

INFORME DE TPI

Redes de Datos 2

Grupo J

Gabriel Ollier - 02958/4

Joaquin Chanquia - 02887/7

12. Ejercicio para entregar en grupo y defender en coloquio.

- a) Resolver el direccionamiento IPv4 con el bloque IP asignado. Considerar los enlaces punto a punto salvo la red de n9, n10, n11, n13 y n14. Para la red de n9 considerar 40 hosts; para la red de n11 y n13, 328 hosts y para la red de n14, 500 hosts.

Direccionamiento IP Asignado

- **IPv4:** 140.222.16.0/21
- **IPv6:** 2001:fef8:df80::/48

Se utilizarán enlaces punto a punto en la mayoría de las redes, exceptuando la red compuesta por los nodos n9, n10, n11, n13 y n14.

Subredes IPv4 para Redes Específicas

Red de n14

- **Requerimiento de hosts:** 500 hosts (9 bits para hosts, máscara /23).
- **Asignación:** 140.222.16.0/23
- **Direcciones IP:**
 - Nodo n8: 140.222.16.1/23
 - Nodo n14: 140.222.16.2/23

Red de n11 y n13

- **Requerimiento de hosts:** 328 hosts (9 bits para hosts, máscara /23).
- **Asignación:** 140.222.18.0/23
- **Direcciones IP:**
 - Nodo n5: 140.222.18.1/23
 - Nodo n11: 140.222.18.2/23
 - Nodo n13: 140.222.18.3/23

Red de n9

- **Requerimiento de hosts:** 40 hosts (6 bits para hosts, máscara /26).
- **Asignación:** 140.222.20.0/26
- **Direcciones IP:**
 - Nodo n8: 140.222.20.1/26
 - Nodo n9: 140.222.20.2/26

Redes Punto a Punto (Enlaces)

Cada enlace requiere 2 bits para hosts, utilizando una máscara /30.

Red n1 - n2:

- Subred: 140.222.20.64/30
- IPs: n1: 140.222.20.65/30, n2: 140.222.20.66/30

Red n2 - n3:

- Subred: 140.222.20.68/30
- IPs: n2: 140.222.20.69/30, n3: 140.222.20.70/30

Red n1 - n3:

- Subred: 140.222.20.72/30
- IPs: n1: 140.222.20.73/30, n3: 140.222.20.74/30

Red n3 - n5:

- Subred: 140.222.20.76/30
- IPs: n3: 140.222.20.77/30, n5: 140.222.20.78/30

Red n5 - n6:

- Subred: 140.222.20.80/30
- IPs: n5: 140.222.20.81/30, n6: 140.222.20.82/30

Red n3 - n6:

- Subred: 140.222.20.84/30
- IPs: n3: 140.222.20.85/30, n6: 140.222.20.86/30

Red n3 - n15:

- Subred: 140.222.20.88/30
- IPs: n3: 140.222.20.89/30, n15: 140.222.20.90/30

Red n4 - n15:

- Subred: 140.222.20.92/30
- IPs: n4: 140.222.20.93/30, n15: 140.222.20.94/30

Red n2 - n4:

- Subred: 140.222.20.96/30
- IPs: n2: 140.222.20.97/30, n4: 140.222.20.98/30

Red n2 - n7:

- Subred: 140.222.20.100/30
- IPs: n2: 140.222.20.101/30, n7: 140.222.20.102/30

Red n7 - n8:

- Subred: 140.222.20.104/30
- IPs: n7: 140.222.20.105/30, n8: 140.222.20.106/30

Red n1 - n10:

- Subred: 192.168.1.0/24 (RFC-1918)
- IPs: n1: 192.168.1.1/24, n10: 192.168.1.2/24

- b) Configurar la red detrás de n5 (n5,n13,n11) y la Red C con el bloque IPv6 asignado.

Red n5, n11 y n13

- **Asignación IPv6:** 2001:fef8:df80:0::/64
- **Direcciones IP:**
 - Nodo n5: 2001:fef8:df80:0::1/64
 - Nodo n11: 2001:fef8:df80:0::2/64
 - Nodo n13: 2001:fef8:df80:0::3/64

Red C

Esta red abarca los siguientes enlaces:

Red n2 - n7:

- Subred IPv6: 2001:fef8:df80:1::/64
- IPs: n2: 2001:fef8:df80:1::1/64, n7: 2001:fef8:df80:1::2/64

Red n7 - n8:

- Subred IPv6: 2001:fef8:df80:2::/64
- IPs: n7: 2001:fef8:df80:2::1/64, n8: 2001:fef8:df80:2::2/64

Red n8 - n9:

- Subred IPv6: 2001:fef8:df80:3::/64
- IPs: n8: 2001:fef8:df80:3::1/64, n9: 2001:fef8:df80:3::2/64

Red n8 - n14:

- Subred IPv6: 2001:fef8:df80:4::/64
- IPs: n8: 2001:fef8:df80:4::1/64, n14: 2001:fef8:df80:4::2/64

c) Resolver con ruteo estático la topología.

Primero, para cada router se generó la tabla de enrutamiento de acuerdo con sus conexiones directas y redes alcanzables. En estas tablas se especifican la red destino, máscara, siguiente salto (Next Hop) y el dispositivo de salida. Esta estructura permite que cada router dirija correctamente el tráfico hacia su destino final, optimizando la conectividad en toda la red.

Para consultar las tablas completas de enrutamiento acceder al siguiente archivo:
[GrupoJ-TablasRuteo.xlsx](#)

Para facilitar la configuración y administración de las rutas estáticas en la aplicación CORE, se optó por simplificar las tablas de ruteo, se ha decidido utilizar la diagonal n3-n2 como la ruta principal, apoyándose en una serie de consideraciones que simplifican el flujo de tráfico como uso de Redes Default: Se definió una ruta por defecto que dirige el tráfico hacia rutas de mayor prioridad en caso de no coincidir con ninguna específicas, reducción de Rutas Específicas: Se agruparon ciertas subredes para reducir el número de entradas en las tablas de ruteo. Camino más corto y directo: La diagonal n3-n2 conecta directamente los nodos centrales n3 y n2, que actúan como puntos estratégicos entre las redes principales (Red A y Red C) y el núcleo central (Red B). Esto permite que el tráfico principal siga una ruta directa, reduciendo la latencia al evitar saltos innecesarios entre otros nodos, y manejar el balance de carga y eficiencia de recursos: Al definir la diagonal como la ruta principal, se asegura una utilización óptima de recursos, concentrando el tráfico en un enlace específico y evitando la dispersión de la carga en múltiples rutas.

El siguiente paso consistió en configurar el ruteo estático dentro del archivo de configuración de CORE para cada router. Para esto, se agregó la tabla de ruteo correspondiente en el servicio StaticRoute de cada router. Esto permite que, al iniciar la simulación, cada router conozca previamente sus rutas.

Para establecer el ruteo estático, se utilizaron los comandos proporcionados por los docentes en el formato siguiente:

```
ip route add <subred_destino>/<máscara> via <IP_next_hop>
```

Por ejemplo:

```
ip route add 140.222.20.92/30 via 140.222.20.98
```

Se configura una ruta específica para el rango 140.222.20.92/30, que envía el tráfico hacia el next hop en 140.222.20.98.

```
ip route add default via 140.222.20.70
```

Se define una ruta por defecto para el tráfico que no coincide con ninguna ruta específica, redirigiéndolo al next hop 140.222.20.70.

Para verificar que cada router tenga la tabla de ruteo correcta, se ejecutó el comando ip route show en la terminal de cada router dentro de CORE. Esto permitió confirmar que la configuración de ruteo estático se había aplicado exitosamente y que cada router reflejaba las rutas definidas en su tabla de ruteo.

En el siguiente enlace se puede acceder a los comandos utilizados en cada router:

[GrupoJ-ComandosRuteo.docx](#)

