
M1 CRYPTIS

Report Network Infrastructure

presented by
DO Duy Huy Hoang
NGUYEN Thi Mai Phuong

May 16, 2021

1 Introduction

The goal of the project is to study the L2TPv3 protocol for making level 2 tunnels. It will be used to "trunking" VLANs and pooling services such as DHCP. Finally, using an "intelligent" configuration, we will limit the use of the tunnel when the hosts want to connect to the Internet by allowing them to do it directly from their "Internet side".

2 Implementation

2.1 Implement network architecture

Create the network

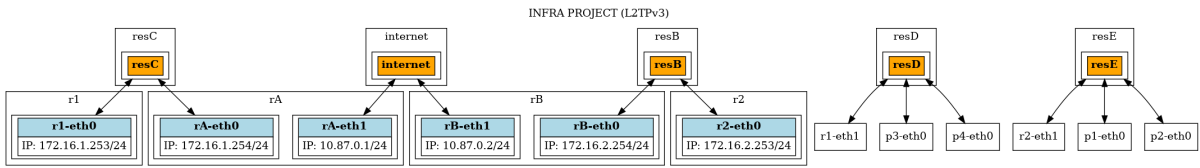


Figure 1: Network architecture

We configured the network following the proposed scheme, drawing inspiration from the TPs. Figure 1 shows our network architecture. The initial interface, here are **r1-eth1** and **r2-eth1** do not have IP configuration.

Configuration file: **build_architecture**.

DHCP server

A DHCP server will be implemented on router **r1** to configure the various hosts of the two VLANs (whether or not they are separated by the Internet). To start the DHCP server to assign IP addresses automatically, we use the following commands.

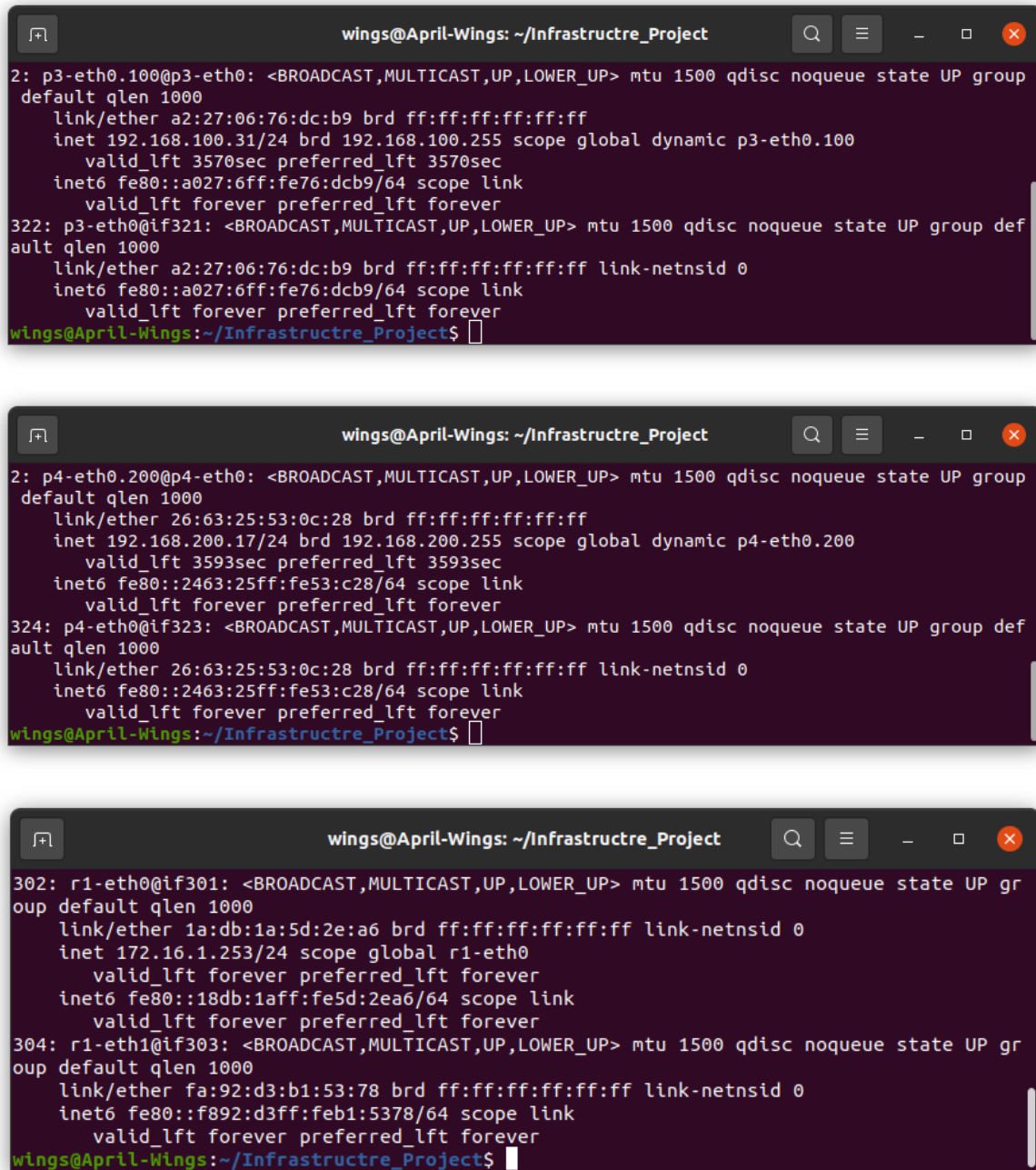
```

1 # run DHCP sever
2 sudo ip netns exec r1 dnsmasq -d -z -i r1-eth1.100 -F 192.168.100.0,
   ↪ 192.168.100.60, 255.255.255.0
3 sudo ip netns exec r1 dnsmasq -d -z -i r1-eth1.200 -F 192.168.200.0,
   ↪ 192.168.200.60, 255.255.255.0
4 # connect to get IP address
5 sudo ip netns exec p3 dhclient p3-eth0.100
6 sudo ip netns exec p4 dhclient p4-eth0.200

```

Then, we obtains the configuration of host **p3**, host **p4** and interface **r1-eth1**.

- **p3**: MAC Address = a2:27:06:76:dc:b9, IP Address = 192.168.100.31
- **p4**: MAC Address = 26:63:25:53:0c:28, IP Address = 192.168.200.17
- **r1-eth1**: MAC Address = fa:92:d3:b1:53:78



```
wings@April-Wings: ~/Infrastructre_Project
2: p3-eth0.100@p3-eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue state UP group
default qlen 1000
    link/ether a2:27:06:76:dc:b9 brd ff:ff:ff:ff:ff:ff
    inet 192.168.100.31/24 brd 192.168.100.255 scope global dynamic p3-eth0.100
        valid_lft 3570sec preferred_lft 3570sec
    inet6 fe80::a027:6ff:fe76:dcb9/64 scope link
        valid_lft forever preferred_lft forever
322: p3-eth0@if321: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue state UP group def
ault qlen 1000
    link/ether a2:27:06:76:dc:b9 brd ff:ff:ff:ff:ff:ff link-netnsid 0
    inet6 fe80::a027:6ff:fe76:dcb9/64 scope link
        valid_lft forever preferred_lft forever
wings@April-Wings:~/Infrastructre_Project$

wings@April-Wings: ~/Infrastructre_Project
2: p4-eth0.200@p4-eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue state UP group
default qlen 1000
    link/ether 26:63:25:53:0c:28 brd ff:ff:ff:ff:ff:ff
    inet 192.168.200.17/24 brd 192.168.200.255 scope global dynamic p4-eth0.200
        valid_lft 3593sec preferred_lft 3593sec
    inet6 fe80::2463:25ff:fe53:c28/64 scope link
        valid_lft forever preferred_lft forever
324: p4-eth0@if323: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue state UP group def
ault qlen 1000
    link/ether 26:63:25:53:0c:28 brd ff:ff:ff:ff:ff:ff link-netnsid 0
    inet6 fe80::2463:25ff:fe53:c28/64 scope link
        valid_lft forever preferred_lft forever
wings@April-Wings:~/Infrastructre_Project$

wings@April-Wings: ~/Infrastructre_Project
302: r1-eth0@if301: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue state UP gr
oup default qlen 1000
    link/ether 1a:db:1a:5d:2e:a6 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::18db:1aff:fe5d:2ea6/64 scope link
        valid_lft forever preferred_lft forever
304: r1-eth1@if303: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue state UP gr
oup default qlen 1000
    link/ether fa:92:d3:b1:53:78 brd ff:ff:ff:ff:ff:ff link-netnsid 0
    inet6 fe80::f892:d3ff:feb1:5378/64 scope link
        valid_lft forever preferred_lft forever
wings@April-Wings:~/Infrastructre_Project$
```

