istic Informatique Électronique

Architecture Réseaux Entreprises (ARES)

Routage et filtrage dans les réseaux Linux

Brice - Ekane (brice.ekane@univ-rennes.fr)

ISTIC Rennes - France

2025-2026

git clone https://github.com/bekane/ares-2025.git

Plan du module

Références bibliographiques (1/2)

Documentation officielle

- Netfilter Project : `https://www.netfilter.org/`
- Manuels Linux : `man iptables`, `man nft`, `man ip`
- Documentation kernel Linux (networking) : `https://www.kernel.org/doc/html/latest/networking/index.html`

Guides pratiques

- *Linux Firewalls: Attack Detection and Response with iptables, psad, and fwsnort*, Michael Rash, No Starch Press.
- ArchWiki – iptables : `https://wiki.archlinux.org/title/iptables`
- ArchWiki – nftables : `https://wiki.archlinux.org/title/nftables`
- Debian Wiki – nftables : `https://wiki.debian.org/nftables`

Références bibliographiques (2/2)

Articles de référence

- P. Ayuso, *Mastering Linux Security and Hardening*, Packt, 2022.
- Blog RedHat – nftables vs iptables : `https://www.redhat.com/en/blog/iptables-nftables`

Plan

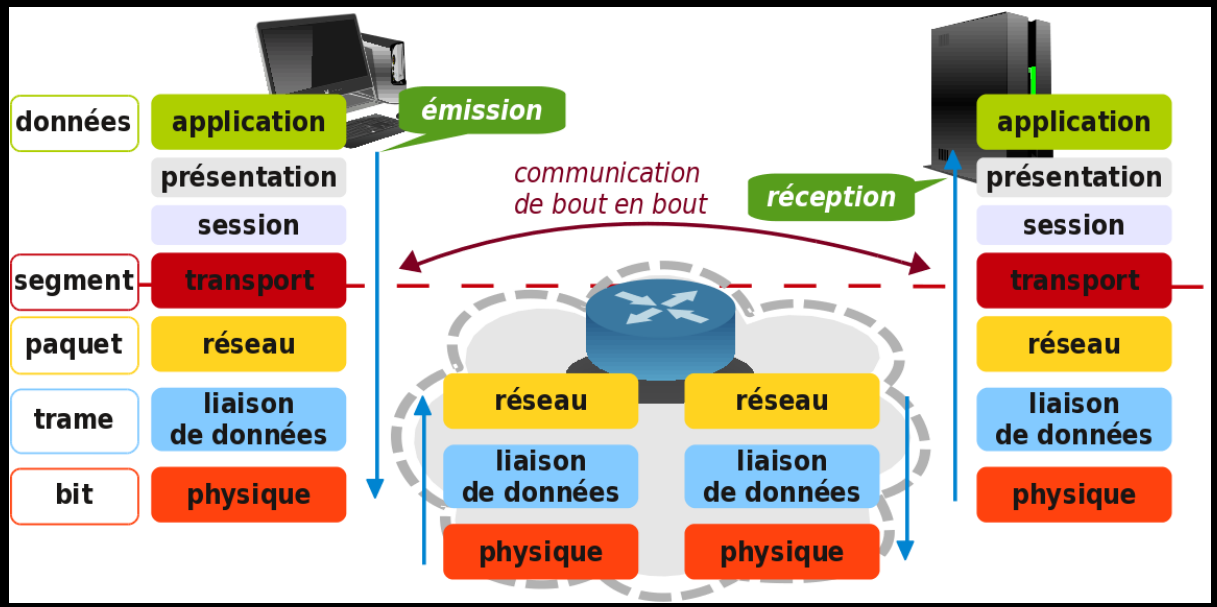
Objectifs pédagogiques

- Comprendre les bases du routage IP sous Linux.
- Maîtriser les outils de configuration des routes et de filtrage.
- Diagnostiquer et sécuriser les flux réseau via les pare-feux.

Qu'est-ce que le routage ?

- C'est l'art de **choisir le meilleur chemin** pour un paquet de données.
- Un **routeur** est l'appareil qui prend cette décision.
- L'objectif : que le paquet arrive à destination le plus rapidement et efficacement possible.

Routeur et couche 3



de: https://inetdoc.developpez.com/tutoriels/modelisation-reseau/#L2-3-2

- connecte plusieurs réseau
- utilise l'adress ip pour prendre la decision.

Format des paquets - Frame Ethernet

	6 bytes	MAC address of the receiving device, or broadcast address (FF:FF:FF:FF:FF:FF), or multicast address (least significant bit of the first byte is set to 1)
	6 bytes	MAC address of the transmitting device
	2 bytes	Protocol type (e.g. IPv4, IPv6, ARP) or encapsulation method (VLAN tag) used in the payload
	46-1500 bytes	Payload data being transmitted. If payload is smaller than 46 bytes, padding is appended to the payload to reach the minimum 64-byte frame size
	4 bytes	"Frame Check Sequence" is a cyclic redundancy check (CRC) value used for error detection

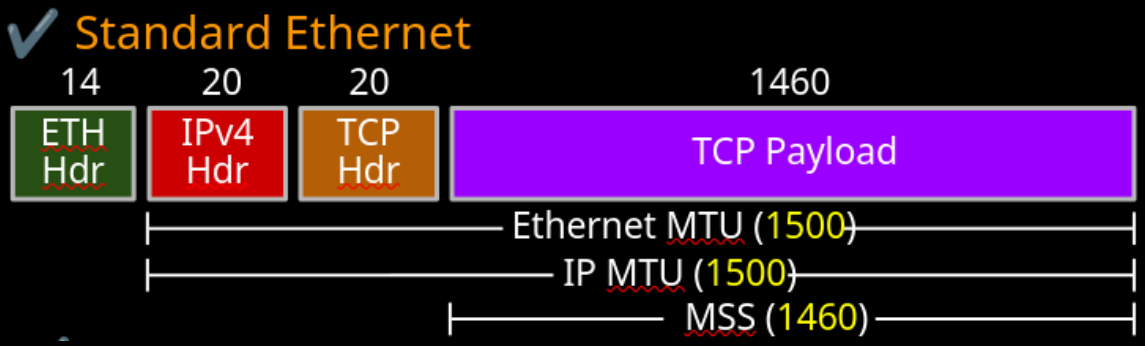
Minimum frame size	**64 bytes** (header + payload + FCS)
Maximum frame size	**1518 bytes** (header + payload + FCS) for standard Ethernet frames, or up-to **9018 bytes** for jumbo frames

Format des paquets - Payload Ethernet

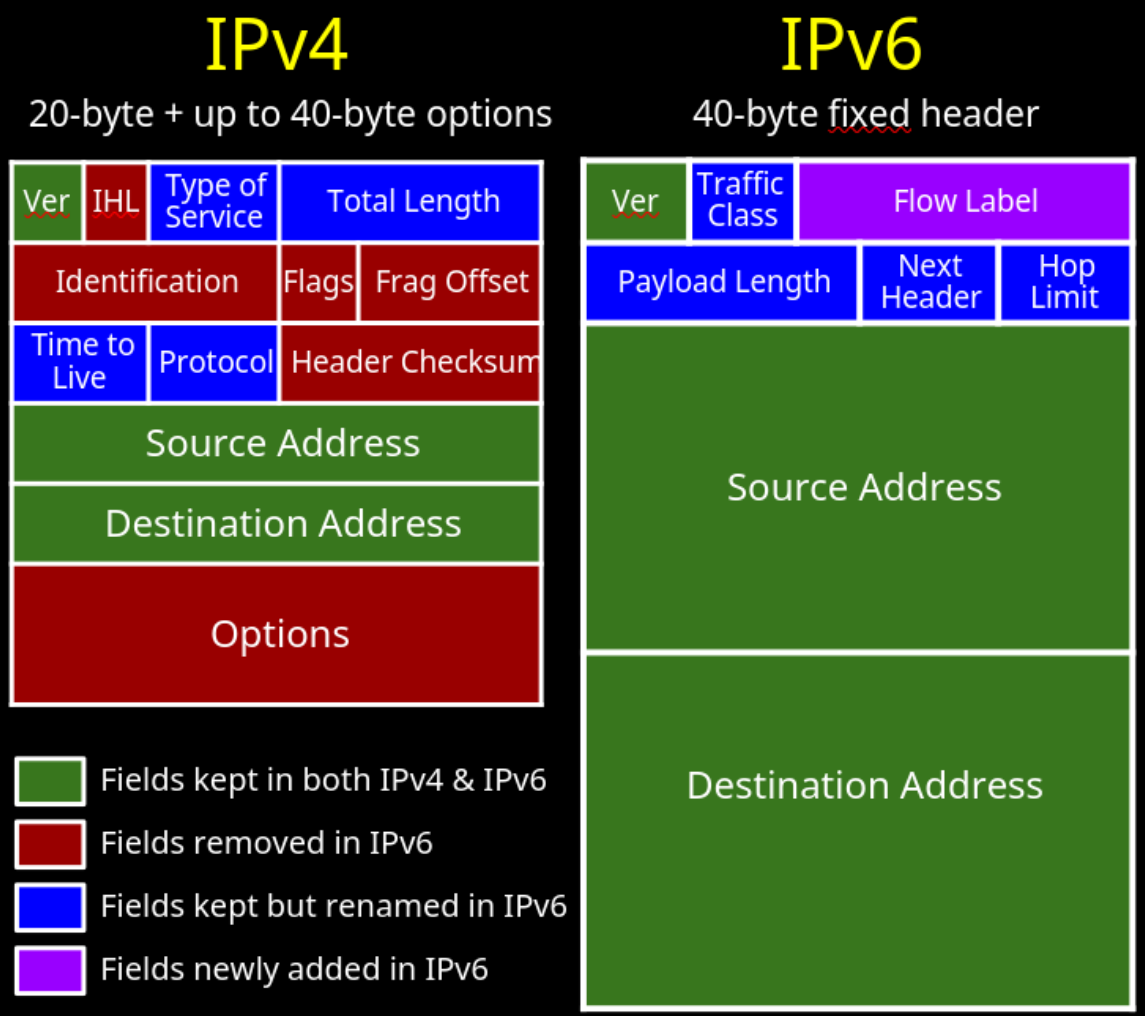
Ethernet MTU (Maximum Transmission Unit) is the maximum Layer 3 (IP) packet size that fits in an Ethernet frame's payload (typically 1500B)