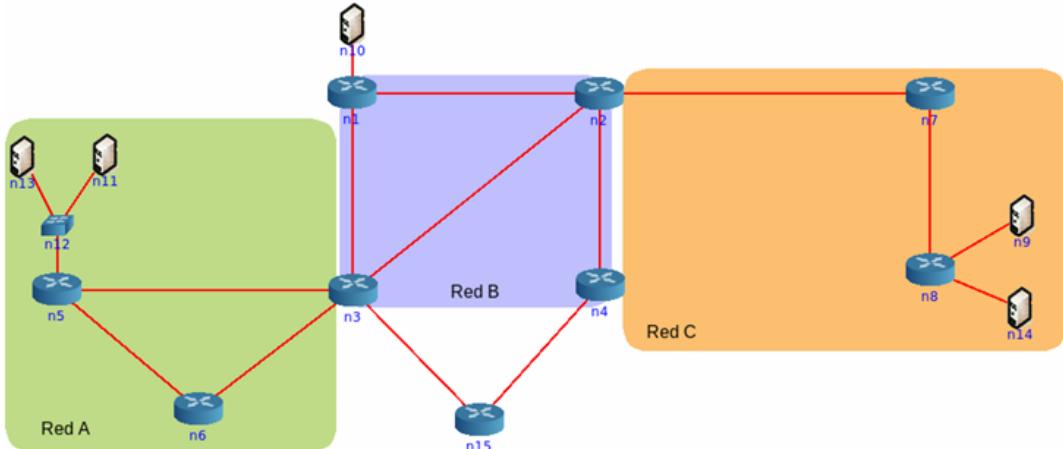
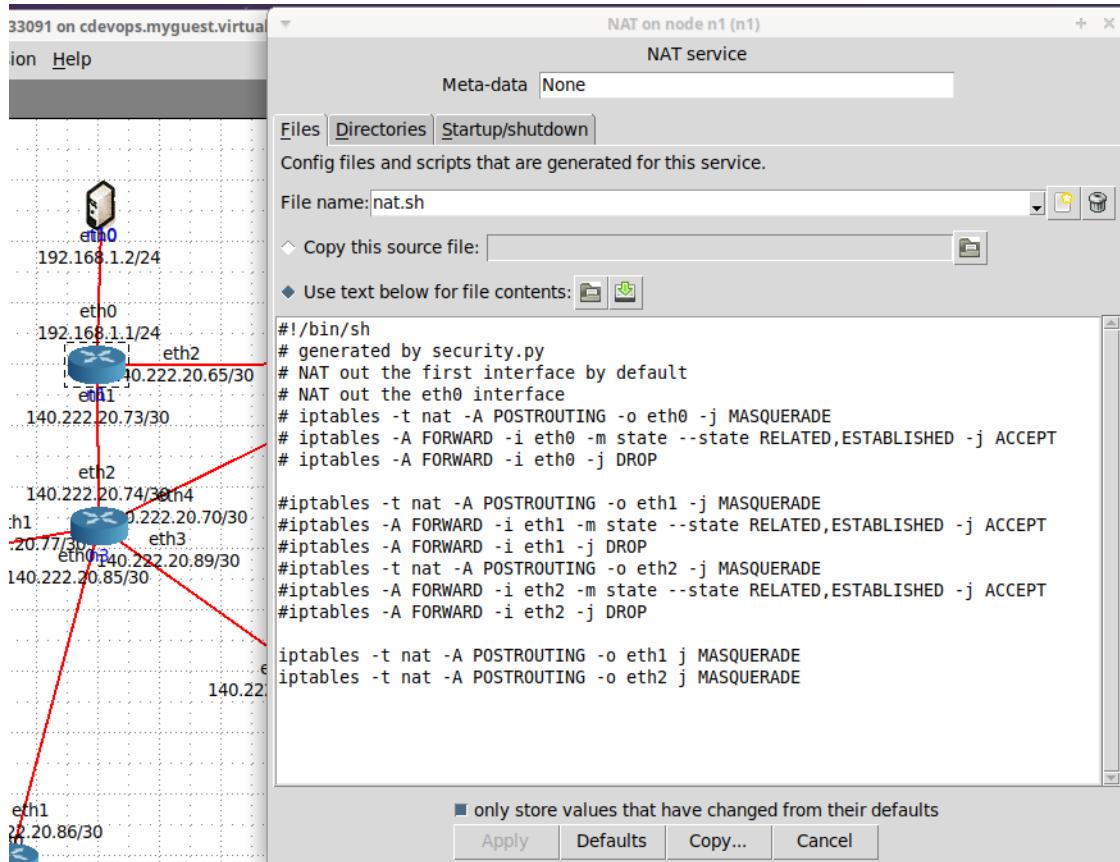


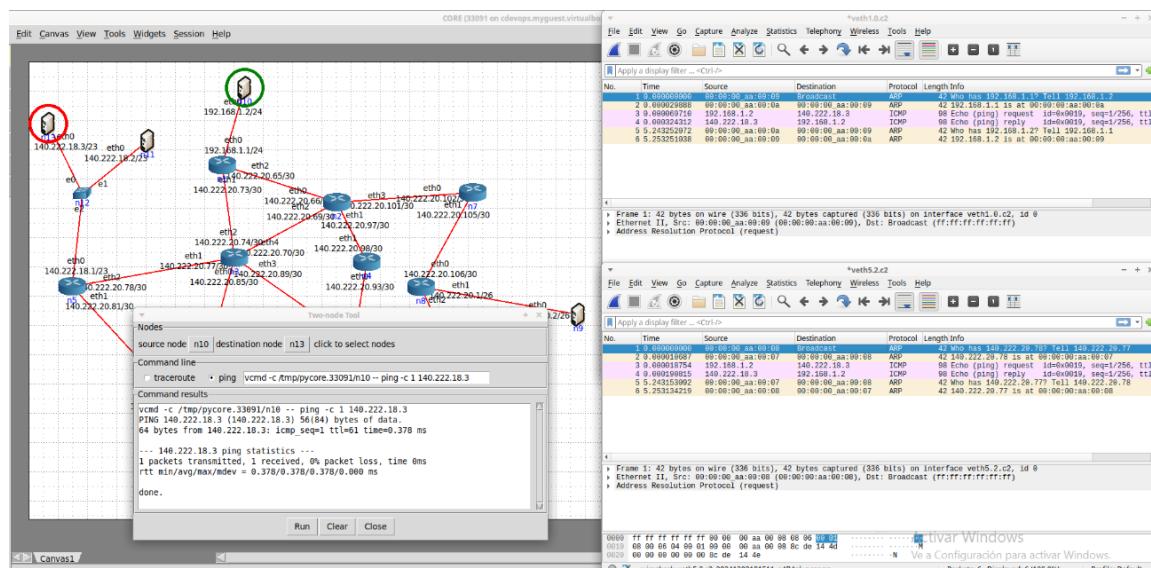
Figura 5: Diagrama de topología para TPI

d) Alternativo: Asignar a n10 una IP según RFC-1918 y configurar NAT en n1 para que pueda alcanzar al resto de los equipos.

En esta captura se puede ver la configuración necesaria para habilitar el servicio de NAT en el nodo n1. Habilitando las interfaces eth1 y eth2 como salidas a la red publica.



En la siguiente imagen se ve una prueba de un ping exitoso desde el nodo n10 en la red privada a el nodo n13 fuera de la misma

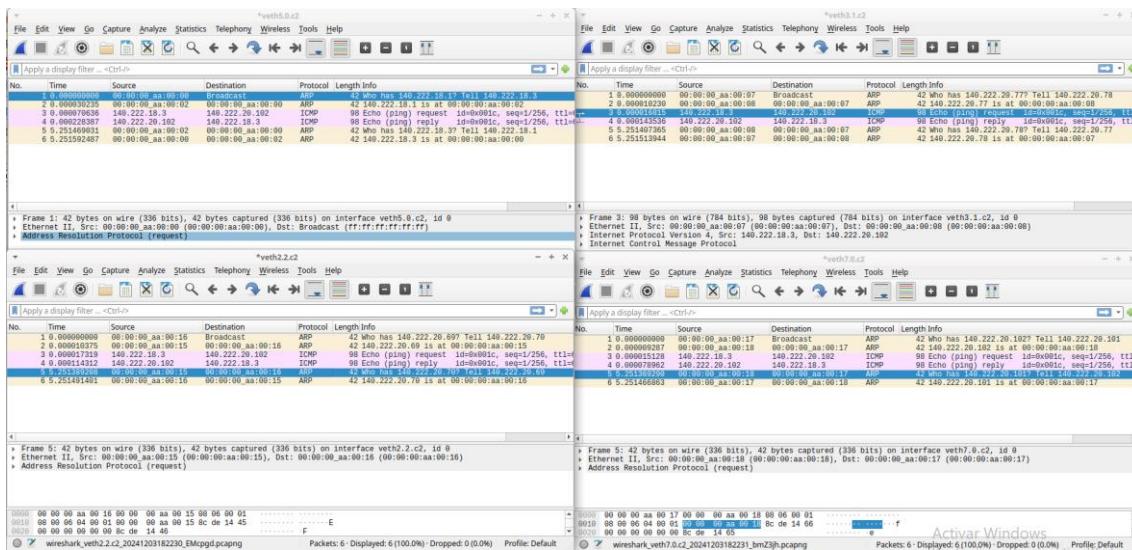


e) Realizar test con ping (ICMP) y traceroute para probar que funciona la topología.

Para este ejercicio realizamos varias pruebas de ping y traceroute y ya en otros incisos del trabajo mostramos el resultado de los mismos.

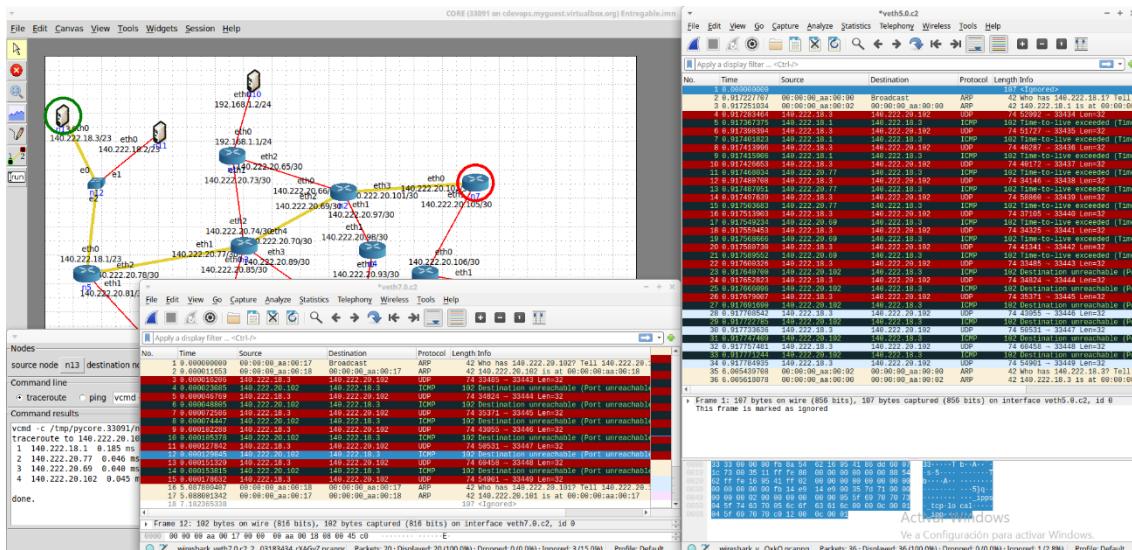
f) Capturar puntualmente el tráfico de n13 hacia n7 y analizar: ARP e ICMP.

