
NETWORK INFRASTRUCTURE PROJECT

**IPsec-secured L2TPv3 tunnel for
implementing VLAN trunking Comparison
with VXLANs**

Author

Viet-Huy HA

Anh-Kiet DUONG

Contents

1	Introduction	3
2	Network Architecture	3
3	Problem Solution	4
3.1	Sniffing network traffic	4
3.2	The L2TPv3 protocol and the VXLANs technology	5
3.2.1	L2TPv3 and VxLAN Comparison	5
3.2.2	Implementing VxLAN in Linux using Open vSwitch	6
3.2.3	L2TPv3/VXLAN traffic encryption	6
3.2.4	Comparing L2TPv3 and VxLAN with MPLS	7
3.3	L2TPv3 tunnel in IP encapsulation mode between R1 and R2	8
3.3.1	ARP Protocol	8
3.3.2	DHCP Service	8
3.3.3	TCP connection	9
3.4	L2TPv3 IP/UDP Tunnel vs GRE Tunnel Comparison	9
3.4.1	IP encapsulation mode	9
3.4.2	UDP encapsulation mode	10
3.4.3	GRE	11
3.5	IPsec on L2TPv3 tunnel	12
3.6	"Intelligent" Internet access	14
3.6.1	Access to the Internet	14
3.6.2	Access to the Internet on R1 and R2	15
3.6.3	P1 accesses Internet through R1	16
3.6.4	Redirect the P1 and P2 towards R2	16
3.7	Prohibit traffic between VLAN100 and VLAN200	17
3.7.1	Using "iptables" rules	17
3.7.2	Using "Policy Routing"	17
3.7.3	Result	18
	Appendices	18

1 Introduction

The project aims to study the L2TPv3 protocol, RFC 3931, to create level 2 tunnels. It will be used to "trunking" VLANs and share services such as DHCP. Finally, with an "intelligent" configuration, we will limit the use of the tunnel when hosts want to connect to the Internet by allowing them to do so directly from their "Internet side". In this project, we will apply theories and guidelines to solve the problems posed.

2 Network Architecture

Based on the original schematic of this project and inspired by other TP labs, we configured this project as follows:

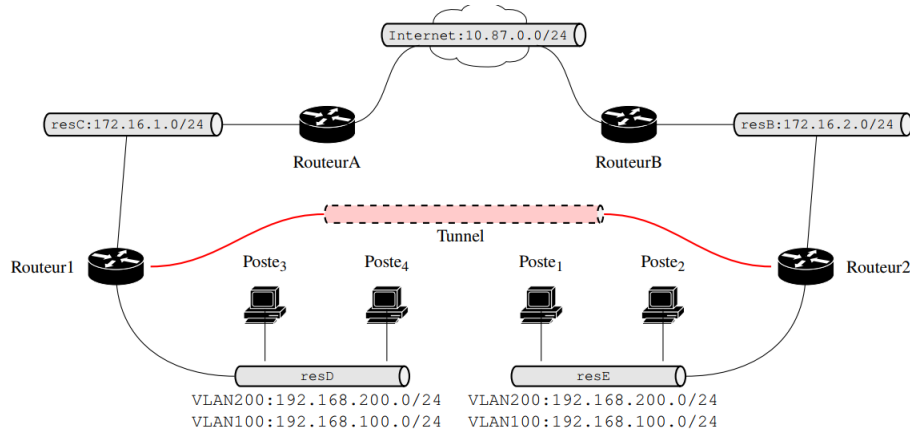


Figure 1: Original Architecture

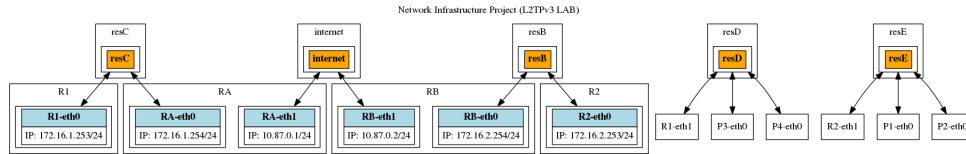


Figure 2: Result Architecture

Note: The initial interface, here are R1-eth1 and R2-eth1 do not have IP configuration.

DHCP Server

We used DHCP servers for assigning IP addresses automatically by using command:

```
1 ip netns exec R1 dnsmasq -d -z -i R1-eth1.100 --except-interface=lo -F
  192.168.100.0,192.168.100.100,255.255.255.0 &
2 ip netns exec R1 dnsmasq -d -z -i R1-eth1.200 --except-interface=lo -F
  192.168.200.0,192.168.200.100,255.255.255.0 &
3 ip netns exec P3 dhclient P3-eth0.100 &
4 ip netns exec P4 dhclient P4-eth0.200 &
```

We obtained IP configuration of host P3, P4 and interface R1-eth1:

```
2: P3-eth0.100@P3-eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue state UP group default qlen 1000
    link/ether 42:0a:1d:b1:01:1c brd ff:ff:ff:ff:ff:ff
    inet 192.168.100.58/24 brd 192.168.100.255 scope global P3-eth0.100
        valid_lft forever preferred_lft forever
    inet6 fe80::400a:1dff:feb1:11c/64 scope link
        valid_lft forever preferred_lft forever
30: P3-eth0@if29: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue state UP group default qlen 1000
    link/ether 42:0a:1d:b1:01:1c brd ff:ff:ff:ff:ff:ff link-netnsid 0
    inet6 fe80::400a:1dff:feb1:11c/64 scope link
        valid_lft forever preferred_lft forever
root@ubuntu:/home/ahn/Desktop/NetInfra/project# [P3]
```

Figure 3: P3 configuration

P3: MAC Address = 42:0a:1d:b1:01:1c, IP Address = 192.168.100.58

```
2: P4-eth0.200@P4-eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue state UP group default qlen 1000
    link/ether 46:04:8a:2d:77:7d brd ff:ff:ff:ff:ff:ff
    inet 192.168.200.80/24 brd 192.168.200.255 scope global P4-eth0.200
        valid_lft forever preferred_lft forever
    inet6 fe80::4404:8aff:fe2d:777d/64 scope link
        valid_lft forever preferred_lft forever
32: P4-eth0@if31: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue state UP group default qlen 1000
    link/ether 46:04:8a:2d:77:7d brd ff:ff:ff:ff:ff:ff link-netnsid 0
    inet6 fe80::4404:8aff:fe2d:777d/64 scope link
        valid_lft forever preferred_lft forever
root@ubuntu:/home/ahn/Desktop/NetInfra/project# [P4]
```

Figure 4: P4 configuration

P4: MAC Address = 46:04:8a:2d:77:7d, IP Address = 192.168.200.80

```
18: R1-eth0@if17: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue state UP group default qlen 1000
    link/ether 1e:85:0d:07:d4:28 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::1c85:dff:fe07:d428/64 scope link
        valid_lft forever preferred_lft forever
20: R1-eth1@if19: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue state UP group default qlen 1000
    link/ether ba:ef:9b:f8:89:80 brd ff:ff:ff:ff:ff:ff link-netnsid 0
    inet6 fe80::b8ef:9bff:fe8:8980/64 scope link
        valid_lft forever preferred_lft forever
root@ubuntu:/home/ahn/Desktop/NetInfra/project# [R1]
```

Figure 5: R1-eth1 configuration

R1-eth1: MAC Address = ba:ef:9b:f8:89:80

3 Problem Solution

3.1 Sniffing network traffic

In this section, we use configuration file named **"Q1_VLAN"**

For this problem, we used "tcpdump" to "sniffing" the network and check if VLAN encapsulation is working:

```
1 tcpdump -lnvv -i R1-eth1 -e vlan
```