IP MTU is the maximum IP packet size that can be transmitted without IP fragmentation, which is usually equal to Ethernet MTU, unless additional encapsulation reduces it

MSS (Maximum Segment Size) is the maximum TCP payload size, calculated as IP MTU - (IP + TCP header sizes)



Format des paquets - Entête IPv4 et IPv6



Linux et routage sur matériel standard

- Routage statique = routes manuelles -> limité aux petites topologies.
- Limites : capacité CPU/RAM, absence d'accélération matérielle (ASIC, TCAM).

Linux et routage sur matériel standard

Routage dynamique

- Réseaux moyens/grands -> besoin de protocoles dynamiques (RIP, OSPF, BGP...).
- Démon de routage (**FRR**, **Quagga**) = logiciel qui parle ces protocoles.

Plan

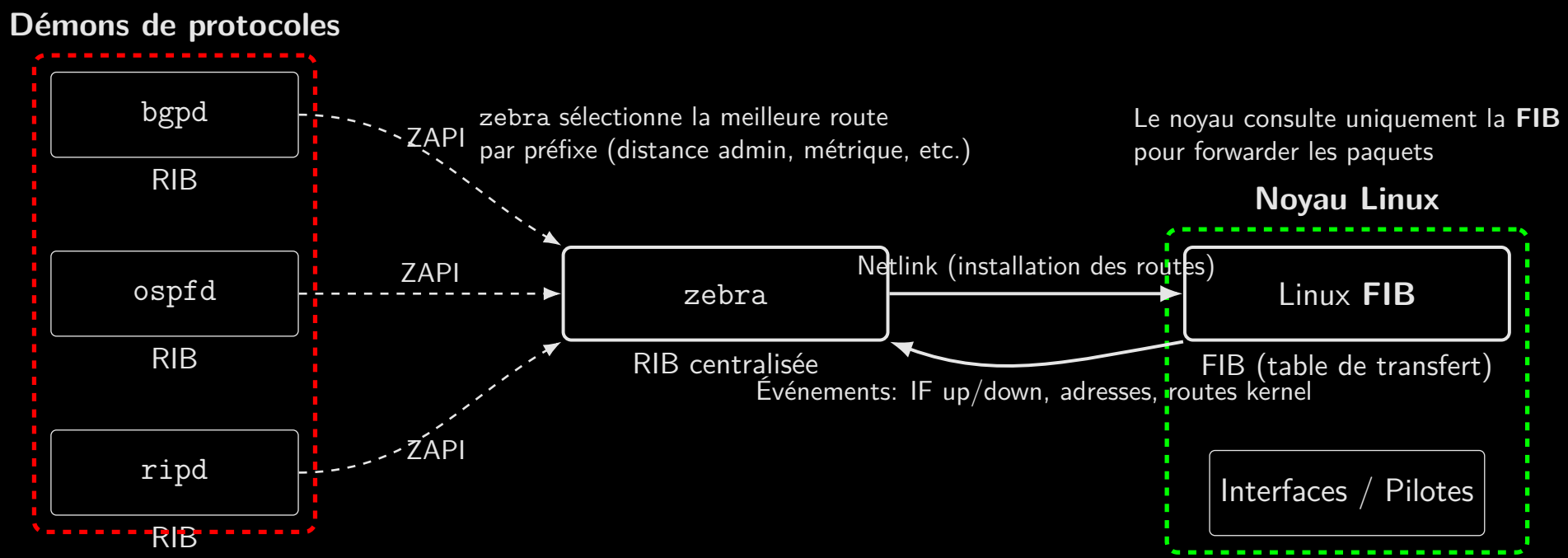
2. Structures internes

FRR sous Linux

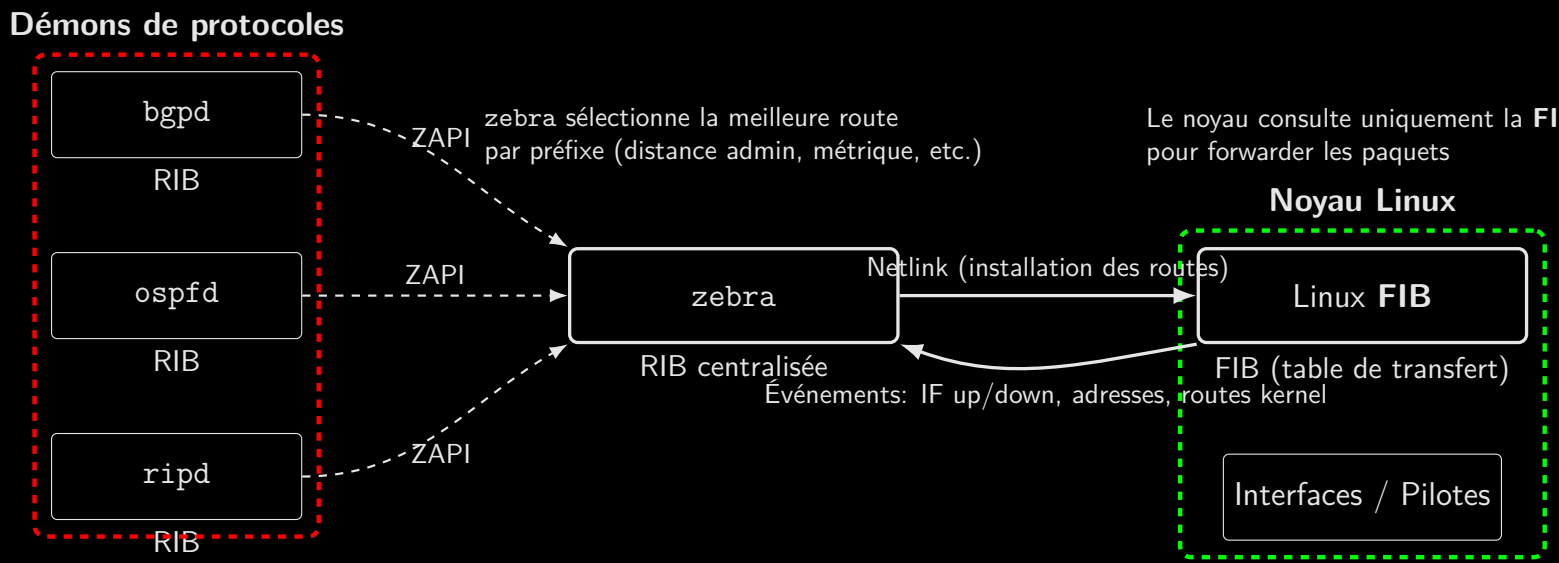
FRR (FRRouting)

- Fork moderne de Quagga (2016), développement actif.
- Protocoles supplémentaires : EVPN, BFD, Segment Routing, VRF avancé.
- Automatisation (API, YANG). dans le module 8

Implémentation FRR sur Linux



Mise à jour dynamique de la FIB

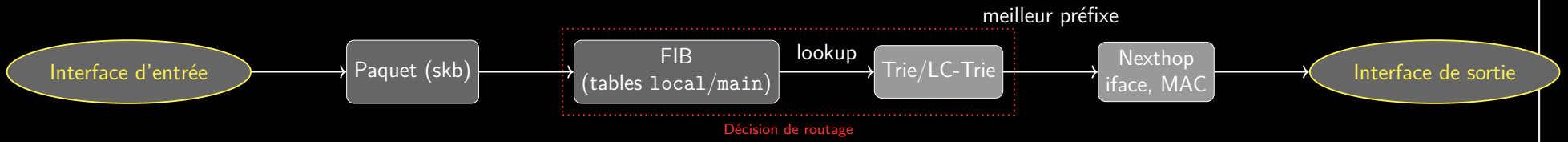


- Les démons FRR (BGP, OSPF, RIP, . . .) remplissent chacun leur **RIB**.
- **Zebra** fusionne et choisit la meilleure route.
- La **FIB du noyau** (LC-Trie) est mise à jour pour le forwarding.

