UNIVERSIDAD NACIONAL DEL COMAHUE

Centro Regional Zona Atlántica

Tecnicatura Universitaria en Administración de Sistemas y Software Libre

Redes de datos

Laboratorio Nro: 4 – Enrutamiento dinámico y NAT en GNU/Linux

Alumno : Maidana Maximiliano Miguel

Objetivo

El objetivo del laboratorio consiste en experimentar sobre el simulador GNS3 enrutamiento dinámico en un sistema autónomo, para alcanzar determinados servicios distantes del host que los solicita.

Entrega del Informe de Laboratorio

Se debe subir a la plataforma un archivo comprimido (.tar.gz) conteniendo un documento donde se expliquen los resultados obtenidos[1].
Para individualizar los trabajos, cada captura de pantalla de GNS3 deberá incluir un cartel con el nombre completo del/la alumno/a creado en el propio programa (ver opción “add a note”).

Recursos

Para la realización del trabajo sobre el simulador, serán necesarios:

- 3 Routers GNU/Linux con Quagga instalado desde la imagen *ajnouri/quagga_alpine*.
- 1 Servidor *Web* a partir de la imagen oficial de *nginx*[2] en *dockerhub*.
- 2 clientes *alpine* con cliente *curl* (un navegador web en modo texto) y *nmap*.

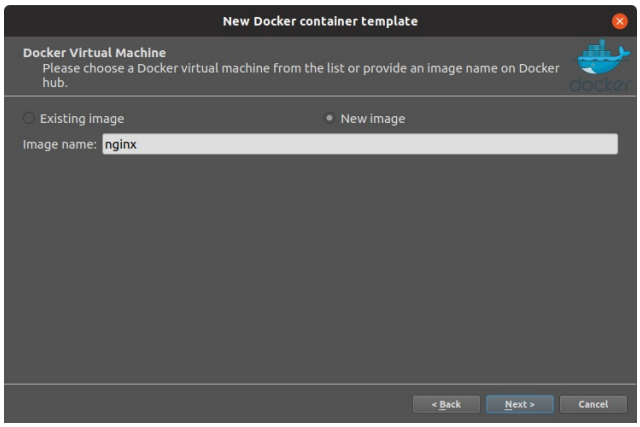
Preparación del laboratorio

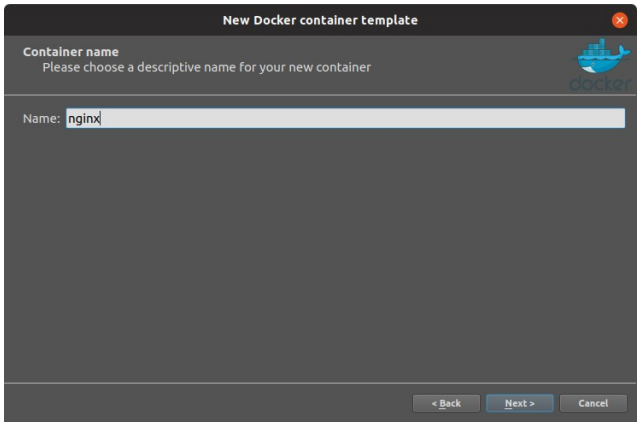
La obtención de la imagen *ajnouri/quagga_alpine* se explicó detalladamente en el material teórico.

De la misma forma se procede con las imágenes de los servidores necesarios desde *dockerhub.*

1 Identificando en una carátula el nombre del estudiante y el número de laboratorio.
2 Nginx es servidor web ligero

Para el servidor Web:





Para los clientes *alpine* con navegador y nmap no existe una imagen oficial disponible por lo cual hay que crearlo a partir de un archivo *Dockerfile* de la siguiente manera:

En el host anfitrión del simulador crear un directorio nuevo, por ejemplo uno llamado *docker*:

```
ubuntu@curza:~$ mkdir docker
```

Se cambia a ese directorio y se crea un archivo vacío llamado *Dockerfile* (tener en cuenta la mayúscula):

```
ubuntu@curza:~$ cd docker
ubuntu@curza:~/docker$ touch Dockerfile
```

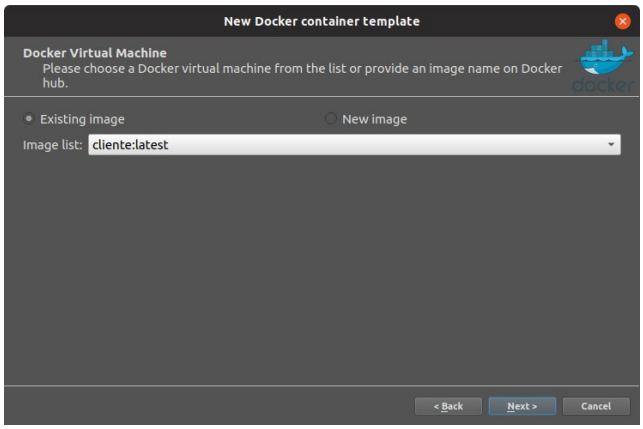
Con un editor de textos, editar el archivo *Dockerfile* e incorporar el siguiente contenido (textual):

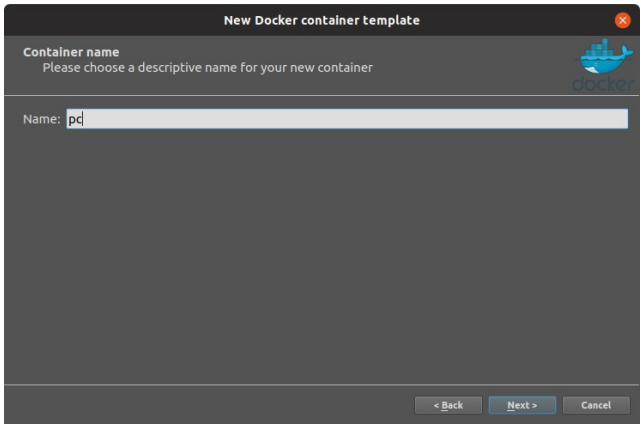
```
FROM alpine:latest
LABEL Alpine, curl y nmap
RUN apk add --update curl nmap && rm -rf /var/cache/apk/*
```

Una vez guardado el archivo y sin cambiar de directorio, desde la línea de comandos ejecutar (no omitir el punto final):

```
ubuntu@curza:~/docker$ docker build -t cliente .
```

Ahora existe una imagen de nombre *cliente* que puede usarse como en los casos anteriores:



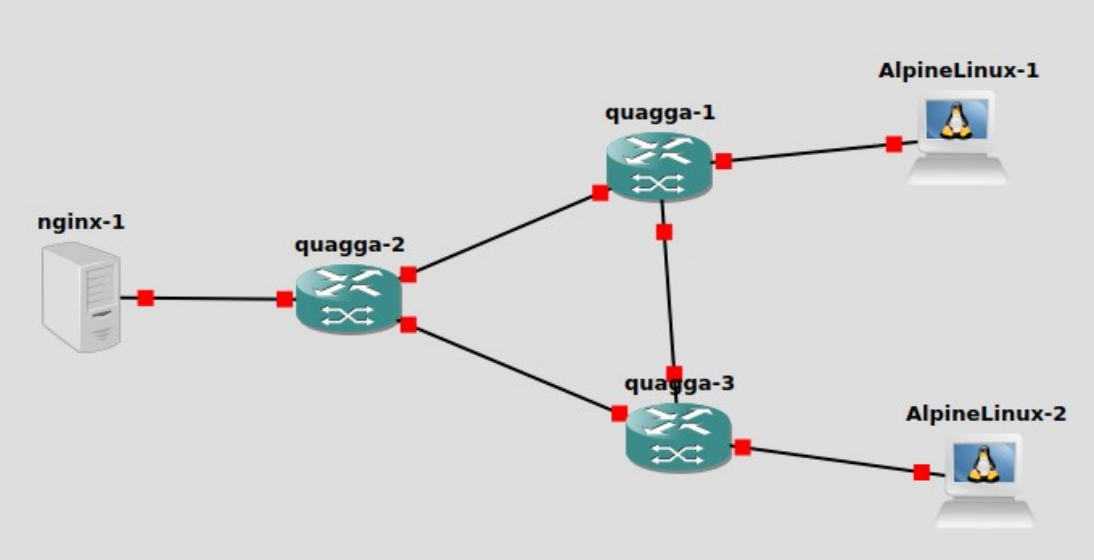


Actividades

1. Mediante el simulador GNS3 armar una red como la que se detalla a continuación. Hacer el subnetting con redes con máscara /24 para las LAN donde queda el servidor y las máquinas Alpine. Entre routers la máscara deberá ser /30 como se mostró en clase teórica.

   Use las siguientes redes: 172.16.1.0/24 para la LAN del servidor web, 172.16.2.0/24 para la LAN de la PC 1, 172.16.3.0/24 para la LAN de la PC 2. Entre routers utilice la red 192.168.10.0 (haga subnetting según corresponda).

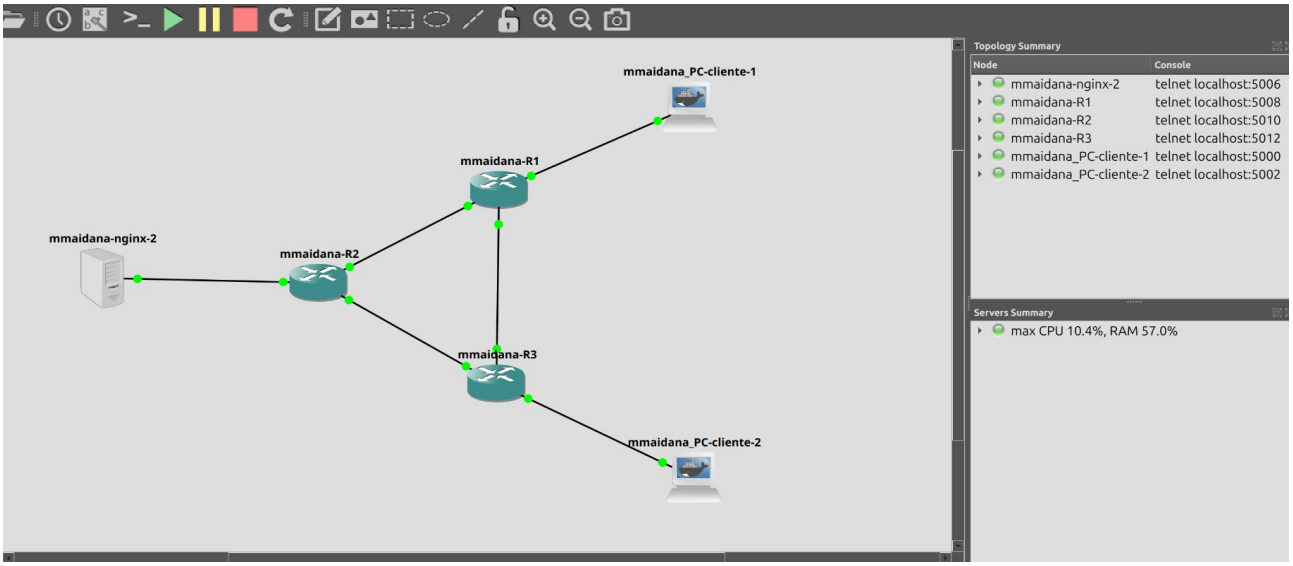
   Habilite RIP en los routers como mostrado en clase teórica.



2 Las máquinas Alpine Linux deben ser con la nueva imagen con curl y nmap.

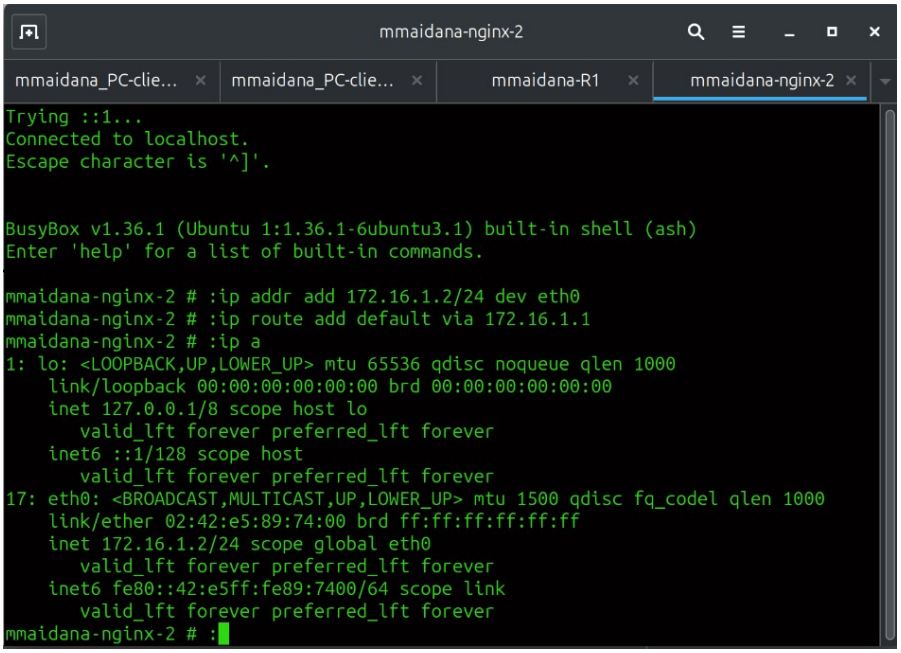
Se debe iniciar la simulación y ejecutar las siguientes acciones, adjuntando en cada caso capturas de pantalla (de ambos extremos cuando corresponda):