Figure 2: Configuration of r1-eth1, p3 and p4

Sniffing network traffic

We test the proper functioning of the VLAN encapsulation by "sniffing" the network using `tcpdump`. In this part, we captured traffics on the `r1-eth1` interface of router `r1`.

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

Now we try a ping between `p3` and `p4`. The traffic was captured on the `r1-eth1` interface was presented in Figure 3.

```
wings@April-Wings: ~/Infrastructre_Project
tcpdump: listening on r1-eth1, link-type EN10MB (Ethernet), capture size 262144 bytes
01:31:11.247289 a2:27:06:76:dc:b9 > fa:92:d3:b1:53:78, ethertype 802.1Q (0x8100), length 102: vlan 100,
p 0, ethertype IPv4, (tos 0x0, ttl 64, id 25668, offset 0, flags [DF], proto ICMP (1), length 84)
192.168.100.31 > 192.168.200.17: ICMP echo request, id 29087, seq 1, length 64
01:31:11.247312 fa:92:d3:b1:53:78 > 26:63:25:53:0c:28, ethertype 802.1Q (0x8100), length 102: vlan 200,
p 0, ethertype IPv4, (tos 0x0, ttl 63, id 25668, offset 0, flags [DF], proto ICMP (1), length 84)
192.168.100.31 > 192.168.200.17: ICMP echo request, id 29087, seq 1, length 64
01:31:11.247457 26:63:25:53:0c:28 > fa:92:d3:b1:53:78, ethertype 802.1Q (0x8100), length 102: vlan 200,
p 0, ethertype IPv4, (tos 0x0, ttl 64, id 18654, offset 0, flags [none], proto ICMP (1), length 84)
192.168.200.17 > 192.168.100.31: ICMP echo reply, id 29087, seq 1, length 64
01:31:11.247467 fa:92:d3:b1:53:78 > a2:27:06:76:dc:b9, ethertype 802.1Q (0x8100), length 102: vlan 100,
p 0, ethertype IPv4, (tos 0x0, ttl 63, id 18654, offset 0, flags [none], proto ICMP (1), length 84)
192.168.200.17 > 192.168.100.31: ICMP echo reply, id 29087, seq 1, length 64
01:31:12.250617 a2:27:06:76:dc:b9 > fa:92:d3:b1:53:78, ethertype 802.1Q (0x8100), length 102: vlan 100,
p 0, ethertype IPv4, (tos 0x0, ttl 64, id 25813, offset 0, flags [DF], proto ICMP (1), length 84)
```

Figure 3: Capture VLAN traffic between p3 and p4

We can see that the first packet is an ARP ping between p3 and the interface `r1-eth1`, VLAN 100. And in the next packet, the interface `r1-eth1` of router `r1` makes an ARP to reach p4 and transmit the ICMP packet, VLAN 200.

VLAN configuration file: *config-VLAN*.

2.2 L2TPv3 protocol and VXLANs technology

Compare L2TPv3 and VxLAN

L2TPv3 or Layer 2 Tunnel Protocol Version 3, is a tunneling protocol that provides a VPN connection in the data link layer. The L2 frames are encapsulated as IP packets to enable the transfer of L2 frames between routers, allowing networks on the same segment to be established at multiple branches. There are two encapsulation methods: **L2TPv3 over IP** which encapsulates frames as IP packets and **L2TPv3 over UDP** which encapsulates frames as UDP packets.

Meanwhile, **VXLAN** or Virtual Extensible LAN, is an evolution of efforts to standardize an overlay encapsulation protocol. In the past, Data Centers used VLAN to configure for Layer 2 isolation. Since the scale grew and the need to expand Layer 2 networks in the Data Center increased, the limitations of VLANs began to surface. The introduction of VXLAN overcomes these limitations of the VLAN technology itself. We already know that VLAN uses the tag on the layer 2 frame for encapsulation and can scale up to 4000 VLANs. VXLAN, on the other hand, encapsulates the MAC in UDP and is capable of scaling up to 16 million VxLAN segments.

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

Table 1: Comparison of **L2TPv3** and **VXLAN**

Implement VXLAN on Linux (Open vSwitch)

By default, Open vSwitch will use the assigned IANA port for VXLAN, which is 4789. However, it is possible to configure the destination UDP port manually on a per-VXLAN tunnel basis. An example of this configuration is provided below.:

```
1 add-br br0
2 add-port br0 vxlan1 -- set interface vxlan1 type=vxlan
   ↪ options:remote_ip=192.168.1.2 options:key=flow options:dst_port=8472
```

L2TPv3/VXLAN traffic encryption

L2TPv3 and VXLAN are as secure as their underlying network. So if we need security, use them over an encrypted transport, like IPsec. IPsec or Internet Protocol Security is a secure network protocol suite that authenticates and encrypts the packets of data to provide secure encrypted communication between two computers over an Internet Protocol network. It can be implemented in a host-to-host transport mode, as well as in a network tunneling mode. And for this project, we will use the IPsec transport mode, which only encrypts the payload of the IP packet. The implementation details will be shown in subsection 2.6.

Compare with MPLS

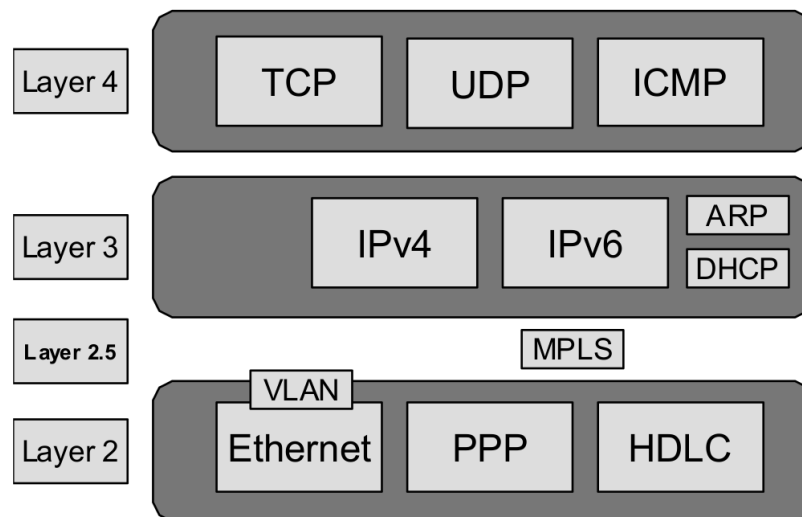


Figure 4: Multiprotocol Label Switching

MPLS or Multiprotocol Label Switching is a routing technique in telecommunications networks that directs data from one node to the next based on short path labels rather than long network addresses, thus avoiding complex lookups in a routing table and speeding traffic flows. It lies between layer 2 and 3 of the model OSI the Data Link and the Network Layer. That's why it is also known as 2.5 layer protocol.

