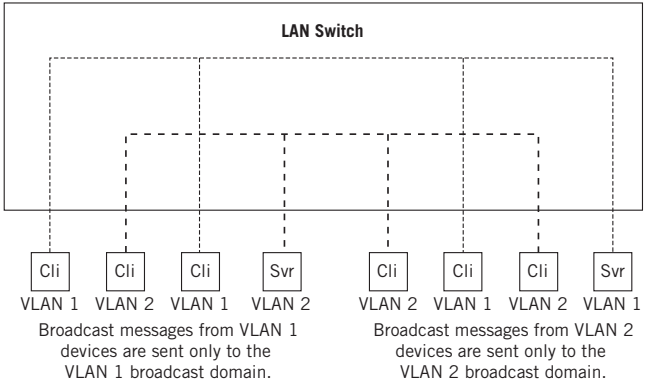
Le switch

Il permet :

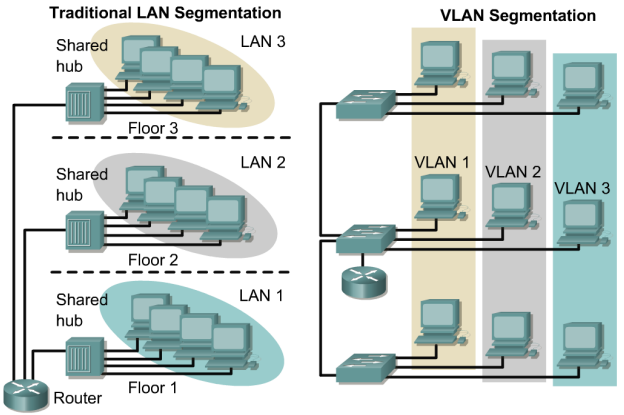
- de segmenter un domaine de diffusion entre plusieurs domaines de collisions, avec plus de deux ports de connexion à la manière d'un pont ;
- de créer plusieurs domaines de diffusion segmentés :
  - ◇ en associant un ou plusieurs ports à un domaine de diffusion ;
  - ◇ en assurant un travail de pont entre ces différents ports ;
  - ◇ en permettant l'utilisation des autres ports libres pour définir un ou plusieurs autres domaines de diffusion.

  Dans ce cas, on dira que le switch créé des LANs virtuels, appelés **VLANs**.

Les VLANs



Les domaines de diffusion, «broadcast domain», VLAN1 et VLAN2 sont des «entités logiques» dont les matériels sont connectés par des pont virtuels.

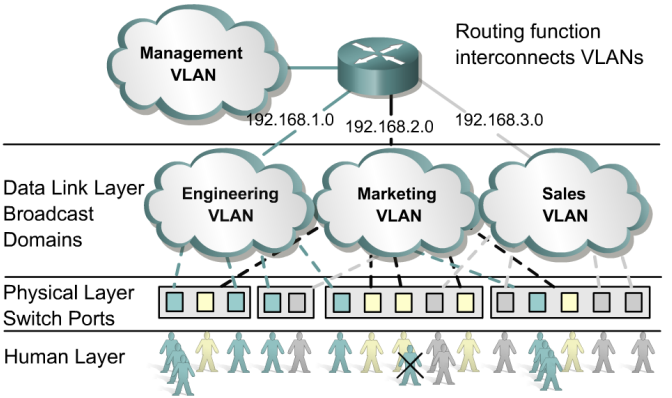


La segmentation du réseau de l'organisation peut être réalisée avec :

- une approche «traditionnelle» :
  - différents LAN utilisant des ponts, switch ou répéteurs ;
  - un routeur interconnectant ces différents LANs ;
- une approche «VLAN» :
  - un ou plusieurs switch en mode VLAN ;
  - un routeur interconnectant ces différents VLANs.

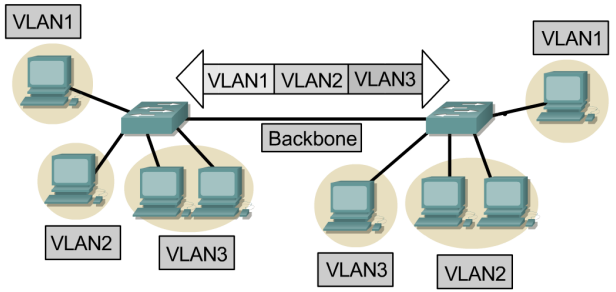
L'avantage du VLAN est sa flexibilité :

- il est possible d'associer un port à un VLAN particulier ;
- chacun de ses ports peut être connecté à un site particulier ;
- une personne peut rester dans le même bureau tout en migrant dans un VLAN différent ;
- son usage se combine avec celui d'un routeur :
  - *le routeur reste nécessaire aux échanges entre VLANs*



Les trames de VLAN

La gestion des VLANs se fait au travers de l'affectation d'un port d'un switch à un domaine de diffusion particulier, c-à-d à un VLAN particulier.

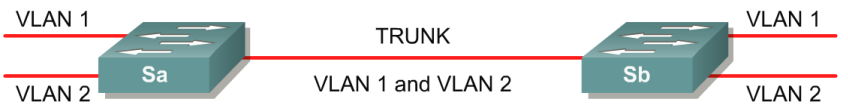


Lorsque plusieurs switchs doivent être reliés afin de créer les différents VLANs, il est nécessaire d'échanger entre ces switchs des trames appartenant à différents VLANs, il est **nécessaire** de savoir à quel VLAN appartient une trame.

- on doit «noter» à quel VLAN chaque trame appartient ;
- on utilise la norme IEEE 802.1Q, qui étend la trame 802.3 :
  - elle utilise le «frame tagging», c-à-d de l'étiquetage de trame ;
  - les étiquettes permettent de savoir à quel domaine de diffusion appartient la trame ;
  - il est possible de mettre plusieurs *tags* dans une trame

Le Trunk ou le multiplexage de VLAN

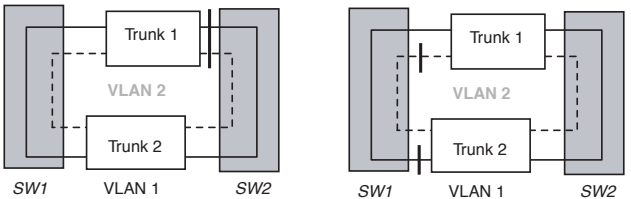
On appelle «Trunk» la liaison où circule les trames des différents VLANs.



Amélioration des protocoles «anti-boucles»

Le protocole STP, «Spanning Tree Protocol» à été modifié :

- RSTP, «Rapid Spanning Tree», «802.1w» : amélioration de l'arbre de recouvrement tenant compte des VLANs ;
- MSTP, «Multiple Spanning Tree», «802.1Q» : permet de créer des abres de recouvrement multiples pour les différents VLANS :



Configuration avec deux VLAN véhiculés sur deux trunks distincts entre les commutateurs SW1 et SW2.
Un gros trait signifie qu'un port est bloqué.

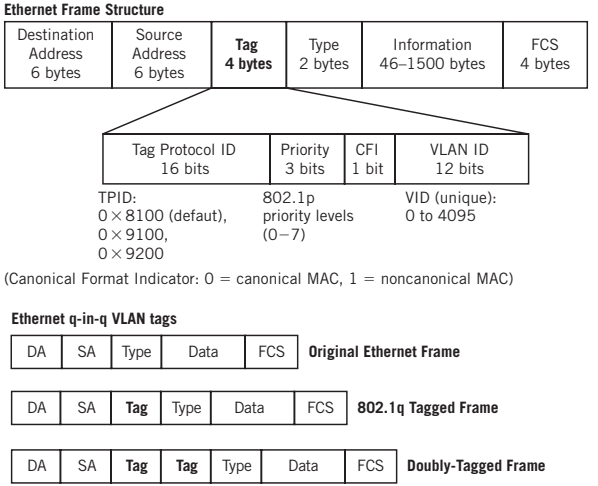
a. Configuration utilisant un seul arbre couvrant : pour éviter la création d'une boucle entre SW1 et SW2, on ne peut pas se servir d'un des deux liens pour écouler le trafic normal des VLAN.
Ici, SW2 a ses deux ports bloqués : le second lien sert uniquement de secours en cas de panne du premier.
b. Configuration utilisant deux arbres couvrants : dans ce cas, un des trunks transporte les données d'un VLAN, tandis que l'autre véhicule les données de l'autre VLAN.
En cas de panne d'un trunk, le lien survivant peut transporter les données des deux VLAN.

Création de «commutateur-routeur»

Il combine les fonctionnalités d'un commutateur avec un routeur :

- routage interVLAN en fonction de l'adressage IP ;
- routage dynamique avec des protocoles de routage comme RIP, OSPF et BGP ;
- protocole VRRP, «Virtual Router Redundancy Protocol», RFC 2338 : permet d'introduire de la redondance de routeur pour le choix du routeur par défaut ;
- du contrôle d'accès, «Access Control List» : bloquer l'accès d'un sous-réseau à un autre sous-réseau.

La norme 802.1Q



Elle ajoute :

– 4 octets au format de la trame, entre l'@MAC source et le champs type contenu dans la trame ;
La taille maximale de la trame passe de 1518 octets à 1522 octets.
– un identifiant de VLAN qui peut aller de 0 à 4095.

Il est possible d'utiliser un **double étiquetage**, appelé «q in q» pour permettre à un FAI d'avoir ses propres VLANs an plus de ceux du client (dans le cas ou les sites du client sont dispersés et doivent communiquer entre eux par l'intermédiaire du FAI).

La norme 802.1p intègre des notions de priorités pour favoriser le trafic d'un VLAN et faire de la QoS.

Description :

- le TPID, «*Tag Protocol Identifier*» : prend la place du type dans une trame 802.3 et identifie la trame comme étant une trame de VLAN (valeur `0x8100`) ;
- PCP, «*Priority Code Point*» : champ sur 3 bits allant de la priorité 1 la plus faible à 7 la plus forte (la valeur 0 indique pas de priorité). Ces priorités peuvent être associées à des *classes* de trafic : voix, vidéo, données… ;
- CFI, «*Canonical Format Indicator*» : compatibilité avec les Token Ring : une trame dont le CFI est à 1, c-à-d non canonical, alors la trame ne doit pas être relayée vers un port non *tagé*, non associé à un VLAN ;
- VID, «*VLAN Identifier*» : identifie le VLAN, le numéro 1 est associé à un VLAN de gestion administrative, et la valeur 0 indique que la trame n'appartient à aucun VLAN.

Les VLANs «Port Based»

Un port du switch est affecté à un seul VLAN :

- tout le trafic entrant, «*ingress*», est transmis uniquement dans le VLAN associé au port :
  - ◇ pour du trafic en broadcast, il sera diffusé uniquement vers tous les ports associés au même VLAN ;
  - ◇ pour du trafic unicast/multicast, il sera transmis vers le ou les ports destinations du même VLAN ;
- la **sécurité** est maximale : il n'est pas possible d'accéder à un autre VLAN que celui associé.

Les VLANs «802.1Q»

- ▷ le switch gère les étiquettes, «*tag*», de VLANs qui intégre un identifiant de VLAN, «VID» ;
- ▷ un port :
  - ◇ est associé à un VLAN par défaut (appelé PVID, «*Port VLAN IDentifier*») ;
  - ◇ appartient à un groupe de VLANs, exprimé sous forme d'une liste de VIDs ;

Le travail du switch est le suivant :

- □ Pour le *trafic en entrée*, «*ingress*», du port :
  - ◇ si la trame **ne possède pas** d'étiquette, «*untagged*» : elle rejoint le VLAN par défaut associé au port ;
  - ◇ si la trame **est étiquettée**,«*tagged*», on récupère le VID indiqué dedans :
    - ⋆ si ce VID **correspond** à celui d'un des VLANs du groupe auquel appartient le port : la trame rejoint le VLAN dont le numéro est indiqué dans l'étiquette ;
    - ⋆ si ce VID **ne correspond pas** à celui d'un des VLANs du groupe auquel appartient le port : la trame est détruite.
- □ Pour le *trafic en sortie*, «*egress*», du port :
  - ◇ «*tagged*» : la trame en sortie est étiquetté avec le numéro du VLAN associé au port ;
  - ◇ «*untagged*» : le trame en sortie ne possède pas d'étiquette VLAN.

- on ajoute et active une nouvelle interface associée à une interface déjà existante :

xterm

```
pef@olympus:~$ sudo ip link add link eth0 name vlan1 type vlan id 200
pef@olympus:~$ ip link list dev vlan1
3: vlan1@eth0: <BROADCAST,MULTICAST> mtu 1500 qdisc noop state DOWN
mode DEFAULT group default
    link/ether 00:0c:29:b6:6d:7c brd ff:ff:ff:ff:ff:ff
pef@olympus:~$ sudo ip link set dev vlan1 up
```

- on configure l'adresse de la machine sur le VLAN 200 (différente de celle de `eth0`) :

xterm

```
pef@olympus:~$ sudo ip address add dev vlan1 10.1.1.1/24
pef@olympus:~$ ip address list dev vlan1
3: vlan1@eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc
noqueue state UP group default
    link/ether 00:0c:29:b6:6d:7c brd ff:ff:ff:ff:ff:ff
    inet 10.1.1.1/24 scope global vlan1
       valid_lft forever preferred_lft forever
    inet6 fe80::20c:29ff:feb6:6d7c/64 scope link
       valid_lft forever preferred_lft forever
```


- on envoie un paquet ICMP «*echo request*» sur l'interface du VLAN :

xterm
```
pef@olympus:~$ ping -c 1 -I vlan1 10.1.1.2
```

- on «*sniffe*» le trafic transmis sur l'interface `vlan1` :

xterm
```
pef@olympus:~$ sudo tcpdump -nvveX -i vlan1
tcpdump: listening on vlan1, link-type EN10MB (Ethernet), capture size
65535 bytes

21:16:34.705960 00:0c:29:b6:6d:7c > ff:ff:ff:ff:ff:ff, ethertype ARP
(0x0806), length 42: Ethernet (len 6), IPv4 (len 4),
Request who-has 10.1.1.2 tell 10.1.1.1, length 28
   0x0000:  0001 0800 0604 0001 000c 29b6 6d7c 0a01  ..........).m|..
   0x0010:  0101 0000 0000 0000 0a01 0102            ............
```

- et sur l'interface `eth0`, avec encapsulation 802.1Q :

xterm
```
pef@olympus:~$ sudo tcpdump -nvveX -i eth0 not tcp
tcpdump: listening on eth0, link-type EN10MB (Ethernet), capture size
65535 bytes

21:23:01.625402 00:0c:29:b6:6d:7c > ff:ff:ff:ff:ff:ff, ethertype
802.1Q (0x8100), length 46: vlan 200, p 0, ethertype ARP,
Ethernet (len 6), IPv4 (len 4), Request who-has 10.1.1.2 tell
10.1.1.1, length 28
   0x0000:  00c8 0806 0001 0800 0604 0001 000c 29b6  ..............).
   0x0010:  6d7c 0a01 0101 0000 0000 0000 0a01 0102  m|..............
```

Adressage de groupe et switches/routeurs

Lors de la création du groupe

Exemple avec la commande «`socat`» :

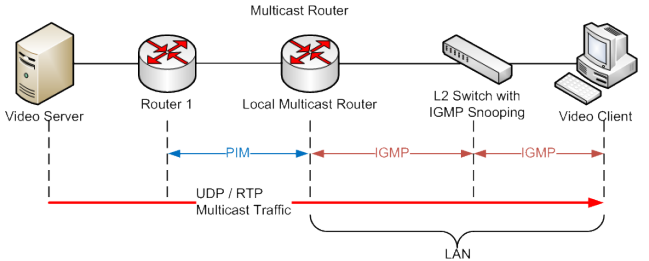
xterm
```
pef@cerberus:~$ socat stdio udp-recvfrom:7182,ip-add-membership=224.0.0.127:eth0,fork
```

et les paquets IGMP diffusés (`TO_EX` {empty} ⟹ quitter l'exclusion donc joindre) :

xterm
```
root@starfox:~# tcpdump -nvveX -i eth0 igmp
tcpdump: listening on eth0, link-type EN10MB (Ethernet), capture size 65535 bytes
02:44:53.375237 00:0c:29:0f:31:a1 > 01:00:5e:00:00:16, ethertype IPv4 (0x0800), length 60: (tos 0xc0, ttl 1, id 0,
  offset 0, flags [DF], proto IGMP (2), length 40, options (RA))
    192.168.127.238 > 224.0.0.22: igmp v3 report, 1 group record(s) [gaddr 224.0.0.127 to_ex { }]
   0x0000:  46c0 0028 0000 4000 0102 c362 c0a8 7fee  F..(..@....b....
   0x0010:  e000 0016 9404 0000 2200 f97e 0000 0001  ........"..~....
   0x0020:  0400 0000 e000 007f 0000 0000 0000       ..............
```

Lors de la terminaison du groupe

xterm
```
02:45:18.986933 00:0c:29:0f:31:a1 > 01:00:5e:00:00:16, ethertype IPv4 (0x0800), length 60: (tos 0xc0, ttl 1, id 0,
  offset 0, flags [DF], proto IGMP (2), length 40, options (RA))
    192.168.127.238 > 224.0.0.22: igmp v3 report, 1 group record(s) [gaddr 224.0.0.127 to_in { }]
   0x0000:  46c0 0028 0000 4000 0102 c362 c0a8 7fee  F..(..@....b....
   0x0010:  e000 0016 9404 0000 2200 fa7e 0000 0001  ........"..~....
   0x0020:  0300 0000 e000 007f 0000 0000 0000       ..............
```



Le «switch» écoute les ports qui diffusent ces messages IGMP pour limiter le multicast des paquets, il réalise du «*IGMP snooping*».

Les messages IGMP sont diffusés vers le groupe `224.0.0.22`.

Exemple d'utilisation : diffusion d'un flux RTP, «*Real Time Protocol*» pour la diffusion d'un flux vidéo au travers d'un routeur.

Solution de DNS pour réseau local

C'est un protocole proposé par Apple (à la base utilisé par Bonjour), présent sur GNU/Linux (Avahi) :

- basé sur les mêmes formats de paquet que le DNS traditionnel :
  - chaque ordinateur possède sa propre liste d'enregistrements DNS (A, MX, SRV, etc) ;
  - chaque ordinateur peut diffuser la liste des services qu'il offre grâce au champ SRV ;
  - domaine local appelé «`.local`» ;
- échangé de manière différentes dans le réseau :
  - utilisation de l'adresse multicast `224.0.0.251` pour contacter toutes les machines du réseau local auquel on est connecté ;
  - utilise le port `5353` en `UDP` ;
  - les paquets ont les caractéristiques suivantes :

```
MAC address 01:00:5E:00:00:FB
IPv4 address 224.0.0.251 or IPv6 address FF02::FB
UDP port 5353
```

  - se combine avec l'auto-configuration des adresses IP, APIPA, «*Automatic Private IP Addressing*», (`169.254.0.0/16`), en l'absence de serveur DHCP ou de configuration manuelle ;

Il existe également UPnP SSDP, «Simple Service Discovery Protocol» de Microsoft qui n'utilise pas le format de requête/réponse du DNS.

Exemple de paquet mDNS obtenu avec Scapy

xterm

```
>>> l=sniff(count=3,filter="udp and port 5353")
>>> DNS(str(l[0][UDP].payload))
<DNS  id=0 qr=0L opcode=QUERY aa=0L tc=0L rd=0L ra=0L z=0L rcode=ok qdcount=1 ancount=0 nscount=1
arcount=0
qd=<DNSQR  qname='iPhone de Pierre-Francois Bonnefoi._avexvscreen._udp.local.' qtype=ALL
qclass=32769 |> an=None ns=<DNSRR
rrname='iPhone de Pierre-Francois Bonnefoi._avexvscreen._udp.local.' type=SRV rclass=IN ttl=120
rdata='\x00\x00\x00\x00\x17q"iPhone-de-Pierre-Francois-Bonnefoi\xc0B' |> ar=None |>
```

Recherche de machine par service fourni (RFC 2782)

Un nouveau type d'enregistrement a été défini dans les DNS, suivant le format suivant :

```
_Service._Proto.Name TTL Class SRV Priority Weight Port Target
```

Exemple : `_sip._udp.unilim.fr 43200 IN SRV 10 10 5060 sipserveur.unilim.fr`

- le service est SIP ;
- le protocole est UDP ;
- la durée de vie de l'association est de 12h (43200 secondes) ;
- la classe est Internet ;
- le type d'enregistrement est SRV ;
- la priorité est de 10 (les valeurs inférieures sont utilisées préférentiellement) ;
- le poids est de 10 (pour pouvoir faire de l'équilibrage de charge) ;
- le port associé au SIP est 5060 ;
- le FQDN, «*Full Qualified Domain Name*» du serveur est `sipserveur.unilim.fr`.

Exemple de paquet mDNS sniffé avec Scapy

xterm

```
>>> paquet_sniffe.payload
<IP  version=4L ihl=5L tos=0x0 len=160 id=18153 flags= frag=0L ttl=255 proto=udp chksum=0xa889
src=192.168.42.54 dst=224.0.0.251[[\vcrlf]] options='' |<UDP  sport=mdns dport=mdns len=140
chksum=0x9788 |<Raw  load='\x00\x00\x00\x00\x00\x01\x00\x00\x00\x01\x00\x00[[\vcrlf]]#iPhone de
Pierre-Fran\xc3\xa7ois Bonnefoi\x0c_avexvscreen\x04_udp\x05lo
cal\x00\x00\xff\x80\x01\xc0\x0c\x00![[\crlf]]\x00\x01\x00\x00\x00x\x00\x00\x00\x00\x00\x17q"iPhone-
de-Pierre-Francois-Bonnefoi\xc0B'
|>>>
>>> DNS(str(paquet_sniffe[UDP].payload))
<DNS  id=0 qr=0L opcode=QUERY aa=0L tc=0L rd=0L ra=0L z=0L rcode=ok qdcount=1 ancount=0 nscount=1
arcount=0 qd=<DNSQR [[\vcrlf]] qname='iPhone de Pierre-Fran\xc3\xa7ois
Bonnefoi._avexvscreen._udp.local.' qtype=ALL qclass=32769 |> an=None [[\vcrlf]]ns=<DNSRR
rrname='iPhone de Pierre-Fran\xc3\xa7ois [[\crlf]]Bonnefoi._avexvscreen._udp.local.' type=SRV
rclass=IN ttl=120 rdata='\x00\x00\x00\x00\x17q[[ \vcrlf]]"iPhone-de-Pierre-Francois-Bonnefoi\xc0B'
|> ar=None |>
```

DNS & Administration :
reverse, équilibrage et sécurité

Exemple : une requête DNS vers «google.com»

xterm

```
bonnefoi@msi:~$ dig +trace @164.81.1.5 www.google.com
; <<>> DiG 9.7.3 <<>> +trace @164.81.1.5 www.google.com
; (1 server found)
;; global options: +cmd
.        344794   IN NS j.root-servers.net.
.        344794   IN NS g.root-servers.net.
.        344794   IN NS l.root-servers.net.
.        344794   IN NS k.root-servers.net.
.        344794   IN NS c.root-servers.net.
.        344794   IN NS f.root-servers.net.
.        344794   IN NS e.root-servers.net.
.        344794   IN NS m.root-servers.net.
.        344794   IN NS b.root-servers.net.
.        344794   IN NS d.root-servers.net.
.        344794   IN NS h.root-servers.net.
.        344794   IN NS a.root-servers.net.
.        344794   IN NS i.root-servers.net.
;; Received 272 bytes from 164.81.1.5#53(164.81.1.5) in 1 ms
com.        172800   IN NS a.gtld-servers.net.
com.        172800   IN NS b.gtld-servers.net.
com.        172800   IN NS c.gtld-servers.net.
com.        172800   IN NS d.gtld-servers.net.
com.        172800   IN NS e.gtld-servers.net.
com.        172800   IN NS f.gtld-servers.net.
com.        172800   IN NS g.gtld-servers.net.
com.        172800   IN NS h.gtld-servers.net.
com.        172800   IN NS i.gtld-servers.net.
com.        172800   IN NS j.gtld-servers.net.
com.        172800   IN NS k.gtld-servers.net.
com.        172800   IN NS l.gtld-servers.net.
com.        172800   IN NS m.gtld-servers.net.
;; Received 492 bytes from 193.0.14.129#53(k.root-servers.net) in 31 ms
google.com.    172800   IN NS ns2.google.com.
google.com.    172800   IN NS ns1.google.com.
google.com.    172800   IN NS ns3.google.com.
google.com.    172800   IN NS ns4.google.com.
;; Received 168 bytes from 192.12.94.30#53(e.gtld-servers.net) in 34 ms
www.google.com.      604800   IN CNAME www.l.google.com.
www.l.google.com. 300   IN A  209.85.227.147
www.l.google.com. 300   IN A  209.85.227.106
www.l.google.com. 300   IN A  209.85.227.103
www.l.google.com. 300   IN A  209.85.227.99
www.l.google.com. 300   IN A  209.85.227.105
www.l.google.com. 300   IN A  209.85.227.104
;; Received 148 bytes from 216.239.32.10#53(ns1.google.com) in 26 ms
```

- la configuration de la machine `msi.unilim.fr`:

xterm

```
bonnefoi@msi:~$ more /etc/resolv.conf
search msi.unilim.fr unilim.fr
nameserver 164.81.60.1
nameserver 164.81.1.4
nameserver 164.81.1.5
```

- la requête fait appel au serveur DNS `164.81.1.5` connu de la machine `msi.unilim.fr`, mais demandé **explicitement** dans l'utilisation de l'outil `dig`.
- le serveur, *resolver*, `164.81.1.5` réalise une requête à différents «root servers»
- il obtient une réponse de la part du serveur racine «k.root-servers.net» qui lui donne la liste des serveurs «GTLD», «Generic Top Level Domain», chargés du domaine «`.com`» ;
- le serveur «`e.gtld-servers.net`» lui renvoi la liste des serveurs DNS en charge du domaine `google.com`;
- le serveur DNS «`ns1.google.com`» renvoi une liste d'adresse correspondant à la machine `google.com`.

Les serveurs racines présents dans le système d'exploitation

```
; Cache file:
. IN NS A.ROOT-SERVERS.NET.
A.ROOT-SERVERS.NET.   IN  A   198.41.0.4
. IN NS B.ROOT-SERVERS.NET.
B.ROOT-SERVERS.NET.   IN  A   128.9.0.107
. IN NS C.ROOT-SERVERS.NET.
C.ROOT-SERVERS.NET.   IN  A   192.33.4.12
. IN NS D.ROOT-SERVERS.NET.
D.ROOT-SERVERS.NET.   IN  A   128.8.10.90
. IN NS E.ROOT-SERVERS.NET.
```

Resolution inverse

Elle consiste a obtenir le nom de domaine à partir de l'adresse IP :
- pour faciliter la compréhension des humains
- pour des raisons de sécurité

Elle est plus délicate que de nom vers IP car le système DNS est organisé pour la résolution de nom
⟶ *recherche exhaustive ???*

Solution : utiliser les adresses comme des noms :
- le domaine «`in-addr.arpa`»
- les noms des noeuds correspondent aux octets de l'adresse IP en ordre inverse
- le domaine `in-addr.arpa` a 256 sous-domaines,
- chacun de ces sous-domaines a 256 sous-domaines,
- chacun de ces sous-domaines a, *à son tour*, 256 sous-domaines,
- le 4ème niveau correspond à un NS connaissant le nom de domaine associé à cette adresse IP.

Le nom de domaine associé à la résolution inverse est noté selon l'adresse IP inversée car la résolution d'un nom de domaine se fait de droite à gauche :

Exemple : `43.60.81.164.in-addr.arpa`

Résolution :

```
in-addr.arpa  A.ROOT-SERVER.NET
164.in-addr.arpa  NS.RIPE.NET
.in-addr.arpa  NS.RIPE.NET
60.81.164.in-addr.arpa  msi.unilim.fr
```

À l'aide de la commande `dig` et de l'option `-x` :

xterm

```
darkstar:~ pef$ dig +short @164.81.1.4 -x 164.81.60.163
portable-pf.msi.unilim.fr.
darkstar:~ pef$ dig +short -x 208.67.222.222
resolver1.opendns.com.
```

Les données d'un serveur DNS sont enregistrées dans une base identifiée par les noms de domaine correspondants (chaque enregistrement est appelé RR, «Resource Records»).

Types d'enregistrements :

- `SOA` : décrit l'autorité administrative (Start of Authority)
- `NS` : liste de serveurs de nom pour ce domaine
- `A` : correspondance nom -> adresse
- `PTR` : correspondance adresse -> nom
- `CNAME` : alias ou surnom
- `TXT` : texte
- `HINFO` : description machine, plus utilisé pour des questions de sécurité
- `RR` = { classe, type, clé, valeur } où classe est IN pour INTERNET

Exemple :

```
1   SOA = Start of Authority Spécifie que ce serveur de nom a autorité sur le domaine
2   ; Database file msi.unilim.fr zone.
3   unilim.fr                IN    SOA msi.unilim.fr
4   (     2006110204 ;              serial number (ex : AAAAMMJJnn)
5   21600 ;                  refresh (ex : 6h période de mise à jour des secondaires)
6   3600 ;             retry (ex: 1h durée minimale avant procaine interrogation)
7   86400 ;             expire (ex: 1 jour)
8   3600 ) ;           minimum TTL des entrées dans le cache
9                     NS   dns.msi.unilim.fr
10                   TXT   « Departement Informatique »
11  ishtar              IN    A       164.81.60.43
12  msi                 IN   A      164.81.60.6
13  www                 IN   CNAME    msi ; des alias
14  164.81.60.6.in-addr      IN   PTR      msi.unilim.fr
```

Le serveur de courrier

MX = Mail eXchanger Permet l'adressage email sur la base du nom de domaine plutôt que sur l'adresse du (des) serveur(s) de mail :

- `bonnefoi@unilim.fr` plutot que `bonnefoi@msi.unilim.fr` ;
- permet à l'émetteur d'ignorer quelle est la machine serveur de mail ;
- permet le deplacement du gestionnaire de mail vers une autre machine ;
- permet la gestion de plusieurs serveurs de mail avec priorité dans l'ordre de consultation des serveurs

L'enregistrement MX est utilisés par les MTA, «*Mail Transfer Agent*», en tenant compte des priorités :

Exemple pour l'Université de Limoges :

xterm
```
bonnefoi@msi:~$ dig +short mx unilim.fr
50 mail.unilim.fr.
```

le serveur d'envoi de courrier de l'Université

Exemple pour Google :

xterm
```
bonnefoi@msi:~$ dig +short mx google.com
30 alt2.aspmx.l.google.com.
10 aspmx.l.google.com.
20 alt1.aspmx.l.google.com.
50 alt4.aspmx.l.google.com.
40 alt3.aspmx.l.google.com.
```

Le MTA utilise le protocole **SMTP**, «*Simple Mail Transfer Protocol*», en tant que client.

