



MASTER 1 CRYPTIS - INFORMATIQUE
FACULTÉ DES SCIENCES ET TECHNIQUES

Projet/ Infrastructure Réseaux

TUNNEL L2TPv3 SÉCURISÉ PAR IPSEC POUR LA MISE EN ŒUVRE DE TRUNKING DE VLANs COMPARAISON AVEC LES VXLAN

Encadré par :
Pr. Pierre-François BONNEFOI
Pr. Emmanuel CONCHON

Réalisé par :
Lansana DIAKITE
Ruslan KASHEPAROV

Année Universitaire 2023 - 2024

Contents

1	Introduction	2
2	Architecture du Réseau	2
2.1	Architecture de Départ	2
2.2	Architecture avec VLANS	2
2.3	Test de l'encapsulation VLAN	3
3	Protocole L2TPv3 et la Technologie des VXLANs	4
3.1	L2TPv3	4
3.2	VxLANS	4
3.3	Comparaison entre les deux technologies :	4
3.4	Mise en place dans Linux des VXLANs dans Open vSwitch	5
3.5	Solutions de chiffrement du trafic L2TPv3 ou VXLAN	5
3.6	Comparaison avec MPLS	5
4	Établissement du tunnel L2TPv3	6
4.1	Configuration du tunnel en mode encapsulation IP	6
4.2	Test du Protocole ARP	6
4.3	Test du Service DHCP sur le Routeur 1	8
4.4	Connexion SOCAT entre le Routeur1 et le Poste 1	9
5	Mise en oeuvre de L2TPV3 en mode Encapsulation IP et UDP	11
5.1	Comparaison du mode IP et UDP	11
5.2	Comparaison avec un tunnel GRE en mode GRETAP	12
5.3	Comparaison des MTUs	13
6	Chiffrement IPSEC sur le tunnel L2TPV3 en mode IP	14
6.1	Mise en place du chiffrement IPsec:	14
6.2	Comparaison via iperf de la vitesse de transport avec et sans IPsec	15
6.3	Capture du trafic L2TPV3 en mode chiffrement IPsec	16
7	Accès Internet	17
7.1	Configuration du Switch Internet Et la machine réelle	17
7.2	Accès à Internet via Routeur 1:	17
7.3	Verification du traffic Internet de P1 passant R1	18
7.4	Passage de P1 et P2 Via R2 Pour Internet à l'aide de dnsmaq	19
7.5	Test de l'Accès à Internet de P1 via le Routeur 2	20
8	Interdiction du trafic entre VLAN 100 et VLAN 200	21
8.1	Avec iptables:	21
8.2	A l'aide de la Policy Routing:	22
9	Organisation du Projet:	23
10	Conclusion	24
11	References	24

1 Introduction

Ce projet vise à explorer le protocole L2TPv3 pour la création de tunnels de niveau 2. Le tunnel ainsi établi servira à faire du "Trunking" de VLANs et à partager des services tels que le DHCP. Nous envisageons également plusieurs configurations, offrant la possibilité de choisir entre une encapsulation IP ou UDP, ainsi que de déployer des tunnels L2TPv3 ou GRE selon les besoins spécifiques du réseau. Pour renforcer la sécurité des communications, une couche de chiffrement via IPsec sera intégrée au tunnel. De plus, une configuration intelligente sera mise en place pour réguler l'utilisation du tunnel lorsque les hôtes souhaitent accéder à Internet. Cette configuration permettra aux hôtes de se connecter directement depuis leur côté "Internet", réduisant ainsi la dépendance du tunnel pour les activités en ligne.

Nous implémenterons également des mesures de sécurité afin de prévenir toute communication non autorisée entre les VLANs. en utilisant des règles de Firewall et de Policy Routing.

2 Architecture du Réseau

En nous appuyant sur les techniques apprises lors des travaux pratiques antérieurs et en utilisant les *netns* et *OpenvSwitch*, nous avons mis en place les réseaux selon les schémas fournis dans l'énoncé.

2.1 Architecture de Départ

Après avoir écrit le script pour le réseau tel que vu sur le schéma 1, nous avons utilisé le script Python **network_graph.py** fourni pour générer une image du réseau créé, comme présenté sur la figure ci-dessous[1]:

Script : build_architecture

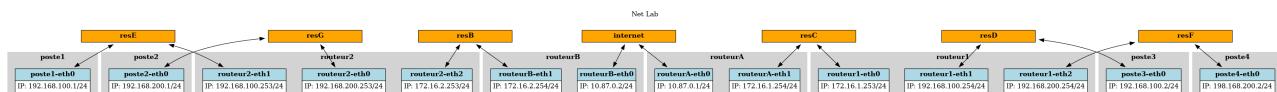


Fig. 1. Image du réseau 1

2.2 Architecture avec VLANs

Dans la mise en place du schéma 2 du réseau, intégrant des VLANs, nous avons initialement établi la configuration réseau comme avec l'architecture de départ. Ensuite, pour l'intégration des VLANs de ports, nous avons associé les VLANs 100 et 200 aux interfaces eth1 des Routeur1 et Routeur2. De même, nous avons configuré les postes 1 et 3 pour utiliser le VLAN 100, tandis que les postes 2 et 4 ont été attribués au VLAN 200, conformément au schéma fourni.

Table 1. Récapitulatif de la configuration des VLANs

Périphérique	VLAN	
	Eth1	Eth1.X
Routeur1	Eth1	Eth1.100, Eth1.200
Routeur2	Eth1	Eth1.100, Eth1.200
Poste1	-	Eth1.100
Poste2	-	Eth1.200
Poste3	-	Eth1.100
Poste4	-	Eth1.200

Ci-dessous [2] la figure obtenue après l'exécution **network_graph.py** sur le réseau. **Script : script_Vlans**

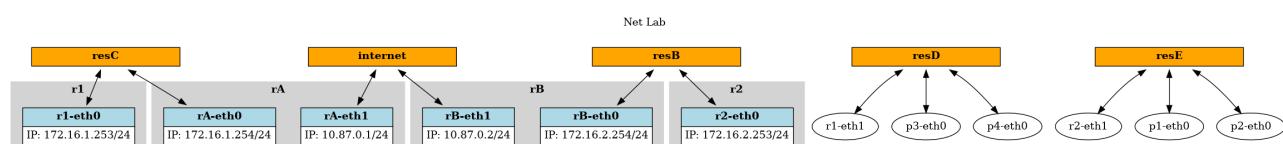


Fig. 2. Image du Réseau 2 * VLANs

2.3 Test de l'encapsulation VLAN

Afin de tester le bon fonctionnement de l'**encapsulation VLAN**, nous avons lancé une capture du trafic réseau en utilisant `tcpdump` sur l'interface `r1-eth1` du routeur `r1`.

```
sudo ip netns exec r1 tcpdump -lnvv -i r1-eth1 -e vlan
```

Maintenant, nous allons lancer un ping entre les postes **p3** et **p4**. D'abord voici la configuration IP et addresses MAC de P3, R1 et P4

```
ubuntu@ubuntu2004:~/infra_res/projet$ sudo ip netns exec p3 ip -br address show
lo          UNKNOWN      127.0.0.1/8 ::1/128
p3-eth0.100@p3-eth0 UP           192.168.100.2/24 fe80::90c7:1aff:fe08:f568/64
p3-eth0@if37    UP           fe80::90c7:1aff:fe08:f568/64
ubuntu@ubuntu2004:~/infra_res/projet$ sudo ip netns exec p4 ip -br address show
lo          UNKNOWN      127.0.0.1/8 ::1/128
p4-eth0.200@p4-eth0 UP           192.168.200.2/24 fe80::ccb7:8eff:fe10:9a77/64
p4-eth0@if39    UP           fe80::ccb7:8eff:fe10:9a77/64
ubuntu@ubuntu2004:~/infra_res/projet$ sudo ip netns exec r1 ip -br address show
lo          UNKNOWN      127.0.0.1/8 ::1/128
r1-eth1.100@r1-eth1 UP           192.168.100.254/24 fe80::6816:6ff:fe2b:a2d9/64
r1-eth1.200@r1-eth1 UP           192.168.200.254/24 fe80::6816:6ff:fe2b:a2d9/64
r1-eth0@if17    UP           172.16.1.253/24 fe80::2c8a:13ff:fe6f:ce8e/64
r1-eth1@if19    UP           fe80::6816:6ff:fe2b:a2d9/64
```

Fig. 3. Configuration IP de p3, p4 et r1

```
ubuntu@ubuntu2004:~/infra_res/projet$ sudo ip netns exec p3 ip link show p3-eth0
67: p3-eth0@if66: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue state UP mode DEFAULT group default qlen 1000
    link/ether 72:c1:ae:5a:6d:a0 brd ff:ff:ff:ff:ff:ff link-netnsid 0
ubuntu@ubuntu2004:~/infra_res/projet$ sudo ip netns exec p4 ip link show p4-eth0
69: p4-eth0@if68: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue state UP mode DEFAULT group default qlen 1000
    link/ether 0e:90:2f:44:c4:bc brd ff:ff:ff:ff:ff:ff link-netnsid 0
ubuntu@ubuntu2004:~/infra_res/projet$ sudo ip netns exec r1 ip link show r1-eth1
49: r1-eth1@if48: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue state UP mode DEFAULT group default qlen 1000
    link/ether de:b2:07:c5:8e:b8 brd ff:ff:ff:ff:ff:ff link-netnsid 0
```

Fig. 4. Adresses MAC de p3, p4 et r1

Table 2. Adresses MAC et IP des interfaces

Hôte	Adresse IP	Adresse MAC
P3	192.168.100.2/24	72:c1:ae:5a:6d:a0
P4	192.168.200.2/24	0e:90:2f:44:c4:bc
R1	192.168.100.254/24 192.168.200.254/24	de:b2:07:c5:8e:b8

```
ubuntu@ubuntu2004:~/infra_res/projet$ sudo ip netns exec r1 tcpdump -l -v -i r1-eth1 -e vlan
tcpdump: listening on r1-eth1, link-type EN10MB (Ethernet), capture size 262144 bytes
06:42:25.865568 72:c1:ae:5a:6d:a0 > ff:ff:ff:ff:ff:ff, ethertype 802.1Q (0x8100), length 46: vlan 100, p 0, ethertype ARP, Ethernet (len 6), IPv4 (len 4),
Request who-has 192.168.100.254 tell 192.168.100.2, length 28
06:42:25.865653 de:b2:07:c5:8e:b8 > 72:c1:ae:5a:6d:a0, ethertype 802.1Q (0x8100), length 46: vlan 100, p 0, ethertype ARP, Ethernet (len 6), IPv4 (len 4),
Reply 192.168.100.254 ls-at de:b2:07:c5:8e:b8, length 28
06:42:25.865671 72:c1:ae:5a:6d:a0 > de:b2:07:c5:8e:b8, ethertype 802.1Q (0x8100), length 102: vlan 100, p 0, ethertype IPv4, (tos 0x0, ttl 64, id 2772, offset 0, flags [DF], proto ICMP (1), length 84)
    192.168.100.2 > 192.168.200.2: ICMP echo request, id 12121, seq 1, length 64
06:42:25.866194 de:b2:07:c5:8e:b8 > 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 192.168.200.2 tell 192.168.200.254, length 28
06:42:25.866396 0e:90:2f:44:c4:bc > de:b2:07:c5:8e:b8, ethertype 802.1Q (0x8100), length 46: vlan 200, p 0, ethertype ARP, Ethernet (len 6), IPv4 (len 4),
Reply 192.168.200.2 ls-at 0e:90:2f:44:c4:bc, length 28
06:42:25.866617 de:b2:07:c5:8e:b8 > 0e:90:2f:44:c4:bc, ethertype 802.1Q (0x8100), length 102: vlan 200, p 0, ethertype IPv4, (tos 0x0, ttl 63, id 2772, offset 0, flags [DF], proto ICMP (1), length 84)
    192.168.100.2 > 192.168.200.2: ICMP echo request, id 12121, seq 1, length 64
06:42:25.867438 0e:90:2f:44:c4:bc > de:b2:07:c5:8e:b8, ethertype 802.1Q (0x8100), length 102: vlan 200, p 0, ethertype IPv4, (tos 0x0, ttl 64, id 28556, offset 0, flags [none], proto ICMP (1), length 84)
    192.168.100.2 > 192.168.100.2: ICMP echo reply, id 12121, seq 1, length 64
06:42:25.867490 de:b2:07:c5:8e:b8 > 72:c1:ae:5a:6d:a0, ethertype 802.1Q (0x8100), length 102: vlan 100, p 0, ethertype IPv4, (tos 0x0, ttl 63, id 28556, offset 0, flags [none], proto ICMP (1), length 84)
    192.168.100.2 > 192.168.100.2: ICMP echo reply, id 12121, seq 1, length 64
06:42:26.865522 72:c1:ae:5a:6d:a0 > de:b2:07:c5:8e:b8, ethertype 802.1Q (0x8100), length 102: vlan 100, p 0, ethertype IPv4, (tos 0x0, ttl 64, id 2983, offset 0, flags [DF], proto ICMP (1), length 84)
```