captura de pantalla con topografia y simulacion iniciada

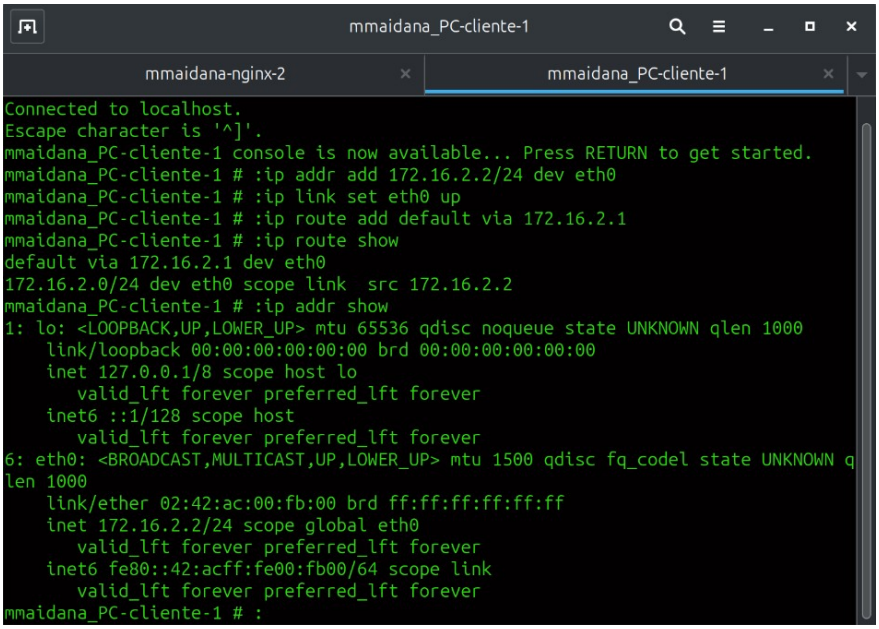


captura asignacion de IPs y levantar interfaces y comprobacion

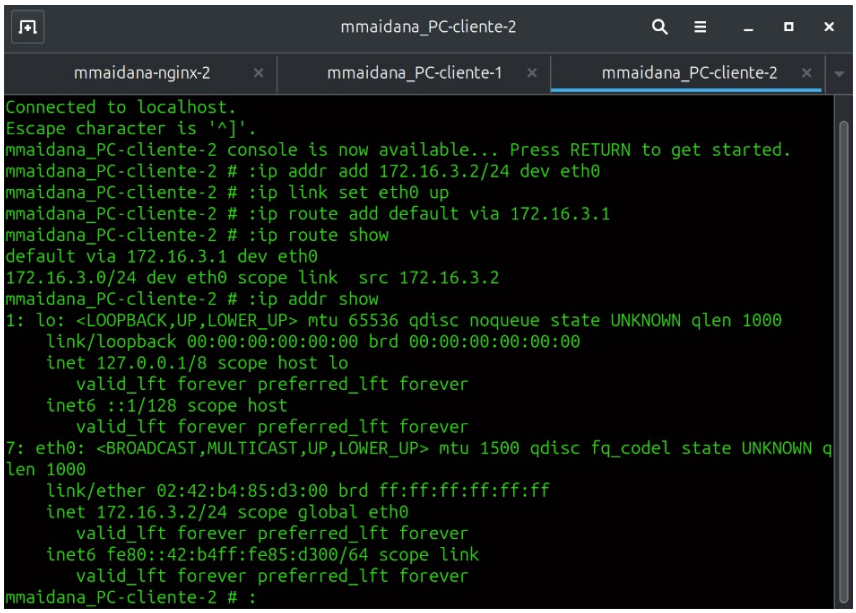
captura consola : servidor mmaidana-nginx-2



captura consola : pc : mmaidana_PC-cliente-1

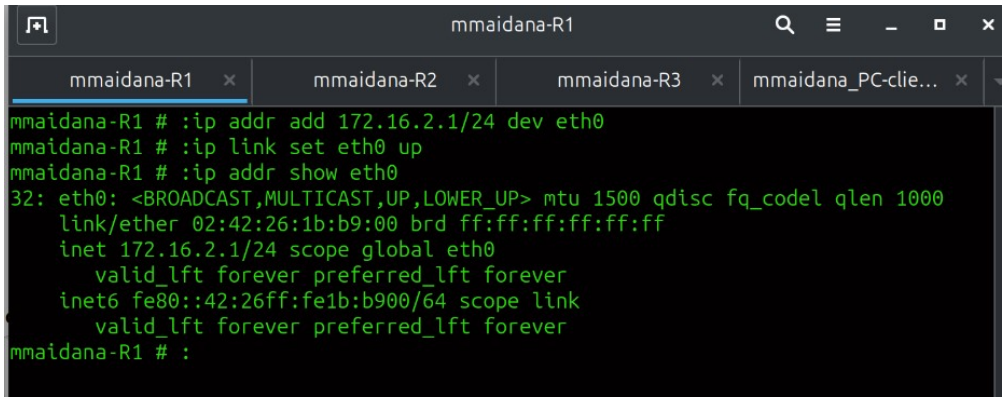


captura consola : pc : mmaidana_PC-cliente-2

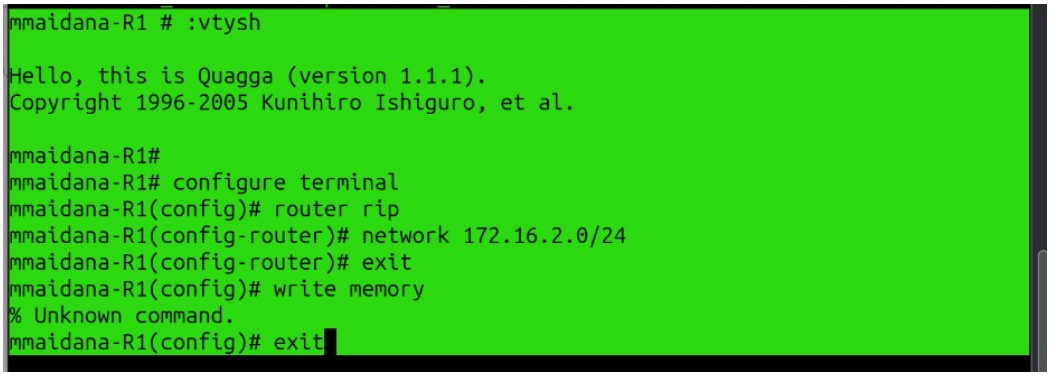


mmaidana-R1 configurado con la IP 172.16.2.1/24 en eth0 para mmaidana_PC-cliente1

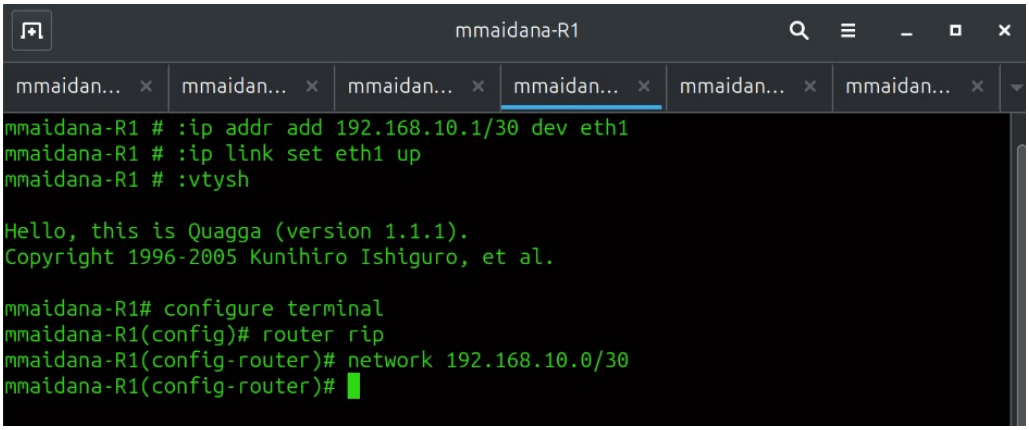
captura consola : router : mmaidana-R1



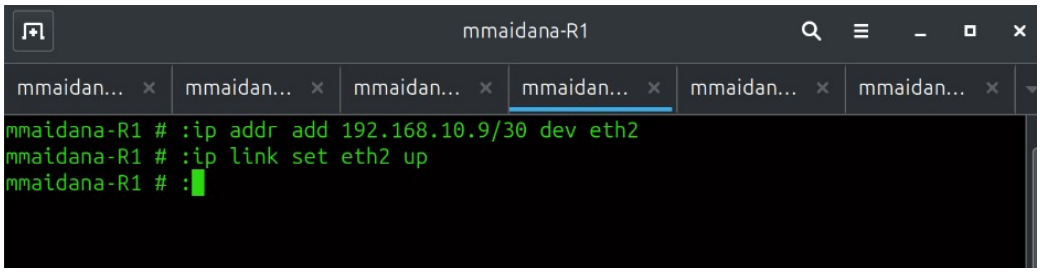
configuracion RIP en mmaidana-R1:



Enlace de IPs entre Routers R1-R2



Enlace de IPs entre Routers R1-R3



mmaidana-R2 configurado con la IP 172.16.1.1/24 en eth0 para servidor

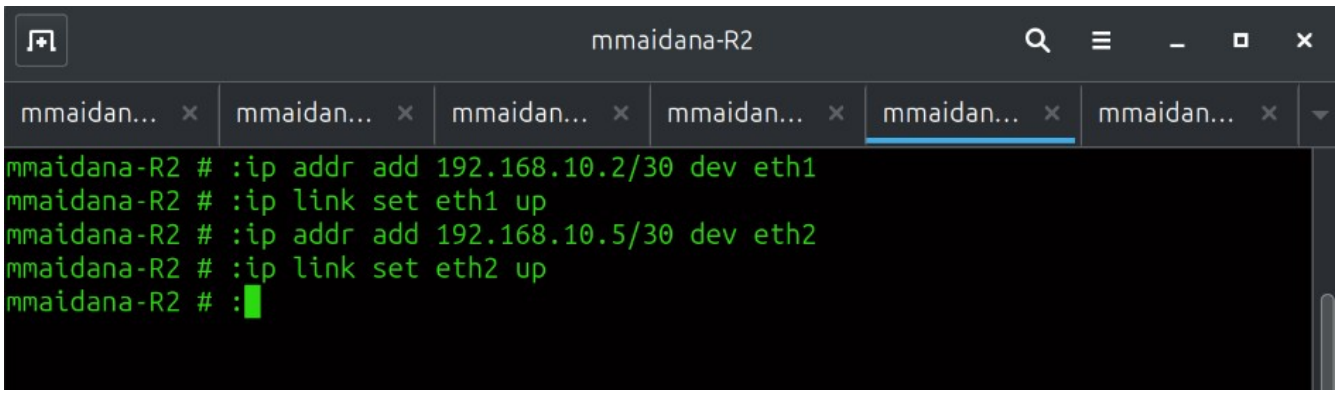
captura consola : router : mmaidana-R2

```
mmaidana-R2 # :ip addr add 172.16.1.1/24 dev eth0
mmaidana-R2 # :ip link set eth0 up
mmaidana-R2 # :
mmaidana-R2 # :# Verificar
mmaidana-R2 # :ip addr show eth0
33: eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc fq_codel qlen 1000
    link/ether 02:42:f8:a9:0a:00 brd ff:ff:ff:ff:ff:ff
    inet 172.16.1.1/24 scope global eth0
       valid_lft forever preferred_lft forever
    inet6 fe80::42:f8ff:fea9:a00/64 scope link
       valid_lft forever preferred_lft forever
mmaidana-R2 # :
```

configuracion RIP en mmaidana-R2:

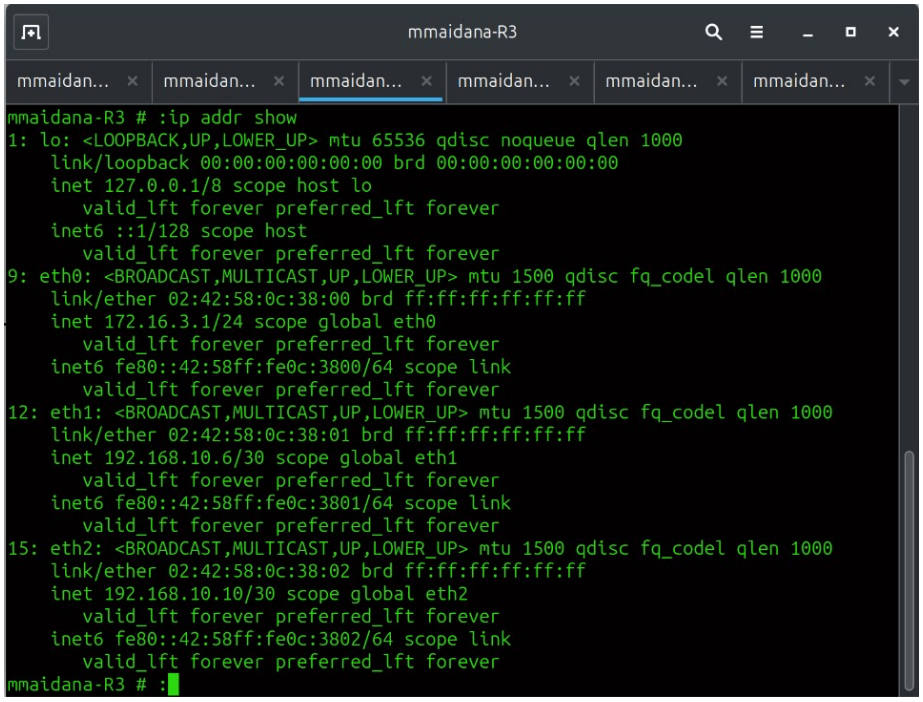
```
mmaidana-R2 # :vtysh

Hello, this is Quagga (version 1.1.1).
Copyright 1996-2005 Kunihiro Ishiguro, et al.

mmaidana-R2#
mmaidana-R2# configure terminal
mmaidana-R2(config)# router rip
mmaidana-R2(config-router)# network 172.16.1.0/24
mmaidana-R2(config-router)# exit
mmaidana-R2(config)# write memory
% Unknown command.
mmaidana-R2(config)# exit
mmaidana-R2#
```