After we use ping command from P3 to P4, we captured traffic on R1-eth1 interface below:

```

root@ubuntu: /home/ahn/Desktop/NetInfra/project
root@ubuntu: /home/ahn/Desktop/NetInfra/project 139x14
root@ubuntu: /home/ahn/Desktop/NetInfra/project# [P3] ping -c 5 192.168.200.80
PING 192.168.200.80 (192.168.200.80) 56(84) bytes of data.
64 bytes from 192.168.200.80: icmp_seq=1 ttl=63 time=0.941 ms
64 bytes from 192.168.200.80: icmp_seq=2 ttl=63 time=0.062 ms
64 bytes from 192.168.200.80: icmp_seq=3 ttl=63 time=0.063 ms
64 bytes from 192.168.200.80: icmp_seq=4 ttl=63 time=0.065 ms
64 bytes from 192.168.200.80: icmp_seq=5 ttl=63 time=0.090 ms

--- 192.168.200.80 ping statistics ---
5 packets transmitted, 5 received, 0% packet loss, time 410lms
rtt min/avg/max/mdev = 0.062/0.244/0.941/0.348 ms
root@ubuntu: /home/ahn/Desktop/NetInfra/project# [P3]

root@ubuntu: /home/ahn/Desktop/NetInfra/project 139x31
root@ubuntu: /home/ahn/Desktop/NetInfra/project# [R1] tcpdump -lnvv -i R1-eth1 -e vlan
tcpdump: listening on R1-eth1, link-type EN10MB (Ethernet), capture size 262144 bytes
06:45:51.783168 42:0a:1d:b1:01:1c > ba:ef:9b:f8:89:80, ethertype 802.1Q (0x8100), length 102: vlan 100, p 0, ethertype IPv4, (tos 0x0, ttl
192.168.100.58 > 192.168.200.80: ICMP echo request, id 11696, seq 1, length 64
06:45:51.783185 ba:ef:9b:f8:89:80 > 46:04:8a:2d:77:7d, ethertype 802.1Q (0x8100), length 102: vlan 200, p 0, ethertype IPv4, (tos 0x0, ttl
192.168.100.58 > 192.168.200.80: ICMP echo request, id 11696, seq 1, length 64
06:45:51.783485 46:04:8a:2d:77:7d > ba:ef:9b:f8:89:80, ethertype 802.1Q (0x8100), length 102: vlan 200, p 0, ethertype IPv4, (tos 0x0, ttl
192.168.200.80 > 192.168.100.58: ICMP echo reply, id 11696, seq 1, length 64
06:45:51.783491 ba:ef:9b:f8:89:80 > 42:0a:1d:b1:01:1c, ethertype 802.1Q (0x8100), length 102: vlan 100, p 0, ethertype IPv4, (tos 0x0, ttl
192.168.200.80 > 192.168.100.58: ICMP echo reply, id 11696, seq 1, length 64
06:45:52.812343 42:0a:1d:b1:01:1c > ba:ef:9b:f8:89:80, ethertype 802.1Q (0x8100), length 102: vlan 100, p 0, ethertype IPv4, (tos 0x0, ttl
192.168.100.58 > 192.168.200.80: ICMP echo request, id 11696, seq 2, length 64
06:45:52.812355 ba:ef:9b:f8:89:80 > 46:04:8a:2d:77:7d, ethertype 802.1Q (0x8100), length 102: vlan 200, p 0, ethertype IPv4, (tos 0x0, ttl
192.168.100.58 > 192.168.200.80: ICMP echo request, id 11696, seq 2, length 64
06:45:52.812370 46:04:8a:2d:77:7d > ba:ef:9b:f8:89:80, ethertype 802.1Q (0x8100), length 102: vlan 200, p 0, ethertype IPv4, (tos 0x0, ttl
192.168.200.80 > 192.168.100.58: ICMP echo reply, id 11696, seq 2, length 64
06:45:52.812372 ba:ef:9b:f8:89:80 > 42:0a:1d:b1:01:1c, ethertype 802.1Q (0x8100), length 102: vlan 100, p 0, ethertype IPv4, (tos 0x0, ttl
192.168.200.80 > 192.168.100.58: ICMP echo reply, id 11696, seq 2, length 64
06:45:53.836274 42:0a:1d:b1:01:1c > ba:ef:9b:f8:89:80, ethertype 802.1Q (0x8100), length 102: vlan 100, p 0, ethertype IPv4, (tos 0x0, ttl
192.168.100.58 > 192.168.200.80: ICMP echo request, id 11696, seq 3, length 64
06:45:53.836286 ba:ef:9b:f8:89:80 > 46:04:8a:2d:77:7d, ethertype 802.1Q (0x8100), length 102: vlan 200, p 0, ethertype IPv4, (tos 0x0, ttl
192.168.100.58 > 192.168.200.80: ICMP echo request, id 11696, seq 3, length 64
06:45:53.836300 46:04:8a:2d:77:7d > ba:ef:9b:f8:89:80, ethertype 802.1Q (0x8100), length 102: vlan 200, p 0, ethertype IPv4, (tos 0x0, ttl
192.168.200.80 > 192.168.100.58: ICMP echo reply, id 11696, seq 3, length 64
06:45:53.836302 ba:ef:9b:f8:89:80 > 42:0a:1d:b1:01:1c, ethertype 802.1Q (0x8100), length 102: vlan 100, p 0, ethertype IPv4, (tos 0x0, ttl
192.168.200.80 > 192.168.100.58: ICMP echo reply, id 11696, seq 3, length 64
06:45:54.860460 42:0a:1d:b1:01:1c > ba:ef:9b:f8:89:80, ethertype 802.1Q (0x8100), length 102: vlan 100, p 0, ethertype IPv4, (tos 0x0, ttl
192.168.100.58 > 192.168.200.80: ICMP echo request, id 11696, seq 4, length 64
06:45:54.860472 ba:ef:9b:f8:89:80 > 46:04:8a:2d:77:7d, ethertype 802.1Q (0x8100), length 102: vlan 200, p 0, ethertype IPv4, (tos 0x0, ttl
192.168.100.58 > 192.168.200.80: ICMP echo request, id 11696, seq 4, length 64
06:45:54.860486 46:04:8a:2d:77:7d > ba:ef:9b:f8:89:80, ethertype 802.1Q (0x8100), length 102: vlan 200, p 0, ethertype IPv4, (tos 0x0, ttl

```

Figure 6: Captured VLAN traffic

Analysis

- As we can see, the first packet is ARP ping between P3 and R1-eth1 interface, VLAN 100
- On next packets, we can see that the router interface R1-eth1 in turn generates an ARP to reach host P4 and transmits the ICMP packet, VLAN 200.

3.2 The L2TPv3 protocol and the VXLANs technology

3.2.1 L2TPv3 and VxLAN Comparison

L2TPv3 (Layer 2 Tunneling Protocol Version 3) and VxLAN (Virtual Extensible Local Area Network, RFC 7348) are both networking technologies designed to create virtual networks and extend Layer 2 connectivity across different locations.