Que fait Linux quand un paquet arrive ?

Selection du meilleur chemin

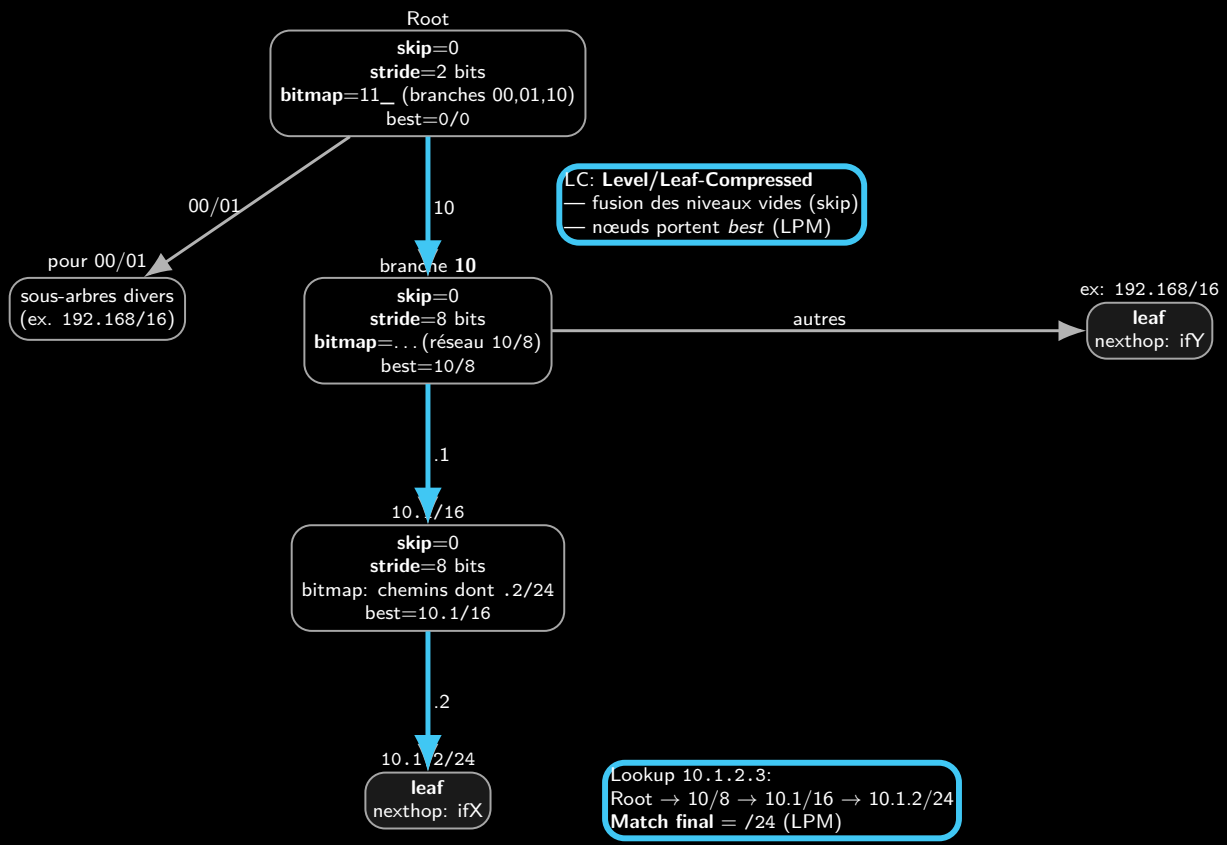
- La **FIB** (Forwarding Information Base) garde le chemin le plus rapide pour arriver à destination.
- C'est la FIB qui est utilisée pour faire avancer le paquet.



LC-Trie (Linux FIB) — principe et lookup

Exemple de préfixes: 0/0, 10/8, 10.1/16, 10.1.2/24, 192.168/16

Lookup illustré: 10.1.2.3 (chemin en cyan)



Idée clé : **compression des niveaux** (skip) + **strides variables** ⇒ moins de nœuds, lookup rapide.

Plan

3. Bases du routage

Activation du routage

Fichiers clés

- `/proc/sys/net/ipv4/ip_forward` : 0 = désactivé, 1 = activé
- `/etc/sysctl.conf` : configuration persistante (`net.ipv4.ip_forward=1`)

Astuce

Vérifier l'état avec : `cat /proc/sys/net/ipv4/ip_forward`

Configuration des interfaces réseau

Fichiers selon la distribution

- Debian/Ubuntu : `/etc/network/interfaces`
- Ubuntu récents : `/etc/netplan/*.yaml`
- RHEL / CentOS : `/etc/sysconfig/network-scripts/ifcfg-*`

Rôle

Ces fichiers définissent les adresses IP, masques et passerelles.

Routage et firewall

Table de routage

- `/proc/net/route` : table du noyau (format brut)
- `/etc/iproute2/rt_tables` : routage avancé (policy routing)

NAT et firewall

- `/etc/iptables/rules.v4` ou `rules.v6`
- `/etc/nftables.conf` (systèmes récents)

Activation du routage

1 pour activer, 0 pour désactiver

- **Temporairement** :

```
# IPv4
echo 1 | sudo tee /proc/sys/net/ipv4/ip_forward
# ou
sudo sysctl -w net.ipv4.ip_forward=1

# IPv6
echo 1 | sudo tee /proc/sys/net/ipv6/conf/all/forwarding
sudo sysctl -w net.ipv6.conf.all.forwarding=1
```

- **Permanent** (dans /etc/sysctl.conf) :

```
net.ipv4.ip_forward = 1
net.ipv6.conf.all.forwarding = 1
# appliquer les changements
sudo sysctl -p
```

Routage statique : exemples

```
# Ajouter une route réseau
sudo ip route add 192.168.2.0/24 via 192.168.1.1

# Ajouter une route vers un hôte spécifique
sudo ip route add 10.10.10.5 via 192.168.1.254

# Supprimer une route
sudo ip route del 192.168.2.0/24 via 192.168.1.1

# Afficher la table de routage
sudo ip route show
# ou
route -n
```

Attention

L'IP après `via` doit être directement atteignable par le routeur.

Table de routage et NAT

Table de routage

- Afficher : `ip route show`
- Ajouter une route : `ip route add <réseau> via <passerelle>`
- Définir une passerelle par défaut : `ip route add default via <IP_gateway>`

NAT (accès Internet pour le LAN)

- Indispensable si adresses internes privées.
- Exemple avec `iptables` : `iptables -t nat -A POSTROUTING -o eth1 -j MASQUERADE`

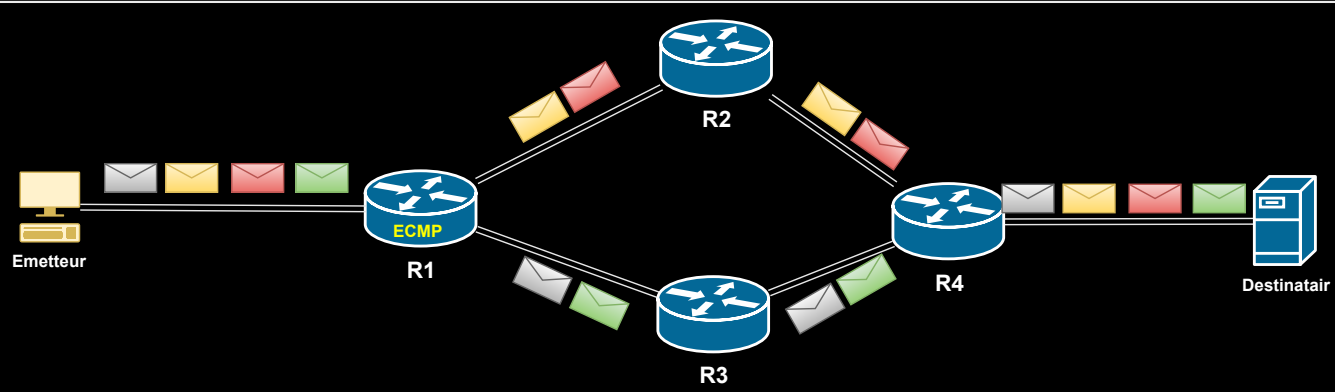
Plan

4. Routage avancé

Multipath et ECMP (Equal-Cost Multi-Path)

Principe

Le **multipath routing** permet d'utiliser plusieurs chemins entre une source et une destination.