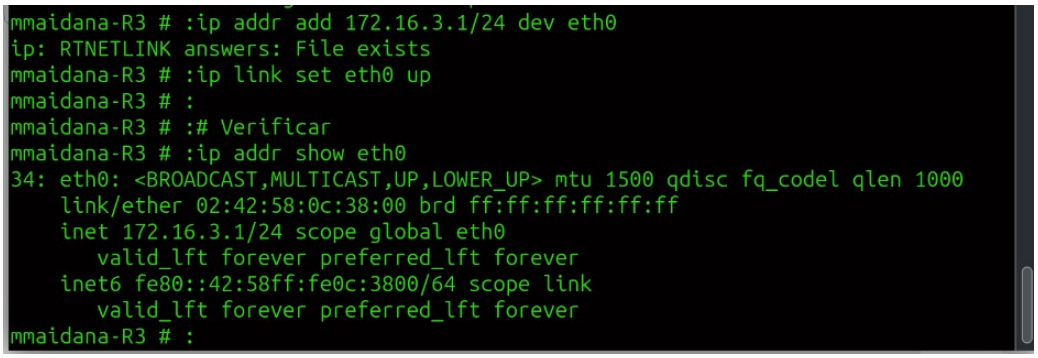
Enlace de IPs entre Routers R2 -- R1 y R2 --R3



mmaidana-R3 configurado con la IP 172.16.3.1/24 en eth0 para mmaidana_PC-cliente2

captura consola : router : mmaidana-R3

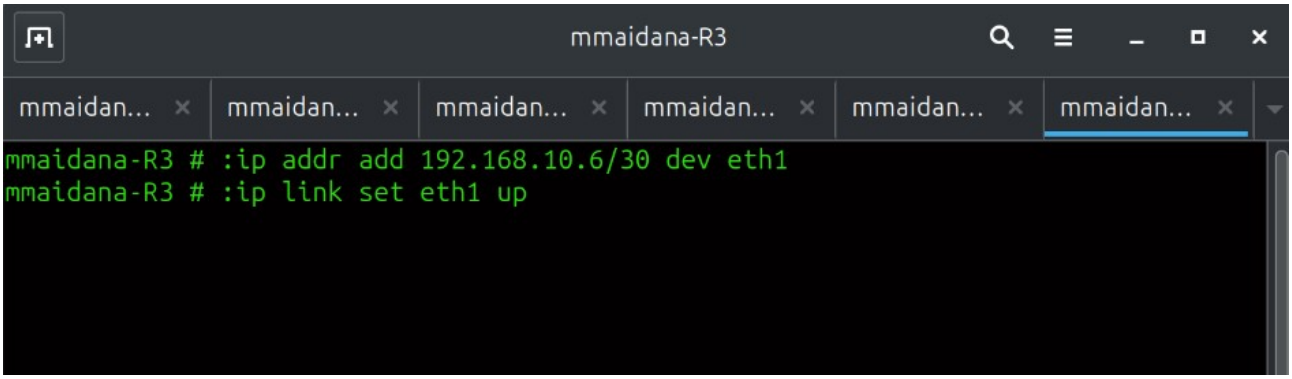




configuracion RIP en mmaidana-R3:

```
mmaidana-R3 # :vtysh

Hello, this is Quagga (version 1.1.1).
Copyright 1996-2005 Kunihiro Ishiguro, et al.

mmaidana-R3#
mmaidana-R3# # Agregar la red a RIP
mmaidana-R3# configure terminal
mmaidana-R3(config)# router rip
mmaidana-R3(config-router)# network 172.16.3.0/24
There is a same network configuration 172.16.3.0/24
mmaidana-R3(config-router)# exit
mmaidana-R3(config)# write memory
% Unknown command.
mmaidana-R3(config)# exit
mmaidana-R3#
```

Enlace de IPs entre Routers R3 -- R2



3 Comprobar mediante los comandos ip que desde las PCs se alcanza el servidor y mostrar las rutas que utilizan las PCs en cada caso.

En mmaidana_PC-cliente-1:

PC1 →al Servidor

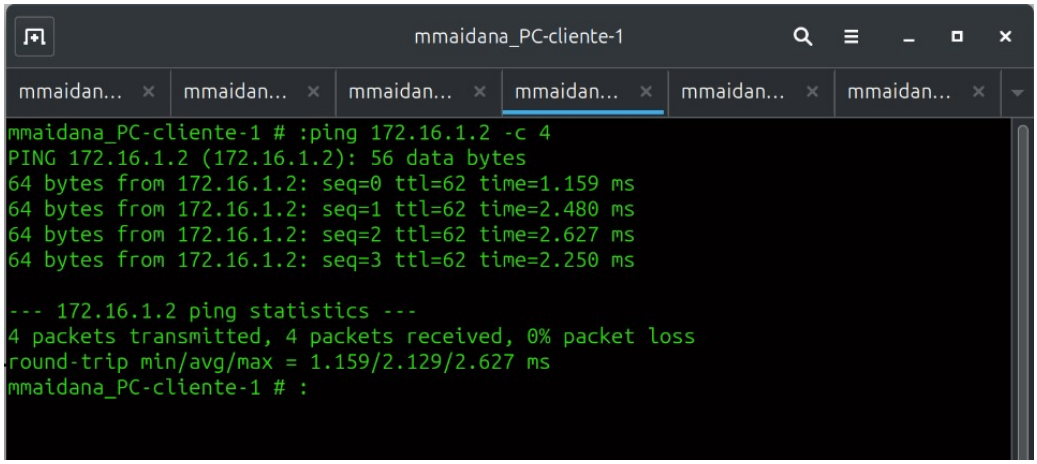
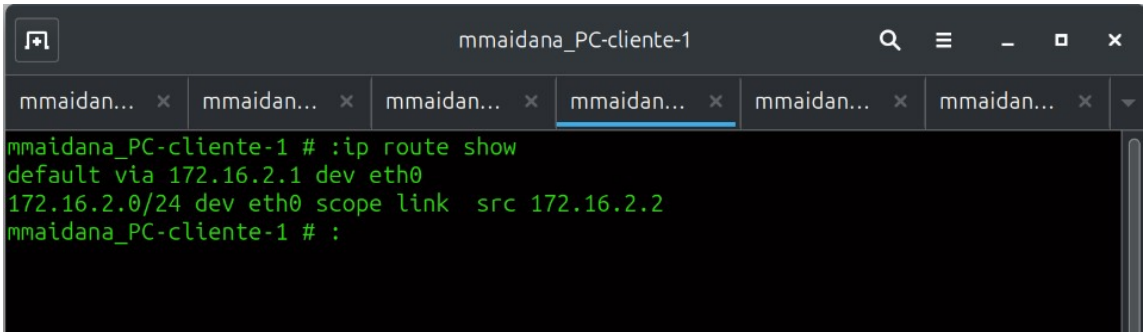
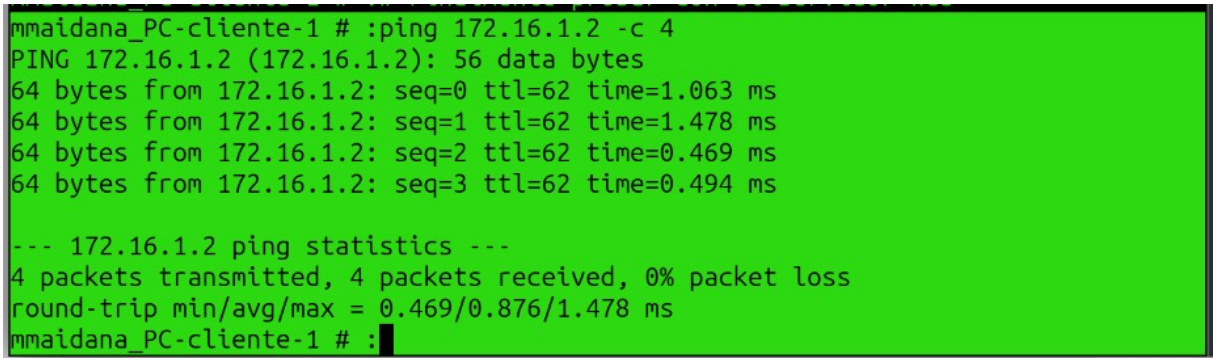


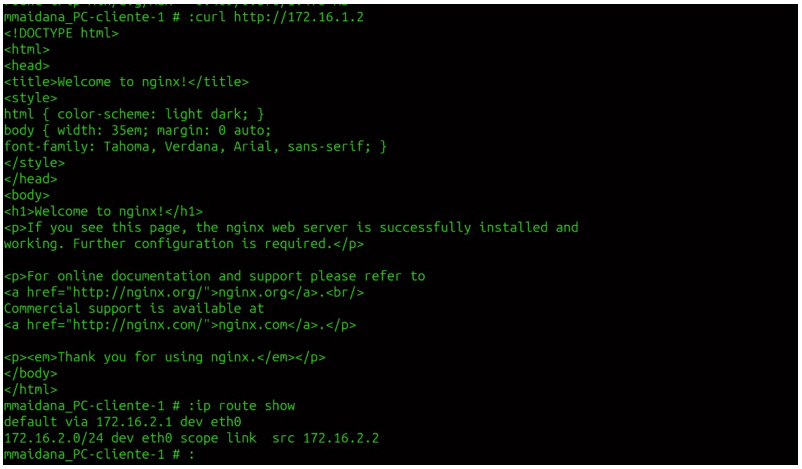
Tabla de Ruta (PC1)



PC1 → R1: Ping exitoso



PC1 → **Servidor Web**: Ping exitoso - pasa por múltiples routers



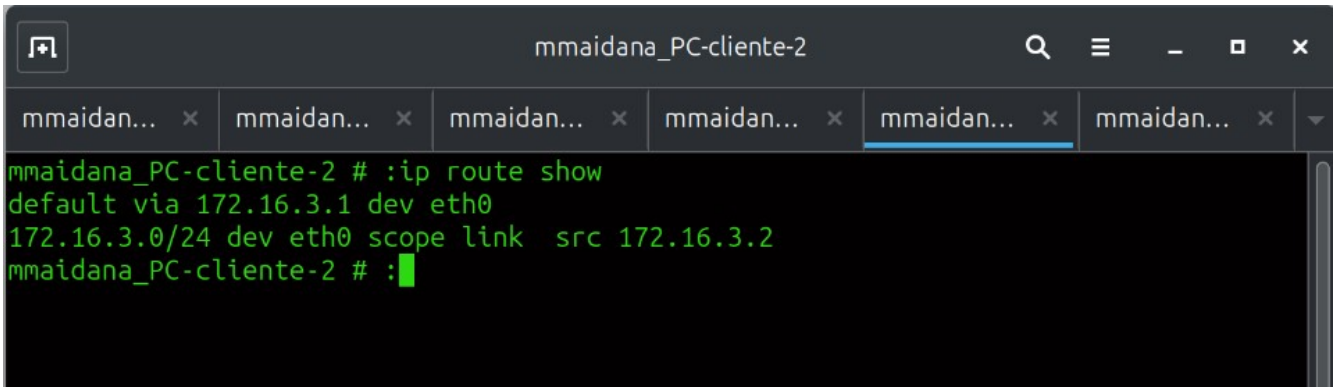
En mmaidana_PC-cliente-2:

mmaidana_PC-cliente2 → mmaidana-R3: Ping exitoso

mmaidana_PC-cliente2 → Servidor Web: Ping exitoso

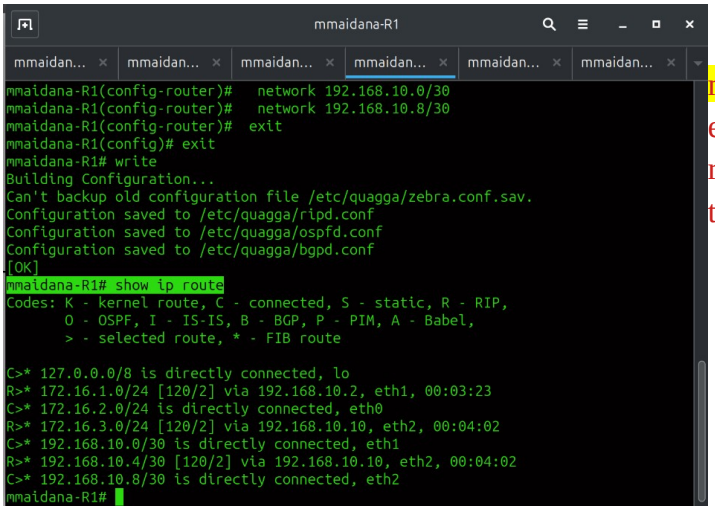
```
mmaidana_PC-cliente-2 # :ping 172.16.3.1 -c 2
PING 172.16.3.1 (172.16.3.1): 56 data bytes
64 bytes from 172.16.3.1: seq=0 ttl=64 time=0.394 ms
64 bytes from 172.16.3.1: seq=1 ttl=64 time=0.767 ms

--- 172.16.3.1 ping statistics ---
2 packets transmitted, 2 packets received, 0% packet loss
round-trip min/avg/max = 0.394/0.580/0.767 ms
mmaidana_PC-cliente-2 # :
mmaidana_PC-cliente-2 # :# Luego probar al servidor web
mmaidana_PC-cliente-2 # :ping 172.16.1.2 -c 4
PING 172.16.1.2 (172.16.1.2): 56 data bytes
64 bytes from 172.16.1.2: seq=0 ttl=62 time=5.162 ms
64 bytes from 172.16.1.2: seq=1 ttl=62 time=0.788 ms
64 bytes from 172.16.1.2: seq=2 ttl=62 time=1.003 ms
64 bytes from 172.16.1.2: seq=3 ttl=62 time=0.963 ms

--- 172.16.1.2 ping statistics ---
4 packets transmitted, 4 packets received, 0% packet loss
round-trip min/avg/max = 0.788/1.979/5.162 ms
mmaidana_PC-cliente-2 # :
```

