Facultad de INFORMÁTICA | UNIVERSIDAD NACIONAL DE LA PLATA

Redes de Datos II

Trabajo Práctico Integrador

Grupo 0T:
Carrer, Estanislao 02501/2
Guillen, Isidro 02610/6
Ramírez Tolentino, Fernando 01964/8

2025 - Ingeniería en Computación.

Recursos asignados

Para el desarrollo del Trabajo Práctico Integrador, al grupo se le asignaron dos bloques de direcciones IP, una IPv4 y una IPv6:

- **Dirección IPv4:** 46.90.16.0/21.
- **Dirección IPv6:** 2001:db81::/32.

Ejercicio 12 - Práctica 3

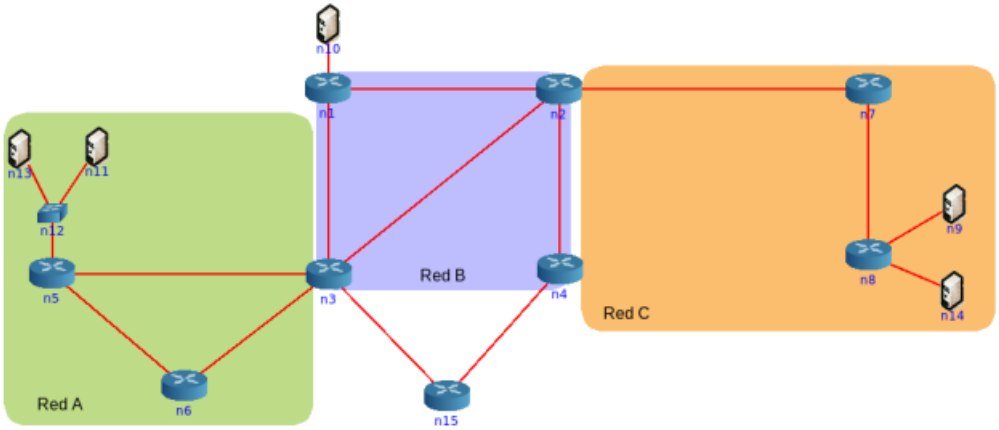


Figura 1. Diagrama de topología para TPI

Direccionamiento IPv4

Red de n14 (conectada a n8): 500 hosts

Se necesita que $2^n - 2 \geq 500$, siendo n los bits de host, por lo tanto, como $2^9 - 2 = 510$, los bits de host serán 9. Mientras que la máscara será: 255.255.254.0 = /23.

Red de n11 y n13 (conectadas a n5): 328 hosts

Se necesita que $2^n - 2 \geq 328$, siendo n los bits de host, por lo tanto, como $2^9 - 2 = 510$, los bits de host serán 9. Mientras que la máscara será: 255.255.254.0 = /23.

Red de n9 (conectada a n8): 40 hosts

Se necesita que $2^n - 2 \geq 40$, siendo n los bits de host, por lo tanto, como $2^6 - 2 = 62$, los bits de host serán 6. Mientras que la máscara será: 255.255.255.192 = /26 .

Red de n10 (conectada a n1): 2 hosts

Se necesita que $2^n - 2 \geq 2$, siendo n los bits de host, por lo tanto, como $2^2 - 2 = 2$, los bits de host serán 2. Mientras que la máscara será: 255.255.255.252 = /30 .

Enlaces P2P (Routers)

Se necesita que $2^n - 2 \geq 2$, siendo n los bits de host, por lo tanto, como $2^2 - 2 = 2$, los bits de host serán 2. Mientras que la máscara será: 255.255.255.252 = /30 .

La siguiente tabla (Tabla 1) detalla el esquema de direccionamiento resultante, especificando nombre de la red, dirección de red, máscara, rango de hosts asignables y dirección de broadcast en función de los requisitos de cada subred.

Tabla 1. Direccionamiento IPv4

Nombre	Requerimiento	Máscara	Dirección	Rango	Broadcast
LAN n14	500 hosts	/23	46.90.22.0	.22.1-.23.254	.23.255
LAN n11/13	328 hosts	/23	46.90.20.0	.20.1-.21.254	.21.255
LAN n9	40 hosts	/26	46.90.19.194	.19.195-.19.254	.19.255
LAN n10	2 hosts	/26	46.90.19.190	.19.191-.19.192	.19.193
P2P n5-n6	2 hosts	/30	46.90.19.184	.19.185-.19.186	.19.187
P2P n5-n3	2 hosts	/30	46.90.19.180	.19.181-.19.183	.19.183
P2P n6-n3	2 hosts	/30	46.90.19.176	.19.177-.19.178	.19.179
P2P n3-n15	2 hosts	/30	46.90.19.172	.19.173-.19.174	.19.175