Para este ejercicio capturamos el tráfico en los nodos n5, n3, n2 y n7 y vimos como fueron apareciendo los mensajes ARP que eran enviados para descubrir la dirección MAC del puerto del siguiente destino del paquete ICMP con el echo request y luego con el echo reply.



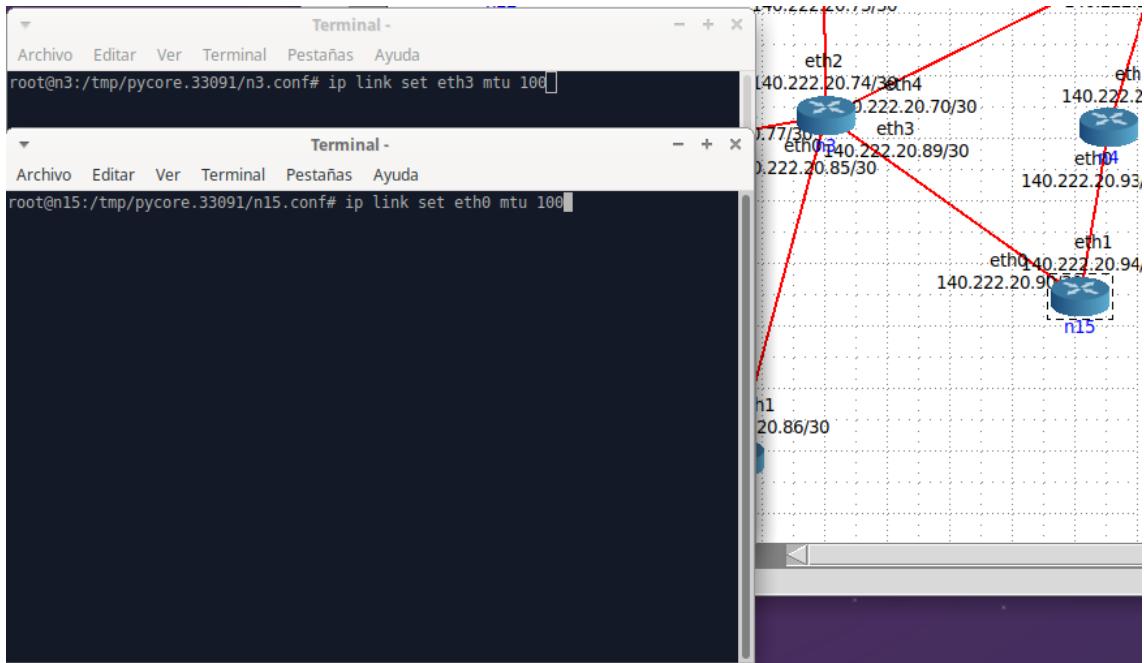
g) Realizar un traceroute entre los mismos equipos, capturar los mensajes.

La siguiente es una captura del tráfico en los nodos n5 y n7 para un traceroute entre los mismos. Se puede observar como por el nodo n5 llegan todos los mensajes del traceroute efectuado, incluso los que tienen un TTL corto para descubrir la topología, mientras que en el nodo n7 solo llegan los últimos mensajes que ya no son interrumpidos por un corto TTL, sino que indican un puerto inalcanzable en el mismo nodo n7.

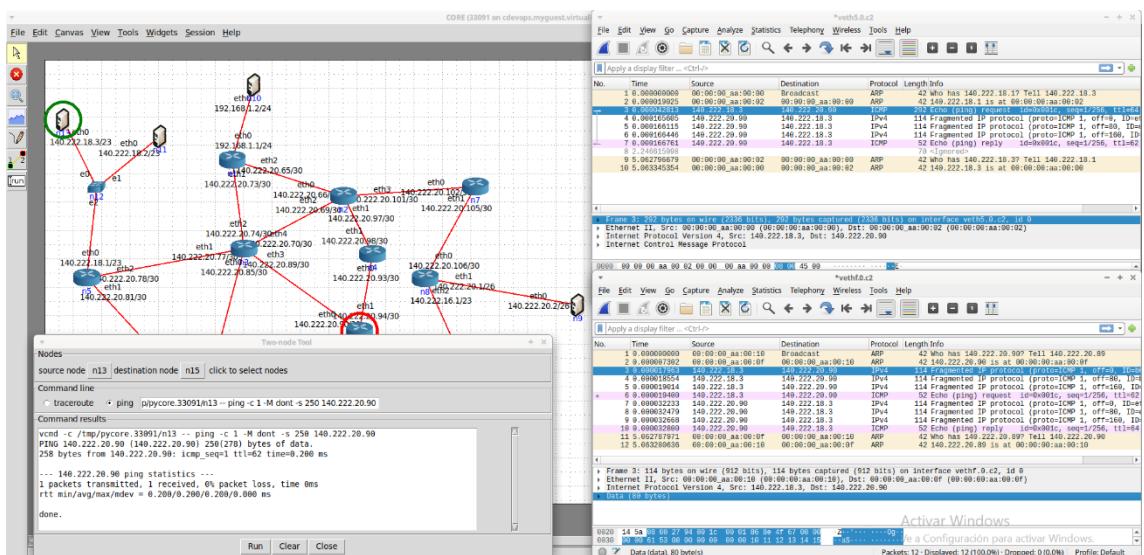


h) Alternativo: Modificar los MTU para ver la fragmentación.

Para modificar el MTU utilizamos el siguiente comando para cambiar a un MTU de 100 entre los nodos n3 y n15:

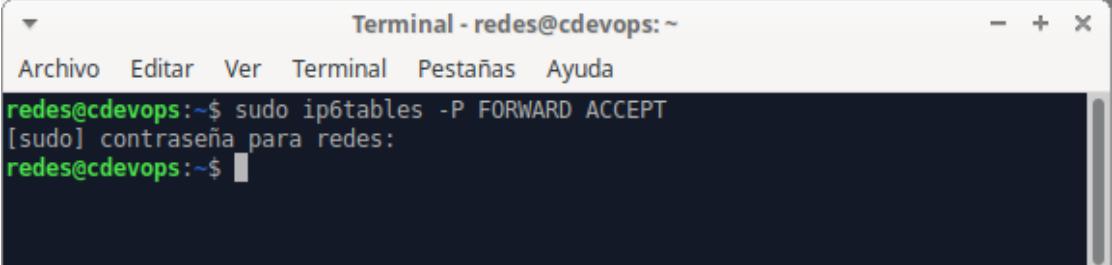


Y para probarlo realizamos un ping con el parámetro “-s 250” para enviar datos con un tamaño de 250 bytes. Además agregamos el parámetro “-M dont” para evitar problemas. Capturamos el tráfico de los paquetes en los nodos n5, que a la ida va completo y solo aparece fragmentada la respuesta que había pasado antes por ese camino, y el nodo n15 que tanto en el ping inicial como en la respuesta ya aparece fragmentado.



- i) Probar conectividad en las redes con IPv6 (por separado), capturar tráfico y analizar ICMPv6.

El siguiente comando lo realizamos para habilitar la conexión con IPv6.

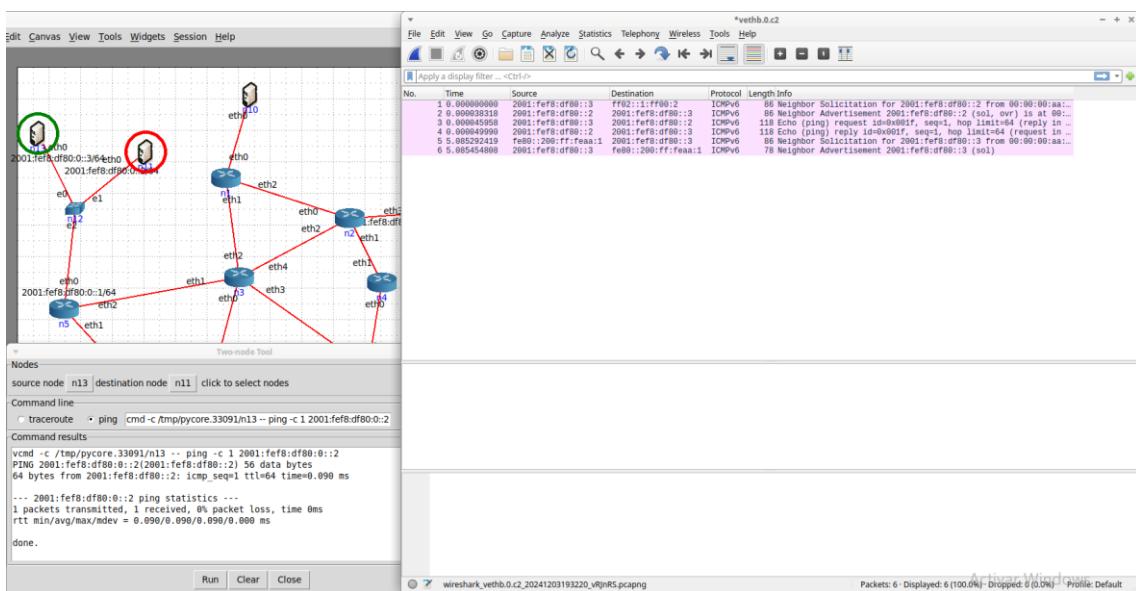


```
Terminal - redes@cdevops:~
```

Archivo Editar Ver Terminal Pestañas Ayuda

```
redes@cdevops:~$ sudo ip6tables -P FORWARD ACCEPT
[sudo] contraseña para redes:
redes@cdevops:~$
```

