UNIVERSITY OF LIMOGES

Faculty of Science and Technology

Master 1

Cryptology and Information Security (CRYPTIS)

Informatic Course

Network Infrastructure - Semester 2

Implement VLAN trunking with IPsec-secured L2TPv3 tunnel

DUONG Dang-Hung

Professor: Pierre-François BONNEFOI

May 7, 2023

Table of contents

1 Build the architecture

From this original network topology, where the local networks (`resD`, `resE`, `resF`, `resG`) are disjoint at layer 2:

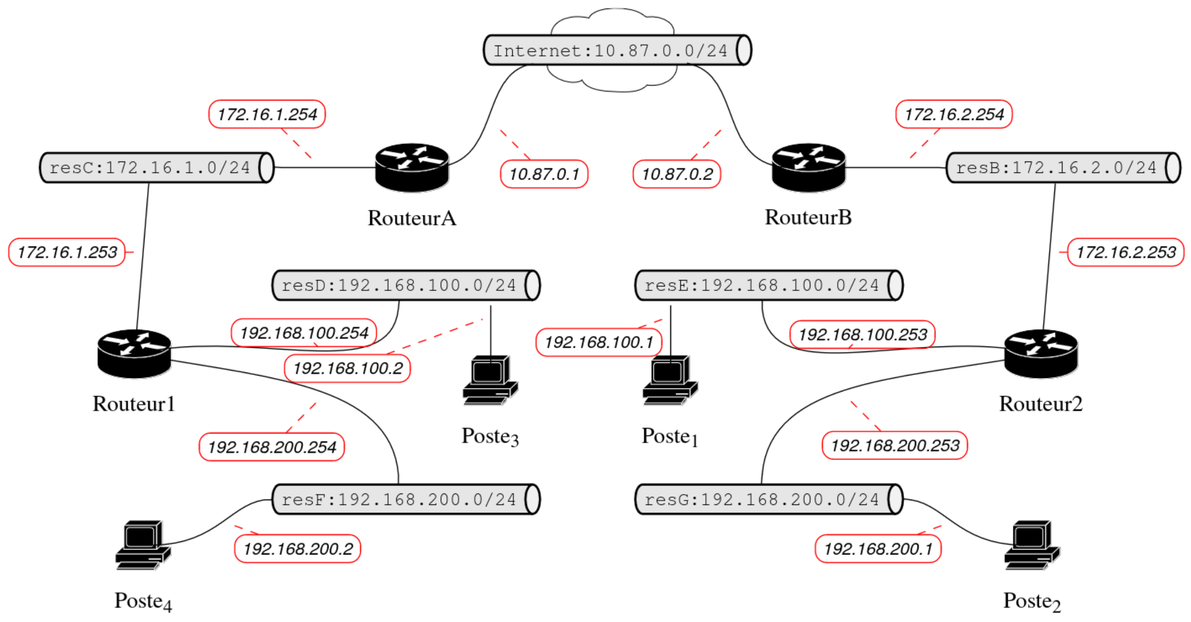


Figure 1: Original network topology

However, our goal is to make the local networks connected at layer 2 and build the following network in Figure 2 below:

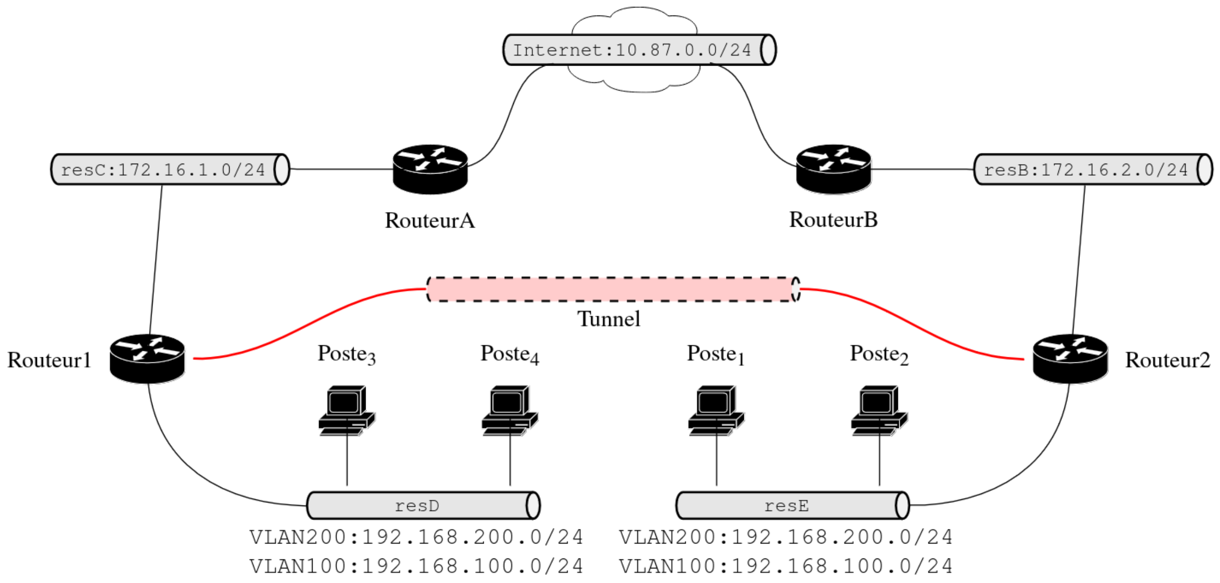


Figure 2: Desired network topology

The project starts with building the architecture. Firstly, we create the net namespaces and switches, corresponding to the routers, work stations and their networks:

```
ip netns add routA
ip netns add routB
ip netns add rout1
ip netns add rout2
ip netns add poste1
ip netns add poste2
ip netns add poste3
ip netns add poste4
```

```
ovs-vsctl add-br internet
ovs-vsctl add-br resB
ovs-vsctl add-br resC
ovs-vsctl add-br resD
ovs-vsctl add-br resE
```

Then, we configure each routers and hosts. This includes creating and adding the links between devices, turn the links up, assign them with IP address, configure their routing tables, etc.

Configure `routA`:

```
ip link add routA-eth0 type veth peer name resC-routA
ip link add routA-eth1 type veth peer name internet-routA
ip link set routA-eth0 netns routA
ip link set routA-eth1 netns routA
ovs-vsctl add-port resC resC-routA
ip link set dev resC-routA up
ovs-vsctl add-port internet internet-routA
ip link set dev internet-routA up
ip netns exec routA ip link set dev lo up
ip netns exec routA ip link set dev routA-eth0 up
ip netns exec routA ip link set dev routA-eth1 up
ip netns exec routA ip a add dev routA-eth0 172.16.1.254/24
ip netns exec routA ip a add dev routA-eth1 10.87.0.1/24
ip netns exec routA ip r add 172.16.2.0/24 via 10.87.0.2 dev routA-eth1
ip netns exec routA sysctl net.ipv4.conf.all.forwarding=1
```

Configure `routB`:

```
ip link add routB-eth0 type veth peer name resB-routB
ip link add routB-eth1 type veth peer name internet-routB
ip link set routB-eth0 netns routB
ip link set routB-eth1 netns routB
ovs-vsctl add-port resB resB-routB
ip link set dev resB-routB up
ovs-vsctl add-port internet internet-routB
ip link set dev internet-routB up
ip netns exec routB ip link set dev lo up
ip netns exec routB ip link set dev routB-eth0 up
ip netns exec routB ip link set dev routB-eth1 up
ip netns exec routB ip a add dev routB-eth0 172.16.2.254/24
ip netns exec routB ip a add dev routB-eth1 10.87.0.2/24
ip netns exec routB ip r add 172.16.1.0/24 via 10.87.0.1 dev routB-eth1
ip netns exec routB sysctl net.ipv4.conf.all.forwarding=1
```

Configure `rout1`:

```
ip link add rout1-eth0 type veth peer name resC-rout1
ip link add rout1-eth1 type veth peer name resD-rout1
ip link set rout1-eth0 netns rout1
ip link set rout1-eth1 netns rout1
ovs-vsctl add-port resC resC-rout1
ovs-vsctl add-port resD resD-rout1 trunks=100,200
ip link set dev resC-rout1 up
ip link set dev resD-rout1 up
ip netns exec rout1 ip link set dev lo up
ip netns exec rout1 ip link set dev rout1-eth0 up
ip netns exec rout1 ip link set dev rout1-eth1 up
ip netns exec rout1 ip a add dev rout1-eth0 172.16.1.253/24
ip netns exec rout1 ip r add 10.87.0.0/24 via 172.16.1.254 dev rout1-eth0
ip netns exec rout1 ip r add 172.16.2.0/24 via 172.16.1.254 dev rout1-eth0
ip netns exec rout1 sysctl net.ipv4.conf.all.forwarding=1
```

Configure `rout2`:

```
ip link add rout2-eth0 type veth peer name resB-rout2
ip link add rout2-eth1 type veth peer name resE-rout2
ip link set rout2-eth0 netns rout2
ip link set rout2-eth1 netns rout2
ovs-vsctl add-port resB resB-rout2
ovs-vsctl add-port resE resE-rout2 trunks=100,200
ip link set dev resB-rout2 up
ip link set dev resE-rout2 up
ip netns exec rout2 ip link set dev lo up
ip netns exec rout2 ip link set dev rout2-eth0 up
ip netns exec rout2 ip link set dev rout2-eth1 up
ip netns exec rout2 ip a add dev rout2-eth0 172.16.2.253/24
ip netns exec rout2 ip r add 10.87.0.0/24 via 172.16.2.254 dev rout2-eth0
ip netns exec rout2 ip r add 172.16.1.0/24 via 172.16.2.254 dev rout2-eth0
ip netns exec rout2 sysctl net.ipv4.conf.all.forwarding=1
```

Configure `poste1`:

```
ip link add poste1-eth0 type veth peer name resE-poste1
ip link set poste1-eth0 netns poste1
ip netns exec poste1 ip link set dev poste1-eth0 address 00:11:22:00:01:00
ovs-vsctl add-port resE resE-poste1 tag=100
ip link set dev resE-poste1 up
ip netns exec poste1 ip link set dev lo up
ip netns exec poste1 ip link set dev poste1-eth0 up
ip netns exec poste1 ip a add dev poste1-eth0 192.168.100.1/24
ip netns exec poste1 ip r add default dev poste1-eth0 via 192.168.100.254
```

Configure `poste2`:

```
ip link add poste2-eth0 type veth peer name resE-poste2
ip link set poste2-eth0 netns poste2
ip netns exec poste2 ip link set dev poste2-eth0 address 00:11:22:00:02:00
ovs-vsctl add-port resE resE-poste2 tag=200
ip link set dev resE-poste2 up
ip netns exec poste2 ip link set dev lo up
ip netns exec poste2 ip link set dev poste2-eth0 up
ip netns exec poste2 ip a add dev poste2-eth0 192.168.200.1/24
ip netns exec poste2 ip r add default dev poste2-eth0 via 192.168.200.254
```

Configure `poste3`:

```
ip link add poste3-eth0 type veth peer name resD-poste3
ip link set poste3-eth0 netns poste3
ip netns exec poste3 ip link set dev poste3-eth0 address 00:11:22:00:03:00
ovs-vsctl add-port resD resD-poste3 tag=100
ip link set dev resD-poste3 up
ip netns exec poste3 ip link set dev lo up
ip netns exec poste3 ip link set dev poste3-eth0 up
ip netns exec poste3 ip a add dev poste3-eth0 192.168.100.2/24
ip netns exec poste3 ip r add default dev poste3-eth0 via 192.168.100.254
```

Configure `poste4`:

```
ip link add poste4-eth0 type veth peer name resD-poste4
ip link set poste4-eth0 netns poste4
ip netns exec poste4 ip link set dev poste4-eth0 address 00:11:22:00:04:00
ovs-vsctl add-port resD resD-poste4 tag=200
ip link set dev resD-poste4 up
ip netns exec poste4 ip link set dev lo up
ip netns exec poste4 ip link set dev poste4-eth0 up
ip netns exec poste4 ip a add dev poste4-eth0 192.168.200.2/24
ip netns exec poste4 ip r add default dev poste4-eth0 via 192.168.200.254
```

Note that in the configuration of `rout1` and `rout2`, we created 2 VLANs with ID 100 and 200 to be able to separate the communication inside specific networks `192.168.100.0/24` and `192.168.200.0/24`. For this to function, we have to add particular options `trunks` and `tag` when connecting the hosts and routers to the switches, so that the switches are configured to transfer traffic with "VLAN tagging" and "VLAN trunking":

- The "tag" option is used to configure a port to have a specific VLAN tag (or multiple tags if using the – tag=VLAN1 – tag=VLAN2 syntax). This means that any traffic received on this port with the specified VLAN tag(s) will be associated with the specified VLAN(s) and forwarded accordingly. Thus we apply it on the connection link between the routers and the switches.

- The "trunks" option is used to configure a port as a trunk port, which means that it will carry traffic from multiple VLANs. Any traffic received on this port will not have a specific VLAN tag associated with it, but will rather have a VLAN tag added by the switch based on the VLAN configuration. Thus we apply it on the connection link between the hosts (postes) and the switches.

Also, for the configuration to be more convenient in the following exercises, I decided to set up static MAC addresses for the network cards of the host devices (which is quite reasonable since in real life each device is also assigned with a constant MAC address instead of having random ones):

- `poste1-eth0`: `00:11:22:00:01:00`
- `poste2-eth0`: `00:11:22:00:02:00`
- `poste3-eth0`: `00:11:22:00:03:00`
- `poste4-eth0`: `00:11:22:00:04:00`

After having finished constructing the network topology, we would have this graph (since the network is quite big, the graph might need to be zoomed to be seen):

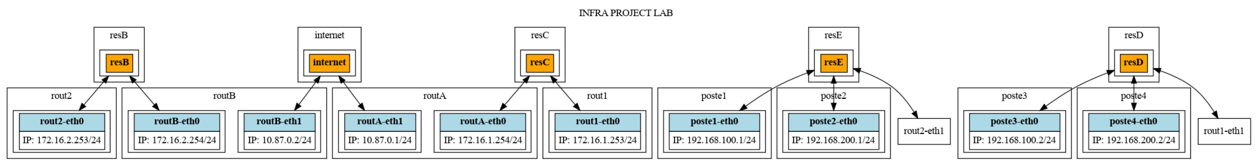
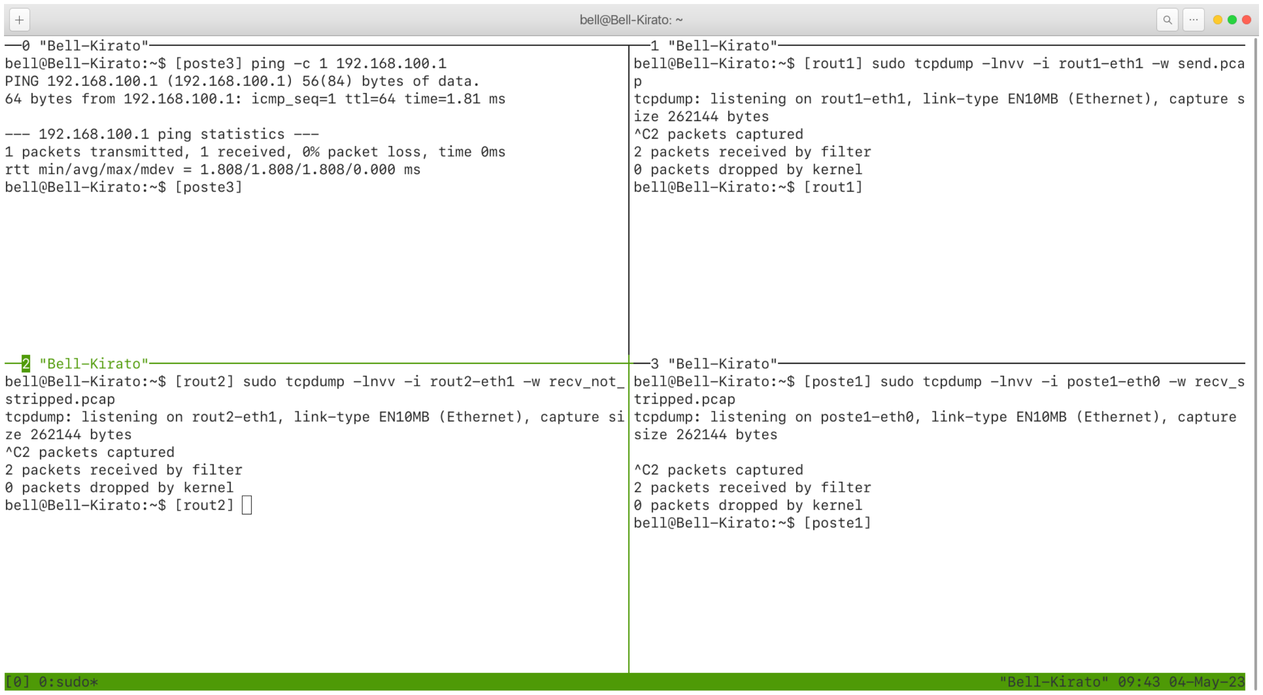


Figure 3: Network graph

2 Test the functioning of VLANs

We will try to test the functioning of VLANs configured and inspect the nature of the traffic to verify the functioning of "VLAN tagging". In this example, we will ping from `poste1` to `poste3` (both in VLAN100) (the process is the same in the case of VLAN200), while sniffing on 2 VLAN trunking interfaces `rout1-eth1`, `rout1-eth2`, and the interface of the destination host `poste1-eth0` using `tcpdump`:



```
bell@Bell-Kirato: ~
─0 "Bell-Kirato"──────
bell@Bell-Kirato:~$ [poste3] ping -c 1 192.168.100.1
PING 192.168.100.1 (192.168.100.1) 56(84) bytes of data.
64 bytes from 192.168.100.1: icmp_seq=1 ttl=64 time=1.81 ms

--- 192.168.100.1 ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 1.808/1.808/1.808/0.000 ms
bell@Bell-Kirato:~$ [poste3]
```

```
─1 "Bell-Kirato"──────
bell@Bell-Kirato:~$ [rout1] sudo tcpdump -lnvv -i rout1-eth1 -w send.pca
p
tcpdump: listening on rout1-eth1, link-type EN10MB (Ethernet), capture s
ize 262144 bytes
^C2 packets captured
2 packets received by filter
0 packets dropped by kernel
bell@Bell-Kirato:~$ [rout1]
```

```
─2 "Bell-Kirato"──────
bell@Bell-Kirato:~$ [rout2] sudo tcpdump -lnvv -i rout2-eth1 -w recv_not_
stripped.pcap
tcpdump: listening on rout2-eth1, link-type EN10MB (Ethernet), capture si
ze 262144 bytes
^C2 packets captured
2 packets received by filter
0 packets dropped by kernel
bell@Bell-Kirato:~$ [rout2]
```

```
─3 "Bell-Kirato"──────
bell@Bell-Kirato:~$ [poste1] sudo tcpdump -lnvv -i poste1-eth0 -w recv_s
tripped.pcap
tcpdump: listening on poste1-eth0, link-type EN10MB (Ethernet), capture
size 262144 bytes

^C2 packets captured
2 packets received by filter
0 packets dropped by kernel
bell@Bell-Kirato:~$ [poste1]
```

```
[0] 0:sudo*                                        "Bell-Kirato" 09:43 04-May-23
```

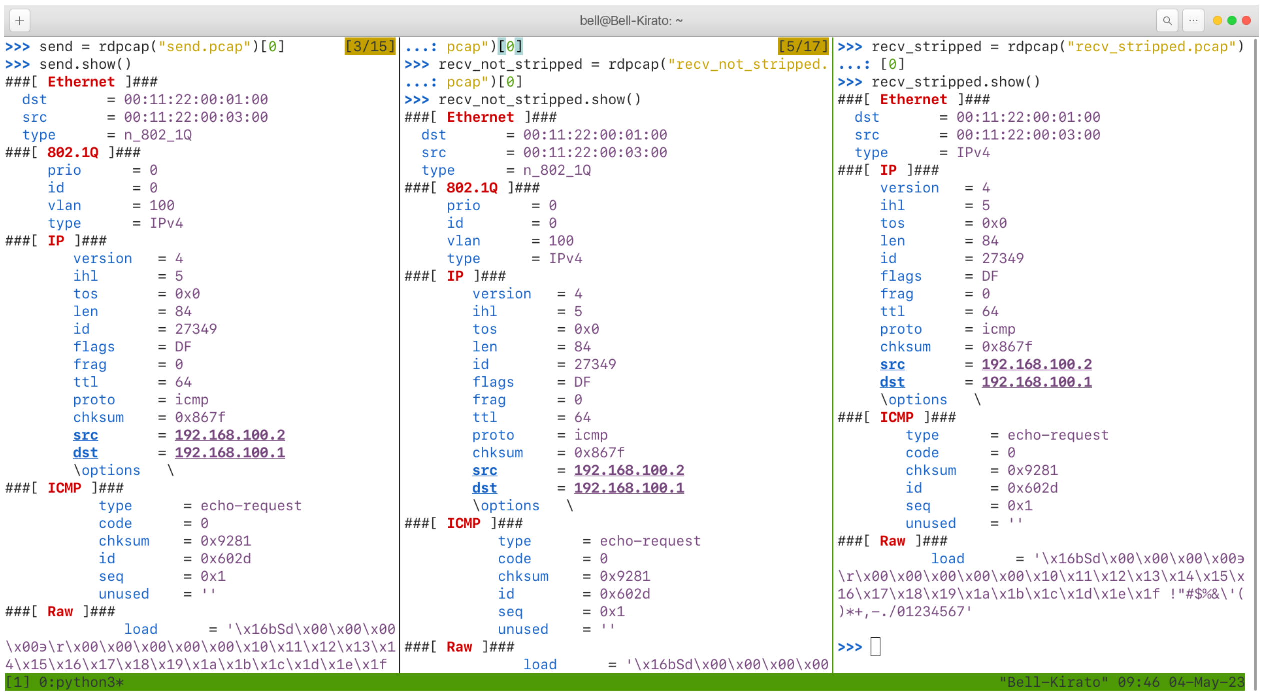
Figure 4: Ping from `poste3` to `poste1`

Here, the packets captured are saved in `.pcap` files so as to be analyzed later:

- Packets received on `rout1-eth1` are saved in `send.pcap`
- Packets received on `rout2-eth1` are saved in `recv_not_stripped.pcap`
- Packets received on `poste1-eth0` are saved in `recv_stripped.pcap`

Note that we have configured the MAC address of `poste1` to be `00:11:22:00:01:00`, and that of `poste3` to be `00:11:22:00:03:00`. This information will be shown in the following Figure 5.