Une requête plus concise

xterm

```
bonnefoi@msi:~$ dig +noall +answer web.unilim.fr
web.unilim.fr.      83023 IN A 164.81.1.61
```

Pour IPv6 avec des champs adresses 4 fois plus grand (AAAA)

xterm

```
bonnefoi@msi:~$ dig +short AAAA www.renater.fr
2001:660:3001:4002::10
```

Interrogation du SOA

xterm

```
bonnefoi@msi:~$ dig soa unilim.fr
; <<>> DiG 9.7.3 <<>> soa unilim.fr
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 36568
;; flags: qr rd ra; QUERY: 1, ANSWER: 1, AUTHORITY: 3, ADDITIONAL: 6
;; QUESTION SECTION:
;unilim.fr.         IN   SOA
;; ANSWER SECTION:
unilim.fr.      86400   IN   SOA   limdns.unilim.fr. postmaster.unilim.fr. 2011091501 28800 7200 3600000 86400
;; AUTHORITY SECTION:
unilim.fr.      78813   IN   NS   cnudns.cines.fr.
unilim.fr.      78813   IN   NS   limdns2.unilim.fr.
unilim.fr.      78813   IN   NS   limdns.unilim.fr.
;; ADDITIONAL SECTION:
cnudns.cines.fr.   2583   IN   A   193.48.169.40
cnudns.cines.fr.   2583   IN   AAAA   2001:660:6301:301::2:1
limdns.unilim.fr.   73317   IN   A   164.81.1.4
limdns.unilim.fr.   85383   IN   AAAA   2001:660:6201:1::4
limdns2.unilim.fr.   73317   IN   A   164.81.1.5
limdns2.unilim.fr.   85383   IN   AAAA   2001:660:6201:1::5

;; Query time: 18 msec
;; SERVER: 164.81.60.1#53(164.81.60.1)
;; WHEN: Thu Sep 15 13:25:13 2011
;; MSG SIZE  rcvd: 276
```

Le TTL qui décroit en direct et «Round Robin»

On fait plusieurs requêtes en suivant, et on constate que la durée de vie de la validité de la réponse diminue :

xterm

```
bonnefoi@msi:~$ dig +noall +answer gmail.com
gmail.com.      300   IN   A   74.125.230.86
gmail.com.      300   IN   A   74.125.230.87
gmail.com.      300   IN   A   74.125.230.88
gmail.com.      300   IN   A   74.125.230.85
bonnefoi@msi:~$ dig +noall +answer gmail.com
gmail.com.      298   IN   A   74.125.230.85
gmail.com.      298   IN   A   74.125.230.86
gmail.com.      298   IN   A   74.125.230.87
gmail.com.      298   IN   A   74.125.230.88
bonnefoi@msi:~$ dig +noall +answer gmail.com
gmail.com.      271   IN   A   74.125.230.87
gmail.com.      271   IN   A   74.125.230.88
gmail.com.      271   IN   A   74.125.230.85
gmail.com.      271   IN   A   74.125.230.86
bonnefoi@msi:~$ dig +noall +answer gmail.com
gmail.com.      30    IN   A   74.125.230.87
gmail.com.      30    IN   A   74.125.230.88
gmail.com.      30    IN   A   74.125.230.85
gmail.com.      30    IN   A   74.125.230.86
```

xterm

```
bonnefoi@msi:~$ dig +noall +answer gmail.com
gmail.com.      25   IN   A   74.125.230.86
gmail.com.      25   IN   A   74.125.230.87
gmail.com.      25   IN   A   74.125.230.88
gmail.com.      25   IN   A   74.125.230.85
bonnefoi@msi:~$ dig +noall +answer gmail.com
gmail.com.      3   IN   A   74.125.230.86
gmail.com.      3   IN   A   74.125.230.87
gmail.com.      3   IN   A   74.125.230.88
gmail.com.      3   IN   A   74.125.230.85
bonnefoi@msi:~$ dig +noall +answer gmail.com
gmail.com.      2   IN   A   74.125.230.85
gmail.com.      2   IN   A   74.125.230.86
gmail.com.      2   IN   A   74.125.230.87
gmail.com.      2   IN   A   74.125.230.88
bonnefoi@msi:~$ dig +noall +answer gmail.com
gmail.com.      1   IN   A   74.125.230.88
gmail.com.      1   IN   A   74.125.230.85
gmail.com.      1   IN   A   74.125.230.86
gmail.com.      1   IN   A   74.125.230.87
```

On remarque que la réponse du serveur DNS est :

- ⋆ multiple ;
- ⋆ l'ordre de ces réponses change avec le temps

Il correspond à **répartir un trafic** (HTTP par exemple) afin d'**équilibrer la charge** entre différents serveurs.

On «fait tourner» différentes réponses (algo. de «*Round-Robin*»).

Échanger «*d'autres*» informations au travers du protocole DNS

On utilise le champ «`TXT`» du DNS et un serveur «*complice*» :

xterm

```
pef@starfox: $ dig 4.1.81.164.origin.asn.cymru.com TXT +short
"1935 | 164.81.0.0/16 | FR | ripencc |"
pef@ns505411:/home/pef$ dig AS1935.asn.cymru.com TXT +short
"1935 | EU | ripencc |  | FR-RENATER-LIMOUSIN Reseau Regional Limousin, FR"
```

DOH, «*DNS Queries over HTTPS*», RFC 8484

Protéger les requêtes DNS de l'utilisateur d'un observateur extérieur en utilisant un échange basé sur **HTTPS** pour les réaliser :

- nécessite l'emploi de **serveurs supportant** ce protocole : non accepté par tous les serveurs DNS ;
- établie une connexion HTTPS basée sur TLS ou HTTP/2 (pour accélérer les requêtes suivantes) ;
- utilise les méthodes GET ou POST, comme par exemple, celle de Google :
  - `https://dns.google/dns-query` – RFC 8484 (GET and POST)
  - `https://dns.google/resolve?` – JSON API (GET)
- accepte des **requêtes** au format :
  - texte, avec une URL contenant l'entrée DNS demandée ;
  - binaire, reprenant le format de la requête DNS originale en l'encodant en «*URL safe base64*» ;
- retourne des **réponses** au format :
  - texte, basée sur JSON ;
  - binaire, correspondant au format original du DNS, appelé aussi «*wireformat*» ;

xterm

```
$ socat - openssl:dns.google.com:443,verify=0
GET /resolve?name=www.unilim.fr&type=A HTTP/1.0
Host: dns.google.com

HTTP/1.0 200 OK
Strict-Transport-Security: max-age=31536000; includeSubDomains; preload
Access-Control-Allow-Origin: *
Date: Mon, 16 Sep 2019 08:25:11 GMT
Expires: Mon, 16 Sep 2019 08:25:11 GMT
Cache-Control: private, max-age=131
Content-Type: application/x-javascript; charset=UTF-8
Server: HTTP server (unknown)
X-XSS-Protection: 0
X-Frame-Options: SAMEORIGIN
Alt-Svc: quic=":443"; ma=2592000; v="46,43,39"
Accept-Ranges: none
Vary: Accept-Encoding

{"Status": 0,"TC": false,"RD": true,"RA": true,"AD": false,"CD": false,"Question":[ {"name":
"www.unilim.fr.","type": 1}],"Answer":[ {"name": "www.unilim.fr.","type": 5,"TTL": 131,"data":
"web-proxy-2.unilim.fr."},{"name": "web-proxy-2.unilim.fr.","type": 1,"TTL": 131,"data":
"164.81.1.97"}]}
```

type de requête DNS : A, AAAA, TXT, etc.

ICMP : le protocole de maintenance

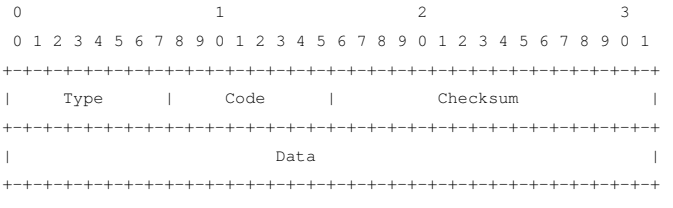
- effectue un contrôle des échanges ;
- permet de «déboguer» le réseau IP à l'aide de message ;
- utilise différents types de message avec des significations différentes en deux catégories :
  - messages d'erreurs ;
  - messages de requêtes ;

Attention

Pour des questions de sécurité certains messages ICMP «actifs» ne sont plus interprétés et d'autres de «surveillance» sont filtrés.

Message Returned	Description / Interpretation
Destination Unreachable	This tells the source host that there is a problem delivering a packet. The problem is that either the destination host is down or its internet connection is down.
Time Exceeded	It has taken too long for a packet to be delivered. The packet has been discarded.
Source Quench	The source is sending data faster than it can be forwarded. This message requests that the sender slow down.
Redirect	The router sending this message has received some packet for which another router, which is also directly connected to the sender, would have had a better route. The message tells the sender to use the better router.
Echo	This is used by the `ping` command to verify connectivity. The sender will issue an "echo request" message and will receive an "echo reply" from the other host if a path is found between the two.
Parameter Problem	This is used to identify a parameter that is incorrect.
Timestamp	This is used to measure roundtrip time to particular hosts.
Address Mask Request/Reply	This is used to inquire about and learn the correct subnet mask to be used.
Router Advertisement and Selection	This is used to allow hosts to dynamically learn the IP addresses of the routers attached to the subnet.

Le format du message ICMP :
le contenu des données dépend du type.



ICMP : les messages et leurs usages

Type	Meaning	Codes	Data	Router Sends	Router Receives	Host Sends	Host Receives
0	Echo reply	0	Varies	M	M	M	M
8	Echo request	0	Varies	M	M	M	M
13	Timestamp request	0	12 bytes	Opt	Opt	Opt	Opt
14	Timestamp reply	0	12 bytes	Opt	Opt	Opt	Opt
15	Information request	0	0 bytes	Obs	Obs	Obs	Obs
16	Information reply	0	0 bytes	Obs	Obs	Obs	Obs
17	Mask request	0	4 bytes	M	M	Opt	Opt
18	Mask reply	0	4 bytes	M	M	Opt	Opt
37	Domain name request	0	0 bytes	M	M	M	M
38	Domain name reply	0	0 bytes	M	M	M	M

Obs, obsolete; Opt, optional; M, mandatory.

Type	Meaning	Codes	Data	Router Sends	Router Receives	Host Sends	Host Receives
1	Unassigned	NA	NA	NA	NA	NA	NA
2	Unassigned	NA	NA	NA	NA	NA	NA
6	Alternate host address	0	(4 bytes)	(Prohibited)	(Prohibited)	Opt	Opt
9	Router advertisement	0	Varies	M	Opt	Prohibited	Opt
10	Router solicitation	0	0 bytes	M	M	Opt	Opt
19	Reserved–security	NA	NA	NA	NA	NA	NA
20–29	Reserved–robustness	NA	NA	NA	NA	NA	NA
30	Traceroute	0–1	Varies	Opt	Opt	M	M
31	Datagram conversion error	0–11	Varies	?	?	?	?
32	Mobile host redirect	0	Varies	Opt	Opt	Opt	Opt
33	IPv6 where-are-you	0	?	Opt	Opt	Opt	Opt
34	IPv6 I-am-here	0	?	Opt	Opt	Opt	Opt
35	Mobile registration request	0, 16	Varies	Opt	Opt	Opt	Opt
36	Mobile registration reply	0, 16	Varies	Opt	Opt	Opt	Opt
39	SKIP	0	Varies	Opt	Opt	Opt	Opt
40	Photurius	0–3	Varies	Exp	Exp	Exp	Exp

Exp, expired; Obs, obsolete; Opt, optional; M, mandatory; NA, not applicable.

- M, *Mandatory* : obligatoire ;
- *actuellement* : les machines filtrent les paquets «*echo request*» pour être «invisibles» sur le réseau (mode «*stealth*» sous Mac os X par ex.) ;
- le protocole «SKIP», «*Simple Key-Management for Internet Protocol*» pour l'utilisation des protocoles AH, «*Authentication Header*» et ESP, «*Encapsulating Security Protocol*» et choix des protocoles cryptographiques ;
- le protocole «photuris» sert également à la cryptographie : «*Session-Key Management Protocol*».

Pour la reconnaissance d'un réseau

- **comprendre l'organisation** de l'environnement de la cible ;
- **obtenir des informations** sur la cible pour préparer une attaque ;
- utiliser les **bonnes techniques et outils** pour les différentes phases de l'attaque ;

Les techniques

- «*ICMP Sweep*» : «balayer» la plage d'adresses d'un réseau (réalisable automatiquement par l'outil «`nmap`» par ex.) ;
- «*Traceroute*» : envoi successif de datagrammes avec une valeur de TTL incrémentée à chaque datagramme :
  - lorsqu'un routeur «*forward*» un datagramme, il décrémente le TTL ;
  - lorsque la TTL arrive à zéro, un paque ICMP «*time excedeed*» est retourné à l'expéditeur, ce qui permet à celui-ci de découvrir l'adresse du routeur ;
  - on obtient des informations sur le chemin des datagrammes et sur la topologie du réseau ;
- «*Firewalking*» : améliorer le «traceroute» pour identifier les ports ouverts sur un **firewall** filtrant les communications et obtenir les règles de configuration du firewall :
  - première phase : réaliser un traceroute «classique» pour déterminer le nombre n de sauts, «*hops*», jusqu'au firewall ;
  - deuxième phase : envoyer des datagrammes avec une TTL égale à $n + 1$ et associé à un protocole choisi :
    - si un paquet ICMP «*time excedeed*» est reçu, alors le datagramme a réussi à traverser le firewall ;
    - si rien n'est reçu, alors on en déduit qu'une règle du firewall a filtré le datagramme.
- «*Inverse Mapping*» : cette technique permet de «mapper» le réseau interne qui est protégé par un firewall :
  - l'attaquant envoie des paquets ICMP «*echo reply*» vers une plage d'adresses présumées derrière le firewall ;
  - à l'arrivée de ces paquets ICMP de réponse, le firewall autorise à passer car **il ne maintient pas** la liste des requêtes ICMP (sauf si le firewall est de bon niveau ainsi que l'administrateur) ;
  - s'il existe un routeur interne, ce routeur retournera un paquet ICMP «*Host unreachable*» pour chaque hôte indisponible du réseau, ce qui informe, par déduction, l'attaquant sur les machines présentes dans le réseau.

- «*L'OS fingerprinting*» : reconnaissance de l'OS présent sur la machine cible :
  - l'attaquant envoi un datagramme UDP avec le bit «DF» à 1 vers la cible avec un numéro de port fermé ;
  - la cible retourne un paquet ICMP «*Destination port Unreachable*» à l'attaquant ;
  - en analysant le format du paquet ICMP reçu, l'attaquant peut obtenir des informations sur la cible ;

    Exemple : un paquet est envoyé depuis un Linux vers un Mac et un PC et la réponse est différente :

xterm

```
>>> paquet_mac
<IP  frag=0 proto=udp dst=192.168.127.1 |<UDP  dport=56467 |>>

>>> reponse_mac
<IP  version=4L ihl=5L tos=0x0 len=56 id=9103 flags= frag=0L ttl=64 proto=icmp chksum=0xd747
src=192.168.127.1\
dst=192.168.127.156 options=[]
|<ICMP  type=dest-unreach code=port-unreachable chksum=0x202c unused=0
|<IPerror  version=4L ihl=5L tos=0x0 len=28 id=1 flags= frag=0L ttl=64 proto=udp
chksum=0xfae1 src=192.168.127.156\
 dst=192.168.127.1 options=[]
|<UDPerror  sport=domain dport=56467 len=8 chksum=0x0 |>>>>



>>> paquet_pc
<IP  frag=0 proto=udp dst=192.168.127.153 |<UDP  dport=56467 |>>

>>> reponse_pc
<IP  version=4L ihl=5L tos=0x0 len=56 id=2471 flags= frag=0L ttl=128 proto=icmp
chksum=0xb097 src=192.168.127.153\
dst=192.168.127.156 options=[]
|<ICMP  type=dest-unreach code=port-unreachable chksum=0x7d9d unused=0
|<IPerror  version=4L ihl=5L tos=0x0 len=28 id=1 flags= frag=0L ttl=64 proto=udp
chksum=0xfa49 src=192.168.127.156\
 dst=192.168.127.153 options=[]
|<UDPerror  sport=domain dport=56467 len=8 chksum=0xa28e |>>>>
```

Jeu des 7 erreurs : quelles sont-elles ?

- «*L'OS fingerprinting*» : **reconnaissance de l'OS** présent sur la machine cible :

  Autre exemple : les paquets envoyés par la commande «ping» depuis un Mac et depuis un PC :

```
xterm
>>> ping_mac
<Ether  dst=00:11:de:ad:be:ef src=00:50:56:c0:00:08 type=IPv4 |<IP  version=4L ihl=5L
tos=0x0 len=84 id=54669\
 flags= frag=0L ttl=64 proto=icmp chksum=0x252d src=192.168.127.1 dst=192.168.127.156
options=[]
|<ICMP  type=echo-request code=0 chksum=0xdbb7 id=0xdd4e seq=0x0
|<Raw
load='Ph\xc7\x8e\x00\x02;\xfd\x08\t\n\x0b\x0c\r\x0e\x0f\x10\x11\x12\x13\x14\x15\x16\x17\x18\x19\x1a\x1b\
\x1c\x1d\x1e\x1f !"#$%&\'()*+,-./01234567' |>>>>
>>> ping_pc
<Ether  dst=00:11:de:ad:be:ef src=00:0c:29:d1:27:40 type=IPv4 |<IP  version=4L ihl=5L
tos=0x0 len=60 id=758\
 flags= frag=0L ttl=128 proto=icmp chksum=0xb740 src=192.168.127.157 dst=192.168.127.156
options=[]
|<ICMP  type=echo-request code=0 chksum=0x4d5a id=0x1 seq=0x1
|<Raw  load='abcdefghijklmnopqrstuvwabcdefghi' |>>>>
```

- «*ICMP route redirect*» : c'est un paquet envoyé par un routeur lorsqu'il :
  - reçoit du trafic d'un hôte ;
  - trouve dans sa table de routage que l'adresse de «prochain saut» à laquelle doit être envoyé ce trafic est dans le même réseau que celui de l'hôte ;
  - il est possible de faire un attaque «*Man-in-the-Middle*» :
    - l'attaquant prend le contrôle du routeur R1 ;
    - l'attaquant envoi un paquet ICMP redirection vers la victime en prenant la place du routeur R2, utilisé par la victime ;
    - la redirection demande à la victime de modifier sa table de routage pour passer par R1 au lieu de R2 ;
    - l'attaquant intercepte le trafic sur R1.
- les paquets ICMP «*oversized*» : un paquet ICMP d'une **taille maximale** envoyé vers la victime fait **planter** sa pile TCP/IP ;
- «*ICMP Router Discovery*» : ce sont des paquets transmis dans le réseau pour permettre à un hôte de **découvrir les routeurs présents** : «*router solicitation*».

  Un attaquant peut répondre par un «*router advertisement*» et prendre la place du routeur par défaut pour la victime.

- «*ICMP floods*» : en **inondant** la victime de message ICMP, on peut la **ralentir** et, éventuellement, l'**empêcher de communiquer** :
  - envoi depuis le même hôte ;
  - «*smurf attack*» : on envoie un paquet ICMP «*echo request*» en diffusion sur le réseau en mettant l'adresse de la victime comme adresse source ;
  - la victime est «submergée» par toutes les paques ICMP «*echo reply*» qu'elle reçoit de toutes les machines.

  Ne fonctionne plus.

- le «*tunneling*» : **encapsuler** le trafic d'un protocole comme TCP ou UDP dans le **contenu des paquets ICMP** échangés entre un hôte compromis (où l'attaquant à pris le contrôle) et une machine située à l'extérieur du réseau :
  - le trafic est difficile à découvrir ;
  - le trafic peut être chiffré afin de le protéger.

  This vulnerability exists because RFC 792, which is IETF's rules governing ICMP packets, allows for an arbitrary data length for any type 0 (echo reply) or 8 (echo message) ICMP packets.

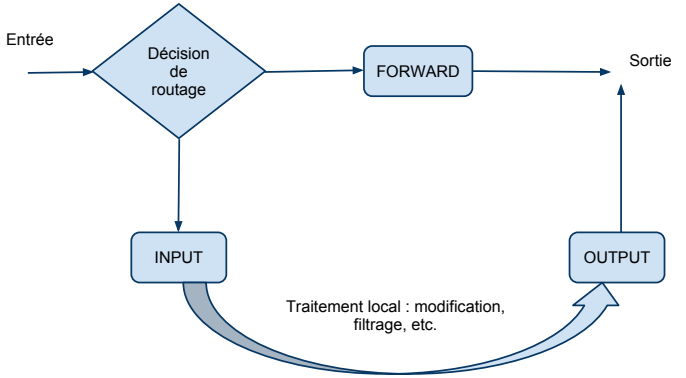
xterm

```
>>> p=IP(dst='192.168.127.1')/ICMP(type='echo-request')/'ceci est un test'
>>> sr(p)
Begin emission:
*Finished to send 1 packets.
Received 1 packets, got 1 answers, remaining 0 packets
(<Results: TCP:0 UDP:0 ICMP:1 Other:0>, <Unanswered: TCP:0 UDP:0 ICMP:0 Other:0>)
>>> _[0][0]
(<IP  frag=0 proto=icmp dst=192.168.127.1
|<ICMP  type=echo-request
|<Raw  load='ceci est un test' |>>>,

<IP  version=4L ihl=5L tos=0x0 len=44 id=29697 flags= frag=0L ttl=64 proto=icmp chksum=0x86e1
src=192.168.127.1 dst=192.168.127.156
options=[] |<ICMP  type=echo-reply code=0 chksum=0x2ee8 id=0x0 seq=0x0 |<Raw  load='ceci est un
test' |<Padding  load='\x00\x00' |>>>>)
```

Fonctionnement d'un Firewall sur un système

Lorsqu'un datagramme arrive sur une interface par l'intermédiaire de la couche de niveau 2, c-à-d ethernet, une décision de routage doit être prise, en «amont» de la couche de niveau 3 du système :

*il est possible **d'analyser le paquet** avant de l'introduire dans la pile TCP/IP locale*



a. si le système **n'assure pas** les fonctions de routeur :
   - ◇ le paquet est à destination du système :
     il est considéré comme en «Entrée», *INPUT*, du système :
     - ⋆ il est filtré et/ou éventuellement modifié par le Firewall ;
     - ⋆ s'il est accepté, il est dirigé vers un processus s'il en existe un capable de le gérer ;
   - ◇ le paquet n'est pas à destination du système : il est ignoré.

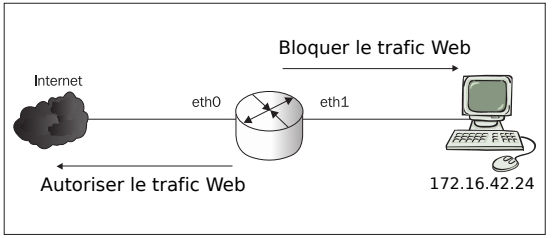
b. si le système **assure** les fonctions de routeur :
   - ◇ le paquet est à destination du système : il est considéré comme en «Entrée», *INPUT*, du système :
     - ⋆ il est filtré et/ou éventuellement modifié par le Firewall ;
     - ⋆ s'il est accepté, il est dirigé vers un processus s'il en existe un capable de le gérer ;
   - ◇ le paquet n'est pas à destination du système :
     - ⋆ choix de «*forwarder*» le paquet vers une interface de sortie en fonction des règles de routage ;
     - ⋆ modifier le datagramme en sortie : intégration de la QoS, traduction d'@IP, *etc.*

La «Sortie», *OUTPUT*, correspond à des datagrammes créés et émis par le système lui-même.

Stateless *vs* Stateful

Exemple de scenario :

- ⋆ *On veut bloquer les communications web entrantes*
- ⋆ *On veut autoriser les communications web sortantes*



Comment faire ?

- ▷ Le «web» indique des communications TCP vers le port 80 ;
- ▷ Qelle différence entre «sortantes» en «entrantes» ?
  - ◇ les paquets «sortants» vont de `172.16.42.24` vers Internet ;
  - ◇ les paquets «entrants» vont d'Internet vers `172.16.42.24` ;

□ **Première proposition** :

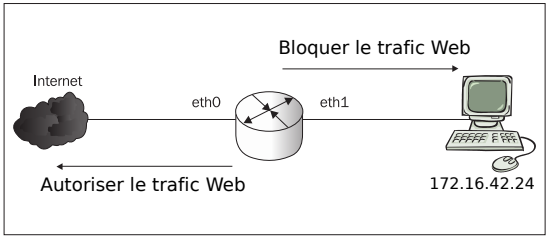
- ◇ on bloque les paquets «entrants» et on autorise ceux «sortants» : *ça ne marche pas !*
- ◇ **Car...** une communication TCP est composée de paquets «entrants» et sortants :
  - ⋆ des segments TCP sortent avec la requête utilisateur `GET / HTTP/1.0`…
  - ⋆ des segments TCP entrent avec la réponse du serveur `HTTP/1.1 200 ok`…
  - ⋆ il y a des ACKs, des PUSH *etc*.

□ **Seconde proposition** :

- ◇ on laisse passer les segments de données dans les deux sens ;
- ◇ on contrôle l'établissement des connexions TCP :
  - ⋆ on autorise seulement les segments contenant un SYN en provenance ou à destination du port 80 :
- ◇ Problème : un vilain hacker choisit le port source 80 pour établir sa connexion depuis Internet vers `172.16.42.24` : le segment contient un SYN et provient du port 80 : *Bingo !*
- ◇ *On est pas arrivé à bloquer le trafic interdit !*

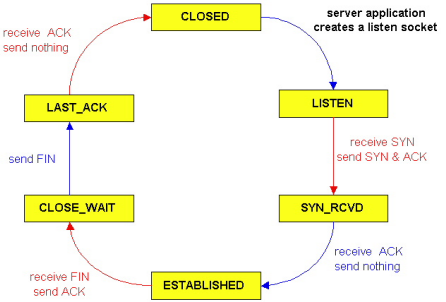
Exemple de scenario (suite) :

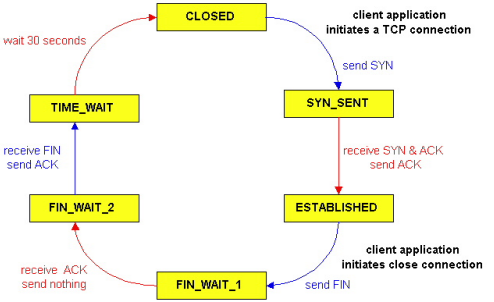
- *On veut bloquer les communications web entrantes*
- *On veut autoriser les communications web sortantes*



- **Troisième proposition** : on utilise le «handshake TCP» et on mémorise un état
  1. on autorise uniquement les paquets SYN depuis `172.16.42.24` vers Internet ;
  2. lorsque l'on voit passer un segment SYN autorisé, on **mémorise** l'établissement de la connexion (état `NEW`) ;
  3. si une connexion a été initiée en (2) alors on autorise le segment SYN/ACK a traversé depuis Internet ;
  4. si un ACK **correspondant** à cette connexion est envoyé depuis `172.16.42.24`, la connexion est considérée comme `ESTABLISHED` et on autorise tous les segments associés ;
  5. lorsqu'il n'y a plus d'activité ou lorsque l'on intercepte l'envoi réciproque de FIN, on **oublie l'état** mémorisé et plus aucun segments associés ne peuvent traversés (pour éviter de confondre une connexion sans échange d'une connexion terminée sans échange mutuel de FIN, on peut utiliser le TCP Keep-Alive).

Cette troisième proposition repose sur la mémorisation d'un état, «**stateful**», correspondant de l'automate TCP :





L'outil de firewall sous Linux : Netfilter

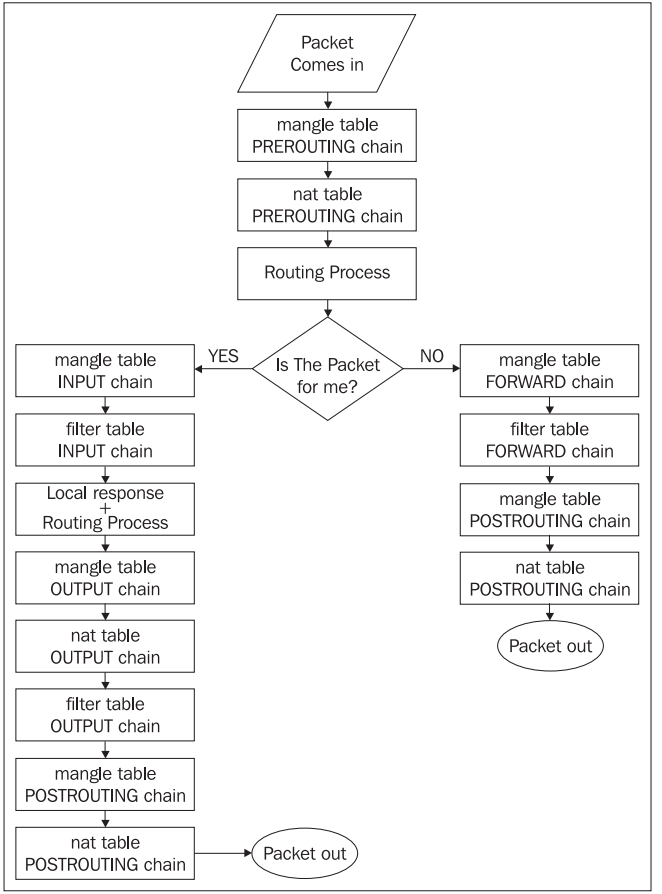
«*Give me your invitation, Sir*»
«Ken Sent Me…»

On parle de : SPI, «*Stateful Packet Inspection*».

La commande permettant de contrôler le firewall netfilter est :

* `iptables` sous IPv4
* `ip6tables` sous IPv6.

La table à apprendre ⟶



La création d'un règle de firewall, se fait à l'aide de la commande `iptables` en choisissant la table ainsi que la chaîne en faisant partie dans laquelle on veut entrer la règle.
Chacune de ses tables correspond à différentes opérations sur les paquets traversant la pile TCP/IP du système : on a jouté des «hooks» ou crochets, dans les différentes étapes du routage, où l'on peut faire intervenir des règles afin de laisser ou non le paquet transiter (éventuellement après modification).

Il existe différentes `tables`, et pour chacune de ces **tables** il existe différentes **chaînes** :

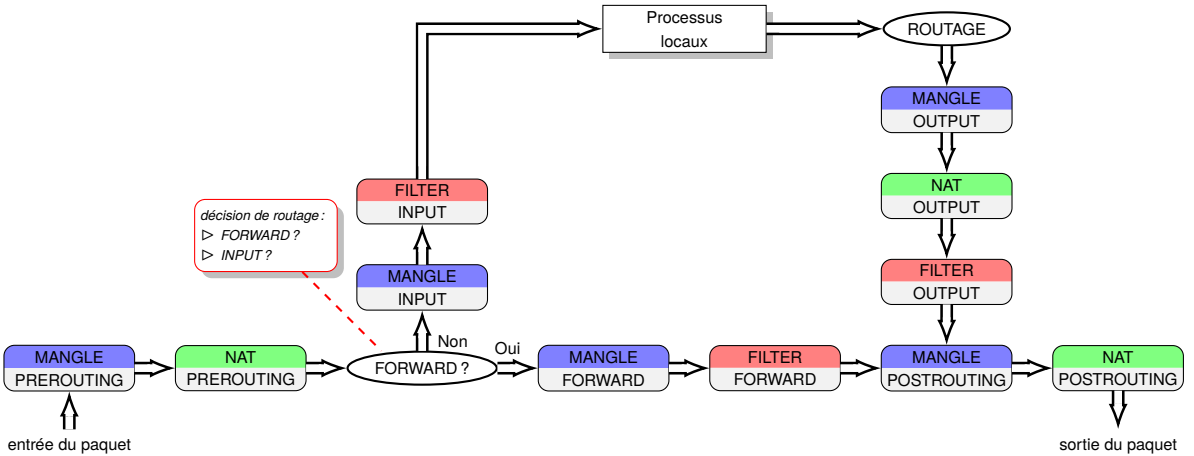
- `filter` : celle utilisée par défaut lorsque l'on indique pas explicitement la table ;
  - `INPUT` : traite les paquets à destination de la machine locale ;
  - `FORWARD` : traite les paquets qui traverseront par routage la machine locale (ne pas oublier d'activer l'option de «forwarding» du système) ;
  - `OUTPUT` : traite les paquets créés sur la machine locale et à destination de l'extérieur ;

- `nat` : pour effectuer de la traduction d'adresse source et/ou destination du datagramme (peut impacter les ports dans le cas d'UDP et de TCP)
  - `PREROUTING` : traite les paquets tout juste entrés et avant qu'ils ne soient éventuellement «forwardés» ;
  - `OUTPUT` : traite les paquets créés localement avant qu'il ne soient routés ;
  - `POSTROUTING` : traite les paquets modifiés avant qu'ils ne quittent la machine mais après avoir été routés ;

- `mangle` : pour effectuer des transformations sur certains champs du datagramme et de son contenu.
  - `PREROUTING`, `INPUT`, `OUTPUT`, `POSTROUTING`, `FORWARD`.