ECMP : Equal-Cost Multi-Path

- Plusieurs routes ayant le **même coût** sont installées dans la table.
- Le noyau répartit le trafic entre ces chemins.

Multipath et ECMP (Equal-Cost Multi-Path)

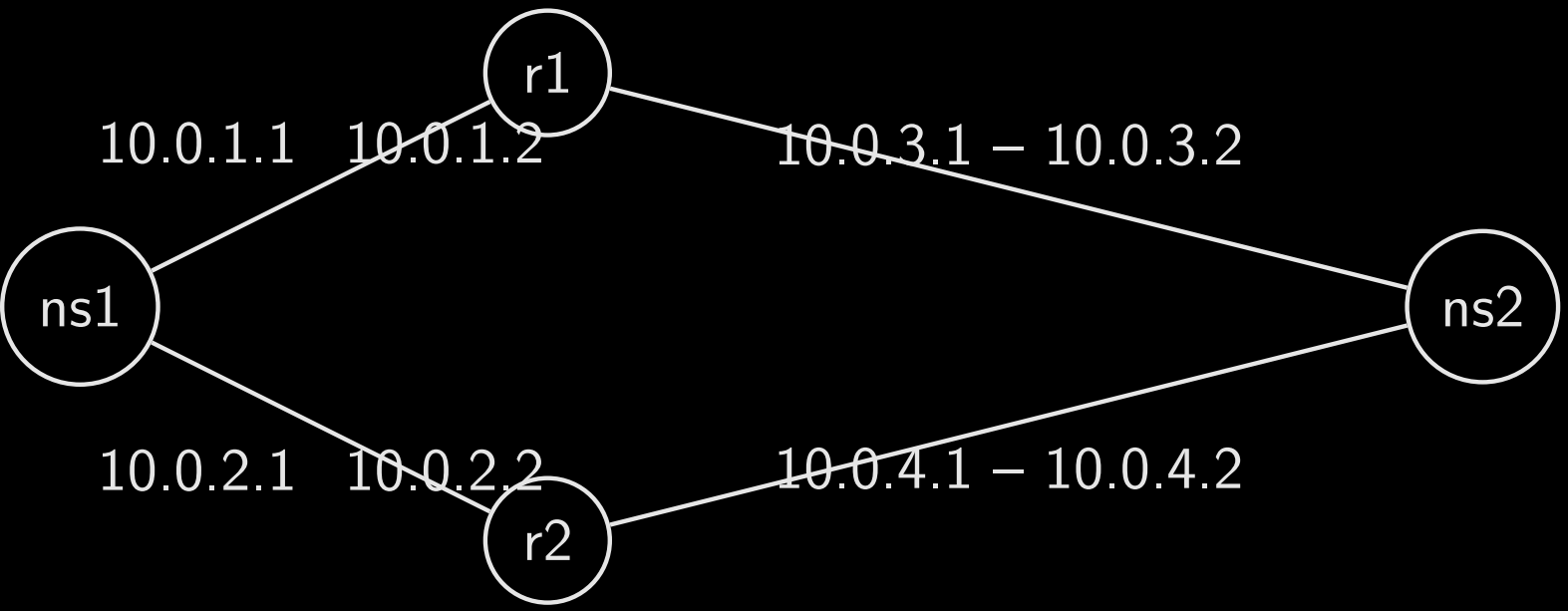
Bénéfices

- **Répartition de charge** (load balancing).
- **Résilience** : un chemin alternatif existe immédiatement en cas de panne.
- **Optimisation des ressources** réseau.

Limites

- Pas de prise en compte des différences de capacité entre liens.
- Hashing basé sur 5-tuple (src/dst IP, ports, protocole) $\Rightarrow$ peut créer des déséquilibres.

Multipath et ECMP - Exemple (1/5)



Multipath et ECMP - Exemple (2/5)

```
#!/bin/bash
set -e

# Nettoyage
ip -all netns del 2>/dev/null || true

# Création des namespaces
ip netns add ns1
ip netns add ns2
ip netns add r1
ip netns add r2

# Création des veth pairs
ip link add veth-ns1-r1 type veth peer name veth-r1-ns1
ip link add veth-ns1-r2 type veth peer name veth-r2-ns1
ip link add veth-r1-ns2 type veth peer name veth-ns2-r1
ip link add veth-r2-ns2 type veth peer name veth-ns2-r2
```

espace de nom réseau

carte virtuel a deux extrémitées

Multipath et ECMP - Exemple (3/5)

```
# Attacher aux namespaces
ip link set veth-ns1-r1 netns ns1
ip link set veth-ns1-r2 netns ns1
ip link set veth-r1-ns1 netns r1
ip link set veth-r2-ns1 netns r2
ip link set veth-r1-ns2 netns r1
ip link set veth-ns2-r1 netns ns2
ip link set veth-r2-ns2 netns r2
ip link set veth-ns2-r2 netns ns2

# Adressage IP
ip netns exec ns1 ip addr add 10.0.1.1/24 dev veth-ns1-r1
ip netns exec ns1 ip addr add 10.0.2.1/24 dev veth-ns1-r2
ip netns exec r1 ip addr add 10.0.1.2/24 dev veth-r1-ns1
ip netns exec r1 ip addr add 10.0.3.1/24 dev veth-r1-ns2
ip netns exec r2 ip addr add 10.0.2.2/24 dev veth-r2-ns1
ip netns exec r2 ip addr add 10.0.4.1/24 dev veth-r2-ns2
ip netns exec ns2 ip addr add 10.0.3.2/24 dev veth-ns2-r1
ip netns exec ns2 ip addr add 10.0.4.2/24 dev veth-ns2-r2
```

associer aux namespaces une carte virtuelle

définir les addresses IP

Multipath et ECMP - Exemple (4/5)

```
# Activer les interfaces
for ns in ns1 ns2 r1 r2; do
  ip netns exec $ns ip link set lo up
  ip netns exec $ns ip link set $(ip netns exec \
    $ns ip link show | awk -F: '/veth/{print $2}') up
done
# Activer le forwarding dans r1 et r2
ip netns exec r1 sysctl -w net.ipv4.ip_forward=1
ip netns exec r2 sysctl -w net.ipv4.ip_forward=1
# Routes multipath dans ns1
ip netns exec ns1 ip route add 10.0.3.0/24 \
    nexthop via 10.0.1.2 dev veth-ns1-r1 \
    nexthop via 10.0.2.2 dev veth-ns1-r2
ip netns exec ns1 ip route add 10.0.4.0/24 \
    nexthop via 10.0.1.2 dev veth-ns1-r1 \
    nexthop via 10.0.2.2 dev veth-ns1-r2
# Routes retour dans ns2
ip netns exec ns2 ip route add 10.0.1.0/24 via 10.0.3.1 dev veth-ns2-r1
ip netns exec ns2 ip route add 10.0.2.0/24 via 10.0.4.1 dev veth-ns2-r2
```

fonction routeur

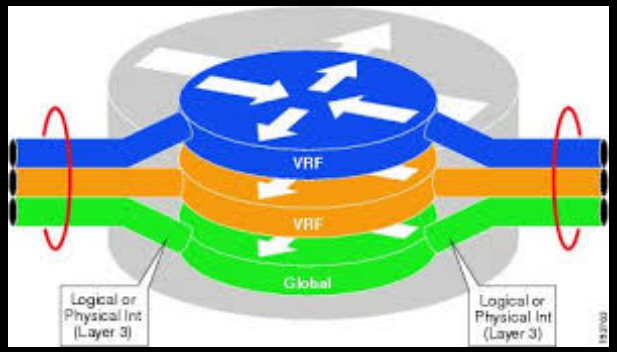
Multipath et ECMP - Exemple (5/5)

```
# Teste avec :
  ip netns exec ns1 ping -I 10.0.2.1 -c3 10.0.4.2
```

VRF (Virtual Routing and Forwarding)

Définition

Un **VRF** est une instance de table de routage séparée dans le même routeur ou hôte Linux.



Source : https://interpip.es/linux/creating-a-vrf-and-running-services-inside-it-on-linux/

Objectif

- Segmentation du trafic réseau sans matériel supplémentaire.
- Isolation logique entre clients, applications ou environnements.
- Similaire à des **VPN de niveau 3**.