To inspect these captured packets, `scapy` is utilized:



```
bell@Bell-Kirato: ~
```

```
>>> send = rdpcap("send.pcap")[0]         [3/15]
>>> send.show()
###[ Ethernet ]###
  dst       = 00:11:22:00:01:00
  src       = 00:11:22:00:03:00
  type      = n_802_1Q
###[ 802.1Q ]###
     prio      = 0
     id        = 0
     vlan      = 100
     type      = IPv4
###[ IP ]###
        version   = 4
        ihl       = 5
        tos       = 0x0
        len       = 84
        id        = 27349
        flags     = DF
        frag      = 0
        ttl       = 64
        proto     = icmp
        chksum    = 0x867f
        src       = 192.168.100.2
        dst       = 192.168.100.1
        \options   \
###[ ICMP ]###
           type      = echo-request
           code      = 0
           chksum    = 0x9281
           id        = 0x602d
           seq       = 0x1
           unused    = ''
###[ Raw ]###
              load      = '\x16bSd\x00\x00\x00
\x00э\r\x00\x00\x00\x00\x00\x10\x11\x12\x13\x1
4\x15\x16\x17\x18\x19\x1a\x1b\x1c\x1d\x1e\x1f
```

```
...: pcap")[0]                                [5/17]
>>> recv_not_stripped = rdpcap("recv_not_stripped.
...: pcap")[0]
>>> recv_not_stripped.show()
###[ Ethernet ]###
  dst       = 00:11:22:00:01:00
  src       = 00:11:22:00:03:00
  type      = n_802_1Q
###[ 802.1Q ]###
     prio      = 0
     id        = 0
     vlan      = 100
     type      = IPv4
###[ IP ]###
        version   = 4
        ihl       = 5
        tos       = 0x0
        len       = 84
        id        = 27349
        flags     = DF
        frag      = 0
        ttl       = 64
        proto     = icmp
        chksum    = 0x867f
        src       = 192.168.100.2
        dst       = 192.168.100.1
        \options   \
###[ ICMP ]###
           type      = echo-request
           code      = 0
           chksum    = 0x9281
           id        = 0x602d
           seq       = 0x1
           unused    = ''
###[ Raw ]###
              load      = '\x16bSd\x00\x00\x00\x00
```

```
>>> recv_stripped = rdpcap("recv_stripped.pcap")
...: [0]
>>> recv_stripped.show()
###[ Ethernet ]###
  dst       = 00:11:22:00:01:00
  src       = 00:11:22:00:03:00
  type      = IPv4
###[ IP ]###
     version   = 4
     ihl       = 5
     tos       = 0x0
     len       = 84
     id        = 27349
     flags     = DF
     frag      = 0
     ttl       = 64
     proto     = icmp
     chksum    = 0x867f
     src       = 192.168.100.2
     dst       = 192.168.100.1
     \options   \
###[ ICMP ]###
        type      = echo-request
        code      = 0
        chksum    = 0x9281
        id        = 0x602d
        seq       = 0x1
        unused    = ''
###[ Raw ]###
           load      = '\x16bSd\x00\x00\x00\x00э
\r\x00\x00\x00\x00\x00\x10\x11\x12\x13\x14\x15\x
16\x17\x18\x19\x1a\x1b\x1c\x1d\x1e\x1f !"#$%&\'(
)*+,-./01234567'

>>> 
```

```
[1] 0:python3*                                              "Bell-Kirato" 09:46 04-May-23
```

Figure 5: Analyzing packets handled by VLAN tagging with `scapy`

From Figure 5, we see that the packets, which are the same ICMP Request from `poste3`, received on interface `rout1-eth1` and `rout2-eth1` has an additional `802.1Q` header, with a field `vlan = 100`. This is exactly the VLAN header (or the *VLAN tag*) which we are expecting. The process can be explained as follows:

- `poste3` sends an ICMP Request to the destination 192.168.100.1 (`poste1`)
- The packet travels to the port `resD-poste3` of switch `resD` which was configured with `tag=100`
- The packet arrives at the port and is injected with the `802.1Q` header which has a tag `vlan = 100`, then it is forwarded to `rout1`
- The packet goes from `rout1` to `rout2` (through the tunnel, which we will discuss in the following sessions)
- The packet arrives at `rout2` and continues to switch `resE` using interface `rout2-eth1`
- The packet travels to the port `resE-rout2` which was configured with `trunks=100,200`
- Since the packet was tagged with the VLAN ID: `vlan = 100`, which is included in the option `trunks=100,200`, switch `resE` finds the port configured with `tag=100`, which is `resE-poste1`, or the one peered with `poste1-eth0`, strips the VLAN tag, and then forward the packet to that output port
- `poste1` receives the packet with no VLAN tag

3 Study L2TPv3 and VXLAN technology

3.1 L2TPv3 vs VXLAN

To compare L2TPv3 and VXLAN, we will briefly study their similarities and differences.

Overall, they both encapsulate layer 2 packets inside IP packets and transport them over an IP network, for example the Internet. However, apart from this point, there are a couple of differences between their purposes and applications.

Regarding L2TPv3:

- It is an improved version of L2TP (Layer 2 Tunneling Protocol), which is a tunneling protocol.
- As a tunnel, it has 2 endpoints only: L2TP access concentrator (LAC) and the L2TP network server (LNS).
- It allows the creation of a virtual private dialup network (VPDN) to connect a remote client to its corporate network by using a shared infrastructure, which could be the Internet or a service provider's network.
- It is a layer 2 protocol, which means it was designed to provide a way to transport data link layer protocols, such as Ethernet frames, over an IP network.

On the other hand, VXLAN:

- Is a network virtualization technology.
- Is capable of creating an overlay network or tunnel.
- Can have multiple endpoints named VTEPs (VXLAN Tunnel EndPoints)
- It provides a scalable and flexible method for building virtual networks that can span multiple physical networks $\longrightarrow$ VXLAN plays an important role in cloud and data center environments.
- Is a protocol used to extend Layer 2 networks over a Layer 3 infrastructure.
- Is typically used in data center environments to provide network virtualization and overlay networks.
- It operates between layer 2 and layer 3 $\longrightarrow$ A "layer 2.5" protocol.

3.2 Implement VXLAN in Linux with Open vSwitch

In order to implement VXLAN in Linux with Open vSwitch, it is essential to create a bridge with type VXLAN, configure the bridge as well as the system appropriately to connect to the VXLAN. For example, in our project, if we would like to create a VXLAN for VLAN100: On `poste1`:

- Create a bridge:

```
ovs-vsctl add-br br0
```

- Turn the bridge up:

```
ip link set br0 up
```

- Create the VXLAN interface to represent its VTEP:

```
ovs-vsctl add-port br0 vx_poste3 -- set interface vx_poste3 type=vxlan
    options:remote_ip=192.168.100.2 options:local_ip=192.168.100.1
```

Symmetrically, we configure `poste3`:

- Create a bridge:

```
ovs-vsctl add-br br0
```

- Turn the bridge up:

```
ip link set br0 up
```

- Create the VXLAN interface to represent its VTEP:

```
ovs-vsctl add-port br0 vx_poste1 -- set interface vx_poste1 type=vxlan
    options:remote_ip=192.168.100.1 options:local_ip=192.168.100.2
```

The overlay network is now operational, and everything connected to br0 on either machine will appear to be connected to the same layer 2 network.

3.3 L2TPv3 traffic encryption solutions

Compared to the original L2TP, L2TPv3 supports encryption of tunnel traffic using a variety of encryption algorithms, such as 3DES, AES, and Blowfish, therefore provides improved security through the use of IPsec encryption and authentication, which can be used to protect the L2TPv3 control and data packets. When IPsec is used for encryption, it provides security at the network layer and creates a virtual private network (VPN) tunnel between the L2TPv3 tunnel endpoints. This enables secure transmission of data over the public Internet. This type of combination is well-known as "L2TPv3 over IPSec", which we might see later. Besides IPSec, there are several other options such as SSL/TLS, MACsec, AAA, etc.

3.4 L2TPv3/VXLAN vs MPLS

In comparison, they have a few similarities:

- All of them allows for the creation of VPNs (Virtual Private Networks) by using labels to segregate traffic from different VPNs, offering secure and scalable connectivity between geographically dispersed sites.
- They make routing decisions for packets by labelling the packet with an additional header.
- All three protocols support encryption and other security mechanisms to protect data in transit.

However, there are some significant differences:

- L2TPv3 is layer 2 protocol, VLAN and MPLS are layer 2.5 technologies.
- Encapsulation:
  - L2TPv3 and VXLAN: encapsulate Layer 2 frames within IP packets
  - MPLS encapsulates IP packets within MPLS headers
- Main purposes:
  - MPLS is designed to improve the performance and efficiency of packet forwarding by using labels to route packets through the network (used to transport layer 3 packets).
  - L2TPv3 is designed to provide a way to transport data link layer protocols over an IP network (used to transport layer 2 packets).
  - VXLAN is designed to building overlaying virtual networks that can span multiple physical networks (used to transport layer 2 packets).
- Along the path of the packet:
  - MPLS: all immediate routers have to examine the label to determine the next hop, the final router in the path removes the label and forwards the packet to its destination $\longrightarrow$ MPLS uses label-switching to forward traffic.
  - L2TPv3 and VXLAN: only the endpoints (tunnel endpoints in the case of L2TPv3, VTEPs in the case of VXLAN) have to look at the label, remove it and forwards the packet to its destination $\longrightarrow$ L2TPv3 and VXLAN use traditional routing protocols.
- To provide end-to-end connectivity for IP-based services:
  - L2TPv3 and VXLAN are widely used within data center and enterprise networks
  - MPLS is usually used within service provider networks

4 Implementing L2TPv3 tunnel

The process of creating the L2TPv3 tunnel in IP encapsulation mode was very well-guided in the subject document.

We implement the tunnel on `rout1` side:

```
# Config tunnel rout1
ip netns exec rout1 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 rout1 ip l2tp add session tunnel_id 3000 session_id 1000
    peer_session_id 2000
ip netns exec rout1 ip link set l2tpeth0 up
ip netns exec rout1 ip a flush dev rout1-eth1
ip netns exec rout1 brctl addbr tunnel
ip netns exec rout1 brctl addif tunnel l2tpeth0
ip netns exec rout1 brctl addif tunnel rout1-eth1
ip netns exec rout1 ip link set tunnel up
ip netns exec rout1 ip link add link tunnel name tunnel.100 type vlan id
    100
ip netns exec rout1 ip link set tunnel.100 up
ip netns exec rout1 ip link add link tunnel name tunnel.200 type vlan id
    200
ip netns exec rout1 ip link set tunnel.200 up
ip netns exec rout1 ip addr add 192.168.100.254/24 dev tunnel.100
ip netns exec rout1 ip addr add 192.168.200.254/24 dev tunnel.200
ip netns exec rout1 ip link set dev l2tpeth0 mtu 1500
```

Symmetrically, we implement the tunnel on `rout2` side:

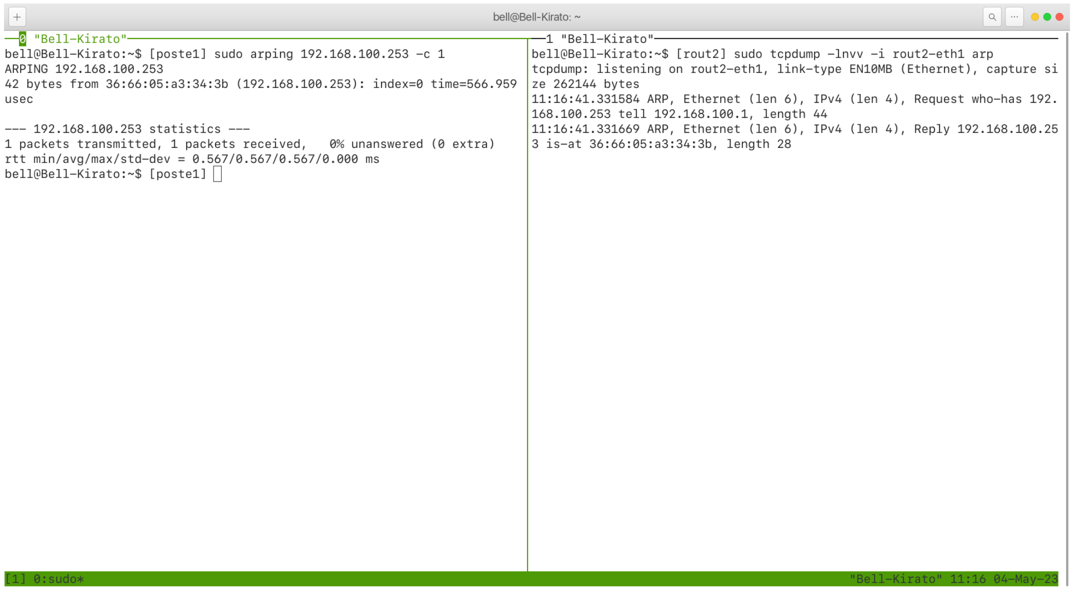
```
# Config tunnel rout2
ip netns exec rout2 ip l2tp add tunnel remote 172.16.1.253 local
    172.16.2.253 encap ip tunnel_id 4000 peer_tunnel_id 3000
ip netns exec rout2 ip l2tp add session tunnel_id 4000 session_id 2000
    peer_session_id 1000
ip netns exec rout2 ip link set l2tpeth0 up
ip netns exec rout2 ip a flush dev rout2-eth1
ip netns exec rout2 brctl addbr tunnel
ip netns exec rout2 brctl addif tunnel l2tpeth0
ip netns exec rout2 brctl addif tunnel rout2-eth1
ip netns exec rout2 ip link set tunnel up
ip netns exec rout2 ip link add link tunnel name tunnel.100 type vlan id
    100
ip netns exec rout2 ip link set tunnel.100 up
ip netns exec rout2 ip link add link tunnel name tunnel.200 type vlan id
    200
ip netns exec rout2 ip link set tunnel.200 up
ip netns exec rout2 ip addr add 192.168.100.253/24 dev tunnel.100
ip netns exec rout2 ip addr add 192.168.200.253/24 dev tunnel.200
ip netns exec rout2 ip link set l2tpeth0 mtu 1500
```

Now that the tunnel has been configured, we can try to see if it was installed successfully and functions well.

4.1 Verify ARP protocol's functioning

First, we will verify to see if the ARP protocol works normally for the discovery of machines regardless of their side connection to the Internet.

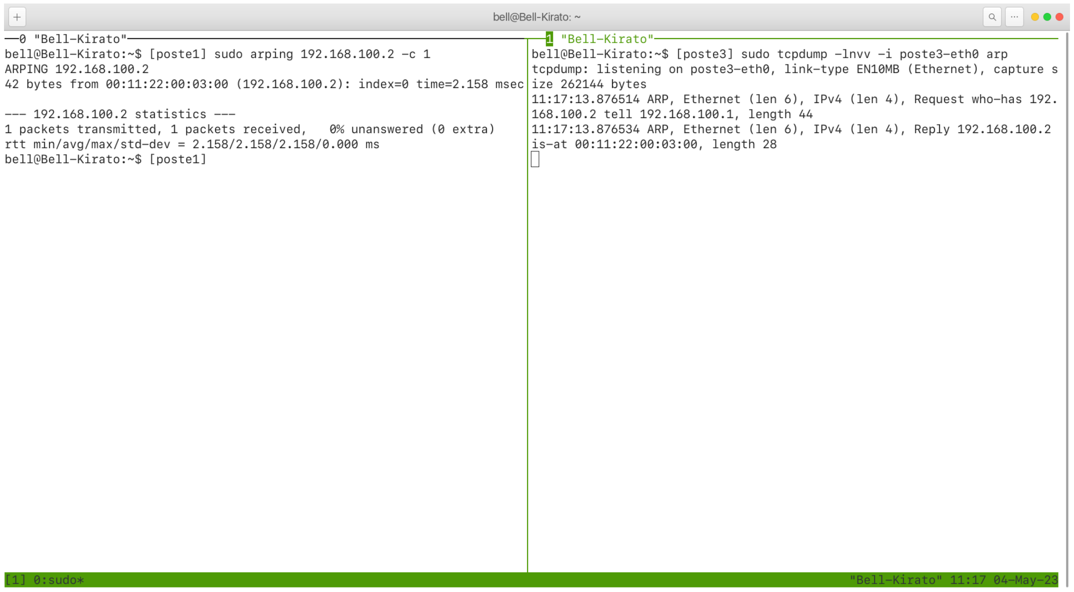
In this example, we will try to send ARP Requests from `poste1` to `rout1` (same side) and `poste3` (opposite side, same VLAN) using the `arping` command.



```
bell@Bell-Kirato:~$ [poste1] sudo arping 192.168.100.253 -c 1
ARPING 192.168.100.253
42 bytes from 36:66:05:a3:34:3b (192.168.100.253): index=0 time=566.959
usec

--- 192.168.100.253 statistics ---
1 packets transmitted, 1 packets received,   0% unanswered (0 extra)
rtt min/avg/max/std-dev = 0.567/0.567/0.567/0.000 ms
bell@Bell-Kirato:~$ [poste1]
```

```
bell@Bell-Kirato:~$ [rout2] sudo tcpdump -lnvv -i rout2-eth1 arp
tcpdump: listening on rout2-eth1, link-type EN10MB (Ethernet), capture si
ze 262144 bytes
11:16:41.331584 ARP, Ethernet (len 6), IPv4 (len 4), Request who-has 192.
168.100.253 tell 192.168.100.1, length 44
11:16:41.331669 ARP, Ethernet (len 6), IPv4 (len 4), Reply 192.168.100.25
3 is-at 36:66:05:a3:34:3b, length 28
```

Figure 6: ARP Protocol between `poste1` and `rout1` (same side)



```
bell@Bell-Kirato:~$ [poste1] sudo arping 192.168.100.2 -c 1
ARPING 192.168.100.2
42 bytes from 00:11:22:00:03:00 (192.168.100.2): index=0 time=2.158 msec

--- 192.168.100.2 statistics ---
1 packets transmitted, 1 packets received,   0% unanswered (0 extra)
rtt min/avg/max/std-dev = 2.158/2.158/2.158/0.000 ms
bell@Bell-Kirato:~$ [poste1]
```

```
bell@Bell-Kirato:~$ [poste3] sudo tcpdump -lnvv -i poste3-eth0 arp
tcpdump: listening on poste3-eth0, link-type EN10MB (Ethernet), capture s
ize 262144 bytes
11:17:13.876514 ARP, Ethernet (len 6), IPv4 (len 4), Request who-has 192.
168.100.2 tell 192.168.100.1, length 44
11:17:13.876534 ARP, Ethernet (len 6), IPv4 (len 4), Reply 192.168.100.2
is-at 00:11:22:00:03:00, length 28
```

Figure 7: ARP Protocol between `poste1` and `poste3` (opposite side)

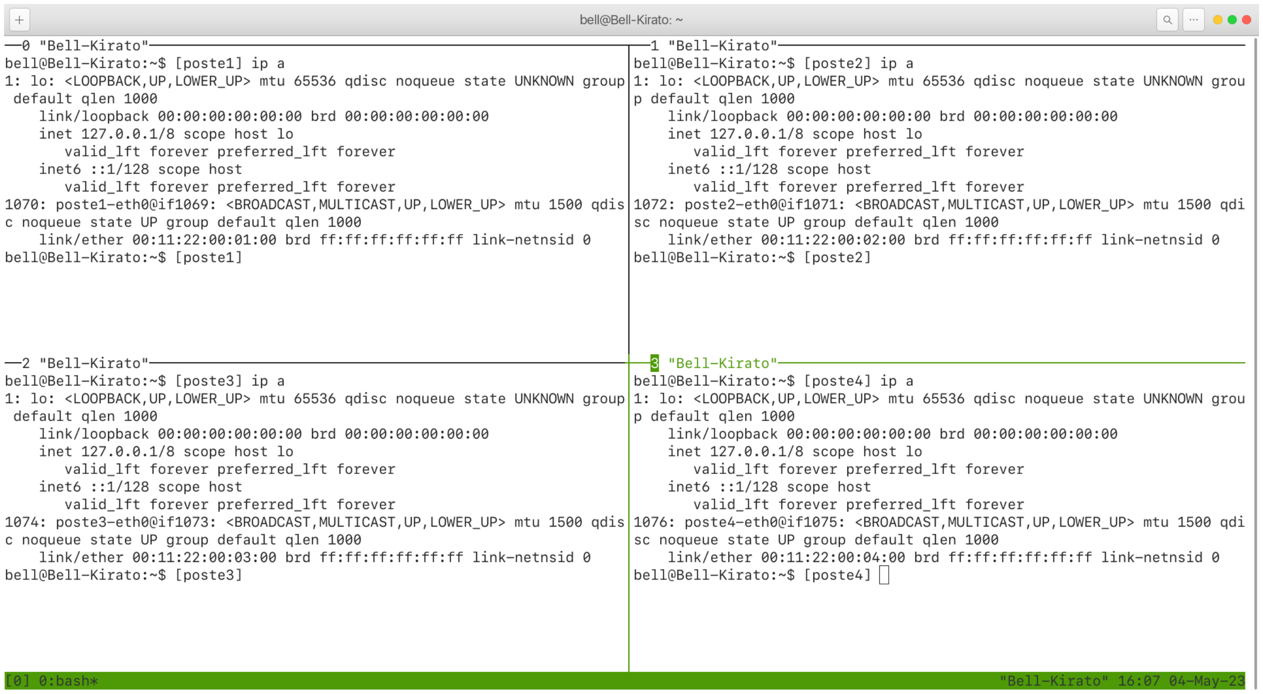
It can be seen from Figure 6 and Figure 8 that the ARP Protocol works well in this case.