Le parkour du datagramme

Quand un paquet se présente sur une interface, il circule de la manière suivante :



Chaque table intervient à une étape du parcours du datagramme dans ce circuit et peut intervenir sur ce parcours :

- ⋆ modifier le datagramme ;
- ⋆ bloquer le datagramme.

Un datagramme peut être traité :

- ▷ **avant la décision de routage** dans la chaîne `PREROUTING`,
- ▷ **après la décision de routage** dans la chaîne `POSTROUTING`.

Gestion des règles : définition & application à l'aide de la commande `iptables`

Avec `iptables`, on peut réaliser les différentes opérations pour une règle :

* `-A` ou `--append` : ajouter
* `-D` ou `--delete` : effacer
* `-I` ou `--insert` : insérer
* `-R` ou `--replace` : remplacer

Il faut spécifier la **table** et indiquer l'**opération** suivie de la **chaîne concernée**.

Ainsi, avec `iptables -t filter -A INPUT`*, on peut ajouter une règle pour la table de filtrage et dans la chaîne INPUT correspondant aux paquets en entrée d'interface réseau.*

Enfin une **chaîne** doit finir par une «cible» ou «target» afin de faire circuler le paquet.
Cette cible est indiquée avec l'option `-j` pour «jump».
Ainsi, la règle `iptables -t filter -A INPUT -j DROP` *va rejeter tous les paquets en entrée.*

Dans le cas de plusieurs interfaces, on peut préciser avec `-i` l'interface d'entrée et `-o` l'interface de sortie.
Il faut être cohérent : un `-i eth0` *s'applique sur la chaîne INPUT mais peut-être pas sur la chaîne OUTPUT.*

Quand un paquet entre dans une chaîne, il parcours les règles dans l'ordre de leur définition :

* jusqu'à **en trouver une qui s'applique sur lui** : il poursuit son chemin dans la «cible» indiquée (éventuellement une autre chaine lorsqu'elle a été définie par l'utilisateur) ;

* jusqu'à **ce qu'il n'y en est plus** : la **règle générale** ou «policy» s'applique : soit ACCEPT ou DROP (on définie la policy associée à un chaîne avec l'option `-P` et en indiquant une «cible» qui doit être non définie par l'utilisateur).
  Exemple : `iptables -t filter -P FORWARD DROP`.

Le datagramme parcours les règles en cherchant la première qui lui correspond, «*match*».

```
 `INPUT'                          `test'
----------------------------    ----------------------------
| Règle1: -p ICMP -j DROP  |    | Règle1: -s 192.168.1.1   |
|--------------------------|    |--------------------------|
| Règle2: -p TCP -j test   |    | Règle2: -d 192.168.1.1   |
|--------------------------|    ----------------------------
| Règle3: -p UDP -j DROP   |
----------------------------
```

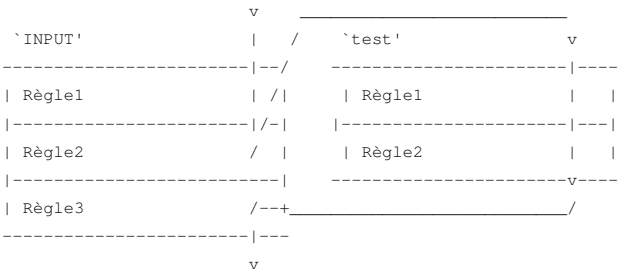
Pour **définir ses propres chaînes** et pouvoir **combiner des sélections complexes**, on utilise l'option :

`iptables -N nom_chaine`

(sur l'exemple ci-contre, la chaîne «test» a été ajoutée)

On sort de la chaine utilisateur :

- ⋆ avec la «cible» **RETURN** qui permet de retourner dans la chaîne d'où l'on a été envoyé
- ⋆ par **épuisement des règles sans concordance** (circulation ci-contre).



Faire la **concordance** entre le datagramme (ou même la trame qui l'a contenu) et une **règle** :

- ◇ avec des options de la commande `iptables` : `-p` ou `--protocol` : qui permet de spécifier le protocole que doit contenir le paquet (tcp, udp ou icmp), `-s` pour l'@IP source, `-d` pour l'@IP destination ;
- ◇ avec de nombreux modules et l'option `-m` ou `-match` : cette option est suivie du nom du module à utiliser, ainsi que des options spécifiques à ce module préfixée avec `--`

De même, il existe **différentes cibles**, *target*, qui sont paramètrables avec des options.

Par protocole

* `tcp` : *permet de traiter les connexions TCP et les options présentes dans les segments* :
  - `--syn` : *indique que le segment doit avoir le bit SYN à 1 à l'exclusion des autres. Cela permet de désigner les demandes d'établissement connexions* ;
  - `--tcp-flags` : *permet d'indiquer les drapeaux qui doivent être présents : `SYN`, `RST`, `ACK`, `FIN`, `URG`, `PSH` et `ALL` et `NONE`.* Il est nécessaire de donner une liste des drapeaux que l'on veut surveiller sous forme d'une liste de valeurs séparées par des virgules puis après un espace des drapeaux qui doivent être à 1. Par exemple : `--tcp-flags SYN,RST,ACK,FIN` (uniquement `SYN` est équivalent à `--syn`) ;
  - `--mss` : *permet de prendre en compte la taille maximale du segment*.
  - `--source-port` ou `--sport` : *on peut éventuellement indiquer un intervalle avec la notation `80:90`* ;
  - `--destination-port` ou `--dport` : *idem*.

```
iptables -t filter -A OUTPUT -p tcp --syn --sport 80 -j DROP
```

* `udp` : *permet de traiter le protocole UDP*
  - `--source-port` ou `--sport` : *on peut éventuellement indiquer un intervalle avec la notation `80:90`* ;
  - `--destination-port` ou `--dport` : *idem*.

```
iptables  -A INPUT -i eth0 -p udp --dport 67:68 --sport 67:68 -j ACCEPT
```

Cette règle permet de laisser passer les paquets DHCP.

* `icmp` : *pour la gestion du protocole icmp* :
  - `--icmp-type` : *permet de spécifier le type du paquet icmp*

```
iptables -A INPUT -p icmp --icmp-type 8 -j DROP
```

Cette règle supprime les messages ICMP d'«echo request»

Par état ou «state» : *Permet de tenir compte de l'état d'une connexion pour disposer d'un firewall «statefull»*

- `--state` : *où l'état est indiqué par une liste d'états de connexion séparés par des virgules*
- `NEW` : *correspond à l'établissement d'une nouvelle connexion* ;
- `ESTABLISHED` : *correspond aux paquets d'une connexion déjà établie* ;
- `RELATED` : *le paquet est en rapport avec une connexion déjà existante (un message ICMP d'erreur ou par exemple, le protocole FTP, où deux connexions sont liées, l'une de contrôle et l'autre de données)* ;
- `INVALID` : le datagramme ne peut être identifié ou ne correspond pas à un état attendu.

La configuration initiale du firewall : verrouillage maximal

```
# Vide toutes les chaînes
/sbin/iptables --flush
# Autorise tout le trafic possible sur l'interface réseau interne de boucle (loopback)
/sbin/iptables -A INPUT -i lo -j ACCEPT
/sbin/iptables -A OUTPUT -o lo -j ACCEPT

# Positionne les Policies par défaut : on bloque tout
/sbin/iptables -P INPUT DROP
/sbin/iptables -P OUTPUT DROP
/sbin/iptables -P FORWARD DROP

# Autorise le trafic déjà établie à passer à travers le routeur
/sbin/iptables -A INPUT -m state --state ESTABLISHED,RELATED -j ACCEPT
# Autorise tout trafic en sortie (outbound)
/sbin/iptables -A OUTPUT -m state --state NEW,ESTABLISHED -j ACCEPT
```

On autorise les datagrammes associés à une connexion établie, `ESTABLISHED, RELATED` (autorisée par une règle utilisant `NEW`).

*Ensuite, viendront les règles autorisant **explicitement** l'établissement d'une connexion, `--syn` , suivant une concordance précise.*

Par exemple, pour les connexions SSH :

```
iptables -t filter -D INPUT -p tcp --dport 22 --syn -j ACCEPT
```

ou de manière équivalente :

```
iptables -t filter -D INPUT -p tcp --dport 22 -m state --state NEW -j ACCEPT
```

On peut vérifier avec la commande `sudo conntrack -L`, l'autorisation du port 22, «SSH» :

```
tcp      6 431264 ESTABLISHED src=192.168.42.122 dst=192.168.42.83 sport=58143 dport=22 src=192.168.42.83
dst=192.168.42.122 sport=22 dport=58143 [ASSURED] mark=0 use=2
```

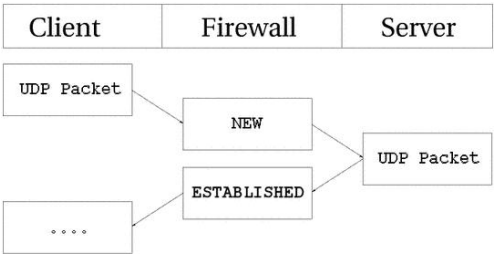
sur les anciennes versions de Linux, on consultait le fichier spécial `/proc/net/ip_conntrack`.

On peut également utiliser la commande `sudo conntrack -E` pour une suveillance en «temps réel».

Gestion des datagrammes UDP et ICMP

Pour permettre au Firewall d'accepter les datagrammes en retour, la partie «*connection tracking*» de netfilter :

- ▷ enregistre les datagrammes en sortie ;
- ▷ autorise pendant une certaine durée un datagramme en retour :
  - ◇ de TSAP inverse à celui envoyé dans le cas du protocole UDP ;

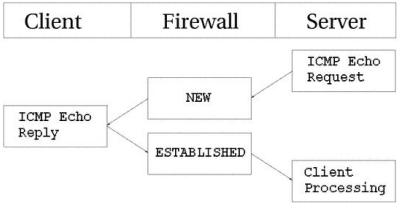


Avec la commande `conntrack` :

```
udp 17 30 src=192.168.1.2 dst=192.168.1.5 sport=137
dport=1025 [UNREPLIED] src=192.168.1.5 dst=192.168.1.2
sport=1025 dport=137 use=1
```

La valeur 30 indique un temps d'attente de 30 secondes.
`[UNREPLIED]` indique que le paquet réponse n'a pas été reçu.

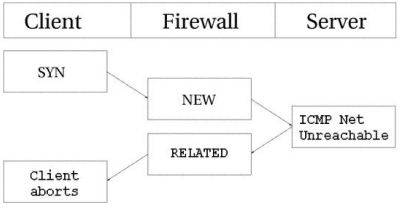
  - ◇ de réponse ICMP dans le cas de l'envoi d'un «echo request» (pour un *ping*) ;

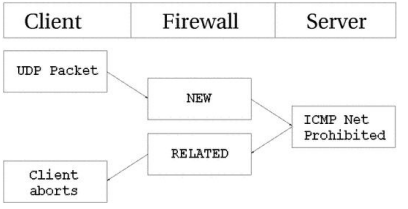


Pour le protocole ICMP, on a une notion de :

- ⋆ «NEW, ESTABLISHED» : lors d'un «ICMP Echo Request» ;
- ⋆ «NEW, RELATED» : lorsque les protocoles UDP et TCP sont susceptibles d'échouer, on autorise le paquet ICMP d'erreur à traverser le firewall en retour.

Ne pas oublier d'autoriser le trafic «RELATED» :





Commandes utiles pour la gestion de l'ensemble des règles du firewall

En utilisant l'option supplémentaire `-h` *on peut obtenir de l'aide sur l'utilisation d'un module donné :* `iptables -m limit -h`.

L'utilisation d'un «!» devant une sélection permet de faire le contraire (c'est la négation de l'option de sélection).

Pour nettoyer une table des chaînes définies, il faut utiliser l'option `-F` pour «flush» :

xterm
```
$ iptables -t nat -F PREROUTING
```

Pour afficher l'ensemble des règles du firewall :

xterm
```
$ sudo iptables -t filter -L
```

Seul l'administrateur peut afficher le contenu des règles de firewall, pour des raisons de sécurité.

xterm
```
Chain INPUT (policy ACCEPT)
target     prot opt source     destination
ACCEPT     tcp  --  anywhere   anywhere   tcp dpt:ssh flags:FIN,SYN,RST,ACK/SYN limit: avg 3/min burst
1
DROP       tcp  --  anywhere   anywhere   tcp dpt:ssh flags:FIN,SYN,RST,ACK/SYN

Chain FORWARD (policy ACCEPT)
target     prot opt source     destination

Chain OUTPUT (policy ACCEPT)
target     prot opt source     destination
```

Commandes utiles pour la gestion de l'ensemble des règles du firewall

Pour surveiller, l'application des règles et voir si elles se déclenchent :

```
xterm
$ sudo watch ebtables -L --Lc
```

```
xterm
$ sudo watch iptables -t nat -nvL
```

Pour sauvegarder/restaurer ce que l'on a définit dans iptables :

- `iptables-save`
  - `-c` : permet de conserver les valeurs de compteurs à fin de statistiques ;
  - `-t` : permet de sélectionner la table à sauvegarder (si l'option est omise, toutes les tables sont sauvegardées).
- `iptables-restore`
  - `-c` : permet de restaurer les valeurs de compteur ;
  - `-n` : permet de ne pas effacer les règles existantes avant restauration.

Pour réinitialiser les compteurs de concordance de règles du firewall :

```
xterm
$ sudo iptables -Z FORWARD
```

Pour suivre l'activité des transmissions de trames du protocole `TCP` sous shell avec `tcpdump` :

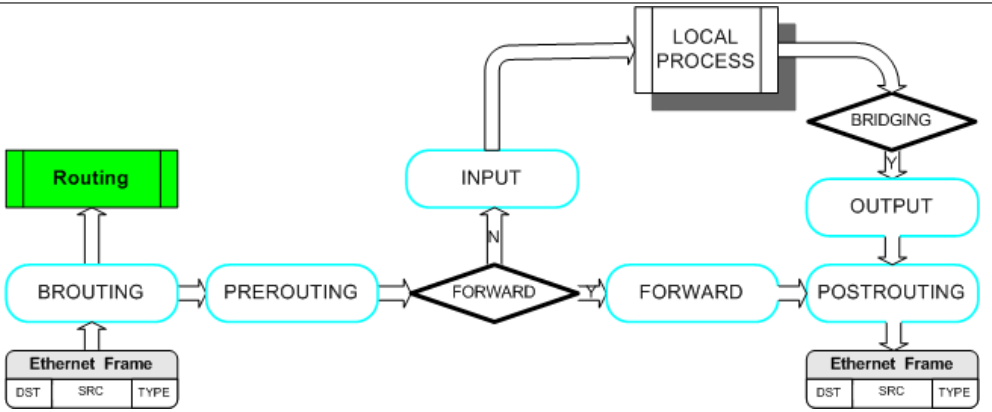
```
xterm
$ sudo tcpdump -ev -i eth0 tcp and port 80
```

Ici, on ne s'intéresse qu'au trafic TCP, de port source ou destination 80.

On peut également utiliser l'affichage «temps réel» des événements avec `conntrack` :

```
xterm
$ sudo conntrack -E
```

Filtrage & Segmentation



Il est possible d'établir des règles de firewall sur les ponts, c-à-d sur les trames ethernet de niveau 2, à l'aide d'une extension de NetFilter les «`ebtables`».

Pour pouvoir les utiliser, il faut installer le paquetage suivant :

xterm
```
$ sudo apt-get install ebtables
```

Il est ainsi possible :

▷ de faire du filtrage en tenant compte que l'on travaille au niveau des trames :

```
ebtables -P FORWARD DROP
ebtables -A FORWARD -p Ipv4 -j ACCEPT
ebtables -A FORWARD -p ARP -j ACCEPT
```

Comme pour le firewall de niveau 3, basé sur iptables, on peut définir une «policy», puis autorisé le trafic IP et ARP seulement.

Les options «-P», pour «policy», «-A» pour «Add», «p» pour «protocol» sont similaires à celles utilisées pour `netfilter` *qui sera vu plus loin.*

Il est possible de faire :

▷ du filtrage :

```
ebtables -A FORWARD -p IPV4 --ip-protocol 1 -j DROP
```

Ici, on utilise la table par défaut, «filter», et on met une règle sur la section «FORWARD» afin de filtrer les trames contenant un datagramme IP et un contenu de protocole numéro 1, c-à-d ICMP.

▷ faire de la protection anti-spoofing :

```
ebtables -A FORWARD -p Ipv4 -ip-src 172.16.2.5 -s ! 00:11:22:33:44:55 -j DROP
```

Ici, la trame ne traversera le pont que si l'adresse MAC et l'adresse IP correspondent.

Remarque

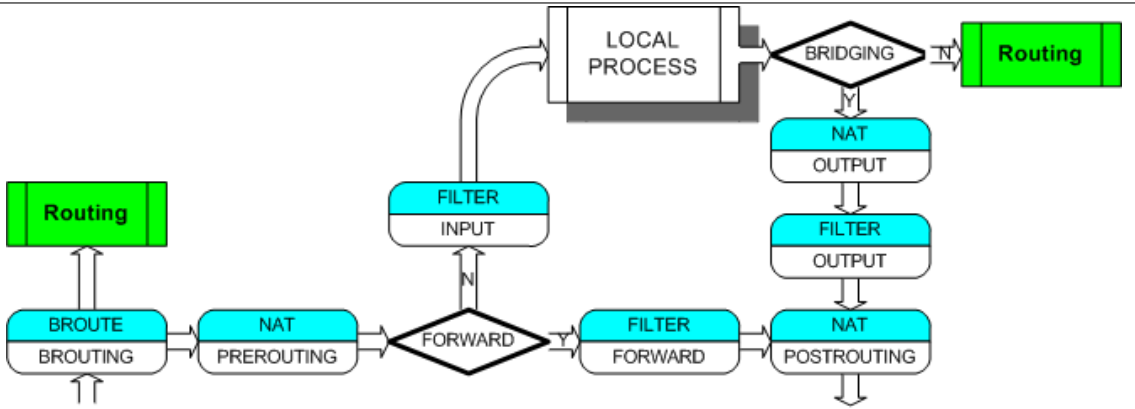
Cette règle peut également s'écrire avec iptables :

```
iptables -A FORWARD -s 172.16.1.4 -m mac --mac-source ! 00:11:22:33:44:55 -j DROP
```

Le traitement est moins rapide car ebtables intervient en amont d'iptables.

On peut également traiter plusieurs associations, en utilisant une *policy* basée sur DROP :

```
ebtables -A FORWARD -p IPv4 --among-dst 00:11:22:33:44:55=172.16.1.4, 00:11:33:44:22:55=
-j ACCEPT
```

▷ faire de la «traduction» d'adresse, «NAT», sur les adresses MAC, seulement pour des adresses qui existent sur des interfaces différentes (si la trame est réécrite avec une adresse MAC destination qui est sur l'interface d'où elle est venue intialement, elle ne sera pas retransmise) :

```
ebtables -t nat -A PREROUTING -d 00:11:22:33:44:55 -i eth0 -j dnat --to-destination
54:44:33:22:11:00
```

Ici, on utilise la table «nat» pour réécrire des adresses MAC dans les trames au passage dans le pont.

```
ebtables -t nat -A POSTROUTING -s 00:11:22:33:44:55 -i eth0 -j snat --to-source
54:44:33:22:11:00 --snat-target ACCEPT
```

Ici, on change l'adresse MAC source.

▷ répondre automatiquement à des requêtes arp :

```
ebtables -t nat -A PREROUTING -p arp --arp-opcode Request --arp-ip-dst 192.168.127.2
-j arpreply --arpreply-mac de:ad:be:ef:ca:fe
```

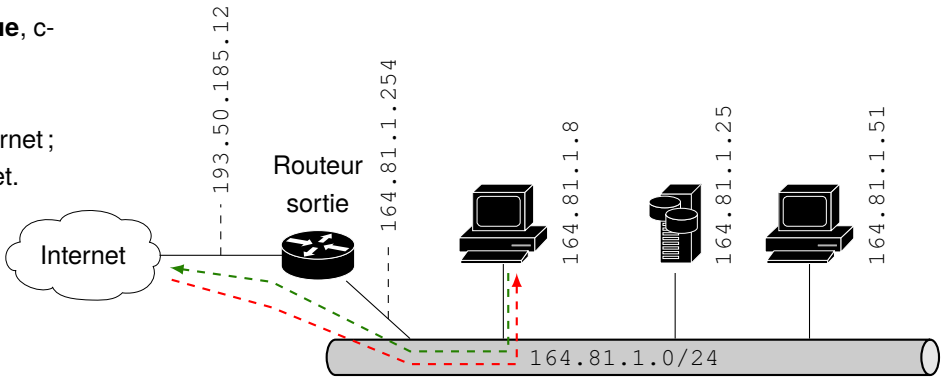
Ici, on peut répondre automatiquement à une requête ARP, par exemple en donnant une adresse MAC différente pour une adresse IP connue et faire du MiTM…

- le «**filtrage**» : *autoriser ou non des connexions* :
  - sur le poste local (où tourne le Firewall) ;
  - sur les machines du réseau situées derrière le routeur/firewall ;
- le «**NAT**» : *faire de la traduction d'@IP lors du routage des datagrammes* :
  - partage d'@IP globale ;
  - protection des postes du réseau situés derrière le routeur/firewall ;
  - mise en place de «proxy transparent» ;
  - faire du «port forwarding» ;
- la «**QoS**» : faire du «*trafic shaping*», c-à-d adapter les datagrammes en sortie du routeur/firewall à la QoS :
  - classifier le trafic : reconnaître les différents flux (services TCP, RTP, «*Real-Time-Protocol*» basé sur UDP, VoIP, IPtv, p2p, *etc.*). On parle de «*Deep Packet Inspection*», car on s'intéresse à son contenu.
  - marquer les datagrammes pour adapter leur prise en charge lors de leur acheminement suivant de la QoS :
    - en sortie du routeur : en contrôlant le débit ;
    - lors de l'acheminement du datagramme : en fixant les champs TOS/DSCP pour les autres routeurs ;
  - faire de l'équilibrage de charge ;
  - tenir compte de quota, des heures de bureau *etc.*
- la protection **contre les attaques** :
  - contre les attaques «brute force» :
    - ralentir les accès en général ;
    - ralentir les accès suivant leur origine ;
  - protéger contre les paquets surdimensionnés (le «*ping of death*», les «*buffer overflow*», *etc.*),
  - protéger contre des tentatives de réseaux connus (par exemple de Chine ou de Fédération de Russie…)
- la **programmation** en collaboration avec le firewall : le «NFQueue».

Utilisation d'adresses réseaux publiques

Le réseau utilise une **adresse publique**, c-à-d «*routable*» sur Internet.
Une machine :

- ▷ communique directement **vers** Internet ;
- ▷ **peut** être contactée **depuis** Internet.



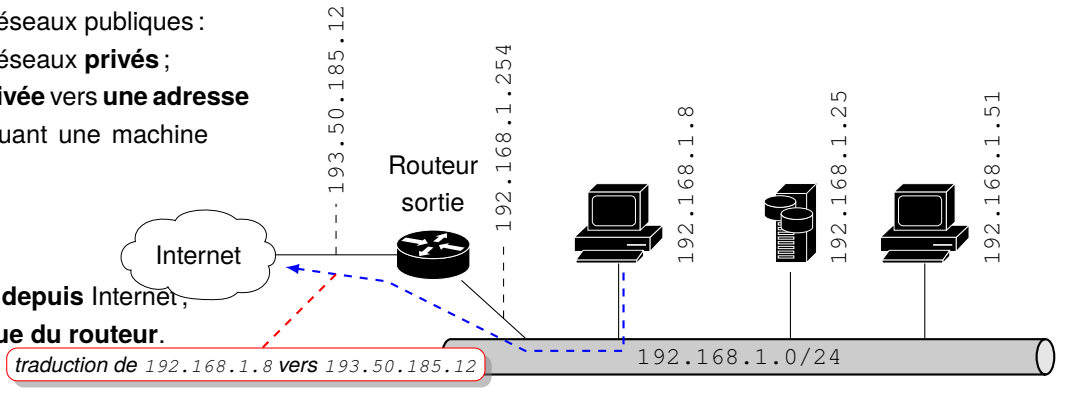
Utilisation d'adresses réseaux privées

On **économise** des adresses réseaux publiques :

- ▷ on utilise des adresses de réseaux **privés** ;
- ▷ on «traduit» une **adresse privée** vers **une adresse publique** et inversement quant une machine veut accéder à Internet.

Une machine :

- ▷ communique **vers** Internet ;
- ▷ **ne peut pas** être contactée **depuis** Internet ;
- ▷ emprunte **l'adresse publique du routeur**.



Le NAT :

- ▷ **Économise** les adresses publiques car **il n'existe plus** d'adresses IPv4 libres ;
- ▷ **Protège** les machines du réseau : elles **ne sont pas accessibles** directement depuis Internet.

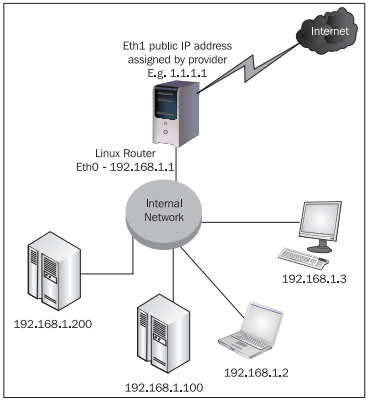
Rappel du contexte

- ▷ pour communiquer sur Internet, il faut une @IP unique et publique (pas privée) ;
- ▷ du fait d'un manque d'@IP publiques, il est **nécessaire de partager** une même @IP avec plusieurs postes ;

La solution du NAT

Le NAT, «*Network Address Translation*» permet de traduire une @IP en une autre : réécrire l'@IP source ou destination d'une machine située *derrière* un routeur réalisant du NAT (c'est le cas des «box ADSL»).

Autre avantage du NAT : **dissimuler** des postes : il ne sont accessibles qu'au travers du routeur et non directement.
*Pour communiquer depuis l'extérieur vers une machine située à l'intérieur, il faut **configurer explicitement** le routeur NAT.*



On utilise les réseaux dits privés, *réseaux non «routables»*, (RFC 1918) :

* `10.0.0.0/8`
* `172.16.0.0/12`
* `192.168.0.0/16`

Pour qu'une machine du réseau privé communique sur Internet, le routeur NAT :

- ◇ **réécrit** dans les datagrammes l'@IP privée de la machine avec sa propre @IP publique ;
- ◇ doit **différencier** ses datagrammes de ceux des machines du réseau privé ;
- ◇ doit **garder une trace** de toutes les connexions TCP et UDP (!!!) qui le traverse ;

On parle de «*connection tracking*».

C'est le principe des «***stateful firewalls***».

Le «connection tracking»

- pour UDP, on **mémorise** l'envoi d'un datagramme UDP et on **autorise** le retour du datagramme UDP correspondant de TSAP inverse (d'où le terme impropre de «connexion» !!!)

```
xterm
pef@solaris:~$ sudo conntrack -L
tcp      6 431996 ESTABLISHED src=199.47.217.147 dst=192.168.42.83 sport=80 dport=51911 src=192.168.42.83
            dst=199.47.217.147 sport=51911 dport=80 [ASSURED] mark=0 use=2
udp      17 3 src=192.168.42.59 dst=192.168.42.255 sport=138 dport=138 [UNREPLIED] src=192.168.42.255
            dst=192.168.42.59 sport=138 dport=138 mark=0 use=2
```

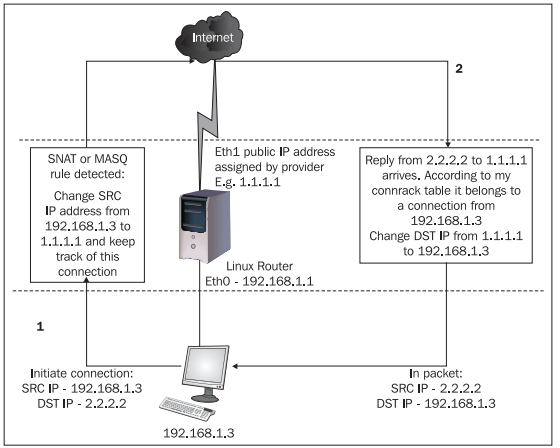
- pour UDP & TCP, on mémorise les @IPs, les numéros de port, le protocole, l'état de la connexion et les timeouts.

Différents scénarios de réécriture d'adresse

- un vers un, *one-to-one*, 1:1 : une seule @IP privée est traduite en une @IP publique ;
- un vers plusieurs, *one-to-many*, 1:many : une @IP privée est réécrite en différentes @IP publiques : pour chaque connexion le routeur NAT choisie une @IP publique parmi un ensemble disponible (utile pour le respect de vie privée ou *privacy*) ;
- plusieurs vers une, *many-to-one*, many:1 : plusieurs @IP privées sont réécrites en une même @IP publique.
  Si l'@IP appartient au routeur NAT, on parle de «masquerading».
- plusieurs vers plusieurs, *many-to-many*, many:many : plusieurs @IP privées sont réécrites vers un ensemble d'@IP publiques.

SNAT, «*Source NAT*» & Masquerade

- ▷ le SNAT correspond à la **réécriture de l'@IP source** du paquet ;
- ▷ en absence de SNAT, une connexion initiée depuis une machine du réseau privée à destination d'Internet est impossible ⟹ les paquets réponses ne sont pas routables !
- ▷ il existe trois formes :
  - ◇ le «*static SNAT*» : une ou plusieurs @IP privées sont réécrites vers la même @IP publique ;
  - ◇ le «*dynamic SNAT*» : une ou plusieurs @IP privées sont réécrites vers un ensemble d'@IP publiques. Netfilter utilise pour chaque nouvelle connexion l'@IP publique la moins récemment utilisée ou une @IP aléatoire.
  - ◇ le «*masquerading*» ou `MASQ` fonctionne comme le «*static NAT*», mais on ne spécifie pas l'@IP à utiliser : c'est celle de l'interface de sortie du routeur empruntée par le datagramme.



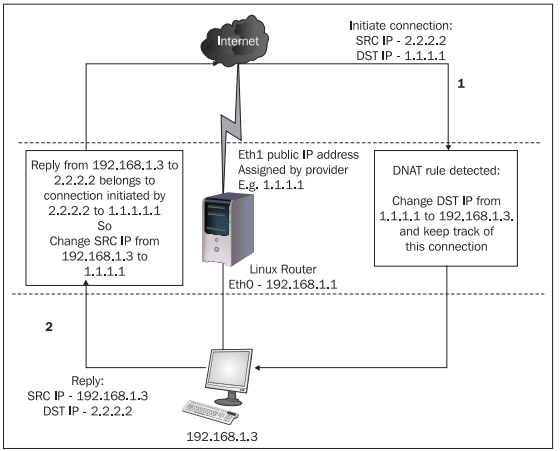
Exemple :

- ⋆ la machine `192.168.1.3` veut communiquer avec `2.2.2.2` ;
- ⋆ le routeur effectue du SNAT et réécrit l'@IP source privée en `1.1.1.1`, son @IP publique de sortie ;
- ⋆ le routeur mémorise la connexion (les informations sont consultables dans `/proc/net/ip_conntrack`) ;
- ⋆ au retour le routeur réécrit l'@IP destination du paquet vers `192.168.1.3`.