P2P n4-n15	2 hosts	/30	46.90.19.168	.19.169-.19.179	.19.171
P2P n1-n3	2 hosts	/30	46.90.19.164	.19.165-.19.166	.19.167
P2P n1-n2	2 hosts	/30	46.90.19.160	.19.161-.19.162	.19.163
P2P n2-n3	2 hosts	/30	46.90.19.156	.19.157-.19.158	.19.159
P2P n2-n4	2 hosts	/30	46.90.19.152	.19.153-.19.154	.19.155
P2P n2-n7	2 hosts	/30	46.90.19.148	.19.149-.19.150	.19.151
P2P n7-n8	2 hosts	/30	46.90.19.144	.19.145-.19.146	.19.147

Para visualizar la implementación del esquema de direccionamiento calculado, se presenta a continuación el diagrama de la topología con las direcciones IPv4 asignadas a cada interfaz y segmento de red.

La prueba del funcionamiento de la topología puede realizarse a partir del archivo **Ejercicio12-Practica3.imn.**

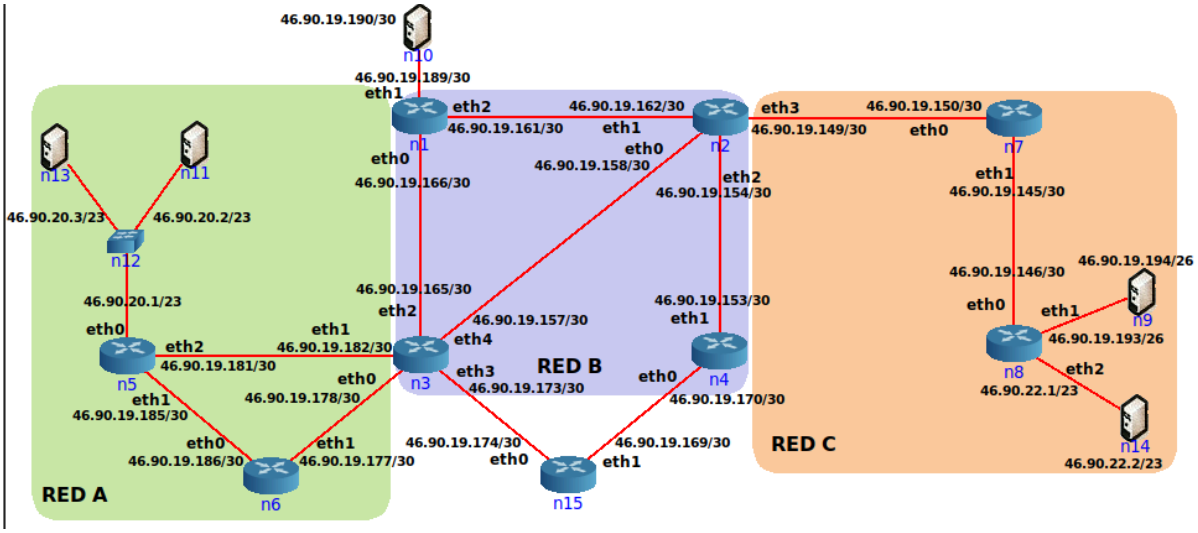


Figura 2. Direcciones IPv4 asignadas

Direccionamiento IPv6

Dado el vasto espacio disponible en el bloque asignado 2001:db81::/32, se adoptó la práctica estándar de asignar un prefijo /64 a cada segmento de red, independientemente de la cantidad de hosts. Se subdividió el bloque principal en prefijos de longitud /48 para identificar las áreas principales de la topología:

- **Red A:** Se le asignó el identificador **000A**, resultando en el bloque **2001:db81:000A::/48**.
- **Red C:** Se le asignó el identificador **000C**, resultando en el bloque **2001:db81:000C::/48**.

La siguiente tabla (Tabla 2) detalla la distribución de los prefijos IPv6 configurados en la topología:

Tabla 2. Direccionamiento IPv6

Área	Descripción de la Red	Prefijo de Sitio (/48)	Prefijo de Red Asignado (/64)
Red A	LAN n5, n11, n13	2001:db81:000A::	2001:db81:000A:0001::/64
Red C	Enlace n2 - n7	2001:db81:000C::	2001:db81:000C:0001::/64
Red C	Enlace n7 - n8	2001:db81:000C::	2001:db81:000C:0002::/64
Red C	LAN n9	2001:db81:000C::	2001:db81:000C:0003::/64
Red C	LAN n14	2001:db81:000C::	2001:db81:000C:0004::/64

Para visualizar la implementación de este esquema, la siguiente figura (Figura 3) ilustra la asignación de direcciones IPv6 a cada host y router dentro de las áreas designadas.