4.2 Verify the DHCP service

We will implement a DHCP server on `rout1` and verify that the DHCP service can be offered to all 4 hosts regardless of their side connection to the internet. Firstly, all the pre-registered IP addresses of the hosts should be flushed:

```
$ sudo ip netns exec poste1 ip a flush dev poste1-eth0
$ sudo ip netns exec poste2 ip a flush dev poste2-eth0
$ sudo ip netns exec poste3 ip a flush dev poste3-eth0
$ sudo ip netns exec poste4 ip a flush dev poste4-eth0
```

We check that all the IP addresses of the network interfaces of the hosts are deleted:



```
bell@Bell-Kirato: ~
─0 "Bell-Kirato"─
bell@Bell-Kirato:~$ [poste1] 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
1070: poste1-eth0@if1069: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdis
c noqueue state UP group default qlen 1000
    link/ether 00:11:22:00:01:00 brd ff:ff:ff:ff:ff:ff link-netnsid 0
bell@Bell-Kirato:~$ [poste1]
```

```
─1 "Bell-Kirato"─
bell@Bell-Kirato:~$ [poste2] ip a
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN grou
p 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
1072: poste2-eth0@if1071: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdi
sc noqueue state UP group default qlen 1000
    link/ether 00:11:22:00:02:00 brd ff:ff:ff:ff:ff:ff link-netnsid 0
bell@Bell-Kirato:~$ [poste2]
```

```
─2 "Bell-Kirato"─
bell@Bell-Kirato:~$ [poste3] 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
1074: poste3-eth0@if1073: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdis
c noqueue state UP group default qlen 1000
    link/ether 00:11:22:00:03:00 brd ff:ff:ff:ff:ff:ff link-netnsid 0
bell@Bell-Kirato:~$ [poste3]
```

```
─3 "Bell-Kirato"─
bell@Bell-Kirato:~$ [poste4] ip a
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN grou
p 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
1076: poste4-eth0@if1075: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdi
sc noqueue state UP group default qlen 1000
    link/ether 00:11:22:00:04:00 brd ff:ff:ff:ff:ff:ff link-netnsid 0
bell@Bell-Kirato:~$ [poste4]
```

```
[0] 0:bash*                                      "Bell-Kirato" 16:07 04-May-23
```

Figure 8: Flush all network interfaces of `poste1`, `poste2`, `poste3`, and `poste4`

To configure the DHCP server on `rout1`, we configure the file `/etc/dnsmasq.conf` by adding these lines:

```
dhcp-range=192.168.100.1,192.168.100.2,12h
dhcp-range=192.168.200.1,192.168.200.2,12h
dhcp-host=00:11:22:00:01:00,192.168.100.1,12h
dhcp-host=00:11:22:00:02:00,192.168.200.1,12h
dhcp-host=00:11:22:00:03:00,192.168.100.2,12h
dhcp-host=00:11:22:00:04:00,192.168.200.2,12h
```

We used option `dhcp-host` in order to configure static IP addresses for the hosts using their MAC address, and option `dhcp-range` to specify the possible IP address range to assign. This is also the reason why I configure static IP addresses for the hosts at the beginning.

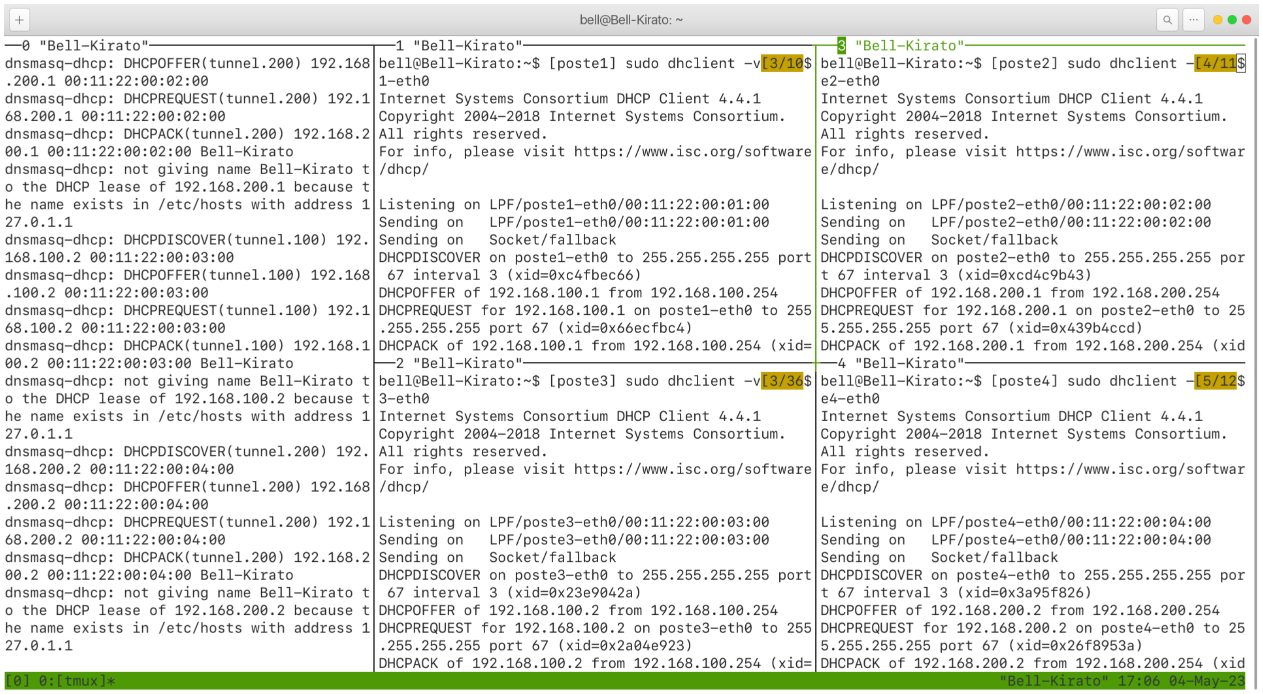
After that, we start the DHCP Server on `rout1` using `dnsmasq` command:

```
$ sudo ip netns exec rout1 dnsmasq -d -z
dnsmasq: started, version 2.80 cachesize 150
dnsmasq: compile time options: IPv6 GNU-getopt DBus i18n IDN DHCP DHCPv6
    no-Lua TFTP conntrack ipset auth nettlehash DNSSEC loop-detect inotify
    dumpfile
dnsmasq-dhcp: DHCP, IP range 192.168.200.1 -- 192.168.200.2, lease time 12
    h
dnsmasq-dhcp: DHCP, IP range 192.168.100.1 -- 192.168.100.2, lease time 12
    h
dnsmasq: reading /etc/resolv.conf
dnsmasq: using nameserver 127.0.0.53#53
dnsmasq: read /etc/hosts - 7 addresses
```

On each host, we request IP address using `dhclient`:

```
$ [poste1] sudo dhclient -v poste1-eth0
$ [poste2] sudo dhclient -v poste2-eth0
$ [poste3] sudo dhclient -v poste3-eth0
$ [poste4] sudo dhclient -v poste4-eth0
```

We can confirm that the DHCP Protocol, including DHCP DISCOVER, DHCP OFFER, DHCP REQUEST, DHCP PACK messages, are carried out normally for all 4 hosts, regardless of their side connection to the Internet:



```
0 "Bell-Kirato"
dnsmasq-dhcp: DHCPOFFER(tunnel.200) 192.168
.200.1 00:11:22:00:02:00
dnsmasq-dhcp: DHCPREQUEST(tunnel.200) 192.1
68.200.1 00:11:22:00:02:00
dnsmasq-dhcp: DHCPACK(tunnel.200) 192.168.2
00.1 00:11:22:00:02:00 Bell-Kirato
dnsmasq-dhcp: not giving name Bell-Kirato t
o the DHCP lease of 192.168.200.1 because t
he name exists in /etc/hosts with address 1
27.0.1.1
dnsmasq-dhcp: DHCPDISCOVER(tunnel.100) 192.
168.100.2 00:11:22:00:03:00
dnsmasq-dhcp: DHCPOFFER(tunnel.100) 192.168
.100.2 00:11:22:00:03:00
dnsmasq-dhcp: DHCPREQUEST(tunnel.100) 192.1
68.100.2 00:11:22:00:03:00
dnsmasq-dhcp: DHCPACK(tunnel.100) 192.168.1
00.2 00:11:22:00:03:00 Bell-Kirato
dnsmasq-dhcp: not giving name Bell-Kirato t
o the DHCP lease of 192.168.100.2 because t
he name exists in /etc/hosts with address 1
27.0.1.1
dnsmasq-dhcp: DHCPDISCOVER(tunnel.200) 192.
168.200.2 00:11:22:00:04:00
dnsmasq-dhcp: DHCPOFFER(tunnel.200) 192.168
.200.2 00:11:22:00:04:00
dnsmasq-dhcp: DHCPREQUEST(tunnel.200) 192.1
68.200.2 00:11:22:00:04:00
dnsmasq-dhcp: DHCPACK(tunnel.200) 192.168.2
00.2 00:11:22:00:04:00 Bell-Kirato
dnsmasq-dhcp: not giving name Bell-Kirato t
o the DHCP lease of 192.168.200.2 because t
he name exists in /etc/hosts with address 1
27.0.1.1
```

```
1 "Bell-Kirato"
bell@Bell-Kirato:~$ [poste1] sudo dhclient -v[3/10$
1-eth0
Internet Systems Consortium DHCP Client 4.4.1
Copyright 2004-2018 Internet Systems Consortium.
All rights reserved.
For info, please visit https://www.isc.org/software
/dhcp/

Listening on LPF/poste1-eth0/00:11:22:00:01:00
Sending on   LPF/poste1-eth0/00:11:22:00:01:00
Sending on   Socket/fallback
DHCPDISCOVER on poste1-eth0 to 255.255.255.255 port
 67 interval 3 (xid=0xc4fbec66)
DHCPOFFER of 192.168.100.1 from 192.168.100.254
DHCPREQUEST for 192.168.100.1 on poste1-eth0 to 255
.255.255.255 port 67 (xid=0x66ecfbc4)
DHCPACK of 192.168.100.1 from 192.168.100.254 (xid=
```

```
3 "Bell-Kirato"
bell@Bell-Kirato:~$ [poste2] sudo dhclient -[4/11$
e2-eth0
Internet Systems Consortium DHCP Client 4.4.1
Copyright 2004-2018 Internet Systems Consortium.
All rights reserved.
For info, please visit https://www.isc.org/softwar
e/dhcp/

Listening on LPF/poste2-eth0/00:11:22:00:02:00
Sending on   LPF/poste2-eth0/00:11:22:00:02:00
Sending on   Socket/fallback
DHCPDISCOVER on poste2-eth0 to 255.255.255.255 por
t 67 interval 3 (xid=0xcd4c9b43)
DHCPOFFER of 192.168.200.1 from 192.168.200.254
DHCPREQUEST for 192.168.200.1 on poste2-eth0 to 25
5.255.255.255 port 67 (xid=0x439b4ccd)
DHCPACK of 192.168.200.1 from 192.168.200.254 (xid
```

```
2 "Bell-Kirato"
bell@Bell-Kirato:~$ [poste3] sudo dhclient -v[3/36$
3-eth0
Internet Systems Consortium DHCP Client 4.4.1
Copyright 2004-2018 Internet Systems Consortium.
All rights reserved.
For info, please visit https://www.isc.org/software
/dhcp/

Listening on LPF/poste3-eth0/00:11:22:00:03:00
Sending on   LPF/poste3-eth0/00:11:22:00:03:00
Sending on   Socket/fallback
DHCPDISCOVER on poste3-eth0 to 255.255.255.255 port
 67 interval 3 (xid=0x23e9042a)
DHCPOFFER of 192.168.100.2 from 192.168.100.254
DHCPREQUEST for 192.168.100.2 on poste3-eth0 to 255
.255.255.255 port 67 (xid=0x2a04e923)
DHCPACK of 192.168.100.2 from 192.168.100.254 (xid=
```

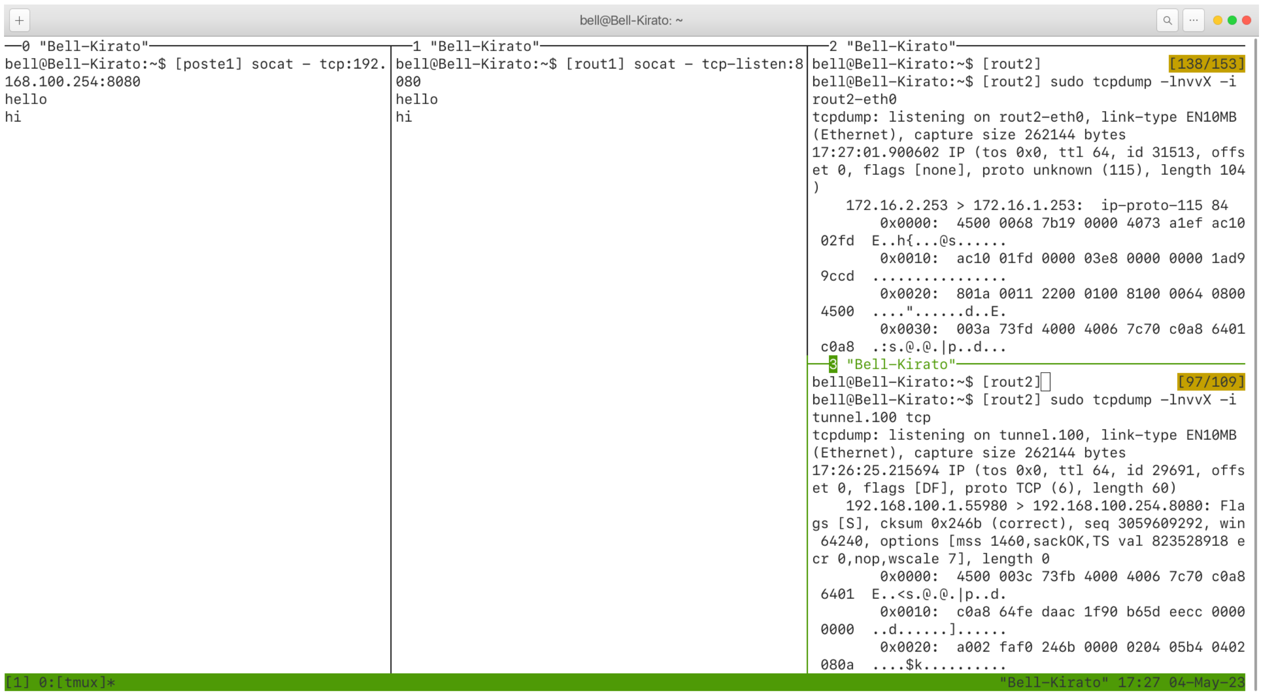
```
4 "Bell-Kirato"
bell@Bell-Kirato:~$ [poste4] sudo dhclient -[5/12$
e4-eth0
Internet Systems Consortium DHCP Client 4.4.1
Copyright 2004-2018 Internet Systems Consortium.
All rights reserved.
For info, please visit https://www.isc.org/softwar
e/dhcp/

Listening on LPF/poste4-eth0/00:11:22:00:04:00
Sending on   LPF/poste4-eth0/00:11:22:00:04:00
Sending on   Socket/fallback
DHCPDISCOVER on poste4-eth0 to 255.255.255.255 por
t 67 interval 3 (xid=0x3a95f826)
DHCPOFFER of 192.168.200.2 from 192.168.200.254
DHCPREQUEST for 192.168.200.2 on poste4-eth0 to 25
5.255.255.255 port 67 (xid=0x26f8953a)
DHCPACK of 192.168.200.2 from 192.168.200.254 (xid
```

```
[0] 0:[tmux]*                                        "Bell-Kirato" 17:06 04-May-23
```

Figure 9: The DHCP Protocol captured on `rout1` and `poste1`, `poste2`, `poste3`, and `poste4`

4.3 Establish TCP connection through the tunnel

We also can verify the functioning of the tunnel by establishing a TCP connection between `rout1` and `poste1` using `socat`:



```
bell@Bell-Kirato: ~

0 "Bell-Kirato"
bell@Bell-Kirato:~$ [poste1] socat - tcp:192.
168.100.254:8080
hello
hi
```

```
1 "Bell-Kirato"
bell@Bell-Kirato:~$ [rout1] socat - tcp-listen:8
080
hello
hi
```

```
2 "Bell-Kirato"
bell@Bell-Kirato:~$ [rout2]                 [138/153]
bell@Bell-Kirato:~$ [rout2] sudo tcpdump -lnvvX -i
rout2-eth0
tcpdump: listening on rout2-eth0, link-type EN10MB
(Ethernet), capture size 262144 bytes
17:27:01.900602 IP (tos 0x0, ttl 64, id 31513, offs
et 0, flags [none], proto unknown (115), length 104
)
    172.16.2.253 > 172.16.1.253:  ip-proto-115 84
        0x0000:  4500 0068 7b19 0000 4073 a1ef ac10
 02fd  E..h{...@s......
        0x0010:  ac10 01fd 0000 03e8 0000 0000 1ad9
 9ccd  ................
        0x0020:  801a 0011 2200 0100 8100 0064 0800
 4500  ....".....d..E.
        0x0030:  003a 73fd 4000 4006 7c70 c0a8 6401
 c0a8  .:s.@.@.|p..d...
```

```
3 "Bell-Kirato"
bell@Bell-Kirato:~$ [rout2]                  [97/109]
bell@Bell-Kirato:~$ [rout2] sudo tcpdump -lnvvX -i
tunnel.100 tcp
tcpdump: listening on tunnel.100, link-type EN10MB
(Ethernet), capture size 262144 bytes
17:26:25.215694 IP (tos 0x0, ttl 64, id 29691, offs
et 0, flags [DF], proto TCP (6), length 60)
    192.168.100.1.55980 > 192.168.100.254.8080: Fla
gs [S], cksum 0x246b (correct), seq 3059609292, win
 64240, options [mss 1460,sackOK,TS val 823528918 e
cr 0,nop,wscale 7], length 0
        0x0000:  4500 003c 73fb 4000 4006 7c70 c0a8
 6401  E..<s.@.@.|p..d.
        0x0010:  c0a8 64fe daac 1f90 b65d eecc 0000
 0000  ..d......]......
        0x0020:  a002 faf0 246b 0000 0204 05b4 0402
 080a  ....$k..........
```

```
[1] 0:[tmux]*                 "Bell-Kirato" 17:27 04-May-23
```

Figure 10: TCP connection established between `rout1` and `poste1`

The TCP connection was established successfully between `rout1` and `poste1`. In Figure 13, it can be seen that when we capture the packets on `rout2`, the TCP packets arrives on interface `tunnel.100` in their original form (TCP packets), while the packets arrives on interface `rout2-eth0` were encapsulated using L2TP protocol (`ip-proto-115`).