VRF (Virtual Routing and Forwarding)

Cas d'usage

- Multi-tenancy (hébergement de plusieurs clients).
- Séparation trafic production / test.
- Opérateurs réseaux : routage client indépendant.

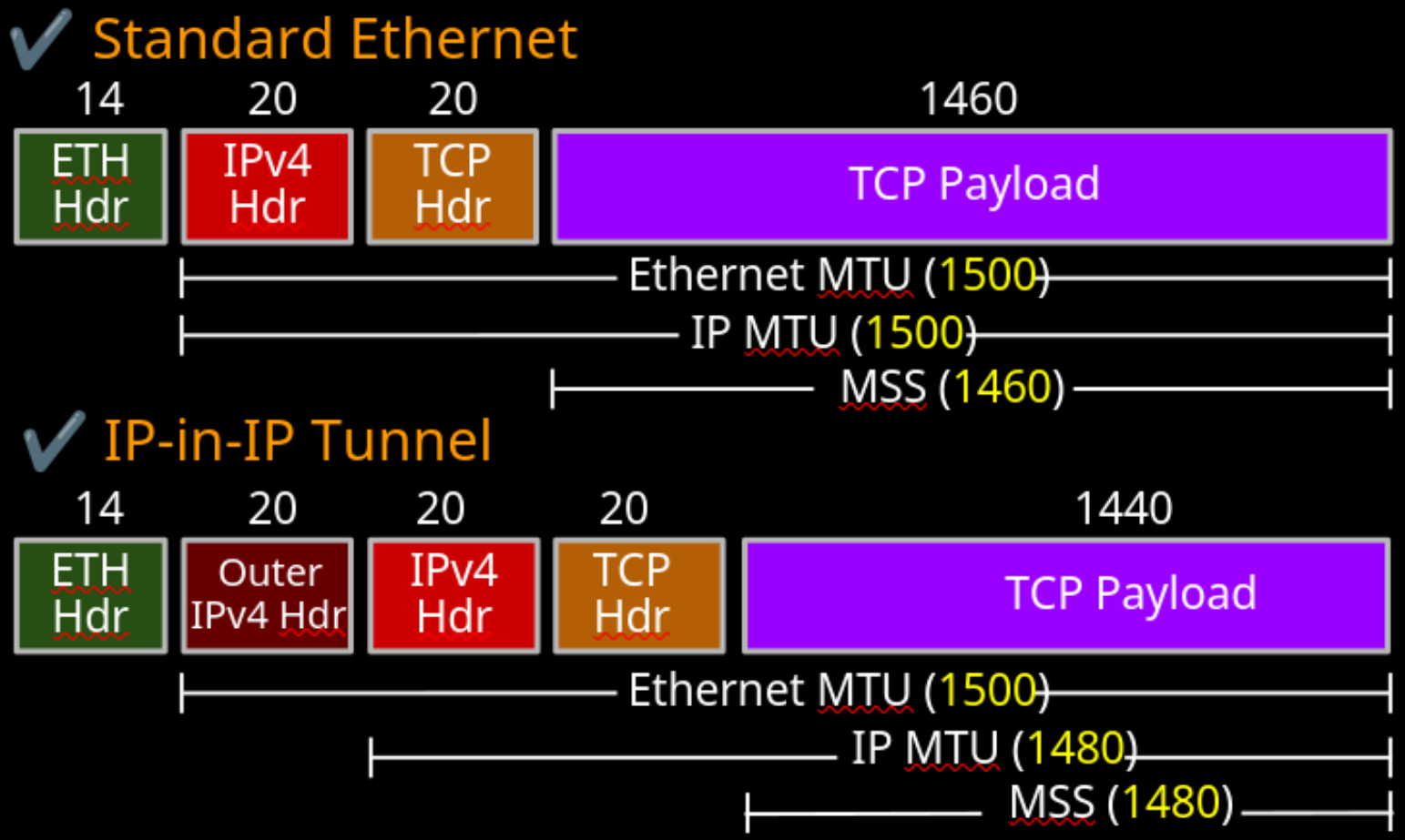
Sous Linux

Depuis le noyau 4.x, les VRFs sont pris en charge nativement :

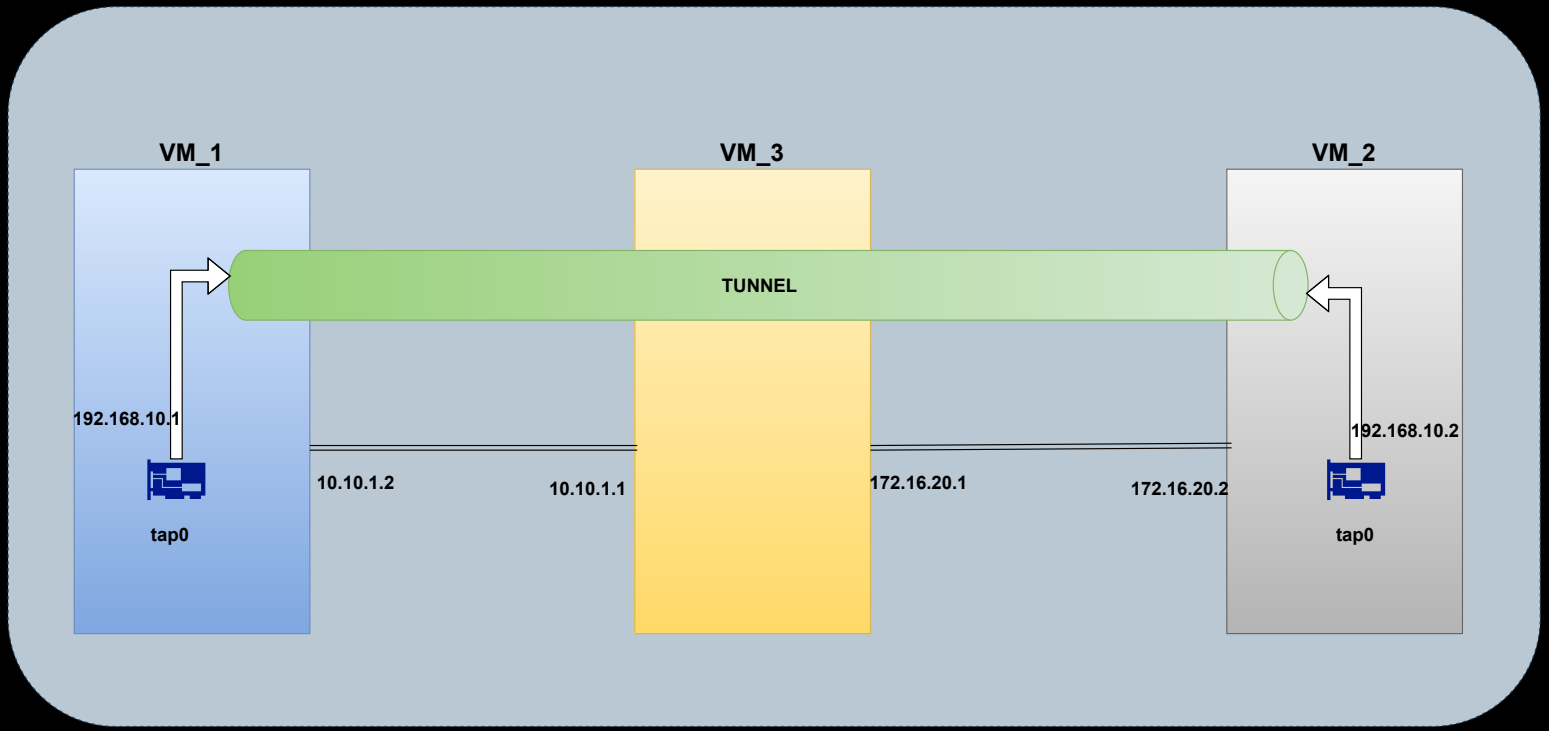
```
ip link add vrf-red type vrf table 100
```

Pour aller loin: https://thelinuxchannel.org/2023/10/vrf-virtual-routing-and-forwarding-introduction/

Encapsulation IP-in-IP



Encapsulation IP-in-IP - Exemples



Encapsulation IP-in-IP - Exemple (1/3)

```
#!/bin/bash

# Cleanup + namespaces
ip netns del host1 2>/dev/null
ip netns del host2 2>/dev/null
ip netns del internet 2>/dev/null

ip netns add host1
ip netns add host2
ip netns add internet

# Create the topology
ip link add veth0 type veth peer name veth1
ip link add veth2 type veth peer name veth3

ip link set veth0 netns host1
ip link set veth1 netns internet
ip link set veth2 netns internet
ip link set veth3 netns host2
```