Fig. 5. Capture du traffic entre P3 et P4 sur R1

Dans cette capture de trafic ci-dessus 5 qui est un PING ICMP entre P3 et P4, nous observons tout d'abord une requête ARP émise par l'hôte P3 (192.168.100.2) appartenant au VLAN 100. Son adresse MAC, 72:c1:ae:5a:6d:a0, sollicite l'adresse MAC du routeur r1-eth1 à l'adresse 192.168.100.254, également dans le VLAN 100. Nous assistons ensuite à la réponse ARP du routeur, confirmant l'adresse MAC demandée. Par la suite, le routeur émet une requête ARP pour découvrir l'adresse MAC de l'hôte P4 (192.168.200.2) dans le VLAN 200, dont l'adresse MAC est 0e:90:2f:44:c4:bc. Nous constatons que le paquet ICMP envoyé par P3 parvient à destination, suivi de l'echo reply. Les paquets ICMP échangés entre P3 et P4 traversent le routeur R1 via les VLANs 100 et 200 et les étiquettes VLAN présentes dans les paquets confirment le bon fonctionnement de l'encapsulation VLAN.

3 Protocole L2TPv3 et la Technologie des VXLANs

3.1 L2TPv3

Le protocole L2TPv3 (*Layer 2 Tunneling Protocol Version 3*) est une technologie de tunnelling de couche 2 utilisée pour transporter le trafic de couche 2 sur un réseau IP. Il encapsule les trames Ethernet ou d'autres trames de niveau 2 dans des paquets IP, ce qui permet leur transmission à travers le réseau. Cette solution est souvent employée pour connecter différents réseaux étendus (WAN), tels que des succursales ou des centres de données distants. Les tunnels L2TPv3 peuvent être établis entre des routeurs, des commutateurs ou d'autres périphériques réseau, permettant ainsi la transmission transparente du trafic de couche 2 entre eux.

3.2 VXLANs

D'un autre côté, la technologie des VXLANs (*Virtual eXtensible LAN*) est un protocole de tunnelling qui encapsule le trafic de couche 2 Ethernet sur un réseau de couche 3 IP. Il s'agit d'une extension des réseaux locaux virtuels (VLAN) et d'un protocole de couche d'application basé sur UDP qui fonctionne sur le port 4789. Les VXLANs surmontent les limitations des réseaux traditionnels de couche 2, telles que les problèmes liés au spanning tree, le nombre limité de VLANs et les grandes tables d'adresses MAC. Ils utilisent un identifiant de réseau logique de 24 bits, ce qui permet d'augmenter le nombre de VLANs et d'isoler davantage le réseau logique pour les grands réseaux comme les centres de données. La technologie VXLAN permet de créer jusqu'à 16 millions de segments VXLAN dans un domaine administratif, ce qui facilite la migration des machines virtuelles entre des serveurs situés dans des domaines de couche 2 distincts.

3.3 Comparaison entre les deux technologies :

En comparaison, le protocole L2TPv3 encapsule les trames Ethernet dans des paquets IP, tandis que les VXLANs encapsulent les adresses MAC dans des paquets UDP.

L2TPv3 est principalement conçu pour les déploiements WAN, tandis que les VXLANs sont mieux adaptés pour les déploiements en Data Center en raison de leur évolutivité et de leurs fonctionnalités avancées, telles que la prise en charge d'un plus grand nombre de VLANs et la possibilité d'isoler le trafic avec des réseaux virtuels (VRF). Les VLANs offrent une capacité de 4096 machines au maximum par VLAN contrairement à VXLAN qui permet d'avoir plus de machines dans une même VLAN. En résumé, le choix entre L2TPv3 et VXLAN dépend des besoins spécifiques de l'environnement réseau, en tenant compte des exigences en matière de sécurité, de flexibilité et de mise à l'échelle.

Table 3. Comparaison entre L2TPv3 et VXLAN

Aspect	L2TPv3	VXLAN
Tunneling mechanism	Point to point	Point to multipoint
Encapsulation methods	IP and UDP	UDP

3.4 Mise en place dans Linux des VXLANs dans Open vSwitch

Par défaut, Open vSwitch utilisera le port IANA assigné pour VXLAN, qui est le 4789. Cependant, il est possible de configurer manuellement le port UDP de destination sur une base de tunnel VXLAN par tunnel. On cree d'abord le switch et ensuite on ajoute l'interface VXLAN avec l'adresse IP correspondante et d'autres options éventuelles, comme la modification du port de destination par défaut. Exemple:

```
Créer le switch br0
ovs-vsctl add-br br0.

Ajout de l'interface
add-port br0 vxlan1 -- set interface vxlan1 type=vxlan options:key=f1ow
options:remote_ip=192.168.100.2 options:dst_port=8472
```

3.5 Solutions de chiffrement du trafic L2TPv3 ou VXLAN

L2TPv3 et VXLAN sont aussi sécurisés que leur couche sous-jacente. Ils sont souvent utilisés avec un protocole de chiffrement tel que IPsec pour garantir la sécurité des données lors de leur transit. IPsec propose deux protocoles principaux pour assurer la sécurité du trafic :

- Le protocole AH (*Authentication Header*) assure l'intégrité et l'authentification de l'origine des datagrammes IP, à l'exception de certains champs pouvant changer lors du transfert.
- Le protocole ESP (*Encapsulating Security Payload*) chiffre le trafic pour assurer la confidentialité et l'authenticité des données, tout en protégeant contre les attaques par rejet et garantissant l'intégrité des données.

Ces deux protocoles peuvent être utilisés séparément ou ensemble, et offrent deux modes de fonctionnement : le mode Transport et le mode Tunnel. Le mode Transport établit une liaison sécurisée directe entre deux équipements, tandis que le mode Tunnel crée un passage sécurisé entre deux routeurs, permettant le transit sécurisé des datagrammes IP.

Cependant, alors que L2TPv3 prend en charge nativement plusieurs protocoles de chiffrement, VXLAN nécessite généralement une solution de chiffrement supplémentaire comme IPsec pour sécuriser le trafic. En termes de sécurité, si la confidentialité des données est une préoccupation majeure, L2TPv3 peut être la meilleure option en raison de sa prise en charge native de plusieurs protocoles de chiffrement. Cependant, si la flexibilité et la facilité de configuration sont plus importantes, VXLAN peut être préférable, bien que nécessitant souvent une configuration supplémentaire pour assurer la sécurité du trafic.

3.6 Comparaison avec MPLS

Le protocole MPLS (*Multiprotocol Label Switching*) est une solution de routage dans les réseaux de télécommunications qui dirige les données d'un nœud à l'autre en se basant sur des étiquettes de chemin plutôt que sur de longues adresses réseau, ce qui évite les recherches complexes dans une table de routage et accélère le flux de trafic. Il se situe entre les couches 2 et 3 du modèle OSI, la couche liaison de données et la couche réseau, d'où son appellation de protocole de couche 2.5.

Contrairement à L2TPv3 et VXLAN, MPLS est complètement séparé du reste du trafic et ne nécessite pas de chiffrement supplémentaire. De plus, MPLS offre la possibilité de réaliser de la Qualité de Service (QoS), ce qui permet de garantir un meilleur débit et un jitter plus faible, en théorie.

Tout comme L2TPv3 et VXLAN, MPLS peut être utilisé pour transporter différents types de trafic, mais il offre également des avantages distincts. Cependant, l'implémentation de MPLS peut être plus coûteuse que celle de L2TPv3 ou de VXLAN. MPLS nécessite un support matériel spécifique à chaque extrémité du réseau, tandis que VXLAN nécessite uniquement un support matériel pour

l'encapsulation au niveau des bords du réseau. De plus, MPLS exige un support de bout en bout, ce qui peut nécessiter le remplacement de l'ensemble du matériel réseau, tandis que VXLAN peut fonctionner avec un matériel existant dans le cœur du réseau.

Bien que MPLS propose de nombreux avantages, cette technologie reste beaucoup plus coûteuse à mettre en place que L2TPv3 ou VXLANs. Toutes ces technologies ont tous leurs propres avantages et inconvénients. Le choix dépendra des besoins spécifiques de l'environnement réseau, notamment en termes de sécurité, de performance et de coût.

4 Établissement du tunnel L2TPv3

4.1 Configuration du tunnel en mode encapsulation IP

Après avoir terminé la configuration et les tests d'encapsulation [VLAN](#), nous avons retiré les interfaces [r1-eth1.100](#) et [r1-eth.200](#) du routeur 1 et avons vidé la configuration de [r1-eth1](#) pour nous assurer qu'elle ne possédait pas d'adresses IP. Ensuite, en nous basant sur les exemples fournis dans le sujet, nous avons créé et activé le tunnel [l2tpeth0](#). Un pont nommé "[tunnel](#)" a été créé et les interfaces [l2tpeth0](#) et [r1-eth1](#) ont été ajoutées à ce pont. Par la suite, pour le VLAN de port, nous avons connecté les interfaces [tunnel.100](#) et [tunnel.200](#) aux [VLANs](#) 100 et 200 respectivement, et les avons toutes activées.

Les configurations détaillées pour le routeur 1 sont présentées ci-dessous.

```
#FLUSH
ip netns exec r1 ip link del r1-eth1.100
ip netns exec r1 ip link del r1-eth1.200
ip netns exec r1 ip a flush dev r1-eth1

#TUNELLING
ip netns exec r1 ip l2tp add tunnel remote 172.16.2.253 local 172.16.1.253 encap ip tunnel_id 3000 peer_tunnel_id 4000
ip netns exec r1 ip l2tp add session tunnel_id 3000 session_id 1000 peer_session_id 2000
ip netns exec r1 ip link set l2tpeth0 up

ip netns exec r1 brctl addbr tunnel
ip netns exec r1 brctl addif tunnel l2tpeth0
ip netns exec r1 brctl addif tunnel r1-eth1
ip netns exec r1 ip link set tunnel up

#VLANS
ip netns exec r1 ip link add link tunnel name tunnel.100 type vlan id 100
ip netns exec r1 ip link set tunnel.100 up
ip netns exec r1 ip link add link tunnel name tunnel.200 type vlan id 200
ip netns exec r1 ip link set tunnel.200 up
ip netns exec r1 ip addr add 192.168.100.254/24 dev tunnel.100
ip netns exec r1 ip addr add 192.168.200.254/24 dev tunnel.200
```

A noter que des configurations identiques ont été appliquées au routeur 2.