Both MPLS and VXLAN require specific hardware support to operate at line rate but VXLAN only requires hardware support for encapsulation at the edge of the network and thus network cores do not necessarily need replacing. MPLS demands end to end support. In another hand, we have L2TPv3 which is a technology that caters more for L2 forwarding between edges, meanwhile MPLS is more of a core technology.

2.3 L2TPv3 tunnel in IP encapsulation mode

In this section, follow the requirement of question 3, we implement the L2TPv3 tunnel in IP encapsulation mode between r1 and r2.

Configuration file: *config_L2TPv3_IP*.

ARP protocol

To verify that ARP protocol works normally, we sniff our network by using `tcpdump` on the `tunnel` interface of `r1` and make a ping between `p1` and `p4`. Figure 5 displays the result of command line `tcpdump`.

```
wings@April-Wings: ~/Infrastructure_Project

inet6 fe80::d0fd:5ff:fe34:7ed4/64 scope link
    valid_lft forever preferred_lft forever
wings@April-Wings:~/Infrastructure_Project$ sudo ip netns exec p4 ping 192.168.100.104
PING 192.168.100.104 (192.168.100.104) 56(84) bytes of data.
64 bytes from 192.168.100.104: icmp_seq=1 ttl=63 time=0.742 ms
64 bytes from 192.168.100.104: icmp_seq=2 ttl=63 time=0.292 ms
64 bytes from 192.168.100.104: icmp_seq=3 ttl=63 time=0.293 ms
64 bytes from 192.168.100.104: icmp_seq=4 ttl=63 time=0.297 ms
^C
--- 192.168.100.104 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 3055ms
rtt min/avg/max/mdev = 0.292/0.406/0.742/0.194 ms
wings@April-Wings:~/Infrastructure_Project$
```

```
wings@April-Wings:~/Infrastructure_Project$ sudo ip netns exec r1 tcpdump -lnvv -i tunnel -e vlan
tcpdump: listening on tunnel, link-type EN10MB (Ethernet), capture size 262144 bytes
02:32:41.643587 72:01:3f:77:af:46 > 26:a9:b3:e2:d5:59, ethertype 802.1Q (0x8100), length 102: vlan 200,
p 0, ethertype IPv4, (tos 0x0, ttl 64, id 5787, offset 0, flags [DF], proto ICMP (1), length 84)
    192.168.200.17 > 192.168.100.104: ICMP echo request, id 32756, seq 1, length 64
02:32:41.643610 26:a9:b3:e2:d5:59 > 0e:2a:2c:75:45:f3, ethertype 802.1Q (0x8100), length 102: vlan 100,
p 0, ethertype IPv4, (tos 0x0, ttl 63, id 5787, offset 0, flags [DF], proto ICMP (1), length 84)
    192.168.200.17 > 192.168.100.104: ICMP echo request, id 32756, seq 1, length 64
02:32:41.644138 0e:2a:2c:75:45:f3 > 26:a9:b3:e2:d5:59, ethertype 802.1Q (0x8100), length 102: vlan 100,
p 0, ethertype IPv4, (tos 0x0, ttl 64, id 51029, offset 0, flags [none], proto ICMP (1), length 84)
    192.168.100.104 > 192.168.200.17: ICMP echo reply, id 32756, seq 1, length 64
02:32:41.644151 26:a9:b3:e2:d5:59 > 72:01:3f:77:af:46, ethertype 802.1Q (0x8100), length 102: vlan 200,
p 0, ethertype IPv4, (tos 0x0, ttl 63, id 51029, offset 0, flags [none], proto ICMP (1), length 84)
    192.168.100.104 > 192.168.200.17: ICMP echo reply, id 32756, seq 1, length 64
02:32:42.650395 72:01:3f:77:af:46 > 26:a9:b3:e2:d5:59, ethertype 802.1Q (0x8100), length 102: vlan 200,
p 0, ethertype IPv4, (tos 0x0, ttl 64, id 5945, offset 0, flags [DF], proto ICMP (1), length 84)
    192.168.200.17 > 192.168.100.104: ICMP echo request, id 32756, seq 2, length 64
02:32:42.650443 26:a9:b3:e2:d5:59 > 0e:2a:2c:75:45:f3, ethertype 802.1Q (0x8100), length 102: vlan 100,
p 0, ethertype IPv4, (tos 0x0, ttl 63, id 5945, offset 0, flags [DF], proto ICMP (1), length 84)
    192.168.200.17 > 192.168.100.104: ICMP echo request, id 32756, seq 2, length 64
02:32:42.650597 0e:2a:2c:75:45:f3 > 26:a9:b3:e2:d5:59, ethertype 802.1Q (0x8100), length 102: vlan 100,
```

Figure 5: Verify ARP protocol

DHCP service

With the L2TPv3 tunnel, we can run the DHCP server to assign IP addresses automatically not only for p3 and p4 but also for p1 and p2. To start DHCP server, we execute the following command:

```

1 # run DHCP sever
2 sudo ip netns exec r1 dnsmasq -d -z -i tunnel.100 -F
   ↪ 192.168.100.0,192.168.100.180,255.255.255.0
3 sudo ip netns exec r1 dnsmasq -d -z -i tunnel.200 -F
   ↪ 192.168.200.0,192.168.200.180,255.255.255.0
4 # connect to get IP address
5 sudo ip netns exec p1 dhclient p1-eth0.100
6 sudo ip netns exec p3 dhclient p3-eth0.100
7 sudo ip netns exec p4 dhclient p4-eth0.200
8 sudo ip netns exec p2 dhclient p2-eth0.200

```

Figure 6 displays the log of DHCP Server on r1 when p2 connect to get the IP address and result on p2.

The figure consists of two terminal screenshots. The top screenshot shows the DHCP server logs on r1, where the server is configured to serve two networks: 192.168.100.0/24 and 192.168.200.0/24. The logs show the server receiving a DHCPDISCOVER from p2-eth0.200, offering an IP address of 192.168.200.176, and the client accepting the offer. The bottom screenshot shows the network configuration for p2-eth0.200, which has been assigned the IP address 192.168.200.176 and the MAC address d2:fd:05:34:7e:d4. The configuration also shows the link's MTU, QoS, and other parameters.

```

wings@April-Wings: ~/Infrastructre_Project
dnsmasq: reading /etc/resolv.conf
dnsmasq: using nameserver 127.0.0.53#53
dnsmasq: reading /etc/resolv.conf
dnsmasq: using nameserver 127.0.0.53#53
dnsmasq-dhcp: DHCPDISCOVER(tunnel.200) 192.168.200.84 d2:fd:05:34:7e:d4
dnsmasq-dhcp: DHCPOFFER(tunnel.200) 192.168.200.176 d2:fd:05:34:7e:d4
dnsmasq-dhcp: DHCPREQUEST(tunnel.200) 192.168.200.176 d2:fd:05:34:7e:d4
dnsmasq-dhcp: DHCPACK(tunnel.200) 192.168.200.176 d2:fd:05:34:7e:d4 April-Wings
dnsmasq-dhcp: not giving name April-Wings to the DHCP lease of 192.168.200.176 because the name exists i
n /etc/hosts with address 127.0.1.1

wings@April-Wings: ~/Infrastructre_Project
2: p2-eth0.200@p2-eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue state UP group default
t qlen 1000
    link/ether d2:fd:05:34:7e:d4 brd ff:ff:ff:ff:ff:ff
    inet 192.168.200.176/24 brd 192.168.200.255 scope global dynamic p2-eth0.200
        valid_lft 3592sec preferred_lft 3592sec
    inet6 fe80::d0fd:5ff:fe34:7ed4/64 scope link
        valid_lft forever preferred_lft forever
349: p2-eth0@if348: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue state UP group default ql
en 1000
    link/ether d2:fd:05:34:7e:d4 brd ff:ff:ff:ff:ff:ff link-netnsid 0
    inet6 fe80::d0fd:5ff:fe34:7ed4/64 scope link
        valid_lft forever preferred_lft forever
wings@April-Wings:~/Infrastructre_Project$