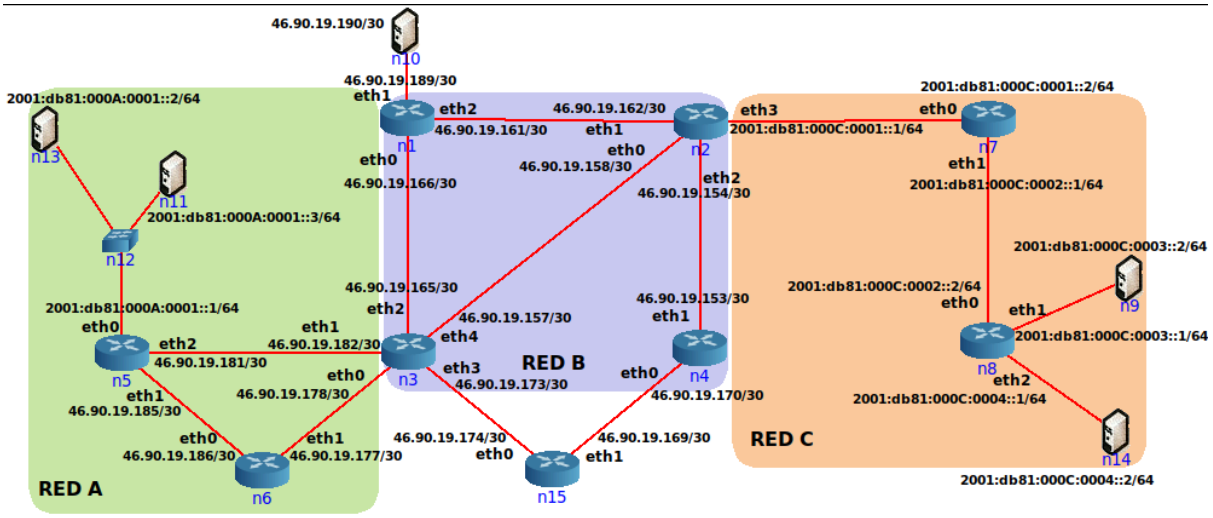


Figura 3. Direcciones IPv6 asignadas

Configuración de NAT y Direccionamiento Privado

Se asignó el bloque privado **192.168.1.0/24** a la interfaz LAN del router n1 y al host n10.

Para permitir que el host n10 tenga conectividad con el resto de la red, se configuró Source NAT (Masquerade) en el router de borde n1. El comando aplicado utiliza iptables para enmascarar el tráfico saliente, haciendo que los paquetes de la red privada parezcan originarse desde la interfaz pública de n1:

iptables -t nat -A POSTROUTING -s 192.168.1.0/24 -j MASQUERADE

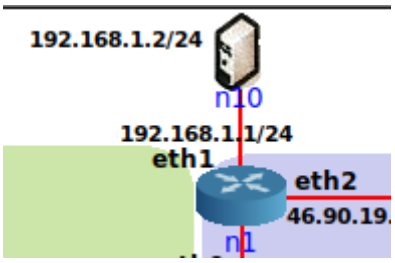


Figura 4. Dirección privada

La prueba del funcionamiento de esta configuración podrá realizarse a partir del archivo **Ejercicio12-Practica3-Alternativos.imn.**

Interacción ARP e ICMP

Para verificar el funcionamiento de la topología y analizar el proceso de encapsulamiento y resolución de direcciones, se realizó una captura de tráfico durante una prueba de conectividad (Ping) desde el host n13 hacia el router n7. La captura puede visualizarse desde wireshark en el archivo **Practica3-ejercicio12-incisoE.pcapng.**

Ruta y Saltos (Traceroute)

Se ejecutó un traceroute desde el host n13 hacia n14. La captura que se utilizó para el análisis puede visualizarse en el archivo **Practica3-ejercicio12-incisoF.pcapng.**

Fragmentación por Modificación de MTU

Se modificó el MTU del enlace punto a punto entre el router n2 y el router n7, reduciéndolo de su valor estándar (1500 bytes) a 500 bytes. Esto obliga a cualquier paquete superior a este tamaño a ser fragmentado para poder atravesar dicho enlace.

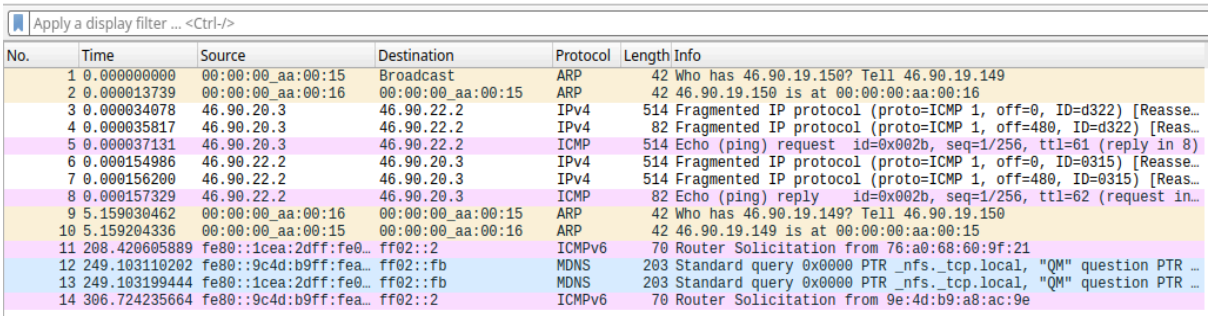
Los comandos aplicados en las interfaces correspondientes fueron:

- En Router n7 (Interfaz eth0): **ip link set dev eth0 mtu 500**
- En Router n2 (Interfaz eth3): **ip link set dev eth3 mtu 500**
- Se verificó la configuración en ambos extremos mediante el comando ip link show.

Desde el host n13, se generó tráfico ICMP hacia n14 con un tamaño de carga útil superior al MTU configurado. El comando utilizado fue:

ping 46.90.22.2 -s 1000 -c 1 -M dont

- **-s 1000:** Define un tamaño de datos (payload) de 1000 bytes. Sumando las cabeceras IP e ICMP (20 + 8 bytes), el paquete total es de 1028 bytes, lo cual excede el límite de 500 bytes del enlace n2-n7.
- **-M dont:** Indica explícitamente que no se establezca el bit "Don't Fragment" (DF), permitiendo así que los routers intermedios fragmenten el paquete si es necesario.



Apply a display filter ... <Ctrl-/>

No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000000	00:00:00_aa:00:15	Broadcast	ARP	42	Who has 46.90.19.150? Tell 46.90.19.149
2	0.000013739	00:00:00_aa:00:16	00:00:00_aa:00:15	ARP	42	46.90.19.150 is at 00:00:00:aa:00:16
3	0.000034078	46.90.20.3	46.90.22.2	IPv4	514	Fragmented IP protocol (proto=ICMP 1, off=0, ID=d322) [Reasse…
4	0.000035817	46.90.20.3	46.90.22.2	IPv4	82	Fragmented IP protocol (proto=ICMP 1, off=480, ID=d322) [Reas…
5	0.000037131	46.90.20.3	46.90.22.2	ICMP	514	Echo (ping) request id=0x002b, seq=1/256, ttl=61 (reply in 8)
6	0.000154986	46.90.22.2	46.90.20.3	IPv4	514	Fragmented IP protocol (proto=ICMP 1, off=0, ID=0315) [Reasse…
7	0.000156200	46.90.22.2	46.90.20.3	IPv4	514	Fragmented IP protocol (proto=ICMP 1, off=480, ID=0315) [Reas…
8	0.000157329	46.90.22.2	46.90.20.3	ICMP	82	Echo (ping) reply id=0x002b, seq=1/256, ttl=62 (request in…
9	5.159030462	00:00:00_aa:00:16	00:00:00_aa:00:15	ARP	42	Who has 46.90.19.149? Tell 46.90.19.150
10	5.159204336	00:00:00_aa:00:15	00:00:00_aa:00:16	ARP	42	46.90.19.149 is at 00:00:00:aa:00:15
11	208.420605889	fe80::1cea:2dff:fe0…	ff02::2	ICMPv6	70	Router Solicitation from 76:a0:68:60:9f:21
12	249.103110202	fe80::9c4d:b9ff:fea…	ff02::fb	MDNS	203	Standard query 0x0000 PTR _nfs._tcp.local, "QM" question PTR …
13	249.103199444	fe80::1cea:2dff:fe0…	ff02::fb	MDNS	203	Standard query 0x0000 PTR _nfs._tcp.local, "QM" question PTR …
14	306.724235664	fe80::9c4d:b9ff:fea…	ff02::2	ICMPv6	70	Router Solicitation from 9e:4d:b9:a8:ac:9e

Figura 5. Captura de paquetes fragmentados

Verificación de Conectividad y Enrutamiento IPv6

Para que la conectividad y enrutamiento IPv6 funcione debemos activar los siguientes comandos desde una terminal de linux fuera de la herramienta CORE:

- Habilitar reenvío de paquetes IPv6: **sudo ip6tables -P FORWARD ACCEPT**
- Verificar si la política cambió a "Chain FORWARD (policy ACCEPT)":
  sudo ip6tables -L FORWARD

La prueba del funcionamiento de la topología puede realizarse a partir del archivo **Ejercicio12-Practica3.imn.**

Para el análisis de ICMPv6 se utilizaron las siguientes capturas: **Practica3-ejercicio12-incisoH-redA.pcapng**; **Practica3-ejercicio12-incisoH-redC.pcapng.**

Ejercicio 10 - Práctica 3 complementaria

Cálculo y Asignación de Direcciones

Para asignar el bloque IPv4 a la nueva red de 40 hosts para la red n16 y n15, se analizó el espacio de direcciones disponible restante del Ejercicio 12 de la práctica 3.
El bloque libre comenzaba a partir de la dirección 46.90.16.0 hasta los límites inferiores de las redes ya asignadas.