Pour consulter l'intégralité des scripts utilisés pour cette partie, veuillez vous référer à : [l2tpv3_ip.sh](#).

4.2 Test du Protocole ARP

Pour bien illustrer le fonctionnement du protocole ARP, nous avons intercepté le trafic entre p1 et p3, situé de l'autre côté à travers l'interface tunnel de R2 à l'aide de tcpdump.

```
root@ubuntu2004:/home/ubuntu/infra_res/projet# [r2] sudo tcpdump -i tunnel -lnvv not ip6
tcpdump: listening on tunnel, link-type EN10MB (Ethernet), capture size 262144 bytes
13:57:02.211071 ARP, Ethernet (len 6), IPv4 (len 4), Request who-has 192.168.100.2 tell 192.168.100.1, length 28
13:57:02.212066 ARP, Ethernet (len 6), IPv4 (len 4), Reply 192.168.100.2 is-at f2:e5:20:35:ac:de, length 28
13:57:02.212566 IP (tos 0x0, ttl 64, id 36664, offset 0, flags [DF], proto ICMP (1), length 84)
    192.168.100.1 > 192.168.100.2: ICMP echo request, id 28575, seq 1, length 64
13:57:02.215776 IP (tos 0x0, ttl 64, id 54380, offset 0, flags [none], proto ICMP (1), length 84)
    192.168.100.2 > 192.168.100.1: ICMP echo reply, id 28575, seq 1, length 64
13:57:03.212754 IP (tos 0x0, ttl 64, id 36725, offset 0, flags [DF], proto ICMP (1), length 84)
    192.168.100.1 > 192.168.100.2: ICMP echo request, id 28575, seq 2, length 64
13:57:03.212909 IP (tos 0x0, ttl 64, id 54477, offset 0, flags [none], proto ICMP (1), length 84)
    192.168.100.2 > 192.168.100.1: ICMP echo reply, id 28575, seq 2, length 64
13:57:04.225894 IP (tos 0x0, ttl 64, id 36904, offset 0, flags [DF], proto ICMP (1), length 84)
    192.168.100.1 > 192.168.100.2: ICMP echo request, id 28575, seq 3, length 64
13:57:04.226013 IP (tos 0x0, ttl 64, id 54643, offset 0, flags [none], proto ICMP (1), length 84)
    192.168.100.2 > 192.168.100.1: ICMP echo reply, id 28575, seq 3, length 64
13:57:07.429238 ARP, Ethernet (len 6), IPv4 (len 4), Request who-has 192.168.100.1 tell 192.168.100.2, length 28
13:57:07.429440 ARP, Ethernet (len 6), IPv4 (len 4), Reply 192.168.100.1 is-at 3a:c4:f9:c2:fc:92, length 28
```

Fig. 6. Capture du trafic entre P3 et P1 sur l'interface tunnel de R2

L'analyse du trafic (PING ICMP) ci-dessus [6] montre un échange de requêtes ARP suivi de réponses ICMP, indiquant le bon fonctionnement des requêtes ARP.

- Une requête ARP est émise par l'hôte 192.168.100.1 pour déterminer l'adresse MAC associée à l'adresse IP 192.168.100.2.
- Une réponse ARP est reçue, indiquant que l'adresse IP 192.168.100.2 est associée à l'adresse MAC f2:e5:20:35:ac:de qui est bien P3.
- Des échanges de requêtes echo request ICMP sont effectués entre 192.168.100.1 (P1) et 192.168.100.2(P3).
- Des réponses ICMP echo reply sont reçues, confirmant la connectivité entre les deux hôtes à travers le tunnel; on remarque aussi un TTL de 64.
- Une requête ARP est ensuite envoyée par l'hôte 192.168.100.2 pour déterminer l'adresse MAC associée à l'adresse IP 192.168.100.1.
- Une réponse ARP est reçue, indiquant que l'adresse IP 192.168.100.1 est associée à l'adresse MAC 3a:c4:f9:c2:fc:92 qui est bien P1.

Ces échanges démontrent que les requêtes ARP ont été correctement traitées, permettant la résolution des adresses IP en adresses MAC et assurant ainsi la connectivité entre les hôtes du réseau.

Un autre exemple avec ARPING

```
root@ubuntu2004:/home/ubuntu/infra_res/projet# [p1] arping -I p1-eth0.100 192.168.100.2
ARPING 192.168.100.2
42 bytes from f2:e5:20:35:ac:de (192.168.100.2): index=0 time=1.975 msec
42 bytes from f2:e5:20:35:ac:de (192.168.100.2): index=1 time=4.264 usec
42 bytes from f2:e5:20:35:ac:de (192.168.100.2): index=2 time=5.030 usec
42 bytes from f2:e5:20:35:ac:de (192.168.100.2): index=3 time=7.120 usec
42 bytes from f2:e5:20:35:ac:de (192.168.100.2): index=4 time=2.538 usec
42 bytes from f2:e5:20:35:ac:de (192.168.100.2): index=5 time=3.073 usec
42 bytes from f2:e5:20:35:ac:de (192.168.100.2): index=6 time=4.079 usec
42 bytes from f2:e5:20:35:ac:de (192.168.100.2): index=7 time=7.307 usec
42 bytes from f2:e5:20:35:ac:de (192.168.100.2): index=8 time=3.384 usec
42 bytes from f2:e5:20:35:ac:de (192.168.100.2): index=9 time=4.752 usec
```

Fig. 7. ARPING entre P1 et P3

4.3 Test du Service DHCP sur le Routeur 1

Pour le service DHCP, nous allons lancer le serveur DHCP sur le routeur1 et ensuite faire une requete DHCP depuis P1, P2, P3, P4 pour demontrer le bon fonctionnement du service :

Pour la mise en place du serveur DHCP, on va lancer un serveur DHCP pour le VLAN 100 qui va distribuer les adresse de 192.168.100.1/24 à 192.168.100.100/24; Et un autre pour le VLAN 200 qui va distribuer les adresses IPS de 192.168.200.1/24 à 192.168.200.100/24;

Voici les commandes effectuées et les captures de l'échange:

```
#Serveur DHCP pour les machines du VLAN 100
sudo ip netns exec r1 dnsmasq -d -z -i tunnel.100 --dhcp-range=192.168.100.1,
192.168.100.100,255.255.255.0
#Serveur DHCP pour les machines du VLAN 200
sudo ip netns exec r1 dnsmasq -d -z -i tunnel.200 --dhcp-range=192.168.200.1,
192.168.200.100,255.255.255.0
#Poste 1 Client DHCP
sudo ip netns exec p1 dhclient p1-eth0.100
```

```
ubuntu@ubuntu2004:~/infra_res/projet$ sudo ip netns exec r1 dnsmasq -d -z -i tunnel.100 --dhcp-range=192.168.100.1,192.168.
100.100,255.255.255.0
dnsmasq: démarré, version 2.90 (taille de cache 150)
dnsmasq: options à la compilation : IPv6 GNU-getopt DBus no-UBus i18n IDN DHCP DHCPv6 no-Lua Conntrack ipset no-nftset
auth cryptohash DNSSEC loop-detect inotify dumpfile
dnsmasq-dhcp: DHCP, IP range 192.168.100.1 -- 192.168.100.100, lease time 1h
dnsmasq-dhcp: DHCP, sockets bound exclusively to interface tunnel.100
dnsmasq: Lecture de /etc/resolv.conf
dnsmasq: utilise le serveur de nom 127.0.0.53#53
dnsmasq: read /etc/hosts - 9 names
dnsmasq-dhcp: DHCPDISCOVER(tunnel.100) 3a:c4:f9:c2:fc:92
dnsmasq-dhcp: DHCPOFFER(tunnel.100) 192.168.100.66 3a:c4:f9:c2:fc:92
dnsmasq-dhcp: DHCPDISCOVER(tunnel.100) 3a:c4:f9:c2:fc:92
dnsmasq-dhcp: DHCPOFFER(tunnel.100) 192.168.100.66 3a:c4:f9:c2:fc:92
dnsmasq-dhcp: DHCPREQUEST(tunnel.100) 192.168.100.66 3a:c4:f9:c2:fc:92
dnsmasq-dhcp: DHCPACK(tunnel.100) 192.168.100.66 3a:c4:f9:c2:fc:92 ubuntu2004
```

Fig. 8. DHCP Server listening on R1

```
ubuntu@ubuntu2004:~/infra_res/projet$ sudo ip netns exec p1 dhclient p1-eth0.100
ubuntu@ubuntu2004:~/infra_res/projet$ sudo ip netns exec p1 ip a
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group default qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
        valid_lft forever preferred_lft forever
    inet6 ::1/128 scope host
        valid_lft forever preferred_lft forever
2: p1-eth0.100@p1-eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue state UP group default qlen 1000
    link/ether 3a:c4:f9:c2:fc:92 brd ff:ff:ff:ff:ff:ff
    inet 192.168.100.1/24 scope global p1-eth0.100
        valid_lft forever preferred_lft forever
        inet 192.168.100.66/24 brd 192.168.100.255 scope global secondary dynamic p1-eth0.100
            valid_lft 3571sec preferred_lft 3571sec
        inet6 fe80::38c4:f9ff:fe92:64 scope link
            valid_lft forever preferred_lft forever
150: p1-eth0@if149: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue state UP group default qlen 1000
    link/ether 3a:c4:f9:c2:fc:92 brd ff:ff:ff:ff:ff:ff link-netnsid 0
    inet6 fe80::38c4:f9ff:fe92:64 scope link
        valid_lft forever preferred_lft forever
```

Fig. 9. Machine P1 gaining an IP adress

Sur la capture [8] Le serveur DHCP a été configuré sur l'interface `tunnel.100` du routeur `r1`. Il distribue des adresses IP de la plage 192.168.100.1 à 192.168.100.100, avec un bail de 1 heure. Lorsqu'une machine effectue une demande DHCP (`DHCPDISCOVER`), le serveur lui offre une adresse IP disponible (`DHCPOFFER`). Lorsque la machine accepte cette offre en envoyant une demande DHCP (`DHCPREQUEST`), le serveur lui accorde définitivement l'adresse IP (`DHCPACK`). Dans cet exemple, la machine avec l'adresse MAC `3a:c4:f9:c2:fc:92` qui est bien P1 s'est vue attribuer l'adresse IP `192.168.100.66` avec le nom d'hôte `ubuntu2004`. Ce qui est confirmé en affichant la configuration IP de P1 sur la capture [9]. . On remarque que les adresses IP et MAC concordent bien et donc que le poste 1 se configure bien à partir d'un DHCP s'exécutant sur le routeur 1.