Tabla de ruta PC 2



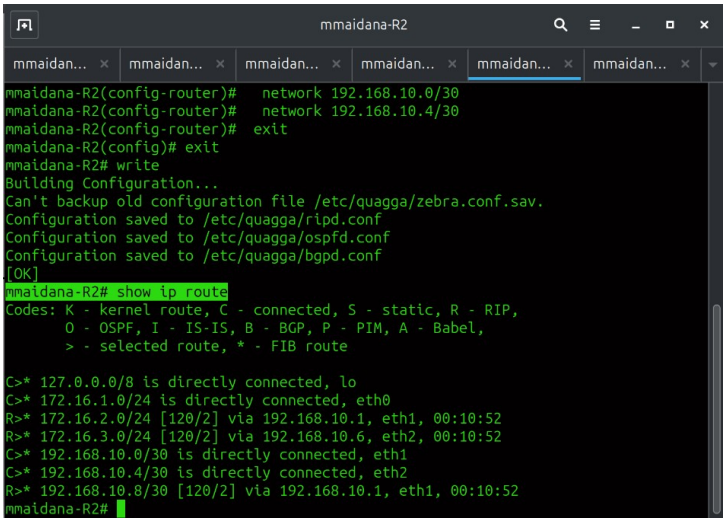
4 Analizar las tablas de enrutamiento de cada router. ¿Cómo se distinguen las rutas que se aprendieron mediante *RIP*?

mmaidana-R1



captura que confirma que mmaidana-R1 ha aprendido exitosamente la ruta para llegar a la red de mmaidana_PC-cliente-2 a través del router mmaidana-R3.

mmaidana-R2

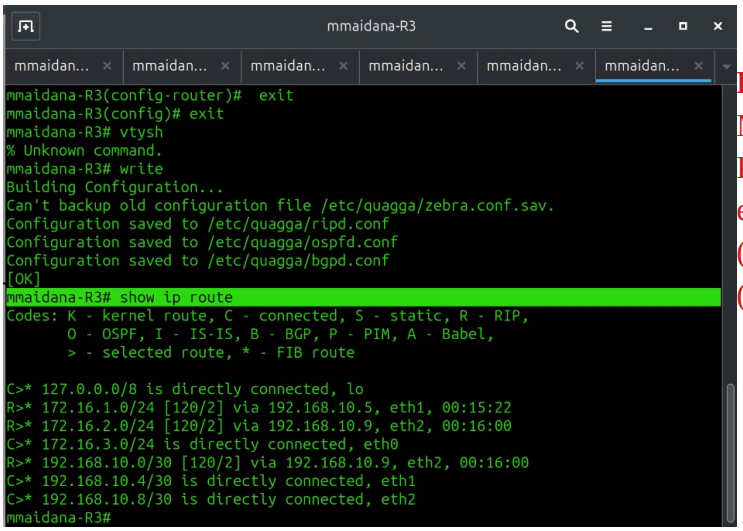


este router es el único que tiene conexión directa con los otros dos, por lo que aprende rutas de ambos lados:

A través de R1 (via 192.168.10.1): Ha aprendido la ruta hacia la red de PC1 (172.16.2.0/24).

A través de R3 (via 192.168.10.6): Ha aprendido la ruta hacia la red de PC2 (172.16.3.0/24).

mmaidana-R3



Rutas Conectadas (C): Muestra correctamente la red de PC2 (172.16.3.0/24) y los enlaces hacia R2 (192.168.10.4/30) y R1 (192.168.10.8/30).

Rutas RIP (R): Ha aprendido correctamente sobre las redes remotas:

La red del servidor (172.16.1.0/24) a través de R2 (via 192.168.10.5).

La red de PC1 (172.16.2.0/24) a través de R1 (via 192.168.10.9).

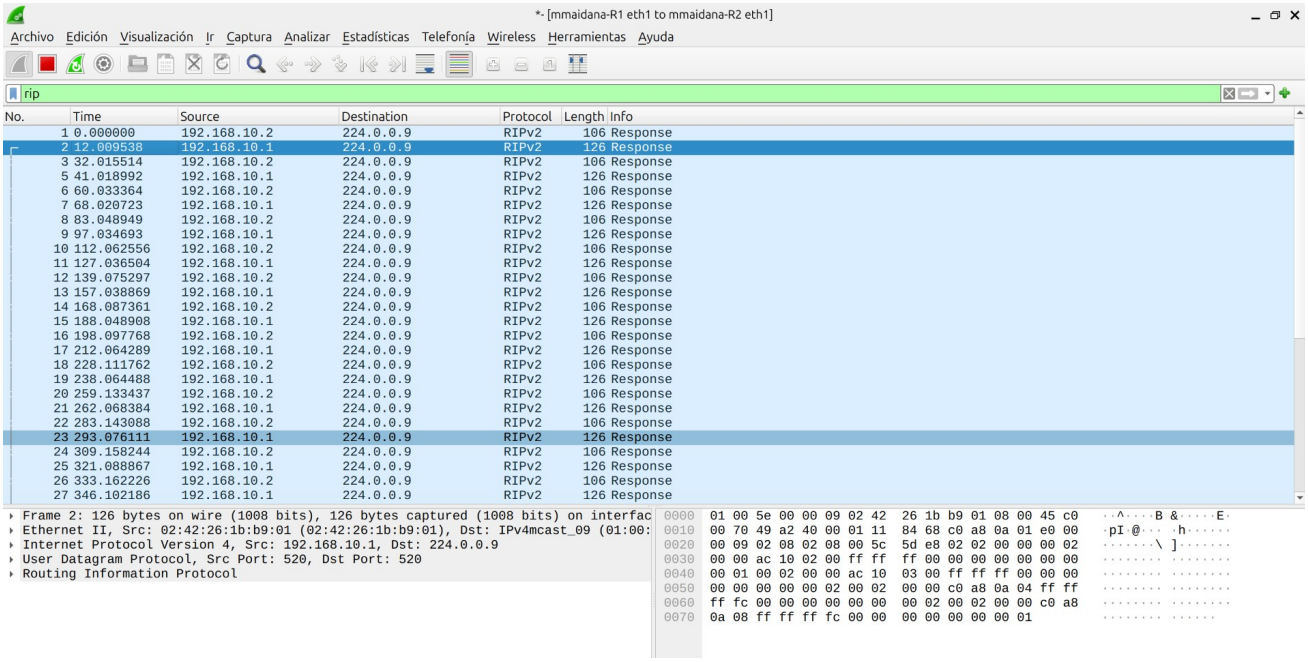
Al ejecutar show ip route en cada router, se pueden analizar las rutas que la red ha aprendido. Las rutas aprendidas dinámicamente a través de RIP se distinguen porque comienzan con una letra **"R"** .

Como la topología es un triángulo donde todos los routers tienen conexión directa con los otros dos, cada router aprende por RIP las redes LAN a las que no está conectado directamente.

- **En R1:** Se observa la ruta "R" hacia la LAN de PC2 (172.16.3.0/24) y la LAN del Servidor (172.16.1.0/24).
- **En R2:** Se observa la ruta "R" hacia la LAN de PC1 (172.16.2.0/24) y la LAN de PC2 (172.16.3.0/24).
- **En R3:** Se observa la ruta "R" hacia la LAN de PC1 (172.16.2.0/24) y la LAN del Servidor (172.16.1.0/24).

Esto confirma que el protocolo RIP está funcionando correctamente y ha distribuido la información de todas las redes a todos los routers.

5 Capturar tráfico *RIP* mediante la utilidad *Wireshark*. Seleccionar un mensaje *RIP* y responder las siguientes preguntas, analizando los distintos niveles de la pila TCP/IP (no olvidar las capturas de pantalla):

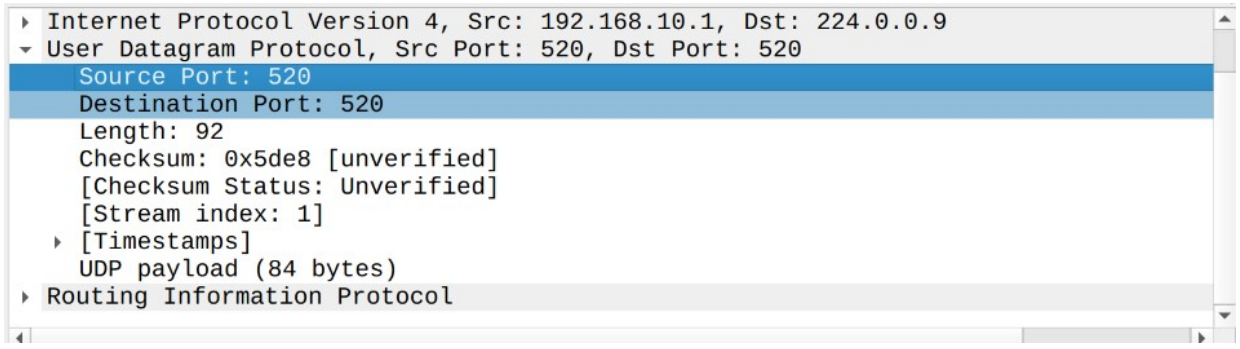


5.1 ¿Qué protocolo de transporte utiliza *RIP* para enviar los mensajes?

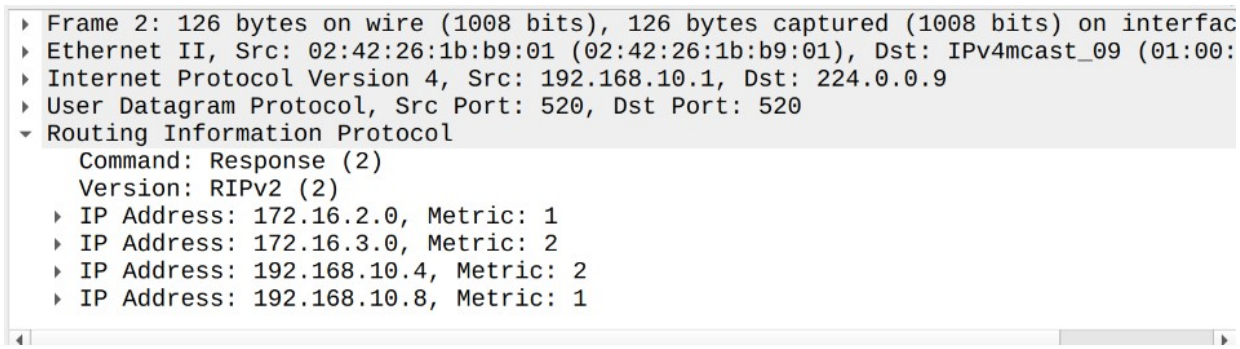
UDP (User Datagram Protocol)

5.2 ¿Por cuál puerto?

El puerto de origen (Src Port) y el puerto de destino (Dst Port) es el 520.



5.3 ¿Que información envía?



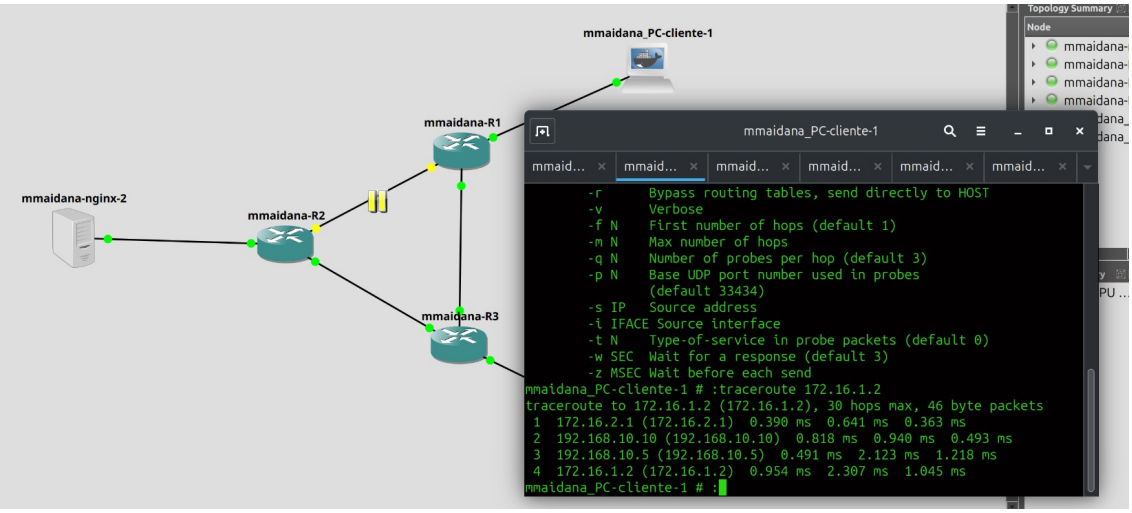
IP Address: 172.16.2.0, Metric: 1: R1 le dice a sus vecinos: " puedo llegar a la red `172.16.2.0` y me cuesta 1 solo salto (porque está directamente conectada a mí)".