- Requerimiento: 40 hosts → se requieren 6 bits de host ($2^6 - 2 \geq 62$ hosts útiles).
- Máscara resultante: /26.

Al intentar asignar el bloque inmediato superior disponible **46.90.19.128/26**, se detectó un conflicto con los bloques ya asignados a los enlaces punto a punto (que ocupan el rango alto de la subred .19). Por tal motivo, se seleccionó el siguiente bloque libre contiguo en el rango inferior para garantizar la disponibilidad: **46.90.19.64/26**

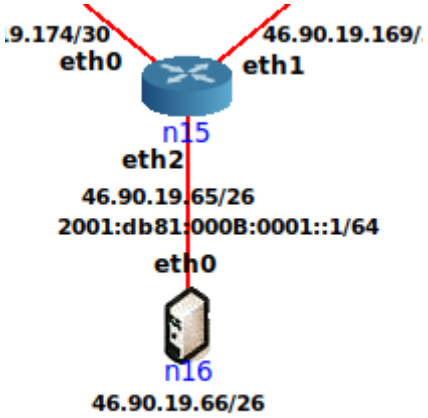


Figura 6. Nueva red de 40 hosts

Implementación de SLAAC y RDNSS

Se realiza la siguiente configuración dentro del archivo **/etc/radvd.conf** del router n15 con el editor de texto vim, quedando el archivo de la siguiente manera:

```
interface eth2
{
    AdvSendAdvert on;
    MinRtrAdvInterval 3;
    MaxRtrAdvInterval 10;
    prefix 2001:db81:000B:0001::/64
    {
        AdvOnLink on;
        AdvAutonomous on;
        AdvRouterAddr on;
    };
    RDNSS 2001:4860:4860::8888 2001:4860:4860::8844
    {
        AdvRDNSSLifetime 30;
    };
};
```

En una terminal de linux fuera de la herramienta CORE instalamos: **sudo apt install radvd.** Luego dentro del servicio **Static Route** del router n15 colocamos lo siguiente: **radvd -n -d 2 -C /etc/radvd.conf.**

Para el análisis se utilizó la captura **Practica3-complemento-RDNSS.pcapng** con el fin de visualizar el paso a paso desde wireshark.

Puede probarse el funcionamiento de la topología a partir del archivo **Ejercicio10-Practica3-complemento.imn.**

Ejercicio 22 - Práctica 4

Servicio UDP

En este caso, se levantó un servicio de UDP utilizando netcat estableciendo una conexión entre los hosts n13 y n9, se configuró el host n9 como servidor UDP en el puerto 8000 permitiendo al host que espere mensajes por ese puerto. Desde n13 se envía un mensaje al puerto n9 que está escuchando. El mensaje es recibido por n9 y se procede a analizar el estado de los sockets involucrados.

```
root@n13:/tmp/pycore.44883/n13.conf# netstat -an | grep udp
udp        0      0 46.90.20.3:36368        46.90.19.194:8000       ESTABLECIDO
root@n13:/tmp/pycore.44883/n13.conf#
```

Figura 7. Servicio UDP

El estado ESTABLISHED es un estado lógico local del sistema operativo y no del protocolo de red. Significa que el socket ha sido configurado para enviar y recibir datos únicamente desde una dirección remota específica, aunque no exista una sesión real ni intercambio de paquetes de negociación (handshake) en la red.

Prueba con cliente programado

Para esta prueba, primero se verifica la correcta configuración del archivo /etc/inetd.conf.

- *echo dgram udp wait root internal* → Se comprueba la existencia de esta linea de codigo y se inicia el servicio:

```
root@n9:/tmp/pycore.45411/n9.conf# /etc/init.d/openbsd-inetd restart
 * Restarting internet superserver inetd
   ...done.
root@n9:/tmp/pycore.45411/n9.conf# netstat -anu | grep :7
udp        0      0 0.0.0.0:7               0.0.0.0:*
root@n9:/tmp/pycore.45411/n9.conf#
```

Figura 8. Conexiones en el nodo

Luego, se ejecuta el script en python que envía mensajes a n9 desde n13. En este caso, se verifica el funcionamiento a continuación.

```
<nando/Redes-de-Datos-II/Recursos-TPI/cliente_udp.py
Enviando datos a 46.90.19.194:7...
¡Éxito! Recibí eco desde ('46.90.19.194', 7): Hola, soy el nodo n11
Cerrando socket.
```

Figura 9. Cliente y Servidor UDP

Envío sin un proceso esperando

Se envía un mensaje desde n13 a n9 donde no existía un proceso escuchando, analizando el Wireshark se puede ver que el stack TCP/IP genera una respuesta de tipo ICMP 'Destination Unreacheable: Port Unreachable', significa que el paquete IP llegó al host de destino, pero no hay ninguna aplicación o proceso escuchando en el puerto TCP/UDP especificado, es decir, el puerto no está disponible.