4.4 Connexion SOCAT entre le Routeur1 et le Poste 1

Pour vérifier qu'on peut bien établir une connexion SOCAT entre le Routeur 1 (R1) et P1 à travers le tunnel, nous avons procédé comme suit :

Étape 1 : Lancer un socat listener sur le routeur en mode serveur

```
sudo ip netns exec r1 socat tcp-listen:8888,reuseaddr,fork -
```

```
^Cubuntu@ubuntu2004:~/infra_res/projet$ sudo ip netns exec r1 socat tcp-listen:8888,reuseaddr,fork -
BOnjour
Don't panic!
it worksssssss!
```

Fig. 10. TCP Server listening on R1

Étape 2 : Lancer un socat client sur P1

```
sudo ip netns exec p1 socat - tcp:192.168.100.254:8888
```

```
ubuntu@ubuntu2004:~/infra_res/projet$ sudo ip netns exec p1 socat - tcp:192.168.100.254:8888
BOnjour
Don't panic!
it worksssssss!
```

Fig. 11. TCP Client on P1

Étape 3 : Intercepter le trafic avec tcpdump sur le tunnel

```
sudo ip netns exec r1 tcpdump -l nvvv -i tunnel not ip6
```

```
ubuntu@ubuntu2004:~/infra_res/projet$ sudo ip netns exec r1 tcpdump -l nvvv -i tunnel tcp
tcpdump: listening on tunnel, link-type EN10MB (Ethernet), capture size 262144 bytes
06:12:29.037294 IP (tos 0x0, ttl 64, id 10065, offset 0, flags [DF], proto TCP (6), length 60)
    192.168.100.66.60176 > 192.168.100.254.8888: Flags [S], cksum 0x38c6 (correct), seq 3077120814, win 64240, options [mss 1460,sackOK,TS val 2385990431 ecr 0,nop,wscale 7]
, length 0
06:12:29.037294 ecr 2385990431,nop,wscale 7], length 0
    192.168.100.254.8888 > 192.168.100.66.60176: Flags [.], cksum 0xe9b2 (incorrect -> 0xe9b2), seq 2816039195, ack 3077120815, win 64676, options [mss 1418,sackOK,TS val 3
280963827 ecr 2385990431,nop,wscale 7], length 0
06:12:29.041831 IP (tos 0x0, ttl 64, id 10066, offset 0, flags [DF], proto TCP (6), length 52)
    192.168.100.66.60176 > 192.168.100.254.8888: Flags [.], cksum 0x1303 (correct), seq 1, ack 1, win 502, options [nop,nop,TS val 2385990432 ecr 3280963827], length 0
06:12:43.604838 IP (tos 0x0, ttl 64, id 10067, offset 0, flags [DF], proto TCP (6), length 60)
    192.168.100.66.60176 > 192.168.100.254.8888: Flags [.], cksum 0x47d2 (correct), seq 1:9, ack 1, win 502, options [nop,nop,TS val 2386004999 ecr 3280963827], length 8
06:12:43.604922 IP (tos 0x0, ttl 64, id 50331, offset 0, flags [DF], proto TCP (6), length 52)
    192.168.100.254.8888 > 192.168.100.66.60176: Flags [.], cksum 0x4ab8 (incorrect -> 0xa128), seq 1, ack 9, win 506, options [nop,nop,TS val 3280978394 ecr 2386004999], le
ngth 0
06:12:53.862612 IP (tos 0x0, ttl 64, id 50332, offset 0, flags [DF], proto TCP (6), length 65)
    192.168.100.254.8888 > 192.168.100.66.60176: Flags [P.], cksum 0x4ac5 (incorrect -> 0x065e), seq 1:14, ack 9, win 506, options [nop,nop,TS val 3280988652 ecr 2386004999]
, length 13
06:12:53.863190 IP (tos 0x0, ttl 64, id 10068, offset 0, flags [DF], proto TCP (6), length 52)
    192.168.100.66.60176 > 192.168.100.254.8888: Flags [.], cksum 0x50fb (correct), seq 9, ack 14, win 502, options [nop,nop,TS val 2386015257 ecr 3280988652], length 0
06:13:35.949660 IP (tos 0x0, ttl 64, id 10069, offset 0, flags [DF], proto TCP (6), length 68)
    192.168.100.66.60176 > 192.168.100.254.8888: Flags [P.], cksum 0xc4c5 (correct), seq 9:25, ack 14, win 502, options [nop,nop,TS val 2386057344 ecr 3280988652], length 16
06:13:35.949809 IP (tos 0x0, ttl 64, id 50333, offset 0, flags [DF], proto TCP (6), length 52)
    192.168.100.254.8888 > 192.168.100.66.60176: Flags [.], cksum 0x4ab8 (incorrect -> 0x0818), seq 14, ack 25, win 506, options [nop,nop,TS val 3281030739 ecr 2386057344], length 0
06:18:01.934087 IP (tos 0x0, ttl 64, id 10070, offset 0, flags [DF], proto TCP (6), length 55)
    192.168.100.66.60176 > 192.168.100.254.8888: Flags [P.], cksum 0x8e8f (correct), seq 25:28, ack 14, win 502, options [nop,nop,TS val 2386322329 ecr 3281030739], length 3
06:18:00.934067 IP (tos 0x0, ttl 64, id 50334, offset 0, flags [DF], proto TCP (6), length 52)
    192.168.100.254.8888 > 192.168.100.66.60176: Flags [.], cksum 0x4ab8 (incorrect -> 0xfida), seq 14, ack 28, win 506, options [nop,nop,TS val 3281295724 ecr 2386322329], length 0
06:18:02.093779 IP (tos 0x0, ttl 64, id 10071, offset 0, flags [DF], proto TCP (6), length 53)
    192.168.100.66.60176 > 192.168.100.254.8888: Flags [P.], cksum 0x634e (correct), seq 28:29, ack 14, win 502, options [nop,nop,TS val 2386323488 ecr 3281295724], length 1
06:18:02.093796 IP (tos 0x0, ttl 64, id 50335, offset 0, flags [DF], proto TCP (6), length 52)
    192.168.100.254.8888 > 192.168.100.66.60176: Flags [.], cksum 0x4ab8 (incorrect -> 0x8ecb), seq 14, ack 29, win 506, options [nop,nop,TS val 3281296883 ecr 2386323488], length 0
06:18:18.953901 IP (tos 0x0, ttl 64, id 50336, offset 0, flags [DF], proto TCP (6), length 52)
    192.168.100.254.8888 > 192.168.100.66.60176: Flags [P.], cksum 0x4ab8 (incorrect -> 0xa6ee), seq 14, ack 29, win 506, options [nop,nop,TS val 3281313743 ecr 2386323488], length 0
06:18:18.007326 IP (tos 0x0, ttl 64, id 10072, offset 0, flags [DF], proto TCP (6), length 52)
    192.168.100.66.60176 > 192.168.100.254.8888: Flags [.], cksum 0x64e1 (correct), seq 29, ack 15, win 502, options [nop,nop,TS val 2386340401 ecr 3281313743], length 0
06:18:19.456282 IP (tos 0x0, ttl 64, id 10073, offset 0, flags [DF], proto TCP (6), length 52)
    192.168.100.66.60176 > 192.168.100.254.8888: Flags [P.], cksum 0x631e (correct), seq 29, ack 15, win 502, options [nop,nop,TS val 2386340851 ecr 3281313743], length 0
06:18:19.456526 IP (tos 0x0, ttl 64, id 0, offset 0, flags [DF], proto TCP (6), length 52)
    192.168.100.254.8888 > 192.168.100.66.60176: Flags [.], cksum 0x6123 (correct), seq 15, ack 30, win 506, options [nop,nop,TS val 3281314246 ecr 2386340851], length 0
```

Fig. 12. TCP traffic between R1 and P1

On voit bien que R1 et P1 arrivent à bien établir une connexion avec SOCAT et à s'échanger des données.

Le trafic TCP capturé par `tcpdump` [12] montre une communication entre l'hôte 192.168.100.66 P1 et le serveur R1 sur le port 8888. Voici un résumé de l'analyse du trafic :

1. À 06:12:29, l'hôte 192.168.100.66 envoie un paquet TCP avec le drapeau SYN (synchronisation) au serveur sur le port 8888 pour initialiser une connexion.
2. Le serveur répond immédiatement avec un paquet TCP SYN-ACK pour confirmer la demande de connexion.
3. L'hôte 192.168.100.66 envoie ensuite un paquet TCP ACK pour confirmer la réponse du serveur et établir la connexion.
4. À 06:12:43, l'hôte 192.168.100.66 envoie des données (paquet TCP avec le drapeau PSH) au serveur.
5. Le serveur répond avec des paquets TCP ACK pour indiquer la réception des données.
6. La communication continue avec des échanges de données entre l'hôte et le serveur, suivis de réponses ACK.
7. À 06:18:18, le serveur envoie un paquet TCP avec le drapeau FIN pour terminer la connexion.
8. L'hôte répond avec un paquet TCP ACK pour confirmer la fermeture de la connexion.
9. À 06:18:19, l'hôte envoie également un paquet TCP avec le drapeau FIN pour signaler la fin de sa partie de la connexion.
10. Le serveur répond avec un paquet TCP ACK pour confirmer la fermeture de la connexion.

En résumé, cette capture montre une communication TCP normale entre l'hôte 192.168.100.66 qui est P1 et le serveur R1 sur le port 8888, comprenant l'initialisation de la connexion, l'échange de données et la fermeture de la connexion et l'endroit de la capture du trafic montre qu'il passe bien par le tunnel.

5 Mise en œuvre de L2TPv3 en mode Encapsulation IP et UDP

Nous avons mis en œuvre le tunnel L2TPv3 en mode encapsulation UDP en suivant la même méthodologie que pour l'encapsulation IP qui est détaillée dans la section précédente. Nos ports sources et destinations sont respectivement (2024, 2025) du côté de R1 et (2025, 2024) du côté de R2.

Le fichier de script utilisé est [L2TPV3_UDP.sh](#).

Cette mise en place permet d'établir une communication sécurisée entre les routeurs R1 et R2 en utilisant le protocole L2TPv3 avec une encapsulation UDP.

5.1 Comparaison du mode IP et UDP

Pour faire une comparaison, on va faire un ping ICMP entre p2 et p4 en capturant les paquets sur RA dans les deux modes

```
ubuntu@ubuntu2004:~/infra_res/projet$ sudo ip netns exec rA tcpdump -lnvv not ip6
tcpdump: listening on rA-eth0, link-type EN10MB (Ethernet), capture size 262144 bytes
07:13:17.815483 IP (tos 0x0, ttl 62, id 37027, offset 0, flags [none], proto unknown (115), length 102)
    172.16.2.253 > 172.16.1.253: ip-proto-115 82
07:13:19.863171 IP (tos 0x0, ttl 64, id 26399, offset 0, flags [none], proto unknown (115), length 98)
    172.16.1.253 > 172.16.2.253: ip-proto-115 78
07:13:21.911661 IP (tos 0x0, ttl 62, id 37028, offset 0, flags [none], proto unknown (115), length 102)
    172.16.2.253 > 172.16.1.253: ip-proto-115 82
07:13:24.991493 ARP, Ethernet (len 6), IPv4 (len 4), Request who-has 172.16.1.254 tell 172.16.1.253, length 28
07:13:24.991548 ARP, Ethernet (len 6), IPv4 (len 4), Reply 172.16.1.254 is-at 72:a3:f8:0b:a6:9d, length 28
07:13:33.458913 IP (tos 0x0, ttl 62, id 37029, offset 0, flags [none], proto unknown (115), length 130)
    172.16.2.253 > 172.16.1.253: ip-proto-115 110
07:13:33.459466 IP (tos 0x0, ttl 64, id 26400, offset 0, flags [none], proto unknown (115), length 130)
    172.16.1.253 > 172.16.2.253: ip-proto-115 110
07:13:38.558868 ARP, Ethernet (len 6), IPv4 (len 4), Request who-has 172.16.1.253 tell 172.16.1.254, length 28
07:13:38.559432 IP (tos 0x0, ttl 64, id 26401, offset 0, flags [none], proto unknown (115), length 74)
    172.16.1.253 > 172.16.2.253: ip-proto-115 54
07:13:38.559460 ARP, Ethernet (len 6), IPv4 (len 4), Reply 172.16.1.253 is-at 4e:f5:5f:0e:c0:98, length 28
07:13:38.559604 IP (tos 0x0, ttl 62, id 37030, offset 0, flags [none], proto unknown (115), length 74)
    172.16.2.253 > 172.16.1.253: ip-proto-115 54
07:13:54.680013 IP (tos 0x0, ttl 62, id 37031, offset 0, flags [none], proto unknown (115), length 102)
    172.16.2.253 > 172.16.1.253: ip-proto-115 82
07:13:54.681665 IP (tos 0x0, ttl 62, id 37032, offset 0, flags [none], proto unknown (115), length 98)
    172.16.2.253 > 172.16.1.253: ip-proto-115 78
^C
13 packets captured
13 packets received by filter
0 packets dropped by kernel
```