```

Figure 6: Verify DHCP service

TCP connection

We now establish a TCP connection using socat between r1 and p1 through tunnel as shown in Figure 7. As we can see, we can exchange message between r1 and p1 (interface tunnel.100 with address 192.168.100.254) through the tunnel on port 4444.

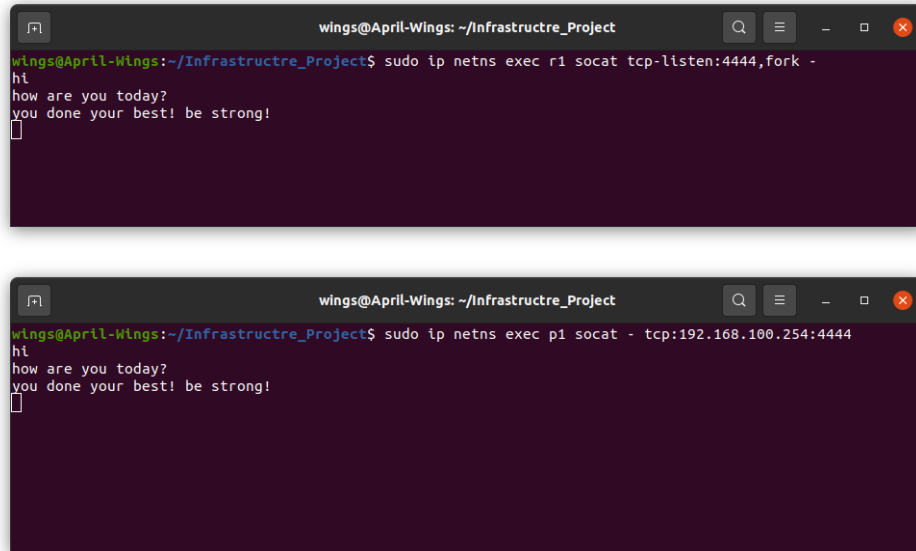


Figure 7: Verify TCP connection

2.4 Compare IP and UDP encapsulation mode of L2TPv3

To observe the difference between L2TPv3 tunneling in IP and UDP mode, we will implement the L2TPv3 tunnel with UDP encapsulation mode to our network and using ports 2020 and 2021.

Configuration file: *config_L2TPv3_UDP*.

Then, with each mode, we sniffed packets on the *rA* and do a ping from *p3* to *p2*. Finally, we analyzed these packets with **Wireshark**.

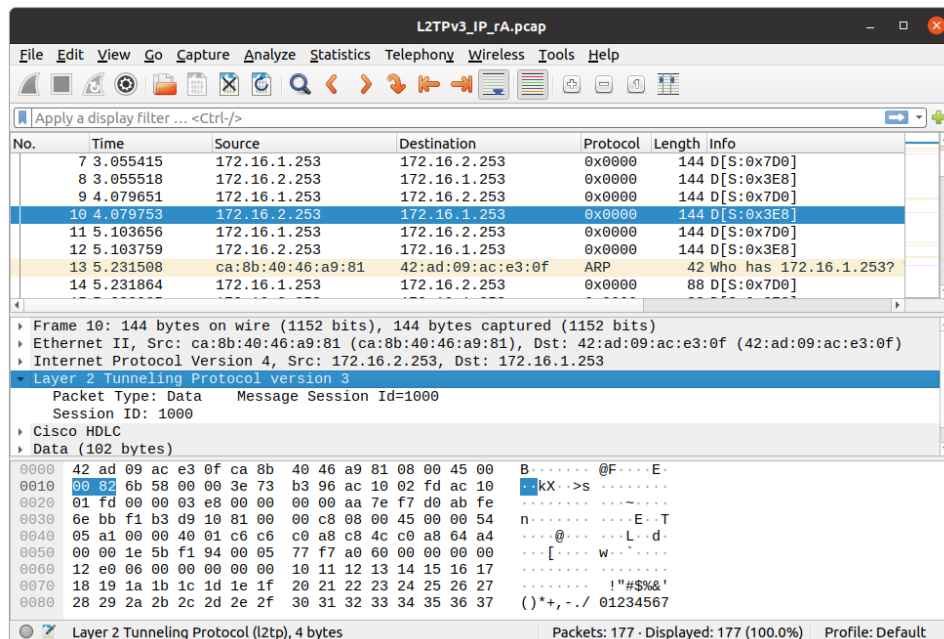


Figure 8: L2TPv3 in IP encapsulation mode

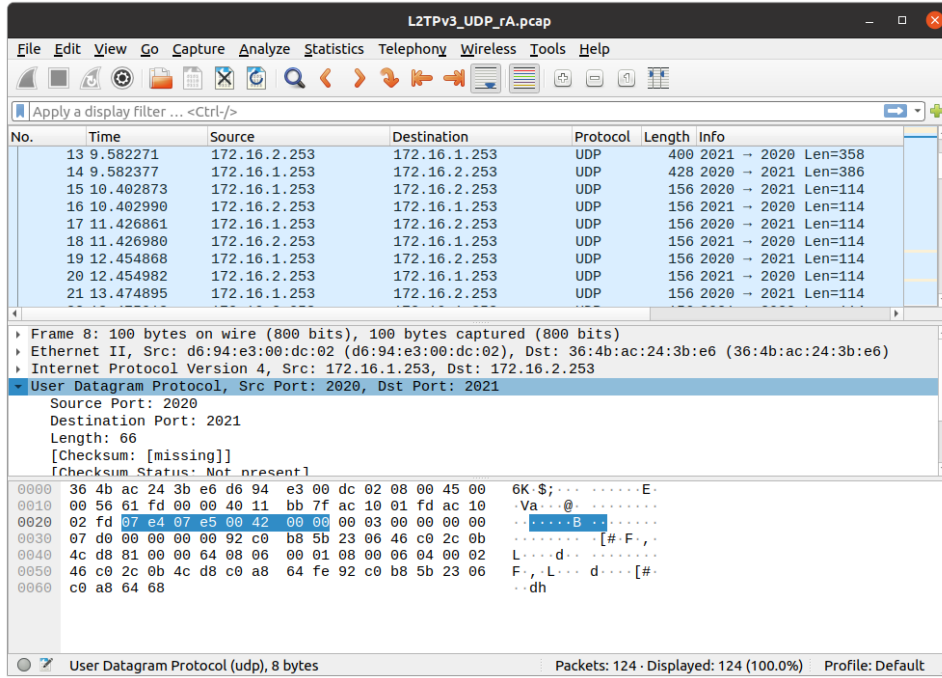


Figure 9: L2TPv3 in UDP encapsulation mode

As we can see, although we ping with the IP addresses of p3 (belong to VLAN100) and p2 (belong to VLAN200), the source IP and destination IP in these packets are 172.16.1.253 (interface `rout1-eth0`) and 172.16.2.253 (interface `rout2-eth0`).