5 Compare different tunnel options

5.1 IP encapsulation vs. UDP encapsulation

Here we would try to compare 2 encapsulation modes for the tunnel: IP and UDP.

IP encapsulation mode is originally applied:

```
ip netns exec rout1 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 rout2 ip l2tp add tunnel remote 172.16.1.253 local
   172.16.2.253 encap ip tunnel_id 4000 peer_tunnel_id 3000
```

After the network architecture is built successfully, we listen on interface `routA-eth0` of `routA`, so as to capture the packet encapsulated in IP mode and save them in the file `encap_ip.pcap`:

```
$ sudo ip netns exec routA tcpdump -lnvvX -i routA-eth0 -w encap_ip.pcap
```

Then we send a ping from `poste3` to `poste1`:

```
$ sudo ip netns exec poste3 ping 192.168.100.1 -c 1
```

To use UDP encapsulation mode, we have to change the `encap ip` argument to `encap udp`, and we also need to specify the UDP ports which the routers on each end of the tunnel use to communicate. In this example, we will choose port 8888 for `rout1` and port 9999 for `rout2`:

```
ip netns exec rout1 ip l2tp add tunnel remote 172.16.2.253 local
   172.16.1.253 encap udp udp_sport 8888 udp_dport 9999 tunnel_id 3000
   peer_tunnel_id 4000
ip netns exec rout2 ip l2tp add tunnel remote 172.16.1.253 local
   172.16.2.253 encap udp udp_sport 9999 udp_dport 8888 tunnel_id 4000
   peer_tunnel_id 3000
```

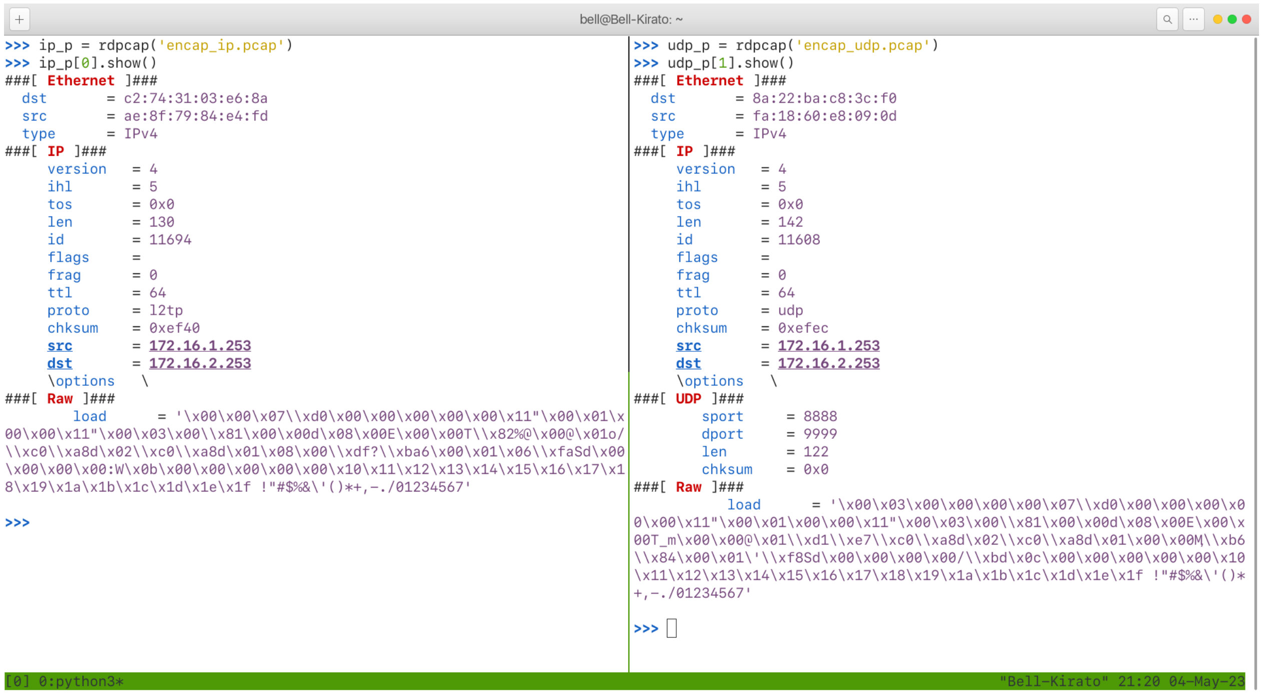
After the network architecture is re-built successfully, we do the same process as above to capture the packet encapsulated in UDP mode and save them in the file `encap_udp.pcap`:

```
$ sudo ip netns exec routA tcpdump -lnvvX -i routA-eth0 -w encap_udp.pcap
```

Then once again, we send a ping from `poste3` to `poste1`:

```
$ sudo ip netns exec poste3 ping 192.168.100.1 -c 1
```

Once we have the packets captures in the two `.pcap` files, we can utilize `scapy` to analyze and compare them:



bell@Bell-Kirato: ~

```
>>> ip_p = rdpcap('encap_ip.pcap')
>>> ip_p[0].show()
###[ Ethernet ]###
  dst       = c2:74:31:03:e6:8a
  src       = ae:8f:79:84:e4:fd
  type      = IPv4
###[ IP ]###
     version   = 4
     ihl       = 5
     tos       = 0x0
     len       = 130
     id        = 11694
     flags     =
     frag      = 0
     ttl       = 64
     proto     = l2tp
     chksum    = 0xef40
     src       = 172.16.1.253
     dst       = 172.16.2.253
     \options   \
###[ Raw ]###
        load      = '\x00\x00\x07\\xd0\x00\x00\x00\x00\x00\x11"\x00\x01\x
00\x00\x11"\x00\x03\x00\\x81\x00\x00d\x08\x00E\x00\x00T\\x82%@\x00@\x01o/
\\xc0\\xa8d\x02\\xc0\\xa8d\x01\x08\x00\\xdf?\\xba6\x00\x01\x06\\xfaSd\x00
\x00\x00\x00:W\x0b\x00\x00\x00\x00\x00\x10\x11\x12\x13\x14\x15\x16\x17\x1
8\x19\x1a\x1b\x1c\x1d\x1e\x1f !"#$%&\'()*+,-./01234567'

>>>
```

```
>>> udp_p = rdpcap('encap_udp.pcap')
>>> udp_p[1].show()
###[ Ethernet ]###
  dst       = 8a:22:ba:c8:3c:f0
  src       = fa:18:60:e8:09:0d
  type      = IPv4
###[ IP ]###
     version   = 4
     ihl       = 5
     tos       = 0x0
     len       = 142
     id        = 11608
     flags     =
     frag      = 0
     ttl       = 64
     proto     = udp
     chksum    = 0xefec
     src       = 172.16.1.253
     dst       = 172.16.2.253
     \options   \
###[ UDP ]###
        sport     = 8888
        dport     = 9999
        len       = 122
        chksum    = 0x0
###[ Raw ]###
           load      = '\x00\x03\x00\x00\x00\x00\x07\\xd0\x00\x00\x00\x0
0\x00\x11"\x00\x01\x00\x00\x11"\x00\x03\x00\\x81\x00\x00d\x08\x00E\x00\x
00T_m\x00\x00@\x01\\xd1\\xe7\\xc0\\xa8d\x02\\xc0\\xa8d\x01\x00\x00M\\xb6
\\x84\x00\x01\'\\xf8Sd\x00\x00\x00\x00/\\xbd\x0c\x00\x00\x00\x00\x00\x10
\x11\x12\x13\x14\x15\x16\x17\x18\x19\x1a\x1b\x1c\x1d\x1e\x1f !"#$%&\'()*
+,-./01234567'

>>>
```

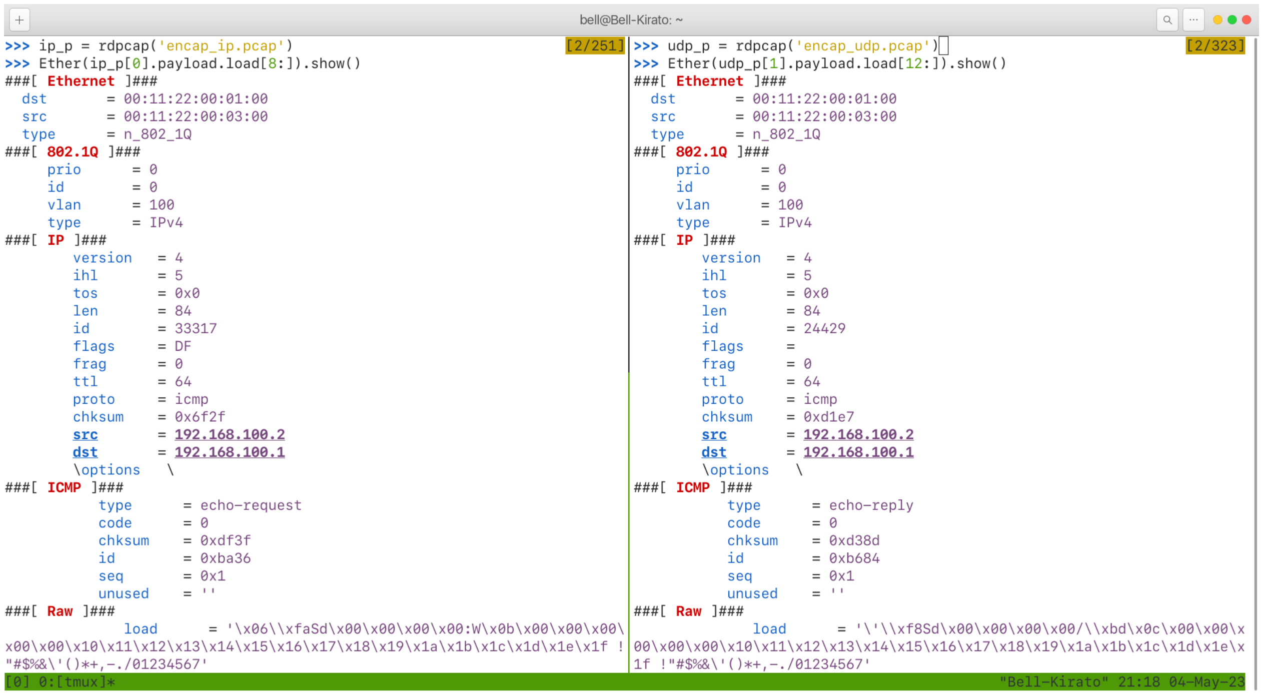
```
[0] 0:python3*                                         "Bell-Kirato" 21:20 04-May-23
```

Figure 11: Encapsulated packet in IP encapsulation mode vs. UDP encapsulation mode

The MAC address in these two packets are not the same because we re-built the network architecture to change the tunnel mode, and since we were not interested in specifying static MAC address for `routA` and `routB`, they are assigned with another random MAC address after the network is re-built.

We can already spot the difference between the two modes: the encapsulated packets in UDP encapsulation mode has an additional UDP header injected between the IP header and the L2TP encapsulated packet (as specified by the field `proto`, or simply saying "next header" in the `IP` header of the packets). The UDP header specifies the source port and the destination port that the 2 routers at 2 ends of the tunnel used to communicate.

We can also try to retrieve the original packet from the payload of the encapsulated packets. For those in IP encapsulation mode, the original packet content is located after the 8th byte of the payload, while in UDP encapsulation mode, it is located after the 12th byte:



```
>>> ip_p = rdpcap('encap_ip.pcap')                                   [2/251]
>>> Ether(ip_p[0].payload.load[8:]).show()
###[ Ethernet ]###
  dst       = 00:11:22:00:01:00
  src       = 00:11:22:00:03:00
  type      = n_802_1Q
###[ 802.1Q ]###
     prio      = 0
     id        = 0
     vlan      = 100
     type      = IPv4
###[ IP ]###
        version   = 4
        ihl       = 5
        tos       = 0x0
        len       = 84
        id        = 33317
        flags     = DF
        frag      = 0
        ttl       = 64
        proto     = icmp
        chksum    = 0x6f2f
        src       = 192.168.100.2
        dst       = 192.168.100.1
        \options   \
###[ ICMP ]###
           type      = echo-request
           code      = 0
           chksum    = 0xdf3f
           id        = 0xba36
           seq       = 0x1
           unused    = ''
###[ Raw ]###
              load      = '\x06\\xfaSd\x00\x00\x00\x00:W\x0b\x00\x00\x00\
x00\x00\x10\x11\x12\x13\x14\x15\x16\x17\x18\x19\x1a\x1b\x1c\x1d\x1e\x1f !
"#$%&\'()*+,-./01234567'
[0] 0:[tmux]*
```

```
>>> udp_p = rdpcap('encap_udp.pcap')                                 [2/323]
>>> Ether(udp_p[1].payload.load[12:]).show()
###[ Ethernet ]###
  dst       = 00:11:22:00:01:00
  src       = 00:11:22:00:03:00
  type      = n_802_1Q
###[ 802.1Q ]###
     prio      = 0
     id        = 0
     vlan      = 100
     type      = IPv4
###[ IP ]###
        version   = 4
        ihl       = 5
        tos       = 0x0
        len       = 84
        id        = 24429
        flags     =
        frag      = 0
        ttl       = 64
        proto     = icmp
        chksum    = 0xd1e7
        src       = 192.168.100.2
        dst       = 192.168.100.1
        \options   \
###[ ICMP ]###
           type      = echo-reply
           code      = 0
           chksum    = 0xd38d
           id        = 0xb684
           seq       = 0x1
           unused    = ''
###[ Raw ]###
              load      = '\'\\xf8Sd\x00\x00\x00\x00/\\xbd\x0c\x00\x00\x
00\x00\x00\x10\x11\x12\x13\x14\x15\x16\x17\x18\x19\x1a\x1b\x1c\x1d\x1e\x
1f !"#$%&\'()*+,-./01234567'
                                                "Bell-Kirato" 21:18 04-May-23
```

Figure 12: Obtaining original packet from payload in IP vs. UDP encap modes

5.2 Compare to tunnel made with GRE in GRETAP mode

Instead of L2TPv3 tunnel, it is also possible to set up GRE tunnel in GRETAP mode.

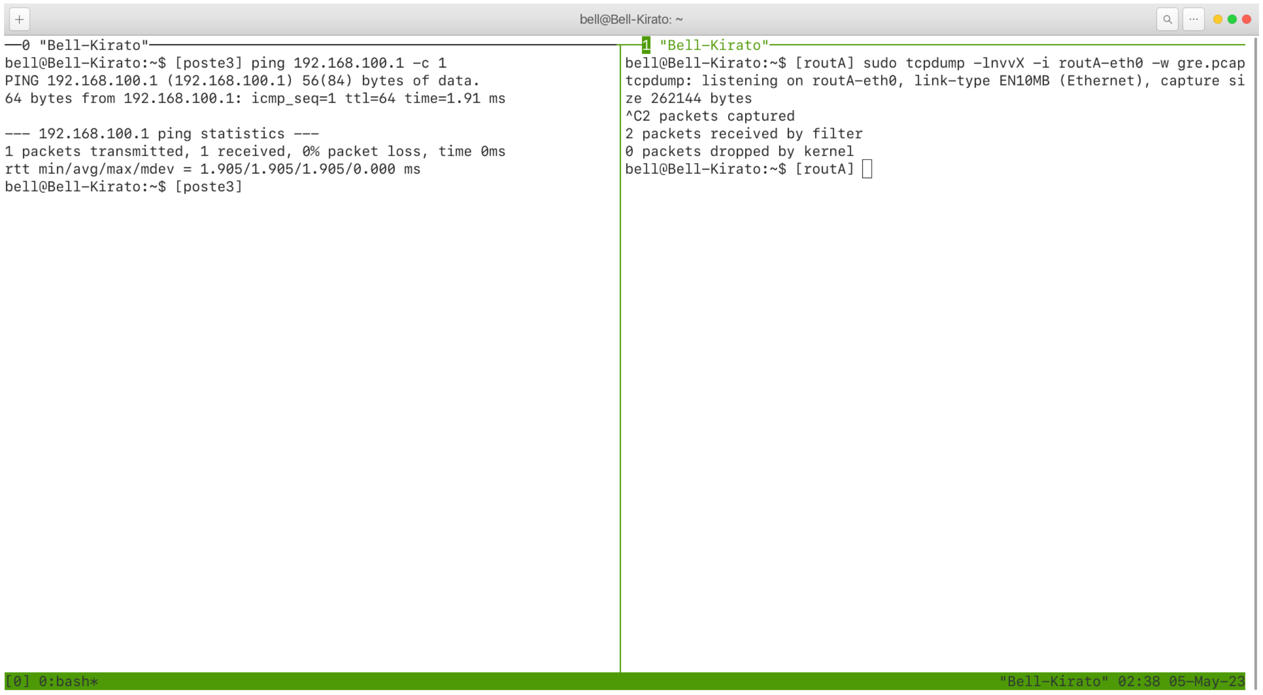
To configure `rout1`:

```
$ sudo ip netns exec rout1 ip l add eoip1 type gretap remote 172.16.2.253
    local 172.16.1.253 nopmtudisc
$ sudo ip netns exec rout1 ip l set dev eoip1 up
$ sudo ip netns exec rout1 brctl addbr tunnel
$ sudo ip netns exec rout1 brctl addif tunnel eoip1
$ sudo ip netns exec rout1 brctl addif tunnel rout1-eth1
$ sudo ip netns exec rout1 ip l set tunnel up
$ sudo ip netns exec rout1 ip l add link tunnel name tunnel.100 type vlan
    id 100
$ sudo ip netns exec rout1 ip l set tunnel.100 up
$ sudo ip netns exec rout1 ip l add link tunnel name tunnel.200 type vlan
    id 200
$ sudo ip netns exec rout1 ip l set tunnel.200 up
$ sudo ip netns exec rout1 ip a add 192.168.100.254/24 dev tunnel.100
$ sudo ip netns exec rout1 ip a add 192.168.200.254/24 dev tunnel.200
```

To configure `rout2`:

```
$ sudo ip netns exec rout2 ip l add eoip1 type gretap remote 172.16.1.253
    local 172.16.2.253 nopmtudisc
$ sudo ip netns exec rout2 ip l set dev eoip1 up
$ sudo ip netns exec rout2 brctl addbr tunnel
$ sudo ip netns exec rout2 brctl addif tunnel eoip1
$ sudo ip netns exec rout2 brctl addif tunnel rout2-eth1
$ sudo ip netns exec rout2 ip l set tunnel up
$ sudo ip netns exec rout2 ip l add link tunnel name tunnel.100 type vlan
    id 100
$ sudo ip netns exec rout2 ip l set tunnel.100 up
$ sudo ip netns exec rout2 ip l add link tunnel name tunnel.200 type vlan
    id 200
$ sudo ip netns exec rout2 ip l set tunnel.200 up
$ sudo ip netns exec rout2 ip a add 192.168.100.253/24 dev tunnel.100
$ sudo ip netns exec rout2 ip a add 192.168.200.253/24 dev tunnel.200
```

After having finished configuring the tunnel for `rout1` and `rout2`, we can try to capture the traffic. In this example, once again we will capture the packets arrive at interface `routA-eth0` of `routA`, while sending ICMP requests from `poste3` to `poste1`:



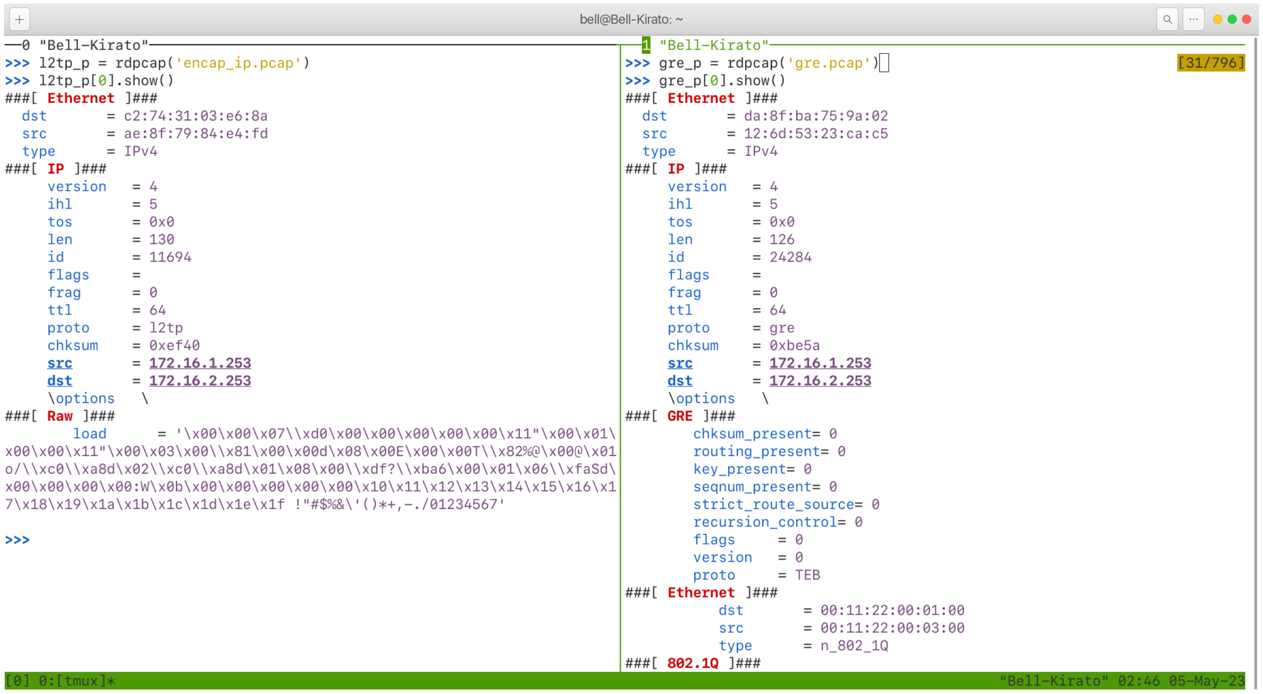
```
bell@Bell-Kirato: ~

—0 "Bell-Kirato"—
bell@Bell-Kirato:~$ [poste3] ping 192.168.100.1 -c 1
PING 192.168.100.1 (192.168.100.1) 56(84) bytes of data.
64 bytes from 192.168.100.1: icmp_seq=1 ttl=64 time=1.91 ms

--- 192.168.100.1 ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 1.905/1.905/1.905/0.000 ms
bell@Bell-Kirato:~$ [poste3]
```

```
1 "Bell-Kirato"—
bell@Bell-Kirato:~$ [routA] sudo tcpdump -lnvvX -i routA-eth0 -w gre.pcap
tcpdump: listening on routA-eth0, link-type EN10MB (Ethernet), capture si
ze 262144 bytes
^C2 packets captured
2 packets received by filter
0 packets dropped by kernel
bell@Bell-Kirato:~$ [routA]
```

```
[0] 0:bash*                                        "Bell-Kirato" 02:38 05-May-23
```

Figure 13: Capture traffic on `routA-eth0` after setting up GRETAP tunnel

Then, we can use `scapy` to analyze the difference between the packets encapsulated using L2TPv3 protocol and GRETAP protocol:



```
─0 "Bell-Kirato"─────────────────────────────────
>>> l2tp_p = rdpcap('encap_ip.pcap')
>>> l2tp_p[0].show()
###[ Ethernet ]###
  dst       = c2:74:31:03:e6:8a
  src       = ae:8f:79:84:e4:fd
  type      = IPv4
###[ IP ]###
     version   = 4
     ihl       = 5
     tos       = 0x0
     len       = 130
     id        = 11694
     flags     =
     frag      = 0
     ttl       = 64
     proto     = l2tp
     chksum    = 0xef40
     src       = 172.16.1.253
     dst       = 172.16.2.253
     \options   \
###[ Raw ]###
        load      = '\x00\x00\x07\\xd0\x00\x00\x00\x00\x00\x11"\x00\x01\
x00\x00\x11"\x00\x03\x00\\x81\x00\x00d\x08\x00E\x00\x00T\\x82%@\x00@\x01
o/\\xc0\\xa8d\x02\\xc0\\xa8d\x01\x08\x00\\xdf?\\xba6\x00\x01\x06\\xfaSd\
x00\x00\x00\x00:W\x0b\x00\x00\x00\x00\x00\x10\x11\x12\x13\x14\x15\x16\x1
7\x18\x19\x1a\x1b\x1c\x1d\x1e\x1f !"#$%&\'()*+,-./01234567'

>>>
```

```
1 "Bell-Kirato"──────────────────────────────────
>>> gre_p = rdpcap('gre.pcap')                                   [31/796]
>>> gre_p[0].show()
###[ Ethernet ]###
  dst       = da:8f:ba:75:9a:02
  src       = 12:6d:53:23:ca:c5
  type      = IPv4
###[ IP ]###
     version   = 4
     ihl       = 5
     tos       = 0x0
     len       = 126
     id        = 24284
     flags     =
     frag      = 0
     ttl       = 64
     proto     = gre
     chksum    = 0xbe5a
     src       = 172.16.1.253
     dst       = 172.16.2.253
     \options   \
###[ GRE ]###
        chksum_present= 0
        routing_present= 0
        key_present= 0
        seqnum_present= 0
        strict_route_source= 0
        recursion_control= 0
        flags     = 0
        version   = 0
        proto     = TEB
###[ Ethernet ]###
           dst       = 00:11:22:00:01:00
           src       = 00:11:22:00:03:00
           type      = n_802_1Q
###[ 802.1Q ]###
```

```
[0] 0:[tmux]*                                  "Bell-Kirato" 02:46 05-May-23
```

Figure 14: Packets encapsulated using L2TPv3 vs. GRETAP

Notice how the original packet remains its original form (with its own Ethernet header, `802.1Q` or VLAN header, etc.) and are encapsulated with an additional GRE header, then another IP header and Ethernet header, instead of being encapsulated in the payload of the new packet in the form of raw bytes as in the case of L2TP protocol.

We can also see that the MTU on the tunnel interfaces of the routers are reduced to 1462, instead of 1500, due to the increasing information in packet headers.

```
$ sudo ip netns exec rout1 ip l show eoip1
5: eoip1@NONE: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1462 qdisc fq_codel
    master tunnel state UNKNOWN mode DEFAULT group default qlen 1000
     link/ether ee:de:d5:f0:73:ea brd ff:ff:ff:ff:ff:ff
```

6 Setting up IPsec encryption on L2TPv3 tunnel

In order to set up IPsec encryption, combining both AH (*Authentication Header*) and ESP (*Encapsulating Security Payload*) in "transport" mode, on the L2TPv3 tunnel between `rout1` and `rout2`, we will utilize the `ip xfrm` command. Each protocol requires its own SA for each direction, so 4 SAs are needed to a VPN using AH+ESP.

First, we choose an arbitrary SPI (*Security Parameters Index*) for the SAs (*Security Associations*), and randomly generate the the keys for SHA256 and AES protocols. AH SAs use 256-bit long and ESP SAs use 160-bit long keys:

```
# Choose a random SPI (Security Parameters Index)
index=0x12345678

# Generate random secret keys for symmetric encryption: AH SAs using 256
    bit (32 byte) long and ESP SAs using 160 bit (20 byte) long keys
ah_key_1=$(openssl rand -hex 32)
esp_key_1=$(openssl rand -hex 20)
ah_key_2=$(openssl rand -hex 32)
esp_key_2=$(openssl rand -hex 20)
```

After having the keys and SPI, we can configure the SAD (*Security Association Database*) using `ip xfrm state` and SPD (*Security Policy Database*) using `ip xfrm policy`. On `rout1`:

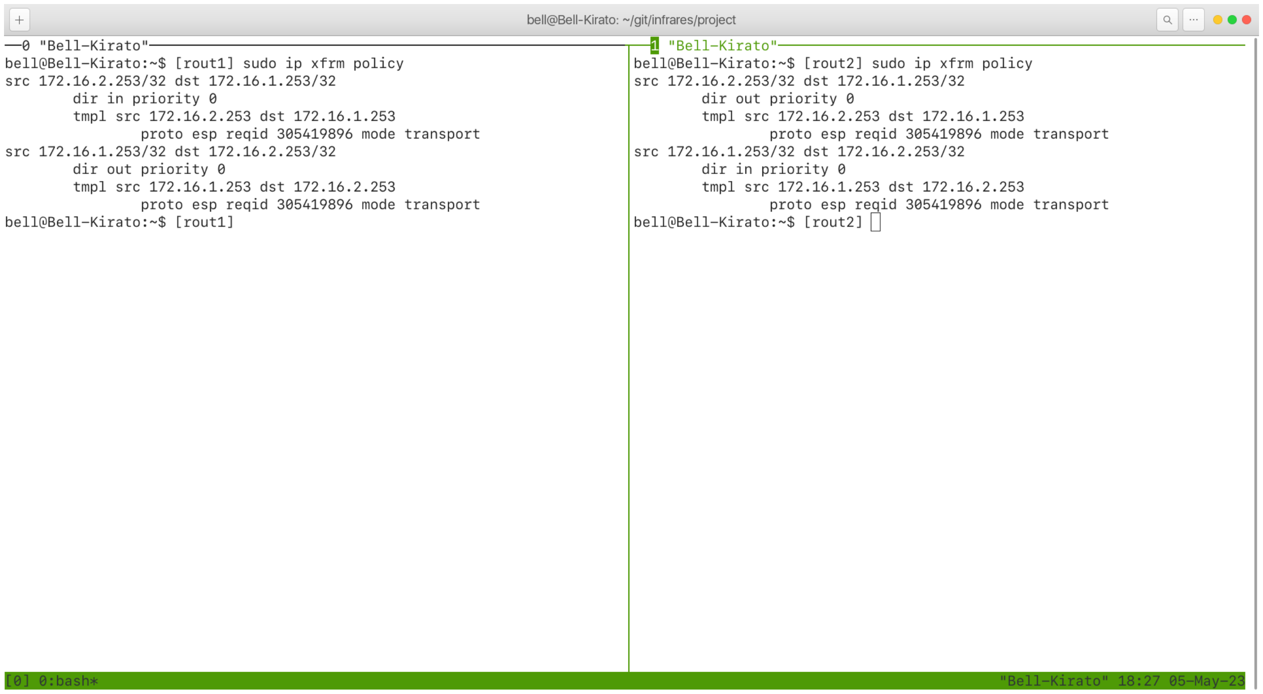
```
# # Config IPsec on rout1 end
# Flush the SAD and SPD
sudo ip netns exec rout1 ip xfrm state flush
sudo ip netns exec rout1 ip xfrm policy flush

# Security association
sudo ip netns exec rout1 ip xfrm state add src 172.16.1.253 dst
    172.16.2.253 proto esp spi $index reqid $index mode transport auth
    sha256 0x$ah_key_1 enc "rfc3686(ctr(aes))" 0x$esp_key_1
sudo ip netns exec rout1 ip xfrm state add src 172.16.2.253 dst
    172.16.1.253 proto esp spi $index reqid $index mode transport auth
    sha256 0x$ah_key_2 enc "rfc3686(ctr(aes))" 0x$esp_key_2

# Security policies
sudo ip netns exec rout1 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 $index mode transport
sudo ip netns exec rout1 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 $index mode transport
```

Symmetrically, on `rout2`:

```
# # Config IPsec on rout2 end
# Flush the SAD and SPD
sudo ip netns exec rout2 ip xfrm state flush
sudo ip netns exec rout2 ip xfrm policy flush

# Security association
sudo ip netns exec rout2 ip xfrm state add src 172.16.1.253 dst
    172.16.2.253 proto esp spi $index reqid $index mode transport auth
    sha256 0x$ah_key_1 enc "rfc3686(ctr(aes))" 0x$esp_key_1
sudo ip netns exec rout2 ip xfrm state add src 172.16.2.253 dst
    172.16.1.253 proto esp spi $index reqid $index mode transport auth
    sha256 0x$ah_key_2 enc "rfc3686(ctr(aes))" 0x$esp_key_2

# Security policies
sudo ip netns exec rout2 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 $index mode transport
sudo ip netns exec rout2 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 $index mode transport
```

All the configurations are in the file `config_ipsec`.

We can check if the SAs are configured correctly using `ip xfrm policy` and `ip xfrm state`:



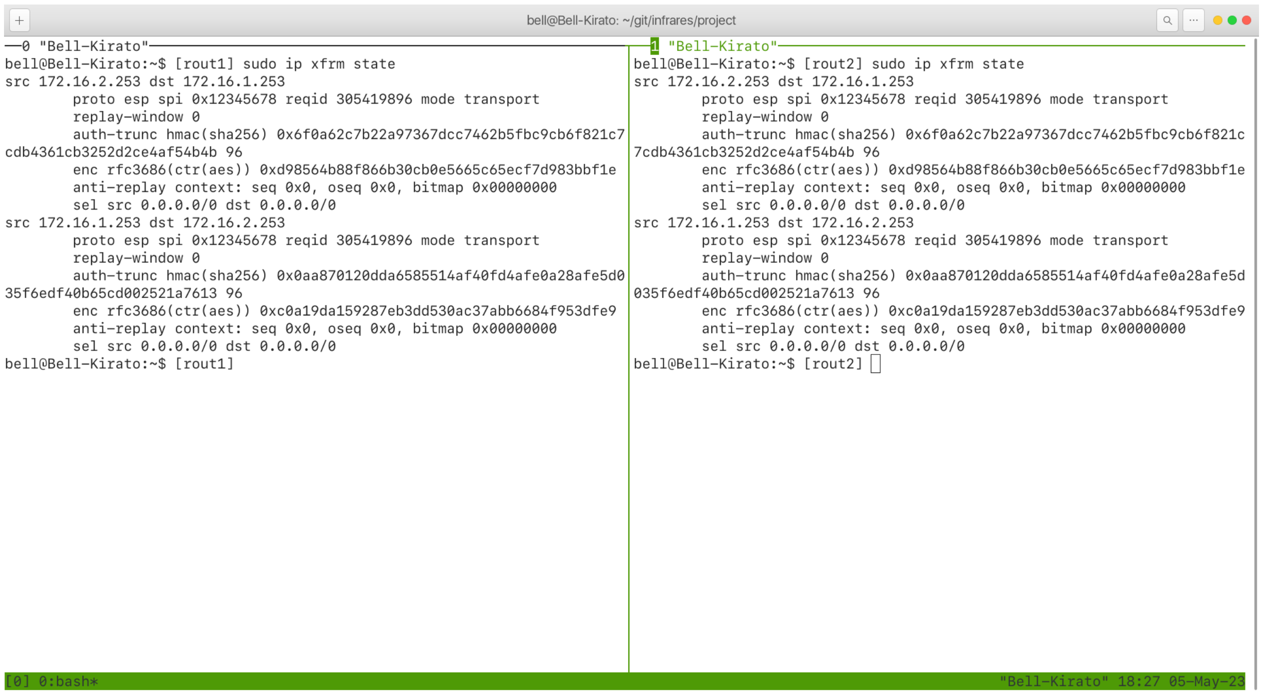
bell@Bell-Kirato: ~/git/infrares/project

```
—0 "Bell-Kirato"—
bell@Bell-Kirato:~$ [rout1] sudo ip xfrm policy
src 172.16.2.253/32 dst 172.16.1.253/32
        dir in priority 0
        tmpl src 172.16.2.253 dst 172.16.1.253
                proto esp reqid 305419896 mode transport
src 172.16.1.253/32 dst 172.16.2.253/32
        dir out priority 0
        tmpl src 172.16.1.253 dst 172.16.2.253
                proto esp reqid 305419896 mode transport
bell@Bell-Kirato:~$ [rout1]
```

```
1 "Bell-Kirato"—
bell@Bell-Kirato:~$ [rout2] sudo ip xfrm policy
src 172.16.2.253/32 dst 172.16.1.253/32
        dir out priority 0
        tmpl src 172.16.2.253 dst 172.16.1.253
                proto esp reqid 305419896 mode transport
src 172.16.1.253/32 dst 172.16.2.253/32
        dir in priority 0
        tmpl src 172.16.1.253 dst 172.16.2.253
                proto esp reqid 305419896 mode transport
bell@Bell-Kirato:~$ [rout2] 
```

```
[0] 0:bash*                                        "Bell-Kirato" 18:27 05-May-23
```

Figure 15: Verify IPsec policy on `rout1` and `rout2`



bell@Bell-Kirato: ~/git/infrares/project

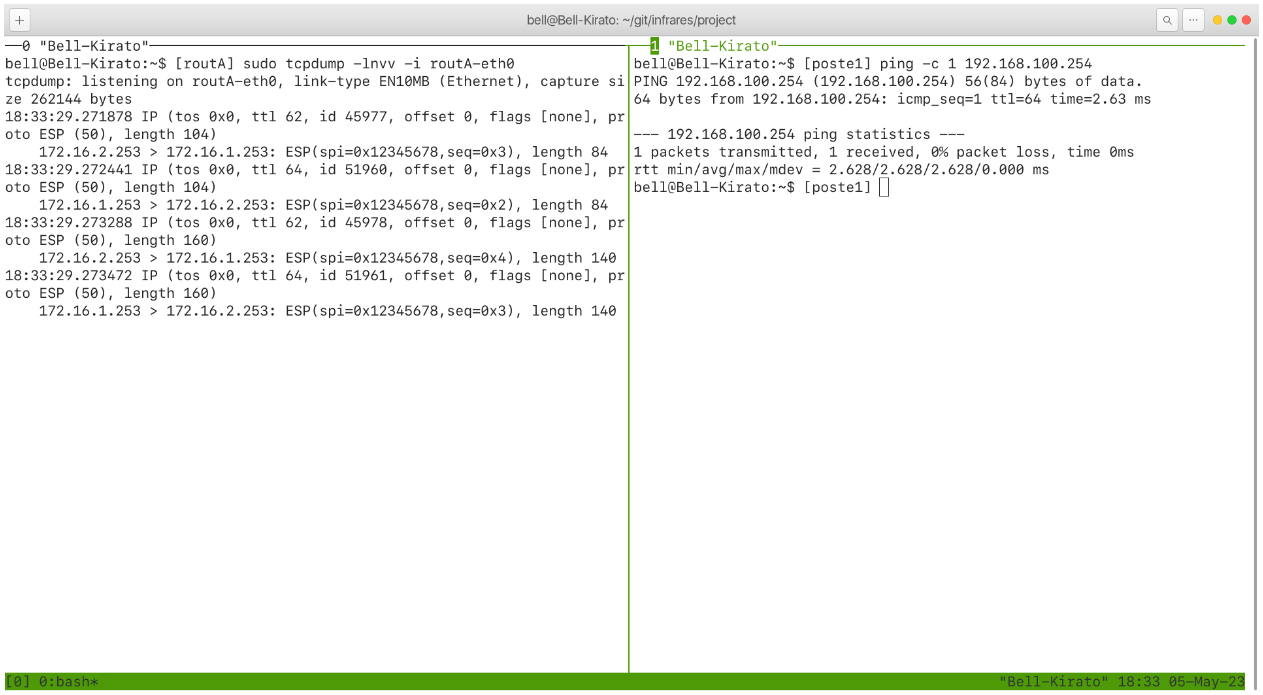
```
—0 "Bell-Kirato"—
bell@Bell-Kirato:~$ [rout1] sudo ip xfrm state
src 172.16.2.253 dst 172.16.1.253
        proto esp spi 0x12345678 reqid 305419896 mode transport
        replay-window 0
        auth-trunc hmac(sha256) 0x6f0a62c7b22a97367dcc7462b5fbc9cb6f821c7
cdb4361cb3252d2ce4af54b4b 96
        enc rfc3686(ctr(aes)) 0xd98564b88f866b30cb0e5665c65ecf7d983bbf1e
        anti-replay context: seq 0x0, oseq 0x0, bitmap 0x00000000
        sel src 0.0.0.0/0 dst 0.0.0.0/0
src 172.16.1.253 dst 172.16.2.253
        proto esp spi 0x12345678 reqid 305419896 mode transport
        replay-window 0
        auth-trunc hmac(sha256) 0x0aa870120dda6585514af40fd4afe0a28afe5d0
35f6edf40b65cd002521a7613 96
        enc rfc3686(ctr(aes)) 0xc0a19da159287eb3dd530ac37abb6684f953dfe9
        anti-replay context: seq 0x0, oseq 0x0, bitmap 0x00000000
        sel src 0.0.0.0/0 dst 0.0.0.0/0
bell@Bell-Kirato:~$ [rout1]
```

```
1 "Bell-Kirato"—
bell@Bell-Kirato:~$ [rout2] sudo ip xfrm state
src 172.16.2.253 dst 172.16.1.253
        proto esp spi 0x12345678 reqid 305419896 mode transport
        replay-window 0
        auth-trunc hmac(sha256) 0x6f0a62c7b22a97367dcc7462b5fbc9cb6f821c
7cdb4361cb3252d2ce4af54b4b 96
        enc rfc3686(ctr(aes)) 0xd98564b88f866b30cb0e5665c65ecf7d983bbf1e
        anti-replay context: seq 0x0, oseq 0x0, bitmap 0x00000000
        sel src 0.0.0.0/0 dst 0.0.0.0/0
src 172.16.1.253 dst 172.16.2.253
        proto esp spi 0x12345678 reqid 305419896 mode transport
        replay-window 0
        auth-trunc hmac(sha256) 0x0aa870120dda6585514af40fd4afe0a28afe5d
035f6edf40b65cd002521a7613 96
        enc rfc3686(ctr(aes)) 0xc0a19da159287eb3dd530ac37abb6684f953dfe9
        anti-replay context: seq 0x0, oseq 0x0, bitmap 0x00000000
        sel src 0.0.0.0/0 dst 0.0.0.0/0
bell@Bell-Kirato:~$ [rout2] 
```

```
[0] 0:bash*                                        "Bell-Kirato" 18:27 05-May-23
```