Fig. 13. ICMP between P2 and P4 in IP Mode

```
ubuntu@ubuntu2004:~/infra_res/projet$ sudo ip netns exec rA tcpdump -lnvv not ip6
tcpdump: listening on rA-eth0, link-type EN10MB (Ethernet), capture size 262144 bytes
07:08:47.480149 IP (tos 0x0, ttl 64, id 34374, offset 0, flags [none], proto UDP (17), length 110)
    172.16.1.253.2024 > 172.16.2.253.2025: [no cksum] UDP, length 82
07:08:47.480296 IP (tos 0x0, ttl 62, id 47038, offset 0, flags [none], proto UDP (17), length 110)
    172.16.2.253.2025 > 172.16.1.253.2024: [no cksum] UDP, length 82
07:09:03.020556 IP (tos 0x0, ttl 62, id 47039, offset 0, flags [none], proto UDP (17), length 142)
    172.16.2.253.2025 > 172.16.1.253.2024: [no cksum] UDP, length 114
07:09:03.020888 IP (tos 0x0, ttl 64, id 34375, offset 0, flags [none], proto UDP (17), length 142)
    172.16.1.253.2024 > 172.16.2.253.2025: [no cksum] UDP, length 114
07:09:03.865874 IP (tos 0x0, ttl 62, id 47040, offset 0, flags [none], proto UDP (17), length 110)
    172.16.2.253.2025 > 172.16.1.253.2024: [no cksum] UDP, length 82
07:09:03.867677 IP (tos 0x0, ttl 64, id 34376, offset 0, flags [none], proto UDP (17), length 114)
    172.16.1.253.2024 > 172.16.2.253.2025: [no cksum] UDP, length 86
07:09:08.215654 IP (tos 0x0, ttl 64, id 34377, offset 0, flags [none], proto UDP (17), length 86)
    172.16.1.253.2024 > 172.16.2.253.2025: [no cksum] UDP, length 58
07:09:08.215713 IP (tos 0x0, ttl 62, id 47041, offset 0, flags [none], proto UDP (17), length 86)
    172.16.2.253.2025 > 172.16.1.253.2024: [no cksum] UDP, length 58
07:09:08.215817 IP (tos 0x0, ttl 62, id 47042, offset 0, flags [none], proto UDP (17), length 86)
    172.16.2.253.2025 > 172.16.1.253.2024: [no cksum] UDP, length 58
07:09:08.215944 IP (tos 0x0, ttl 64, id 34378, offset 0, flags [none], proto UDP (17), length 86)
    172.16.1.253.2024 > 172.16.2.253.2025: [no cksum] UDP, length 58
```

Fig. 14. ICMP between P2 and P4 in UDP Mode

On remarque que pour le trafic ICMP capturé en mode [IP](#), le protocole IP ([proto unknown](#)) n'est pas visible lorsqu'on analyse les trames passant par le routeur A (voir Figure 13), tandis que dans le mode [UDP](#), le protocole UDP ([proto UDP](#)) est clairement identifié dans le datagramme IP (voir Figure 14). De plus, lors de l'utilisation du mode d'encapsulation [IP](#) avec [L2TPv3](#), les paquets contiennent un

en-tête L2TPv3 ([IP-proto-115](#)), comme illustré dans la Figure 13. En revanche, dans la Figure 14, lorsque nous utilisons le mode d'encapsulation UDP avec L2TPv3, les paquets contiennent un en-tête UDP avec les ports [2024](#) et [2025](#), conformément à notre configuration. En outre, la longueur des paquets en mode UDP diffère de celle en mode IP, ce qui est attribuable à la différence de longueur de ces en-têtes.

5.2 Comparaison avec un tunnel GRE en mode GRETAP

GRETAP (Generic Routing Encapsulation over Tunneling Protocol) est un protocole de tunnelling utilisé pour encapsuler des trames Ethernet dans des paquets IP. L'unité de transmission maximale (MTU) de GRETAP est de 1462 octets.

Le GRE ou Generic Routing Encapsulation est l'un des mécanismes de tunnelisation disponibles qui peut encapsuler une grande variété de types de paquets de protocole à l'intérieur de tunnels IP. Il fournit un chemin privé pour transporter des paquets à travers un réseau public en encapsulant ou en tunnelisant les paquets. Nous utilisons un tunnel GRE lorsque nous avons besoin d'établir une relation de voisinage entre 2 routeurs et de transporter des paquets d'un protocole sur un autre.

Dans cette section, nous mettrons en place un tunnel GRE en mode GRETAP et le comparerons au tunnel L2TPv3.

La configuration du tunnel GRE en mode GRETAP est relativement simple car une partie du travail est déjà effectuée lors de la configuration du tunnel L2TPv3 et du pont. Les scripts pour l'établissement du tunnel GRE en mode GRETAP sont disponibles dans le fichier [gre-greTAP.sh](#).

```
ubuntu@ubuntu2004:~/infra_res/projet$ sudo ip netns exec rA tcpdump -l nvv not ip6
tcpdump: listening on rA-eth0, link-type EN10MB (Ethernet), capture size 262144 bytes
15:22:36.680263 IP (tos 0x0, ttl 62, id 42244, offset 0, flags [none], proto GRE (47), length 126)
    172.16.2.253 > 172.16.1.253: GREv0, Flags [none], length 106
        IP (tos 0x0, ttl 64, id 64757, offset 0, flags [DF], proto ICMP (1), length 84)
        192.168.100.1 > 192.168.100.2: ICMP echo request, id 60836, seq 1, length 64
15:22:36.680721 IP (tos 0x0, ttl 64, id 38631, offset 0, flags [none], proto GRE (47), length 126)
    172.16.1.253 > 172.16.2.253: GREv0, Flags [none], length 106
        IP (tos 0x0, ttl 64, id 60861, offset 0, flags [none], proto ICMP (1), length 84)
        192.168.100.2 > 192.168.100.1: ICMP echo reply, id 60836, seq 1, length 64
15:22:41.788336 ARP, Ethernet (len 6), IPv4 (len 4), Request who-has 172.16.1.253 tell 172.16.1.254, length 28
15:22:41.788761 ARP, Ethernet (len 6), IPv4 (len 4), Request who-has 172.16.1.254 tell 172.16.1.253, length 28
15:22:41.788755 ARP, Ethernet (len 6), IPv4 (len 4), Reply 172.16.1.253 is-at 8a:81:15:38:07:c4, length 28
15:22:41.788767 ARP, Ethernet (len 6), IPv4 (len 4), Reply 172.16.1.254 is-at da:22:b0:b2:ea:29, length 28
15:22:41.788796 IP (tos 0x0, ttl 64, id 38930, offset 0, flags [none], proto GRE (47), length 70)
    172.16.1.253 > 172.16.2.253: GREv0, Flags [none], length 50
        ARP, Ethernet (len 6), IPv4 (len 4), Request who-has 192.168.100.1 tell 192.168.100.2, length 28
15:22:41.788821 IP (tos 0x0, ttl 62, id 42276, offset 0, flags [none], proto GRE (47), length 70)
    172.16.2.253 > 172.16.1.253: GREv0, Flags [none], length 50
        ARP, Ethernet (len 6), IPv4 (len 4), Request who-has 192.168.100.2 tell 192.168.100.1, length 28
15:22:41.788900 IP (tos 0x0, ttl 62, id 42277, offset 0, flags [none], proto GRE (47), length 70)
    172.16.2.253 > 172.16.1.253: GREv0, Flags [none], length 50
        ARP, Ethernet (len 6), IPv4 (len 4), Reply 192.168.100.1 is-at 72:de:c5:24:07:93, length 28
15:22:41.789016 IP (tos 0x0, ttl 64, id 38931, offset 0, flags [none], proto GRE (47), length 70)
    172.16.1.253 > 172.16.2.253: GREv0, Flags [none], length 50
        ARP, Ethernet (len 6), IPv4 (len 4), Reply 192.168.100.2 is-at 9e:b9:3a:c7:18:17, length 28
^C
```

Fig. 15. ICMP Between P1 and P3 in GRE Mode

Dans la figure précédente (voir Figure 15), qui est une capture sur rA d'un ping ICMP entre P1 et P3, on constate que chaque paquet GRE contient un en-tête GREv0. On peut remarquer que les paquets ICMP (protocole 1) sont encapsulés dans les paquets IP à l'intérieur des paquets GRE. On remarque qu'avec GRE, on peut visualiser le datagramme encapsulé et donc voir les deux machines qui communiquent (ici 192.168.100.1 et 192.168.100.2).

5.3 Comparaison des MTUs

Pour vérifier la MTU, nous avons ouvert un terminal séparé dans lequel nous avons exécuté la commande suivante à chaque fois que nous changeons de mode ou de tunnel :

Pour le tunnel L2TPv3 en mode IP et UDP

```
sudo ip netns exec r1 ifconfig tunnel100
```

```
ubuntu@ubuntu2004:~/infra_res/projet$ sudo ip netns exec r1 ifconfig tunnel.100
tunnel.100: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 1446
    inet 192.168.100.254 netmask 255.255.255.0 broadcast 0.0.0.0
    inet6 fe80::5cb7:21ff:fe26:a2cc prefixlen 64 scopeid 0x20<link>
        ether 5e:b7:21:26:a2:cc txqueuelen 1000 (Ethernet)
        RX packets 22 bytes 1368 (1.3 KB)
        RX errors 0 dropped 0 overruns 0 frame 0
        TX packets 13 bytes 1026 (1.0 KB)
        TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

ubuntu@ubuntu2004:~/infra_res/projet$ sudo ip netns exec r1 ifconfig tunnel.100
tunnel.100: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 1458
    inet 192.168.100.254 netmask 255.255.255.0 broadcast 0.0.0.0
    inet6 fe80::6c77:99ff:fecb:e422 prefixlen 64 scopeid 0x20<link>
        ether 6e:77:99:cb:e4:22 txqueuelen 1000 (Ethernet)
        RX packets 22 bytes 1368 (1.3 KB)
        RX errors 0 dropped 0 overruns 0 frame 0
        TX packets 12 bytes 956 (956.0 B)
        TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0
```

Fig. 16. First MTU is UDP Mode and second one is IP Mode

Pour le tunnel GRE:

```
sudo ip netns exec r1 ifconfig eoip1
```

```
ubuntu@ubuntu2004:~/infra_res/projet$ sudo ip netns exec r1 ifconfig eoip1
eoip1: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 1462
    inet6 fe80::ec8a:c1ff:feb9:269 prefixlen 64 scopeid 0x20<link>
        ether ee:8a:c1:b9:02:69 txqueuelen 1000 (Ethernet)
        RX packets 93 bytes 7705 (7.7 KB)
        RX errors 0 dropped 0 overruns 0 frame 0
        TX packets 105 bytes 7103 (7.1 KB)
        TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0
```

Fig. 17. MTU in GRE Mode

Dans les captures précédentes (voir Figures 16 et 17), on observe que la **MTU** pour un tunnel **L2TPv3** en mode **UDP** est de 1446 octets, en mode **IP** elle est de 1458 octets, tandis qu'avec un tunnel **GRE**, elle atteint 1462 octets. Cette différence s'explique par les différents en-têtes utilisés. Dans le cas d'un tunnel **GRE**, la MTU maximale serait de 1476 octets (1500 octets moins 24 octets), ce qui est logique compte tenu des 20 octets nécessaires pour l'en-tête IP encapsulée et des 4 octets pour l'en-tête GRE.