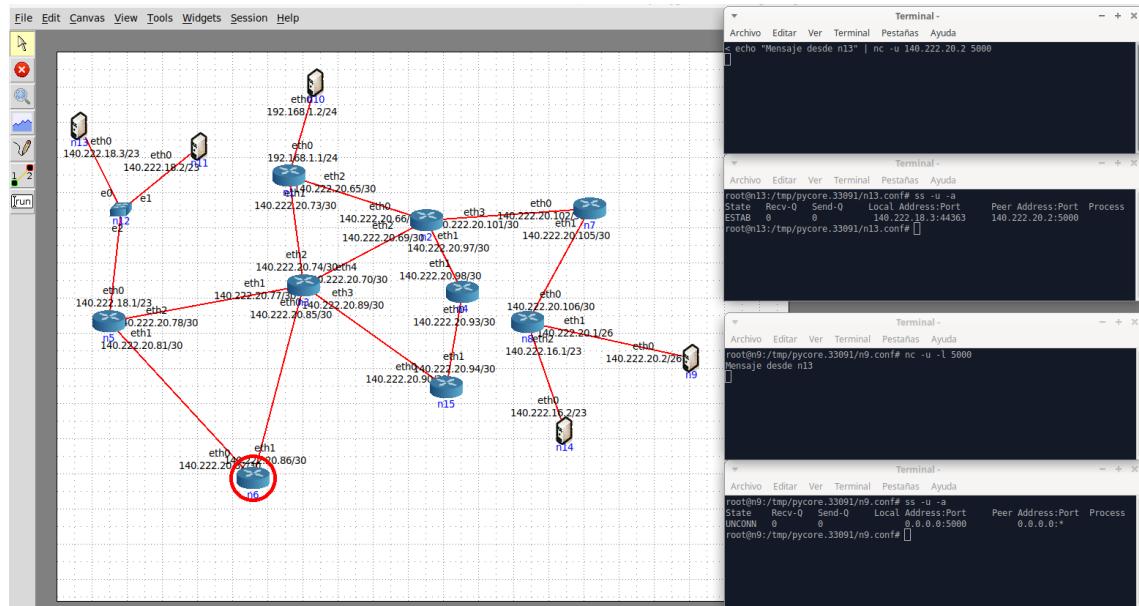
La siguiente es una captura del tráfico en un ping por IPv6 entre el nodo n13 y el nodo n11.



22. Ejercicios UDP a entregar. Correr sobre la topología de la figura 6:

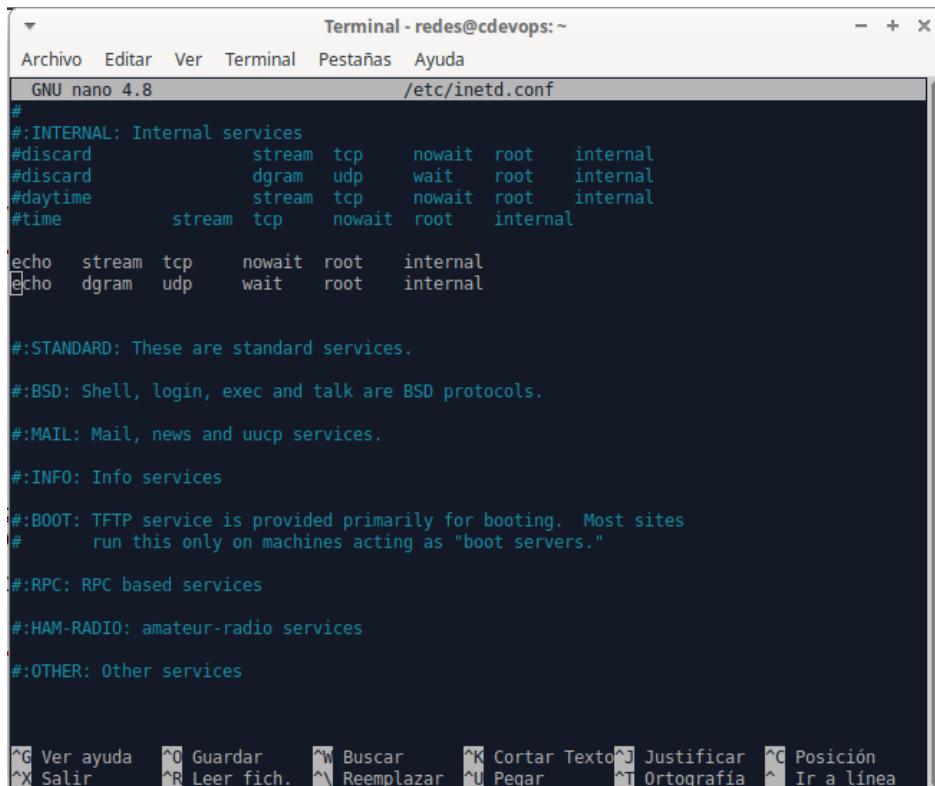
- a) Levantar un servicio UDP con la herramienta nc (netcat) en el host n9 y enviar información desde n13 usando el mismo comando. Ver el estado de los sockets en ambos extremos. ¿Qué significa el estado ESTABLISHED en UDP?

En este ejercicio se levantó un servicio de UDP utilizando netcat estableciendo una conexión entre los host n13 y n9, se configuro el host n9 como servidor UDP en el puerto 5000 permitiendo al host que espere mensajes por ese puerto. Luego, desde el host n13 se envía un mensaje al puerto en escucha de n9, como se muestra en la imagen el mensaje es recibido por n9 y se procede a analizar el estado de los sockets involucrados, se observan dos posibles estados como el de UNCONN indicando que el socket está en escucha y no recibió tráfico reciente, y el estado ESTABLISHED el cual indica tráfico reciente en el socket, por más que UDP sea protocolo sin conexión.



b) Levantar en el super daemon inetd o similar con el servicio UDP echo y probar con un cliente programado en el lenguaje de su elección mediante la API socket contra este servicio. Inspeccionar el estado. Ver de generar datos hacia el “servidor” desde más de un Nodo.

Primero revisamos la configuración de servicios donde se encontraba configurado el servicio UDP echo como se muestra en la imagen



The screenshot shows a terminal window titled "Terminal - redes@cdevops:~". The file being edited is "/etc/inetd.conf". The content of the file is as follows:

```
# :INTERNAL: Internal services
#discard stream tcp nowait root internal
#discard dgram udp wait root internal
#daytime stream tcp nowait root internal
#time stream tcp nowait root internal

echo stream tcp nowait root internal
echo dgram udp wait root internal

#:STANDARD: These are standard services.

#:BSD: Shell, login, exec and talk are BSD protocols.

#:MAIL: Mail, news and uucp services.

#:INFO: Info services

#:BOOT: TFTP service is provided primarily for booting. Most sites
#       run this only on machines acting as "boot servers."

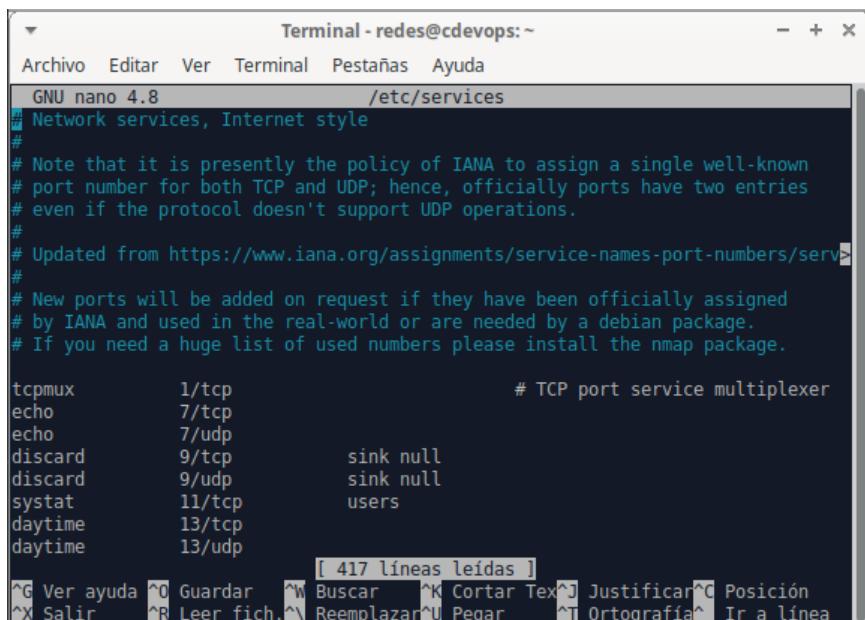
#:RPC: RPC based services

#:HAM-RADIO: amateur-radio services

#:OTHER: Other services
```

At the bottom of the terminal window, there are various keyboard shortcuts for navigating and editing the file.