L2TPv3

L2TPv3 is a Layer 2 tunneling protocol that encapsulates Layer 2 frames (such as Ethernet) for transport over IP networks. It is primarily used to interconnect geographically dispersed sites or different VLANs within a data center while maintaining the same Layer 2 domain. L2TPv3 supports 2 packet encapsulation formats: UDP and IP

VxLAN

VxLAN, on the other hand, is a Layer 2 over Layer 3 tunneling protocol that encapsulates Layer 2 Ethernet frames within Layer 3 UDP packets for transport over IP networks. It is specifically designed for data center network virtualization, allowing the creation of large-scale multi-tenant virtual networks and overcoming limitations of traditional VLANs.

- Increased scalability: VLANs are limited to 4096 unique identifiers (based on a 12-bit VLAN ID), which can be insufficient in large-scale, multi-tenant data center environments. VxLAN uses a 24-bit VxLAN Network Identifier (VNI), allowing for up to 16 million unique virtual networks. This significantly increases the number of possible virtual networks, enabling better support for large-scale deployments and multi-tenancy.
- Simplified Layer 2 network design: Traditional VLANs often require complex configurations of Spanning Tree Protocol (STP) to prevent loops and ensure efficient use of links in a Layer 2 network. VxLAN operates at Layer 2 over Layer 3, encapsulating Layer 2 Ethernet frames within Layer 3 UDP packets for transport over IP networks. This approach enables the use of Layer 3 routing and equal-cost multipath (ECMP) load balancing, which simplifies network design and improves network utilization by eliminating the need for STP and its associated complexities.

L2TPv3 is more suited for extending Layer 2 connectivity between sites in a WAN or interconnecting VLANs, while VxLAN is designed for large-scale data center network virtualization. VxLAN offers better scalability and multicast support compared to L2TPv3.

3.2.2 Implementing VxLAN in Linux using Open vSwitch

VxLAN can be implemented in Linux using Open vSwitch (OVS), an open-source software switch designed to work in virtualized environments. OVS supports VxLAN encapsulation, allowing for the creation and management of virtual networks across multiple physical hosts. To configure VxLAN using OVS, you would need to create and configure OVS bridges, add ports, and set the appropriate VxLAN tunnel parameters.

We created a bridge, added port to it and added vxlan1 tunnel on the bridge, using following commands

```
1 ovs - vsctl add -br br0
2 ovs - vsctl add - port br0 vxlan1 -- set interface vxlan1 type = vxlan
   options : remote_ip = 172.16.3.233 options :key= flow options : dst_port =
   8472
```

3.2.3 L2TPv3/VXLAN traffic encryption

L2TPv3 doesn't provide built-in encryption. However, It can be combined with IPsec to encrypt the tunnel traffic. This provides data confidentiality, integrity, and authentication between the tunnel endpoints.

VxLAN doesn't offer native encryption either. However, It can be secured using different approaches, such as MACsec (for point-to-point links) or by leveraging encryption solutions provided by the underlying physical network, such as IPsec or SSL/TLS VPNs.

3.2.4 Comparing L2TPv3 and VxLAN with MPLS

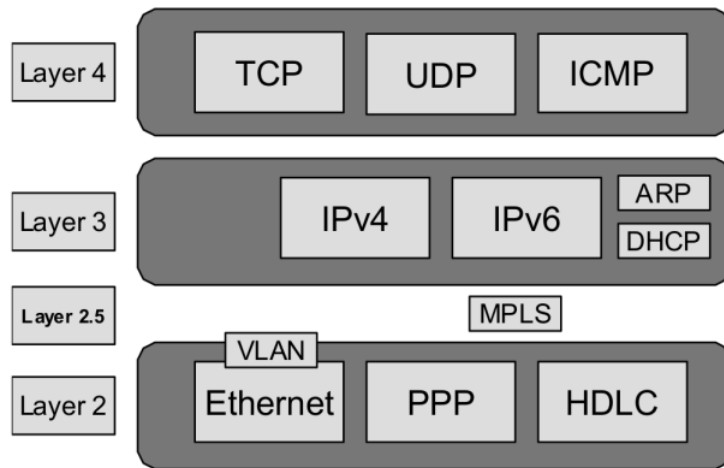


Figure 7: Multiprotocol Label Switching

MPLS (Multiprotocol Label Switching) is a protocol used for high-performance traffic forwarding and network virtualization. It differs from L2TPv3 and VxLAN in that it operates at the Layer 2.5, allowing for more granular control over traffic engineering and Quality of Service (QoS). MPLS can be used to create Layer 2 (VPLS) or Layer 3 (MPLS-VPN) virtual networks.

L2TPv3 and VxLAN rely on IP networks for transport, while MPLS uses label switching for forwarding packets. MPLS provides better traffic engineering capabilities and QoS, but it may require specialized hardware and can be more complex to deploy and manage. On the other hand, L2TPv3 and VxLAN can be deployed using standard IP networks and offer simpler setup and management, particularly in data center environments.

3.3 L2TPv3 tunnel in IP encapsulation mode between R1 and R2

3.3.1 ARP Protocol

In this section, we use configuration file named **Q2_IPTunnel**.

To verify that ARP protocol works normally, we sniff our network by using tcpdump on the tunnel interface on R1 and use a "ping" command sent from P1 to P4.

```
root@ubuntu: /home/ahn/Desktop/NetInfra/project
root@ubuntu: /home/ahn/Desktop/NetInfra/project 86x10
link/ether 4e:09:82:9a:02:73 brd ff:ff:ff:ff:ff:ff
inet 192.168.100.176/24 brd 192.168.100.255 scope global P1-eth0.100
    valid lft forever preferred lft forever
inet6 fe80::4c09:82ff:fe9a:273/64 scope link
    valid lft forever preferred lft forever
26: P1-eth0@if25: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue state UP gr
link/ether 4e:09:82:9a:02:73 brd ff:ff:ff:ff:ff:ff link-netnsid 0
inet6 fe80::4c09:82ff:fe9a:273/64 scope link
    valid lft forever preferred lft forever
root@ubuntu: /home/ahn/Desktop/NetInfra/project# [P1]

root@ubuntu: /home/ahn/Desktop/NetInfra/project 174x10
root@ubuntu: /home/ahn/Desktop/NetInfra/project# [P1] ping -c 2 192.168.200.153
PING 192.168.200.153 (192.168.200.153) 56(84) bytes of data:
64 bytes from 192.168.200.153: icmp_seq=1 ttl=63 time=1.47 ms
64 bytes from 192.168.200.153: icmp_seq=2 ttl=63 time=0.272 ms

--- 192.168.200.153 ping statistics ---
2 packets transmitted, 2 received, 0% packet loss, time 1002ms
rtt min/avg/max/mdev = 0.272/0.871/1.470/0.599 ms
root@ubuntu: /home/ahn/Desktop/NetInfra/project# [P1]

root@ubuntu: /home/ahn/Desktop/NetInfra/project 174x19
root@ubuntu: /home/ahn/Desktop/NetInfra/project# [R1] tcpdump -lnvv -i tunnel -e vlan
tcpdump: listening on tunnel, link-type EN10MB (Ethernet), capture size 262144 bytes
13:17:58.139360 4e:09:82:9a:02:73 > 72:c:f:ad:70:f9:41, ethertype 802.1Q (0x8100), length 102: vlan 100, p 0, ethertype IPv4, (tos 0x0, ttl 64, id 2709, offset 0, flags [DF],
192.168.100.176 > 192.168.200.153: ICMP echo request, id 3321, seq 1, length 64
13:17:58.139378 72:c:f:ad:70:f9:41 > 6e:83:93:f1:91:f9, ethertype 802.1Q (0x8100), length 102: vlan 200, p 0, ethertype IPv4, (tos 0x0, ttl 63, id 2709, offset 0, flags [DF],
192.168.100.176 > 192.168.200.153: ICMP echo request, id 3321, seq 1, length 64
13:17:58.139610 6e:83:93:f1:91:f9 > 72:c:f:ad:70:f9:41, ethertype 802.1Q (0x8100), length 102: vlan 200, p 0, ethertype IPv4, (tos 0x0, ttl 64, id 12261, offset 0, flags [none],
192.168.200.153 > 192.168.100.176: ICMP echo reply, id 3321, seq 1, length 64
13:17:58.139619 72:c:f:ad:70:f9:41 > 4e:09:82:9a:02:73, ethertype 802.1Q (0x8100), length 102: vlan 100, p 0, ethertype IPv4, (tos 0x0, ttl 63, id 12261, offset 0, flags [none],
192.168.200.153 > 192.168.100.176: ICMP echo reply, id 3321, seq 1, length 64
13:17:59.141383 4e:09:82:9a:02:73 > 72:c:f:ad:70:f9:41, ethertype 802.1Q (0x8100), length 102: vlan 100, p 0, ethertype IPv4, (tos 0x0, ttl 64, id 2880, offset 0, flags [DF],
192.168.100.176 > 192.168.200.153: ICMP echo request, id 3321, seq 2, length 64
13:17:59.141398 72:c:f:ad:70:f9:41 > 6e:83:93:f1:91:f9, ethertype 802.1Q (0x8100), length 102: vlan 200, p 0, ethertype IPv4, (tos 0x0, ttl 63, id 2880, offset 0, flags [DF],
192.168.100.176 > 192.168.200.153: ICMP echo request, id 3321, seq 2, length 64
13:17:59.141444 6e:83:93:f1:91:f9 > 72:c:f:ad:70:f9:41, ethertype 802.1Q (0x8100), length 102: vlan 200, p 0, ethertype IPv4, (tos 0x0, ttl 64, id 12411, offset 0, flags [none],
192.168.200.153 > 192.168.100.176: ICMP echo reply, id 3321, seq 2, length 64
13:17:59.141454 72:c:f:ad:70:f9:41 > 4e:09:82:9a:02:73, ethertype 802.1Q (0x8100), length 102: vlan 100, p 0, ethertype IPv4, (tos 0x0, ttl 63, id 12411, offset 0, flags [none],
192.168.200.153 > 192.168.100.176: ICMP echo reply, id 3321, seq 2, length 64
```