In Figure 8, when we use L2TPv3 in IP encapsulation mode, packets contain a L2TPv3 header. Meanwhile, in Figure 9, when we use L2TPv3 in UDP encapsulation mode, packets contain an UDP header with the ports 2020 and 2021, same as our configuration. Besides, the length of packets with mode UDP is 156 bytes, greater than the one with mode IP, which is 144 bytes. This caused by the difference length of these header. Moreover, protocols of packets in two mode are also different. In IP encapsulation mode, the protocol identifier is 0x0000 and marks as `Unknown`. Meanwhile, in UDP encapsulation mode, the protocol is UDP.

Pcap files: *L2TPv3_IP_rA.pcap* and *L2TPv3_UDP_rA.pcap*.

2.5 L2TPv3 tunnel and GRE tunnel

GRE or Generic routing encapsulation is one of the available tunneling mechanisms that can encapsulate a wide variety of protocol packet types inside IP tunnels. It provides a private path for transporting packets through an otherwise public network by encapsulating or tunneling the packets. We use GRE tunnel when we need to form a neighborhood between 2 routers and transport packets of one protocol over another protocol.

In this section, we will creating tunnel GRE with mode GRE-TAP, compare it to the L2TPv3 tunnel.

Configuration file: *config-GRE*.

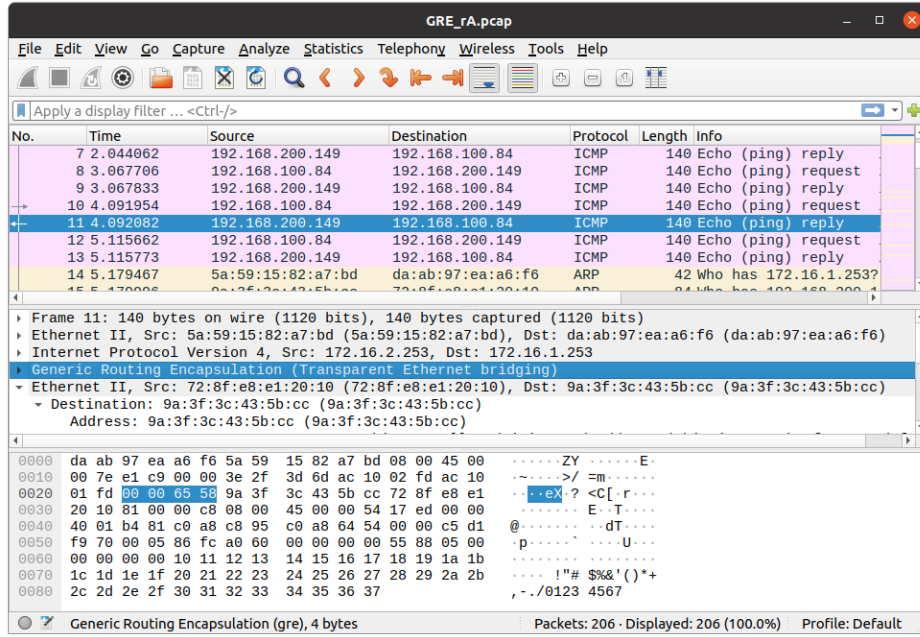


Figure 10: GRE tunnel with GRE-TAP mode

Figure 10 displays a captured traffic on a tunnel that configure with GRE in GRE-TAP mode. We notice that an Ethernet header and an IP header (192.168.200.149 and 192.168.100.84) is wrapped in side another Ethernet header and another IP header (172.16.2.253 and 172.16.1.253). For Maximum transmission unit or MTU, we see it on GRE is 1476, L2TPv3 in mode IP is 1458 and L2TPv3 in mode UDP is 1446. They are different because of different lengths of headers.

Pcap file: *GRE_rA.pcap*

2.6 Implement IPsec on L2TPv3 tunnel

Using TP files, we set up IPsec encryption on our L2TPv3 tunnel by IP encapsulation. The configuration on r1 as the below.

```

1 # Flush the SAD and SPD
2 ip netns exec r1 ip xfrm state flush
3 ip netns exec r1 ip xfrm policy flush
4 # AH SAs using 256 bit long and ESP SAs using 160 bit long keys
5 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
  → 0x323730ed6f1b9ff0cb084af15b197e862b7c18424a7cdfb74cd385ae23bc4f17 enc
  → "rfc3686(ctr(aes))" 0x27b90b8aec1ee32a8150a664e8faac761e2d305b
6 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
  → 0x44d65c50b7581fd3c8169cf1fa0ebb24e0d55755b1dc43a98b539bb144f2067f enc
  → "rfc3686(ctr(aes))" 0x9df7983cb7c7eb2af01d88d36e462b5f01d10bc1
7 # Security policies
8 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
9 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

```

Then, we execute the similar commands on r2.

Configuration file: *config_L2TPv3_IP_IPSec*.

In the next step, we capture packets through tunnel to see the effect of IPsec. As we can see in Figure 11, there is no L2TPv3 header in this packet. Instead, it has a header named Encapsulating Security Payload (ESP). The protocol of packet is also ESP. We do not see the field of data in this packet. Length of packet is 174 bytes, greater than the one without IPsec.

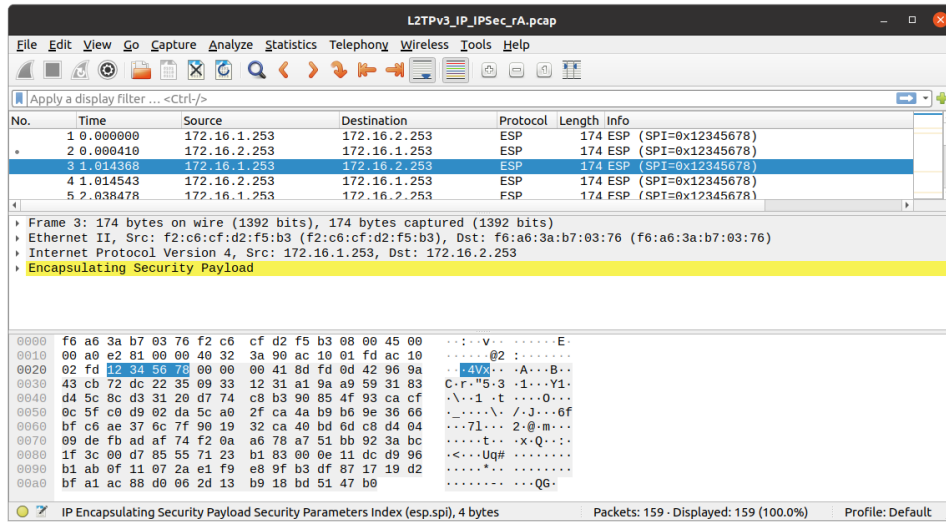


Figure 11: Speed of transferring without IPsec

Now, we use the `iperf` command to get and compare the transfer speed between r1 and p1 with and without the encryption.