Figure 16: Verify IPsec state on `rout1` and `rout2`

After verifying that IPsec is configured on `rout1` and `rout2` appropriately, the packets are captured on `routA`'s interface `routA-eth0` with `tcpdump`. We will try to ping from `poste1` to `rout1` and inspect the traffic:



```
bell@Bell-Kirato: ~/git/infrares/project
```

```
─0 "Bell-Kirato"──────────────────────────────────────────
bell@Bell-Kirato:~$ [routA] sudo tcpdump -lnvv -i routA-eth0
tcpdump: listening on routA-eth0, link-type EN10MB (Ethernet), capture si
ze 262144 bytes
18:33:29.271878 IP (tos 0x0, ttl 62, id 45977, offset 0, flags [none], pr
oto ESP (50), length 104)
    172.16.2.253 > 172.16.1.253: ESP(spi=0x12345678,seq=0x3), length 84
18:33:29.272441 IP (tos 0x0, ttl 64, id 51960, offset 0, flags [none], pr
oto ESP (50), length 104)
    172.16.1.253 > 172.16.2.253: ESP(spi=0x12345678,seq=0x2), length 84
18:33:29.273288 IP (tos 0x0, ttl 62, id 45978, offset 0, flags [none], pr
oto ESP (50), length 160)
    172.16.2.253 > 172.16.1.253: ESP(spi=0x12345678,seq=0x4), length 140
18:33:29.273472 IP (tos 0x0, ttl 64, id 51961, offset 0, flags [none], pr
oto ESP (50), length 160)
    172.16.1.253 > 172.16.2.253: ESP(spi=0x12345678,seq=0x3), length 140
```

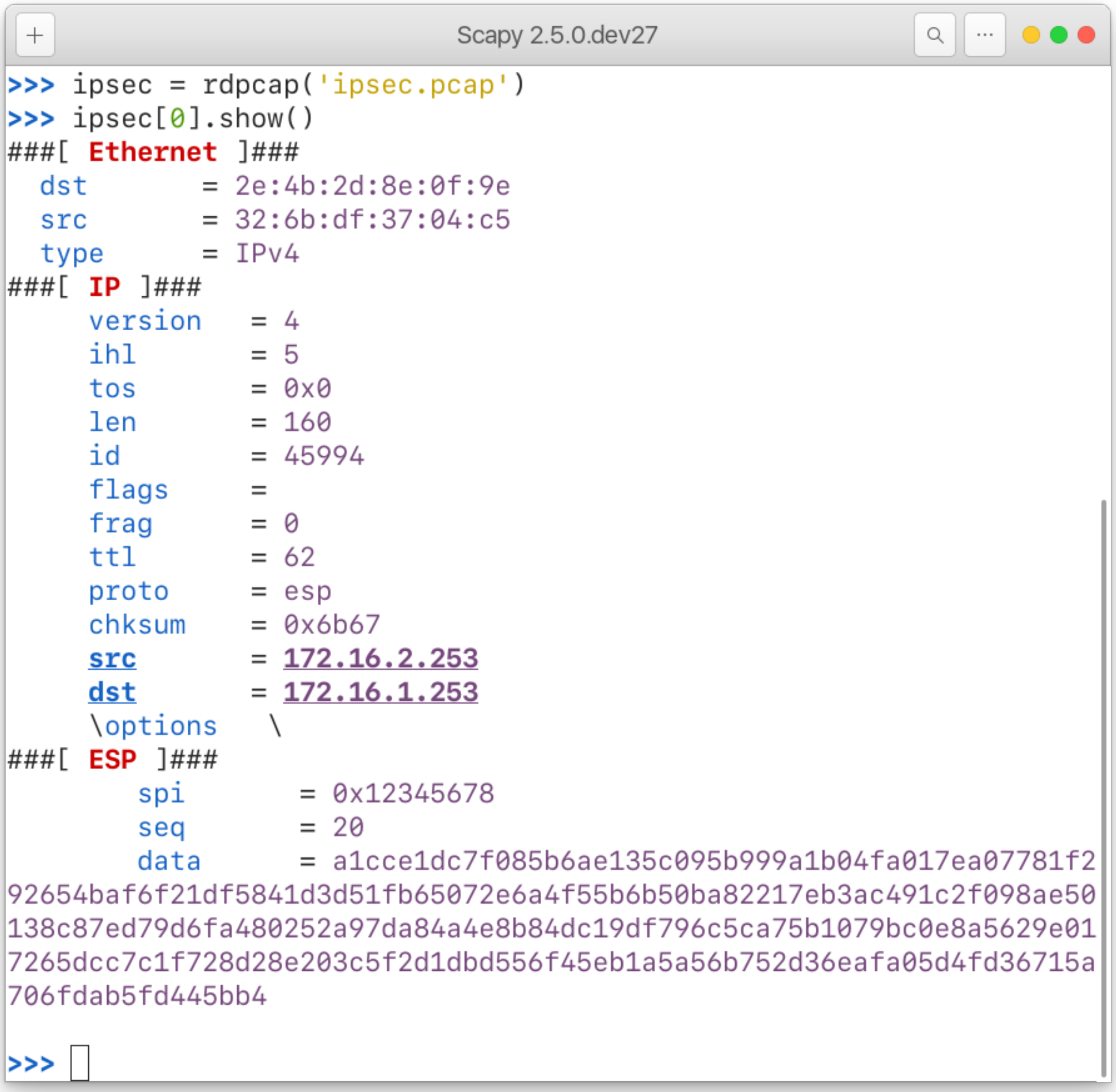
```
─1 "Bell-Kirato"──────────────────────────────────────────
bell@Bell-Kirato:~$ [poste1] ping -c 1 192.168.100.254
PING 192.168.100.254 (192.168.100.254) 56(84) bytes of data.
64 bytes from 192.168.100.254: icmp_seq=1 ttl=64 time=2.63 ms

--- 192.168.100.254 ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 2.628/2.628/2.628/0.000 ms
bell@Bell-Kirato:~$ [poste1]
```

```
[0] 0:bash*                                          "Bell-Kirato" 18:33 05-May-23
```

Figure 17: Inspect traffic going through `routA` after implementing IPsec on L2TPv3 tunnel

It can be seen from Figure 25 that the packets are now `ESP` packets, which were encrypted with ESP protocol, instead of `ip-proto-115` packets as we have seen before.

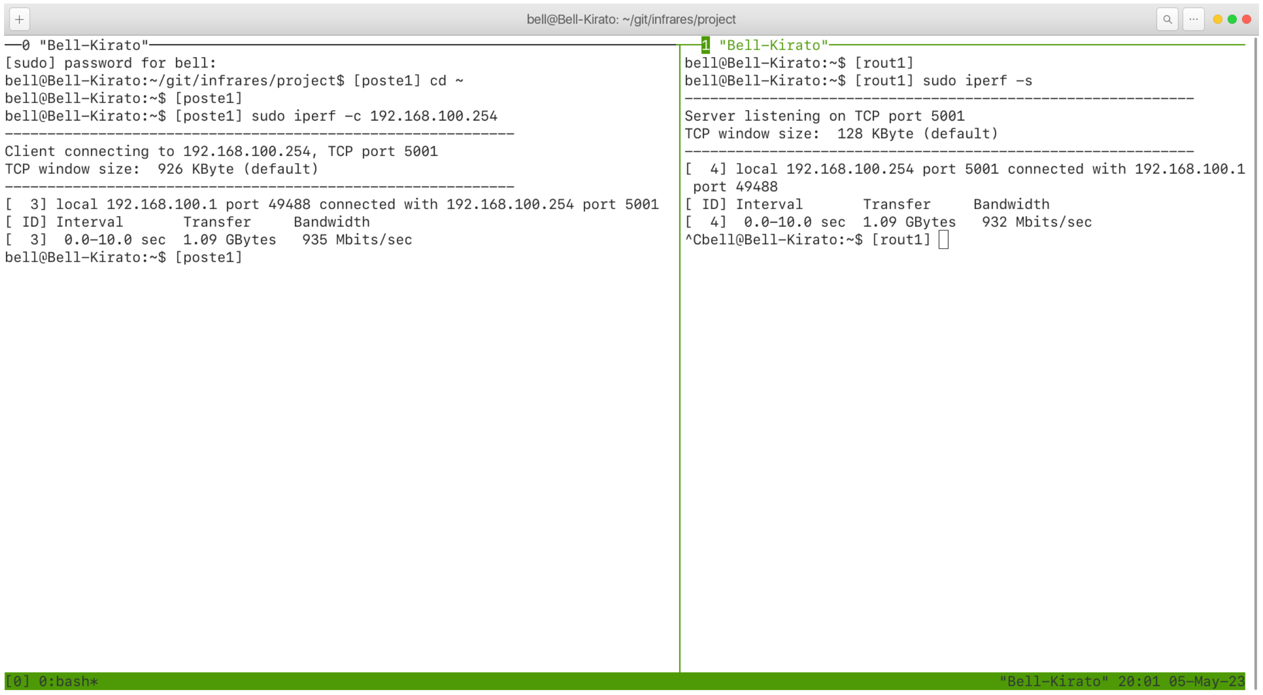
We cannot access the content of the encrypted packets captured on an intermediary (`routA` in this case) even if we try to use `scapy`, as the content of original packet was encrypted and can be seen from the field `data` of the `ESP` header:



```
>>> ipsec = rdpcap('ipsec.pcap')
>>> ipsec[0].show()
###[ Ethernet ]###
  dst       = 2e:4b:2d:8e:0f:9e
  src       = 32:6b:df:37:04:c5
  type      = IPv4
###[ IP ]###
     version   = 4
     ihl       = 5
     tos       = 0x0
     len       = 160
     id        = 45994
     flags     =
     frag      = 0
     ttl       = 62
     proto     = esp
     chksum    = 0x6b67
     src       = 172.16.2.253
     dst       = 172.16.1.253
     \options   \
###[ ESP ]###
        spi       = 0x12345678
        seq       = 20
        data      = a1cce1dc7f085b6ae135c095b999a1b04fa017ea07781f2
92654baf6f21df5841d3d51fb65072e6a4f55b6b50ba82217eb3ac491c2f098ae50
138c87ed79d6fa480252a97da84a4e8b84dc19df796c5ca75b1079bc0e8a5629e01
7265dcc7c1f728d28e203c5f2d1dbd556f45eb1a5a56b752d36eafa05d4fd36715a
706fdab5fd445bb4

>>>
```

Figure 18: Analyze the encrypted packet captured on an intermediary

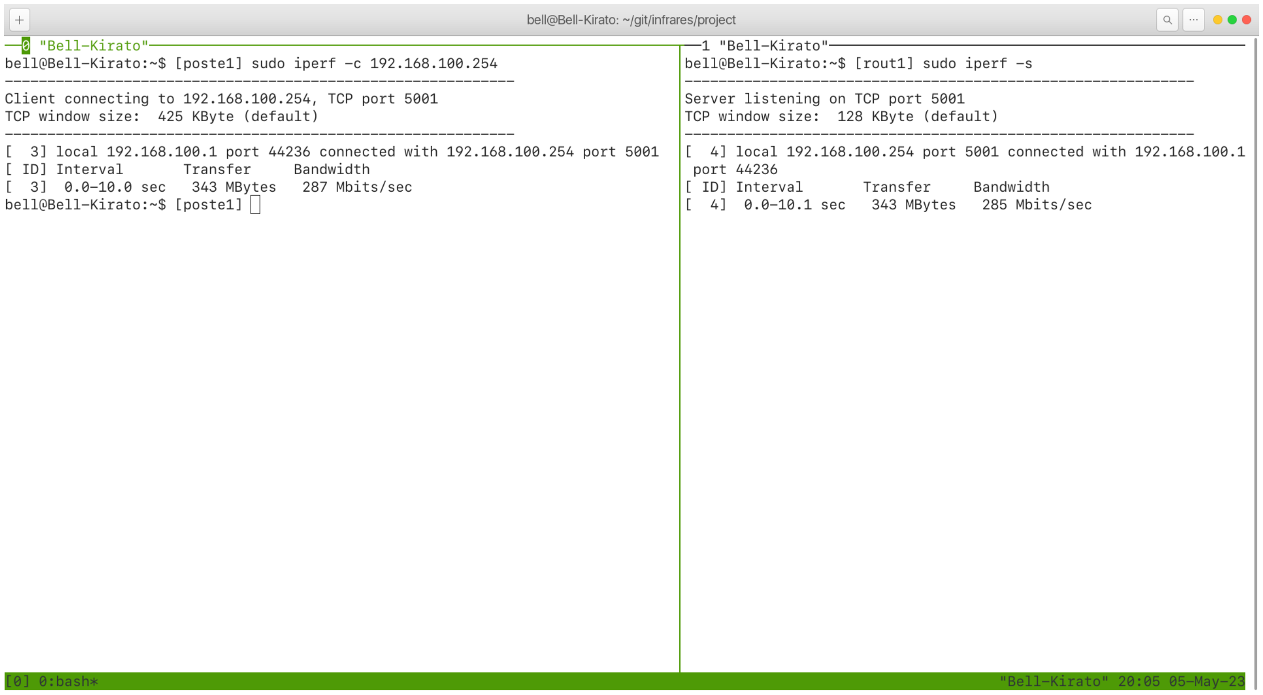
The bandwidth between `poste1` and `rout1` sharply decreased after applying IPsec, due to the encrypting process:



```
bell@Bell-Kirato: ~/git/infrares/project
─0 "Bell-Kirato"─────────────────────────────
[sudo] password for bell:
bell@Bell-Kirato:~/git/infrares/project$ [poste1] cd ~
bell@Bell-Kirato:~$ [poste1]
bell@Bell-Kirato:~$ [poste1] sudo iperf -c 192.168.100.254
------------------------------------------------------------
Client connecting to 192.168.100.254, TCP port 5001
TCP window size:  926 KByte (default)
------------------------------------------------------------
[  3] local 192.168.100.1 port 49488 connected with 192.168.100.254 port 5001
[ ID] Interval       Transfer     Bandwidth
[  3]  0.0-10.0 sec  1.09 GBytes   935 Mbits/sec
bell@Bell-Kirato:~$ [poste1]
```

```
─1 "Bell-Kirato"─────────────────────────────
bell@Bell-Kirato:~$ [rout1]
bell@Bell-Kirato:~$ [rout1] sudo iperf -s
------------------------------------------------------------
Server listening on TCP port 5001
TCP window size:  128 KByte (default)
------------------------------------------------------------
[  4] local 192.168.100.254 port 5001 connected with 192.168.100.1
 port 49488
[ ID] Interval       Transfer     Bandwidth
[  4]  0.0-10.0 sec  1.09 GBytes   932 Mbits/sec
^Cbell@Bell-Kirato:~$ [rout1]
[0] 0:bash*                                 "Bell-Kirato" 20:01 05-May-23
```

Figure 19: Transfer speed and bandwidth before implementing IPsec



```
bell@Bell-Kirato: ~/git/infrares/project
─0 "Bell-Kirato"─────────────────────────────
bell@Bell-Kirato:~$ [poste1] sudo iperf -c 192.168.100.254
------------------------------------------------------------
Client connecting to 192.168.100.254, TCP port 5001
TCP window size:  425 KByte (default)
------------------------------------------------------------
[  3] local 192.168.100.1 port 44236 connected with 192.168.100.254 port 5001
[ ID] Interval       Transfer     Bandwidth
[  3]  0.0-10.0 sec   343 MBytes   287 Mbits/sec
bell@Bell-Kirato:~$ [poste1]
```

```
─1 "Bell-Kirato"─────────────────────────────
bell@Bell-Kirato:~$ [rout1] sudo iperf -s
------------------------------------------------------------
Server listening on TCP port 5001
TCP window size:  128 KByte (default)
------------------------------------------------------------
[  4] local 192.168.100.254 port 5001 connected with 192.168.100.1
 port 44236
[ ID] Interval       Transfer     Bandwidth
[  4]  0.0-10.1 sec   343 MBytes   285 Mbits/sec
[0] 0:bash*                                 "Bell-Kirato" 20:05 05-May-23
```

Figure 20: Transfer speed and bandwidth after implementing IPsec

7 “Smart” Internet Access

7.1 Provide Internet access to the local network

To connect our virtual network to the real Internet, an interface `veth0` on the real Linux machine is created to connect itself to the switch `internet`, thus connect our local virtual network to the real Linux machine. We also need to assign an IP address to the interface `veth0`, which we choose to be `10.87.0.254`.

```
# Provide Internet access through the internet switch
ip l add veth0 type veth peer name internet-real
ovs-vsctl add-port internet internet-real
ip l set dev internet-real up
ip l set dev veth0 up
ip a add dev veth0 10.87.0.254/24
```

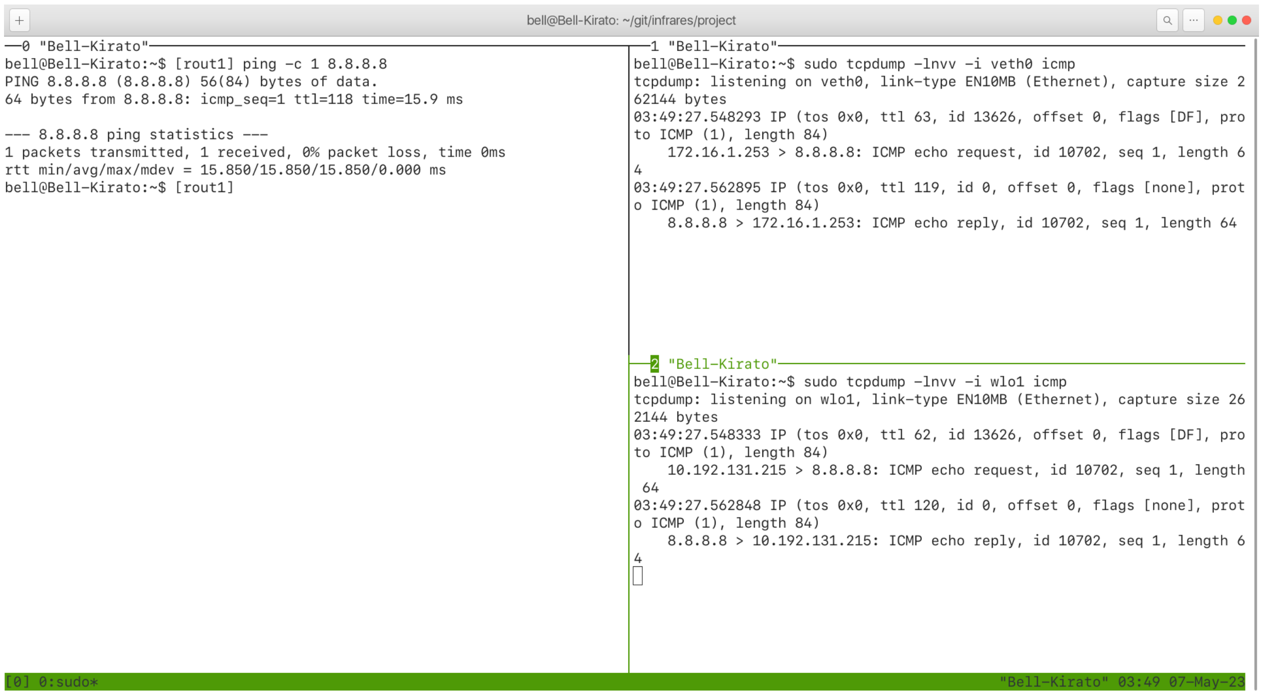
Then we modify the routing tables of the real Linux machine and the routers to enable the traffic between them, and we need to enable IPv4 forwarding on the Linux machine as well:

```
# Config the routing tables for the routers
ip netns exec routA ip r add default via 10.87.0.254 dev routA-eth1
ip netns exec rout1 ip r add default via 172.16.1.254 dev rout1-eth0
ip netns exec routB ip r add default via 10.87.0.254 dev routB-eth1
ip netns exec rout2 ip r add default via 172.16.2.254 dev rout2-eth0

# Config the routing table of the Linux machine
sysctl net.ipv4.conf.all.forwarding=1
ip r add 172.16.1.0/24 via 10.87.0.1 dev veth0
ip r add 172.16.2.0/24 via 10.87.0.2 dev veth0
```