Encapsulation IP-in-IP - Exemple (2/3)

```
# Host1
ip netns exec host1 ip addr add 10.10.1.2/24 dev veth0
ip netns exec host1 ip link set veth0 up
ip netns exec host1 ip link set lo up

# Host2
ip netns exec host2 ip addr add 172.16.20.2/24 dev veth3
ip netns exec host2 ip link set veth3 up
ip netns exec host2 ip link set lo up

# Internet
ip netns exec internet ip addr add 10.10.1.1/24 dev veth1
ip netns exec internet ip link set veth1 up
ip netns exec internet ip addr add 172.16.20.1/24 dev veth2
ip netns exec internet ip link set veth2 up
ip netns exec internet ip link set lo up
ip netns exec internet sysctl -w net.ipv4.ip_forward=1
```

Encapsulation IP-in-IP - Exemple (3/3)

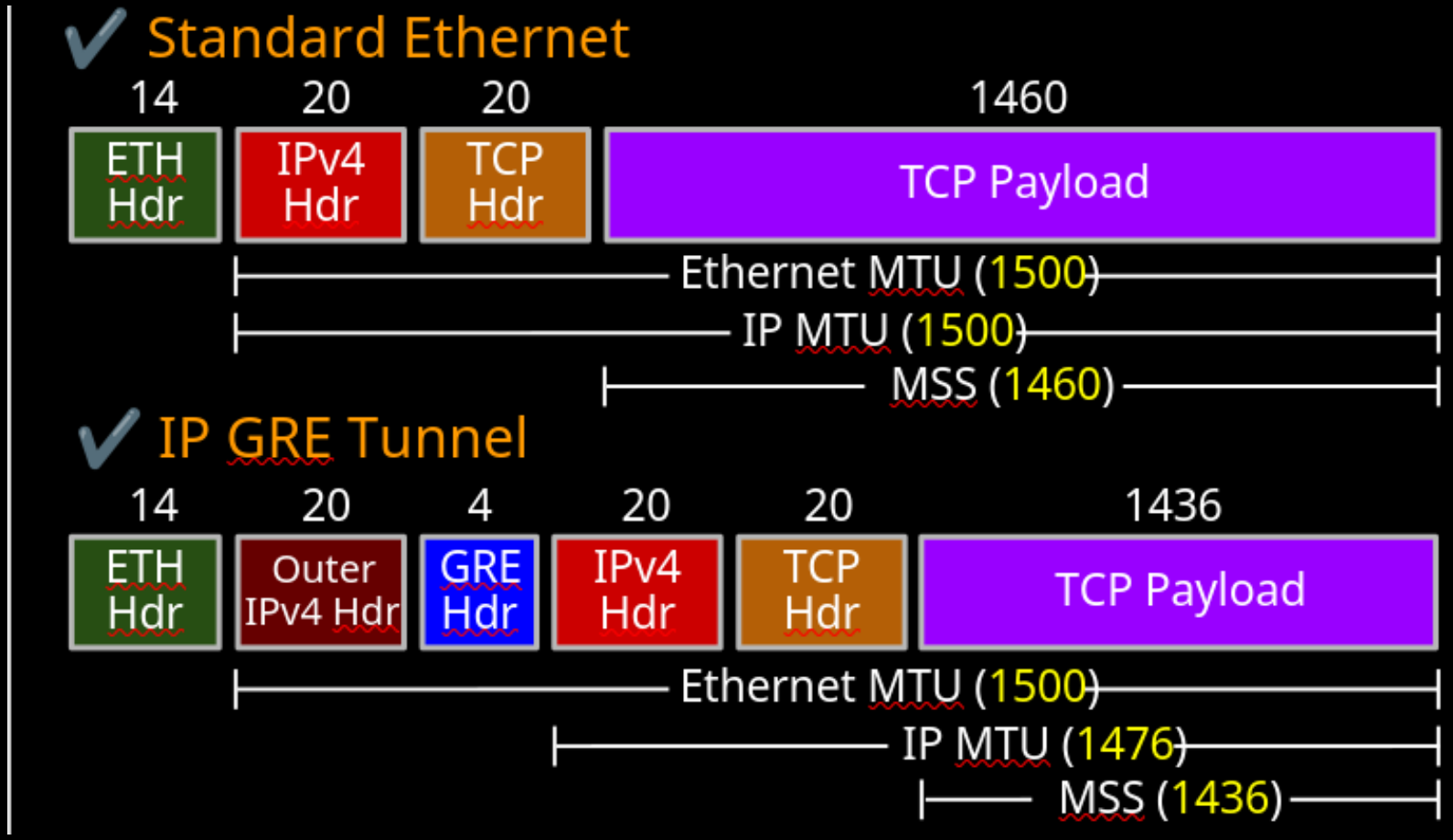
```
# Add routes so tunnel endpoints can talk
ip netns exec host1 ip route add 172.16.20.0/24 via 10.10.1.1
ip netns exec host2 ip route add 10.10.1.0/24 via 172.16.20.1

# IPIP tunnel host1
ip netns exec host1 ip tunnel add tap0 mode ipip \
  local 10.10.1.2 remote 172.16.20.2 ttl 255
ip netns exec host1 ip addr add 192.168.10.1/30 dev tap0
ip netns exec host1 ip link set tap0 up

# IPIP tunnel host2
ip netns exec host2 ip tunnel add tap0 mode ipip \
  local 172.16.20.2 remote 10.10.1.2 ttl 255
ip netns exec host2 ip addr add 192.168.10.2/30 dev tap0
ip netns exec host2 ip link set tap0 up

exit 0
```

Encapsulation GRE



Encapsulation GRE - Exemple

```
    #!/bin/bash
# Create the namespaces
ip netns add host1
ip netns add host2
ip netns add internet

# Create the topology
ip link add veth0 type veth peer name veth1
ip link add veth2 type veth peer name veth3

ip link set veth0 netns host1
ip link set veth1 netns internet
ip link set veth2 netns internet
ip link set veth3 netns host2

# Host1
ip netns exec host1 ip addr add 10.10.1.2/24 dev veth0
ip netns exec host1 ip link set veth0 up
ip netns exec host1 ip link set lo up

# Host2
ip netns exec host2 ip addr add 172.16.20.2/24 dev veth3
```

Encapsulation GRE - Exemple (1/3)

```
#!/bin/bash

# Cleanup + namespaces
ip netns del host1 2>/dev/null
ip netns del host2 2>/dev/null
ip netns del internet 2>/dev/null

ip netns add host1
ip netns add host2
ip netns add internet

# Create the topology
ip link add veth0 type veth peer name veth1
ip link add veth2 type veth peer name veth3

ip link set veth0 netns host1
ip link set veth1 netns internet
ip link set veth2 netns internet
ip link set veth3 netns host2
```

Encapsulation GRE - Exemple (2/3)

```
# Host1
ip netns exec host1 ip addr add 10.10.1.2/24 dev veth0
ip netns exec host1 ip link set veth0 up
ip netns exec host1 ip link set lo up

# Host2
ip netns exec host2 ip addr add 172.16.20.2/24 dev veth3
ip netns exec host2 ip link set veth3 up
ip netns exec host2 ip link set lo up

# Internet
ip netns exec internet ip addr add 10.10.1.1/24 dev veth1
ip netns exec internet ip link set veth1 up
ip netns exec internet ip addr add 172.16.20.1/24 dev veth2
ip netns exec internet ip link set veth2 up
ip netns exec internet ip link set lo up
ip netns exec internet sysctl -w net.ipv4.ip_forward=1
```

Encapsulation GRE - Exemple (3/3)

```
# Add routes
ip netns exec host1 ip route add 172.16.20.0/24 via 10.10.1.1
ip netns exec host2 ip route add 10.10.1.0/24 via 172.16.20.1

# GRE tunnel host1
ip netns exec host1 ip tunnel add gre1 mode gre \
  local 10.10.1.2 remote 172.16.20.2 ttl 255
ip netns exec host1 ip addr add 192.168.10.1/30 dev gre1
ip netns exec host1 ip link set gre1 up

# GRE tunnel host2
ip netns exec host2 ip tunnel add gre1 mode gre \
  local 172.16.20.2 remote 10.10.1.2 ttl 255
ip netns exec host2 ip addr add 192.168.10.2/30 dev gre1
ip netns exec host2 ip link set gre1 up

exit 0
```