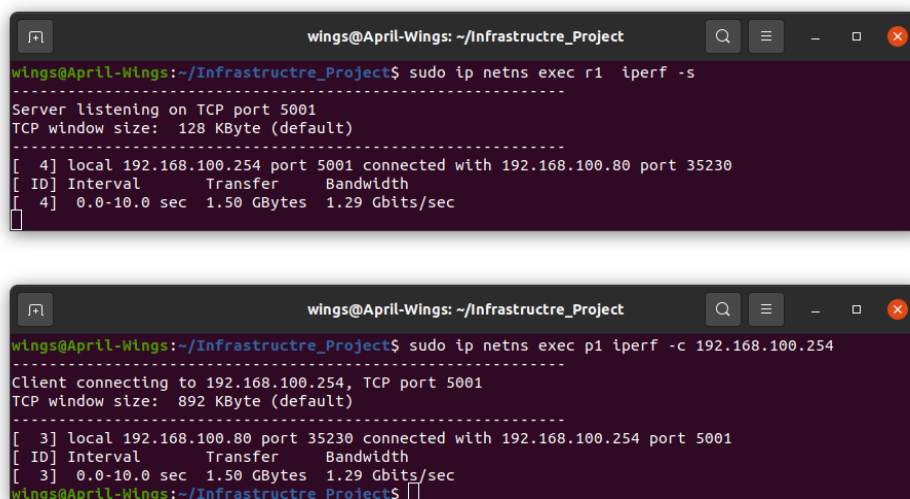


Figure 12: Speed of transferring without IPsec

```
wings@April-Wings: ~/Infrastructure_Project
wings@April-Wings:~/Infrastructure_Project$ sudo ip netns exec r iperf -s
Server listening on TCP port 5001
TCP window size: 128 KByte (default)
-----
[  4] local 192.168.100.254 port 5001 connected with 192.168.100.17 port 57344
[ ID] Interval      Transfer    Bandwidth
[  4] 0.0-10.0 sec  589 MBytes  493 Mbits/sec
wings@April-Wings:~/Infrastructure_Project$

wings@April-Wings: ~/Infrastructure_Project
wings@April-Wings:~/Infrastructure_Project$ sudo ip netns exec p1 iperf -c 192.168.100.254
Client connecting to 192.168.100.254, TCP port 5001
TCP window size: 204 KByte (default)
-----
[  3] local 192.168.100.17 port 57344 connected with 192.168.100.254 port 5001
[ ID] Interval      Transfer    Bandwidth
[  3] 0.0-10.0 sec  589 MBytes  494 Mbits/sec
wings@April-Wings:~/Infrastructure_Project$
```

Figure 13: Speed of transferring with IPsec

In the Figure 12, we can notice that for an interval of 10 seconds, `iperf` has transfer 1.50 GBytes and 1.29 GBits/sec Bandwidth without IPsec. In case with IPsec, the value of transfer is 589 MBytes and Bandwidth is 494 Mbits/sec. It seems that L2TPv3 without encryption is 2.5 times faster than L2TPv3 with encryption. This happens because IPsec encapsulates the IP packets. As a result, the IP packet becomes longer and this leads to fragmentation. The receiver also needs to reassemble packets. Fragmentation, reassembly, encryption and decryption of fragments consume many CPU resources.

2.7 "Intelligent" Internet access

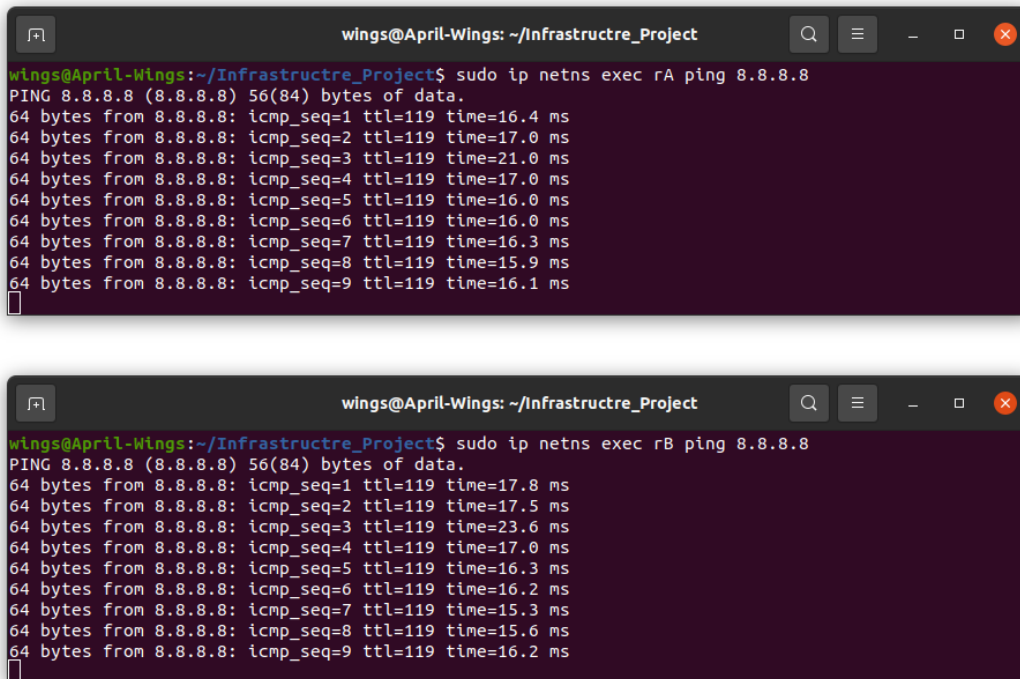
Configuration file: *config_Internet_Access*

Access to the Internet

To check our internet connection, we use the IPv4 addresses 8.8.8.8. With `rA` and `rB`, we execute the following commands:

```
1 # internet
2 ip a add dev internet 10.87.0.254/24
3 iptables -t nat -A POSTROUTING -s 10.87.0.0/24 -j MASQUERADE
4 # rA
5 ip netns exec rA ip route add default via 10.87.0.254
6 ip netns exec rA iptables -t nat -A POSTROUTING -s 172.16.1.0/24 -j MASQUERADE
7 # rB
8 ip netns exec rB ip route add default via 10.87.0.254
9 ip netns exec rB iptables -t nat -A POSTROUTING -s 172.16.2.0/24 -j MASQUERADE
```

Then, we test internet connection of `rA` and `rB` by the pings to address 8.8.8.8.



The figure shows two terminal windows. The top window shows the command `sudo ip netns exec rA ping 8.8.8.8` and its output, which is a successful ping to 8.8.8.8 with 9 packets, each 64 bytes, with various TTL and time values. The bottom window shows the command `sudo ip netns exec rB ping 8.8.8.8` and its output, which is also a successful ping to 8.8.8.8 with 9 packets, each 64 bytes, with various TTL and time values.

```
wings@April-Wings: ~/Infrastructre_Project
wings@April-Wings:~/Infrastructre_Project$ sudo ip netns exec rA ping 8.8.8.8
PING 8.8.8.8 (8.8.8.8) 56(84) bytes of data.
64 bytes from 8.8.8.8: icmp_seq=1 ttl=119 time=16.4 ms
64 bytes from 8.8.8.8: icmp_seq=2 ttl=119 time=17.0 ms
64 bytes from 8.8.8.8: icmp_seq=3 ttl=119 time=21.0 ms
64 bytes from 8.8.8.8: icmp_seq=4 ttl=119 time=17.0 ms
64 bytes from 8.8.8.8: icmp_seq=5 ttl=119 time=16.0 ms
64 bytes from 8.8.8.8: icmp_seq=6 ttl=119 time=16.0 ms
64 bytes from 8.8.8.8: icmp_seq=7 ttl=119 time=16.3 ms
64 bytes from 8.8.8.8: icmp_seq=8 ttl=119 time=15.9 ms
64 bytes from 8.8.8.8: icmp_seq=9 ttl=119 time=16.1 ms

wings@April-Wings: ~/Infrastructre_Project
wings@April-Wings:~/Infrastructre_Project$ sudo ip netns exec rB ping 8.8.8.8
PING 8.8.8.8 (8.8.8.8) 56(84) bytes of data.
64 bytes from 8.8.8.8: icmp_seq=1 ttl=119 time=17.8 ms
64 bytes from 8.8.8.8: icmp_seq=2 ttl=119 time=17.5 ms
64 bytes from 8.8.8.8: icmp_seq=3 ttl=119 time=23.6 ms
64 bytes from 8.8.8.8: icmp_seq=4 ttl=119 time=17.0 ms
64 bytes from 8.8.8.8: icmp_seq=5 ttl=119 time=16.3 ms
64 bytes from 8.8.8.8: icmp_seq=6 ttl=119 time=16.2 ms
64 bytes from 8.8.8.8: icmp_seq=7 ttl=119 time=15.3 ms
64 bytes from 8.8.8.8: icmp_seq=8 ttl=119 time=15.6 ms
64 bytes from 8.8.8.8: icmp_seq=9 ttl=119 time=16.2 ms
```