No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000000	00:00:00_aa:00:00	Broadcast	ARP	42	Who has 46.90.20.1? Tell 46.90.20.3
2	0.000050999	00:00:00_aa:00:02	00:00:00_aa:00:00	ARP	42	46.90.20.1 is at 00:00:00:aa:00:02
3	0.000053111	46.90.20.3	46.90.19.194	UDP	48	58821 → 9999 Len=6
4	0.000286814	46.90.19.194	46.90.20.3	ICMP	76	Destination unreachable (Port unreachable)
5	5.169730383	00:00:00_aa:00:02	00:00:00_aa:00:00	ARP	42	Who has 46.90.20.3? Tell 46.90.20.1
6	5.169754275	00:00:00_aa:00:00	00:00:00_aa:00:02	ARP	42	46.90.20.3 is at 00:00:00:aa:00:00
7	17.188060355	fe80::28be:afff:fe4…	ff02::fb	MDNS	180	Standard query 0x0000 PTR _ftp._tcp.local, "QM" question PTR _nfs._tcp.local, "QM" question PTR _afpover…
8	17.188079582	fe80::144b:61ff:fe8…	ff02::fb	MDNS	180	Standard query 0x0000 PTR _ftp._tcp.local, "QM" question PTR _nfs._tcp.local, "QM" question PTR _afpover…

No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000000	46.90.20.3	46.90.19.194	UDP	48	56363 → 9999 Len=6
2	0.000012178	46.90.19.194	46.90.20.3	ICMP	76	Destination unreachable (Port unreachable)
3	5.136706233	00:00:00_aa:00:1c	00:00:00_aa:00:1b	ARP	42	Who has 46.90.19.193? Tell 46.90.19.194
4	5.136763482	00:00:00_aa:00:1b	00:00:00_aa:00:1c	ARP	42	Who has 46.90.19.194? Tell 46.90.19.193
5	5.136795409	00:00:00_aa:00:1b	00:00:00_aa:00:1c	ARP	42	46.90.19.193 is at 00:00:00:aa:00:1b
6	5.136785674	00:00:00_aa:00:1c	00:00:00_aa:00:1b	ARP	42	46.90.19.194 is at 00:00:00:aa:00:1c

Figura 10. Mensajes ICMP Port Unrecheable

Luego se analiza el traceroute la herramienta que UDP utiliza para encontrar la ruta que siguen los paquetes hacia el destino. Se inicia enviando un paquete UDP a un puerto aleatorio en el host destino, si no hay proceso escuchando, el destino responde con un ICMP. Si durante el proceso debe realizar un salto en la ruta, envía un mensaje ICMP Time Exceeded para indicar que el TLL se hizo 0, y traceroute aumenta progresivamente ese valor.

Tráfico y datagramas

A continuación observamos un paquete enviado desde n13 hacia n9 que corresponde a un paquete UDP encapsulado en uno IPv4.

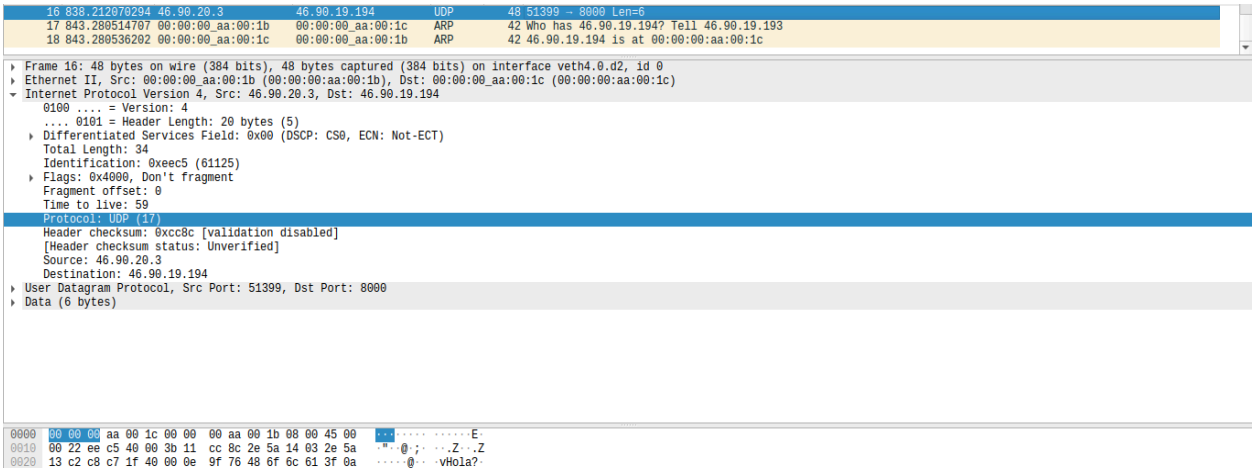


Figura 11. Captura de paquetes fragmentados

Ejercicio 23 - Práctica 4

Servidor TCP con API socket

Se realiza el script de python **servidor_tcp_discard.py** y ejecutamos lo siguiente en cada host:

- Host n9
  - **python3 <ubicación del script>/servidor_tcp_discard.py**
- Host n13
  - **nc <IP_de_n9> 9000**

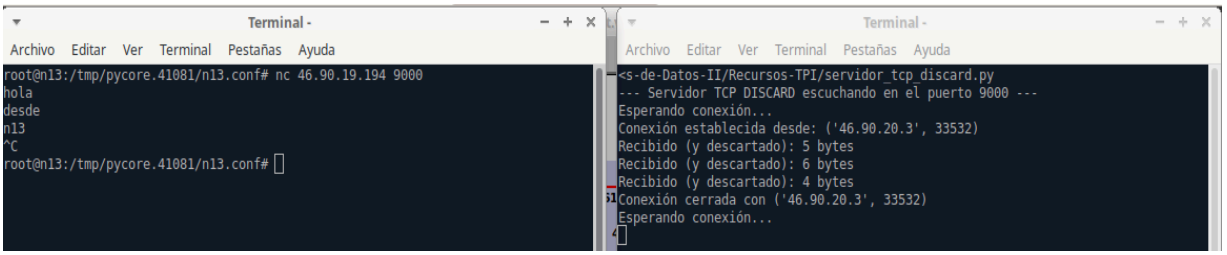


Figura 12. Prueba de conectividad

Servicios extra

Se modificó el archivo **/etc/inetd.conf** para agregar los servicios extra, asegurando que las siguientes líneas se encuentran descomentadas:

- echo dgram udp wait root internal
- discard stream tcp nowait root internal
- daytime stream tcp nowait root internal

Luego se inicia el servicio con el comando **/etc/init.d/openbsd-inetd restart** y se verifica que se está escuchando en los puertos 7, 9 y 13.

```
root@n9:/tmp/pycore.41081/n9.conf# /etc/init.d/openbsd-inetd restart
 * Restarting internet superserver inetd
   ...done.
root@n9:/tmp/pycore.41081/n9.conf# netstat -ant
Conexiones activas de Internet (servidores y establecidos)
Proto  Recib Enviad Dirección local         Dirección remota       Estado
tcp        0      0 0.0.0.0:13              0.0.0.0:*              ESCUCHAR
tcp        0      0 0.0.0.0:9               0.0.0.0:*              ESCUCHAR
tcp        0      0 0.0.0.0:7               0.0.0.0:*              ESCUCHAR
root@n9:/tmp/pycore.41081/n9.conf#
```

Figura 13. Servicios escuchando en sus puertos

Conexión TCP al servicio Discard y Echo

Se utiliza el script de python **cliente_tcp.py** para enviar datos a los servicios de Discard y Echo, de la siguiente manera:

- **python3 <ubicación del script>/cliente_tcp.py 46.90.19.194 7 "Prueba Echo"**
- **python3 <ubicación del script>/cliente_tcp.py 46.90.19.194 9 "Prueba Discard"**

```
do/Redes-de-Datos-II/Recursos-TPI'
<-de-Datos-II/Recursos-TPI/cliente_tcp.py 46.90.19.194 7 "Prueba Echo"
Intentando conectar a 46.90.19.194:7...
Estado: ESTABLISHED (Conexión establecida)
Enviando: Prueba Echo
Recibido: Prueba Echo
Cerrando conexión...
<sos-TPI/cliente_tcp.py 46.90.19.194 9 "Prueba Discard"
Intentando conectar a 46.90.19.194:9...
Estado: ESTABLISHED (Conexión establecida)
Enviando: Prueba Discard
Timeout: El servidor no respondió (Comportamiento normal de Discard).
Cerrando conexión...
root@n13:/tmp/pycore.44707/n13.conf#
```