Luego revisamos que el puerto asignado a este servicio sea el puerto 7, el cual estaba correctamente asignado



The screenshot shows a terminal window titled "Terminal - redes@cdevops:~". The file being edited is "/etc/services". The content of the file is as follows:

```
# Network services, Internet style
#
# Note that it is presently the policy of IANA to assign a single well-known
# port number for both TCP and UDP; hence, officially ports have two entries
# even if the protocol doesn't support UDP operations.
#
# Updated from https://www.iana.org/assignments/service-names-port-numbers/service-names-and-numbers
#
# New ports will be added on request if they have been officially assigned
# by IANA and used in the real-world or are needed by a debian package.
# If you need a huge list of used numbers please install the nmap package.

tcpmux      1/tcp          # TCP port service multiplexer
echo        7/tcp
echo        7/udp
discard     9/tcp      sink null
discard     9/udp      sink null
sysstat    11/tcp      users
daytime     13/tcp
daytime     13/udp
```

At the bottom of the terminal window, there are various keyboard shortcuts for navigating and editing the file. A message "[417 líneas leídas]" is displayed at the bottom center.

Posteriormente se desarrollaron el respectivo código para los clientes (ClienteUDP.py) y para el servidor (ServidorUDP.py) utilizando Python, donde se pudo validar que inetd soporta múltiples solicitudes en simultáneo ya que se ejecutaron dos clientes en n13 y en n14 y los mensajes llegan al destino del servidor.

ClienteUDP.py

```
import socket

def udp_echo_client(server_ip, server_port):
    # Crear un socket UDP
    sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
    # Enviar mensaje al servidor
    message = "Hola, servidor UDP Echo"
    print(f"Enviando: {message}")
    sock.sendto(message.encode(), (server_ip, server_port))
    # Recibir respuesta del servidor
    response, server = sock.recvfrom(1024)
    print(f"Recibido: {response.decode()}")
    sock.close()

# Configurar dirección y puerto del servidor
server_ip = "140.222.20.2" # Cambiar por la IP del servidor
server_port = 7             # Puerto estándar UDP Echo

udp_echo_client(server_ip, server_port)
```

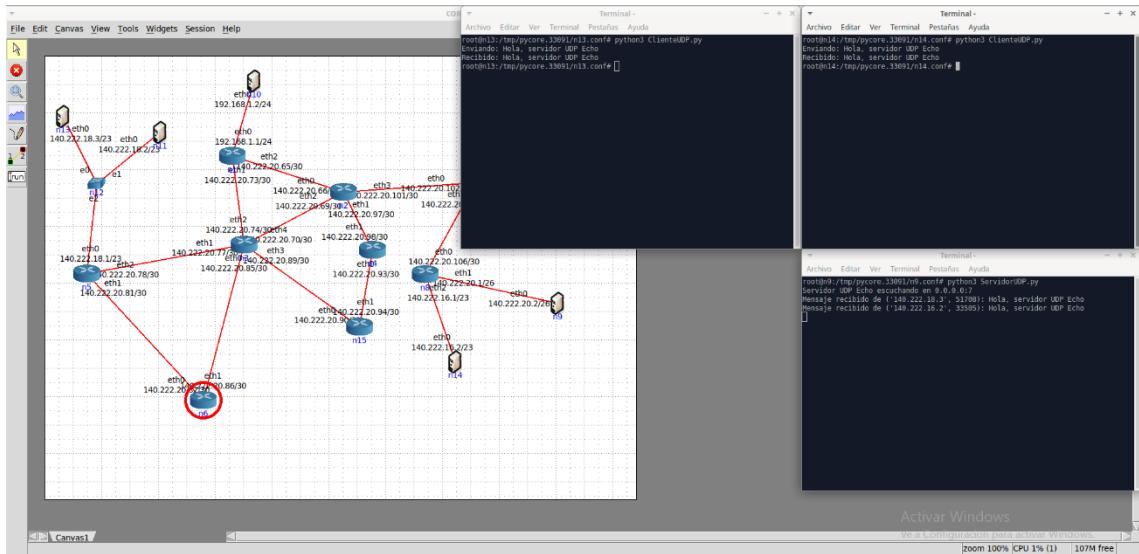
ServidorUDP.py

```
import socket

def udp_echo_server(host, port):
    # Crear socket UDP
    sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
    sock.bind((host, port))
    print(f"Servidor UDP Echo escuchando en {host}:{port}")
    while True:
        # Recibir datos
        data, addr = sock.recvfrom(1024)
        print(f"Mensaje recibido de {addr}: {data.decode()}")
        # Enviar respuesta
        sock.sendto(data, addr)

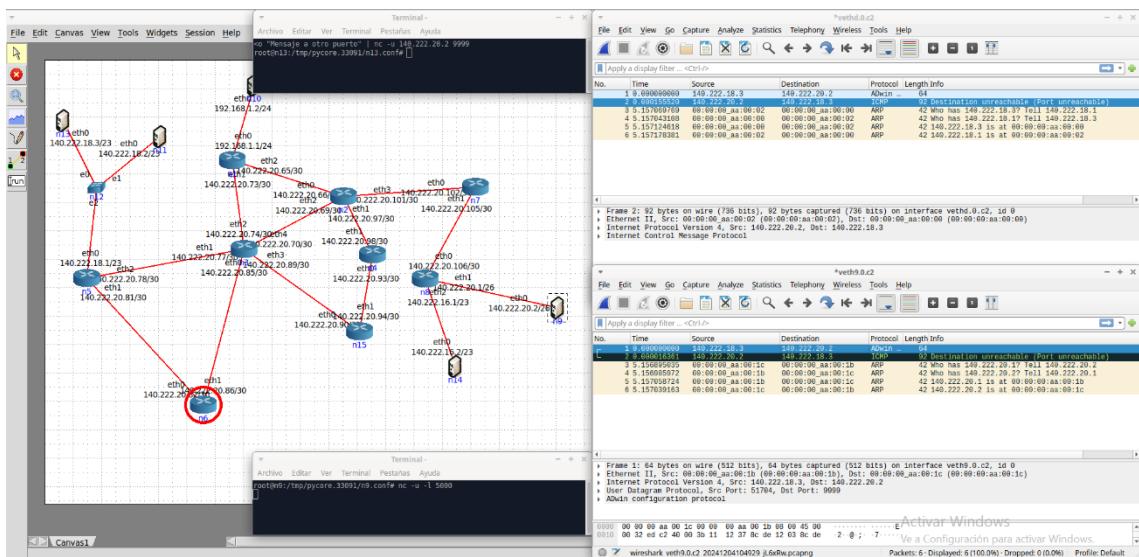
# Configura la IP del servidor y el puerto (puedes usar 7)
server_ip = "0.0.0.0" # Escucha en todas las interfaces
server_port = 7       # Puerto estándar UDP Echo

udp_echo_server(server_ip, server_port)
```

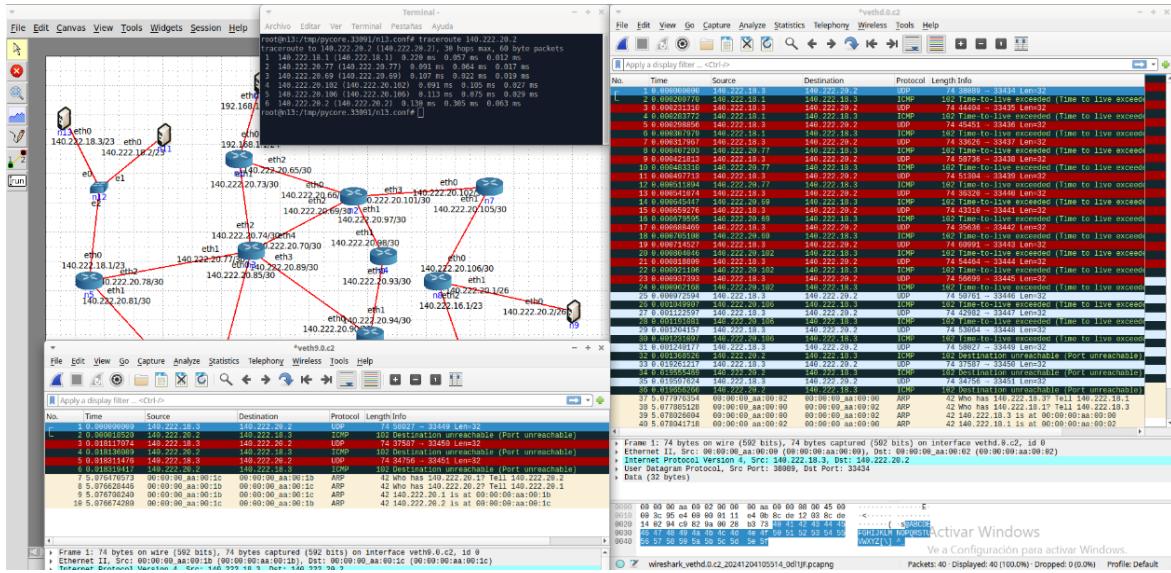


c) Enviar información desde n13 a n9 a un port UDP donde no existe un proceso esperando por recibir datos. ¿Cómo notifica el stack TCP/IP de este hecho? Investigue la herramienta traceroute que ports utiliza y cómo usa estos mensajes (Ver ejercicio de IP con ruteo estático).