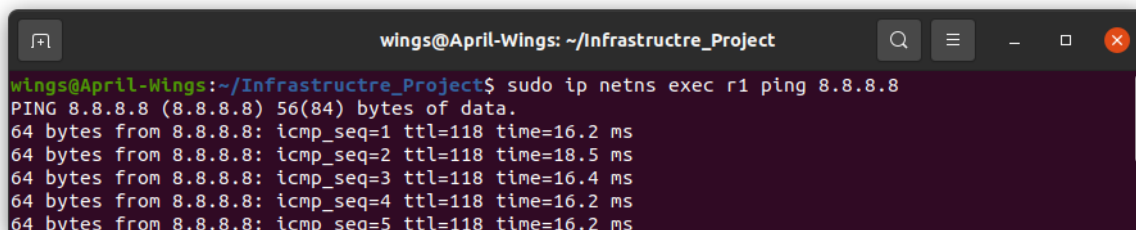
Figure 14: Access Internet from rA and rB

Internet access on r1 and r2

We add Internet access on r1 and also r2 for the hosts of the two VLANs, using rules iptables.

```
1 # r1
2 ip netns exec r1 iptables -t nat -A POSTROUTING -s 192.168.100.0/24 -j MASQUERADE
3 ip netns exec r1 iptables -t nat -A POSTROUTING -s 192.168.200.0/24 -j MASQUERADE
4
5 # r2
6 ip netns exec r2 iptables -t nat -A POSTROUTING -s 192.168.100.0/24 -j MASQUERADE
7 ip netns exec r2 iptables -t nat -A POSTROUTING -s 192.168.200.0/24 -j MASQUERADE
```

Now r1 can connect to the internet as shown in Figure 15. And we have the same result with r2.



The figure shows a terminal window with the command `sudo ip netns exec r1 ping 8.8.8.8` and its output, which is a successful ping to 8.8.8.8 with 5 packets, each 64 bytes, with various TTL and time values.

```
wings@April-Wings: ~/Infrastructre_Project
wings@April-Wings:~/Infrastructre_Project$ sudo ip netns exec r1 ping 8.8.8.8
PING 8.8.8.8 (8.8.8.8) 56(84) bytes of data.
64 bytes from 8.8.8.8: icmp_seq=1 ttl=118 time=16.2 ms
64 bytes from 8.8.8.8: icmp_seq=2 ttl=118 time=18.5 ms
64 bytes from 8.8.8.8: icmp_seq=3 ttl=118 time=16.4 ms
64 bytes from 8.8.8.8: icmp_seq=4 ttl=118 time=16.2 ms
64 bytes from 8.8.8.8: icmp_seq=5 ttl=118 time=16.2 ms
```

Figure 15: Access Internet from r1

Check the traffic from p1 to internet

In this section, we trace the route from p1 to Internet by using `traceroute` and get the result as Figure 16.

```
wings@April-Wings: ~/Infrastructre_Project
wings@April-Wings:~/Infrastructre_Project$ sudo ip netns exec p1 traceroute 8.8.8.8
traceroute to 8.8.8.8 (8.8.8.8), 30 hops max, 60 byte packets
 1 192.168.100.254 (192.168.100.254)  0.567 ms  0.498 ms  0.509 ms
 2 172.16.1.254 (172.16.1.254)  0.572 ms  0.464 ms  0.528 ms
 3 10.87.0.254 (10.87.0.254)  0.468 ms  0.413 ms  0.399 ms
 4 10.189.0.1 (10.189.0.1)  1.865 ms  1.829 ms  1.789 ms
 5 172.22.3.1 (172.22.3.1)  1.732 ms  1.682 ms  1.627 ms
 6 172.21.13.186 (172.21.13.186)  15.780 ms  15.090 ms  15.043 ms
 7 46.193.247.97 (46.193.247.97)  15.660 ms  20.196 ms  15.592 ms
 8 108.170.244.225 (108.170.244.225)  16.742 ms  108.170.245.1 (108.170.245.1)  16.710 ms  16.678 ms
 9 142.250.224.199 (142.250.224.199)  16.126 ms  142.251.64.129 (142.251.64.129)  15.376 ms  64.23 ms
10 8.8.8.8 (8.8.8.8)  15.288 ms  15.258 ms  15.225 ms
wings@April-Wings:~/Infrastructre_Project$
```

Figure 16: Access Internet from p1

We can observe that from p1, the traffic passes to r1 (interface `tunnel.100` with address 192.168.100.254) before going to the Internet.

Redirect the p1 and p2 towards r2

To redirect traffic from p1 and p2 to the Internet via r2 instead of r1 to optimize the access internet, we have set up a **DHCP Relay** on r2. The DHCP Relay will allows DHCP requests from the hosts on its side to be relayed to the DHCP server located on r1 while specifying the default route to use is an interface to itself.

We execute the following command on r2:

```
1 sudo ip netns exec r2 dnsmasq -d
   ↪ --dhcp-relay=192.168.100.253,192.168.100.254,tunnel.100
   ↪ --dhcp-relay=192.168.200.253,192.168.200.254,tunnel.200
   ↪ --dhcp-option=tunnel,3,192.168.100.253
```

Then, we run `dhclient` command line on p1 get the IP address. Figure 17 display the route of r1 to Internet in this configuration.

```
wings@April-Wings:~/Infrastructre_Project$ sudo ip netns exec 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 192.168.100.79
wings@April-Wings:~/Infrastructre_Project$ sudo ip netns exec p1 traceroute 8.8.8.8
traceroute to 8.8.8.8 (8.8.8.8), 30 hops max, 60 byte packets
 1 192.168.100.253 (192.168.100.253)  0.506 ms  0.391 ms  0.368 ms
 2 172.16.2.254 (172.16.2.254)  0.402 ms  0.437 ms  0.366 ms
 3 10.87.0.254 (10.87.0.254)  0.642 ms  0.621 ms  0.635 ms
 4 10.189.0.1 (10.189.0.1)  2.060 ms  2.053 ms  2.046 ms
 5 172.22.3.1 (172.22.3.1)  2.061 ms  2.054 ms  2.024 ms
 6 172.21.13.186 (172.21.13.186)  21.550 ms  21.242 ms  21.210 ms
 7 46.193.247.97 (46.193.247.97)  16.210 ms  15.810 ms  15.784 ms
 8 108.170.244.193 (108.170.244.193)  15.764 ms  15.682 ms  108.170.245.1 (108.170.245.1)  16.588 ms
 9 108.170.232.125 (108.170.232.125)  15.643 ms  64.233.174.175 (64.233.174.175)  15.296 ms  142.25 ms
10 8.8.8.8 (8.8.8.8)  15.247 ms  15.230 ms  15.214 ms
wings@April-Wings:~/Infrastructre_Project$
```