Plan

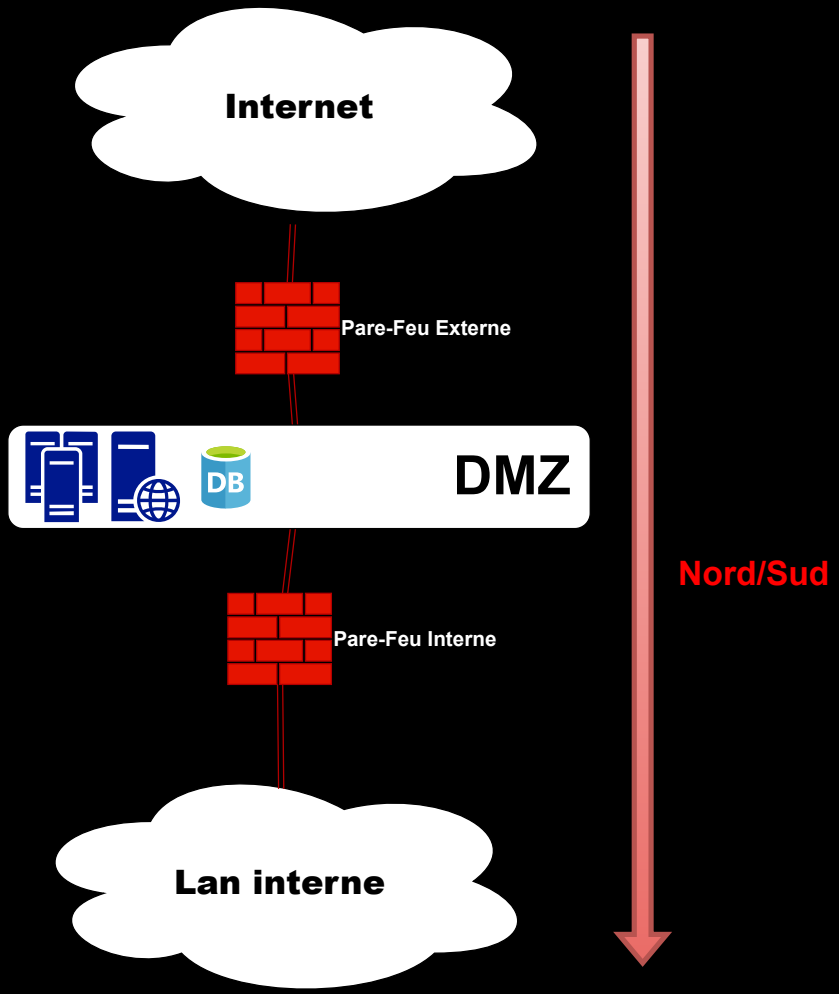
Les Fondamentaux des Pare-Feux

- **Rôle du pare-feu** : Contrôler et filtrer le trafic réseau. Il agit comme un point de contrôle entre des zones de confiance (ex: LAN) et des zones non fiables (ex: Internet).
- **Fonctionnement** : Inspection des paquets, application de règles de filtrage.

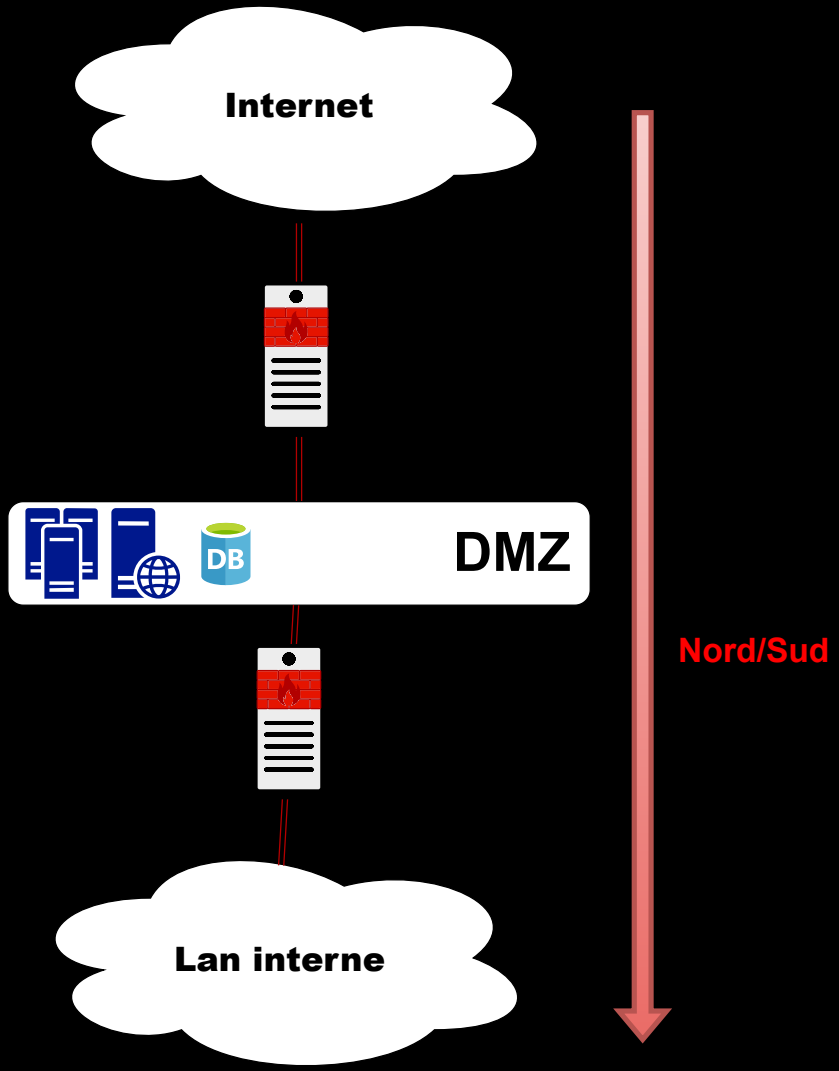
Pare-Feux - Types de Pare-Feux

- **Filtrage de paquets** : Agit aux couches 3 (IP) et 4 (TCP/UDP). Rapide mais simple.
- **Stateful Inspection** : Maintient un état de chaque connexion. Ne laisse passer les paquets "en retour" que si une connexion a été initiée de manière légitime.
- **Relais Applicatif (Proxy)** : Opère à la couche 7. Inspection du contenu des requêtes (ex: URL, en-têtes HTTP). Plus lent mais plus sécurisé.