Finally, we add the NetFilter rules to MASQUERADE all outgoing packets from the real Linux machine’s network interface `wlo1`, which is connected to the internet, since the local network is private, if MASQUERADE is not used, the Internet would not know how to return packets to IP addresses coming from it.

```
iptables -t nat -A POSTROUTING -o wlo1 -j MASQUERADE
```

Now, we can try to send a ping to the internet from the local network. Let's say we want to ping `8.8.8.8` from `rout1`:



```
bell@Bell-Kirato: ~/git/infrares/project

—0 "Bell-Kirato"—
bell@Bell-Kirato:~$ [rout1] ping -c 1 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=15.9 ms

--- 8.8.8.8 ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 15.850/15.850/15.850/0.000 ms
bell@Bell-Kirato:~$ [rout1]
```

```
—1 "Bell-Kirato"—
bell@Bell-Kirato:~$ sudo tcpdump -lnvv -i veth0 icmp
tcpdump: listening on veth0, link-type EN10MB (Ethernet), capture size 2
62144 bytes
03:49:27.548293 IP (tos 0x0, ttl 63, id 13626, offset 0, flags [DF], pro
to ICMP (1), length 84)
    172.16.1.253 > 8.8.8.8: ICMP echo request, id 10702, seq 1, length 6
4
03:49:27.562895 IP (tos 0x0, ttl 119, id 0, offset 0, flags [none], prot
o ICMP (1), length 84)
    8.8.8.8 > 172.16.1.253: ICMP echo reply, id 10702, seq 1, length 64
```

```
—2 "Bell-Kirato"—
bell@Bell-Kirato:~$ sudo tcpdump -lnvv -i wlo1 icmp
tcpdump: listening on wlo1, link-type EN10MB (Ethernet), capture size 26
2144 bytes
03:49:27.548333 IP (tos 0x0, ttl 62, id 13626, offset 0, flags [DF], pro
to ICMP (1), length 84)
    10.192.131.215 > 8.8.8.8: ICMP echo request, id 10702, seq 1, length
 64
03:49:27.562848 IP (tos 0x0, ttl 120, id 0, offset 0, flags [none], prot
o ICMP (1), length 84)
    8.8.8.8 > 10.192.131.215: ICMP echo reply, id 10702, seq 1, length 6
4
```

```
[0] 0:sudo*                                        "Bell-Kirato" 03:49 07-May-23
```

Figure 21: Ping from `rout1` to `8.8.8.8`

We can see that the traffic is well-transferred. The MASQUERADE also functions as expected, as we sniff on both interfaces `veth0` (connecting the real Linux machine to the local virtual network) and `wlo1` (connecting the real Linux machine to the Internet).

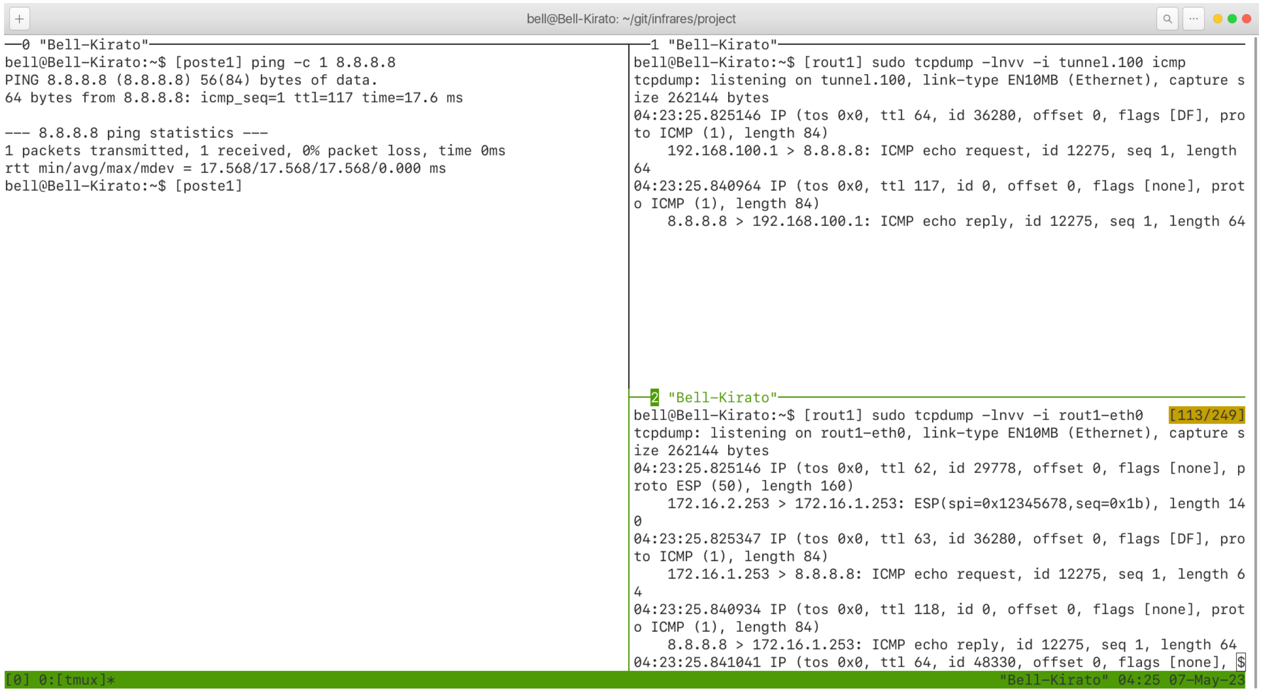
7.2 Provide Internet access to the hosts

To provide Internet to the hosts from both VLANs: `poste1`, `poste2`, `poste3`, and `poste4`, we need to not only MASQUERADE the packets coming out of our real Linux machine, but also those coming out of `rout1` and `rout2`, to go to the `internet` switch as well. The reason is the same as before: the Linux machine does not know how to route packets to our hosts yet, so either we add specific routes to the the routing table of the real Linux machine to route packets directly to those hosts, or we MASQUERADE those coming out of `rout1` and `rout2`. Here the second solution is chosen.

```
ip netns exec rout1 iptables -t nat -A POSTROUTING -o rout1-eth0 -j
   MASQUERADE
ip netns exec rout2 iptables -t nat -A POSTROUTING -o rout2-eth0 -j
   MASQUERADE
```

The second line is actually unnecessary at this point, since the traffic from `poste1` and `poste2` would first be redirected through the tunnel to `rout1`, then goes to the Internet. In brief, the right-hand side network will communicate with the Internet through `rout1`, so there is no need to MASQUERADE traffic coming out of interface `rout2-eth0` of `rout2` at this point. But it is still added as we are going to use it in the upcoming part of the exercise.

We can verify that the communication between `poste1` and the Internet is well-passed through `rout1`:



```
bell@Bell-Kirato: ~/git/infrares/project
```

```
─0 "Bell-Kirato"─
bell@Bell-Kirato:~$ [poste1] ping -c 1 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=117 time=17.6 ms

--- 8.8.8.8 ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 17.568/17.568/17.568/0.000 ms
bell@Bell-Kirato:~$ [poste1]
```

```
─1 "Bell-Kirato"─
bell@Bell-Kirato:~$ [rout1] sudo tcpdump -lnvv -i tunnel.100 icmp
tcpdump: listening on tunnel.100, link-type EN10MB (Ethernet), capture s
ize 262144 bytes
04:23:25.825146 IP (tos 0x0, ttl 64, id 36280, offset 0, flags [DF], pro
to ICMP (1), length 84)
    192.168.100.1 > 8.8.8.8: ICMP echo request, id 12275, seq 1, length
64
04:23:25.840964 IP (tos 0x0, ttl 117, id 0, offset 0, flags [none], prot
o ICMP (1), length 84)
    8.8.8.8 > 192.168.100.1: ICMP echo reply, id 12275, seq 1, length 64
```

```
─2 "Bell-Kirato"─
bell@Bell-Kirato:~$ [rout1] sudo tcpdump -lnvv -i rout1-eth0    [113/249]
tcpdump: listening on rout1-eth0, link-type EN10MB (Ethernet), capture s
ize 262144 bytes
04:23:25.825146 IP (tos 0x0, ttl 62, id 29778, offset 0, flags [none], p
roto ESP (50), length 160)
    172.16.2.253 > 172.16.1.253: ESP(spi=0x12345678,seq=0x1b), length 14
0
04:23:25.825347 IP (tos 0x0, ttl 63, id 36280, offset 0, flags [DF], pro
to ICMP (1), length 84)
    172.16.1.253 > 8.8.8.8: ICMP echo request, id 12275, seq 1, length 6
4
04:23:25.840934 IP (tos 0x0, ttl 118, id 0, offset 0, flags [none], prot
o ICMP (1), length 84)
    8.8.8.8 > 172.16.1.253: ICMP echo reply, id 12275, seq 1, length 64
04:23:25.841041 IP (tos 0x0, ttl 64, id 48330, offset 0, flags [none], $
[0] 0:[tmux]*                                   "Bell-Kirato" 04:25 07-May-23
```

Figure 22: Ping from `poste1` to `8.8.8.8`

7.3 Configure default gateway for the hosts

In this part, we have to provide the hosts with proper configuration using the DHCP server, so that the communication of `poste1` and `poste2` goes through `rout2` instead of `rout1` to optimize the Internet access.

First of all, this is where the MASQUERADE traffic from `rout2` becomes useful, as previously mentioned in Subsection 7.2, since if we are going to communicate to the Internet directly through `rout2`, it is necessary to MASQUERADE traffic coming out of `rout2-eth0`, for the same reason as in the case of `rout1`.

Now, to re-configure the hosts using DHCP protocol, we need to flush the DHCP cache memory on each of the hosts, and delete their default gateway in the routing table.

```
$ sudo ip netns exec poste1 dhclient -r -v
$ sudo ip netns exec poste1 dhclient -r poste1-eth0
$ sudo ip netns exec poste1 ip r del default
$ sudo ip netns exec poste2 dhclient -r -v
$ sudo ip netns exec poste2 dhclient -r poste2-eth0
$ sudo ip netns exec poste2 ip r del default
$ sudo ip netns exec poste3 dhclient -r -v
$ sudo ip netns exec poste3 dhclient -r poste3-eth0
$ sudo ip netns exec poste3 ip r del default
$ sudo ip netns exec poste4 dhclient -r -v
$ sudo ip netns exec poste4 dhclient -r poste4-eth0
$ sudo ip netns exec poste4 ip r del default
```

We update the configuration of the DHCP server that is set up on `rout1` by modifying the file `/etc/dnsmasq.conf`:

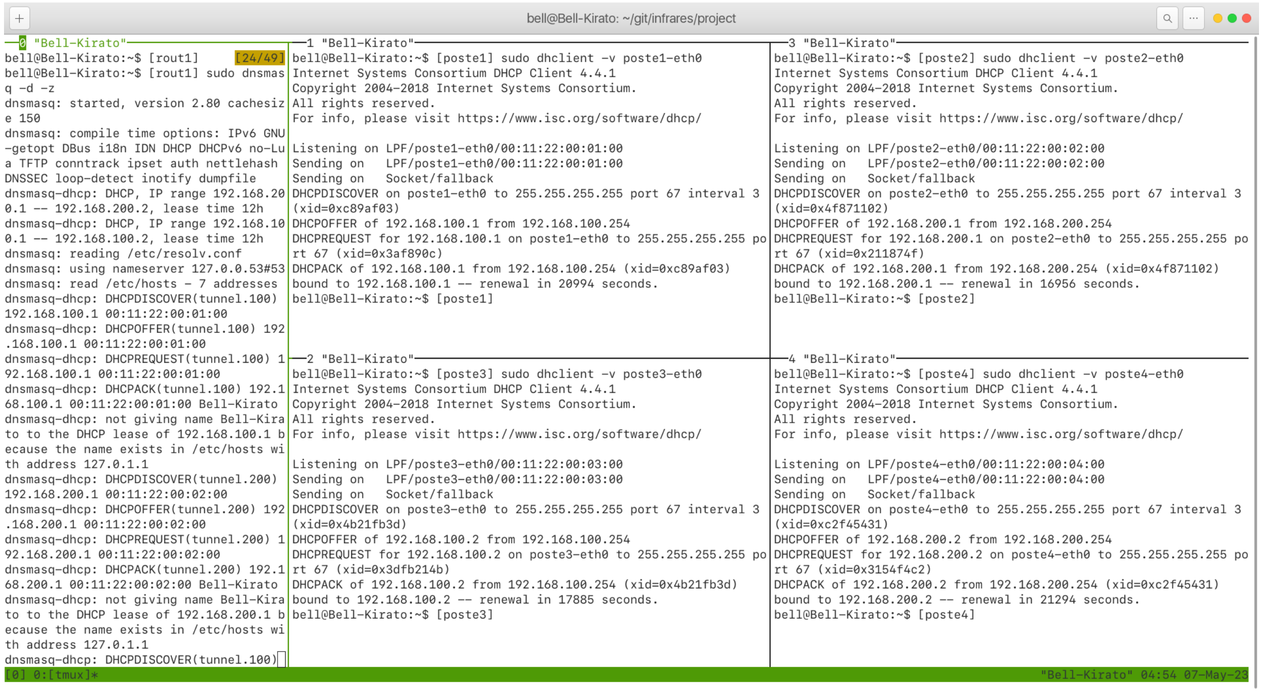
```
dhcp-range=192.168.100.1,192.168.100.2,12h
dhcp-range=192.168.200.1,192.168.200.2,12h
dhcp-option=tag:rout2.100,option:router,192.168.100.253
dhcp-option=tag:rout2.200,option:router,192.168.200.253
dhcp-option=tag:rout1.100,option:router,192.168.100.254
dhcp-option=tag:rout1.200,option:router,192.168.200.254
dhcp-host=00:11:22:00:01:00,set:rout2.100,192.168.100.1,12h
dhcp-host=00:11:22:00:02:00,set:rout2.200,192.168.200.1,12h
dhcp-host=00:11:22:00:03:00,set:rout1.100,192.168.100.2,12h
dhcp-host=00:11:22:00:04:00,set:rout1.200,192.168.200.2,12h
```

The `dhcp-option` and `set` (in `dhcp-host`) are used to specify the default gateway for the hosts. Then, again, on `rout1`, we start the DHCP server:

```
$ [rout1] sudo dnsmasq -d -z
```

and on each host, we run the `dhclient`:

```
$ [<poste-name>] sudo dhclient -v <poste-name>-eth0
```



```
0 "Bell-Kirato"
bell@Bell-Kirato:~$ [rout1]      [24/49]
bell@Bell-Kirato:~$ [rout1] sudo dnsmas
q -d -z
dnsmasq: started, version 2.80 cachesiz
e 150
dnsmasq: compile time options: IPv6 GNU
-getopt DBus i18n IDN DHCP DHCPv6 no-Lu
a TFTP conntrack ipset auth nettlehash
DNSSEC loop-detect inotify dumpfile
dnsmasq-dhcp: DHCP, IP range 192.168.20
0.1 -- 192.168.200.2, lease time 12h
dnsmasq-dhcp: DHCP, IP range 192.168.10
0.1 -- 192.168.100.2, lease time 12h
dnsmasq: reading /etc/resolv.conf
dnsmasq: using nameserver 127.0.0.53#53
dnsmasq: read /etc/hosts - 7 addresses
dnsmasq-dhcp: DHCPDISCOVER(tunnel.100)
192.168.100.1 00:11:22:00:01:00
dnsmasq-dhcp: DHCPOFFER(tunnel.100) 192
.168.100.1 00:11:22:00:01:00
dnsmasq-dhcp: DHCPREQUEST(tunnel.100) 1
92.168.100.1 00:11:22:00:01:00
dnsmasq-dhcp: DHCPACK(tunnel.100) 192.1
68.100.1 00:11:22:00:01:00 Bell-Kirato
dnsmasq-dhcp: not giving name Bell-Kira
to to the DHCP lease of 192.168.100.1 b
ecause the name exists in /etc/hosts wi
th address 127.0.1.1
dnsmasq-dhcp: DHCPDISCOVER(tunnel.200)
192.168.200.1 00:11:22:00:02:00
dnsmasq-dhcp: DHCPOFFER(tunnel.200) 192
.168.200.1 00:11:22:00:02:00
dnsmasq-dhcp: DHCPREQUEST(tunnel.200) 1
92.168.200.1 00:11:22:00:02:00
dnsmasq-dhcp: DHCPACK(tunnel.200) 192.1
68.200.1 00:11:22:00:02:00 Bell-Kirato
dnsmasq-dhcp: not giving name Bell-Kira
to to the DHCP lease of 192.168.200.1 b
ecause the name exists in /etc/hosts wi
th address 127.0.1.1
dnsmasq-dhcp: DHCPDISCOVER(tunnel.100)
```

```
1 "Bell-Kirato"
bell@Bell-Kirato:~$ [poste1] sudo dhclient -v poste1-eth0
Internet Systems Consortium DHCP Client 4.4.1
Copyright 2004-2018 Internet Systems Consortium.
All rights reserved.
For info, please visit https://www.isc.org/software/dhcp/

Listening on LPF/poste1-eth0/00:11:22:00:01:00
Sending on   LPF/poste1-eth0/00:11:22:00:01:00
Sending on   Socket/fallback
DHCPDISCOVER on poste1-eth0 to 255.255.255.255 port 67 interval 3
(xid=0xc89af03)
DHCPOFFER of 192.168.100.1 from 192.168.100.254
DHCPREQUEST for 192.168.100.1 on poste1-eth0 to 255.255.255.255 po
rt 67 (xid=0x3af890c)
DHCPACK of 192.168.100.1 from 192.168.100.254 (xid=0xc89af03)
bound to 192.168.100.1 -- renewal in 20994 seconds.
bell@Bell-Kirato:~$ [poste1]
```

```
3 "Bell-Kirato"
bell@Bell-Kirato:~$ [poste2] sudo dhclient -v poste2-eth0
Internet Systems Consortium DHCP Client 4.4.1
Copyright 2004-2018 Internet Systems Consortium.
All rights reserved.
For info, please visit https://www.isc.org/software/dhcp/

Listening on LPF/poste2-eth0/00:11:22:00:02:00
Sending on   LPF/poste2-eth0/00:11:22:00:02:00
Sending on   Socket/fallback
DHCPDISCOVER on poste2-eth0 to 255.255.255.255 port 67 interval 3
(xid=0x4f871102)
DHCPOFFER of 192.168.200.1 from 192.168.200.254
DHCPREQUEST for 192.168.200.1 on poste2-eth0 to 255.255.255.255 po
rt 67 (xid=0x211874f)
DHCPACK of 192.168.200.1 from 192.168.200.254 (xid=0x4f871102)
bound to 192.168.200.1 -- renewal in 16956 seconds.
bell@Bell-Kirato:~$ [poste2]
```