Figure 8: Verify ARP protocol

3.3.2 DHCP Service

Similar to above section, we used DHCP servers for assigning IP addresses automatically by using command:

```
1 sudo ip netns exec R1 dnsmasq -d -z -i tunnel.100 --except-interface=lo -F
   192.168.100.0,192.168.100.180,255.255.255.0 &
2 sudo ip netns exec R1 dnsmasq -d -z -i tunnel.200 --except-interface=lo -F
   192.168.200.0,192.168.200.180,255.255.255.0 &
3 sudo ip netns exec P1 dhclient P1-eth0.100 &
4 sudo ip netns exec P3 dhclient P3-eth0.100 &
5 sudo ip netns exec P4 dhclient P4-eth0.200 &
6 sudo ip netns exec P2 dhclient P2-eth0.200 &
```


Then we got following result:

```
dnsmasq-dhcp: DHCPDISCOVER(tunnel.200) 2a:da:53:36:2b:14
dnsmasq-dhcp: DHCPOFFER(tunnel.200) 192.168.200.61 2a:da:53:36:2b:14
dnsmasq-dhcp: DHCPREQUEST(tunnel.200) 192.168.200.61 2a:da:53:36:2b:14
dnsmasq-dhcp: DHCPACK(tunnel.200) 192.168.200.61 2a:da:53:36:2b:14 ubuntu
dnsmasq-dhcp: not giving name ubuntu to the DHCP lease of 192.168.200.61 because the name exists in /etc/hosts with address 127.0.1.1
dnsmasq-dhcp: DHCPDISCOVER(tunnel.100) 4e:09:82:9a:02:73
dnsmasq-dhcp: DHCPOFFER(tunnel.100) 192.168.100.176 4e:09:82:9a:02:73
dnsmasq-dhcp: DHCPDISCOVER(tunnel.100) 192.168.100.17 7a:ab:92:b2:fd:da
dnsmasq-dhcp: DHCPOFFER(tunnel.100) 192.168.100.17 7a:ab:92:b2:fd:da
dnsmasq-dhcp: DHCPREQUEST(tunnel.100) 192.168.100.17 7a:ab:92:b2:fd:da
dnsmasq-dhcp: DHCPACK(tunnel.100) 192.168.100.17 7a:ab:92:b2:fd:da ubuntu
dnsmasq-dhcp: not giving name ubuntu to the DHCP lease of 192.168.100.17 because the name exists in /etc/hosts with address 127.0.1.1
dnsmasq-dhcp: DHCPREQUEST(tunnel.100) 192.168.100.176 4e:09:82:9a:02:73
dnsmasq-dhcp: DHCPACK(tunnel.100) 192.168.100.176 4e:09:82:9a:02:73 ubuntu
dnsmasq-dhcp: not giving name ubuntu to the DHCP lease of 192.168.100.176 because the name exists in /etc/hosts with address 127.0.1.1
dnsmasq-dhcp: DHCPDISCOVER(tunnel.200) 192.168.200.174 6e:83:93:f1:91:f9
dnsmasq-dhcp: DHCPOFFER(tunnel.200) 192.168.200.153 6e:83:93:f1:91:f9
dnsmasq-dhcp: DHCPDISCOVER(tunnel.200) 192.168.200.174 6e:83:93:f1:91:f9
dnsmasq-dhcp: DHCPOFFER(tunnel.200) 192.168.200.153 6e:83:93:f1:91:f9
dnsmasq-dhcp: DHCPREQUEST(tunnel.200) 192.168.200.153 6e:83:93:f1:91:f9
dnsmasq-dhcp: DHCPACK(tunnel.200) 192.168.200.153 6e:83:93:f1:91:f9 ubuntu
dnsmasq-dhcp: not giving name ubuntu to the DHCP lease of 192.168.200.153 because the name exists in /etc/hosts with address 127.0.1.1
```

Figure 9: DHCP Server Result

3.3.3 TCP connection

We observed TCP communication between R1 and P1 through tunnel:

```
root@ubuntu: /home/ahn/Desktop/NetInfra/project 174x10
root@ubuntu: /home/ahn/Desktop/NetInfra/project# [P1] socat - tcp:192.168.100.254:2023
Good Evening!
Hi
How are you?
I am so tired!
Nice try!
Hacked ==))

root@ubuntu: /home/ahn/Desktop/NetInfra/project 174x19
root@ubuntu: /home/ahn/Desktop/NetInfra/project# [R1] socat tcp-listen:2023 -
Good Evening!
Hi
How are you?
I am so tired!
Nice try!
Hacked ==))
```

Figure 10: TCP connection

3.4 L2TPv3 IP/UDP Tunnel vs GRE Tunnel Comparison

In this part, we'll use Wireshark for sniffing packets, then analysis to find out difference between encapsulation modes. We capture packets exchange between host P2 and P3.

3.4.1 IP encapsulation mode

Configuration file: Q2_IPTunnel

Captured file: Q4_IP_Communication.pcap.pcapng

To observe the L2TPv3 tunneling in IP mode, we sniff traffic on router RA.

As we can see, L2TPv3 in IP encapsulation mode is displayed by protocol identifier **0x0000** and marked **Unknown** on "tcpdump" console.