DNAT, «*Destination NAT*»

- ▷ le DNAT permet de faire correspondre une @IP publique à une @IP privée ;
- ▷ le DNAT est le contraire du SNAT (si on utilise le SNAT pour traduire une @IP privée en une @IP publique et le DNAT pour faire l'opération inverse sur les mêmes @IP publique et privée, alors on parle de «full NAT»).
- ▷ en absence de DNAT, une connexion initiée depuis Internet à destination d'une machine du réseau privée est impossible ;
  C'est une bonne protection pour les machines du réseau privé.

Le DNAT est utilisé dans le cas de serveurs situés derrière le routeur NAT : la même @IP publique est traduite en différentes @IP privées en fonction **du numéro ou du protocole**. On parle de «*port forwarding*».



Exemple :

- ⋆ une connexion est initiée de `2.2.2.2` vers `1.1.1.1` ;
- ⋆ le routeur NAT détecte que la connexion ne concerne pas lui-même mais un serveur du réseau privé ;
- ⋆ le routeur réécrit l'@IP destination en l'@IP privée du serveur `192.168.1.3`, capable de traiter cette connexion ;
- ⋆ les réponses en provenance de `192.168.1.3` sont réécrites depuis la source `1.1.1.1`

*Sur certaines box ADSL, on parle de DMZ, «Demilitarized Zone» pour la fonction de DNAT : il redirige tout le trafic en provenance de l'extérieur vers **une machine du réseau privé** en plus de faire du masquerading en sortie.*

Le «*Full NAT*» ou «*Full Cone NAT*»

- ⋆ Le SNAT permet de faire de la traduction d'adresse uniquement lorsqu'une connexion est réalisée depuis le réseau privé vers l'extérieur.
  - ◇ la connexion est mémorisée ;
  - ◇ on change l'@IP source vers une @IP publique permettant le retour vers le routeur de sortie ;
  - ◇ les paquets en retour subissent **automatiquement** le DNAT associé ;
- ⋆ Le DNAT permet de réécrire l'@IP destination d'une connexion initiée depuis l'extérieur vers une adresse publique accessible par le routeur et un port ou protocole choisie :
  - ◇ la connexion est mémorisée ;
  - ◇ on change l'@IP destination vers une @IP du réseau privée.
  - ◇ les paquets en retour subissent **automatiquement** le SNAT associé.
- ⋆ Le «*Full NAT*» correspond à
  - ◇ effectuer le DNAT d'une @IP publique **toujours** vers la même @IP privée ;
  - ◇ effectuer le SNAT d'une @IP privée **toujours** vers la même @IP publique ;
  - ◇ ce fonctionnement est appelé «DMZ» par les box ADSL : toute connexion non initiée depuis le réseau privée (donc non référencée dans la table du «*connection tracking*») est redirigée vers la même @IP privée et ce sans autorisation supplémentaire (pas de DNAT à configurer explicitement ni de SNAT non plus).

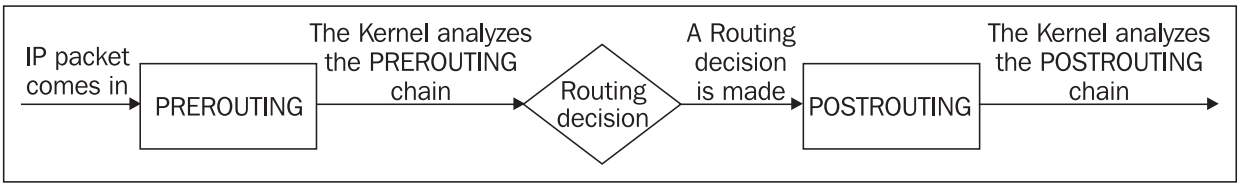
Le PAT, «*Port Address Translation*»

Il correspond à :

- ◇ permettre au routeur de faire du SNAT en mode masquerading (*many-to-one*) en changeant au besoin le port d'origine ;
- ◇ faire du DNAT en fonction d'un numéro de port choisi qui peut être **différent** du port de destination final.

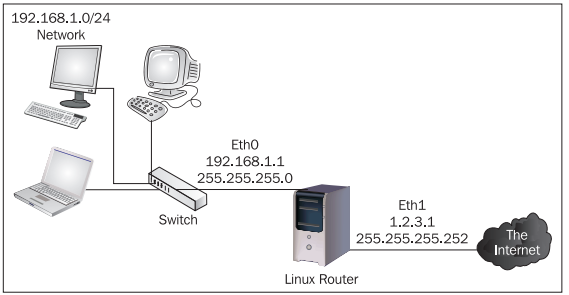
Les chaînes associées

```
xterm
pef@solaris:~$ sudo iptables -t nat -nvL
Chain PREROUTING (policy ACCEPT 0 packets, 0 bytes)
 pkts bytes target     prot opt in     out     source               destination

Chain INPUT (policy ACCEPT 0 packets, 0 bytes)
 pkts bytes target     prot opt in     out     source               destination

Chain OUTPUT (policy ACCEPT 0 packets, 0 bytes)
 pkts bytes target     prot opt in     out     source               destination

Chain POSTROUTING (policy ACCEPT 0 packets, 0 bytes)
 pkts bytes target     prot opt in     out     source               destination
```



- la chaîne `PREROUTING` : elle est traitée **avant** toute décision de routage : c'est dans cette chaîne que l'on effectue le DNAT : on changera l'@IP de destination du datagramme et on laissera le routage trouver la destination que l'on vient de changer ;
- la chaîne `POSTROUTING` : elle est traitée **après** la décision de routage : c'est dans cette chaîne que l'on le SNAT : on a une destination pour le datagramme après le routage et maintenant on peut changer l'origine du datagramme en l'@IP correspondant à l'interface de sortie (c-à-d l'@IP du routeur qui permet d'être joint en retour sur cette interface de sortie).

SNAT



xterm
```
iptables -t nat -A POSTROUTING -s 192.168.1.0/24
      -j SNAT --to 1.2.3.1
```

Ou dans le cas où l'@IP du routeur est configurée dynamiquement (par exemple en DHCP)

xterm
```
iptables -t nat -A POSTROUTING -s 192.168.1.0/24
      -j MASQUERADE
```

Pour donner un ensemble d'@IP :

xterm
```
iptables -t nat -A POSTROUTING -s 192.168.1.0/24
      -j SNAT --to 1.2.4.0- 1.2.4.32
```

DNAT

xterm
```
iptables -t nat -A PREROUTING -d 1.2.4.1 -j DNAT --to 192.168.1.50
```

Redirection vers une machine sur le port SSH, lorsque la connexion vient sur le routeur sur le port 65521 :

xterm
```
iptables -t nat -A PREROUTING -d 1.2.4.2 -p tcp --dport 65521 -j DNAT --to 192.168.1.100:22
```

Proxy transparent

Un proxy transparent est un moyen de forcer les utilisateurs à utiliser un proxy, c-à-d un intermédiaire de connexion, même si leur navigateur n'est pas configuré pour le faire :

xterm
```
iptables -t nat -A PREROUTING -s 192.168.1.0/24 -p tcp --dport 80 -j REDIRECT --to-port 3128
```

Le proxy est installé sur le routeur et le «REDIRECT» effectue un DNAT vers l'@IP du routeur lui-même.

Le proxy peut être utilisé pour servir de cache Web, ou pour filtrer les contenus dangereux.

Contournement d'un accès à un serveur local par l'utilisation du NetFilter local

Imaginons, qu'un serveur, ici ElasticSearch, n'autorise que les accès depuis `localhost` *sur son port 9200.*
À l'aide du DNAT, on réécrit l'adresse des paquets provenant de l'extérieur vers `127.0.0.1`:

xterm
```
$ sudo sysctl -w net.ipv4.conf.all.route_localnet=1❶
$ sudo iptables -t nat -A PREROUTING -p tcp --dport 4000 -j DNAT --to 127.0.0.1:9200
```

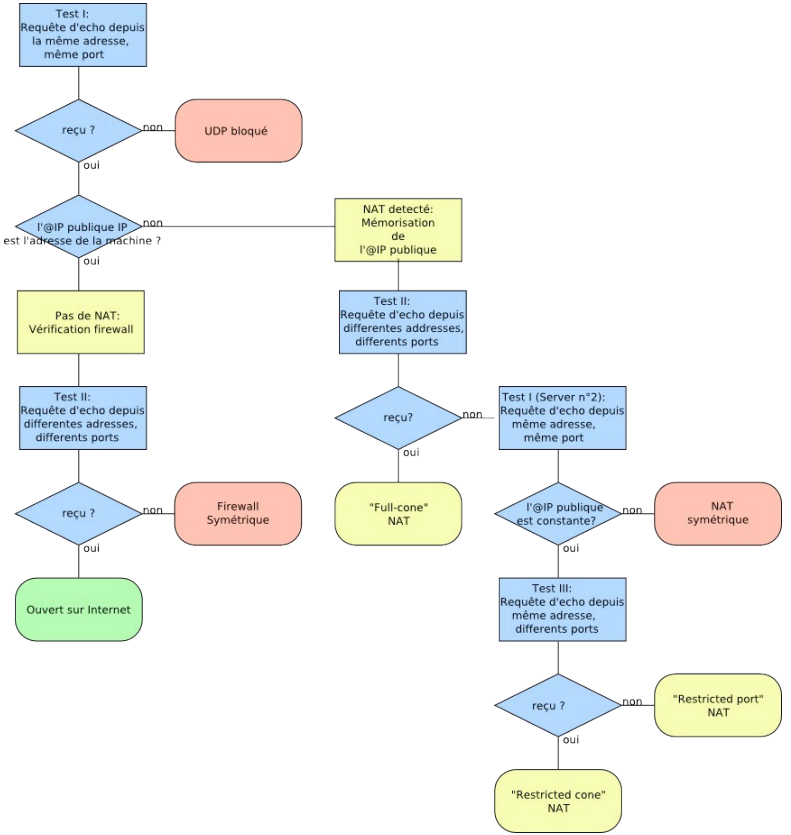
Sans la configuration du noyau avec ❶*, le noyau rejettera le paquet en le considérant comme «martien», car provenant de l'extérieur et possédant une adresse de destination en* `127.0.0.1`.

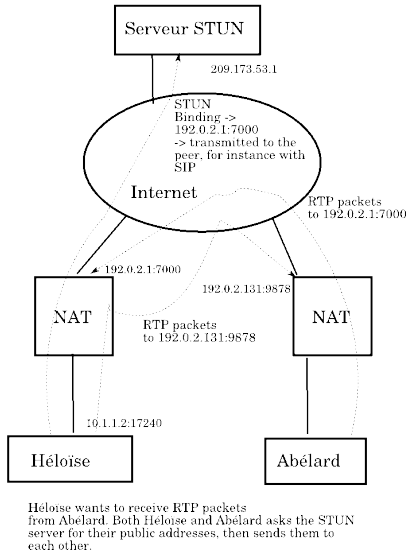
Test d'accès depuis la machine `192.168.0.104` *vers la machine* `192.168.0.130` *hébergeant le serveur :*

xterm
```
$ curl http://192.168.0.130:9200
curl: (7) Failed to connect to 192.168.0.130 port 9200: Connection refused
$ curl --local-port 7680 http://192.168.0.130:4000
{
  "name" : "mU7Gx_r",
  "cluster_name" : "elasticsearch",
  "cluster_uuid" : "nNG_gPDfRd2xOnkIFCJ_Og",
  "version" : {
    "number" : "5.6.2",
    "build_hash" : "57e20f3",
    "build_date" : "2017-09-23T13:16:45.703Z",
    "build_snapshot" : false,
    "lucene_version" : "6.6.1"
  },
  "tagline" : "You Know, for Search"
}
```

xterm
```
$ sudo conntrack -L
tcp      6 431999 ESTABLISHED src=192.168.0.104 dst=192.168.0.130 sport=61977 dport=22
src=192.168.0.130 dst=192.168.0.104
sport=22 dport=61977 [ASSURED] mark=0 use=1
tcp      6 119 TIME_WAIT src=192.168.0.104 dst=192.168.0.130 sport=7680 dport=4000 src=127.0.0.1
dst=192.168.0.104 sport=9200
dport=7680 [ASSURED] mark=0 use=1
conntrack v1.4.3 (conntrack-tools): 7 flow entries have been shown.
```

Ce protocole sert à savoir si l'utilisateur est «derrière» un NAT, comment est configuré le NAT et si un flux UDP entrant est possible (par exemple pour faire de la VoIP).



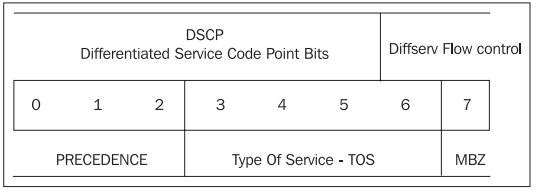


http://www.bortzmeyer.org/5389.html

L'utilisation de cette table permet de **modifier** le contenu de certains champs du datagramme IP :

- le champs TOS, «*Type Of Service*»/DSCP, «*Differentiated Services field*» : modifier la priorité du datagramme IP en fonction de son origine, de son contenu *etc.*
  Cette modification permet de faire de la QoS entre routeurs capables de gérer ces priorités (RFC 2474, 2475).

Precedence Level	Description
7	Stays the same (link layer and routing protocol keep alive)
6	Stays the same (used for IP routing protocols)
5	Express Forwarding (EF)
4	Class 4
3	Class 3
2	Class 2
1	Class 1
0	Best effort



xterm
```
iptables -t mangle -A FORWARD -p tcp --dport 80 -j DSCP --set-dscp 1
iptables -t mangle -A FORWARD -p tcp --dport 80 -j DSCP --set-dscp-class EF
```

- le champs TTL : pour le diminuer : empêcher la sortie de certains datagrammes, ou au contraire l'augmenter : masquer le passage par le routeur ;
  - `--ttl-set` : *positionne la valeur* ;
  - `--ttl-dec` et `--ttl-inc` : décrémente ou incrémente la valeur du TTL.

xterm
```
iptables -t mangle -A PREROUTING -i eth0 -j TTL --ttl-inc 1
```

La table `mangle` : marquer les paquets dans le noyau

La table `mangle` peut **marquer** le datagramme dans le noyau :

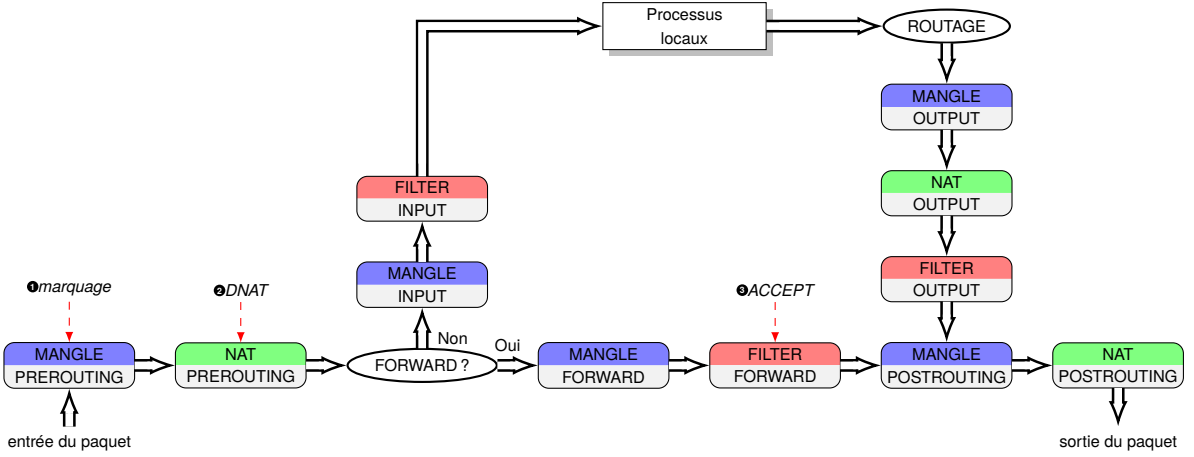
- ▷ cette marque est associée au datagramme lors de son parcours dans son circuit dans la pile TCP/IP ;
- ▷ cette marque n'existe que dans la pile TCP/IP : le contenu du datagramme n'est pas modifié.

Cette **marque** permet de :

- ☐ d'**appliquer de la QoS** sur le paquet en collaboration avec l'outil *traffic control*, `tc`.
  Une fois marqué, il peut rejoindre, par exemple, une file d'envoi associé à une QoS avec l'option `fwmark`.
- ☐ de le **suivre** et le rendre sélectionnable par une règle du firewall malgré une modification réalisée
  Exemple :
  - ◇ la table NAT modifie la destination du datagramme ❷ ;

  ⟹ *le datagramme ne peut plus être sélectionné par la destination dans la table `FILTER`...*
  - ◇ la table `MANGLE` précède la table `NAT` : on le **marque** ❶ ;

  ⟹ on le **sélectionne** ensuite par cette marque ❸ !



Les protocoles d'échanges en «peer-to-peer», c-à-d d'égal à égal :

- sont difficiles à découvrir : les numéros de port varient, les interlocuteurs ont des adresses IP changeante (non connues à l'avance), pas de centralisation (différents serveurs possibles) ;
- consomment beaucoup de débit : de nombreuses connexions peuvent s'établir simultanément rendant difficile le partage de la capacité du support, *bandwidth*, entre les différents utilisateurs du réseau aux dépens des communications prioritaires de l'organisation.

Il est donc **souhaitable** de les contrôler pour soit les interdire, soit leur affecter des débits contraints.

Comment les identifier ? Avec du «*Deep Packet Inspection*», c-à-d en analysant le contenu du datagramme

▷ par le contenu des paquets échangés : *Problème : et si un courrier électronique contient «announce» ?*

```
iptables -N LOGDROP
iptables -F LOGDROP
iptables -A LOGDROP -j LOG --log-prefix "LOGDROP "
iptables -A LOGDROP -j DROP
iptables -A FORWARD -m string --algo bm --string "BitTorrent" -j LOGDROP
iptables -A FORWARD -m string --algo bm --string "BitTorrent protocol" -j LOGDROP
iptables -A FORWARD -m string --algo bm --string "peer_id=" -j LOGDROP
iptables -A FORWARD -m string --algo bm --string ".torrent" -j LOGDROP
iptables -A FORWARD -m string --algo bm --string "announce.php?passkey=" -j LOGDROP
iptables -A FORWARD -m string --algo bm --string "torrent" -j LOGDROP
iptables -A FORWARD -m string --algo bm --string "announce" -j LOGDROP
iptables -A FORWARD -m string --algo bm --string "info_hash" -j LOGDROP
# DHT keyword
iptables -A FORWARD -m string --string "get_peers" --algo bm -j LOGDROP
iptables -A FORWARD -m string --string "announce_peer" --algo bm -j LOGDROP
iptables -A FORWARD -m string --string "find_node" --algo bm -j LOGDROP
```


Sélection selon le contenu

xterm

```
$ sudo iptables -t nat -A PREROUTING  -p udp --dport 53 -m string --hex-string
"|05|query|09|yahooapis|03|com" --algo bm -j DNAT --to-destination 198.245.60.92
```

Ici, on sélectionne les requêtes DNS contenant une recherche du domaine `query.yahooapis.com`.

Le codage d'une question DNS est réalisé de la manière suivante :

- chaque sous-domaine est préfixé par un octet dont la valeur est le nombre de caractères du sous-domaine qu'il précéde :
  Exemple : le sous-domaine «*com*» est sur 3 caractères, «*yahooapis*» est sur 9 caractères et le nom de la machine est `query` sur 5 caractères.
- l'option `--hex-string` du module «*string*» permet d'obtenir l'octet voulu en :
  - ▷ exprimant sa valeur en hexadécimal ;
  - ▷ entourant la valeur par le caractère `|`.

xterm

```
$ iptables -m string --help

string match options:
--from                      Offset to start searching from
--to                        Offset to stop searching
--algo                      Algorithm
[!] --string string         Match a string in a packet
[!] --hex-string string     Match a hex string in a packet
```

L'option «`--algo`» prend la valeur «`bm`», pour Boyer-Moore ou «`kmp`» pour Knuth-Morris-Pratt.
L'option «`--string`» est «*sensible à la casse*».

Un module pour «*suivre*» une séquence d'événements

xterm
```
$ iptables -m recent --help
    --set          Add source address to list, always matches.
    --update       Match if source address in list, also update last-seen time.
    --remove       Match if source address in list, also removes that address from list.
    --name name    Name of the recent list to be used.  DEFAULT used if none given.
    ...
```

- `--set` : ajoute l'@IP source dans la liste ;
- `--update` : vérifie que l'adresse a déjà été vue et mets à jour le timeout de suppression automatique ;
- `--remove` : supprime l'adresse de la liste.

Exemple d'utilisation : blocage d'une requête HTML prédéterminée `GET /w00tw00t.at.ISC.SANS.`

▷ première proposition :

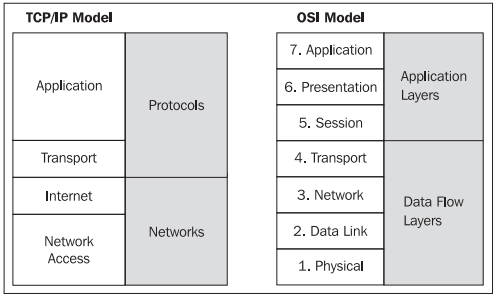
xterm
```
iptables -I INPUT -p tcp --dport 80 -m string --to 70 --algo bm --string 'GET
/w00tw00t.at.ISC.SANS.' -j DROP
```

⟹ **inefficace** : le filtrage lent, peut générer des faux positifs, segment retransmis par le client etc.

▷ seconde proposition : suivre le «*handshake TCP*» et le module «*recent*» :

xterm
```
# Create our w00t chain:
iptables -N w00t
# Redirect incoming TPC packets (port 80 only) to our w00t chain:
iptables -A INPUT -p tcp --dport 80 -j w00t
iptables -A w00t -m recent -p tcp --syn --set
# Wait for the client ACK and update our list:
iptables -A w00t -m recent -p tcp --tcp-flags PSH,SYN,ACK ACK --update
# We received our 3-way handshake: the connection is established
# We're now waiting for the first incoming PSH,ACK. It will contain
# the HTTP (GET, POST, HEAD...) request we are looking for.
# As soon as we get it, we delete our list with '--remove' to ignore any further packets :
iptables -A w00t -m recent -p tcp --tcp-flags PSH,ACK PSH,ACK --remove \
  -m string --to 70 --algo bm --string "GET /w00tw00t.at.ISC.SANS." -j REJECT --reject-with
tcp-reset
```

Le «Layer-7 filter» ⟹ abandonné du à l'augmentation du débit réseau



- des *patchs* à appliquer au noyau : `http://l7-filter.sourceforge.net/`
- utilise le module `ip_conntrack` ⟶ à réserver à des machines avec un CPU suffisant permettant une gestion à plus de 5000 pps, «packet per second» ;
- permet d'identifier Kazaa, HTTP, Jabber, Citrix, Bittorrent, FTP, Gnucleus, eDonkey2000, *etc.*, quelque soit le port utilisé.

Installation et utilisation

Il utilise un répertoire contenant des fichiers de définitions de protocoles disponible sur le site `http://sourceforge.net/projects/l7-filter/`.

Exemple de fichier utilisé pour la détection de bittorent :

```
# Bittorrent - P2P filesharing / publishing tool - http://www.bittorrent.com
# This pattern has been tested and is believed to work well.
# It will, however, not work on bittorrent streams that are encrypted, since
# it's impossible to match (well) encrypted data.

bittorrent

# Does not attempt to match the HTTP download of the tracker
# 0x13 is the length of "bittorrent protocol"
# Second two bits match UDP wierdness
# Next bit matches something Azureus does

# This is not a valid GNU basic regular expression (but that's ok).
^(\x13bittorrent protocol|azver\x01$|get /scrape\?info_hash=get /announce\?info_hash=
|get /client/bitcomet/|GET /data\?fid=)|d1:ad2:id20:|\x08'7P\)[RP]
# This pattern is "fast", but won't catch as much
#^(\x13bittorrent protocol|azver\x01$|get /scrape\?info_hash=)
```

Des EXPRESSIONS RÉGULIÈRES !!!

Le «Layer-7 filter»

xterm
```
routeur:~# iptables -Z
routeur:~# iptables -A OUTPUT -m layer7 --l7proto http
routeur:~# iptables -L OUTPUT -n -v
Chain  OUTPUT (policy ACCEPT 10168 packets, 3433K bytes)
pkts  bytes target   prot  opt   in out   source      destination
0     0              all   --    *     *  0.0.0.0/0   0.0.0.0/0 LAYER7 l7proto http
routeur:~#
```

Pour filtrer :

xterm
```
routeur:~# iptables -A FORWARD -m layer7 --l7proto edonkey -j DROP
```

On peut également faire de la limitation de débit en **marquant** les paquets avant la phase de routage (table mangle en PREROUTING).

Alternative : IPP2P $\Longrightarrow$ intégré dans «*shorewall*» une surcouche de NetFilter

C'est un autre module qui est orienté uniquement sur le trafic P2P.

```
iptables ... -m ipp2p --option ...
```

Option	P2P network	Protocol	Quality
`--edk`	eDonkey, eMule, Kademlia	TCP and UDP	very good
`--kazaa`	KaZaA, FastTrack	TCP and UDP	good
`--gnu`	Gnutella	TCP and UDP	good
`--dc`	Direct Connect	TCP only	good
`--bit`	BitTorrent, extended BT	TCP and UDP	good
`--apple`	AppleJuice	TCP only	(need feedback)
`--winmx`	WinMX	TCP only	(need feedback)
`--soul`	SoulSeek	TCP only	good (need feedback)
`--ares`	Ares, AresLite	TCP only	moderate (DROP only)

D'autres modules

* `time:`

```
xterm
iptables A FORWARD -p tcp -m multiport --dport http,https -o eth0 -i eth1
    -m time --timestart 12:30 --timestop 13:30 --days Mon,Tue,Wed,Thu,Fri -j ACCEPT
```

* `quota:`

```
xterm
iptables -A INPUT -p tcp -m quota --quota 2147483648 -j ACCEPT
iptables -A INPUT -j DROP
```

Dans ce cas là, on préférera faire de la QoS, comme diminuer la priorité que de jeter les datagrammes…

* `nth`: pour faire de l'équilibrage de charge, c-à-d rediriger des connexions vers différents serveurs (round robin)

```
xterm
iptables -A PREROUTING -i eth0 -p tcp --dport 80 -m state --state NEW -m nth --counter 0
--every 4 --packet 0 -j DNAT --to-destination 192.168.0.5:80

iptables -A PREROUTING -i eth0 -p tcp --dport 80 -m state --state NEW -m nth --counter 0
--every 4 --packet 1 -j DNAT --to-destination 192.168.0.6:80

iptables -A PREROUTING -i eth0 -p tcp --dport 80 -m state --state NEW -m nth --counter 0
--every 4 --packet 2 -j DNAT --to-destination 192.168.0.7:80

iptables -A PREROUTING -i eth0 -p tcp --dport 80 -m state --state NEW -m nth --counter 0
--every 4 --packet 3 -j DNAT --to-destination 192.168.0.8:80
```

De manière aléatoire :

```
xterm
iptables -A PREROUTING -i eth0 -p tcp --dport 80 -m state --state NEW -m random --average 25
    -j DNAT --to-destination 192.168.0.5:80

iptables -A PREROUTING -i eth0 -p tcp --dport 80 -m state --state NEW -m random --average 25
    -j DNAT --to-destination 192.168.0.6:80

iptables -A PREROUTING -i eth0 -p tcp --dport 80 -m state --state NEW -m random --average 25
    -j DNAT --to-destination 192.168.0.7:80
iptables -A PREROUTING -i eth0 -p tcp --dport 80 -m state --state NEW -j DNAT --to-destination
192.168.0.8:80
```

L'utilisation du module `limit` et `connlimit`

- protection contre les attaques en déni de service, «DoS», «Denial of Service» : par exemple, limiter le nombre de connexions vers un serveur web (flooding) tout en assurant à tous les clients un accès illimité ;
- empêcher les attaques «brute force» pour deviner les mots de passe ;
- *etc.*

xterm
```
iptables -t filter -A FORWARD -m state --state NEW -p tcp -m multiport --dport http,https -o eth0
-i eth1 -m limit --limit 50/hour --limit-burst 5 -j ACCEPT
```

Explications :

▷ le module `multiport` permet de donner une liste de protocoles basés sur TCP ;

▷ le module `limit` :

◇ la règle n'autorise la connexion que lorsqu'un jeton d'autorisation est disponible ;

◇ `--limit 50/hour` : permet de limiter à l'obtention de 50 jetons par heure

◇ `--limit-burst` : définit la taille maximale du stock de jetons.

Le `limit` indique le rythme de récupération de jeton d'autorisation, alors que le `limit-burst` indique combien de jetons d'autorisation on peut conserver à la fois.

xterm
```
iptables -A INPUT -p tcp --syn --dport 22 -m limit --limit 3/min --limit-burst 1 -j ACCEPT
iptables -A INPUT -p tcp --syn --dport 22 -j DROP
```

Ce module permet de limiter le nombre de connexion réalisée en parallèle depuis le même hôte.

Ici, la règle 1 autorise le passage d'une demande de connexion au serveur SSH, une fois toutes les 20 secondes, et la règle 2 filtre toutes les autres tentatives.

Attention : utilise `--syn` et pas le module `state` ⟹*mieux*, le module `connlimit` qui limite par client :

xterm
```
# allow 2 ssh connections per client host
iptables -p tcp --syn --dport 22 -m connlimit --connlimit-above 2 -j REJECT
```

Mais c'est une règle négative (`REJECT`) qui doit être associé à une règle positive (`ACCEPT`) associée.

Le module `hashlimit`

- ▷ *permet de limiter le nombre paquets pouvant être traités par unité de temps ;*
- ▷ *plus efficace que le module `limit` pour la limitation par client grâce aux hashs ;*
- ▷ *règle positive (`ACCEPT`) compatible avec une policy `DROP` sans règle associée comme avec `connlimit` ;*

xterm
```
iptables -I INPUT -m hashlimit -m tcp -p tcp --dport 22 --hashlimit 1/min --hashlimit-mode srcip
--hashlimit-name ssh -m state --state NEW -j ACCEPT
```

Ici, on interdit plus d'une connexion par utilisateur et par minute afin d'éviter les tentatives d'attaques de type «bruteforce» sur le serveur ssh (découverte du mot de passe d'un utilisateur par essai des différentes combinaisons possibles).

Le module `length`

limiter les paquets à une certaine taille

xterm
```
iptables -A INPUT -p icmp --icmp-type echo-request -m length --length 86:0xffff -j DROP
```

Cela permet d'éviter les «ping of death» qui plantaient la pile TCP/IP de l'ordinateur le recevant (ici, ce sont les paquets de taille supérieure à 85 octets).

Le module `owner`

permet de tenir compte pour les paquets créés localement du processus et de son propriétaire :

xterm
```
iptables -A OUTPUT -p tcp --dport 3306 -d localhost -m owner ! --uid-owner www-data -j REJECT
```

Ici, le port 3306 est associé à MySQL, et on interdit les accès autres que ceux réalisés par l'utilisateur associé au serveur Apache.