Pour finir, **GRE (Generic Routing Encapsulation)** offre un mécanisme de tunneling polyvalent permettant d'encapsuler divers types de paquets de protocoles dans des tunnels IP. Sa MTU maximale est de 1476 octets. **L2TPv3** en mode **IP** encapsule les trames Ethernet dans des paquets IP avec un en-tête **L2TPv3**, offrant une MTU maximale de 1458 octets. En revanche, **L2TPv3** en mode **UDP** utilise l'**UDP** pour encapsuler les trames Ethernet dans des paquets IP, avec des ports spécifiques, offrant une MTU maximale de 1446 octets.

6 Chiffrement IPSEC sur le tunnel L2TPV3 en mode IP

6.1 Mise en place du chiffrement IPsec:

Dans le cadre de la mise en place du chiffrement avec IPsec, nous avons utilisé la commande `ip xfrm` en configurant un AH (*Authentication Header*) et un ESP (*Encapsulating Security Payload*).

IPsec (Internet Protocol Security) est un ensemble de protocoles et de normes de sécurité conçus pour garantir la confidentialité, l'intégrité et l'authenticité des données échangées sur un réseau IP. L'AH (*Authentication Header*) assure l'authentification et l'intégrité des paquets IP en ajoutant un en-tête contenant un code de hachage calculé à partir des données du paquet et d'un secret partagé entre les périphériques communicants. Cette vérification garantit que les données n'ont pas été altérées lors de leur transmission et assure l'authenticité de l'expéditeur.

L'ESP (*Encapsulating Security Payload*) fournit à la fois le chiffrement et l'authentification des données en encapsulant les paquets IP dans une enveloppe sécurisée. Il chiffre le contenu des paquets pour assurer leur confidentialité et ajoute des informations d'authentification pour garantir leur intégrité, utilisant des algorithmes de chiffrement symétriques tels que AES pour le chiffrement et HMAC pour l'authentification.

Voici les commandes qui ont été exécutées sur R1:

```
#!/bin/bash

# Flush the SAD and SPD
ip netns exec r1 ip xfrm state flush
ip netns exec r1 ip xfrm policy flush

# AH SAs using 256 bit long and ESP SAs using 160 bit keys
ip netns exec r1 ip xfrm state add src 172.16.1.253 dst 172.16.2.253
    ↪ proto esp spi 0x12345678 reqid 0x12345678 mode transport auth
    ↪ sha256 0
    ↪ x323730ed6f1b9ff0cb084af15b197e862b7c18424a7cdfb74cd385ae23bc4
    ↪ f17 enc "rfc3686(ctr(aes))" 0
    ↪ x27b90b8aec1ee32a8150a664e8faac761e2d305b

ip netns exec r1 ip xfrm state add src 172.16.2.253 dst 172.16.1.253
    ↪ proto esp spi 0x12345678 reqid 0x12345678 mode transport auth
    ↪ sha256 0
    ↪ x44d65c50b7581fd3c8169cf1fa0ebb24e0d55755b1dc43a98b539bb144f2
    ↪ 067f enc "rfc3686(ctr(aes))" 0
    ↪ x9df7983cb7c7eb2af01d88d36e462b5f01d10bc1

# Politiques de sécurité
ip netns exec r1 ip xfrm policy add src 172.16.2.253 dst
    ↪ 172.16.1.253 dir in tmpl src 172.16.2.253 dst 172.16.1.253
    ↪ proto esp reqid 0x12345678 mode transport
ip netns exec r1 ip xfrm policy add src 172.16.1.253 dst
    ↪ 172.16.2.253 dir out tmpl src 172.16.1.253 dst 172.16.2.253
    ↪ proto esp reqid 0x12345678 mode transport
```

A noter que ces mêmes commandes doivent être exécutées sur R2, l'autre bout du tunnel. Pour la totalité des scripts, voir [l2tpv3_ip_ipsec.sh](#).

6.2 Comparaison via iperf de la vitesse de transport avec et sans IPSec

Pour comparer la vitesse, nous allons lancer un serveur iperf sur R1. Ensuite, depuis P1, nous allons lancer le client iperf. Nous allons effectuer la procédure à la fois avec et sans chiffrement pour une durée de 30s. Voici les commandes à exécuter côté client et côté serveur :

```
#Serveur sur R1
sudo ip netns exec r1 iperf -s
#Client sur P1
sudo ip netns exec r1 iperf -c 192.168.100.254 -t 30
```

Voici les résultats obtenus:

Sans IPSec:

```
ubuntu@ubuntu2004:~/infra_res/projet$ sudo ip netns exec p2 iperf -c 192.168.100.254 -t 30
-----
Client connecting to 192.168.100.254, TCP port 5001
TCP window size: 221 KByte (default)
-----
[ 3] local 192.168.200.1 port 37070 connected with 192.168.100.254 port 5001
[ ID] Interval      Transfer     Bandwidth
[ 3]  0.0-30.0 sec  4.05 GBytes  1.16 Gbits/sec
```

Fig. 18. Transport Speed without IPSec; Client Side

```
ubuntu@ubuntu2004:~/infra_res/projet$ sudo ip netns exec r1 iperf -s
-----
Server listening on TCP port 5001
TCP window size: 128 KByte (default)
-----
[ 4] local 192.168.100.254 port 5001 connected with 192.168.200.1 port 37070
[ ID] Interval      Transfer     Bandwidth
[ 4]  0.0-31.3 sec  4.05 GBytes  1.11 Gbits/sec
```

Fig. 19. Transport Speed without IPSec; Server Side

Et avec IPSec :

```
^Cubuntu@ubuntu2004:~/infra_res/projet$ sudo ip netns exec r1 iperf -s
-----
Server listening on TCP port 5001
TCP window size: 128 KByte (default)
-----
[ 4] local 192.168.100.254 port 5001 connected with 192.168.200.1 port 50656
[ ID] Interval      Transfer     Bandwidth
[ 4]  0.0-30.1 sec  1.83 GBytes  524 Mbits/sec
```

Fig. 20. Transport Speed with IPSec; Server Side

```
ubuntu@ubuntu2004:~/infra_res/projet$ sudo ip netns exec p2 iperf -c 192.168.100.254 -t 30
-----
Client connecting to 192.168.100.254, TCP port 5001
TCP window size: 2.50 MByte (default)
-----
[ 3] local 192.168.200.1 port 50656 connected with 192.168.100.254 port 5001
[ ID] Interval Transfer Bandwidth
[ 3] 0.0-30.0 sec 1.83 GBytes 524 Mbits/sec
```

Fig. 21. Transport Speed with IPsec; Client Side

Nous remarquons qu'avec un intervalle de 30 secondes, iperf a transféré **04 GBytes** à un débit de **1,16 Gbits/sec** sans IPsec. En revanche, avec IPsec, la valeur du transfert est de **1.83 GBytes** et la bande passante est de **524 Mbits/sec**. Il semble que L2TPv3 sans chiffrement soit **2,5 fois plus rapide** que L2TPv3 avec chiffrement. Cela s'explique par le fait qu'IPsec encapsule les paquets IP, ce qui les rend plus longs et entraîne une fragmentation. Le récepteur doit également rassembler les paquets. La fragmentation, la rassemblement, le chiffrement et le déchiffrement des fragments consomment de nombreuses ressources CPU.

Le chiffrement des paquets avec IPsec a donc un impact significatif sur le débit. Cette diminution du débit est principalement due au temps nécessaire pour chiffrer les paquets. Le débit lors de l'utilisation du chiffrement avec IPsec dépend également de l'algorithme utilisé pour le chiffrement des paquets. Toutefois, la MTU et le MSS (**Maximum Segment Size**) ne sont pas impactés par le chiffrement, ce qui est cohérent.

6.3 Capture du trafic L2TPV3 en mode chiffrement IPsec

Comme d'habitude, on va se lancer un tcpdump sur RA et on va essayer de pinger P4 depuis P2 en ayant mis le chiffrement IPsec sur le tunnel L2TPV3 en mode IP. Voici le trafic capturé :

```
ubuntu@ubuntu2004:~/infra_res/projet$ sudo ip netns exec rA tcpdump -lnvv not ip6
tcpdump: listening on rA-eth0, link-type EN10MB (Ethernet), capture size 262144 bytes
11:12:09.909282 IP (tos 0x0, ttl 62, id 33538, offset 0, flags [none], proto ESP (50), length 160
    172.16.2.253 > 172.16.1.253: ESP(spi=0x12345678,seq=0x15c1b4), length 140
11:12:09.909985 IP (tos 0x0, ttl 64, id 43898, offset 0, flags [none], proto ESP (50), length 160
    172.16.1.253 > 172.16.2.253: ESP(spi=0x12345678,seq=0x24ae3), length 140
11:12:15.054336 ARP, Ethernet (len 6), IPv4 (len 4), Request who-has 172.16.1.253 tell 172.16.1.254, length 28
11:12:15.054631 ARP, Ethernet (len 6), IPv4 (len 4), Request who-has 172.16.1.254 tell 172.16.1.253, length 28
11:12:15.054634 ARP, Ethernet (len 6), IPv4 (len 4), Reply 172.16.1.254 is-at aa:fa:0e:52:ca:9e, length 28
11:12:15.054667 IP (tos 0x0, ttl 64, id 43899, offset 0, flags [none], proto ESP (50), length 104
    172.16.1.253 > 172.16.2.253: ESP(spi=0x12345678,seq=0x24ae4), length 84
11:12:15.054691 ARP, Ethernet (len 6), IPv4 (len 4), Reply 172.16.1.253 is-at ce:23:5f:8f:7f:c8, length 28
11:12:15.054744 IP (tos 0x0, ttl 62, id 33539, offset 0, flags [none], proto ESP (50), length 104
    172.16.2.253 > 172.16.1.253: ESP(spi=0x12345678,seq=0x15c1b5), length 84
11:12:15.054867 IP (tos 0x0, ttl 62, id 33540, offset 0, flags [none], proto ESP (50), length 104
    172.16.2.253 > 172.16.1.253: ESP(spi=0x12345678,seq=0x15c1b6), length 84
11:12:15.054883 IP (tos 0x0, ttl 64, id 43900, offset 0, flags [none], proto ESP (50), length 104
    172.16.1.253 > 172.16.2.253: ESP(spi=0x12345678,seq=0x24ae5), length 84
^C
```

Fig. 22. ICMP Ping between P2 and P4 With L2TPV3 and IPsec

Dans la capture précédente (voir Figure [22]), nous constatons qu'il n'y a pas d'en-tête L2TPv3 (ip Proto 115) dans ces paquets. À la place, il contient un en-tête appelé Encapsulating Security Payload (ESP). Le protocole du paquet est également ESP. Nous ne voyons pas le champ de données dans ce paquet. La longueur du paquet est également supérieure à celui sans IPsec.