No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000000	172.16.1.253	172.16.2.253	0x0000	112	D[S:0x7D0]
3	0.000038823	172.16.1.253	172.16.2.253	0x0000	112	D[S:0x7D0]
4	16.384459522	172.16.1.253	172.16.2.253	0x0000	116	D[S:0x7D0]
5	16.384490620	172.16.1.253	172.16.2.253	0x0000	116	D[S:0x7D0]
6	16.384541914	172.16.2.253	172.16.1.253	0x0000	112	D[S:0x3E8]
7	16.384548350	172.16.2.253	172.16.1.253	0x0000	112	D[S:0x3E8]
10	82.588299229	172.16.2.253	172.16.1.253	0x0000	144	D[S:0x3E8]
11	82.588493567	172.16.1.253	172.16.2.253	0x0000	144	D[S:0x7D0]
12	83.616391907	172.16.2.253	172.16.1.253	0x0000	144	D[S:0x3E8]
13	83.616427368	172.16.1.253	172.16.2.253	0x0000	144	D[S:0x7D0]
14	87.808338464	172.16.1.253	172.16.2.253	0x0000	88	D[S:0x7D0]
17	87.808616865	172.16.2.253	172.16.1.253	0x0000	88	D[S:0x3E8]
8	21.503857871	ae:c6:7f:0f:51:2d	fa:e7:db:f9:8e:6d	ARP	42	Who has 172.16.1.253? Tell 172.16.1.254
9	21.504134959	fa:e7:db:f9:8e:6d	ae:c6:7f:0f:51:2d	ARP	42	172.16.1.253 is at fa:e7:db:f9:8e:6d

▶ Frame 1: 112 bytes on wire (896 bits), 112 bytes captured (896 bits) on interface 0
 ▶ Ethernet II, Src: fa:e7:db:f9:8e:6d (fa:e7:db:f9:8e:6d), Dst: ae:c6:7f:0f:51:2d (ae:c6:7f:0f:51:2d)
 ▶ Internet Protocol Version 4, Src: 172.16.1.253, Dst: 172.16.2.253
 ▶ Layer 2 Tunneling Protocol version 3
 Packet Type: Data Message Session Id=2000
 Session ID: 2000
 Cisco HDLC

Offset	Hex	ASCII
0000	ae c6 7f 0f 51 2d fa e7 db f9 8e 6d 08 00 45 00	...Q...m..E..
0010	00 62 da 51 00 00 40 73 42 bd ac 10 01 fd ac 10	..b.Q..@s B.....
0020	02 fd 00 00 07 d0 00 00 00 00 33 33 00 00 00 0233.....
0030	1a 57 6c ca b4 a6 8e dd 60 00 00 00 00 19 3a ff	..WL.....
0040	fe 80 00 00 00 00 00 00 18 57 6c ff fe ca b4 a6WL.....
0050	ff 02 00 00 00 00 00 00 00 00 00 00 00 00 02
0060	85 00 07 9e 00 00 00 00 01 01 1a 57 6c ca b4 a6WL...

Figure 11: IP encapsulation mode

3.4.2 UDP encapsulation mode

Configuration file: Q4.UDPTunnel

Captured file: Q4_UDP_communication.pcapng

Next, we will monitor traffic of the L2TPv3 tunneling in UDP mode by implementing on port 2022 and 2023.

Through this section, we can clearly see the difference between IP encapsulation mode and UDP encapsulation mode

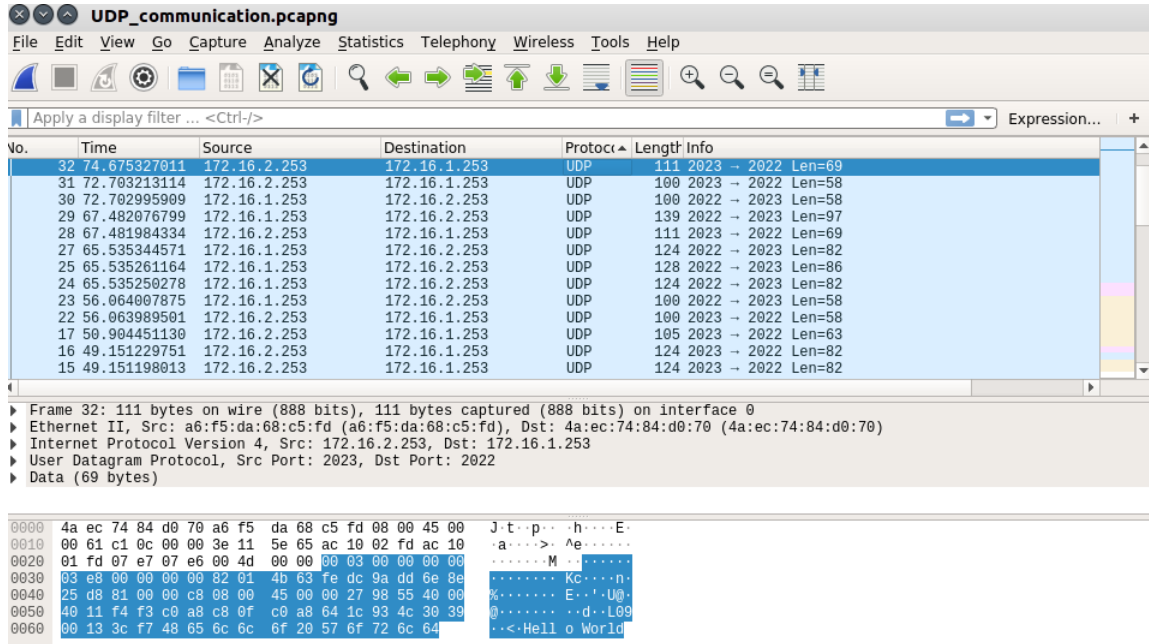


Figure 12: UDP encapsulation mode

3.4.3 GRE

Configuration file: Q4.GRETunnel

Captured file: Q4_GRE.communication.pcapng

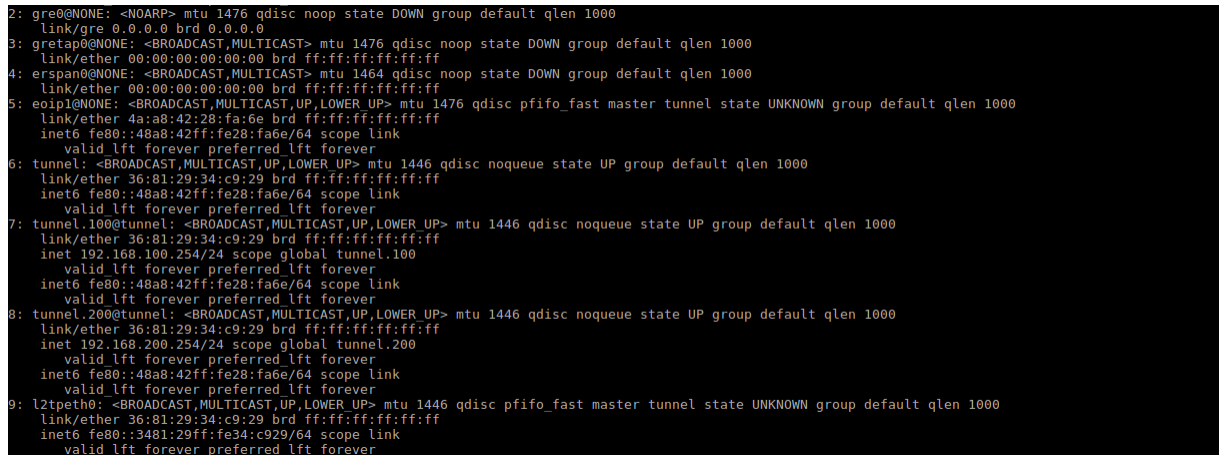


Figure 13: MTU

GRE (Generic Routing Encapsulation) is a tunneling protocol that enables encapsulation of a wide variety of network layer protocols inside point-to-point links. It was developed by Cisco Systems and described in RFC 2784. GRE is commonly used to establish secure and private communication channels between networks over the public internet or other untrusted networks.