En este ejercicio, se envía un mensaje desde n13 a un puerto n9 donde no existía un proceso escuchando, analizando el respectivo Wireshark (Imagen1) se puede ver que el stack TCP/IP genera una respuesta de tipo ICMP Destination Unreachable Port Unreachable notificando a n13 que el puerto no está disponible



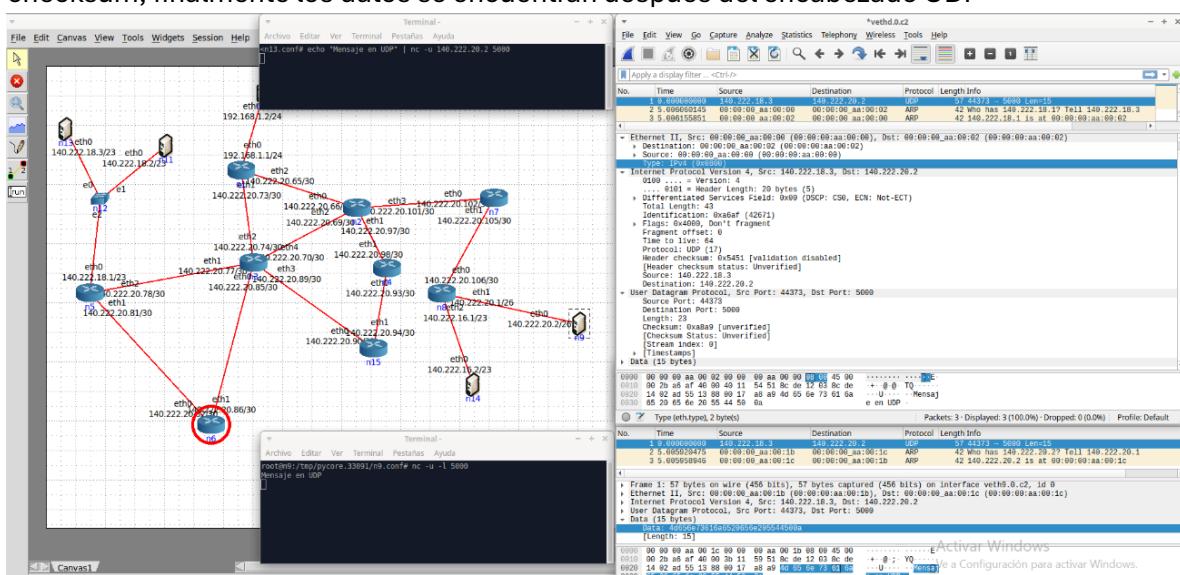
Luego analizamos traceroute la herramienta que UDP utiliza para descubrir la ruta que siguen los paquetes hacia el destino, inicia enviando un paquete UDP a un puerto aleatorio en el host destino, si no hay proceso escuchando el destino responde con un ICMP como mencionamos anteriormente, si durante el proceso debe realizar un salto en la ruta envia un mensaje ICMP Time Exceeded para indicar que el TTL se hizo 0, y traceroute aumenta progresivamente es valor



d) Para las pruebas anteriores capturar tráfico y ver el formato de los datagramas

UDP y como se encapsulan en IP.

En la imagen se puede observar un paquete enviado desde n13 hacia n9 en el puerto 5000 corresponde a un paquete UDP que se encapsula dentro de un paquete IP (protocolo IPv4), se puede observar los encabezados IP como la longitud de la cabecera, protocolo, ip origen, ip destino, TTL, entre otros. Por otro lado, se observan los campos del encabezado de UDP como el puerto origen (cliente), puerto destino (servidor), longitud y checksum, finalmente los datos se encuentran después del encabezado UDP



23. Ejercicios TCP a entregar. Correr sobre la topología de la figura 6 (continuación de lo que venían realizando en la topología en la práctica anterior):

a) Programar en el lenguaje de su elección mediante la API socket un servidor TCP que escuche conexiones en el puerto 9000 y que los datos que reciba los descarte. Correr en el nodo n9 y enviar datos desde otro nodo usando la herramienta nc (netcat) o telnet.

Decidimos utilizar el lenguaje Python para el código del servidor, este mismo inicia un servicio de escucha en el puerto 9000 que al aceptar una conexión TCP descarte todos los mensajes recibidos.

```
import socket

def tcp_server(host, port):
    # Crear un socket TCP
    server_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    server_socket.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)

    # Asociar el socket a la dirección y puerto
    server_socket.bind((host, port))
    server_socket.listen(5)
    print(f"Servidor TCP escuchando en {host}:{port}")

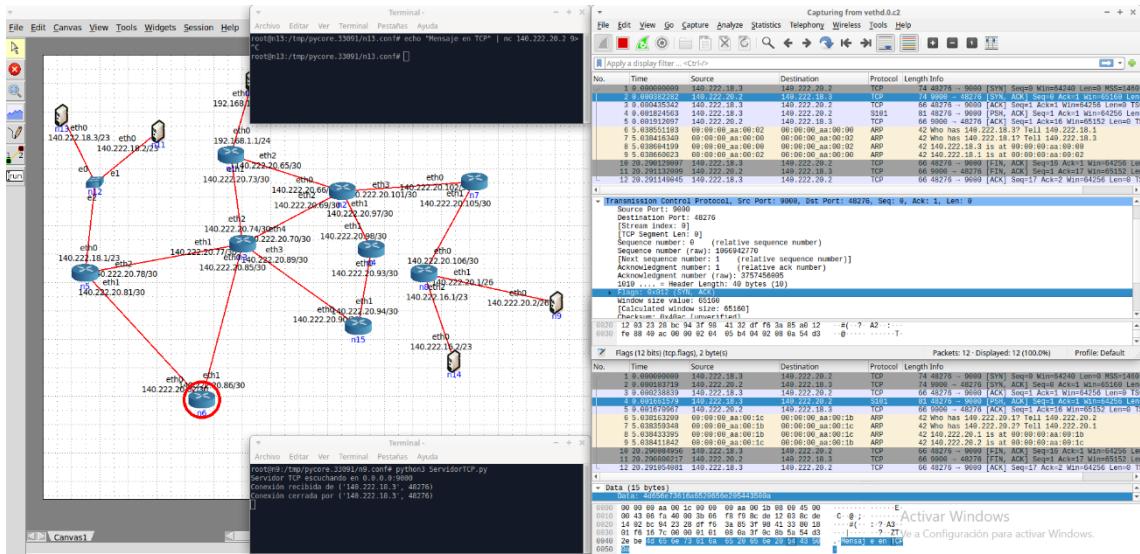
    while True:
        # Aceptar conexión entrante
        client_socket, client_address = server_socket.accept()
        print(f"Conexión recibida de {client_address}")

        # Recibir y descartar datos
        while True:
            data = client_socket.recv(1024) # Recibir hasta 1024 bytes
            if not data: # Si no hay datos, el cliente cerró la conexión
                break
        print(f"Conexión cerrada por {client_address}")
        client_socket.close()

# Configuración del servidor
HOST = "0.0.0.0" # Escuchar en todas las interfaces
PORT = 9000

tcp_server(HOST, PORT)
```

En la siguiente imagen se ve una captura de los mensajes recibidos y enviados en una comunicación entre el nodo n13, que envía el mensaje “Mensaje en TCP” al nodo n9 que descarta el mismo por el código desarrollado.



b) En el nodo n9 levantar con el super daemon inetd algunos servicios extras, como tcp echo, discard y otros. Chequear los servicios TCP y UDP activos.

Para agregar servicios al daemon inetd modificamos el archivo /etc/inetd.conf y agregamos los servicios discard y chargen tanto para UDP como TCP y los servicios daytime y time para TCP. Vimos los servicios activos con el comando “netstat -tuln”

```

Terminal - redes@cdevops: ~
Archivo Editar Ver Terminal Pestañas Ayuda
GNU nano 4.8          /etc/inetd.conf
# /etc/inetd.conf: see inetd(8) for further informations.
#
# Internet superserver configuration database
#
# Lines starting with "#:LABEL:" or "#<off>" should not
# be changed unless you know what you are doing!
#
# If you want to disable an entry so it isn't touched during
# package updates just comment it out with a single '#' character.
#
# Packages should modify this file by using update-inetd(8)
#
# <service_name> <sock_type> <proto> <flags> <user> <server_path> <args>
#
#:INTERNAL: Internal services
echo stream tcp nowait root internal
echo dgram udp wait root internal
discard stream tcp nowait root internal
discard dgram udp wait root internal
chargen stream tcp nowait root internal
chargen dgram udp wait root internal
daytime stream tcp nowait root internal
time stream tcp nowait root internal

#:STANDARD: These are standard services.

#:BSD: Shell, login, exec and talk are BSD protocols.

#:MAIL: Mail, news and uucp services.

root@n9:/tmp/pycore.33091/n9.conf# service utilitools-inetd status
* inetd is running
root@n9:/tmp/pycore.33091/n9.conf# netstat -tuln
Conexiones activas de Internet (solo servidores)
Proto Recib Enviad Dirección local Direccion remota Estado
tcp 0 0 0.0.0.19 0.0.0.* ESCUCHAR
tcp 0 0 0.0.0.0:7 0.0.0.* ESCUCHAR
tcp 0 0 0.0.0.0:13 0.0.0.* ESCUCHAR
tcp 0 0 0.0.0.0:9 0.0.0.* ESCUCHAR
tcp 0 0 0.0.0.0:37 0.0.0.* ESCUCHAR
udp 0 0 0.0.0.0:7 0.0.0.* ESCUCHAR
udp 0 0 0.0.0.0:9 0.0.0.* ESCUCHAR
udp 0 0 0.0.0.0:19 0.0.0.* ESCUCHAR
root@n9:/tmp/pycore.33091/n9.conf# 
```

c) Desde el nodo n13 realizar una conexión TCP y enviar datos mediante un programa cliente de su elección al servicio discard, y al echo. Capturar el tráfico con la herramienta tcpdump o wireshark y analizar la cantidad de segmentos, los flags utilizados y las opciones extras que llevan los encabezados tcp.