IP Address: 172.16.3.0, Metric: 2: R1 le dice: " sé cómo llegar a la red `172.16.3.0` y me cuesta 2 saltos (porque tengo que pasar por R3 para llegar)".

IP Address: 192.168.10.4, Metric: 2: Le dice: "Sé cómo llegar al enlace entre R2 y R3, y me cuesta 2 saltos".

IP Address: 192.168.10.8, Metric: 1: Le dice: "Puedo llegar al enlace que tengo con R3 en 1 solo salto".

6 Suspender el enlace directo entre los routers **quagga-1** y **quagga-2**, esperar el tiempo de convergencia y mostrar la ruta entre **PC 1** y **ServidorWeb**. Comparar con la obtenida en el punto 3 y explicar. Restablecer el enlace.

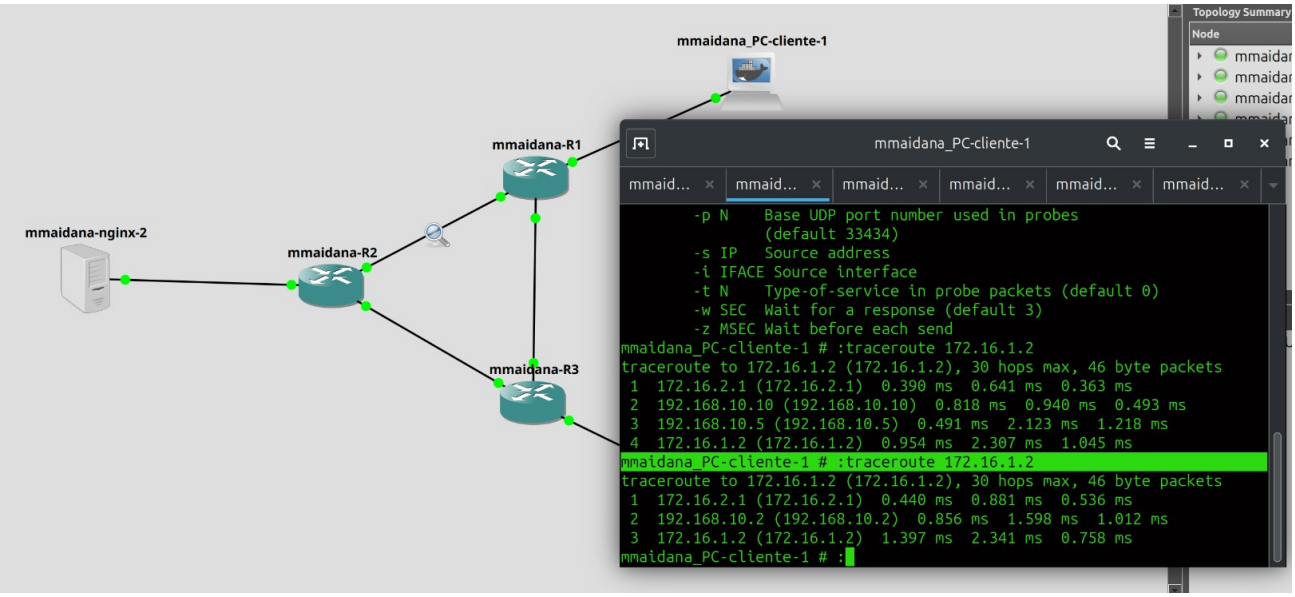


Analisis de la ruta que muestra el traceroute:

1. **172.16.2.1**: El primer salto es mmaidana-R1, el gateway por defecto. Esto es normal.
2. **192.168.10.10**: El segundo salto es la interfaz de mmaidana-R3. Esto confirma que, al no poder usar la ruta directa a R2, R1 ha enviado el paquete hacia R3.
3. **192.168.10.5**: El tercer salto es la interfaz de mmaidana-R2. Esto muestra que el paquete viajó exitosamente de R3 a R2.
4. **172.16.1.2**: El último salto es el servidor web, mmaidana-nginx-2.

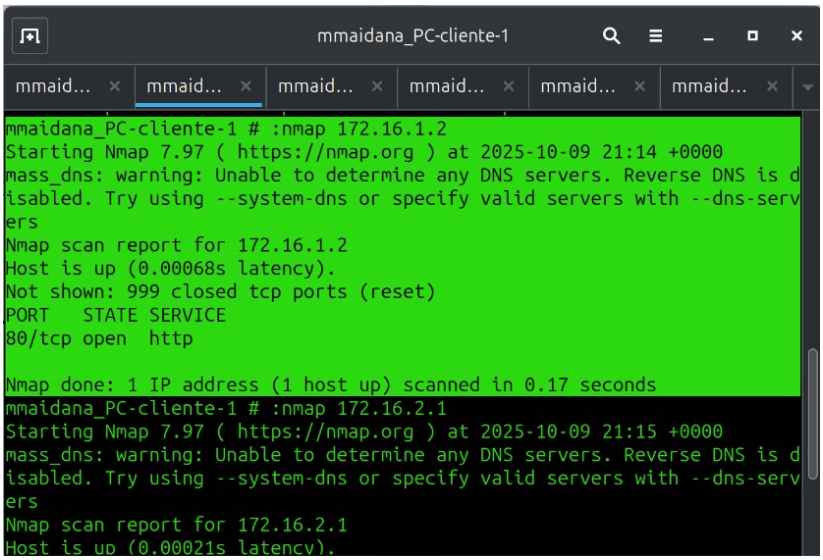
La nueva ruta es PC1 -> R1 -> R3 -> R2 -> Servidor. Como se puede ver, la red se adaptó exitosamente a la caída del enlace directo entre R1 y R2, utilizando el camino alternativo a través de R3. Esto demuestra que el enrutamiento dinámico con RIP funcionó correctamente.

Captura de enlace reestablecido



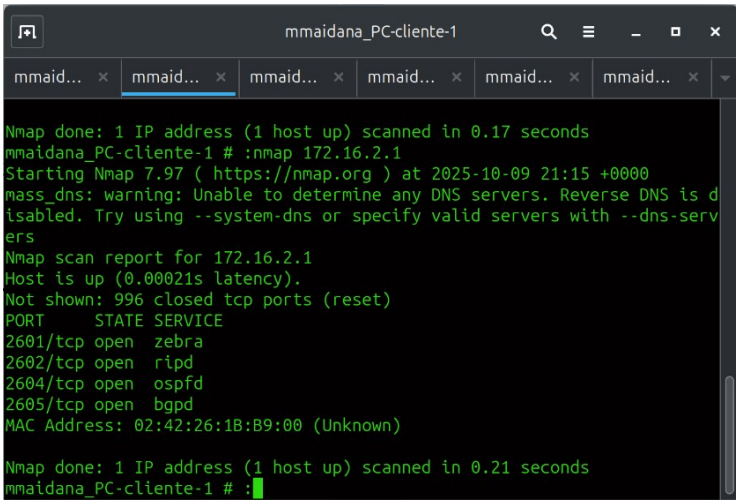
7 Desde alguna de las PCs, comprobar mediante el comando *nmap* qué puertos están abiertos en el servidor web y en los routers. Analizar la respuesta.

Captura desde mmaidana_PC-cliente1 hacia servidor web



nmap confirma que el host 172.16.1.2 está activo (Host is up) y, lo más importante, muestra que el puerto **80/tcp está abierto (open)** y corriendo el servicio http. Esto verifica que el servidor web nginx está funcionando correctamente y es accesible desde la red.

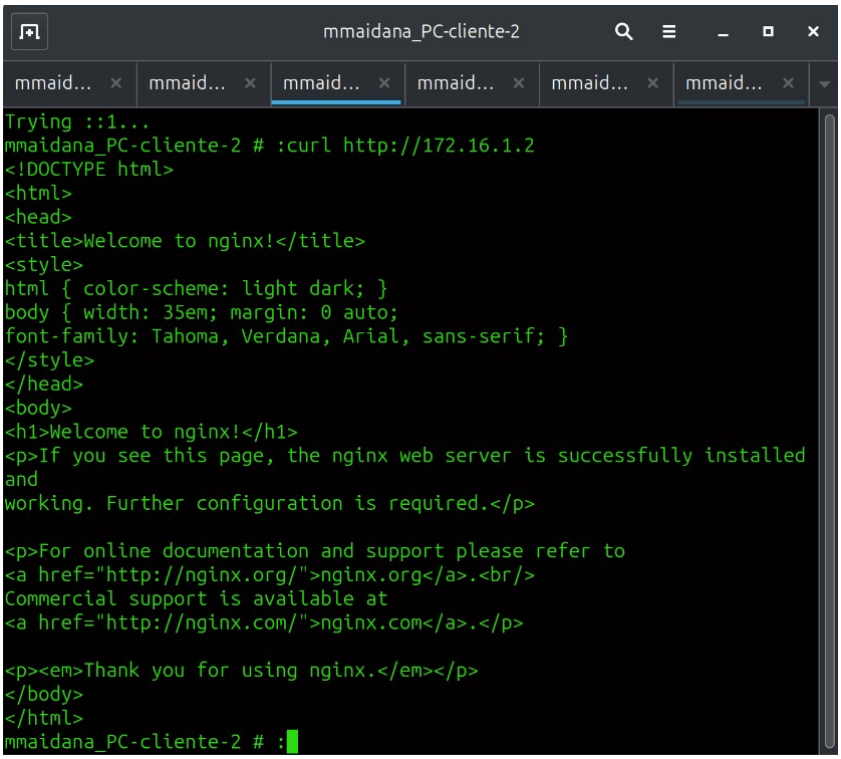
Captura desde mmaidana_PC-cliente1 hacia mmaidana-R1



Lo que nmap ha descubierto en el router mmaidana-R1 (172.16.2.1) son los puertos que utiliza el software de enrutamiento Quagga para su administración. Cada servicio de enrutamiento corre como un demonio separado y escucha en su propio puerto:

- 2601/tcp (zebra): Es el demonio principal de Quagga que maneja la tabla de enrutamiento del kernel de Linux.
- 2602/tcp (ripd): Es el demonio específico para el protocolo RIP, que es el que estás usando.
- 2604/tcp (ospfd): Es el demonio para OSPF (otro protocolo de enrutamiento).
- 2605/tcp (bgpd): Es el demonio para BGP (otro protocolo de enrutamiento).

8 Desde **pc-2** conectar con el **ServidorWeb** mediante *curl* y obtener una captura con la respuesta del servidor. Explicar.



esto confirma la conectividad completa de extremo a extremo desde la segunda LAN hasta el servidor

NAT

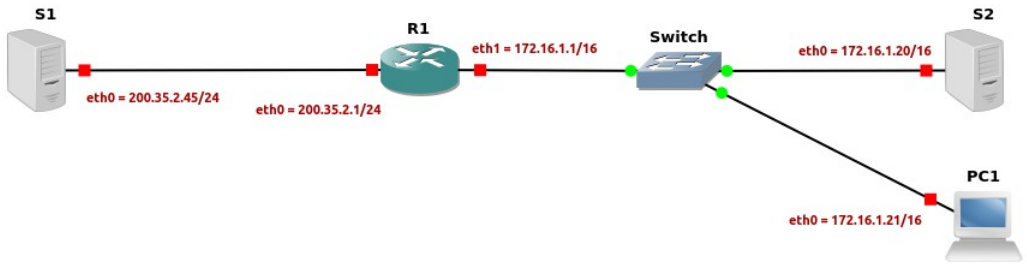
Recursos

Para la realización del trabajo sobre el simulador, serán necesarios:

- 1 router GNU/Linux a simular con la imagen de alpine. (R1)
- 2 servidores web *nginx.* (S1, S2)
- 1 cliente *alpine* con *wget*. Está incluido por defecto. (PC1)
- 1 switch ethernet. (Switch)
-

Preparación del laboratorio

Crear mediante el simulador la siguiente topología:



Muy importante: configurar los dispositivos de la siguiente manera:

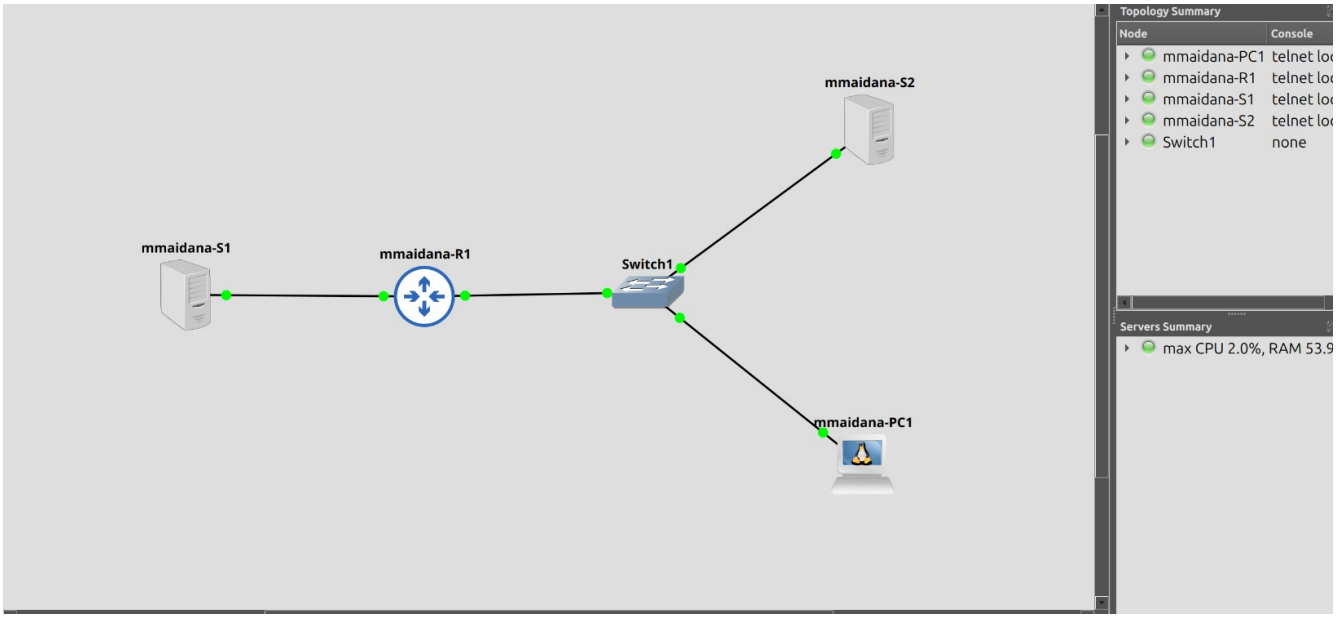
- En **S1** *no configurar el gateway por defecto*. Deberá tenerse especial cuidado en realizar esta acción, de lo contrario no se obtendrán los resultados deseados.
  En el ejercicio se ha conectado **S1** directamente a la interfaz externa del router por simplicidad y en la realidad podría estar en cualquier lugar, siempre que tenga una dirección IP pública conocida. Al no configurar el gateway, no tiene posibilidad de alcanzar directamente a los dispositivos del otro lado del router.
- En **PC1** y **S2**, configurar como gateway por defecto el router, a través de su interfaz eth1.
- Para preparar R1 utilizar el siguiente Dockerfile:

  ```
  FROM alpine

  RUN apk add iptables
  ```

  y luego ejecutar (no olvidar el punto del final):

  ```
  $ docker build -t alpine_iptables:v1 .
  ```

Captura de pantalla con la topologia iniciada



Actividades

1. Iniciar el laboratorio y comprobar que todos los dispositivos tienen su correspondiente configuración. Verificar que las direcciones de las interfaces están de acuerdo a la figura.

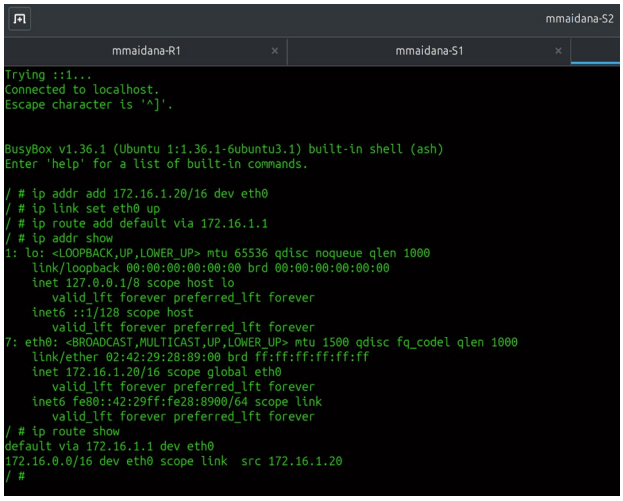
Captura configuracion en mmaidana-R1

```
/ # ip link set eth0 up
/ #
/ # # Interfaz "interna" que conecta al switch
/ # ip addr add 172.16.1.1/16 dev eth1
/ # ip link set eth1 up
/ #
/ # # Habilitar enrutamiento
/ # echo 1 > /proc/sys/net/ipv4/ip_forward
/ # ip addr show
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
/ # ip addr show
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue 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
5: eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc fq_codel qlen 1000
    link/ether 02:42:c6:85:39:00 brd ff:ff:ff:ff:ff:ff
    inet 200.35.2.1/24 scope global eth0
       valid_lft forever preferred_lft forever
    inet6 fe80::42:c6ff:fe85:3900/64 scope link
       valid_lft forever preferred_lft forever
8: eth1: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc fq_codel qlen 1000
    link/ether 02:42:c6:85:39:01 brd ff:ff:ff:ff:ff:ff
    inet 172.16.1.1/16 scope global eth1
       valid_lft forever preferred_lft forever
    inet6 fe80::42:c6ff:fe85:3901/64 scope link
       valid_lft forever preferred_lft forever
9: eth2: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc fq_codel qlen 1000
    link/ether 02:42:c6:85:39:02 brd ff:ff:ff:ff:ff:ff
    inet6 fe80::42:c6ff:fe85:3902/64 scope link
       valid_lft forever preferred_lft forever
10: eth3: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc fq_codel qlen 1000
    link/ether 02:42:c6:85:39:03 brd ff:ff:ff:ff:ff:ff
    inet6 fe80::42:c6ff:fe85:3903/64 scope link
       valid_lft forever preferred_lft forever
/ #
```

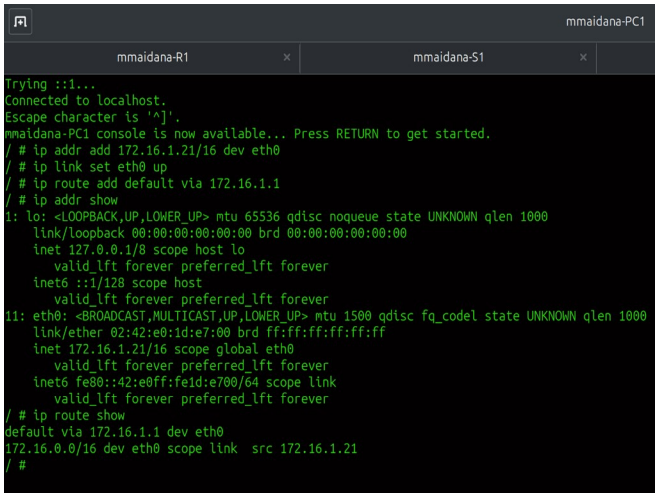
Captura configuracion en mmaidana-S1

```
Trying ::1...
Connected to localhost.
Escape character is '^]'.


BusyBox v1.36.1 (Ubuntu 1:1.36.1-6ubuntu3.1) built-in shell (ash)
Enter 'help' for a list of built-in commands.

/ # ip addr add 200.35.2.45/24 dev eth0
/ # ip link set eth0 up
/ # ip addr show
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue 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
6: eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc fq_codel qlen 1000
    link/ether 02:42:8a:a5:68:00 brd ff:ff:ff:ff:ff:ff
    inet 200.35.2.45/24 scope global eth0
       valid_lft forever preferred_lft forever
    inet6 fe80::42:8aff:fea5:6800/64 scope link
       valid_lft forever preferred_lft forever
/ # ip route show
200.35.2.0/24 dev eth0 scope link  src 200.35.2.45
/ #
```

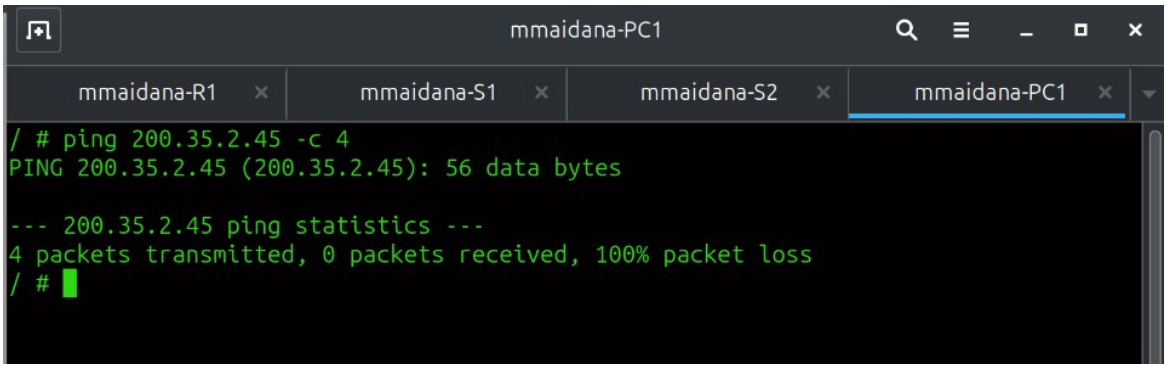
Captura configuracion en mmaidana-S1



Captura configuracion en mmaidana-PC1

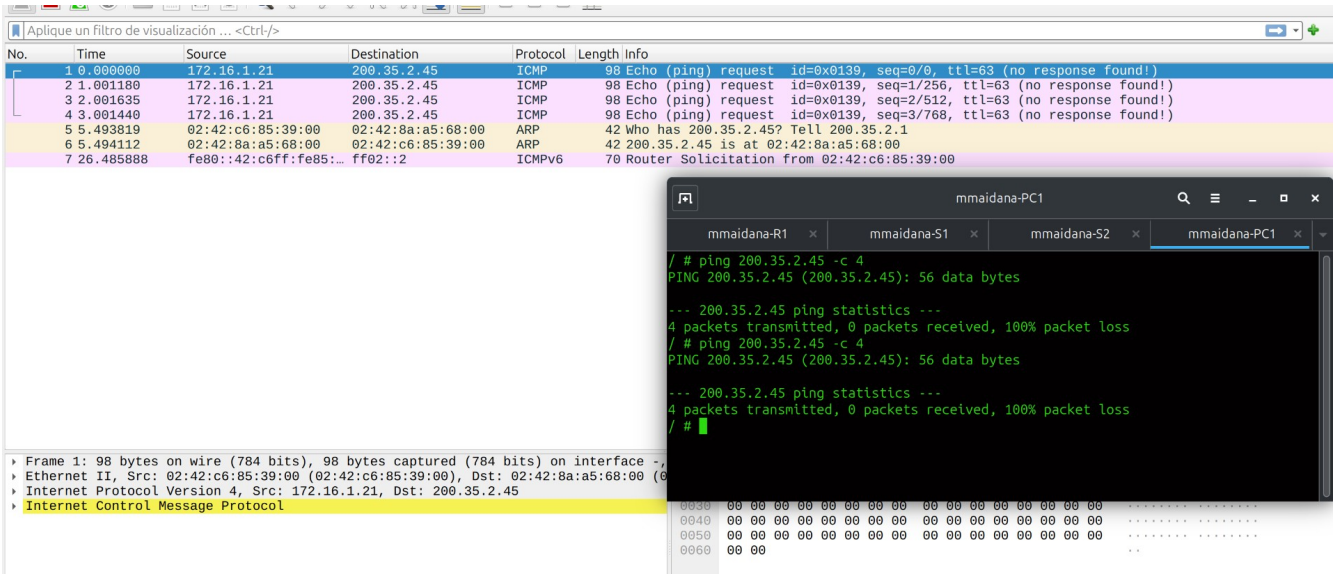


2. Verificar que desde **PC1** no hay conectividad con el servidor **S1**.



La línea 4 packets transmitted, 0 packets received, 100% packet loss confirma que la comunicación desde PC1 hasta S1 ha fallado.

3. Comprobar mediante captura de tráfico en **S1** que llegan los paquetes *echo request* del *ping* desde **PC1**.



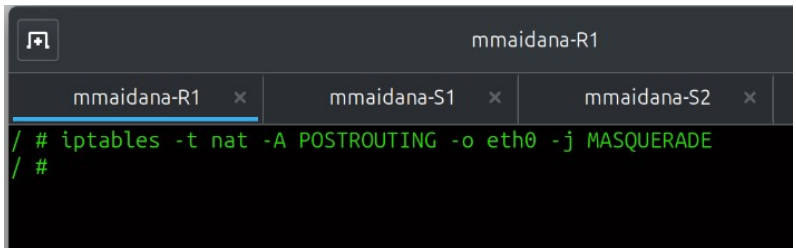
En la ventana de Wireshark : Se ven los paquetes ICMP Echo (ping) request saliendo de la IP de PC1 (172.16.1.21) y llegando correctamente a la IP de S1 (200.35.2.45). Esto confirma que los paquetes **SÍ LLEGAN** al servidor S1.

En la ventana de la consola : Se ve el ping fallando con 100% de paquetes perdidos.

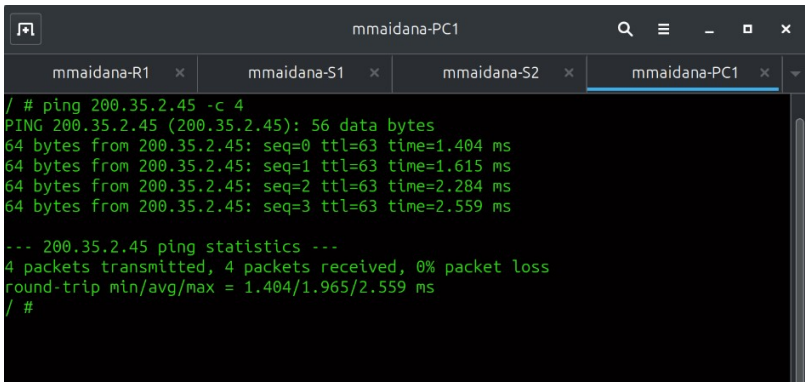
queda demostrado que PC1 envía los pings, S1 los recibe, pero S1 no sabe cómo enviar una respuesta a la dirección privada 172.16.1.21. Por eso el ping falla.