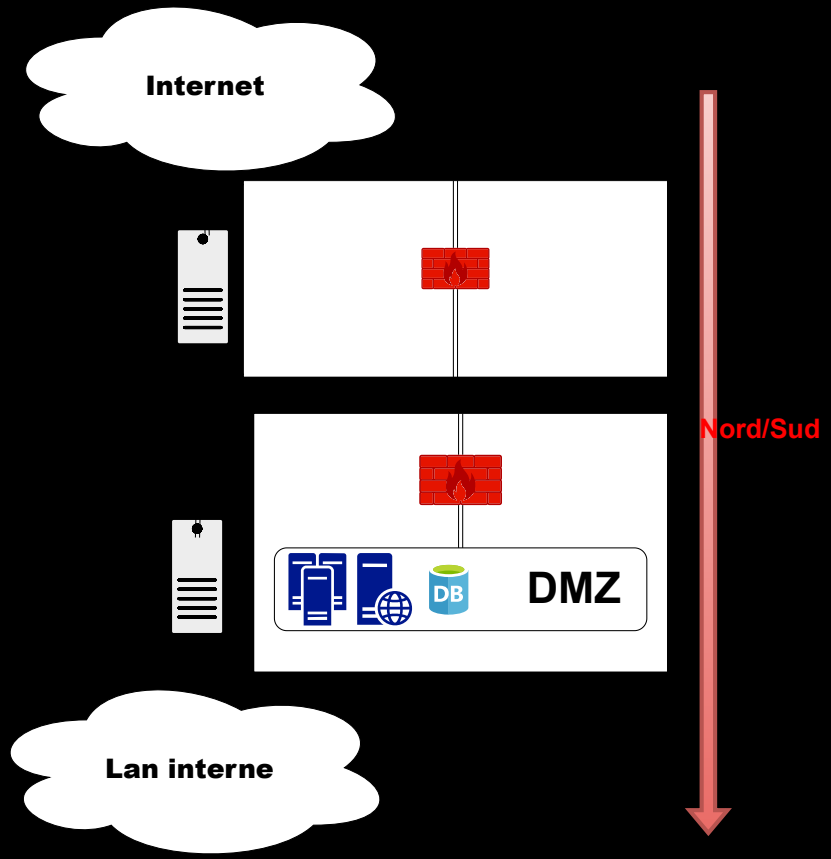
Pare-Feu - principe



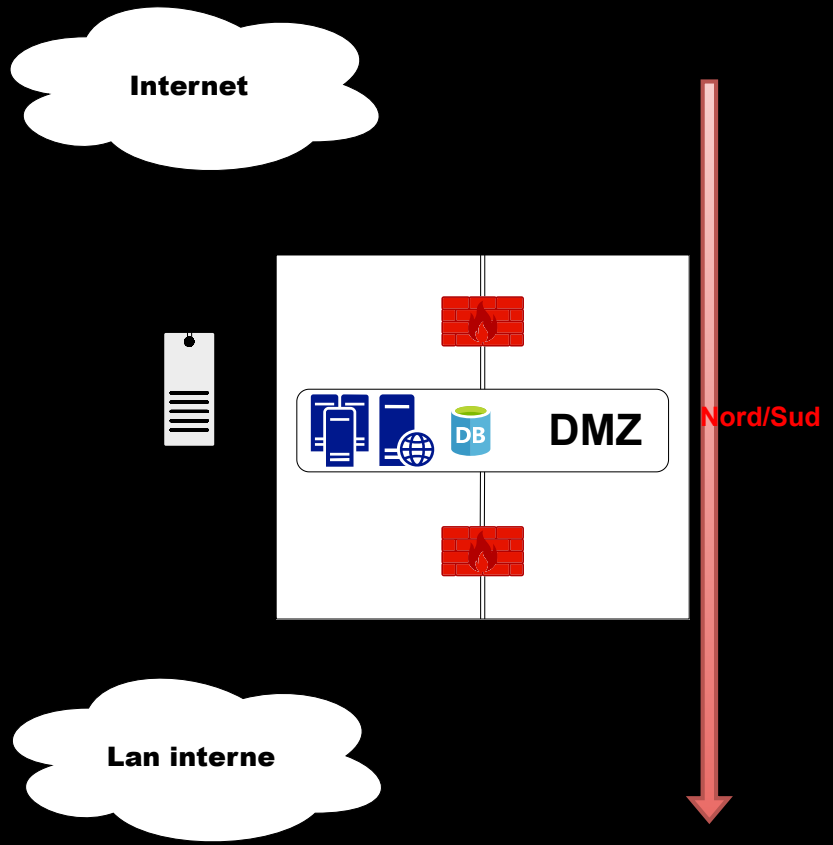
Pare-Feu - virtualisé



Pare-Feu - virtualisé - mutualisé partielle



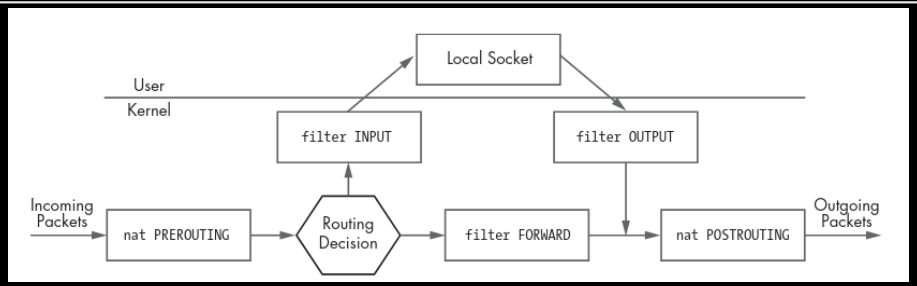
Pare-Feu - virtualisé - mutualisé totale



Introduction à iptables

Qu'est-ce que iptables ?

- `iptables` est un outil en ligne de commande permettant de configurer le pare-feu intégré au noyau Linux (Netfilter).
- Il est utilisé pour filtrer et contrôler le trafic réseau sur une machine.
- Sa fonction principale est la sécurisation du réseau à travers la gestion des paquets IP.



Concepts clés d'iptables

Paquets et filtrage

- **Paquets réseau** : unités de données échangées sur un réseau.
- **Filtrage** : bloquer ou autoriser des paquets en fonction de règles.

Tables et chaînes

- **Tables** : contiennent des règles pour différents types de traitement des paquets.
- **Chaînes** : chaque table possède plusieurs chaînes où les règles sont appliquées.

Les tables dans iptables

Tables courantes

- **filter** : Table par défaut pour le filtrage des paquets.
- **nat** : Pour la traduction d'adresses réseau (NAT).
- **mangle** : Pour modifier les paquets (par exemple, le TTL).
- **raw** : Exempte certains paquets du suivi de connexion (conntrack).

Les chaînes dans iptables

Chaînes principales

- **INPUT** : Gère les paquets entrants vers la machine locale.
- **OUTPUT** : Gère les paquets sortants de la machine locale.
- **FORWARD** : Gère les paquets routés à travers la machine.
- **PREROUTING** : Modifie les paquets avant le routage.
- **POSTROUTING** : Modifie les paquets après le routage.

Syntaxe de base des commandes

Lister les règles existantes

`sudo iptables -L` : Lister les règles de la table par défaut (`filter`).
`sudo iptables -t nat -L` : Lister les règles de la table `nat`.

Ajouter une règle

```
sudo iptables -A INPUT -p tcp -dport 80 -j ACCEPT
```

- Ajoute une règle pour accepter le trafic HTTP (port 80).

Supprimer une règle

```
sudo iptables -D INPUT 1
```

- Supprime la première règle de la chaîne `INPUT`.

Gestion des adresses IP et NAT

Traduction d'adresses réseau (NAT)

`sudo iptables -t nat -A POSTROUTING -o eth0 -j MASQUERADE`

- Utilisé pour masquer les adresses internes lors de la sortie vers Internet.

Filtrage de paquets

`sudo iptables -P INPUT DROP` : Bloque tout le trafic entrant par défaut. `sudo iptables -A INPUT -p tcp -dport 22 -j ACCEPT` : Autorise SSH (port 22).

Sauvegarde et persistance des règles

Sauvegarder les règles

```
sudo iptables-save > /etc/iptables/rules.v4
```

- Sauvegarde les règles actuelles dans un fichier.

Restaurer les règles

```
sudo iptables-restore < /etc/iptables/rules.v4
```

- Charge les règles sauvegardées depuis un fichier.

Exercice pratique

Configuration d'un pare-feu basique

- **Bloquer tout le trafic par défaut** :

```
sudo iptables -P INPUT DROP
```

- **Autoriser SSH et HTTP** :

```
sudo iptables -A INPUT -p tcp -dport 22 -j ACCEPT
sudo iptables -A INPUT -p tcp -dport 80 -j ACCEPT
```

- **Sauvegarder les règles** :

```
sudo iptables-save > /etc/iptables/rules.v4
```

De iptables à nftables

Pourquoi nftables ?

- `nftables` est le successeur d'`iptables`, introduit avec Linux 3.13.
- Objectif : unifier et simplifier la gestion du pare-feu dans le noyau Linux.
- Remplace `iptables`, `ip6tables`, `arptables` et `ebtables`.

Avantages de nftables

- Syntaxe plus simple et homogène.
- Meilleures performances grâce à un moteur commun.
- Gestion dynamique des règles via `nft`.

Syntaxe de base avec nftables

Exemple : autoriser SSH et HTTP

- Créer une table et une chaîne :

```
sudo nft add table inet filter
sudo nft add chain inet filter input { type filter
hook input priority 0; }
```

- Ajouter des règles :

```
sudo nft add rule inet filter input tcp dport 22
accept
sudo nft add rule inet filter input tcp dport 80
accept
```

Lister les règles

```
sudo nft list ruleset
```

NAT dans iptables et nftables

Concepts clés

- **SNAT (Source NAT)** : modifier l'adresse source (ex. accès Internet depuis LAN privé).
- **DNAT (Destination NAT)** : rediriger vers une autre IP/port (ex. serveur web interne).
- **MASQUERADE** : variante dynamique du SNAT, utile pour les connexions sortantes via DHCP.

Exemples iptables

```
sudo iptables -t nat -A POSTROUTING -o eth0 -j
MASQUERADE
sudo iptables -t nat -A PREROUTING -p tcp -dport 8080 -j
DNAT -to-destination 192.168.1.10:80
```

Forwarding et routage

Activation du forwarding

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

- Permet à la machine de router des paquets entre interfaces.
- Utile pour transformer un serveur en passerelle/pare-feu.

Filtrage du trafic routé

- Chaîne `FORWARD` dans iptables.
- Exemple : autoriser le trafic entre LAN et Internet :

```
sudo iptables -A FORWARD -i eth1 -o eth0 -j ACCEPT
sudo iptables -A FORWARD -i eth0 -o eth1 -m state
-state ESTABLISHED,RELATED -j ACCEPT
```

Résumé : iptables vs nftables

iptables

- Outil historique, encore largement utilisé.
- Syntaxe plus verbeuse et répartie en plusieurs utilitaires.

nftables

- Successeur officiel, syntaxe unifiée.
- Meilleures performances et flexibilité.
- Remplace progressivement iptables dans les distributions récentes.