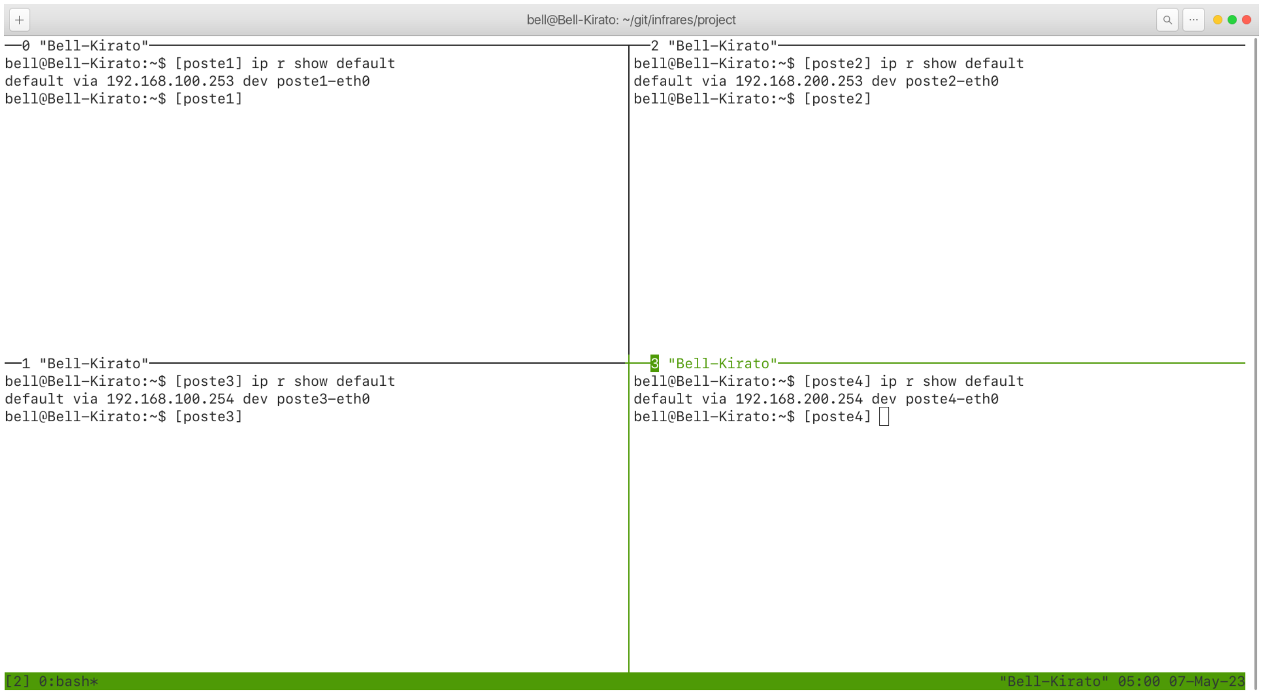
```
2 "Bell-Kirato"
bell@Bell-Kirato:~$ [poste3] sudo dhclient -v poste3-eth0
Internet Systems Consortium DHCP Client 4.4.1
Copyright 2004-2018 Internet Systems Consortium.
All rights reserved.
For info, please visit https://www.isc.org/software/dhcp/

Listening on LPF/poste3-eth0/00:11:22:00:03:00
Sending on   LPF/poste3-eth0/00:11:22:00:03:00
Sending on   Socket/fallback
DHCPDISCOVER on poste3-eth0 to 255.255.255.255 port 67 interval 3
(xid=0x4b21fb3d)
DHCPOFFER of 192.168.100.2 from 192.168.100.254
DHCPREQUEST for 192.168.100.2 on poste3-eth0 to 255.255.255.255 po
rt 67 (xid=0x3dfb214b)
DHCPACK of 192.168.100.2 from 192.168.100.254 (xid=0x4b21fb3d)
bound to 192.168.100.2 -- renewal in 17885 seconds.
bell@Bell-Kirato:~$ [poste3]
```

```
4 "Bell-Kirato"
bell@Bell-Kirato:~$ [poste4] sudo dhclient -v poste4-eth0
Internet Systems Consortium DHCP Client 4.4.1
Copyright 2004-2018 Internet Systems Consortium.
All rights reserved.
For info, please visit https://www.isc.org/software/dhcp/

Listening on LPF/poste4-eth0/00:11:22:00:04:00
Sending on   LPF/poste4-eth0/00:11:22:00:04:00
Sending on   Socket/fallback
DHCPDISCOVER on poste4-eth0 to 255.255.255.255 port 67 interval 3
(xid=0xc2f45431)
DHCPOFFER of 192.168.200.2 from 192.168.200.254
DHCPREQUEST for 192.168.200.2 on poste4-eth0 to 255.255.255.255 po
rt 67 (xid=0x3154f4c2)
DHCPACK of 192.168.200.2 from 192.168.200.254 (xid=0xc2f45431)
bound to 192.168.200.2 -- renewal in 21294 seconds.
bell@Bell-Kirato:~$ [poste4]
```

```
[0] 0:[tmux]*                                          "Bell-Kirato" 04:54 07-May-23
```

Figure 23: The DHCP Protocol captured on `rout1` and `poste1`, `poste2`, `poste3`, and `poste4` (with default gateway config by DHCP)

We can verify again that the default route on each of the hosts has changed:



```
bell@Bell-Kirato: ~/git/infrares/project
```

```
bell@Bell-Kirato:~$ [poste1] ip r show default
default via 192.168.100.253 dev poste1-eth0
bell@Bell-Kirato:~$ [poste1]
```

```
bell@Bell-Kirato:~$ [poste2] ip r show default
default via 192.168.200.253 dev poste2-eth0
bell@Bell-Kirato:~$ [poste2]
```

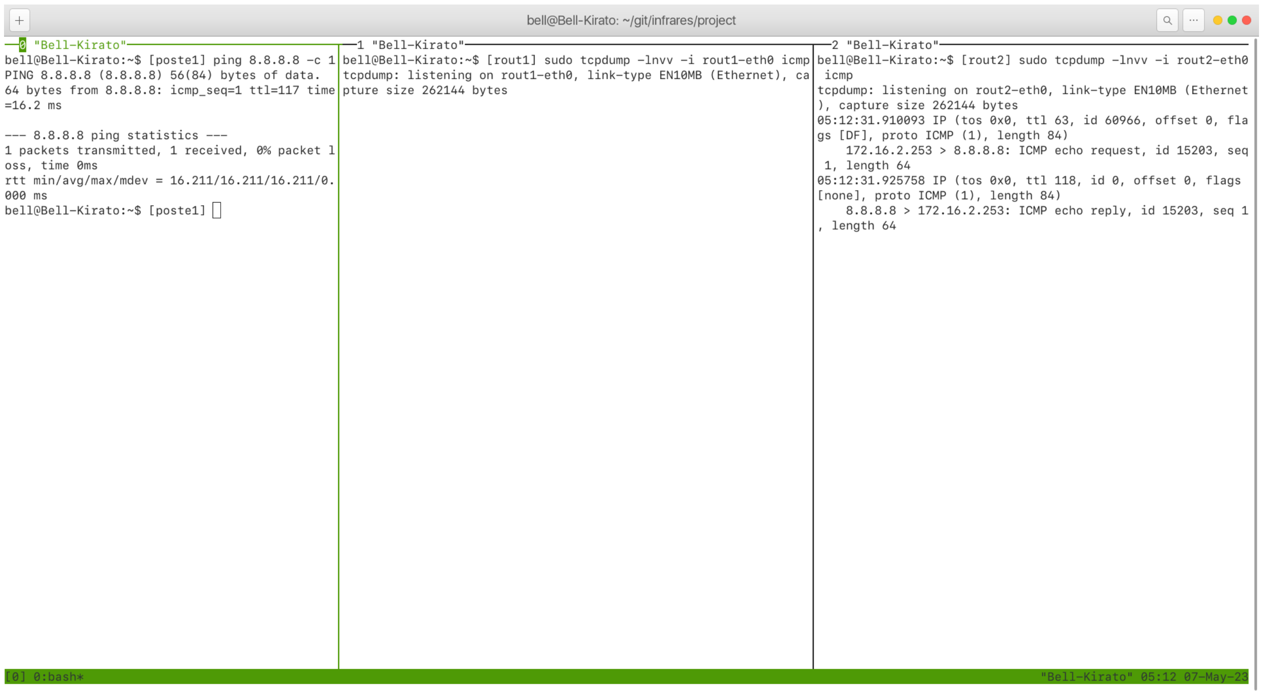
```
bell@Bell-Kirato:~$ [poste3] ip r show default
default via 192.168.100.254 dev poste3-eth0
bell@Bell-Kirato:~$ [poste3]
```

```
bell@Bell-Kirato:~$ [poste4] ip r show default
default via 192.168.200.254 dev poste4-eth0
bell@Bell-Kirato:~$ [poste4]
```

```
[2] 0:bash*                                    "Bell-Kirato" 05:00 07-May-23
```

Figure 24: Modified default route on `poste1`, `poste2`, `poste3`, and `poste4`

And we verify that the traffic from `poste1` to the Internet and back pass through `rout2` instead of `rout1`:



```
bell@Bell-Kirato: ~/git/infrares/project
```

```
bell@Bell-Kirato:~$ [poste1] ping 8.8.8.8 -c 1
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=117 time
=16.2 ms

--- 8.8.8.8 ping statistics ---
1 packets transmitted, 1 received, 0% packet l
oss, time 0ms
rtt min/avg/max/mdev = 16.211/16.211/16.211/0.
000 ms
bell@Bell-Kirato:~$ [poste1]
```

```
bell@Bell-Kirato:~$ [rout1] sudo tcpdump -lnvv -i rout1-eth0 icmp
tcpdump: listening on rout1-eth0, link-type EN10MB (Ethernet), ca
pture size 262144 bytes
```

```
bell@Bell-Kirato:~$ [rout2] sudo tcpdump -lnvv -i rout2-eth0
 icmp
tcpdump: listening on rout2-eth0, link-type EN10MB (Ethernet
), capture size 262144 bytes
05:12:31.910093 IP (tos 0x0, ttl 63, id 60966, offset 0, fla
gs [DF], proto ICMP (1), length 84)
    172.16.2.253 > 8.8.8.8: ICMP echo request, id 15203, seq
 1, length 64
05:12:31.925758 IP (tos 0x0, ttl 118, id 0, offset 0, flags
[none], proto ICMP (1), length 84)
    8.8.8.8 > 172.16.2.253: ICMP echo reply, id 15203, seq 1
, length 64
```

```
[0] 0:bash*                                    "Bell-Kirato" 05:12 07-May-23
```

Figure 25: Communication between `poste1` and the Internet pass through `rout2`

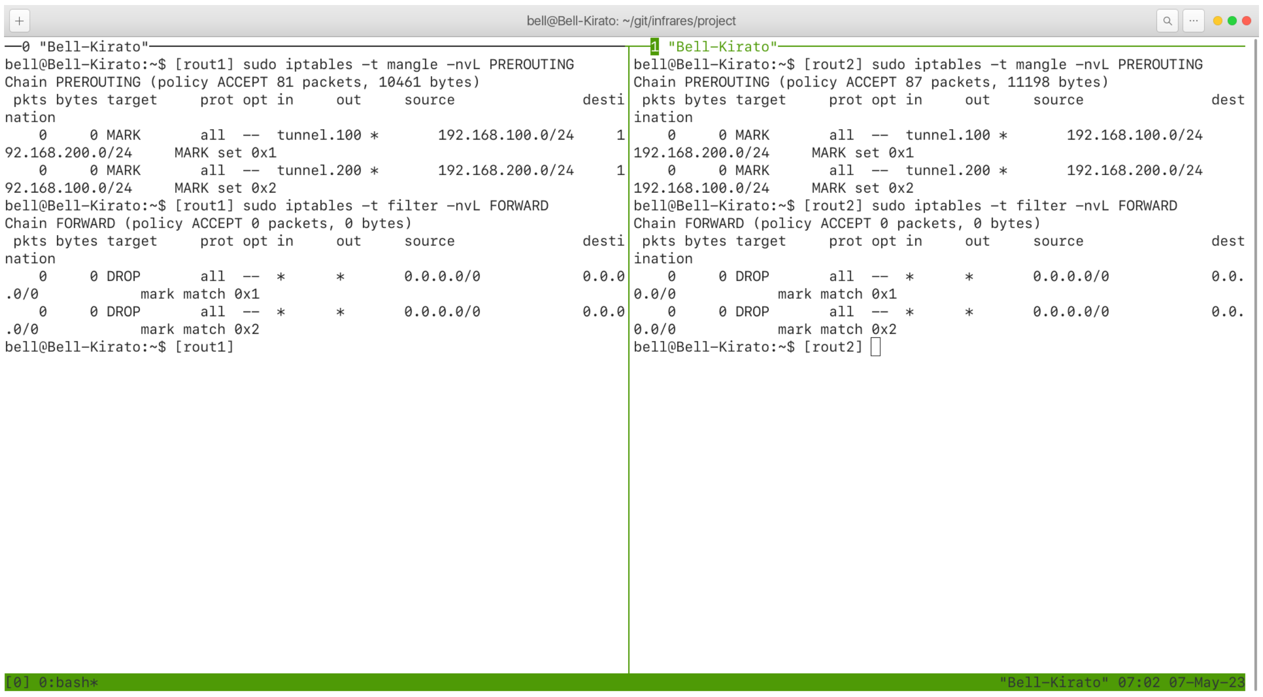
8 Blocking traffic between VLANs

Theoretically, traffics between different VLANs should be isolated (which is exactly the purpose of using VLAN), so we would want to block the traffic between them. We can do this either by using NetFilter or by using Routing Policy.

8.1 Using `iptables` rules

To isolate the traffic between VLAN100 and VLAN200, we would simply want to filter out (drop) any packets having source IP address from one VLAN and destination IP address from the other. These packets, indeed, should already have been decrypted, decapsulated from any kind of encapsulation, etc. Therefore, the configuration on `rout1` and `rout2` should look like this:

```
ip netns exec rout1 iptables -t mangle -A PREROUTING -i tunnel.100 -s
    192.168.100.0/24 -d 192.168.200.0/24 -j MARK --set-mark 1
ip netns exec rout1 iptables -t mangle -A PREROUTING -i tunnel.200 -s
    192.168.200.0/24 -d 192.168.100.0/24 -j MARK --set-mark 2
ip netns exec rout1 iptables -t filter -A FORWARD -m mark --mark 1 -j DROP
ip netns exec rout1 iptables -t filter -A FORWARD -m mark --mark 2 -j DROP

ip netns exec rout2 iptables -t mangle -A PREROUTING -i tunnel.100 -s
    192.168.100.0/24 -d 192.168.200.0/24 -j MARK --set-mark 1
ip netns exec rout2 iptables -t mangle -A PREROUTING -i tunnel.200 -s
    192.168.200.0/24 -d 192.168.100.0/24 -j MARK --set-mark 2
ip netns exec rout2 iptables -t filter -A FORWARD -m mark --mark 1 -j DROP
ip netns exec rout2 iptables -t filter -A FORWARD -m mark --mark 2 -j DROP
```



```
bell@Bell-Kirato:~$ [rout1] sudo iptables -t mangle -nvL PREROUTING
Chain PREROUTING (policy ACCEPT 81 packets, 10461 bytes)
 pkts bytes target     prot opt in     out     source               desti
nation
    0     0 MARK       all  --  tunnel.100 *       192.168.100.0/24     1
92.168.200.0/24     MARK set 0x1
    0     0 MARK       all  --  tunnel.200 *       192.168.200.0/24     1
92.168.100.0/24     MARK set 0x2
bell@Bell-Kirato:~$ [rout1] sudo iptables -t filter -nvL FORWARD
Chain FORWARD (policy ACCEPT 0 packets, 0 bytes)
 pkts bytes target     prot opt in     out     source               desti
nation
    0     0 DROP       all  --  *      *       0.0.0.0/0            0.0.0
.0/0            mark match 0x1
    0     0 DROP       all  --  *      *       0.0.0.0/0            0.0.0
.0/0            mark match 0x2
bell@Bell-Kirato:~$ [rout1]
```

```
bell@Bell-Kirato:~$ [rout2] sudo iptables -t mangle -nvL PREROUTING
Chain PREROUTING (policy ACCEPT 87 packets, 11198 bytes)
 pkts bytes target     prot opt in     out     source               dest
ination
    0     0 MARK       all  --  tunnel.100 *       192.168.100.0/24
192.168.200.0/24     MARK set 0x1
    0     0 MARK       all  --  tunnel.200 *       192.168.200.0/24
192.168.100.0/24     MARK set 0x2
bell@Bell-Kirato:~$ [rout2] sudo iptables -t filter -nvL FORWARD
Chain FORWARD (policy ACCEPT 0 packets, 0 bytes)
 pkts bytes target     prot opt in     out     source               dest
ination
    0     0 DROP       all  --  *      *       0.0.0.0/0            0.0.
0.0/0            mark match 0x1
    0     0 DROP       all  --  *      *       0.0.0.0/0            0.0.
0.0/0            mark match 0x2
bell@Bell-Kirato:~$ [rout2]
```

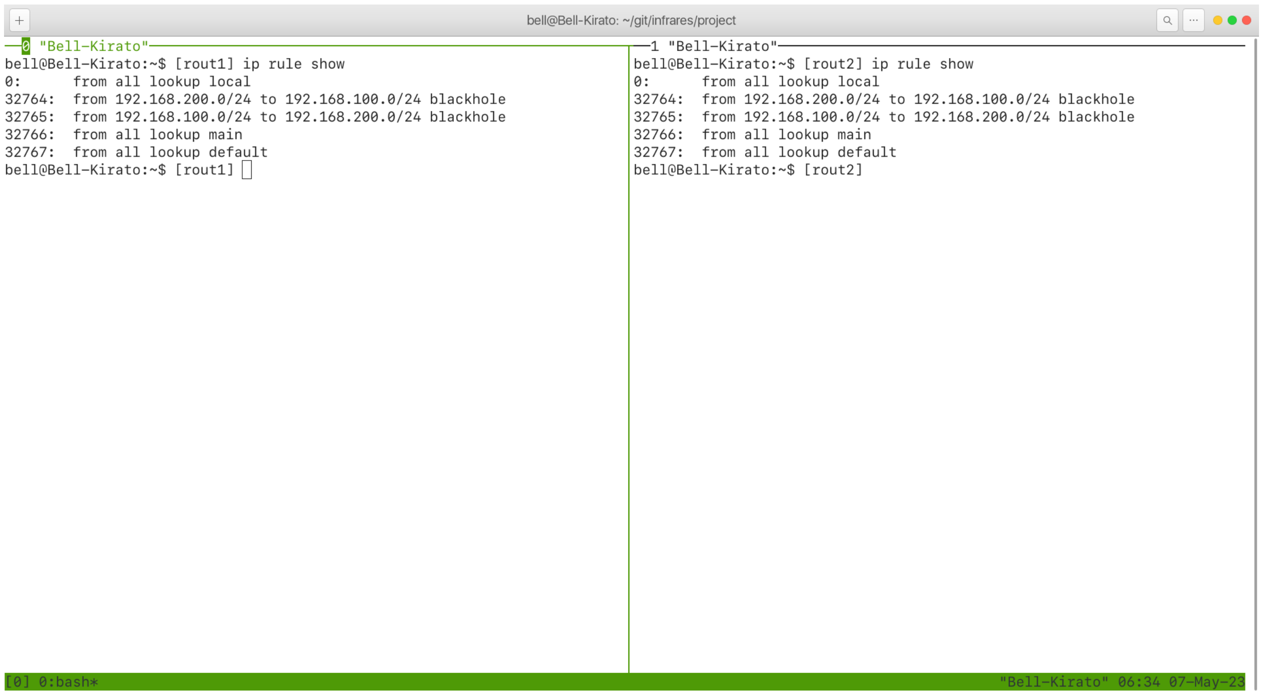
bell@Bell-Kirato: ~/git/infrares/project
0 "Bell-Kirato"
1 "Bell-Kirato"
[0] 0:bash*
"Bell-Kirato" 07:02 07-May-23

Figure 26: NetFilter rules on `rout1` and `rout2`

8.2 Using Routing Policy

In this case, we will take advantage of the `blackhole` route in Routing Policy. So instead of dropping packets that are from one VLAN to another as in `iptables`, we create Routing Policies that put them in the `blackhole` route, using `ip rule`:

```
ip netns exec rout1 ip rule add from 192.168.100.0/24 to 192.168.200.0/24
    blackhole
ip netns exec rout1 ip rule add from 192.168.200.0/24 to 192.168.100.0/24
    blackhole
ip netns exec rout2 ip rule add from 192.168.100.0/24 to 192.168.200.0/24
    blackhole
ip netns exec rout2 ip rule add from 192.168.200.0/24 to 192.168.100.0/24
    blackhole
```



```
bell@Bell-Kirato: ~/git/infrares/project
```

```
0 "Bell-Kirato"
bell@Bell-Kirato:~$ [rout1] ip rule show
0:      from all lookup local
32764:  from 192.168.200.0/24 to 192.168.100.0/24 blackhole
32765:  from 192.168.100.0/24 to 192.168.200.0/24 blackhole
32766:  from all lookup main
32767:  from all lookup default
bell@Bell-Kirato:~$ [rout1]
```

```
1 "Bell-Kirato"
bell@Bell-Kirato:~$ [rout2] ip rule show
0:      from all lookup local
32764:  from 192.168.200.0/24 to 192.168.100.0/24 blackhole
32765:  from 192.168.100.0/24 to 192.168.200.0/24 blackhole
32766:  from all lookup main
32767:  from all lookup default
bell@Bell-Kirato:~$ [rout2]
```

```
[0] 0:bash*                                   "Bell-Kirato" 06:34 07-May-23
```

Figure 27: Routing Policies on `rout1` and `rout2`