7 Accès Internet

7.1 Configuration du Switch Internet Et la machine réelle

Pour configurer un accès à Internet, nous avons effectué les configurations suivantes:

```
# On active le forwarding sur la machine physique
sysctl net.ipv4.conf.all.forwarding=1
sudo sysctl -w net.ipv4.conf.all.rp_filter=0

# On attribue une adresse IP pour l'interface "internet"
ip a add dev internet 10.87.0.254/24

# On configure la translation d'adresse (NAT) pour le sous-reseau
# → 10.87.0.0/24
iptables -t nat -A POSTROUTING -s 10.87.0.0/24 -j MASQUERADE

# Configuration des routes par d faut pour les routeurs A et B
ip netns exec rA ip route add default via 10.87.0.254
ip netns exec rB ip route add default via 10.87.0.254
```

A partir de là, depuis le routeur A ou B, on peut bien accéder à Internet.

```
ubuntu@ubuntu2004:~/infra_res/projet$ sudo ip netns exec rA ping -c 1 8.8.8.8
PING 8.8.8.8 (8.8.8.8) 56(84) bytes of data.
64 octets de 8.8.8.8 : icmp_seq=1 ttl=114 temps=14.8 ms

--- statistiques ping 8.8.8.8 ---
1 paquets transmis, 1 reçus, 0 % paquets perdus, temps 0 ms
rtt min/moy/max/mdev = 14,751/14,751/14,751/0,000 ms
ubuntu@ubuntu2004:~/infra_res/projet$ sudo ip netns exec rB ping -c 1 8.8.8.8
PING 8.8.8.8 (8.8.8.8) 56(84) bytes of data.
64 octets de 8.8.8.8 : icmp_seq=1 ttl=114 temps=17.6 ms

--- statistiques ping 8.8.8.8 ---
1 paquets transmis, 1 reçus, 0 % paquets perdus, temps 0 ms
rtt min/moy/max/mdev = 17,573/17,573/17,573/0,000 ms
ubuntu@ubuntu2004:~/infra_res/projet$ █
```

Fig. 23. Internet Access Enabled On Internet Bridge

7.2 Accès à Internet via Routeur 1:

Pour permettre le transit des paquets entre les postes P3 et P4 et Internet via le Routeur 1, nous avons procédé comme suit :

Puisque depuis rA, nous avons déjà un accès à Internet grâce à la configuration précédente, nous allons configurer une translation d'adresse de réseau (SNAT MASQUERADE) au niveau du Routeur 1. Cela permettra aux paquets de passer par le Routeur 1, puis par rA, avant d'accéder à Internet. De plus, cette configuration assurera également le retour des paquets de manière appropriée.

```
ip netns exec r1 iptables -t nat -A POSTROUTING -s 192.168.100.0/24
# → -j MASQUERADE
ip netns exec r1 iptables -t nat -A POSTROUTING -s 192.168.200.0/24
# → -j MASQUERADE
```

Maintenant, pour les machines P1 et P2, il suffit de repliquer sur le Routeur 2 la même configuration faite sur le routeur 1 mais il nous est demandé de faire passer tout le trafic Internet des machines via

le Routeur 1; Pour faire cela, on va modifier les routes par défaut de P2 et "P4 pour que cela soit R1 et non R2:

```
#On P1 VLAN 100
#delete existing default route which was (192.168.100.253 -> R2)
sudo ip netns exec p1 ip r del default
#adding the new default route (R1)
sudo ip netns exec p1 ip r add default via 192.168.100.254 dev p1-
    ↪ eth0.100

#On P2 VLAN 200
#delete existing default route (192.168.200.253 -> R2)
sudo ip netns exec p2 ip r del default
#adding the new default route (R1)
sudo ip netns exec p2 ip r add default via 192.168.200.254 dev p1-
    ↪ eth0.200
```

7.3 Verification du traffic Internet de P1 passant R1

On se met sur R1 pour lancer un traceroute d'un paquet à direction d'internet afin de prouver qu'il passe bien par le routeur 1. D'abord, revoici la configuration ip du routeur R1: on garde à l'oeil l'interface (tunel.100 : 192.168.100.254)

```
ubuntu@ubuntu2004:~/Infra_res/projet$ sudo ip netns exec r1 ip a
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group default qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
        valid_lft forever preferred_lft forever
    inet6 ::1/128 scope host
        valid_lft forever preferred_lft forever
4: l2tpeth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1458 qdisc fq_codel master tunnel state UNKNOWN group default qlen 1000
    link/ether ca:7a:5a:a2:d4:3c brd ff:ff:ff:ff:ff:ff
    inet6 fe80::c7a:5aff:fea2:d43c/64 scope link
        valid_lft forever preferred_lft forever
5: tunnel: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1458 qdisc noqueue state UP group default qlen 1000
    link/ether ba:b9:11:d1:65:45 brd ff:ff:ff:ff:ff:ff
    inet6 fe80::b8b9:11ff:fed1:6545/64 scope link
        valid_lft forever preferred_lft forever
6: tunnel.100@tunnel: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1458 qdisc noqueue state UP group default qlen 1000
    link/ether ba:b9:11:d1:65:45 brd ff:ff:ff:ff:ff:ff
    inet 192.168.100.254/24 scope global tunnel.100
        valid_lft forever preferred_lft forever
    inet6 fe80::b8b9:11ff:fed1:6545/64 scope link
        valid_lft forever preferred_lft forever
7: tunnel.200@tunnel: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1458 qdisc noqueue state UP group default qlen 1000
    link/ether ba:b9:11:d1:65:45 brd ff:ff:ff:ff:ff:ff
    inet 192.168.200.254/24 scope global tunnel.200
        valid_lft forever preferred_lft forever
    inet6 fe80::b8b9:11ff:fed1:6545/64 scope link
        valid_lft forever preferred_lft forever
228: r1-eth0@if227: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue state UP group default qlen 1000
    link/ether 9e:44:46:a4:3f:43 brd ff:ff:ff:ff:ff:ff link-netnsid 0
    inet 172.16.1.253/24 scope global r1-eth0
        valid_lft forever preferred_lft forever
    inet6 fe80::9c44:46ff:fea4:3f43/64 scope link
        valid_lft forever preferred_lft forever
230: r1-eth1@if229: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue master tunnel state UP group default qlen 1000
    link/ether ba:b9:11:d1:65:45 brd ff:ff:ff:ff:ff:ff link-netnsid 0
ubuntu@ubuntu2004:~/Infra_res/projet$
```

Fig. 24. R1 IP Configuration

Et maintenant, le paquet :

```
ubuntu@ubuntu2004:~/infra_res/projet$ sudo ip netns exec p1 ping -c 1 8.8.8.8
PING 8.8.8.8 (8.8.8.8) 56(84) bytes of data.
64 octets de 8.8.8.8 : icmp_seq=1 ttl=112 temps=17.0 ms

--- statistiques ping 8.8.8.8 ---
1 paquets transmis, 1 reçus, 0 % paquets perdus, temps 0 ms
rtt min/moy/max/mdev = 16,985/16,985/16,985/0,000 ms
ubuntu@ubuntu2004:~/infra_res/projet$ sudo ip netns exec p1 traceroute --icmp 8.8.8.8
traceroute to 8.8.8.8 (8.8.8.8), 64 hops max
  1  192.168.100.254  0,572ms  0,002ms  0,001ms
  2  172.16.1.254  0,000ms  0,002ms  0,001ms
  3  10.87.0.254  0,366ms  0,002ms  0,001ms
  4  10.0.2.2  0,930ms  2,918ms  0,615ms
  5  *  *  *
  6  *  *  *
  7  *  *  *
  8  *  *  *
  9  *  *  *
 10  *  *  *
 11  *  *  *
 12  8.8.8.8  14,119ms  18,534ms  13,600ms
ubuntu@ubuntu2004:~/infra_res/projet$
```

Fig. 25. Sucessful ping and Traceroute from P1 to Internet via R1

On voit bien que le traffic passe par l'interface tunnel.100 de r1. Ce qui confirme la configuration.

7.4 Passage de P1 et P2 Via R2 Pour Internet à l'aide de dnsmaq

Nous avons modifié dans le script de l'exemple précédent les routes par défaut de P1 et P2, ce qui fait que tous les postes possèdent une route par défaut vers le routeur 1. L'objectif dans cette question va être de modifier la route par défaut de poste 1 et 2 lorsqu'il obtient une adresse par DHCP. Il va donc falloir que la route par défaut soit le routeur 2. Comme sur R1, on considère avoir mis en place le SNAT sur R2 comme suit:

```
ip netns exec r2 iptables -t nat -A POSTROUTING -s 192.168.100.0/24
  ↳ -j MASQUERADE
ip netns exec r2 iptables -t nat -A POSTROUTING -s 192.168.200.0/24
  ↳ -j MASQUERADE
```

Il ne reste plus qu'à modifier la commande dnsmasq du côté du serveur DHCP pour indiquer aux postes configurés par DHCP d'utiliser une certaine route par défaut. Pour cela, nous ajoutons simplement les options suivantes à la commande dnsmasq :

- –dhcp-option=option:router,192.168.200.253
- –dhcp-option=option:router,192.168.100.253

DHCP Server :

```
sudo ip netns exec r1 dnsmasq -d -z -i tunnel.100 --dhcp-range
  ↳ =192.168.100.1,192.168.100.100,255.255.255.0 --dhcp-option
  ↳ =option:router,192.168.100.253
```

DHCP Client:

```
sudo ip netns exec p1 dhclient p1-eth0.100
```

On peut aussi lancer le même serveur DHCP sur routeur 2 pour les postes 1 et 2; ils obtiendront directement en plus de l'adresse leur route par défaut qui sera le routeur 2 et l'utiliseront pour aller sur Internet.