Figure 13 shows us the difference between GRE MTU and L2TPv3 MTU. GRE has an MTU length of 1476, while L2TPv3 is only 1446. And we know that GRE header has 24 bytes and L2TPv3 header has 40 bytes.

After configuring GRE in GRE-TAP mode, we capture following result:

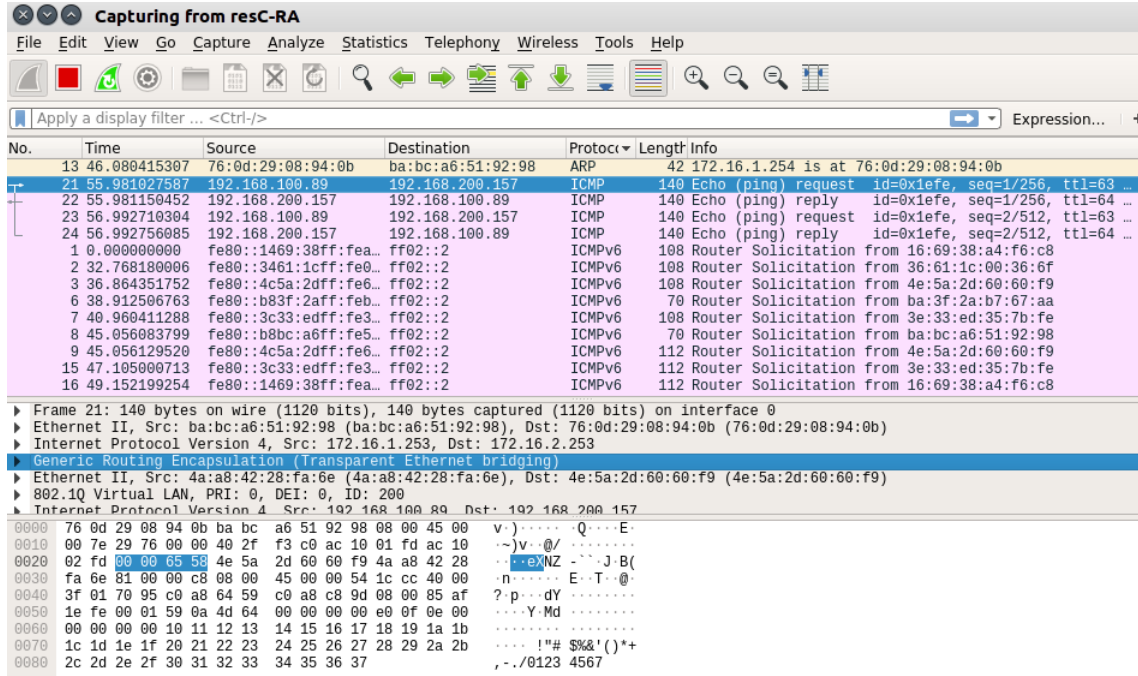


Figure 14: GRE-TAP

3.5 IPsec on L2TPv3 tunnel

We set up IPsec encryption on our L2TPv3 tunnel by IP encapsulation base on TP files (Configuration can be found in file: **Q5_IPSec**). We will use following command for each router R1 and R2:

```
1 ip xfrm state flush
2 ip xfrm policy flush
3 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
   x323730ed6f1b9ff0cb084af15b197e862b7c18424a7cdfb74cd385ae23bc4f17 enc "
   rfc3686(ctr(aes))" 0x27b90b8aecd1ee32a8150a664e8faac761e2d305b
4 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
   x44d65c50b7581fd3c8169cf1fa0ebb24e0d55755b1dc43a98b539bb144f2067f enc "
   rfc3686(ctr(aes))" 0x9df7983cb7c7eb2af01d88d36e462b5f01d10bc1
5 ip xfrm policy add src 172.16.1.253 dst 172.16.2.253 dir in tmpl src
   172.16.1.253 dst 172.16.2.253 proto esp reqid 0x12345678 mode transport
6 ip xfrm policy add src 172.16.2.253 dst 172.16.1.253 dir out tmpl src
   172.16.2.253 dst 172.16.1.253 proto esp reqid 0x12345678 mode transport
```

Next, we capture packets pass through through tunnel to observe the effect of IPsec. Continue to use Wireshark, we got:

The image shows a Wireshark capture of an IPsec packet. The packet list pane shows several packets, with packet 8 selected. The packet details pane shows the structure of the packet: Ethernet II, Internet Protocol Version 4, and Encapsulating Security Payload (ESP). The packet bytes pane shows the raw data of the packet, including the ESP header and payload.

No.	Time	Source	Destination	Protocol	Length	Info
8	6.143730673	172.16.1.253	172.16.2.253	ESP	146	ESP (SPI=0x12345678)
10	9.796661974	172.16.1.253	172.16.2.253	ESP	278	ESP (SPI=0x12345678)
11	9.860859873	172.16.2.253	172.16.1.253	ESP	278	ESP (SPI=0x12345678)
13	12.288049610	172.16.1.253	172.16.2.253	ESP	142	ESP (SPI=0x12345678)
14	14.335034530	172.16.2.253	172.16.1.253	ESP	146	ESP (SPI=0x12345678)
15	16.383920303	172.16.1.253	172.16.2.253	ESP	146	ESP (SPI=0x12345678)

Frame 8: 146 bytes on wire (1168 bits), 146 bytes captured (1168 bits) on interface 0
 Ethernet II, Src: aa:0d:e2:e7:b6:7d (aa:0d:e2:e7:b6:7d), Dst: 52:97:24:df:9b:29 (52:97:24:df:9b:29)
 Internet Protocol Version 4, Src: 172.16.1.253, Dst: 172.16.2.253
 Encapsulating Security Payload

0000 52 97 24 df 9b 29 aa 0d e2 e7 b6 7d 08 00 45 00 R \$ } . . E .
 0010 00 84 c3 45 00 00 40 32 59 e8 ac 10 01 fd ac 10 . . . E . . @ 2 Y
 0020 02 fd 12 34 56 78 00 00 00 0b 61 15 79 7d e6 50 . . . 4Vx . . . a . y } . P
 0030 20 ca d6 d3 9d f7 38 83 ec c4 37 89 a5 42 07 ab 8 . . . 7 . B .
 0040 8e e1 68 fa f8 ab 6b 2c 6a 3f 3c 50 4e 97 45 c2 . . h . . . k , j ? < P N . E .
 0050 57 bc c0 07 93 1a 71 48 17 ad b5 dd fd c4 ad ee W q H
 0060 88 f8 55 f3 58 63 d9 ce 4f 9d e7 67 53 6e 6d cf . . U . Xc . . . 0 . . g S n m .
 0070 71 06 6b be cc ce 69 ea 7d ce ca 9a c5 48 ef 39 q . k . . . i . } H . 9
 0080 f0 9d 96 25 d0 e3 67 fb 2b de f7 01 49 9f 75 9f . . % . . g . + . . . I . u .
 0090 ba 84

Figure 15: IPsec packet

As shown, there is only a header named Encapsulating Security Payload (ESP) and the protocol of packet is also ESP. There is no information about the IP header nor the data field.