4. Agregar, desde la consola auxiliar de **R1**, reglas de *iptables* a la tabla NAT para permitir alcanzar a **S1**, enmascarando las direcciones privadas.

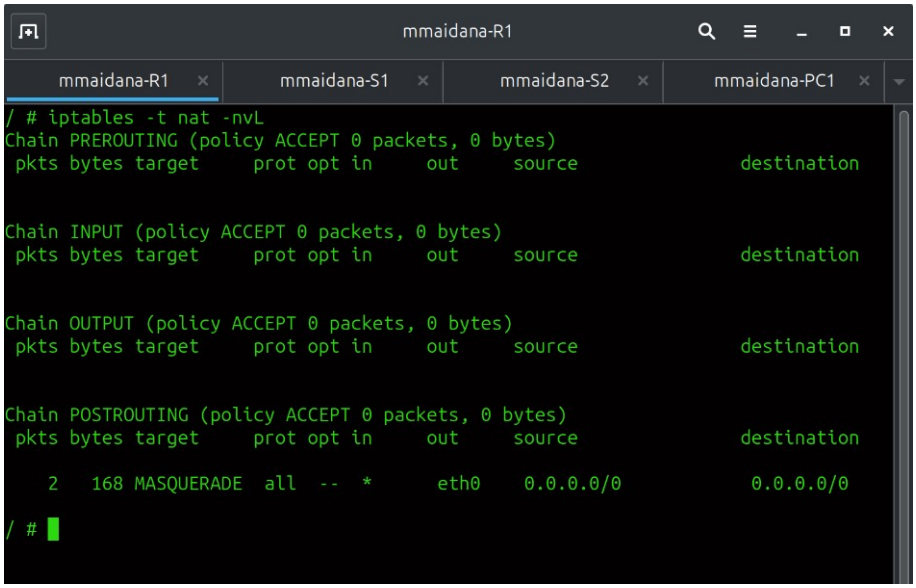
Captura iptables en mmaidana-R1



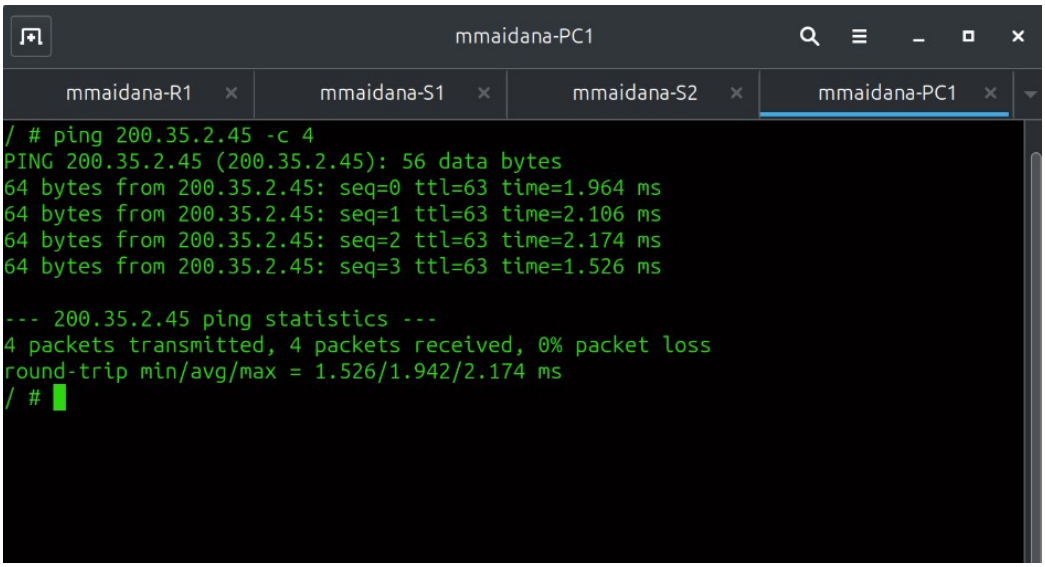
captura de ping desde mmaidana-PC1 hacia el servidor mmaidana-S1



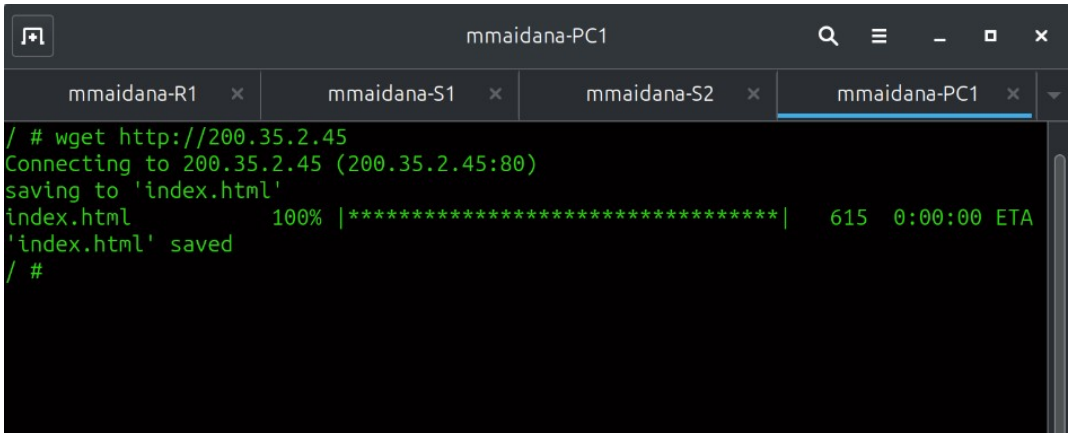
5. Realizar una captura de pantalla en **R1** mostrando la regla aplicada y los contadores. Sugerencia: aplicar el comando *iptables -t nat -nvL*.



6. Comprobar que **PC1** ahora puede comunicarse con **S1**.



7. Acceder a **S1** por *wget* desde **PC1**. Capturar el trafico con *Wireshark* tanto en las interfaces de **R1**, como en la interfaz de **S1**.
   Identificar los paquetes correspondientes a la descarga con *wget* en cada caso.



La salida del comando wget te muestra:

Connecting to 200.35.2.45...: Se estableció la conexión con el servidor web S1.

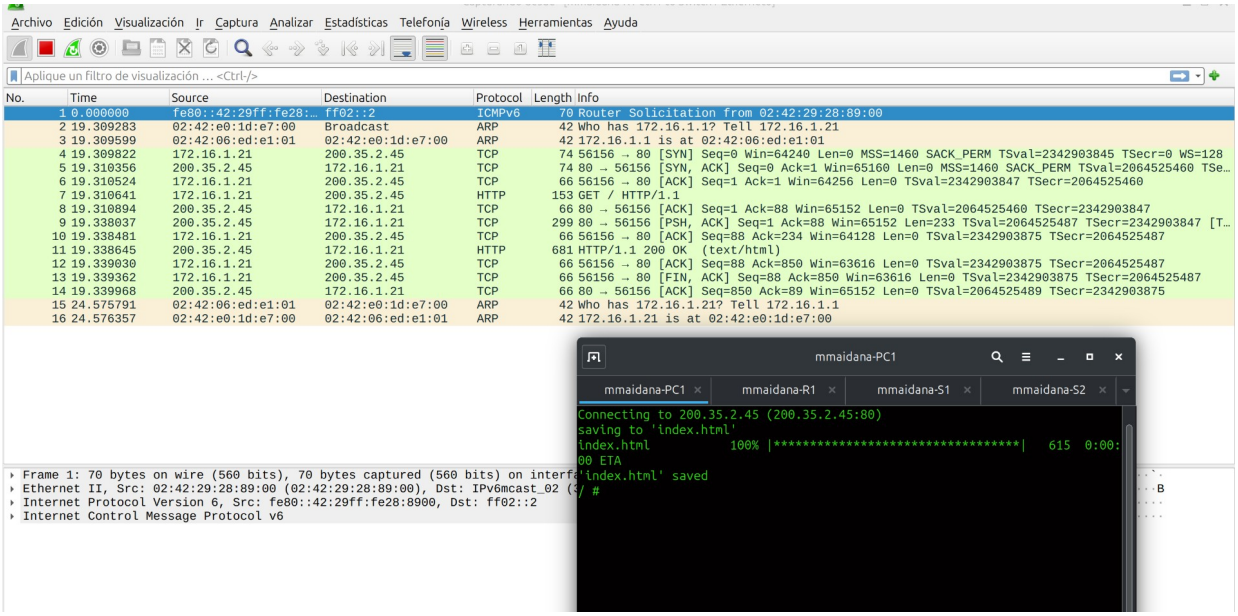
saving to 'index.html': Se está descargando el archivo.

'index.html' saved: El archivo de la página web se descargó correctamente en PC1.

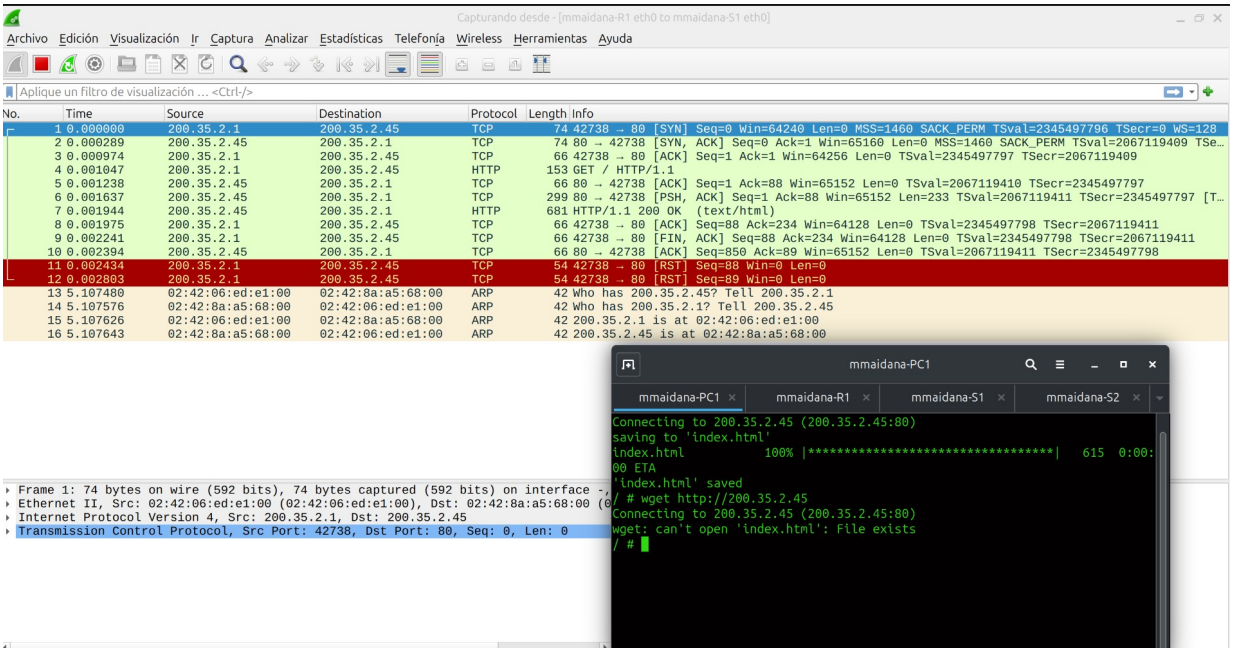
Consola mmaidana-PC1: Muestra que el wget fue exitoso.

Ventana de Wireshark: Muestra el tráfico en la red interna. el **paquete 7**, se ve la solicitud GET / HTTP/1.1. Lo más importante es que la IP de Origen (Source) es **172.16.1.21** (la IP privada de PC1).

Captura Interna Muestra el paquete saliendo de 172.16.1.21

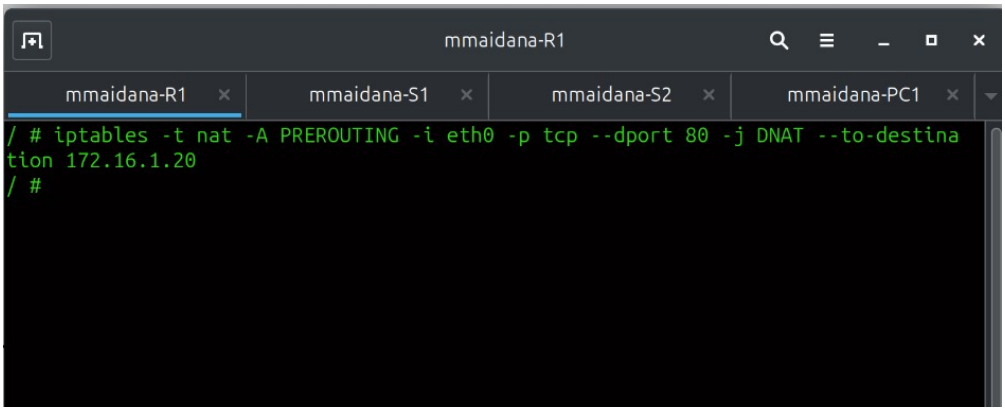


Captura Externa : Muestra el mismo paquete saliendo de 200.35.2.1.