Se implemento un código en Python que genera dos conexiones TCP, una que conecta con el servicio de discard en el puerto 9 y otro que conecta con el servicio echo en el puerto 7.

```
import socket

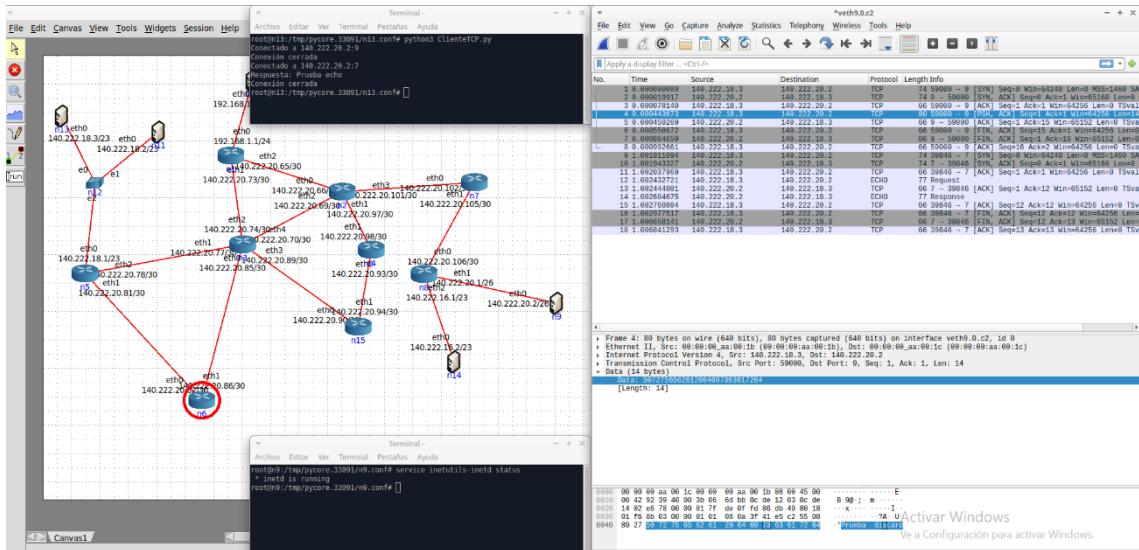
import time # Importar el módulo time para usar sleep

def tcp_client(host, port, message):
    with socket.socket(socket.AF_INET, socket.SOCK_STREAM) as client_socket:
        client_socket.connect((host, port))
        print(f"Conectado a {host}:{port}")
        client_socket.sendall(message.encode())
        if port == 7: # Solo esperar respuesta si el servicio es echo
            response = client_socket.recv(1024)
            print(f"Respuesta: {response.decode()}")
        print("Conexión cerrada")

# Configuración del cliente
HOST = "140.222.20.2" # Cambiar por la IP del nodo n9
PORT_DISCARD = 9
PORT_ECHO = 7

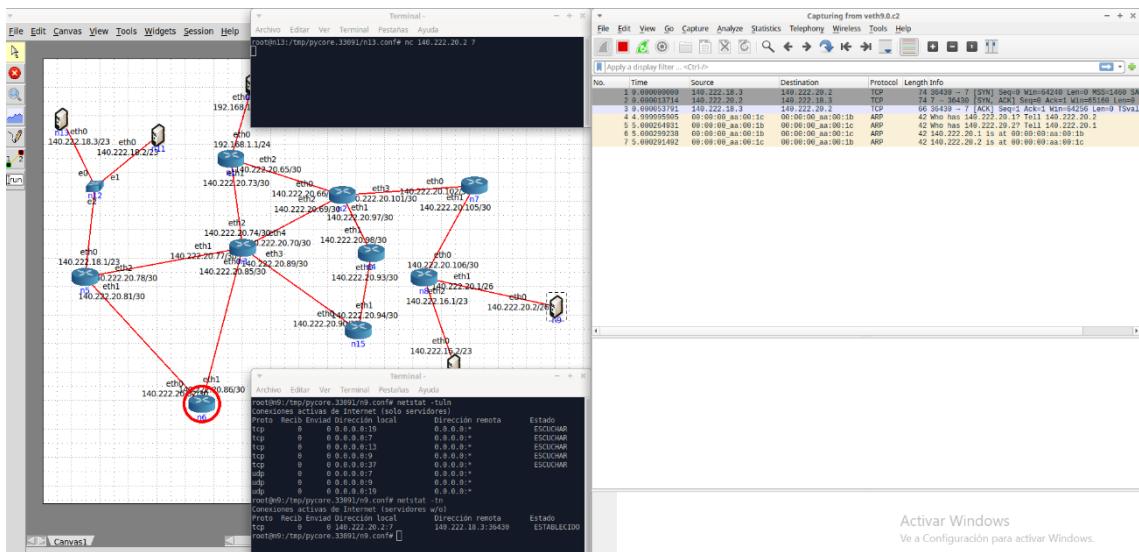
# Enviar datos al servicio discard
tcp_client(HOST, PORT_DISCARD, "Prueba discard")
# Esperar un segundo antes de enviar el mensaje al servicio echo
time.sleep(1) # Pausa la ejecución durante 1 segundo
# Enviar datos al servicio echo
tcp_client(HOST, PORT_ECHO, "Prueba echo")
```

En la siguiente captura se pueden ver las dos conexiones TCP generadas, estas estan separadas por un tiempo de 1 segundo para una mejor interpretación de los mensajes. En cada conexión el nodo 13 comienza enviando un mensaje con el flag SYN y luego el nodo 9 responde con un mensaje con los flags SYN y ACK, luego de esto el nodo n13 envia un mensaje con un flag ACK y sigue prosigue con el envio de los datos. Al finalizar las conexiones el nodo n13 envia un mensaje con los flags FIN y ACK y el nodo n9 responde con un mensaje con estos mismos flags. Para terminar con un mensaje de n13 con el flag ACK.



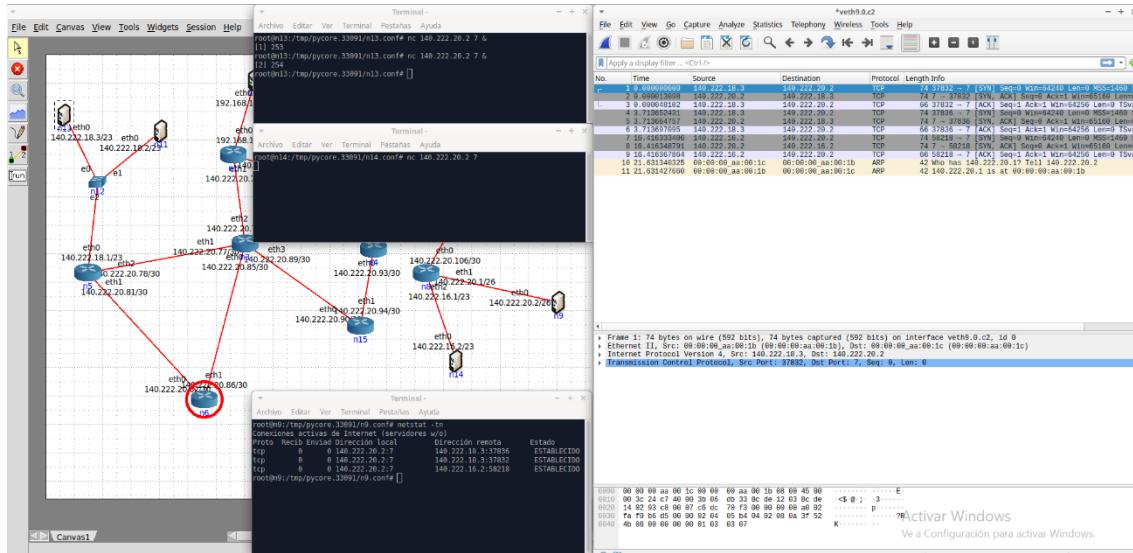
d) Sin cerrar las conexiones chequear los servicios activos y ver los Estados.

Para este ejercicio iniciamos una conexión TCP y no enviamos ningún mensaje y luego utilizamos el comando “netstat -tuln” para ver los servicios activos.