Illustration du module `limit` et de l'option `burst`

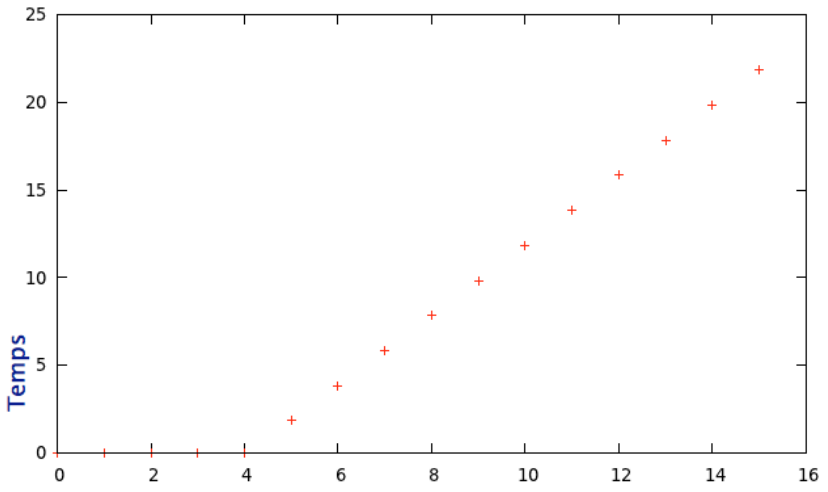
Les règles de firewall suivantes sont ajoutées :

xterm

```
iptables -t filter -A FORWARD -s 192.168.10.21 -d 192.168.11.50 -p udp -m limit --limit  30/m
--limit-burst 5 -j ACCEPT
iptables -t filter -A FORWARD -s 192.168.10.21 -d 192.168.11.50 -p udp -j DROP
```

Explications :

- on crée un jeton d'autorisation toutes les 2 sec. (30 par minute), et on autorise à en stocker au plus 5.
- on filtre tous les paquets qui arrivent et qui ne trouvent pas de jeton d'autorisation.
- on envoie 1000 paquets UDP avec un paquet toutes les 20ms ;
- seuls 16 paquets arrivent à traverser :



Le temps est indiqué en ordonnée et le numéro du paquet reçu est en abscisse.
Les 5 premiers paquets passent immédiatement, «burst», puis un seul paquet passe toute les 2 sec.

le module `recent`

permet de créer une liste dynamique d'adresses IP, et de comparer de différentes façons les paquets avec le contenu de cette liste :

xterm
```
iptables -A INPUT -i eth0 -m recent --name badguys --update --seconds 3600 -j DROP
iptables -A INPUT  -t filter -i eth0 -j DROP -m recent --set --name badguys
```

Explication

- La ligne 1, doit être mise en premier dans les règles du firewall : elle indique que si l'adresse IP du paquet fait partie de la liste alors on le supprime.
- la ligne 2, définie une liste des «*bad guys*», par exemple en la mettant à la fin de toutes les règles du firewall, c-à-d qu'aucune règle ACCEPT n'a pu être trouvée pour un firewall qui filtre par défaut (l'adresse IP de la machine est ajoutée à la liste).

Un classique concernant le protocole SSH, «Secure SHell» :

xterm
```
iptables -A INPUT -i eth0 -p tcp --syn --dport ssh -m recent --name ssh --set
iptables -A INPUT -i eth0 -p tcp --syn --dport ssh -m recent --name ssh --rcheck --seconds
30
--hitcount 2 -j DROP
```

Ici, on protège contre une tentative d'attaque de type «brute force».

Le module `set`

Il est possible :

a. de construire une liste avec la commande `ipset` :
   - ◇ de bloc ou d'@ IP ;
   - ◇ de ports ;
   - ◇ d'@MACs.

b. d'utiliser cette liste dans une règle de la commande `iptables`.

Pour construire l'ensemble :

xterm
```
$ ipset create indesirables nethash
$ ipset add indesirables 164.81.0.0/16
$ ipset add indesirables 192.168.0.0/16
```

Pour l'utiliser :

```
iptables -t filter -A INPUT -m set --match-set indesirables src -j DROP
```

L'intérêt réside dans le fait qu'il est moins coûteux de contrôler des datagrammes par rapport à un set plutôt que par rapport à une liste de règles.

Autres listes :

* `nethash` : une liste d'@IP ;
* `macipmap` : pour des @MAC ;
* `portmap` : pour des numéros de port ;

Pour bloquer les accès par pays : `http://www.ipdeny.com/ipblocks/`.

La journalisation ou «logging»

LOG : *permet de garder une trace du paquet.*

Il faut également répéter la même règle avec une target comme DROP pour jeter le paquet.

xterm
```
iptables -t nat -I PREROUTING -p tcp --destination-port 80 -j LOG
```

Cette information peut être consultée à l'aide de la commande `dmesg` *:*

xterm
```
[34985.564466] IN=eth0 OUT= MAC=00:0c:29:9d:ea:19:00:0c:29:22:ca:2f:08:00 SRC=192.168.127.128
DST=164.81.1.9 LEN=60 TOS=0x10 PREC=0x00 TTL=64 ID=28737 DF PROTO=TCP SPT=37925 DPT=80
WINDOW=5840 RES=0x00 SYN URGP=0
```

◇ `--log-prefix` : *permet d'ajouter un préfixe pour faciliter la recherche (avec la commande* `grep` *par exemple)*

xterm
```
iptables -A INPUT -p tcp -j LOG --log-prefix "INPUT packets"
```

Le module u32

Ce module permet de tester les valeurs d'octets particuliers du datagramme IP.

La syntaxe est `iptables -m u32 --u32 "Start&Mask=Range"` où :

⋆ il travaille sur des mots de 32 bits ;
⋆ «start» indique la position des octets à étudier à partir duquel on récupère les 32 bits (commence à zéro).
⋆ «Mask» permet d'appliquer un masque binaire à partir de cet octet.
⋆ il faut se positionner à la frontière de mots de 32bits (4 octets).

Exemple

▷ chercher si le drapeau MF est à 1 :

xterm
```
iptables -m u32 --u32 "3&0xE0=0x20"
```

▷ *etc.*

Une adresse intéressante `http://www.stearns.org/doc/iptables-u32.v0.1.7.html`

REJECT & TARPIT

- ▷ `REJECT` : *rejette le paquet comme DROP mais renvoi une erreur avec un paquet ICMP*
  - ◇ `--reject-with` : avec une valeur parmi :
    - ⋆ `icmp-net-unreachable`
    - ⋆ `icmp-host-unreachable`
    - ⋆ `icmp-port-unreachable`
    - ⋆ `icmp-proto-unreachable`
    - ⋆ `icmp-net-prohibited`
    - ⋆ `icmp-host-prohibited`
    - ⋆ `icmp-admin-prohibited`

- ▷ `TARPIT` : *permet de «capturer» une connexion TCP sans consommer de ressource et sans répondre aux demandes de terminaison.*
  C'est une protection efficace contre certain vers qui devront attendre longtemps avant de pouvoir établir une nouvelle connexion

```
xterm
iptables -A INPUT -p tcp -m tcp --dport 80 -j TARPIT
```

NFQUEUE : permet de faire passer les paquets dans un programme utilisateur

xterm

```
$sudo iptables -t filter -i bridge_dmz -A FORWARD -p tcp --sport 5001 -j NFQUEUE --queue-num 0
```

```
#!/usr/bin/python

import nfqueue, socket
from scapy.all import *

def traite_paquet(dummy, payload):
   data = payload.get_data()
   pkt = IP(data)
   # accepte
   #payload.set_verdict(nfqueue.NF_ACCEPT)
   # si modifie
   #payload.set_verdict_modified(nfqueue.NF_ACCEPT, str(pkt), len(pkt))
   # si rejete
   #payload.set_verdict(nfqueue.NF_DROP)

q = nfqueue.queue()
q.open()
q.unbind(socket.AF_INET)
q.bind(socket.AF_INET)
q.set_callback(traite_paquet)
q.create_queue(0)

try:
   q.try_run()
except KeyboardInterrupt, e:
   print "interrupted"

q.unbind(socket.AF_INET)
q.close()
```


En utilisant la règle suivante :

xterm
```
sudo iptables -t filter -i bridge_dmz -A FORWARD -p tcp --sport 5001 -j NFQUEUE --queue-num 0
```

On fait passer à travers `NFQUEUE` les segments TCP en provenance de l'outil `iperf` (outil permettant d'évaluer le débit d'une connexion TCP).

Ainsi, à l'aide de l'outil `Scapy`, on modifie la taille des paquets à l'intérieur des segments pour la rendre inférieure à une taille choisie (notée ici `throughput` ou débit).

```
if pkt[TCP].window > throughput :
   pkt[TCP].window = throughput
del pkt[TCP].chksum
payload.set_verdict_modified(nfqueue.NF_ACCEPT, str(pkt), len(pkt))
```

Ce qui donne pour une valeur de `throughput = 250` :

xterm
```
pef@solaris:~$ sudo iptables -nvL
Chain FORWARD (policy ACCEPT 17320 packets, 815K bytes)
 pkts bytes target     prot opt in     out     source               destination
 304K   16M NFQUEUE    tcp  --  bridge_dmz *       0.0.0.0/0       0.0.0.0/0           tcp
spt:5001 NFQUEUE num 0
```

Avec `iperf` :

xterm
```
root@INTERNAL_HOSTA:~/RESEAUX_I/FIREWALL# iperf -c 137.204.212.208
------------------------------------------------------------
Client connecting to 137.204.212.208, TCP port 5001
TCP window size: 16.0 KByte (default)
------------------------------------------------------------
[  3] local 137.204.212.11 port 45352 connected with 137.204.212.208 port 5001
[ ID] Interval       Transfer     Bandwidth
[  3]  0.0-10.5 sec  1.50 MBytes  1.20 Mbits/sec
```


Pour une valeur de `throughput = 1 000`:

xterm
```
root@INTERNAL_HOSTA:~/RESEAUX_I/FIREWALL# iperf -c 137.204.212.208
------------------------------------------------------------
Client connecting to 137.204.212.208, TCP port 5001
TCP window size: 16.0 KByte (default)
------------------------------------------------------------
[  3] local 137.204.212.11 port 45349 connected with 137.204.212.208 port 5001
[ ID] Interval       Transfer     Bandwidth
[  3]  0.0-10.1 sec  5.62 MBytes  4.65 Mbits/sec
```

Pour une valeur de `throughput = 5 000`:

xterm
```
root@INTERNAL_HOSTA:~/RESEAUX_I/FIREWALL# iperf -c 137.204.212.208
------------------------------------------------------------
Client connecting to 137.204.212.208, TCP port 5001
TCP window size: 16.0 KByte (default)
------------------------------------------------------------
[  3] local 137.204.212.11 port 45348 connected with 137.204.212.208 port 5001
[ ID] Interval       Transfer     Bandwidth
[  3]  0.0-10.0 sec  16.2 MBytes  13.6 Mbits/sec
```

Pour une valeur de `throughput = 50 000`:

xterm
```
root@INTERNAL_HOSTA:~/RESEAUX_I/FIREWALL# iperf -c 137.204.212.208
------------------------------------------------------------
Client connecting to 137.204.212.208, TCP port 5001
TCP window size: 16.0 KByte (default)
------------------------------------------------------------
[  3] local 137.204.212.11 port 45347 connected with 137.204.212.208 port 5001
[ ID] Interval       Transfer     Bandwidth
[  3]  0.0-10.0 sec  16.5 MBytes  13.8 Mbits/sec
```


En attendant Ipv6…

Pour IPv6 : *On bloquera tout trafic IPv6 jusqu'à nouvel ordre !*

```
# configuration
PUBIF="eth0"
# nettoyage des tables
ip6tables -F
ip6tables -X
ip6tables -t mangle -F
ip6tables -t mangle -X

# on autorise sur l'interface loopback
ip6tables -A INPUT -i lo -j ACCEPT
ip6tables -A OUTPUT -o lo -j ACCEPT

# on met en place la policy
ip6tables -P INPUT DROP
ip6tables -P OUTPUT DROP
ip6tables -P FORWARD DROP

# on autorise la poursuite des communications autorisées à s'établir
ip6tables -A INPUT -i $PUBIF -m state --state ESTABLISHED,RELATED -j ACCEPT

# on autorise le trafic sortant
ip6tables -A OUTPUT -o $PUBIF -m state --state NEW,ESTABLISHED,RELATED -j ACCEPT

# on autorise l'utilisation du ping en IPv6 (Neighbour Discovery)
ip6tables -A INPUT -i $PUBIF -p ipv6-icmp -j ACCEPT
ip6tables -A OUTPUT -o $PUBIF -p ipv6-icmp -j ACCEPT
```

```
#!/bin/sh

# Désactive la fonction de routage automatique du système
sysctl net.ipv4.ip_forward = 0

# Désactive le traitement des paquets ICMP echo-request envoyés en broadcast ou multicast
# Utile pour éviter les Smurf Attacks
sysctl net.ipv4.icmp_echo_ignore_broadcasts = 1

# Désactive le traitement des paquets routés par la source
# où le chemin de routage est explicitement donné
sysctl net.ipv4.conf.all.accept_source_route = 0

# Active la protection de type Syn cookie et protège contre les Syn Flood
sysctl net.ipv4.tcp_syncookies = 1

# Désactive le traitement des paquets ICMP de redéfinition de route qui détourne le trafic
sysctl net.ipv4.conf.all.accept_redirects = 0

# Ne pas autoriser l'envoi des paquets ICMP de redéfinition de route
sysctl net.ipv4.conf.all.send_redirects = 0

# Active la protection contre les IP Spoofing
# attention à la contradiction avec la fonction de routage...
sysctl net.ipv4.conf.default.rp_filter = 1
sysctl net.ipv4.conf.all.rp_filter = 1

# Journalise les paquets avec des adresses impossibles
net.ipv4.conf.all.log_martians = 1
net.ipv4.conf.default.log_martians = 1
```

Explications sur le transparent suivant ⟹

Explications

- la ligne «25» permet de se protéger de paquets qui ne pourraient pas arriver par l'interface où ils arrivent…
  C-à-d, par exemple, un paquet arrivant par l'interface `eth0` mais avec une adresse qui ne correspondrait qu'à l'interface «eth1» doit être rejeté.
  Cette protection est mise en œuvre par les FAIs ce qui empêche d'envoyer des datagrammes avec une adresse source différente de celle qu'il fournit à son client.

- la ligne «29» permet d'enregistrer les paquets provenant d'adresses impossibles : adresses de groupe, de classe E réservée, adresse locale, de diffusion, *etc.*
  C'est pour cela qu'on les appelle des paquets «martiens»…

- on peut également, dans la cas où le serveur sert de routeur, obliger le serveur à effectuer la défragmentation avant de servir de relais avec les options :
  ```
  sysctl net.ipv4.conf.all.ip_always_defrag = 1
  sysctl net.ipv4.conf.default.ip_always_defrag = 1
  ```

- On peut également bloquer les accès extérieurs à la base de données MySQL :
  ```
  xterm
  iptables -A INPUT -p tcp --dport 3306 -j DROP
  ```
  On pourrait également utiliser le module «`owner`» de NetFilter pour n'autoriser que l'utilisateur «Web» du système.

Les besoins en QoS

- utilisation **grandissante** de protocoles «temps-réels» : appels téléphoniques, IPTv :
  - ◇ problèmes de délais et de pertes de paquets plus **visibles** pour ces protocoles :
    - ⋆ pertes dues à la congestion ou à des corruptions de données ;
    - ⋆ variations dans les délais, *jitter*, de remise des paquets (dégradation du son et/ou de la vidéo) ;
    - ⋆ mélanges des paquets provoquant la suppression de ces paquets et encore plus de délai.

- utilisation de **réseaux «sans-fil»** : *ce sont dans ces réseaux que les manques sont les plus importants*

  Les réseaux «filaires»
  - ◇ gèrent des **milliers de flux** : pas raisonnable de vouloir tous les contrôler ;
  - ◇ sont-ils le «goulot d'étranglement», *bottleneck* ? ***non***
  - ◇ **facile** d'augmenter la capacité ;
  - ◇ temps pour organiser le trafic : $\sim 1\,\mu s$

  Les réseaux «sans-fil», «*wireless*»
  - ◇ gèrent des **dizaines de flux** : c'est possible de les contrôler ;
  - ◇ le réseau **est** le goulot d'étranglement !
  - ◇ **difficile** d'augmenter la capacité ;
  - ◇ temps pour organiser le trafic : $\sim 1ms$
    $\Longrightarrow$*Plus de temps pour prendre une décision !*

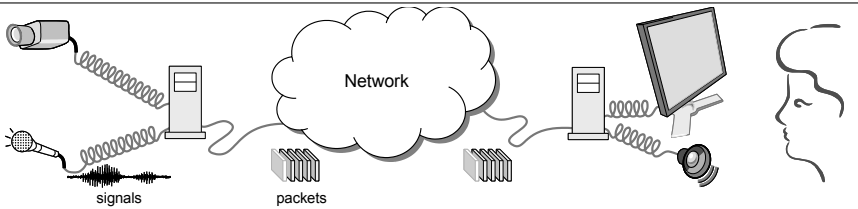
- garantir une **répartition équitable** de la capacité, «*fair use*» :
  - ◇ «*Network Neutrality*» : un FAI ne doit pas bloquer ou dégrader le trafic Internet de ses concurrents dans le but d'accélérer le sien.
  - ◇ **éviter la sur-consommation** des utilisateurs qui téléchargent le plus : vidéo et p2p.
    Dans ce cas là, il est nécessaire de faire du «deep packet inspection» pour analyser le contenu des datagrammes.

QoS : plan de la section

Définition Wikipedia

La qualité de service (QdS) ou Quality of service (QoS) est la capacité à véhiculer dans de bonnes conditions un type de trafic donné, en termes de disponibilité, débit, délais de transmission, gigue, taux de perte de paquets…

- Présentation du trafic «temps réél» avec RTP et comment corriger le «jitter» ;
- La proposition «*Integrated Services*» : c'est le réseau d'interconnexion en mode «circuit-virtuel» simulé ou non, qui assure la QoS :
  - **réservation de ressource** sur les routeurs ;
  - mémorisation de chemins dans chaque routeur ;
  - le paquet **traverse le routeur au plus vite** ;
- La proposition «*Differentiated Services*» : c'est le routeur qui favorise l'envoi de tel ou tel paquet :
  - les paquets sont **classifiés** ;
  - les paquets sont mis dans des **files d'attente**, des «*queues*» ;
  - chaque routeur doit gérer des files d'attentes ;
  - le contenu des files d'attente est envoyé sur le réseau suivant la **hiérarchie définie** entre ces queues.



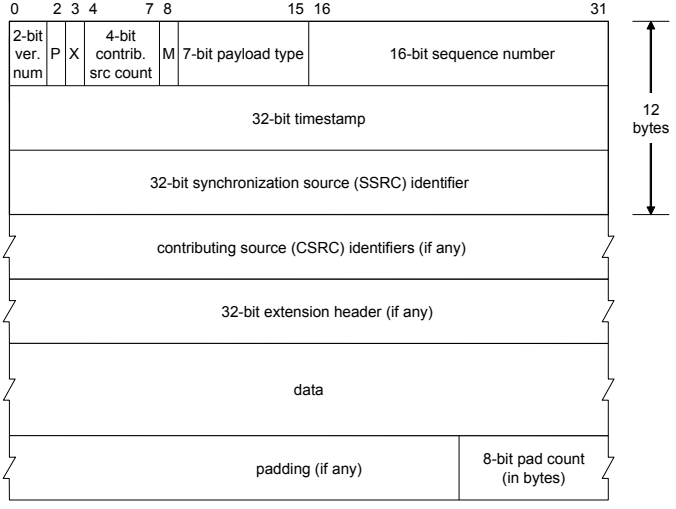
Les débits :

- ▷ constants, **CBR**, «*Constant Bit Rate*» : transmettre la voix ou une vidéo par conversion en un **flux binaire** avec un débit constant :
  - ◇ MPEG-1 : standard de compression à flux à débit constant. Le débit du flux dépend des paramètres sélectionnés pour l'algorithme de compression, tels que la dimension écran de la vidéo, le nombre d'images par seconde et la qualité de la «quantization», c-à-d la quantité d'information conservée en numérique par rapport à la version originale.
    MPEG-1 de mauvaise qualité à $1,15Mbps$ (en Méga-bits par seconde) et de bonne qualité à $3Mbps$.
  - ◇ pour un signal vocal, on va d'un débit de $4kbps$ pour une très forte compression à un débit de $64Kbps$ (un débit de $8Kbps$ à $1,3Mbps$ proche de la qualité d'un CD audio).
- ▷ variables, **VBR**, «*Variable Bit Rate*» : le débit peut varier pour s'adapter au mieux au contenu à transmettre.
  - ◇ MPEG-2 : autorise le VBR : le débit est plus important quand il y a des scènes du film compressé qui bougent rapidement, que lorsque les scènes bougent peu.
    Les transmissions satellites et la TNT SD (MPEG-4 pour la HD).

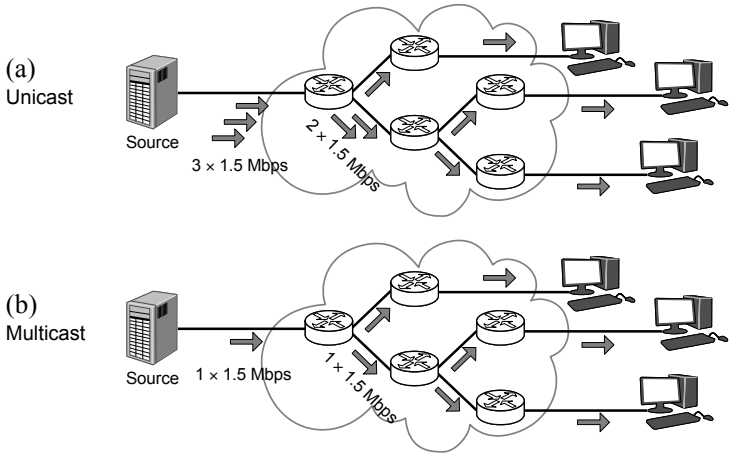
D'autres normes :

- ◦ pour la voix :
  - ◇ G.711 à $64Kbps$;
  - ◇ G.729 à $8Kbps$;
- ◦ pour la vidéo :
  - ◇ H.263 à $64Kbps$, c-à-d $8\,000\ octets$ par seconde ;
  - ◇ H.264 à $10Mbps$, c-à-d $1\,250\,000\ octets$ par seconde pour une vidéo de résolution 720×480 à 30 img/s ;
  - ◇ MPEG-2 à $19,2Mbps$, ou en VBR de $6Mbps$ en moyenne, à $24Mbps$ au maximum ;
  - ◇ MPEG-4 : peut utiliser le H.264.

- ▷ protocole «fourre-tout» qui permet le transport de différents types de flux, il est facilement adaptable ;
- ▷ basé sur UDP avec un numéro de port recommandé par l'IETF de $6970 < port < 6999$.
  Choix du numéro de port : par le protocole d'établissement de session comme SIP, «Session Initiation Protocol»*, ou H245 (utilisés pour la VoIP).*



- ⋆ `P` : padding ;
- ⋆ `X` : indique la présence d'un en-tête d'extension définie par l'application (rarement utilisé : les données contiennent les en-têtes si nécessaire) ;
- ⋆ `contributing source` : indique le nombre d'identifiants de source si plusieurs incluses dans l'en-tête ;
- ⋆ `M` : indicateur si les données sont particulières (par ex. des «frames boundaries») ;
- ⋆ `Payload type` : indique le format du contenu (G722, G711 *etc.*) ;
- ⋆ `Sequence number` : permet au récepteur de supprimer le «jitter», de détecter la perte de paquet et de les réordonner ; incrémenté de 1 à chaque paquet ; valeur de départ choisie aléatoirement *(protection contre les attaques par injection sur des données de paquet chiffrées)*.
- ⋆ `timestamp` : combiné au numéro de séquence pour détecter les trous, «*gap*», dans la transmission ; permet de réordonner des paquets provenant de différentes sources ; dépend de l'horloge de l'émetteur et de l'intervalle d'évolution de cette horloge : plusieurs paquets peuvent avoir le même timestamp.
- ⋆ `SSRC`& `CSRC` : permet d'identifier la ou les sources des flux (les identifiants sont aléatoires) qui sont «mixés» au niveau du récepteur.



Pour un même flux de $1,5Mbps$, reçu simultanément par trois récepteurs différents :

- «unicast» : gaspillage de capacité :
  $3 * 1,5 = 4,5Mbps$ (a) ;
- «multicast» : mutualisation :
  $1,5Mbps$ seulement (b) ;

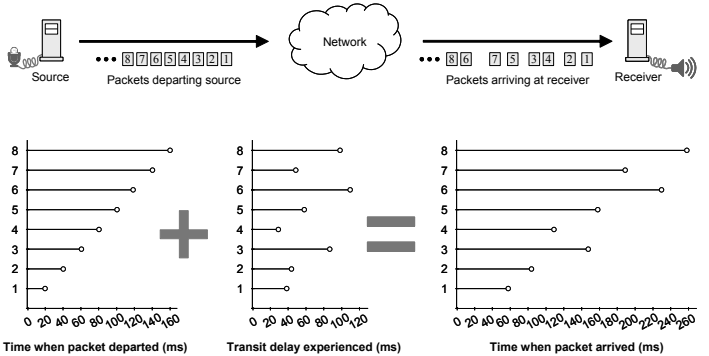
En multicast, les routeurs relaient les datagrammes envoyés sur une adresse de groupe `224.x.y.z` de classe D :

▷ chaque récepteur indique qu'il veut entrer/sortir du groupe, en envoyant :
  ◇ un message au format IGMP, «*Internet Group Management Protocol*», avec l'entrée/sortie et l'@IP du groupe ;
  ◇ encapsulé dans un datagramme :
    ◇ protocole : `2` ;
    ◇ options : `RA`, «routeur alert» : indique au routeur de traiter le paquet même s'il ne lui est pas adressé ;
    ◇ TTL de 1 ;
    ◇ transmis à destination du groupe associé à IGMP : `224.0.0.22` ;

▷ le routeur reçoit le message IGMP :
  ◇ si c'est une entrée pour le groupe `224.x.y.z`, il relais les paquets qu'il reçoit d'une entrée vers le segment où il a reçu le message ;
  ◇ si c'est une sortie, il arrête le relais.

▷ pour optimiser la diffusion et éviter les boucle : un protocole de «spanning tree», appelé CBT, «*Core Based Tree*».

Rappel : le Jitter ou «gigue» correspond à la variation du temps de transmission d'un paquet.



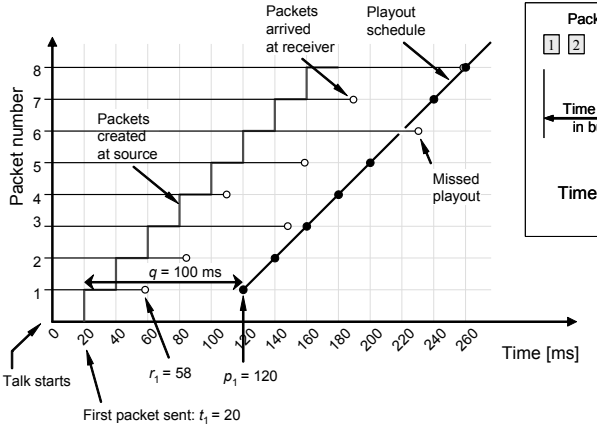
Les délais de transmission de chaque paquet induisent des retards, voire des ré-ordonnancements des paquets sur le récepteur.

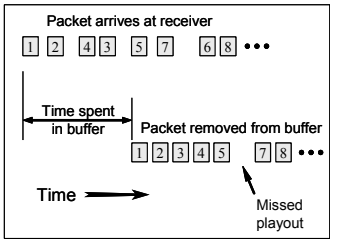
Si les délais sont très variables, on parle de fort jitter.

Un fort jitter est mauvais pour les transmissions «temps-réel».

Pour annuler le jitter :

- t_1 le temps de départ du paquet ;
- d_1 le délai de transmission ;
- on mesure r_1, $r_1 = t_1 + d_1$, le temps d'arrivée du premier paquet ;
- on détermine p_1 le temps auquel le contenu du paquet est «joué» (audio/vidéo) ;
- on choisit q avec $p_i = t_i + q$, et $q > jitter$.





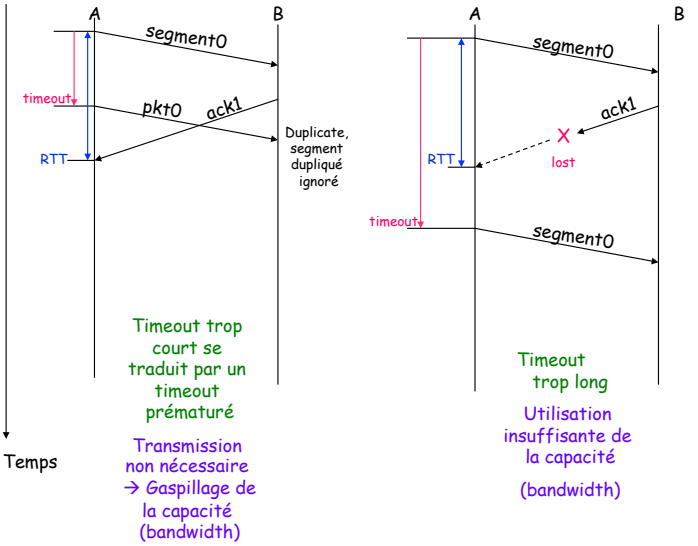
L'émetteur construit et envoie chaque paquet suivant un rythme régulier.

Si pour un paquet reçu i, $r_i > (t_i + q)$ alors le paquet est ignoré (6), sinon il est «bufferisé» et joué suivant le même rythme régulier mais avec le retard q.

TCP & QoS

Comment choisir la durée du «*timeout*» ?

- ⋆ suivant une valeur fixe ? *mais le RTT varie dynamiquement* ;
- ⋆ si **trop court** : risque de retransmettre des segments **pour rien** ;
- ⋆ si **trop long** : réaction **trop lente** aux pertes de segments.



Comment évaluer le RTT ?

- ▷ le «sampleRTT» ou «RTT échantillonné» : temps mesuré entre l'envoi d'un segment et la réception de son acquittement ;
- ▷ le «sampleRTT» varie : il faut «lisser» la mesure :
  - ◇ faire la moyenne, $\text{RTT}_{\text{moyen}}$ ou $EstimatedRTT$, sur plusieurs mesures et pas seulement utiliser la dernière : $EstimatedRTT = (1-\alpha) * EstimatedRTT + \alpha * SampleRTT$ avec $0 < \alpha < 1$
    - ⋆ α proche de 0 : insensible aux variations rapides
    - ⋆ α proche de 1 : très sensible aux variations rapides

La mesure du RTT

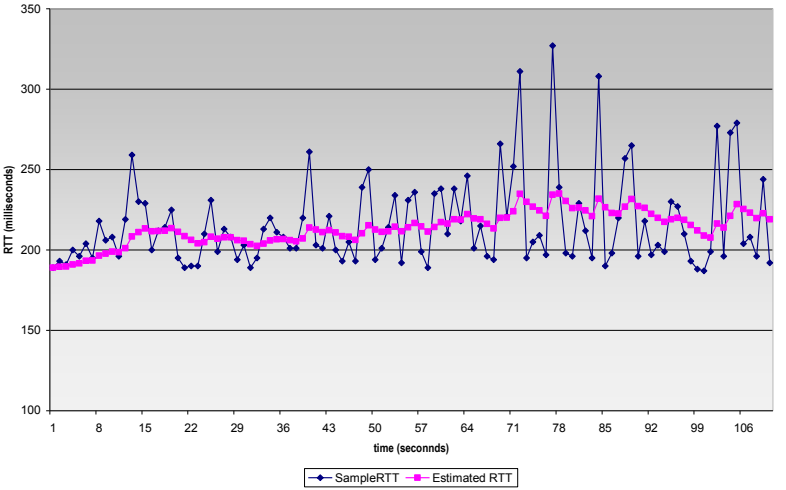
$$EstimatedRTT = (1 - \alpha) * EstimatedRTT + \alpha * SampleRTT$$

Exemple :

$EstimatedRTT = 250ms$
$SampleRTT = 70ms$
$\alpha = 0.125$

Calcul :

$EstimatedRTT = (1 - 0.125) * 250$
$+0.125 * 70 = 218.75 + 8.75 = 227.5ms$
$87,5\%$ de poids au $EstimatedRTT$
$12,5\%$ de poids au $SampleRTT$



Ce calcul permet de

- «lisser» les mesures ;
- obtenir un graphe similaire à celui donné à droite.