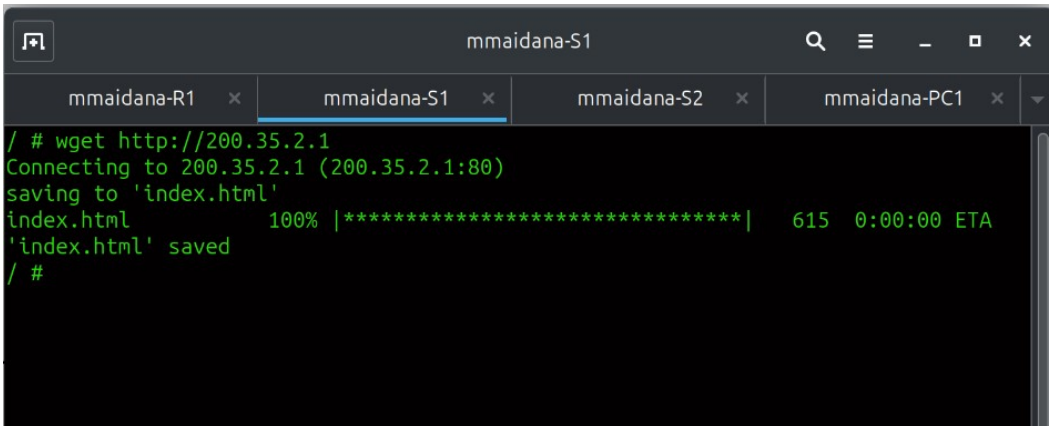


8. Mediante *Destination NAT* en **R1** permitir el acceso por *http* desde **S1** a **S2**. Realizar el seguimiento de los paquetes mediante capturas.

Aplicación de la regla de Destination NAT (DNAT) en R1



Comprobación de acceso exitoso desde S1 tras aplicar la regla de DNAT.

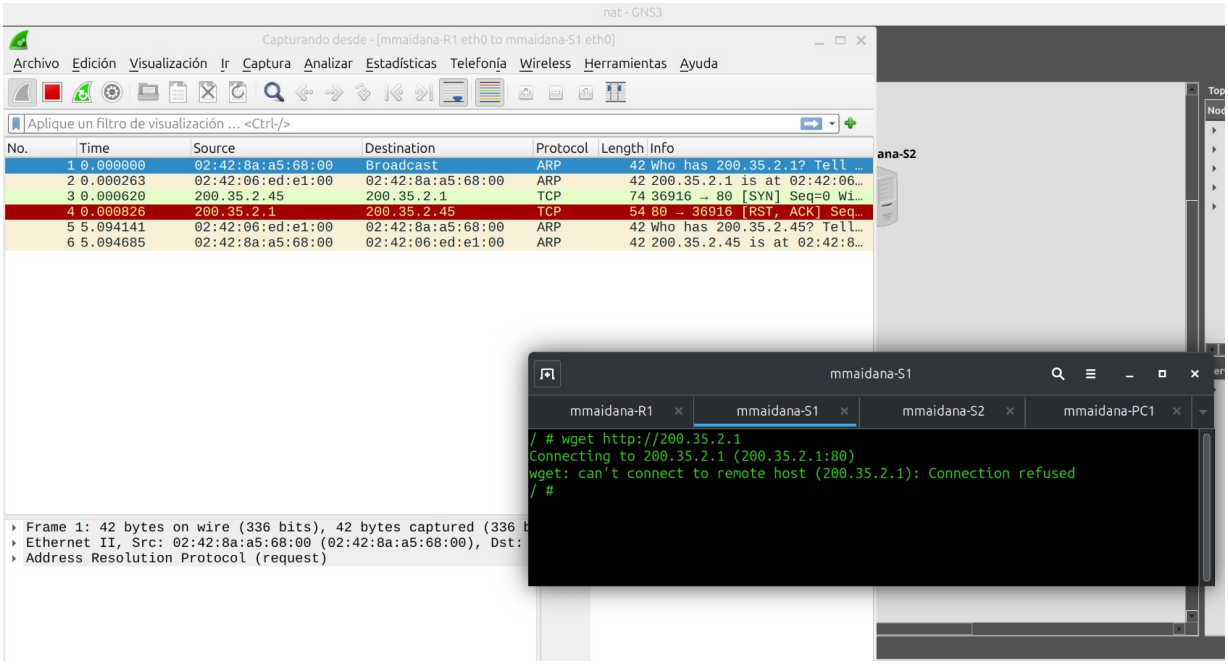


La salida de wget demuestra que la regla de Destination NAT (DNAT) funcionó a la perfección.

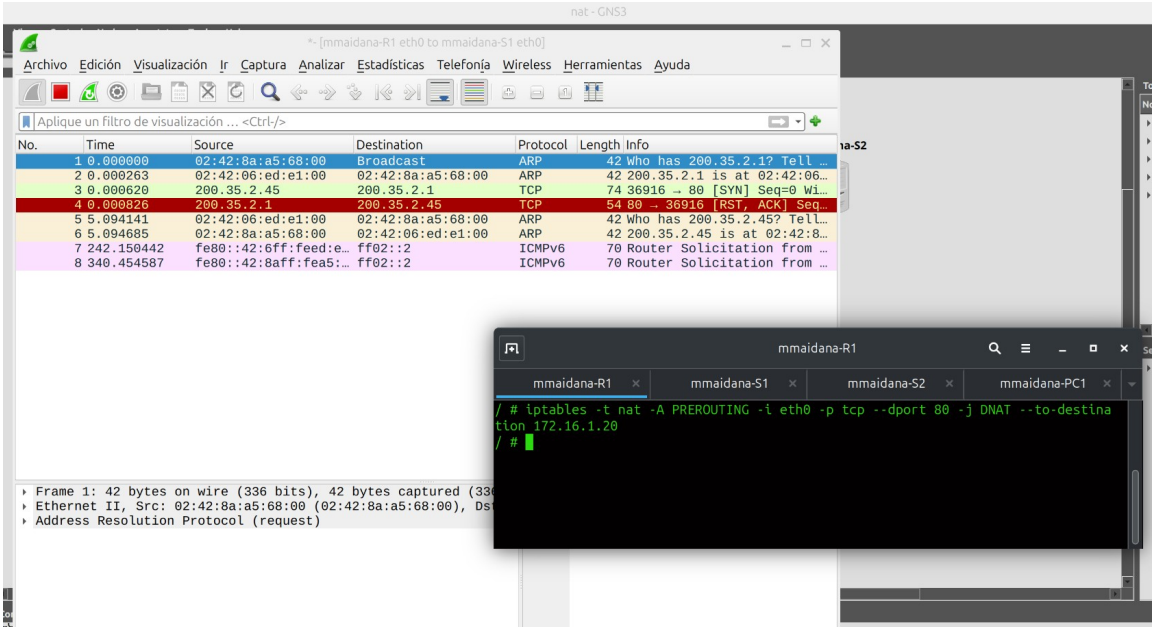
A pesar de que se le pedio a wget que se conectara a la IP del router (200.35.2.1), la regla de iptables interceptó el tráfico y lo redirigió correctamente al servidor interno S2 (172.16.1.20), permitiendo que la descarga del archivo index.html se completara.

Terminal (mmaidana-S1): Muestra el wget fallando con Connection refused.

- **Wireshark (eth0):** Muestra el porqué. El paquete 3 ([SYN]) es S1 intentando iniciar la conexión. El paquete 4 ([RST, ACK]) es el router R1 rechazando activamente esa conexión, porque no tiene ningún servicio escuchando en el puerto 80 y aún no tiene la regla de DNAT.



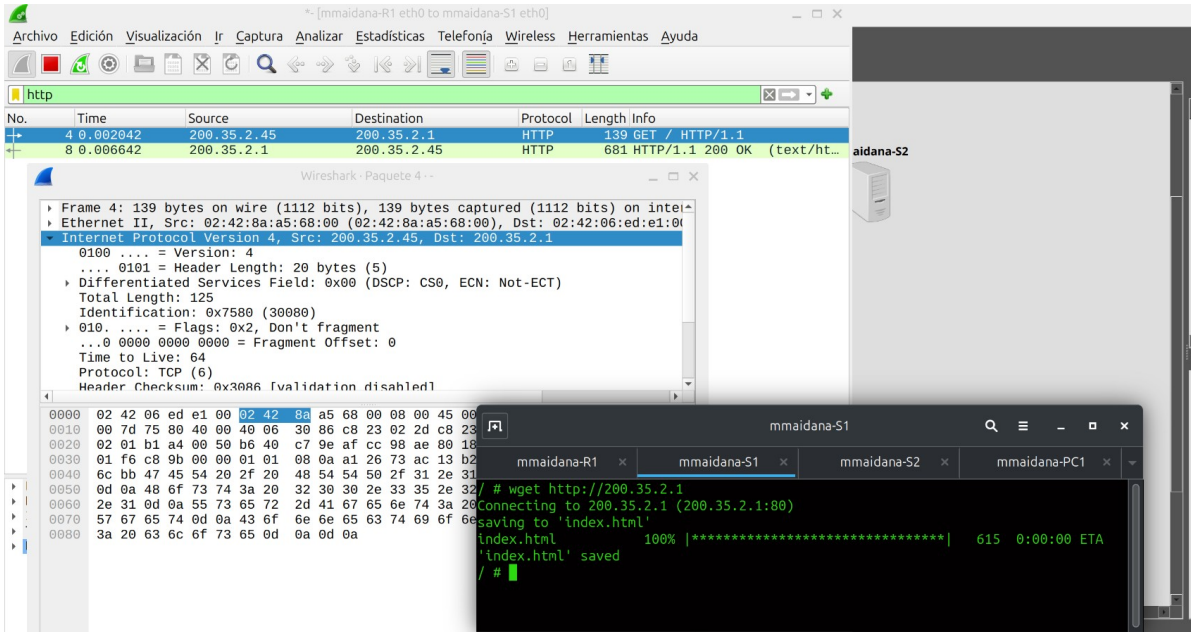
(Wireshark en eth0): Muestra *lo que pasa sin esa regla*. El paquete 3 ([SYN]) es S1 intentando conectarse, y el paquete 4 ([RST, ACK]) es el router R1 rechazando la conexión.



Antes de la traduccion del NAT

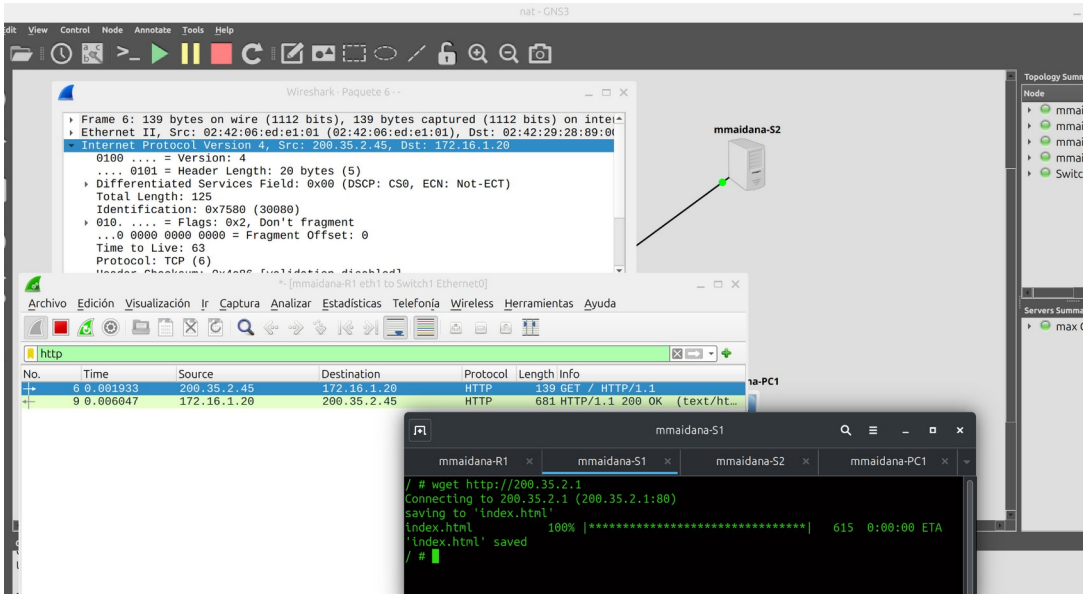
Consola mmaidana-S1: Muestra que el wget http://200.35.2.1 fue exitoso.

- **Wireshark:** El paquete GET / HTTP/1.1 (línea 4) muestra el tráfico *antes* de que el router lo traduzca:
- **Source (Origen):** 200.35.2.45 (El servidor S1)
- **Destination (Destino):** 200.35.2.1 (La IP pública del router R1)



Captura Interna, tomada en el enlace entre mmaidana-R1 (eth1) y Switch1.

- **Wireshark (Ventana del frente):** seleccionado el paquete GET / HTTP/1.1 (línea 9).
- **Detalles del Paquete:** El "Internet Protocol Version 4" muestra:
- **Source (Origen):** 200.35.2.45 (El servidor S1)
- **Destination (Destino):** 172.16.1.20 (La IP privada del servidor S2)



Paquete GET (línea 4):

- **Source (Origen):** 200.35.2.45 (El servidor S1)
- **Destination (Destino):** 200.35.2.1 (La IP pública del router R1)