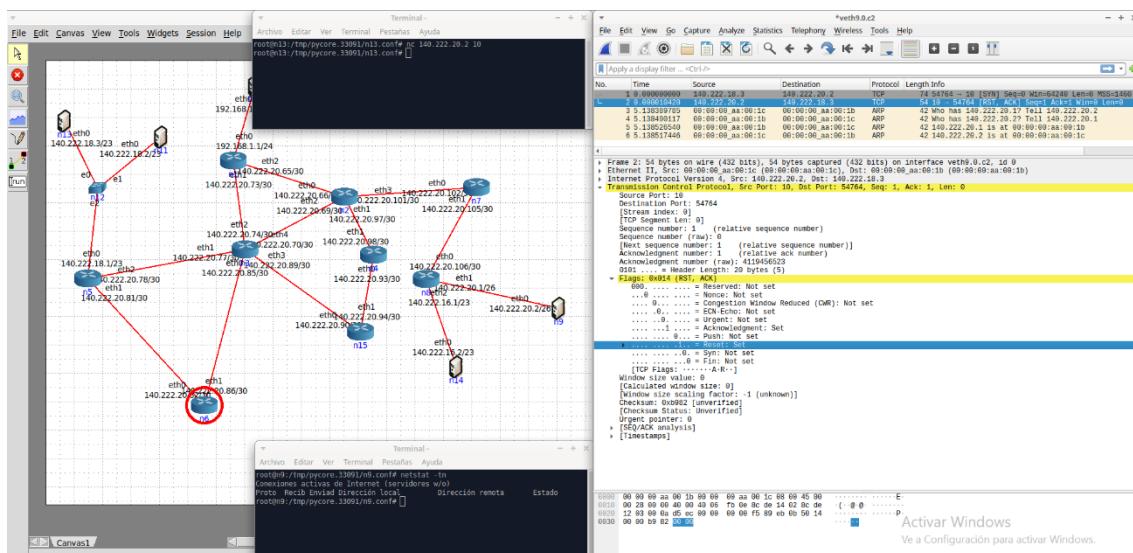
e) Generar nuevas conexiones hacia el nodo n9 e inspeccionar los estados. Por ejemplo realizar varias conexiones simultáneas al servicio tcp echo desde el mismo origen y desde otros nodos.

Para este ejercicio creamos dos conexiones TCP desde el nodo n13 y una desde el nodo n14, todas al puerto 7 del nodo n9. Capturamos los mensajes recibidos y enviados por el nodo n9 con el servicio wireshark.



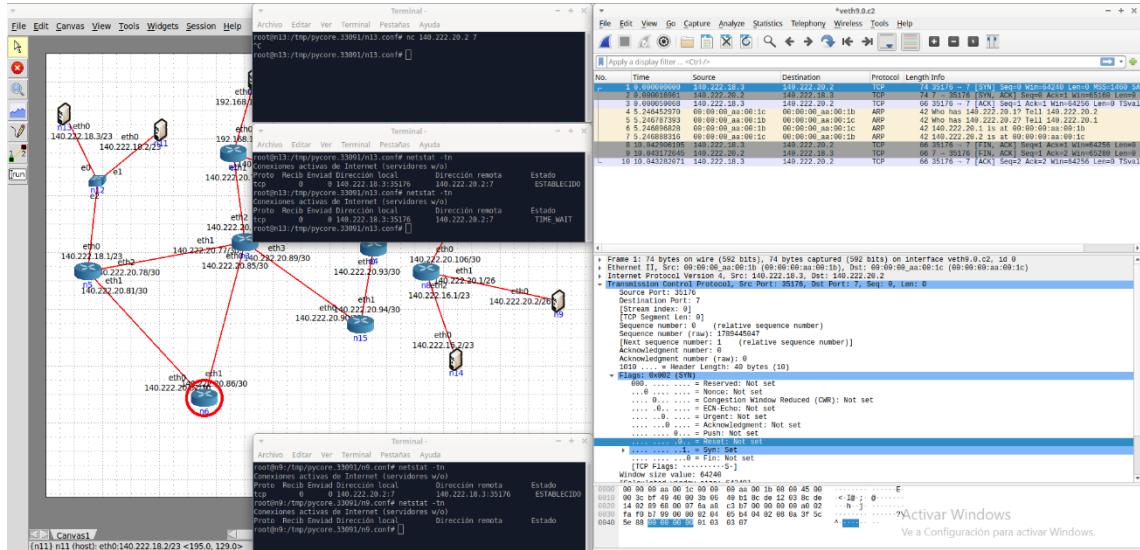
f) Intentar generar conexiones a un puerto donde no existe un proceso esperando por recibir datos. ¿Cómo notifica TCP de este hecho (ver flags)?

En este ejercicio intentamos establecer una conexión TCP con el puerto 10 del nodo n9, que no tenía ningún servicio esperando datos. La respuesta de n9 fue desde el puerto 10 y contenía el flag de ACK y el flag de RST para finalizar la conexión.



g) Cerrar las conexiones y ver el estado de los servicios en ambos lados. ¿En qué estado queda el que hace el cierre activo?

El servicio del nodo que hace el cierre activo queda en estado TIME_WAIT, esperando por si se reciben paquetes atrasados.



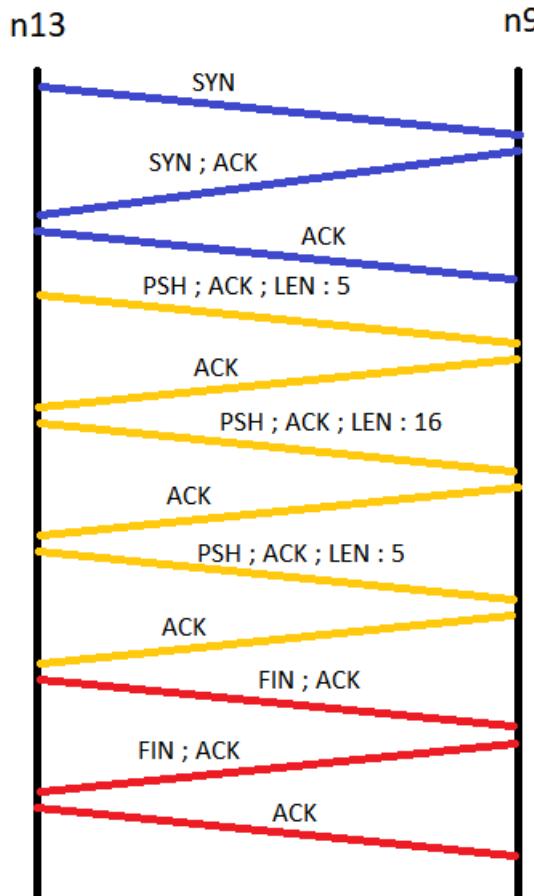
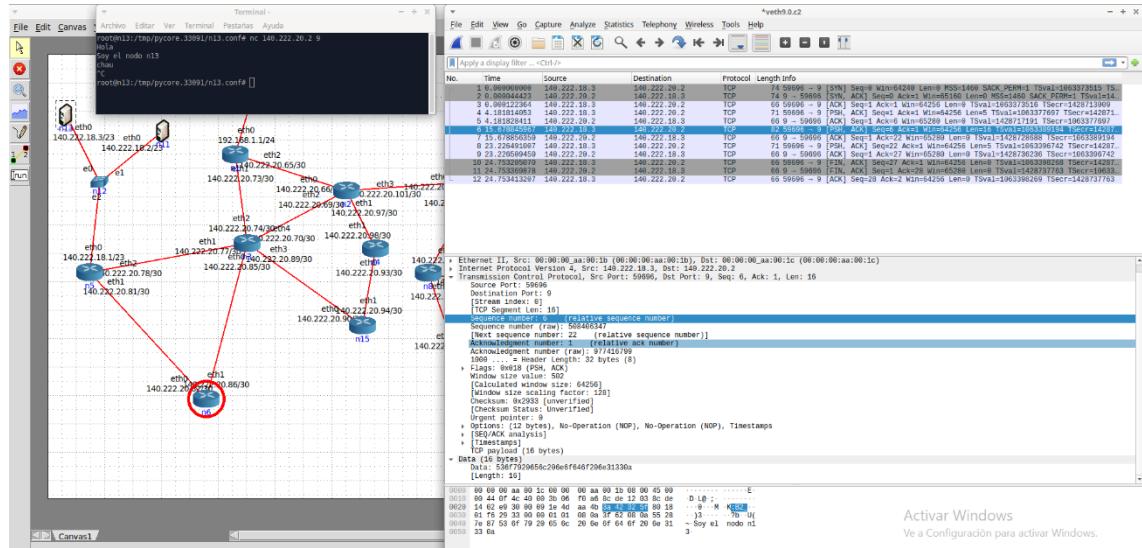
h) Observando la captura indicar la cantidad de segmentos y los flags utilizados.

¿Con cuántos segmentos se cerró la conexión? ¿Existen otras variantes de cierre?

La conexión se cerro con 3 segmentos con los flags: [FIN, ACK] ; [FIN, ACK] ; [ACK]
La otra variante de cierre es de 4 segmentos y se usa si el que no realiza el primer cierre quiere seguir enviando datos. Los flags se envían de la siguiente forma:

[FIN, ACK] ; [ACK] ; [FIN, ACK] ; [ACK]

i) Hacer un diagrama de los segmentos intercambiados con los números de secuencia absolutos para una de las sesiones TCP (Se puede usar la herramienta wireshark u otra).



SN	ACK
0	-
0	1
1	1
1	1
1	6
6	1
1	22
22	1
1	27
27	1
1	28
28	2

j) Alternativo: Realizar una conexión mediante nc indicando un puerto específico para el cliente. Luego cerrar la conexión desde el cliente e intentar abrirla nuevamente. ¿En qué estado está el socket? Investigar valor del 2MSL en la plataforma sobre la cual está haciendo los tests.

En este ejercicio usamos el comando “netstat -tuln” para ver el estado de los servicios en cada instante.

- Con la conexión establecida
- Con la conexión cerrada
- Con una nueva conexión abierta
- Pasados dos veces el tiempo de 2MSL para que la primera conexión cerrada no aparezca en estado TIME_WAIT.

El tiempo de 2MSL en esta plataforma es de 60.