Calcul du temporisateur, timeout

- calcule la valeur du temporisateur en fonction de $EstimatedRTT$ plus une marge de protection :
  - forte variation de $EstimatedRTT \Longrightarrow$ marge de protection plus grande ;
  - mesurer la déviation entre le $SampleRTT$ et le $EstimatedRTT$: le $DevRTT$:
    $DevRTT = (1 - \beta) * DevRTT + \beta * | SampleRTT - EstimatedRTT |$, où $\beta = 0,25$

- Ce qui donne au final : $TimeoutInterval = EstimatedRTT + 4 * DevRTT$

Calcul du RTO, «*Retransmission Timeout*», RFC 2988

- **Chaque fois qu'un segment** contenant des données est **envoyé** (y compris pour une retransmission), un **timer** doit être déclenché d'une durée égale au RTO calculé ;
- lorsqu'un **ACK** est reçu demandant l'envoi de nouvelles données, le **timer est arrêté** (la réception du segment est confirmée) :
  - redémarrer le timer sur la durée du RTO pour l'envoi du **prochain segment** ;
- lorsqu'un *timer* **expire** :
  - retransmettre les premiers segments qui n'ont pas été acquittés par le récepteur ;
  - augmenter le RTO : $RTO = RTO * 2$, on appelle cette opération : *timer backoff* ;
  - redémarrer le timer avec la nouvelle valeur de RTO pour les nouveaux envois.

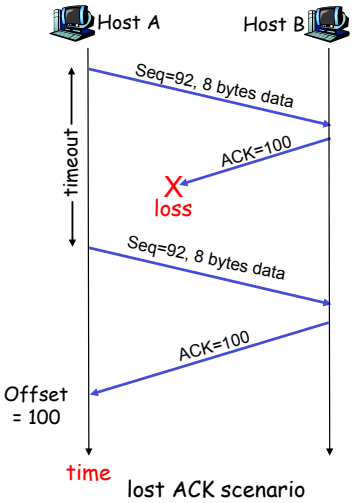
En résumé :

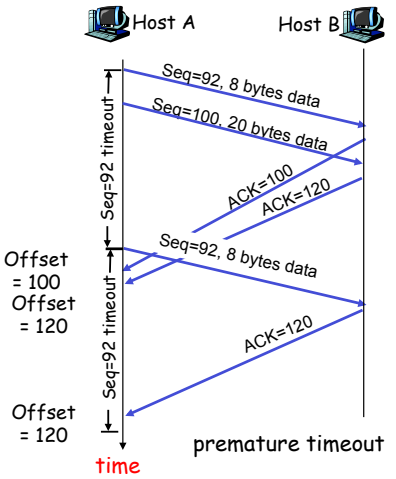
▷ quand un **ACK est reçu**, on calcul le RTO avec $TimeoutInterval = EstimatedRTT + 4 * DevRTT$

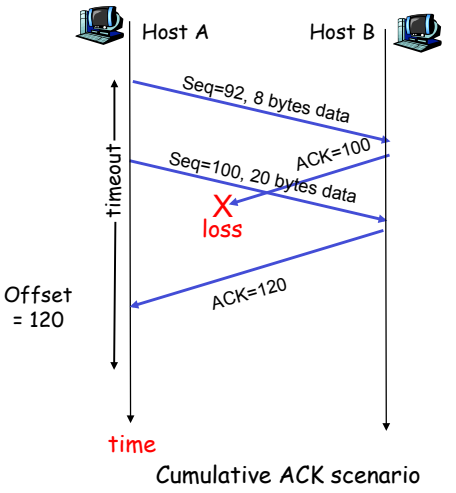
▷ quand un **timeout se produit**, on modifie le RTO : $RTO = RTO * 2$

- lorsque **toutes les données envoyées** ont été acquittées, il faut **arrêter** le timer ;

- Pour le calcul des $SampleRTT$:
  - Constat : en cas de retransmission d'un segment, l'émetteur **ne peut savoir** si l'acquittement s'adresse au segment initial ou retransmis (acquittements ambigüs)
    ⇒ RTT ne peut donc être calculé correctement
    ⇒ TCP ne doit pas mettre à jour le RTT pour les segments retransmis
    Sauf si l'option Timestamp a été utilisée dans le segment.







- ⋆ «*lost ack scenario*» : le ACK se perd et le RTO expire ;
- ⋆ «*premature timeout*» : le RTO expire avant la réception des ACKs ;
- ⋆ «*cumulative ack scenario*» : un ACK se perd mais il devient inutile car un nouveau ACK est envoyé et englobe celui perdu.

Implémentation des timers de retransmission

- ◇ *En théorie, il est plus simple de gérer un timer de retransmission par segment envoyé.*
- ◇ *En pratique, la plupart des implémentations utilise un* ***seul timer*** *par connexion TCP (RFC 2988).*

Rapport avec le contrôle de flux ? Aucun !

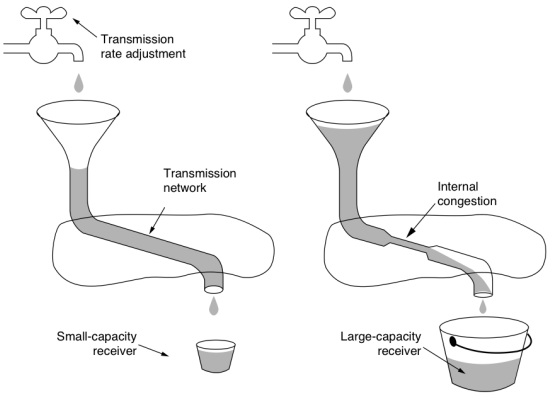
Le **contrôle de flux** ne gère que la capacité du récepteur à traiter les segments reçus.
Ce contrôle de flux utilise un mécanisme de fenêtre.

Problème de la détection de l'occupation réseau

- ⋆ le réseau peut être en situation de **congestion indépendamment** de la situation de l'émetteur et du récepteur ;
- ⋆ le récepteur et l'émetteur doivent **détecter** cette situation :
  - ◇ la **perte d'un paquet** due à une erreur matérielle est **rare** ;
  ⟹ elle doit être due à une congestion du réseau ⇒ *connaissance* !
  - ◇ les *timers* servent à **réémettre un paquet perdu** ⇒ un timer expiré indique une **congestion** !

Image du seau d'eau :

le contrôle de flux sert à adapter le débit du robinet à la taille du seau, mais la congestion fait déborder l'entonnoir (perte de paquets) même si la taille du seau est grande !



Comment gérer une situation de congestion ?

- ▷ découvrir la **congestion** : *prendre en compte les pertes de segments*
  - ◇ situation de **congestion faible** : réception de 3 ACKs dupliqués (des paquets arrivent encore à circuler) ;
  - ◇ situation de **congestion forte** : expiration d'un timer, «timeout»

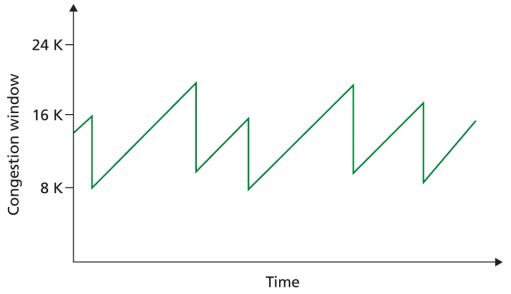
- ▷ limiter le **débit** :
  - ◇ utiliser une nouvelle fenêtre d'*autorisation d'émission* : la fenêtre de **congestion**, «congestion window» ;
  - ◇ la taille de cette fenêtre : $CongWin = w * MSS$ octets, où w varie en fonction de l'état **perçu** du réseau.
  - ◇ fonctionnement :
    - ⋆ La fenêtre réception, $RcvWindow$, & la fenêtre congestion déterminent la quantité d'octets à émettre ;
    - ⋆ L'émetteur envoie au plus une taille t de données, telle que $t < min(CongWin, RcvWindow)$

    Exemple :
    - ⋆ Le récepteur indique «Autorisation d'envoyer 8ko», $RcvWindow = 8ko$
      - ▷ Si l'émetteur dispose d'une $CongWin = 4ko$, il envoie au plus $4ko$ de données ;
      - ▷ Si l'émetteur dispose d'une $CongWin = 32ko$, il envoie au plus $8ko$ de données :

    *En contrôle de flux, c'est le **récepteur** qui contrôle, en contrôle de congestion, c'est l'**émetteur**.*

- ▷ choisir un **algorithme** pour faire évoluer la taille de la $CongWindow$:
  - ◇ AIMD, «*Additive-Increase, Multiplicative-Decrease*» ;
  - ◇ «Slow Start» ;
  - ◇ gestion des pertes de segments :
    - ⋆ TCP Tahoe ;
    - ⋆ TCP Reno ;
    - ⋆ *Autres* ;

AIMD, «*Additive-Increase, Multiplicative-Decrease*»



- ⋆ *incrémenter le débit de transmission, c-à-d la taille de la $CongWindow$, de manière linéaire jusqu'à une perte* ;
- ⋆ «*additive increase*» : augmenter la taille de $CongWindow$ de 1 MSS à chaque RTT, jusqu'à une perte ou que l'on atteigne la taille de la $RcvWindow$;
- ⋆ «*multiplicative decrease*» : diviser par deux la taille de la $CongWindow$ à chaque perte (parce que l'effet de la congestion est exponentiel) ;

On «probe» la capacité de la ligne de transmission et on obtient un graphe en dent de scie.

Le débit initial est faible au démarrage :
Exemple :

▷ À l'établissement de la connexion : $CongWindow = 1MSS$

▷ $MSS = 500$ octets, $RTT = 200ms$

▷ $débit = \frac{CongWindow}{RTT}$ octets/seconde, $débit\ initial = 21 * 500 = 10,5Ko$ par seconde.

*Le débit initial est **trop faible**, il faut changer la manière d'augmenter la taille de la fenêtre.*

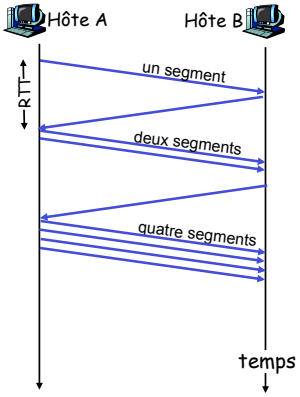
Nouvelle propostion : le «Slow Start» avec accélération rapide

Deux phases :

a. démarrer lentement, «*slow start*» et augmenter **exponentiellement** la $CongWindow$;
On parle de «slow-start» car on commence à une valeur de seulement 1 segment, sans tenir compte de la capacité réelle du réseau qui est, en général, bien supérieure.

b. éviter la congestion, «*congestion avoidance*» en **augmentant** la $CongWindow$ **linéairement** et en **prenant en compte les «signes»** anonciateurs d'une congestion réseau (disparition effective ou présumée de paquets).

Fonctionnement du mode «Slow Start»



- on envoie le **premier segment** :
  s'il est acquitté avant expiration du timer $\Longrightarrow CongWin = CongWin + 1MSS$;
- on envoie le **second segment** : s'il est acquitté, on incrémente la taille de 1, ce qui équivaut à doubler la taille de $CongWin$ à chaque RTT.
- on continue, ainsi, à doubler la taille de $CongWin$ à chaque RTT, jusqu'à :
  - avoir une perte de segment (on réagira comme en mode «*congestion avoidance*») ;
  - atteindre l'envoi de n segments, seuil appelé *ssthresh* ;
- on passe en mode «congestion avoidance» (on cherche à «éviter» la congestion).

Résumé : *aussi longtemps que l'émetteur reçoit des ACKs non dupliqués on continue à incrémenter la taille de la fenêtre de congestion jusqu'à la limite* ssthresh *et on passe en mode «congestion avoidance»*

Fonctionnement du mode «*congestion avoidance*»

L'idée de ce mode est de repasser à l'AIMD et de choisir un comportement lors de la perte d'un segment :

- Il existe **différentes méthodes** pour réagir à la perte d'un segment :
  - TCP Tahoe :
    - distinction entre situation de congestion faible ou forte (3 ACKs dupliqués ou timeout) ;
    - réinitialisation : $ssthresh$ =CongWindow$/2$ («*Slow Start threshold*») ;
    - réinitialisation : $CongWindow = 1MSS$;
    - on effectue un «*fast-retransmit*» : on renvoie le segment sans attendre la fin du RTO (lors de 3 ACKs dupliqués) ;
    - on recommence le mode «slow-start», c-à-d on augmente exponentiellement jusqu'à $ssthresh$ puis linéairement au-delà.

Fonctionnement du mode «*congestion avoidance*»

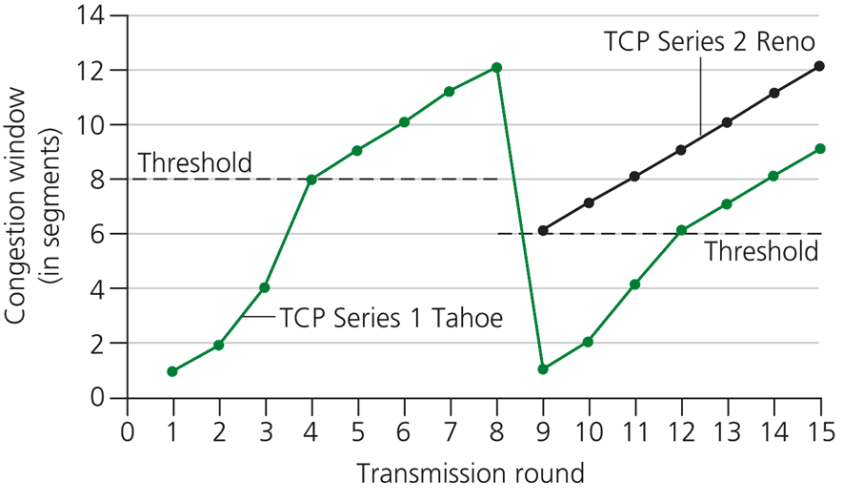
- Il existe différentes méthodes pour réagir à la perte d'un segment :
  - **TCP Reno** :
    - distinction entre congestion faible et de congestion forte :
      - **congestion faible** (3 ACKs dupliqués) :
        - $ssthresh = CongWindow/2$;
        - mode «fast-recovery» : $CongWindow = CongWindow/2$;
        - mode «fast-retransmit» : envoi immédiat du segment «perdu» sans attendre le RTO (3 ACKs dupliqués) ;
      - **congestion forte** (timeout) : *fonctionnement à la Tahoe*
        - $ssthresh = CongWindow/2$
        - $CongWindow = 1MSS$;
        - on effectue un «fast-retransmit» ;
        - on recommence le mode «slow-start» ;
  - **TCP Vegas** : détecte la congestion sur l'allongement progressif de la RTT de chaque segment ;
  - **New Reno** : adapte Reno à l'utilisation de l'option SACK.
  - *etc.*

Résumé :
- *quand* $CongWindow < ssthresh$ *augmentation exponentielle de la taille de* $CongWindow$ $(*2)$*;*
- *quand* $CongWindow \geq ssthresh$ *augmentation linéaire de la taille de* $CongWindow$ $(+1)$*.*

Les **modes** :

- «*fast-retransmit*» : l'émetteur étant persuadé que le segment demandé plusieurs fois dans les ACKs dupliquées est perdu, il le retransmet sans attendre l'expiration du timer («Retransmit TimeOut» ou RTO) ;
- «*fast-recovery*» : divise par deux la $CongWindow$ et attend l'acquittement **complet** de la fenêtre d'autorisation d'envoi, avant de repasser en mode «congestion avoidance» ;
- la différence entre Tahoe & Reno : l'ajout de la notion de «congestion faible» et du «fast-recovery».

Comportement de Tahoe *vs* Reno

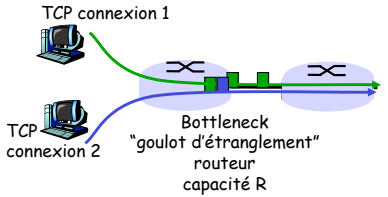


- ⋆ initialement, le $ssthresh = 8 * MSS$
- ⋆ le pic correspond à une perte de segment :
  - ◇ $CongWindow = 12$ au moment du pic ;
  - ◇ $ssthresh = CongWindow/2 = 6MSS$;
  - ◇ Tahoe repart en «slow start» avec $CongWindow = 1MSS$;
  - ◇ Reno utilise mieux la capacité du support : il repart de $CongWindow/2 = ssthresh$;

Et finalement le débit moyen d'une connexion TCP ?

Le débit en «dent de scie» passe de $\frac{(Cg_W * MSS)}{(2RTT)}$ à $\frac{(Cg_W * MSS)}{RTT}$, d'où : $throughput_{average} = \frac{.75 * Cg_W * MSS}{RTT}$

Est-ce que TCP est un protocole «équitable»

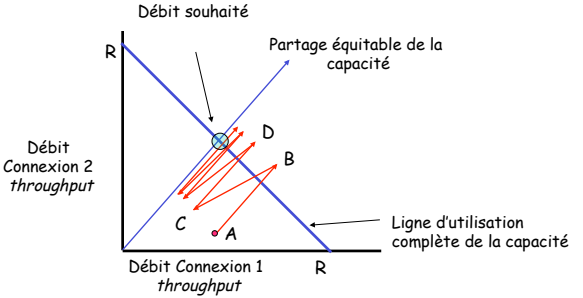


Objectif de **l'équité**, *fairness* :

- ▷ Si K connexions utilisent la même ligne de transmission de capacité R ;
- ▷ chaque connexion doit obtenir un débit de R/K.

On observe que les deux connexions vont se répartir la capacité de la ligne de transmission :

- ◇ en A : les deux connexions atteignent un débit inférieur à R, pas de pertes, la taille de $CongWin$ augmente de 1 à chaque RTT et la courbe évolue à $45°$;
- ◇ en B : le débit des connexions dépasse R et les deux connexions perdent des segments et reviennent en C ;
- ◇ en D : ça recommence…

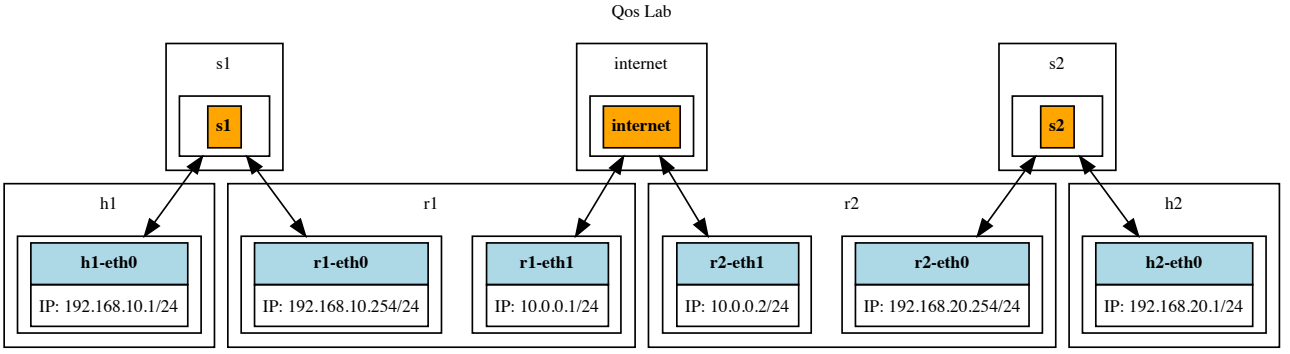


Mais la répartition se fait entre toutes les connexions TCP et non entre celles groupées par application : une application ayant deux connexions sera avantagée par rapport à une autre utilisant une seule connexion.

Équité dans UDP

- ▷ TCP possède un **mécanisme de contrôle de congestion** : il offre un traitement équitable en répartissant la capacité entre les différentes connexions :
- ▷ UDP **ne possède pas ce mécanisme** :
  - ◇ les applications multimédia utilise plutôt UDP (VoIP, «*Voice over IP*», Vidéo) :
    - ⋆ elles ne veulent pas voir leur **débit modifié** par le contrôle de congestion ;
    - ⋆ elles ne veulent pas le **surcoût de TCP** (gestion de fenêtres d'envoi et de réception, gestion d'erreurs) ;
  - ◇ la transmission par UDP est faite suivant un débit constant où **la perte de paquets est tolérée** ;
  - ◇ le trafic UDP peut entraîner une **congestion du réseau**.

Un graphe obtenu directement à partir du fichier de configuration



Trois **switches** :

- s1 : `192.168.10.0/24` ;
- s2 : `192.168.20.0/24` ;
- internet : `10.0.0.0/24` ;

Deux **hôtes** :

- h1 : h1-eth0 `192.168.10.1/24` ;
- h2 : h2-eth0 `192.168.20.1/24` ;

Deux **routeurs** :

- r1 :
  - r1-eth0 `192.168.10.254/24` ;
  - r1-eth1 `10.0.0.1/24`
- r2 :
  - r2-eth0 `192.168.20.254/24` ;
  - r2-eth1 `10.0.0.2/24`

```
#!/bin/bash
# créer les namespaces pour les hôtes
ip netns add h1
ip netns add h2
ip netns add r1
ip netns add r2

# créer le switch
ovs-vsctl add-br internet
ovs-vsctl add-br s1
ovs-vsctl add-br s2

# créer les liens
ip link add h1-eth0 type veth peer name s1-h1
ip link add h2-eth0 type veth peer name s2-h2

ip link add r1-eth0 type veth peer name s1-r1
ip link add r1-eth1 type veth peer name internet-r1

ip link add r2-eth0 type veth peer name s2-r2
ip link add r2-eth1 type veth peer name internet-r2

# accrocher les liens aux namespaces
ip link set h1-eth0 netns h1
ip link set h2-eth0 netns h2
ip link set r1-eth0 netns r1
ip link set r1-eth1 netns r1
ip link set r2-eth0 netns r2
ip link set r2-eth1 netns r2

# connecter les liens au switch
ovs-vsctl add-port s1 s1-h1
ovs-vsctl add-port s1 s1-r1
ovs-vsctl add-port s2 s2-h2
ovs-vsctl add-port s2 s2-r2
ovs-vsctl add-port internet internet-r1
ovs-vsctl add-port internet internet-r2

# activer les interfaces du namespace root
ip link set dev s1-h1 up
```

❶⇒Créer les différents «*netnamespaces*» ;

❷⇒Créer les différents switches : chaque switch correspond à un réseau indépendant ;

❸⇒Créer chaque lien avec deux extrémités ;

❹⇒accrocher une extrémité d'un des liens à un netnamespace ;

❺⇒accrocher l'autre extrémité du lien à un switch ;

❻⇒activer les interfaces du netnamespace «*root*» et des autres netnamespace ;

```
ip link set dev s2-h2 up
ip link set dev s1-r1 up
ip link set dev s2-r2 up
ip link set dev internet-r1 up
ip link set dev internet-r2 up

❶# activer les interfaces des namespaces h1 et h2
ip netns exec h1 ip link set dev h1-eth0 up
ip netns exec h2 ip link set dev h2-eth0 up

❷# activer les interfaces des namespaces r1 et r2
ip netns exec r1 ip link set dev r1-eth0 up
ip netns exec r1 ip link set dev r1-eth1 up
ip netns exec r2 ip link set dev r2-eth0 up
ip netns exec r2 ip link set dev r2-eth1 up

❸# configurer les réseaux sur s1, s2 et internet
ip netns exec h1 ip a add dev h1-eth0 192.168.10.1/24
ip netns exec r1 ip a add dev r1-eth0 192.168.10.254/24

ip netns exec h2 ip a add dev h2-eth0 192.168.20.1/24
ip netns exec r2 ip a add dev r2-eth0 192.168.20.254/24

ip netns exec r1 ip a add dev r1-eth1 10.0.0.1/24
ip netns exec r2 ip a add dev r2-eth1 10.0.0.2/24

❹# configurer les routes sur r1 et r2
ip netns exec r1 ip r add 192.168.20.0/24 via 10.0.0.2
ip netns exec r2 ip r add 192.168.10.0/24 via 10.0.0.1

❺# configurer la route par défaut sur h1 et h2
ip netns exec h1 ip r add default via 192.168.10.254
ip netns exec h2 ip r add default via 192.168.20.254

❻# activer le routage sur r1 et r2
ip netns exec r1 sudo sysctl net.ipv4.conf.all.forwarding=1
ip netns exec r2 sudo sysctl net.ipv4.conf.all.forwarding=1
```

❶⇒activer les interfaces des netnamespaces «*utilisateurs*» ;

❷⇒activer les interfaces des netnamespaces «*routeurs*» ;

❸⇒configurer les adresses des interfaces par rapport aux réseaux auxquels elles sont connectées ;

❹⇒configurer les routes sur les routeurs ;

❺⇒configurer les routes par défaut sur les hôtes ;

❻⇒activer la fonction de routage dans les netnamespaces routeurs.

Évaluation de la vitesse de transfert avec l'outil «`iperf`»

▷ on démarre la partie **serveur** d'«`iperf`» sur «*h2*» :

xterm
```
pef@cube:~/QOS_LAB$ sudo ip netns exec h2 iperf -s
------------------------------------------------------------
Server listening on TCP port 5001
TCP window size: 85.3 KByte (default)
------------------------------------------------------------
[  4] local 192.168.20.1 port 5001 connected with 192.168.10.1 port 36800
[ ID] Interval       Transfer     Bandwidth
[  4]  0.0-10.0 sec  2.65 GBytes  2.28 Gbits/sec
```

▷ on démarre la partie **client** d'«`iperf`» sur «*h1*» :

xterm
```
pef@cube:~/QOS_LAB$ sudo ip netns exec h1 iperf -c 192.168.20.1
------------------------------------------------------------
Client connecting to 192.168.20.1, TCP port 5001
TCP window size: 85.0 KByte (default)
------------------------------------------------------------
[  3] local 192.168.10.1 port 36800 connected with 192.168.20.1 port 5001
[ ID] Interval       Transfer     Bandwidth
[  3]  0.0-10.0 sec  2.65 GBytes  2.28 Gbits/sec
```

▷ on évalue le **RTT** de «*h1*» vers «*h2*» :

xterm
```
pef@cube:~/QOS_LAB$ sudo ip netns exec h1 ping -c 1 192.168.20.1
PING 192.168.20.1 (192.168.20.1) 56(84) bytes of data.
64 bytes from 192.168.20.1: icmp_seq=1 ttl=62 time=0.320 ms
--- 192.168.20.1 ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 0.320/0.320/0.320/0.000 ms
```


Exemple de réseau : évaluation de la QoS, limitation du débit de 100Mbps

On limite le débit avec «`tc`», «*traffic control*» à 100Mbits

▷ on applique la limitation de débit :

xterm
```
pef@cube:~/QOS_LAB$ sudo tc qdisc add dev internet-r2 root tbf rate 100Mbit latency 50ms burst
1M
pef@cube:~/QOS_LAB$ sudo tc qdisc show dev internet-r2
qdisc tbf 8003: root refcnt 2 rate 100Mbit burst 1024Kb lat 50.0ms
```

▷ on démarre la partie **serveur** d'«`iperf`» sur «*h2*» :

xterm
```
pef@cube:~/QOS_LAB$ sudo ip netns exec h2 iperf -s
------------------------------------------------------------
Server listening on TCP port 5001
TCP window size: 85.3 KByte (default)
------------------------------------------------------------
[  6] local 192.168.20.1 port 5001 connected with 192.168.10.1 port 36790
[  6]  0.0-141.1 sec   118 MBytes  7.01 Mbits/sec
```

▷ on démarre la partie **client** d'«`iperf`» sur «*h1*» :

xterm
```
pef@cube:~/QOS_LAB$ sudo ip netns exec h1 iperf -c 192.168.20.1
------------------------------------------------------------
Client connecting to 192.168.20.1, TCP port 5001
TCP window size: 85.0 KByte (default)
------------------------------------------------------------
[  3] local 192.168.10.1 port 36790 connected with 192.168.20.1 port 5001
[ ID] Interval       Transfer     Bandwidth
[  3]  0.0-10.0 sec   118 MBytes  98.6 Mbits/sec
```

▷ on évalue le **RTT** de «*h1*» vers «*h2*» :

xterm
```
pef@cube:~/QOS_LAB$ sudo ip netns exec h1 ping -c 1 192.168.20.1
PING 192.168.20.1 (192.168.20.1) 56(84) bytes of data.
64 bytes from 192.168.20.1: icmp_seq=1 ttl=62 time=0.415 ms
--- 192.168.20.1 ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 0.415/0.415/0.415/0.000 ms
```

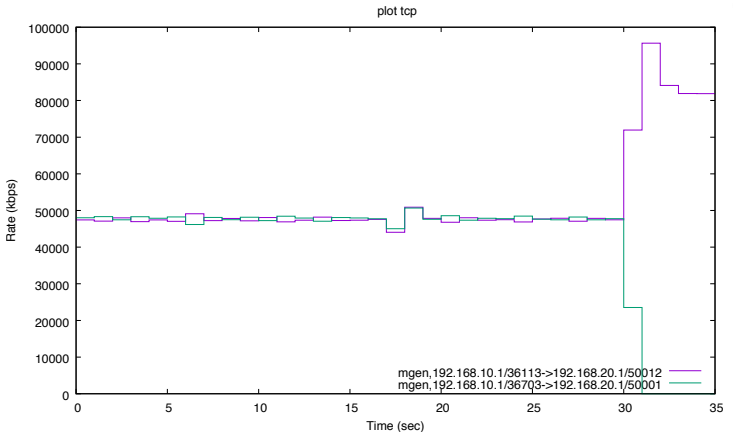
xterm
```
pef@cube:~/QOS_LAB$ cat source_80mbps_tcp.mgn
0.0 ON 1 TCP DST 192.168.20.1/5000 PERIODIC [ 10000.0 1024 ]
0.0 ON 2 TCP DST 192.168.20.1/5001 PERIODIC [ 10000.0 1024 ]
30.0 OFF 1
35.0 OFF 2
```

xterm
```
pef@cube:~/QOS_LAB$ cat destination_tcp.mgn
0.0 LISTEN TCP 5001
0.0 LISTEN TCP 5000
```

xterm
```
pef@cube:~/QOS_LAB$ sudo ip netns exec h2 mgen input destination_tcp.mgn output stats_tcp.drc
```

xterm
```
pef@cube:~/QOS_LAB$ sudo ip netns exec h1 mgen input source_80mbps_tcp.mgn
```

xterm
```
pef@cube:~/QOS_LAB$ trpr mgen input stats_tcp.drc auto X output plot_tcp
```



Comparaison des débits des deux connexions TCP :

- ▷ de 0 à 30s : elle se répartissent le débit de manière équitable ;
- ▷ de 30 à 35s :
  - ◇ celle en vert s'arrête ;
  - ◇ celle en violet reprend la totalité du débit possible.