Et voici la nouvelle configuration IP de P1 : on voit bien qu'il a obtenu une nouvelle adresse et que sa route par défaut pointe maintenant vers le routeur 2.

```
ubuntu@ubuntu2004:~/infra_res/projet$ [p1] ip a
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group default qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
        inet 127.0.0.1/8 scope host lo
            valid_lft forever preferred_lft forever
        inet6 ::1/128 scope host
            valid_lft forever preferred_lft forever
2: p1-eth0@p1-eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue state UP group default qlen 1000
    link/ether b6:0f:0e:3e:ce:22 brd ff:ff:ff:ff:ff:ff
        inet 192.168.100.66/24 brd 192.168.100.255 scope global dynamic p1-eth0.100
            valid_lft 3195sec preferred_lft 3195sec
        inet6 fe80::b40f:eff:fe3e:ce22/64 scope link
            valid_lft forever preferred_lft forever
244: p1-eth0@if243: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue state UP group default qlen 1000
    link/ether b6:0f:0e:3e:ce:22 brd ff:ff:ff:ff:ff:ff link-netnsid 0
        inet6 fe80::b40f:eff:fe3e:ce22/64 scope link
            valid_lft forever preferred_lft forever
ubuntu@ubuntu2004:~/infra_res/projet$ [p1] ip r
default via 192.168.100.253 dev p1-eth0.100
192.168.100.0/24 dev p1-eth0.100 proto kernel scope link src 192.168.100.66
ubuntu@ubuntu2004:~/infra_res/projet$ [p1]
```

Fig. 26. P1 IP Address and Route

7.5 Test de l'Accès à Internet de P1 via le Routeur 2

Comme pour le test effectué avec le passage via R1 et sachant l'adresse de R2 pour le VLAN tunnel.100 (192.168.100.253), on va lancer un ping et un traceroute vers Internet pour prouver que le traffic de P1 passe bien par R2.

```
ubuntu@ubuntu2004:~/infra_res/projet$ [p1] ping -c 1 8.8.8.8
PING 8.8.8.8 (8.8.8.8) 56(84) bytes of data.
64 octets de 8.8.8.8 : icmp_seq=1 ttl=112 temps=29.4 ms

--- statistiques ping 8.8.8.8 ---
1 paquets transmis, 1 reçus, 0 % paquets perdus, temps 0 ms
rtt min/moy/max/mdev = 29,429/29,429/29,429/0,000 ms
ubuntu@ubuntu2004:~/infra_res/projet$ [p1] traceroute --icmp 8.8.8.8
traceroute to 8.8.8.8 (8.8.8.8), 64 hops max
  1  192.168.100.253  0,002ms  0,001ms  0,001ms
  2  172.16.2.254  0,002ms  0,001ms  0,001ms
  3  10.87.0.254  0,000ms  0,001ms  0,001ms
  4  10.0.2.2  0,578ms  0,531ms  0,656ms
  5  *  *  *
  6  *  *  *
  7  *  *  *
  8  *  *  *
  9  *  *  *
  10  *  *  *
  11  *  *  *
  12  8.8.8.8  20,024ms  14,555ms  14,288ms
ubuntu@ubuntu2004:~/infra_res/projet$ [p1]
```

Fig. 27. Sucessful Ping and Traceroute from P1 to Internet via R2

Ce même ping capturé sur l'interface internet avec tcpdump confirme encore plus que cela passe par R2:

```
ubuntu@ubuntu2004:~/infra_res/projet$ sudo ip netns exec r2 sudo tcpdump -nve
tcpdump: listening on tunnel, link-type EN10MB (Ethernet), capture size 262144 bytes
^C19:32:18.472768 b6:0f:0e:3e:ce:22 > 62:a2:1f:c1:8c:64, ethertype 802.1Q (0x8100), length 102: vlan 100, p 0,
  ether type IPv4, (tos 0x0, ttl 64, id 64336, offset 0, flags [DF], proto ICMP (1), length 84)
  192.168.100.66 > 8.8.8.8: ICMP echo request, id 24063, seq 1, length 64
19:32:18.491246 62:a2:1f:c1:8c:64 > b6:0f:0e:3e:ce:22, ethertype 802.1Q (0x8100), length 102: vlan 100, p 0,
  ether type IPv4, (tos 0x0, ttl 112, id 31598, offset 0, flags [none], proto ICMP (1), length 84)
  8.8.8.8 > 192.168.100.66: ICMP echo reply, id 24063, seq 1, length 64

2 packets captured
2 packets received by filter
0 packets dropped by kernel
```

Fig. 28. PING P1 to 8.8.8.8 captured on R2

Fichiers de configuration:

- **internet_access.sh** : Fichier contenant les configurations permettant à tous les postes d'accéder à Internet, avec P1 et P2 passant par R2, et P3 et P4 passant également par R1.
- **build_dhcp.sh** : Fichier contenant les configurations des serveurs DHCP sur R1 et R2.

8 Interdiction du trafic entre VLAN 100 et VLAN 200**8.1 Avec iptables:**

Le blocage des VLANs à l'aide de règles de pare-feu peut être réalisé de différentes manières. Une approche courante consiste à utiliser une politique de filtrage DROP et à n'accepter que les paquets destinés au même VLAN ou à Internet. Une méthode plus simple consiste à rejeter les paquets ayant une source dans un VLAN et une destination dans un autre VLAN. Cette action doit être mise en œuvre sur les routeurs 1 et 2.

```
# Sur R1, on bloque les paquets inter-vlans
ip netns exec r1 iptables -t filter -A FORWARD -s 192.168.200.0/24 -
  ↳ d 192.168.100.0/24 -j DROP
ip netns exec r1 iptables -t filter -A FORWARD -s 192.168.100.0/24 -
  ↳ d 192.168.200.0/24 -j DROP

# Pareil sur R2
ip netns exec r2 iptables -t filter -A FORWARD -s 192.168.200.0/24 -
  ↳ d 192.168.100.0/24 -j DROP
ip netns exec r2 iptables -t filter -A FORWARD -s 192.168.100.0/24 -
  ↳ d 192.168.200.0/24 -j DROP
```

8.2 A l'aide de la Policy Routing:

Nous allons avoir besoin de deux tables de routage, une pour les paquets de chaque vlan;

```

rt_tables
etc > iproute2 > rt_tables
1   #
2   # reserved values
3   #
4   255 local
5   254 main
6   253 default
7   0   unspec
8   #
9   # local
10  #
11  #1  inr.ruhep
12
13  1 vlan 100
14  2 vlan 200
15

```

Fig. 29. Creating two routing tables

Afin de mettre en œuvre cette approche, il est nécessaire de classifier l'origine des paquets. Les paquets provenant de 192.168.100.0 sont dirigés vers une table spécifique où il n'y a aucune route vers le réseau 192.168.200.0. De même, les paquets provenant de 192.168.200.0 sont dirigés vers une autre table spécifique où il n'existe aucune route vers le VLAN 192.168.100.0. Dans le cas où aucune règle de routage ne correspond, les paquets reviennent à la table principale (main).

```

# Sur r1
ip netns exec r1 ip rule add from 192.168.100.0/24 lookup vlan100
ip netns exec r1 ip rule add from 192.168.200.0/24 lookup vlan200
ip netns exec r1 ip route add prohibit 192.168.200.0/24 table
    ↵ vlan100
ip netns exec r1 ip route add prohibit 192.168.100.0/24 table
    ↵ vlan200

# Sur r2
ip netns exec r2 ip rule add from 192.168.100.0/24 lookup vlan100
ip netns exec r2 ip rule add from 192.168.200.0/24 lookup vlan200
ip netns exec r2 ip route add prohibit 192.168.200.0/24 table
    ↵ vlan100
ip netns exec r2 ip route add prohibit 192.168.100.0/24 table
    ↵ vlan200

```

Pour montrer le bon fonctionnement de ces règles, nous essayons de faire les pings suivants:

- VLAN 100 : Entre P3 (192.168.100.2) et P1 (192.168.100.1)

```
ubuntu@ubuntu2004:~/infra_res/projet$ sudo ip netns exec p3 ping 192.168.100.1
PING 192.168.100.1 (192.168.100.1) 56(84) bytes of data.
64 octets de 192.168.100.1 : icmp_seq=1 ttl=64 temps=2.28 ms
64 octets de 192.168.100.1 : icmp_seq=2 ttl=64 temps=0.085 ms
^C
--- statistiques ping 192.168.100.1 ---
2 paquets transmis, 2 reçus, 0 % paquets perdus, temps 1001 ms
rtt min/moy/max/mdev = 0,085/1,184/2,284/1,099 ms
```

Fig. 30. Ping Inside same VLAN

- Inter-VLANS : Entre P3 (192.168.100.2) et P4 (192.168.200.2)

```
ubuntu@ubuntu2004:~/infra_res/projet$ sudo ip netns exec p3 ping 192.168.200.2
PING 192.168.200.2 (192.168.200.2) 56(84) bytes of data.
De 192.168.100.254 icmp_seq=1 Paquet filtré
De 192.168.100.254 icmp_seq=2 Paquet filtré
De 192.168.100.254 icmp_seq=3 Paquet filtré
De 192.168.100.254 icmp_seq=4 Paquet filtré
^C
--- statistiques ping 192.168.200.2 ---
4 paquets transmis, 0 reçus, +4 erreurs, 100 % paquets perdus, temps 3044 ms
```

Fig. 31. Acess Inter-VLANS

Comme nous pouvons le voir sur les deux captures précédentes, nous ne pouvons pas effectuer de ping de p3 à p4 après l'application des configurations. Lors de l'utilisation de la cible "prohibit", nous recevons le message "Paquet Filtré". D'un autre côté, nous pouvons constater que les connexions entre les hôtes à l'intérieur de VLAN100 sont toujours normales.

9 Organisation du Projet:

Tous les scripts nécessaires pour le projet sont disponibles et peuvent être exécutés. Cependant, il est impératif de respecter l'ordre d'exécution correct pour assurer le bon fonctionnement du système.

- [build_projetv1.sh](#) : Met en place le schéma du réseau selon le premier schéma du sujet, sans VLAN.
- [build_projet2.sh](#) : Met en place le schéma du réseau selon le second schéma du sujet, avec configuration des VLANs.
- [l2tpv3_ip.sh](#) : Met en place le tunnel L2TPv3 en mode encapsulation IP.
- [l2tpv3_udp.sh](#) : Met en place le tunnel L2TPv3 en mode encapsulation UDP.
- [l2tpv3_ip_ipsec.sh](#) : Met en place le tunnel L2TPv3 en mode encapsulation IP avec chiffrement IPsec.
- [gre_greTAP.sh](#) : Met en place le tunnel GRE.
- [internet_access.sh](#) : Permet l'accès à Internet via les routeurs r1 et r2 (nécessite l'exécution de [l2tpv3_ip.sh](#)).
- [policy_access_prohibit.sh](#) : Blocage inter-VLANs à l'aide de la politique de routage.
- [iptables_access_prohibit.sh](#) : Blocage inter-VLANs à l'aide des règles iptables.
- [build_dhcp.sh](#) : Scripts pour la configuration du serveur et du client DHCP.

10 Conclusion

Dans ce projet, nous avons démontré les méthodes pour mettre en place un tunnel L2TPv3 sécurisé par IPsec pour l'implémentation du trunking VLAN dans les modes d'encapsulation IP et UDP. De plus, nous avons également comparé le protocole L2TPv3 avec VXLAN et GRE. Enfin, nous avons réussi à mettre en place un accès à Internet "intelligent" et à utiliser des règles iptables, ainsi que le Policy Routing pour interdire le trafic entre différents VLANs.

11 References

1. L2TPv3 <https://www.yamaha.com/products/en/network/techdocs/vpn/l2tpv3/>
 2. VXLAN <https://docs.openvswitch.org/en/latest/faq/vxlan/> <https://www.pcwldd.com/vxlan>
 3. IPsec <https://fr.wikipedia.org/wiki/IPsec>
 4. MPLS https://fr.wikipedia.org/wiki/Multiprotocol_Label_Switching <https://bit.ly/3byI1wp>
5. Autres références :
- Cours d'Infrastructure : <https://p-fb.net/>
 - Layer 2 Tunneling Protocol Version 3 : https://fr.wikipedia.org/wiki/Layer_2_Tunneling_Protocol_Version_3
 - Virtual Extensible LAN : https://fr.wikipedia.org/wiki/Virtual_Extensible_LAN
 - Article sur MPLS vs VPN IPsec : <https://www.netify.co.uk/learning/mpls-vs-vpn-ipsec>
 - Article sur VPN vs MPLS : <https://community.fs.com/blog/vpn-vs-mpls-difference.html>
 - Introduction rapide au Policy Routing Linux : <https://blog.scottlowe.org/2013/05/29/a-quick-introduction-to-linux-policy-routing>