Figure 17: p1 access internet through r2

2.8 Prohibit traffic between VLAN100 and VLAN200

Using iptables rules

We configure the firewall using `iptables` rules as the following.

```

1 # r1
2 ip netns exec r1 iptables -t filter -A FORWARD -s 192.168.200.0/24 -d
  ↳ 192.168.100.0/24 -j DROP
3 ip netns exec r1 iptables -t filter -A FORWARD -s 192.168.100.0/24 -d
  ↳ 192.168.200.0/24 -j DROP
4 # r2
5 ip netns exec r2 iptables -t filter -A FORWARD -s 192.168.200.0/24 -d
  ↳ 192.168.100.0/24 -j DROP
6 ip netns exec r2 iptables -t filter -A FORWARD -s 192.168.100.0/24 -d
  ↳ 192.168.200.0/24 -j DROP

```

With these commands, all traffic has source IP addresses of VLAN100 and destination IP addresses of VLAN200 and vice versa are dropped.

Configuration file: *config-iptables-prohibit-traffic*.

Using "Policy Routing"

For the prohibition with Policy Routing, we added two new routing tables `vlan100` and `vlan200` into file `/etc/iproute2/route_tables` as Figure 18.



Figure 18: Add tables `vlan100` and `vlan200`

Now, we execute the following commands on `r1` and `r2`.

```

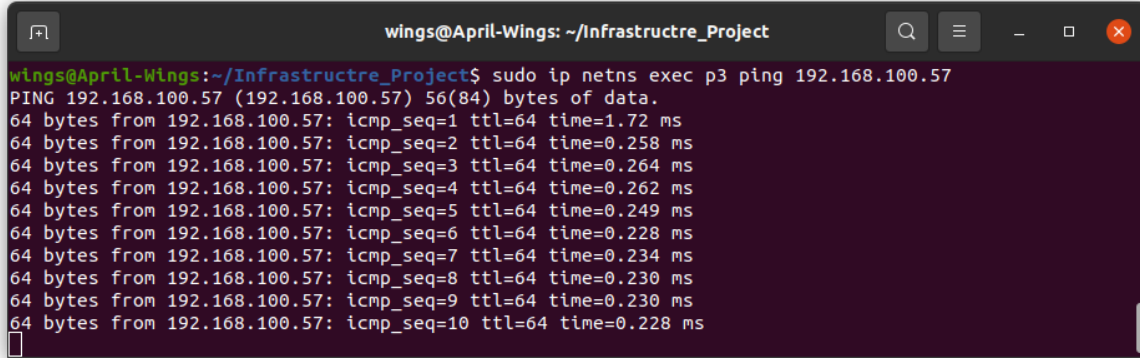
1 # r1
2 ip netns exec r1 ip rule add from 192.168.100.0/24 lookup vlan100
3 ip netns exec r1 ip rule add from 192.168.200.0/24 lookup vlan200
4 ip netns exec r1 ip route add prohibit 192.168.200.0/24 table vlan100
5 ip netns exec r1 ip route add prohibit 192.168.100.0/24 table vlan200
6 # r2
7 ip netns exec r2 ip rule add from 192.168.100.0/24 lookup vlan100
8 ip netns exec r2 ip rule add from 192.168.200.0/24 lookup vlan200
9 ip netns exec r2 ip route add prohibit 192.168.200.0/24 table vlan100
10 ip netns exec r2 ip route add prohibit 192.168.100.0/24 table vlan200

```

Configuration file: *config-policy-routing-prohibit-traffic*.

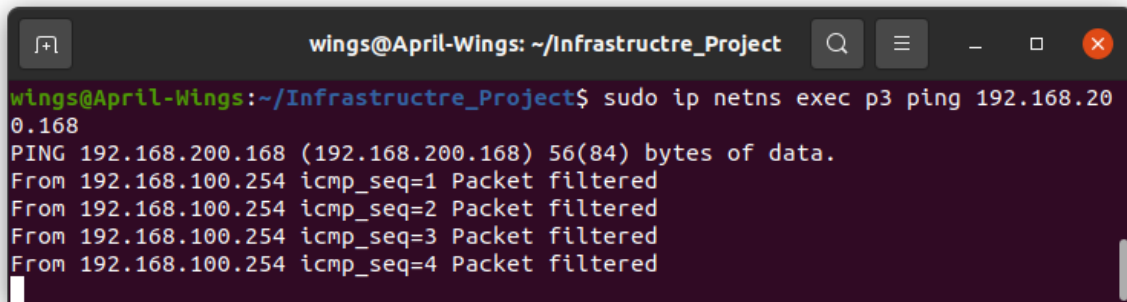
Results

To verify our configuration, we try to ping from p3 (VLAN100) to p1(VLAN100) and p4(VLAN200).



```
wings@April-Wings: ~/Infrastructure_Project
wings@April-Wings:~/Infrastructure_Project$ sudo ip netns exec p3 ping 192.168.100.57
PING 192.168.100.57 (192.168.100.57) 56(84) bytes of data.
64 bytes from 192.168.100.57: icmp_seq=1 ttl=64 time=1.72 ms
64 bytes from 192.168.100.57: icmp_seq=2 ttl=64 time=0.258 ms
64 bytes from 192.168.100.57: icmp_seq=3 ttl=64 time=0.264 ms
64 bytes from 192.168.100.57: icmp_seq=4 ttl=64 time=0.262 ms
64 bytes from 192.168.100.57: icmp_seq=5 ttl=64 time=0.249 ms
64 bytes from 192.168.100.57: icmp_seq=6 ttl=64 time=0.228 ms
64 bytes from 192.168.100.57: icmp_seq=7 ttl=64 time=0.234 ms
64 bytes from 192.168.100.57: icmp_seq=8 ttl=64 time=0.230 ms
64 bytes from 192.168.100.57: icmp_seq=9 ttl=64 time=0.230 ms
64 bytes from 192.168.100.57: icmp_seq=10 ttl=64 time=0.228 ms
```

Figure 19: Traffic from p3 to p1



```
wings@April-Wings: ~/Infrastructure_Project
wings@April-Wings:~/Infrastructure_Project$ sudo ip netns exec p3 ping 192.168.200.168
PING 192.168.200.168 (192.168.200.168) 56(84) bytes of data.
From 192.168.100.254 icmp_seq=1 Packet filtered
From 192.168.100.254 icmp_seq=2 Packet filtered
From 192.168.100.254 icmp_seq=3 Packet filtered
From 192.168.100.254 icmp_seq=4 Packet filtered
```

Figure 20: Traffic from p3 to p4

As we can see the result in Figure 20, we cannot ping from p3 to p4 after applying the above configurations. When using the target prohibit, we receive the message Packet Filtered. In other side, in Figure 19, we can see that the connections between the hosts inside VLAN100 are still normal.

3 Conclusion

In this project, we demonstrated the methods to set up an L2TPv3 tunnel secured by IPsec for the implementation of VLAN trunking in both IP and UDP encapsulation modes. Moreover, we also compared protocol L2TPv3 with VXLAN and GRE. And finally, we successfully build "intelligent" Internet access and use `iptables` rules, as well as Policy Routing to prohibit traffic between different VLANs.

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.pcwdd.com/vxlan>
3. IPsec
<https://en.wikipedia.org/wiki/IPsec>
4. MPLS
https://en.wikipedia.org/wiki/Multiprotocol_Label_Switching
<https://bit.ly/3byI1wp>