On peut s'assurer de la perte des paquets TCP en «*sniffant*» les segments dupliqués avec `tshark`

xterm

```
pef@cube:~/QOS_LAB$ sudo ip netns exec h2 tshark -l -i h2-eth0 -Y 'tcp.analysis.duplicate_ack'
tcp and host 192.168.10.1 > capture_tshark_tcp_ack

Capturing on 'h2-eth0'
6044 ^C
pef@cube:~/QOS_LAB$ more capture_tshark_tcp_ack
39063 2.720524719 192.168.20.1 → 192.168.10.1 TCP 78 [TCP Dup ACK 39061#1] 5000 → 41601 [ACK]
Seq=1 Ack=14954625 Win=1297408 Len=0 TSva
l=4067766353 TSecr=267743691 SLE=14957521 SRE=14958969
39065 2.720646298 192.168.20.1 → 192.168.10.1 TCP 78 [TCP Dup ACK 39061#2] 5000 → 41601 [ACK]
Seq=1 Ack=14954625 Win=1297408 Len=0 TSva
l=4067766353 TSecr=267743691 SLE=14957521 SRE=14960417
39067 2.720764780 192.168.20.1 → 192.168.10.1 TCP 78 [TCP Dup ACK 39061#3] 5000 → 41601 [ACK]
Seq=1 Ack=14954625 Win=1297408 Len=0 TSva
l=4067766353 TSecr=267743691 SLE=14957521 SRE=14961865
39069 2.720886767 192.168.20.1 → 192.168.10.1 TCP 78 [TCP Dup ACK 39061#4] 5000 → 41601 [ACK]
Seq=1 Ack=14954625 Win=1297408 Len=0 TSva
l=4067766353 TSecr=267743691 SLE=14957521 SRE=14963313
39071 2.721009035 192.168.20.1 → 192.168.10.1 TCP 78 [TCP Dup ACK 39061#5] 5000 → 41601 [ACK]
Seq=1 Ack=14954625 Win=1297408 Len=0 TSva
l=4067766353 TSecr=267743691 SLE=14957521 SRE=14964761
39073 2.721131465 192.168.20.1 → 192.168.10.1 TCP 78 [TCP Dup ACK 39061#6] 5000 → 41601 [ACK]
Seq=1 Ack=14954625 Win=1297408 Len=0 TSva
l=4067766353 TSecr=267743691 SLE=14957521 SRE=14966209
39075 2.721249813 192.168.20.1 → 192.168.10.1 TCP 78 [TCP Dup ACK 39061#7] 5000 → 41601 [ACK]
Seq=1 Ack=14954625 Win=1297408 Len=0 TSva
l=4067766353 TSecr=267743691 SLE=14957521 SRE=14967657
39077 2.721370607 192.168.20.1 → 192.168.10.1 TCP 78 [TCP Dup ACK 39061#8] 5000 → 41601 [ACK]
Seq=1 Ack=14954625 Win=1297408 Len=0 TSva
l=4067766353 TSecr=267743691 SLE=14957521 SRE=14969105
39079 2.721494460 192.168.20.1 → 192.168.10.1 TCP 78 [TCP Dup ACK 39061#9] 5000 → 41601 [ACK]
Seq=1 Ack=14954625 Win=1297408 Len=0 TSva
l=4067766353 TSecr=267743691 SLE=14957521 SRE=14970553
...
```

Ici, un segment est dupliqué de nombreuses fois...

Exemple de réseau : évaluation de la QoS – UDP

xterm

```
pef@cube:~/QOS_LAB$ cat source_80mbps_udp.mgn
0.0 ON 1 UDP DST 192.168.20.1/5000 PERIODIC [ 10000.0 1024 ]
0.0 ON 2 UDP DST 192.168.20.1/5001 PERIODIC [ 10000.0 1024 ]
30.0 OFF 1
35.0 OFF 2
```

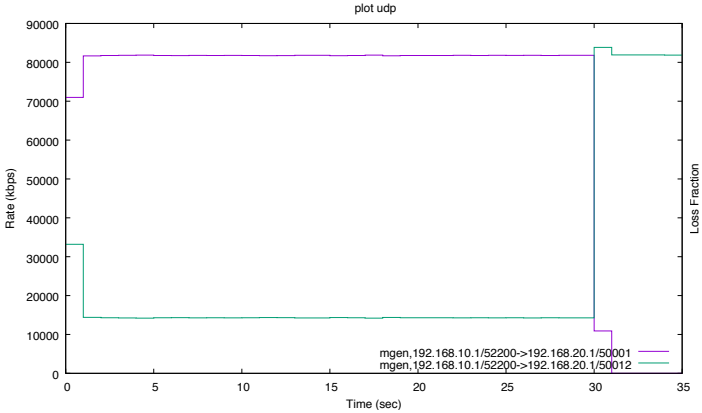
xterm

```
pef@cube:~/QOS_LAB$ cat destination_udp.mgn
0.0 LISTEN UDP 5000
0.0 LISTEN UDP 5001
```

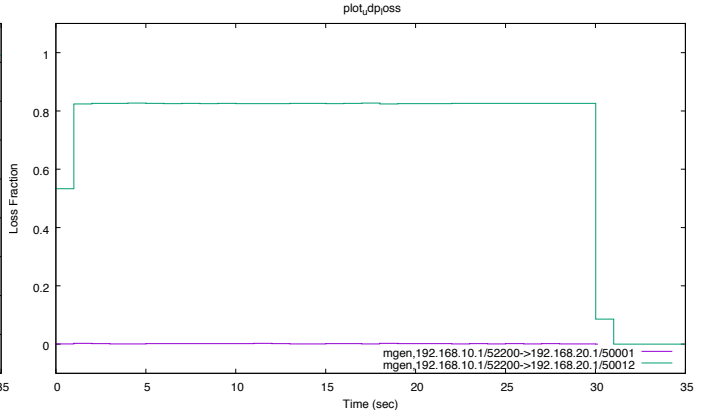
xterm

```
pef@cube:~/QOS_LAB$ trpr mgen input stats_udp.drc auto X output plot_udp
pef@cube:~/QOS_LAB$ trpr mgen input stats_udp.drc auto X loss output plot_udp_loss
pef@cube:~/QOS_LAB$ gnuplot -persistent plot_udp
pef@cube:~/QOS_LAB$ gnuplot -persistent plot_udp_loss
```

Le débit du vert contre celui du violet :



Le pourcentage de pertes subies par le vert comparé au violet :



- de 0 à 30s : l'envoi de paquets UDP en vert est défavorisé : son **taux de perte est maximum** ;
- de 30 à 35s : la transmission verte devient seule : son taux de **perte diminue** et son **débit augmente**.

ECN, «*Explicit Congestion Notification*», RFC 3168

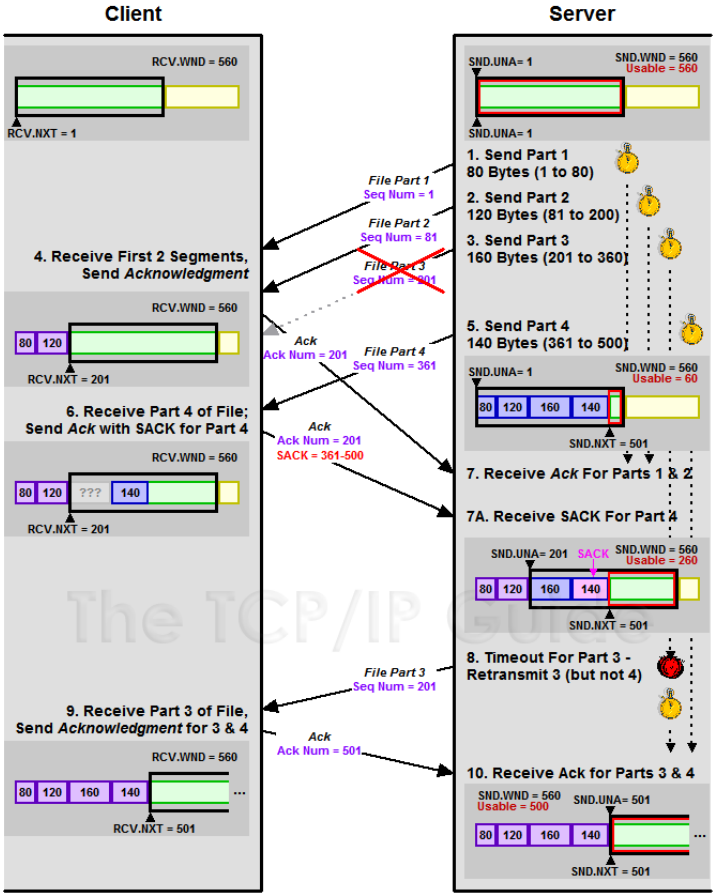
▷ *Habituellement*, la découverte de la **congestion** passe par la **perte de datagramme** :
$\Longrightarrow$ *un routeur jette un paquet qu'il ne peut gérer.*

▷ ECN permet de **notifier les extrémités** que le réseau entre en congestion **sans jeter** le datagramme :
- ◇ il est géré dans le **datagramme IP** (mais en rapport avec un protocole supérieur) ou le **segment TCP** ;
- ◇ l'ECN a été **négocié** lors de l'établissement de la connexion (utilisation du bit associé dans l'entête IP) ;
- ◇ le **routeur** gérant l'ECN **positionne un bit** dans l'entête du datagramme IP ou du segment TCP, au lieu de jeter le datagramme, pour informer, *notifier*, le récepteur de l'**entrée en congestion du routeur**.
- ◇ le **récepteur** :
  - ⋆ fait **écho de cette indication** (au niveau approprié) ;
  - ⋆ **réagit** comme lors lors d'une perte de datagramme (ou du segment encapsulé).

Le mécanisme utilise **deux bits** au choix :
- ⋆ du champ «*Differentiated Services*» ou TOS, «Type Of Service» de l'en-tête IPv4 ;
- ⋆ du champ «*Traffic Class*» de l'en-tête IPv6 ;
- ⋆ du champ «*flags*» de len-tête du segment TCP.

Les différentes **valeurs des bits** et le sens associé :
- ◇ `00`: Non ECN-Capable Transport — Non-ECT
- ◇ `10`: ECN Capable Transport — ECT(0)
- ◇ `01`: ECN Capable Transport — ECT(1)
- ◇ `11`: Congestion Encountered — CE

TCP : l'acquittement sélectif, SACK



- les premier et second segments envoyés par le serveur vers le client sont bien reçus ;
- le 3ème segment envoyé par le serveur vers le client est perdu ;
- le client envoi un acquittement pour la réception des premier et second segments ;
- le 4ème segment est bien reçu par le client ;
- avec le SACK, le client envoi un **acquittement** pour le contenu du 4ème segment bien reçu alors qu'il n'a pas le troisième segment assurant la continuité des données reçues : il y a **un trou** dans les données de la fenêtre de reception !
- grâce au SACK, le serveur renvoi **uniquement** le 4ème segment, au lieu d'envoyer les 3ème et 4ème segments.
- lors de la réception de ce 3ème segment, le client en fera l'acquittement spécifique.

Type	Meaning	Total Length and Description	RFC
0	EOL	1 byte, indicates end of option list (only used if end of options is not end of header)	793
1	NOP	1 byte, no option (used as padding to align header with Header-Length Field)	793
2	MSS	4 bytes, the last 2 of which indicate the maximum payload that one host will try to send another. Can only appear in SYN and does not change.	793 879

Type	Meaning	Total Length and Description	RFC
3	WSCALE	3 bytes, the last establishing a multiplicative (scaling) factor. Supports bit-shifted window values above 65,535.	1072
4	SACKOK	2 bytes, indicating that selective ACKs are permitted.	2018
5	SACK	Of variable length, these are the selective ACKs.	1072
6	Echo	6 bytes, the last 4 of which are to be echoed.	1072
7	Echo reply	6 bytes, the last 4 of which echo the above.	1072
8	Timestamp	10 bytes, the last 8 of which are used to compute the retransmission timer through the RTT calculation. Makes sure that an old sequence number is not accepted by the current connection.	1323
9	POC perm	2 bytes, indicating that the partial order service is permitted.	1693
10	POC profile	3 bytes, the last carrying 2-bit flags.	1693
11	CC	6 bytes, the last 4 providing a segment connection count.	1644
12	CCNEW	6 bytes, the last 4 providing new connection count.	1644
13	CCECHO	6 bytes, the last 4 echoing previous connection count.	1644

Les options 11,12&13 : RFC 1644 proposait de mettre des données dans le SYN et SYN/ACK… *abandonnées !*

Les options 6&7 : RFC 1072 *abandonnées !*

Les options 9&10 : RFC 1693 *abandonnées !*

Certaines de ces options ne sont **négociées** que lors de **l'établissement de la connexion**, pour des questions de rétrocompatibilité : leur présence dans les segments des deux interlocuteurs permet leur utilisation.

L'option 8 : permet de calculer le RTT d'un segment, «*RTT measurement*» (RFC1323) :

- l'émetteur envoie un segment avec un timestamp par rapport à sa propre horloge ;
- le récepteur lui renvoie ce timestamp dans le segment réponse : l'émetteur connait le RTT.

Taille de fenêtre proposée par le récepteur

- ⋆ dépend de la **capacité**, «bandwidth», du réseau ;
- ⋆ dépend du **temps d'aller-retour** d'un paquet, le «RTT» ;
- ⋆ est **calculée** par :

$$\text{Taille\_fenêtre} = \text{capacité} * RTT$$

- ⋆ Exemple :
  - ◇ Sur un réseau Ethernet $100Mbps$, avec un RTT de $5ms$:

$$\text{Taille\_fenêtre} = 100.10^6 * 5.10^{-3} = 500kbits$$

  - ◇ On choisit une taille en puissance de 2 : $512kbits$, soit $64Ko$.

- ⋆ ainsi on **adapte la taille de la fenêtre à la capacité du réseau**, afin d'optimiser son utilisation : l'émetteur recevra un `ACK` pour l'envoi d'une fenêtre complète juste à temps pour recommencer.

Sélection de la taille de la fenêtre de réception en programmation Socket

Sous Python :

```
1 ma_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
2 ma_socket.setsockopt(socket.SOL_SOCKET,socket.SO_RCVBUF, taille)
3 print 'Taille buffer utilisee:',ma_socket.getsockopt(socket.SOL_SOCKET,socket.SO_RCVBUF)
```

Pour le transfert d'un fichier de 10 Mo :

```
xterm
pef@solaris:~$ time python test_option_tcip.py
Connexion : 164.81.60.6
Taille buffer initial: 87380
Taille buffer utilisee: 200000

real  0m8.213s
user  0m0.080s
sys   0m0.620s
```

```
xterm
pef@solaris:~$ time python test_option_tcip.py
Connexion : 164.81.60.6
Taille buffer initial: 87380
Taille buffer utilisee: 2240

real  5m16.549s
user  0m0.244s
sys   0m3.912s
```

Débit maximal d'une connexion TCP

Le **débit** d'une connexion TCP dépend du **temps d'acheminement** des données et de leur **acquittement**, et de la **taille du buffer de réception** :

$$débit \leq \frac{RWIN}{RTT}$$

- RTT est le «*round trip time*»
- $RWIN$ est la taille de la fenêtre de réception.

Exemple avec la création d'un flux avec `iperf`

⋆ On mesure le RTT avec la commande `ping` :

xterm
```
pef@darkstar:~$ ping -c 1 msi.unilim.fr
PING msi.unilim.fr (164.81.60.6): 56 data bytes
64 bytes from 164.81.60.6: icmp_seq=0 ttl=50 time=44.537 ms
```

⋆ On intercepte la communication avec `tcpdump`

xterm
```
pef@darkstar:~$ tcpdump -c 2 -nnvvX tcp and port 5011
reading from file capture_connexion_tcp_msi.pcap, link-type EN10MB (Ethernet)
11:51:34.142682 IP (tos 0x0, ttl 64, id 19178, offset 0, flags [DF], proto TCP (6), length 64,
bad cksum 0 (->2454)!)
    192.168.42.122.55350 > 164.81.60.6.5001: Flags [S], cksum 0xcbac (incorrect -> 0x0979),
seq 3553494331, win 65535,
    options [mss 1460,nop,wscale 3,nop,nop,TS val 65182048 ecr 0,sackOK,eol], length 0

11:51:34.185676 IP (tos 0x0, ttl 50, id 0, offset 0, flags [DF], proto TCP (6), length 60)
    164.81.60.6.5001 > 192.168.42.122.55350: Flags [S.], cksum 0xb687 (correct), seq
4127531836, ack 3553494332, win 5792,
    options [mss 1460,sackOK,TS val 61550677 ecr 65182048,nop,wscale 6], length 0
```

⋆ On apprend que :

- le serveur `msi.unilim.fr` propose un «WindowScale» de 6, ce qui indique que il faut multiplier la taille de la fenêtre par $2^6 = 64$, soit une taille de fenêtre de réception $5792 * 64 = 370688$;
- le débit maximal : $370688/0.0445 = 8Mbps$ si la taille de la fenêtre ne diminue pas au cours de la communication (débit inférieur à la capacité d'un lien Ethernet $10Mbps$).

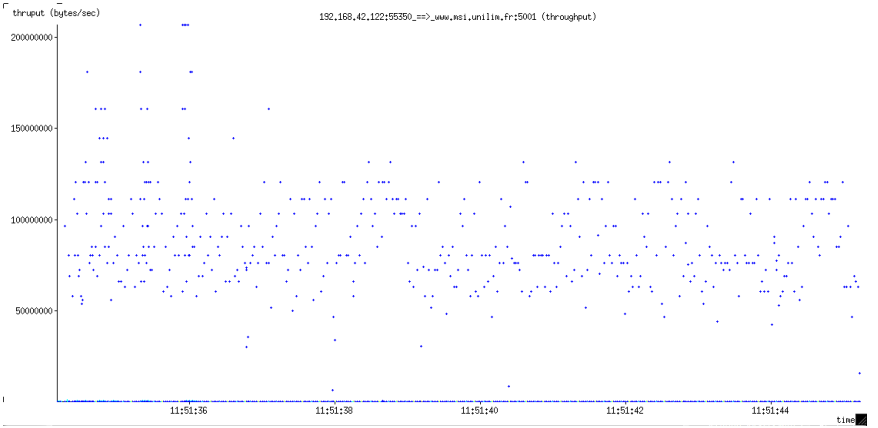
- on utilise la commande «`iperf`» en mode serveur sur `msi.unilim.fr`

xterm
```
bonnefoi@msi:~$ ./bin/iperf -s
------------------------------------------------------------
Server listening on TCP port 5001
TCP window size: 85.3 KByte (default)
------------------------------------------------------------
[ 4] local 164.81.60.6 port 5001 connected with xxx port 55350
[ 4]  0.0-12.2 sec  1.25 MBytes   860 Kbits/sec
```

- en mode client sur «darkstar» (le client envoie un flux vers le serveur) :

xterm
```
pef@darkstar:~/iperf-2.0.5/src$ ./iperf -c msi.unilim.fr
------------------------------------------------------------
Client connecting to msi.unilim.fr, TCP port 5001
TCP window size:  129 KByte (default)
------------------------------------------------------------
[ 3] local 192.168.42.122 port 55350 connected with 164.81.60.6 port 5001
[ID] Interval       Transfer     Bandwidth
[ 3]  0.0-11.0 sec  1.25 MBytes   951 Kbits/sec
```

- on observe avec la commande `tcptrace` :

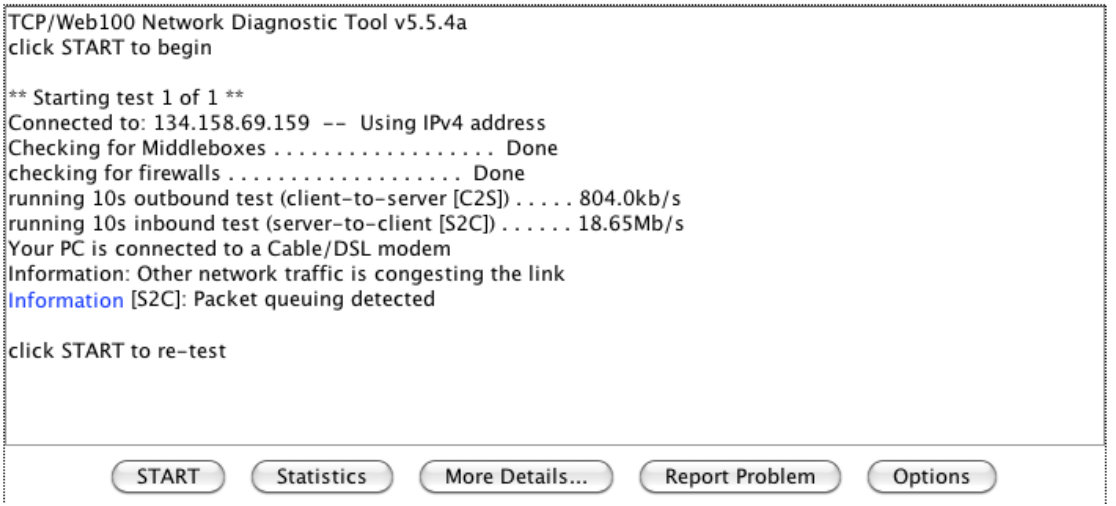


TCP : des mesures sur la connexion

Il existe un certain nombre de serveurs permettant de mesurer :

- les caractériques de la connexion (type de ligne de transmission, modifications réalisées par le FAI, *etc*)
- l'influence sur le protocole TCP :
  - de la configuration du système ;
  - de la connexion.

Sur http://npad.onenet.net/toolkit/gui/perfAdmin/directory.cgi, une liste de serveurs répartis dans le monde.



Dans l'option «more details», on apprend :

- MSS : `1448` octets, modifié par le FAI…

Problèmatique des réseaux à débit haute performance

Ces réseaux sont appelés des **LFN**, «*Long Fat Network*», ils ont un grand **BDP**, «*Bandwidth Delay Product*».

Les améliorations pour gérer ces «éléphants» sont :

- «window scale factor» : agrandir la taille de la **fenêtre de réception** en la multipliant par un coefficient $taille * 2^{coefficient}$;
- «*fast-restransmit*»&«*fast-recovery*» : conserver au mieux le débit ;
- «*Selective Acknowledge*» : éviter de gaspiller du débit ;

- «*Timestamps*» : de $1seconde$ à $1ms$ qui permettent :
  - «RTTM», *Round Trip Time Measurement* : une meilleur mesure du RTT calculable sur chaque échange y compris les segments retransmis ;
  - «PAWS», *Protect Against Wrapped Sequence Numbers* :
    - la taille du numéro de séquence est de 32bits, il faut donc pour «cycler» :

Network	B*8 bits/sec	B bytes/sec	Twrap secs
ARPANET	56kbps	7KBps	3*10**5 (~3.6 days)
DS1	1.5Mbps	190KBps	10**4 (~3 hours)
Ethernet	10Mbps	1.25MBps	1700 (~30 mins)
DS3	45Mbps	5.6MBps	380
FDDI	100Mbps	12.5MBps	170
Gigabit	1Gbps	125MBps	17

Scénario :

- ▷ un segment perdu S_{perdu} avec un numéro de séquence y arrive sur le récepteur ;
- ▷ ce segment perdu a été retransmis depuis longtemps ;
- ▷ la communication à «cyclé» et se trouve au numéro de séquence x ;
- ▷ y est proche de x et fait partie de la $RcvWindow \Rightarrow$ S_{perdu} peut s'insérer de nouveau dans la connexion !

    - On utilise alors le Timestamp pour identifier S_{perdu}, comme étant un vieux segment à ne plus utiliser :

$$\text{Timestamp}(S_{perdu}) < \text{Timestamp}(\text{courant})$$

La configuration du comportement du protocole TCP sous GNU/Linux :

xterm
```
pef@solaris:/proc/sys/net/ipv4$ ls tcp*
tcp_abc                          tcp_fin_timeout       tcp_mtu_probing           tcp_syncookies
tcp_abort_on_overflow            tcp_frto              tcp_no_metrics_save       tcp_syn_retries
tcp_adv_win_scale                tcp_frto_response     tcp_orphan_retries        tcp_thin_dupack
tcp_allowed_congestion_control   tcp_keepalive_intvl   tcp_reordering            tcp_thin_linear_timeouts
tcp_app_win                      tcp_keepalive_probes  tcp_retrans_collapse      tcp_timestamps
tcp_available_congestion_control tcp_keepalive_time    tcp_retries1              tcp_tso_win_divisor
tcp_base_mss                     tcp_low_latency       tcp_retries2              tcp_tw_recycle
tcp_congestion_control           tcp_max_orphans       tcp_rfc1337               tcp_tw_reuse
tcp_cookie_size                  tcp_max_ssthresh      tcp_rmem                  tcp_window_scaling
tcp_dma_copybreak                tcp_max_syn_backlog   tcp_sack                  tcp_wmem
tcp_dsack                        tcp_max_tw_buckets    tcp_slow_start_after_idle tcp_workaround_signed_windows
tcp_ecn                          tcp_mem               tcp_stdurg
tcp_fack                         tcp_moderate_rcvbuf   tcp_synack_retries
```

- `tcp_moderate_rcvbuf` vaut 1 : active le mode «*autotuning*», c-à-d la taille de la fenêtre de réception est adaptée automatiquement pour chaque connexion ;
- `tcp_rmem`&`tcp_wmem` :l'espace mémoire d'une connexion donné sous 3 valeurs :
  - `tcp_rmem 4096 87380 6291456` *minimal, initial, maximum* pour la réception
  - `tcp_wmem 4096 16384 4194304` *minimal, initial, maximum* pour l'émission
- les tailles demandées par l'utilisateur avec `SO_SNDBUF` et `SO_RCVBUF` sont limitées par :

xterm
```
/proc/sys/net/core/rmem_max       - maximum receive window
/proc/sys/net/core/wmem_max       - maximum send window
```

- les options les plus importantes sont activées :

xterm
```
cat /proc/sys/net/ipv4/tcp_timestamps
cat /proc/sys/net/ipv4/tcp_window_scaling
cat /proc/sys/net/ipv4/tcp_sack
```

xterm
```
echo 1 > /proc/sys/net/ipv4/tcp_moderate_rcvbuf
echo 108544 > /proc/sys/net/core/wmem_max
echo 108544 > /proc/sys/net/core/rmem_max
echo "4096 87380 4194304" > /proc/sys/net/ipv4/tcp_rmem
echo "4096 16384 4194304" > /proc/sys/net/ipv4/tcp_wmem
```

Les paramètres à changer pour 4Mo en BDP élevé «*Bandwidth Delay Product*», ou RTT*capacité

QoS et trafic réseau

Integrated Services et Réservation de ressource

- fournie de la QoS à «grain fin», «*fine-grained*» $\Longrightarrow$ plus précis ;
- utilise un **mode «circuit virtuel» simulé** par les routeurs.
  - utilisation de **protocoles dédiés** :
    - RSVP, «*Resource ReSerVation Protocol*» pour établir un chemin le plus court et le libérer ;
    - MPLS, «*Multiprotocol Label Switching*» pour faire de la *commutation* au niveau routeur ;
  - chaque routeur coopérant à la mise en place d'un chemin :
    - conserve un état des flux le traversant ;
    - réserve une partie de ses ressources au traitement d'un flux ;
- beaucoup d'états doivent être mémorisés dans chaque routeur pour la réservation des ressources :
  - fonctionne bien à échelle réduite (gros réseau d'entreprise distribué sur un pays) ;
  - fonctionne mal à grande échelle (pas pour Internet) ;

Differentiated Services et Classes de trafic

- fournie de la QoS à «gros grain», «*coarse-grained*» $\Longrightarrow$ moins précis ;
- rajoute seulement «un peu» de complexité au mode «best-effort» :
  - reste en mode «datagramme» ;
  - rajoute une priorité à chaque datagramme, *mark*, éventuellement suivant seulement deux classes de trafic :
    - une classe normale ;
    - une classe EF, «*Expedited Forwarding*» : le routeur doit faire passer le paquet avec un délai minimal et sans perte.
  - utilise des mécanismes de «queues» ou file d'attente pour traiter les paquets : les retarder, garantir un débit de traversé *throughput* ;
  - chaque queue correspond à une classe de trafic ;
  - le trafic est marqué, *marked*, mesuré, *metered*, régulé, *policed*, et mis en forme, *shaped*.
    $\Longrightarrow$«*traffic shaping*».

coopération des routeurs traversés

configuration du routeur local

«Queuing discipline» : algorithme de gestion de file d'attente

Cet algorithme permet de contrôler :

- le **volume données** envoyées (traffic shaping) ;
- la **priorité** de ces envois en fonction de critères auxquels correspondent ces données.

Il existe **deux types** d'algorithmes de gestion de files d'attente :

- ceux «***classless***» ;
- ceux «***classful***».

Classless Queuing Disciplines (Classless qdiscs)

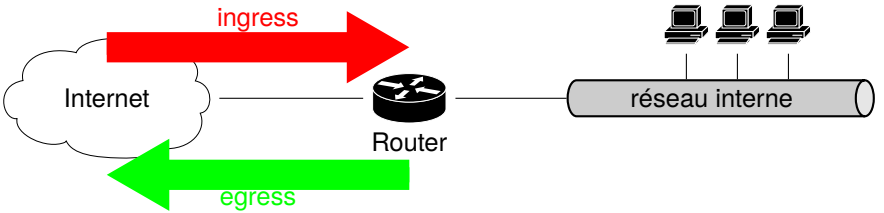
Ils correspondent au modèle le plus simple, parce qu'elles ne peuvent que :

- ▷ accepter ;
- ▷ rejeter ;
- ▷ retarder ;
- ▷ ré-ordonnancer (re-scheduling) ;

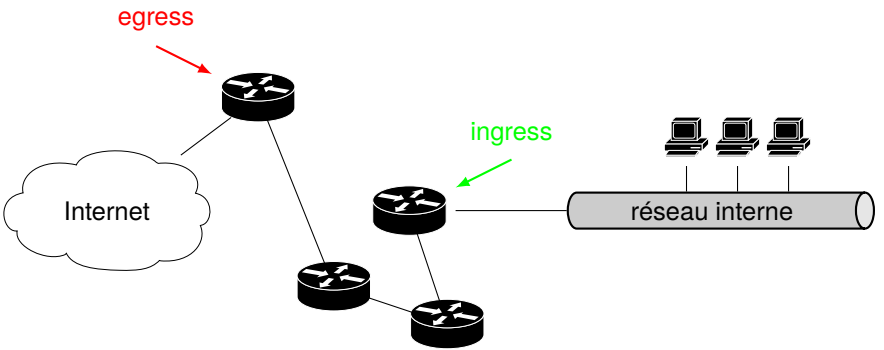
Ils implémentent différents algorithmes :

- ⋆ `pfifo` & `bfifo`, FIFO : le plus simple des «qdisc» qui correspond au «First In First Out».
  Cet algorithme utilise un buffer de taille limité, défini en nombre de paquets, `pfifo`, ou en octets, `bfifo`.
- ⋆ `pfifo_fast` : l'algorithme basé fifo et utilisant 3 files de priorités suivant le TOS ;
- ⋆ `tbf`, «*Token Bucket Filter*» : un algorithme simple à mettre en oeuvre et peu coûteux en CPU, qui permet de ralentir une interface tout en autorisant des courtes rafales d'envoi ;
- ⋆ `SFQ`, «*Stochastic Fair Queuing*» : un des algorithmes les plus utilisés qui permet de répartir de manière équitable les envois de données entre différents flux (EFSQ, «Enhanced SFQ», permet d'avoir un contrôle plus fin) ;
- ⋆ `RED` & `GRED`, «*Random Early Detection*» et «*Generic RED*» : un algorithme utilisable dans le contexte d'un «backbone» avec des débits supérieurs à 100Mbps, pour les paquets **en entrée**.