Finally, we will look at transfer speed between router R1 and host P1, then make a comparison about "Transferring without IPsec" and "Transferring with IPsec". The following table is the result obtained when using the **iperf** command:

```

root@ubuntu:/home/ahn/Desktop/NetInfra/project

root@ubuntu:/home/ahn/Desktop/NetInfra/project# [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 192.168.100.94 port 35484
[ ID] Interval      Transfer      Bandwidth
[ 4] 0.0-10.0 sec  1.96 GBytes  1.68 Gbits/sec

root@ubuntu:/home/ahn/Desktop/NetInfra/project# [P1] iperf -c 192.168.100.254
Client connecting to 192.168.100.254, TCP port 5001
TCP window size: 85.0 KByte (default)
-----
[ 3] local 192.168.100.94 port 35484 connected with 192.168.100.254 port 5001
[ ID] Interval      Transfer      Bandwidth
[ 3] 0.0-10.0 sec  1.96 GBytes  1.68 Gbits/sec
root@ubuntu:/home/ahn/Desktop/NetInfra/project# [P1]

```

Figure 16: Transferring without IPsec

```
root@ubuntu: /home/ahn/Desktop/NetInfra/project
root@ubuntu:/home/ahn/Desktop/NetInfra/project# [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 192.168.100.94 port 40838
[ ID] Interval      Transfer    Bandwidth
[ 4] 0.0-10.0 sec  596 MBytes  498 Mbits/sec

root@ubuntu:/home/ahn/Desktop/NetInfra/project# [P1] iperf -c 192.168.100.254
-----
Client connecting to 192.168.100.254, TCP port 5001
TCP window size: 85.0 KByte (default)
-----
[ 3] local 192.168.100.94 port 40838 connected with 192.168.100.254 port 5001
[ ID] Interval      Transfer    Bandwidth
[ 3] 0.0-10.0 sec  596 MBytes  500 Mbits/sec
root@ubuntu:/home/ahn/Desktop/NetInfra/project# [P1]
```

Figure 17: Transferring with IPsec

A clear victory in speed was demonstrated for an interval of 10s when **iperf** has transfer 1.96 GBytes and 1.68 GBits/sec Bandwidth without IPsec. Meanwhile, there is only 596 MBytes and 500 Mbits/sec Bandwidth for transfer with IPsec. This causes of IPsec encapsulated the IP packets. As a result, the IP packet became longer and lead to fragmentation. Therefore, The receiver needs to reassemble packets.

3.6 "Intelligent" Internet access

Configuration file: Q6.InternetAccess

3.6.1 Access to the Internet

To config for Internet Access, we will use following command:

```
1 sysctl net.ipv4.conf.all.forwarding=1
2 # internet
3 ip a add dev internet 10.87.0.254/24
4 iptables -t nat -A POSTROUTING -s 10.87.0.0/24 -j MASQUERADE
5 # RA
6 ip netns exec RA ip route add default via 10.87.0.254
7 ip netns exec RA iptables -t nat -A POSTROUTING -s 172.16.1.0/24 -j
  MASQUERADE
8 # RB
9 ip netns exec RB ip route add default via 10.87.0.254
10 ip netns exec RB iptables -t nat -A POSTROUTING -s 172.16.2.0/24 -j
  MASQUERADE
```

Then we will check connection by using ping command to Google DNS server, which has IPv4 address 8.8.8.8 or 8.8.8.4.

```
root@ubuntu: /home/ahn/Desktop/NetInfra/project

root@ubuntu:/home/ahn/Desktop/NetInfra/project# [RB] ping -c 3 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=127 time=15.1 ms
64 bytes from 8.8.8.8: icmp_seq=2 ttl=127 time=14.5 ms
64 bytes from 8.8.8.8: icmp_seq=3 ttl=127 time=14.6 ms

--- 8.8.8.8 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2004ms
rtt min/avg/max/mdev = 14.585/14.777/15.132/0.251 ms
root@ubuntu:/home/ahn/Desktop/NetInfra/project# [RB] █

root@ubuntu:/home/ahn/Desktop/NetInfra/project# [RA] ping -c 3 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=127 time=14.1 ms
64 bytes from 8.8.8.8: icmp_seq=2 ttl=127 time=13.9 ms
64 bytes from 8.8.8.8: icmp_seq=3 ttl=127 time=13.4 ms

--- 8.8.8.8 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2003ms
rtt min/avg/max/mdev = 13.478/13.864/14.191/0.294 ms
root@ubuntu:/home/ahn/Desktop/NetInfra/project# [RA] █
```

Figure 18: RA and RA access Internet

3.6.2 Access to the Internet on R1 and R2

On routers R1 and R2, for the hosts of the two VLANs can access to the Internet, we use following iptables rules:

```
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 # R2
5 ip netns exec R2 iptables -t nat -A POSTROUTING -s 192.168.100.0/24 -j
  MASQUERADE
6 ip netns exec R2 iptables -t nat -A POSTROUTING -s 192.168.200.0/24 -j
  MASQUERADE
```

Then we get the results:


```
root@ubuntu:/home/ahn/Desktop/NetInfra/project
root@ubuntu:/home/ahn/Desktop/NetInfra/project# [R1] ping -c 3 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=126 time=14.8 ms
64 bytes from 8.8.8.8: icmp_seq=2 ttl=126 time=14.3 ms
64 bytes from 8.8.8.8: icmp_seq=3 ttl=126 time=13.8 ms

--- 8.8.8.8 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2003ms
rtt min/avg/max/mdev = 13.855/14.351/14.876/0.439 ms
root@ubuntu:/home/ahn/Desktop/NetInfra/project# [R1]

root@ubuntu:/home/ahn/Desktop/NetInfra/project# ./build_bash R2
root@ubuntu:/home/ahn/Desktop/NetInfra/project# [R2] ping -c 3 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=126 time=22.5 ms
64 bytes from 8.8.8.8: icmp_seq=2 ttl=126 time=13.9 ms
64 bytes from 8.8.8.8: icmp_seq=3 ttl=126 time=14.7 ms

--- 8.8.8.8 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2003ms
rtt min/avg/max/mdev = 13.935/17.066/22.525/3.875 ms
root@ubuntu:/home/ahn/Desktop/NetInfra/project# [R2]
```

Figure 19: RA and RA access Internet

3.6.3 P1 accesses Internet through R1

To trace path packet taken from host P1 accesses to the Internet, we will use **tracert** command. Then we can see from P1, packet jumps to next hop R1 (192.168.100.254)

```
root@ubuntu:/home/ahn/Desktop/NetInfra/project# [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) 2.091 ms 2.082 ms 2.078 ms
 2 172.16.1.254 (172.16.1.254) 2.077 ms 2.076 ms 2.073 ms
 3 10.87.0.254 (10.87.0.254) 2.044 ms 2.041 ms 2.040 ms
 4 192.168.81.2 (192.168.81.2) 2.040 ms 2.039 ms 2.040 ms
```

Figure 20: Traffic from P1 goes through R1

3.6.4 Redirect the P1 and P2 towards R2

We will set up a DHCP Relay on R2 for redirecting traffic from P1 and P2 to R2 instead of R1. The DHCP Relay will allow 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 use following command:

```
1 sudo ip netns exec R2 dnsmasq -d --dhcp-relay
    =192.168.100.253,192.168.100.254,tunnel.100 --dhcp-relay
    =192.168.100.253,192.168.200.254,tunnel.200 --dhcp-option=tunnel
    ,3,192.168.100.253
```


3.7 Prohibit traffic between VLAN100 and VLAN200

3.7.1 Using "iptables" rules

Configuration file: Q7_ProhibitTraffic_iptables We will use following command:

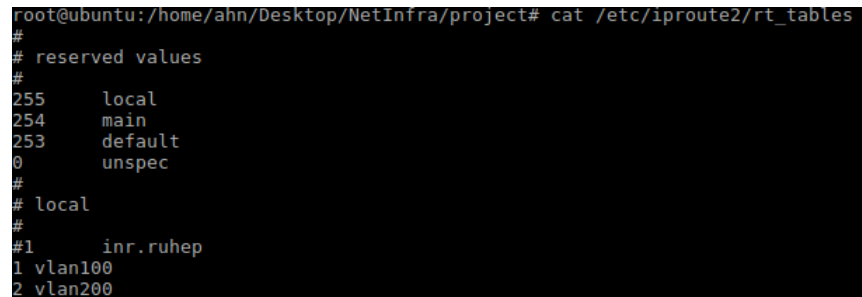
```
1 # R1
2 ip netns exec R1 iptables -t filter -A FORWARD -s 192.168.200.0/24 -d
   192.168.100.0/24 -j DROP
3 ip netns exec R1 iptables -t filter -A FORWARD -s 192.168.100.0/24 -d
   192.168.200.0/24 -j DROP
4 # R2
5 ip netns exec R2 iptables -t filter -A FORWARD -s 192.168.200.0/24 -d
   192.168.100.0/24 -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 with the source IP address of VLAN100 and the destination IP address of VLAN200 and vice versa is dropped.

3.7.2 Using "Policy Routing"

Configuration file: Q7_ProhibitTraffic_policyrouting

For the prohibition with Policy routing, we added two new routing tables vlan100 and vlan200 into file /etc/iproute2/route_tables



```
root@ubuntu:/home/ahn/Desktop/Netinfra/project# cat /etc/iproute2/route_tables
#
# reserved values
#
255    local
254    main
253    default
0      unspec
#
# local
#
#1     inr.ruhep
1     vlan100
2     vlan200
```

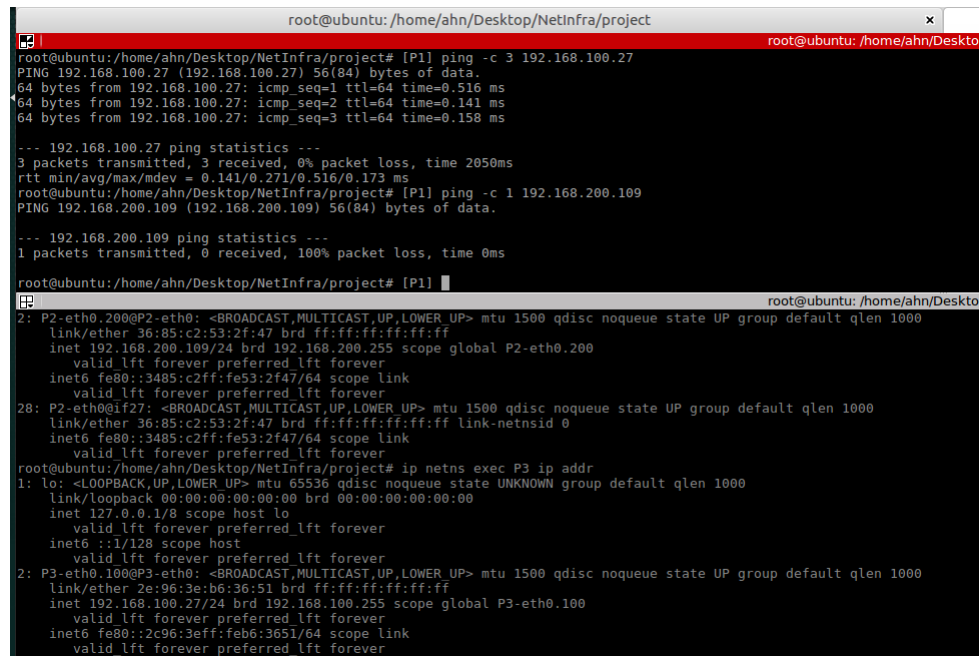
Figure 21: /etc/iproute2/route_tables

Then we execute following command:

```
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
```

3.7.3 Result

After apply above rules or policy, we have final results:



```
root@ubuntu: /home/ahn/Desktop/NetInfra/project
root@ubuntu: /home/ahn/Desktop/NetInfra/project# [P1] ping -c 3 192.168.100.27
PING 192.168.100.27 (192.168.100.27) 56(84) bytes of data.
64 bytes from 192.168.100.27: icmp_seq=1 ttl=64 time=0.516 ms
64 bytes from 192.168.100.27: icmp_seq=2 ttl=64 time=0.141 ms
64 bytes from 192.168.100.27: icmp_seq=3 ttl=64 time=0.158 ms

--- 192.168.100.27 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2050ms
rtt min/avg/max/mdev = 0.141/0.271/0.516/0.173 ms
root@ubuntu: /home/ahn/Desktop/NetInfra/project# [P1] ping -c 1 192.168.200.109
PING 192.168.200.109 (192.168.200.109) 56(84) bytes of data.

--- 192.168.200.109 ping statistics ---
1 packets transmitted, 0 received, 100% packet loss, time 0ms

root@ubuntu: /home/ahn/Desktop/NetInfra/project# [P1]
2: P2-eth0.200@P2-eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue state UP group default qlen 1000
    link/ether 36:85:c2:53:2f:47 brd ff:ff:ff:ff:ff:ff
    inet 192.168.200.109/24 brd 192.168.200.255 scope global P2-eth0.200
        valid_lft forever preferred_lft forever
    inet6 fe80::3485:c2ff:fe53:2f47/64 scope link
        valid_lft forever preferred_lft forever
28: P2-eth0@if27: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue state UP group default qlen 1000
    link/ether 36:85:c2:53:2f:47 brd ff:ff:ff:ff:ff:ff link-netnsid 0
    inet6 fe80::3485:c2ff:fe53:2f47/64 scope link
        valid_lft forever preferred_lft forever
root@ubuntu: /home/ahn/Desktop/NetInfra/project# ip netns exec P3 ip addr
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: P3-eth0.100@P3-eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue state UP group default qlen 1000
    link/ether 2e:96:3e:b6:36:51 brd ff:ff:ff:ff:ff:ff
    inet 192.168.100.27/24 brd 192.168.100.255 scope global P3-eth0.100
        valid_lft forever preferred_lft forever
    inet6 fe80::2c96:3eff:feb6:3651/64 scope link
        valid_lft forever preferred_lft forever
```

Figure 22:

To verify our configuration, we try to ping from P1 (VLAN100) to P3 (VLAN100) and P2 (VLAN200). We can not ping from P1 to P2 after applying the above configurations.

Appendices

Guide for setting lab

Firstly, Run `build_architecture`

- Question 1: Run `Q1.VLAN`
- Question 3: Run `Q2.IPTunnel`
- Question 4: Run one of three option:
 - `Q2.IPTunnel`
 - `Q4.UDPTunnel`
 - `Q4.GRETunnel`
- Question 5: Run `Q2.IPTunnel` + `Q5.IPSec`
- Question 6: Additionally, Run `Q6.InternetAccess`
- Question 7: Follow report

Note: After use config for each question, Run `killdns` and `clean`