Quelques termes

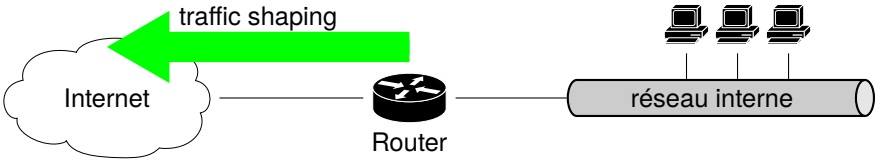


- «*ingress traffic*» : trafic en **entrée** du réseau interne ;
- «*egress traffic*» : trafic en **sortie** du réseau interne ;



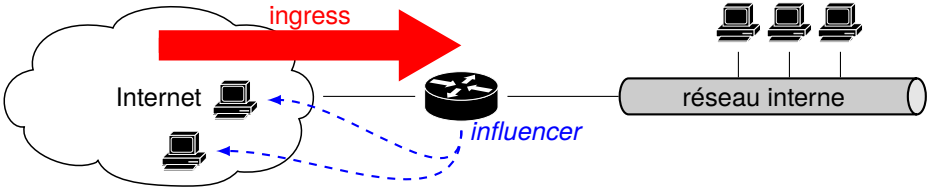
- «*ingress router*» : le routeur qui permet **l'entrée** dans le réseau d'interconnexion vers Internet ;
- «*egress router*» : le routeur qui est à **la sortie** du réseau d'interconnexion vers Internet.

▷ Il est **seulement possible** de modifier le trafic des données **en sortie**.



Cela s'appelle du «*traffic shaping*» : introduire des délais ou jeter des paquets.

▷ Il n'est **pas possible** de contrôler le trafic **en entrée** : elles dépendent de l'activité de matériels que l'on ne contrôle pas.



*Néanmoins, il est **possible de limiter** le trafic en entrée et d'influencer les émetteurs (ralentissement possible du trafic TCP en utilisant les mécanismes de protection contre la congestion : perte de paquet, utilisation d'ECN).*

Cela s'appelle du «*policing*» : **jeter des paquets** avant qu'ils ne soient pris en charge par la partie routage du noyau GNU/Linux (les retarder nécessiterait une ou plusieurs file d'attente importante).

L'outil utilisé est le **RED**, «*Random Early Detection*».

L'algorithme `pfifo_fast`

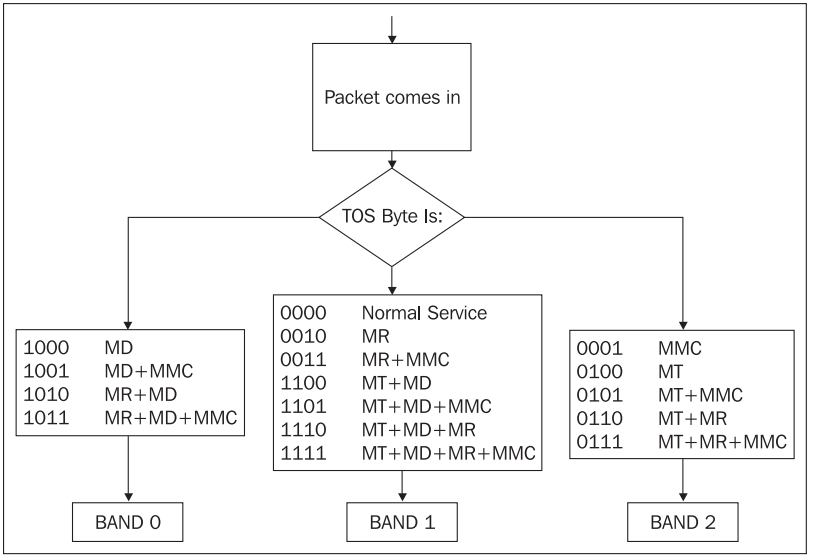
- ⋆ il est proche du `fifo` ;
- ⋆ il est «*classless*», car il réalise une **répartition fixe** des paquets suivant un critère ;
- ⋆ il dispose de 3 *canaux de données*, ou «band», dans lesquels les paquets se répartissent suivant leur TOS :
  - ◇ canal 0 : les paquets ont la plus haute priorité ;
  - ◇ canal 1 : les paquets sont envoyés uniquement lorsqu'il n'y a pas de paquets dans le canal 0 ;
  - ◇ canal 2 : les paquets ont la plus basse priorité : ils ne sont envoyés que lorsqu'il n'y a pas de paquet dans le canal 0 et 1.

Les valeurs de l'octet TOS dans l'en-tête du datagramme IP :

les différentes valeurs des bits :

0 1 2	3 4 5 6	7
PRECEDENCE	Type of Service	Unused

- ◦ 0000 : service normal ;
- ◦ 0001 : MMC, «*Minimize Monetary Cost*» ;
- ◦ 0010 : MR, «*Maximize Reliability*» ;
- ◦ 0100 : MT, «*Maximize Throughput*» ;
- ◦ 1000 : MD, «*Minimize Delay*».



Afficher les différents types de service

À l'aide de la commande :

```
$ sudo tcpdump -vv -i eth0
```

Calculer les priorités

TOS	Bits	Means	Linux Priority	Band
0x0	0	Normal Service	0 Best Effort	1
0x2	1	Minimize Monetary Cost	1 Filler	2
0x4	2	Maximize Reliability	0 Best Effort	1
0x6	3	mmc+mr	0 Best Effort	1
0x8	4	Maximize Throughput	2 Bulk	2
0xa	5	mmc+mt	2 Bulk	2
0xc	6	mr+mt	2 Bulk	2
0xe	7	mmc+mr+mt	2 Bulk	2
0x10	8	Minimize Delay	6 Interactive	0
0x12	9	mmc+md	6 Interactive	0
0x14	10	mr+md	6 Interactive	0
0x16	11	mmc+mr+md	6 Interactive	0
0x18	12	mt+md	4 Int. Bulk	1
0x1a	13	mmc+mt+md	4 Int. Bulk	1
0x1c	14	mr+mt+md	4 Int. Bulk	1
0x1e	15	mmc+mr+mt+md	4 Int. Bulk	1

Binary	Decimcal	Meaning
1000	8	Minimize delay (md)
0100	4	Maximize throughput (mt)
0010	2	Maximize reliability (mr)
0001	1	Minimize monetary cost (mmc)
0000	0	Normal Service

Le «dutch packet» :

MMC + MT + MD + MR = 8+4+2+1

La dernière colonne indique la «priomap», c-à-d dans quel canal est envoyé le paquet de chaque valeur de priorité.

Exemple : le paquet de priorité 4 est envoyé dans le canal 1 (la priorité commence à zéro).

«*Type of Service in the Internet Protocol*», RFC 1349

Protocol	TOS Value	
TELNET (1)	1000	(minimize delay)
FTP		
Control	1000	(minimize delay)
Data (2)	0100	(maximize throughput)
TFTP	1000	(minimize delay)
SMTP (3)		
Command phase	1000	(minimize delay)
DATA phase	0100	(maximize throughput)
Domain Name Service		
UDP Query	1000	(minimize delay)
TCP Query	0000	
Zone Transfer	0100	(maximize throughput)
NNTP	0001	(minimize monetary cost)
ICMP		
Errors	0000	
Requests	0000 (4)	
Responses	<same as request> (4)	

(4) Although ICMP request messages are normally sent with the default TOS, there are sometimes good reasons why they would be sent with some other TOS value. An ICMP response always uses the same TOS value as was used in the corresponding ICMP request message.

La configuration des interfaces

xterm

```
$ ip link
1: eth1: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast state UP qlen 1000
    link/ether 00:11:de:ad:be:ef brd ff:ff:ff:ff:ff:ff
```

C'est du `pfifo_fast` par défaut.

«*Token Bucket Filter*»

Cet algorithme correspond à :

- une configuration :
  - un seau, le *bucket* : qui se remplit de jetons, *tokens*, à une fréquence choisie, *token rate* ;
  - une capacité pour le seau en nombre de jetons ;
- l'association d'un paquet et d'un jeton :
  - le jeton est associé à un paquet ;
  - le jeton est supprimé du seau ;
- deux flux : un pour les jetons et l'autre pour les paquets.

Il existe 3 cas possibles :

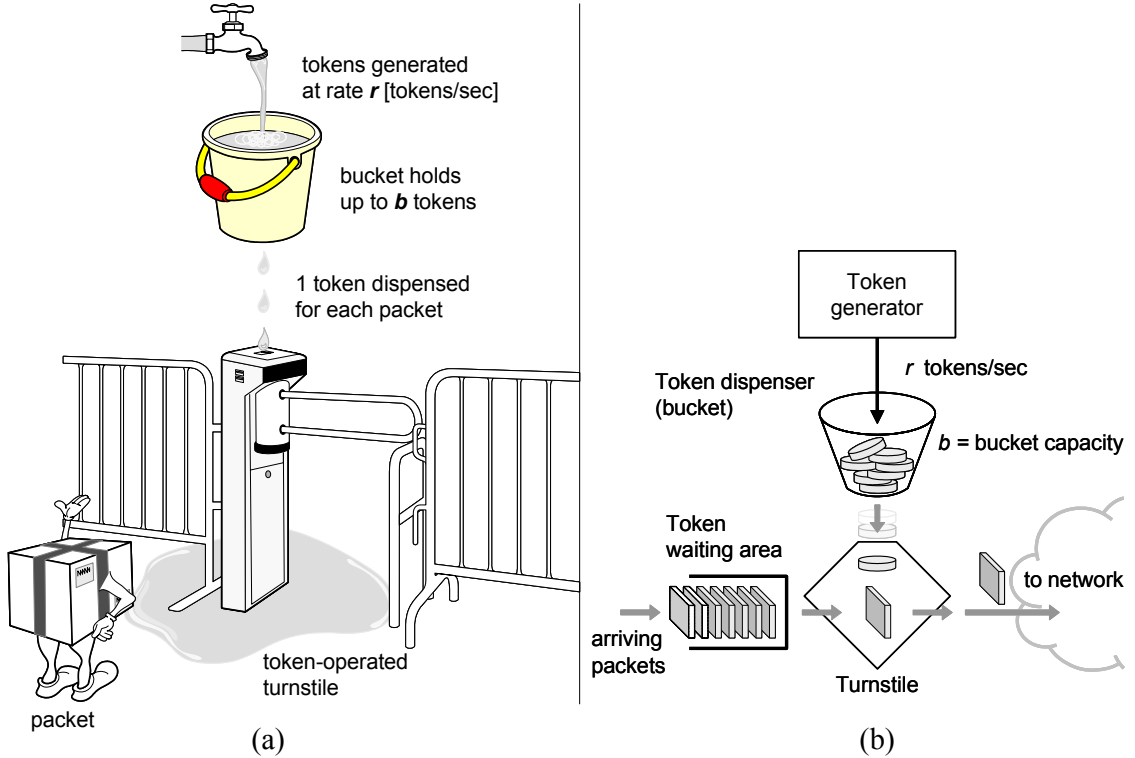
1. les paquets arrivent au **même rythme** que les jetons : chaque paquet est associé à un jeton et traverse la file d'attente sans délai ;

2. les paquets arrivent à un **rythme inférieur** à celui des jetons : une partie seulement des jetons est utilisée, et le reste s'accumule jusqu'à remplir le seau en fonction de sa taille choisie.
   Les jetons non utilisés permettent d'envoyer des paquets à un **débit supérieur** à celui d'arrivée des jetons et pour une courte durée : envoi d'un groupe de paquets en rafale, *data burst*.
   Si le débit continue à dépasser celui des jetons on bascule dans le cas suivant.

3. les paquets arrivent à un **rythme supérieur** à celui d'arrivée des jetons.
   Les jetons vont s'épuiser et la file d'attente va ralentir le débit des paquets jusqu'à en éliminer, situation appelée *overlimit*.

Remarque : L'implémentation dans GNU/Linux correspond à assimiler les jetons à des octets.

Illustration du seau



La différence entre le «*leaky bucket*», (a), et le «*token bucket*», (b) : la possibilité d'un mode «*burst*» dans le «*token bucket*».

«Stochastic Fairness Queuing»

Ce comportement permet de répartir l'utilisation de l'interface entre différents flux, ou conversations :

- connexions TCP ;
- stream UDP (vidéo+son par exemple).

Fonctionnement :

◇ le trafic est décomposé en différentes files d'attente, une pour chaque conversation ;

◇ l'envoi de paquet est ensuite réalisé à tour de rôle depuis chaque file, «round robin» ;
ce qui donne un chance de progresser à chaque conversation, chacune suivant son tour.

On évite ainsi qu'une conversation très active entraîne l'interruption des autres.

Réalisation :

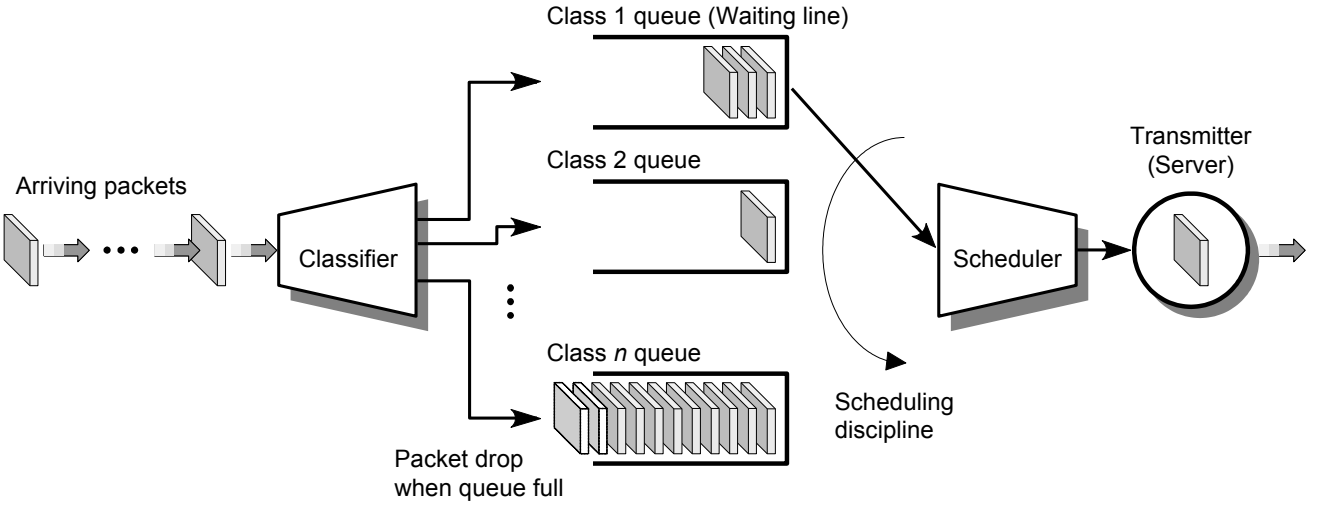
⋆ on définit un certain nombre de files d'attentes ;

⋆ on utilise une fonction de hashage permettant d'associer une conversation à une de ces files d'attentes (par exemple en appliquant la fonction de hashage sur le couple de TSAPs des échanges) ;

⋆ en cas de collision, c-à-d où deux conversations seraient affectées à la même file d'attente, on a un débit divisé par deux.
Pour éviter cette situation, on change régulièrement les paramètres de la fonction de hashage, ce qui permet de ne pas avoir de collision, pour une même conversation, que durant un court moment (quelques secondes).

Remarque : l'intérêt de cet algorithme intervient quand l'interface de sortie est saturée.

Illustration



Ici, le «round robin» est réalisé par le «scheduler».

Exposé du problème

Une «file d'attente», *queue*, est FIFO :

- les paquets de tête traversent le routeur ;
- les paquets entrants se placent en queue, *tail* ;
- lorsqu'un **problème de débit** arrive, c-à-d que des paquets arrivent avec un débit plus grand que celui géré ou autorisé, ces paquets entrants sont **rejetés**, on parle de «*DropTail*» ;
- ces **rejets** vont entraîner des **perturbations** sur les flux affectés :
  - pour un flux TCP, il y aura un «*backoff*» du à l'algorithme de protection contre la congestion ;
  - en affectant plusieurs flux simultanément, tous ces flux vont faire un «*backoff*» : résultat le trafic en entrée va **fortement diminuer** et la capacité de la ligne de transmission va être **sous-employée**.

  $\Longrightarrow$ pour éviter une réaction globale, il faut une solution qui n'affecte que quelques flux et qui soit **équitable**.

Présentation d'une solution

L'idée de l'algorithme RED est de :

- **détecter** le passage en saturation ;
- **prévenir** les flux de cette saturation :
  - en marquant le paquet avec l'ECN s'il est géré (marquer le paquet comme *overlimit*) ;
  - en le rejetant sinon ;
- **traiter** les paquets de manière **aléatoire** : suivant une probabilité proportionnelle à la situation de congestion ;
- **favoriser** TCP au dépends d'UDP qui ne dispose pas de mécanisme de contrôle de congestion ;

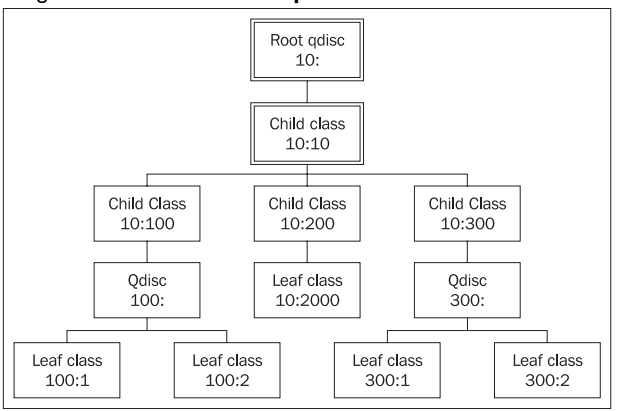
L'algorithme RED :

- le fonctionnement est «*probabiliste*», on définit :
  - une limite min où le paquet a une **faible probabilité** d'être marqué/rejeté (cela correspond à une taille de la queue, puisque pour un paquet arrivant trop vite, il n'est pas traité tout de suite et il est donc mis dans la queue) ;
  - on définit une limite max à partir de laquelle les paquets sont **sûrement marqués/rejetés** (probabilité forte) ;
- tout paquet de la file d'entrée situé après le min a une «chance» d'être marqué/rejeté.

Classful Queuing Disciplines (Classful qdiscs)

- ⋆ une classe est une catégorie de trafic ;
- ⋆ une qdisc «classful» est un algorithme de gestion de file d'attente basé sur l'utilisation d'un partitionnement du trafic en différentes classes :
  - ◇ pour savoir à quelle classe appartient un paquet il faut utiliser un «classifier» ;
  - ◇ la classification peut être réalisée à l'aide de filtres : un filtre s'applique s'il y a une correspondance entre sa définition et le paquet à traiter.
- ⋆ après classification, il est nécessaire de définir l'ordre d'envoi des différents paquets, «scheduling».

L'organisation est **hiérarchique** :



- ▷ chaque interface dispose d'une qdisc **racine** à laquelle est associée une catégorie ;
- ▷ une catégorie peut avoir :
  - ◇ une qdisc parent pour envoyer son trafic (par défaut c'est du pfifo_fast) ;
  - ◇ une catégorie parent ;
  - ◇ une ou plusieurs catégories filles ;
  - ◇ aucun enfant : c'est une catégorie feuille ou «leaf» ;

- ▷ l'identification de ces éléments est faite à l'aide de la notation `<major>:<minor>` :
  - ◇ la racine est identifiée par `1:` ou `1:0`, *ici par «`10:`»* ;
  - ◇ les catégories doivent avoir le même «major» que leur parent, qui doit être unique pour une même interface :
  - ◇ le «minor» doit être unique dans le même qdisc et les catégories associées.

Les algorithmes les plus utilisés sont :

- `PRIO`, «*Priority*» : un algorithme dérivé de `pfifo_fast`, avec la possibilité d'augmenter le nombre de canaux et permettant d'utiliser des filtres basés sur plus d'information que le TOS ;
- `CBQ`, «*Class Based Queuing*» : c'est l'algorithme le plus complexe et aussi le plus utilisé qui utilise également les informations du TOS ;
- `HTB`, «*Hierarchical Token Bucket*» : permet d'aller au-delà des limitations de CBQ.

CBQ

Cet algorithme a pour but de contrôler les débits de sortie d'une interface sans réaliser de calculs complexes :

- il utilise le **temps d'inactivité**, «*idle time*», de l'interface pour déduire son débit effectif :
  - pour réduire le débit d'une interface de 100Mbps à 10Mbps, il faut qu'elle reste inutilisée à 90% ;
  - pour mesurer ce temps d'inactivité, l'algorithme mesure le délai en micro-secondes entre les appels matériels à la carte réseau :
    - problème matériel : la carte réseau n'obtient pas toujours le débit réseau maximum par rapport à la technologie de la carte (mauvais pilote ou mauvaise qualité des composants) ;
    - l'évaluation finale peut être très différente de la valeur réelle.
- il permet de **répartir le débit de la carte** entre différentes catégories de trafic :
  - chaque catégorie reçoit l'allocation d'un débit maximum ;
  - une catégorie peut ou non prendre le débit associé à une autre classe si celle-ci est inactive :
    - du point de vue du «préteur» : «*isolated*» ou «*sharing*» ;
    - du point de vue de «l'emprunteur» : «*bounded*» ou non.
- il permet également de **classifier** suivant «chaque bit» du TOS pour effectuer une prise de décision rapide.

Afficher et supprimer une configuration

Afficher l'algorithme qdisc associé à une interface :

xterm
```
$ tc -s qdisc ls dev eth1
qdisc pfifo_fast 0: root refcnt 2 bands 3 priomap  1 2 2 2 1 2 0 0 1 1 1 1 1 1 1 1
 Sent 1654333 bytes 21160 pkt (dropped 0, overlimits 0 requeues 0)
 backlog 0b 0p requeues 0
```

Supprimer la configuration d'un algorithme qdisc, c-à-d. revenir à la configuration par défaut :

xterm
```
$ sudo tc qdisc del dev DEV root
```

Réaliser la classification à l'aide de filtre

Trafic de source Web et de destination SSH en priorité 1, trafic autre en priorité basse :

xterm
```
# tc filter add dev eth0 protocol ip parent 10: prio 1 u32 match ip dport 22 0xffff flowid 10:1
# tc filter add dev eth0 protocol ip parent 10: prio 1 u32 match ip sport 80 0xffff flowid 10:1
# tc filter add dev eth0 protocol ip parent 10: prio 2 flowid 10:2
```

Le 80 0xffff indique configuration binaire égale à la valeur sur deux octets 80 uniquement.

Trafic identifié par la source ou la destination :

xterm
```
# tc filter add dev eth0 parent 10:0 protocol ip prio 1 u32 match ip dst 4.3.2.1/32 flowid 10:1
# tc filter add dev eth0 parent 10:0 protocol ip prio 1 u32 match ip src 1.2.3.4/32 flowid 10:1
```

Trafic identifié par la source et par un port :

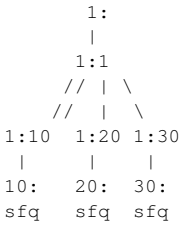
xterm
```
# tc filter add dev eth0 parent 10:0 protocol ip prio 1 u32 match ip src 4.3.2.1/32 match ip sport 80 0xffff
flowid 10:1
```

Trafic identifié par le TOS, (*ici, le trafic dit interactif*) :

xterm
```
# tc filter add dev eth0 parent 1:0 protocol ip prio 10 u32 match ip tos 0x10 0xff flowid 1:4
```

HTB

Exemple :



On veut limiter :

- le trafic Web à 5 Mbps ;
- le trafic SMTP à 3 Mbps ;
- le trafic restant dispose d'un petit débit ou de tout le débit si les deux autres sont absents.
- les deux ensemble ne doivent pas dépasser 6 Mbps.

On configure les qdisc :

xterm

```
# tc qdisc add dev eth0 root handle 1: htb default 30
# tc class add dev eth0 parent 1: classid 1:1 htb rate 6mbit burst 15k
# tc class add dev eth0 parent 1:1 classid 1:10 htb rate 5mbit burst 15k
# tc class add dev eth0 parent 1:1 classid 1:20 htb rate 3mbit ceil 6mbit burst 15k
# tc class add dev eth0 parent 1:1 classid 1:30 htb rate 1kbit ceil 6mbit burst 15k
# tc qdisc add dev eth0 parent 1:10 handle 10: sfq perturb 10
# tc qdisc add dev eth0 parent 1:20 handle 20: sfq perturb 10
# tc qdisc add dev eth0 parent 1:30 handle 30: sfq perturb 10
```

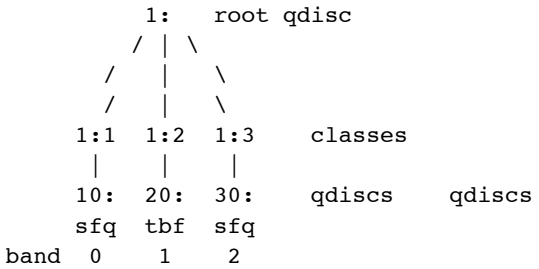
Puis le trafic vers les catégories, avec `tc filter` :

xterm

```
# tc filter add dev eth0 protocol ip parent 1:0 prio 1 u32 match ip dport 80 0xffff flowid 1:10
# tc filter add dev eth0 protocol ip parent 1:0 prio 1 u32 match ip sport 25 0xffff flowid 1:20
```

Classful qdiscs : PRIO

Exemple :



xterm

```
# tc qdisc add dev eth0 root handle 1: prio
# # This *instantly* creates classes 1:1, 1:2, 1:3
# tc qdisc add dev eth0 parent 1:1 handle 10: sfq
# tc qdisc add dev eth0 parent 1:2 handle 20: tbf rate
20kbit buffer 1600 limit 3000
# tc qdisc add dev eth0 parent 1:3 handle 30: sfq
```

Ce qui donne :

xterm

```
# tc -s qdisc ls dev eth0
qdisc prio 1: root refcnt 2 bands 3 priomap  1 2 2 2 1 2 0 0 1 1 1 1 1 1 1 1
 Sent 0 bytes 0 pkt (dropped 0, overlimits 0 requeues 0)
 backlog 0b 0p requeues 0
qdisc sfq 10: parent 1:1 limit 127p quantum 1514b
 Sent 0 bytes 0 pkt (dropped 0, overlimits 0 requeues 0)
 backlog 0b 0p requeues 0
qdisc tbf 20: parent 1:2 rate 20000bit burst 1600b lat 560.0ms
 Sent 0 bytes 0 pkt (dropped 0, overlimits 0 requeues 0)
 backlog 0b 0p requeues 0
qdisc sfq 30: parent 1:3 limit 127p quantum 1514b
 Sent 0 bytes 0 pkt (dropped 0, overlimits 0 requeues 0)
 backlog 0b 0p requeues 0
```

Méthodologie : marquer les paquets

Il est possible de «marquer» un paquet à l'aide **d'iptables** et ensuite de le traiter dans un **qdisc** :

a. on utilise le `fwmark`, c-à-d une marque utilisée pour le «forwarding» ;

b. on place la marque dans la table «mangle» et dans la chaîne `PREROUTING` ;

xterm
```
# iptables -A PREROUTING -t mangle -i eth0 -j MARK --set-mark 6
# iptables -A PREROUTING -i eth0 -t mangle -p tcp --dport 25 -j MARK --set-mark 1
```

c. on utilise `tc filter` pour tenir compte de cette marque :

xterm
```
# tc filter add dev eth1 protocol ip parent 1:0 prio 1 handle 6 fw flowid 1:1
```

Ici, la valeur de la marque est 6, mais elle peut être choisie librement.

Avantages

▷ on **n'apprend pas** la syntaxe de `tc filter` **mais seulement** celle de `iptables` !

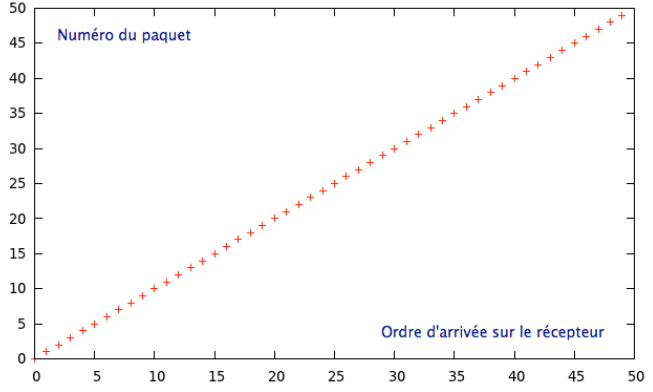
Utilisation de `netem`, «network emulator»

Le module `netem` permet de tester les applications en condition «réelles», on peut ainsi :

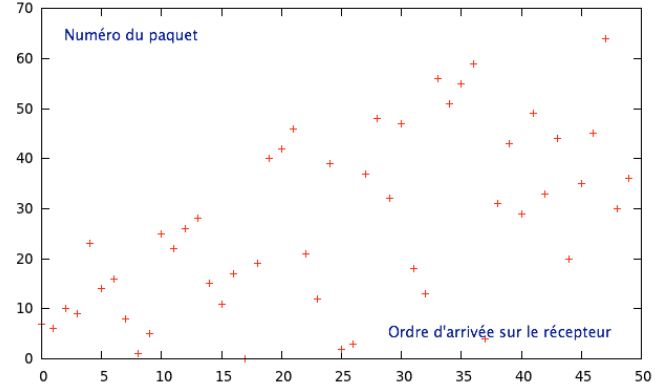
⋆ introduire du retard sur le passage des paquets :

```
xterm
$ sudo tc qdisc add dev bridge_dmz root netem delay 100ms 75ms
```

Chaque paquet est retardé de $100ms$ *avec une variation de* $\pm 75ms$*, ce qui entraîne leur réordonnancement.*



Affichage des 50 premiers paquets sur 1000 envoyés.
Pas d'application de `netem`.



Application de `netem` : les paquets arrivent dans le désordre (réception de paquets jusqu'au numéro 65).

⋆ introduire des pertes de paquets :

```
xterm
tc qdisc change dev eth0 root netem loss 0.1%
```

⋆ abîmés certains paquets aléatoirement :

```
xterm
tc qdisc change dev eth0 root netem corrupt 0.1%
```

L'influence du jitter sur le protocole TCP

Mesure de la commande `iperf` sans «netem» :

xterm

```
root@INTERNAL_HOSTA:~/RESEAUX_I/FIREWALL# iperf -c 137.204.212.208
------------------------------------------------------------
Client connecting to 137.204.212.208, TCP port 5001
TCP window size: 16.0 KByte (default)
------------------------------------------------------------
[  3] local 137.204.212.11 port 42168 connected with 137.204.212.208 port 5001
[ ID] Interval       Transfer     Bandwidth
[  3]  0.0-10.0 sec  1.22 GBytes  1.05 Gbits/sec
```

Ajout de netem :

xterm

```
sudo tc qdisc add dev bridge_dmz root netem delay 100ms 75ms
```

La mesure :

xterm

```
root@INTERNAL_HOSTA:~/RESEAUX_I/FIREWALL# iperf -c 137.204.212.208
------------------------------------------------------------
Client connecting to 137.204.212.208, TCP port 5001
TCP window size: 16.0 KByte (default)
------------------------------------------------------------
[  3] local 137.204.212.11 port 42167 connected with 137.204.212.208 port 5001
[ ID] Interval       Transfer     Bandwidth
[  3]  0.0-10.2 sec  3.38 MBytes  2.79 Mbits/sec
```

En utilisant tcpdump et tcptrace, on peut analyser le flux transmis avec iperf sous l'influence de netem :

xterm
```
tcpdump -i veth_DMZ_WEB -s 80 -w capture_netem_iperf.pcap tcp and port 5001
```

Puis l'analyse :

xterm
```
tcptrace -Tn capture_netem_iperf.pcap
xplot.org -mono a2b_tput.xpl
```

Ce qui donne le graphique suivant, pour l'évolution du débit, *throughput* :