Figura 14. Prueba Echo y Discard con TCP

Apply a display filter ... <Ctrl-/>

No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000000	46.90.20.3	46.90.19.194	TCP	74	40076 → 7 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 SACK_PERM=1 TSval=2248387520 TSecr=0 WS=128
2	0.000013098	46.90.19.194	46.90.20.3	TCP	74	7 → 40076 [SYN, ACK] Seq=0 Ack=1 Win=65160 Len=0 MSS=1460 SACK_PERM=1 TSval=786666725 TSecr=2248387520 W…
3	0.000050884	46.90.20.3	46.90.19.194	TCP	66	40076 → 7 [ACK] Seq=1 Ack=1 Win=64256 Len=0 TSval=2248387520 TSecr=786666725
4	0.000366666	46.90.20.3	46.90.19.194	ECHO	77	Request
5	0.000370177	46.90.19.194	46.90.20.3	TCP	66	7 → 40076 [ACK] Seq=1 Ack=12 Win=65152 Len=0 TSval=786666725 TSecr=2248387520
6	0.000464078	46.90.19.194	46.90.20.3	ECHO	77	Response
7	0.000497582	46.90.20.3	46.90.19.194	TCP	66	40076 → 7 [ACK] Seq=12 Ack=12 Win=64256 Len=0 TSval=2248387521 TSecr=786666726
8	0.000607774	46.90.20.3	46.90.19.194	TCP	66	40076 → 7 [FIN, ACK] Seq=12 Ack=12 Win=64256 Len=0 TSval=2248387521 TSecr=786666726
9	0.003058355	46.90.19.194	46.90.20.3	TCP	66	7 → 40076 [FIN, ACK] Seq=12 Ack=13 Win=65152 Len=0 TSval=786666728 TSecr=2248387521
10	0.003144579	46.90.20.3	46.90.19.194	TCP	66	40076 → 7 [ACK] Seq=13 Ack=13 Win=64256 Len=0 TSval=2248387523 TSecr=786666728
11	4.633618436	46.90.20.3	46.90.19.194	TCP	74	60470 → 9 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 SACK_PERM=1 TSval=2248392154 TSecr=0 WS=128
12	4.633629187	46.90.19.194	46.90.20.3	TCP	74	9 → 60470 [SYN, ACK] Seq=0 Ack=1 Win=65160 Len=0 MSS=1460 SACK_PERM=1 TSval=786671359 TSecr=2248392154 W…
13	4.633664758	46.90.20.3	46.90.19.194	TCP	66	60470 → 9 [ACK] Seq=1 Ack=1 Win=64256 Len=0 TSval=2248392154 TSecr=786671359
14	4.633995426	46.90.20.3	46.90.19.194	TCP	80	60470 → 9 [PSH, ACK] Seq=1 Ack=1 Win=64256 Len=14 TSval=2248392154 TSecr=786671359
15	4.634000924	46.90.19.194	46.90.20.3	TCP	66	9 → 60470 [ACK] Seq=1 Ack=15 Win=65152 Len=0 TSval=786671359 TSecr=2248392154
16	6.635715381	46.90.20.3	46.90.19.194	TCP	66	60470 → 9 [FIN, ACK] Seq=15 Ack=1 Win=64256 Len=0 TSval=2248394156 TSecr=786671359
17	6.638737512	46.90.19.194	46.90.20.3	TCP	66	9 → 60470 [FIN, ACK] Seq=1 Ack=16 Win=65152 Len=0 TSval=786673364 TSecr=2248394156
18	6.638870106	46.90.20.3	46.90.19.194	TCP	66	60470 → 9 [ACK] Seq=16 Ack=2 Win=64256 Len=0 TSval=2248394159 TSecr=786673364

Figura 15. Paquetes Echo y Discard

Estado de servicios

En n9 es posible visualizar los estados de los sockets utilizando el comando netstat.

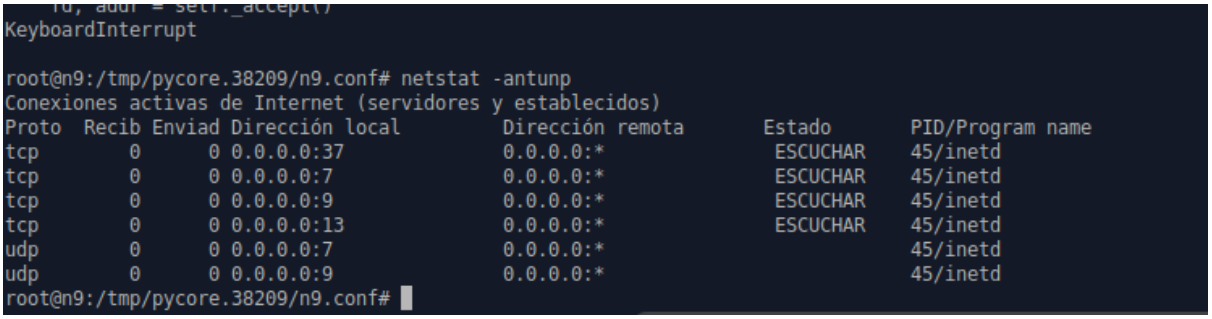


Figura 16. Conexión del servicio

Nuevas conexiones

Se desarrolló un script de python **multi_cliente.py** que utiliza hilos para establecer 4 conexiones TCP simultáneas hacia el puerto 7 (Echo) del nodo n9 y mantenerlas activas durante 15 segundos.

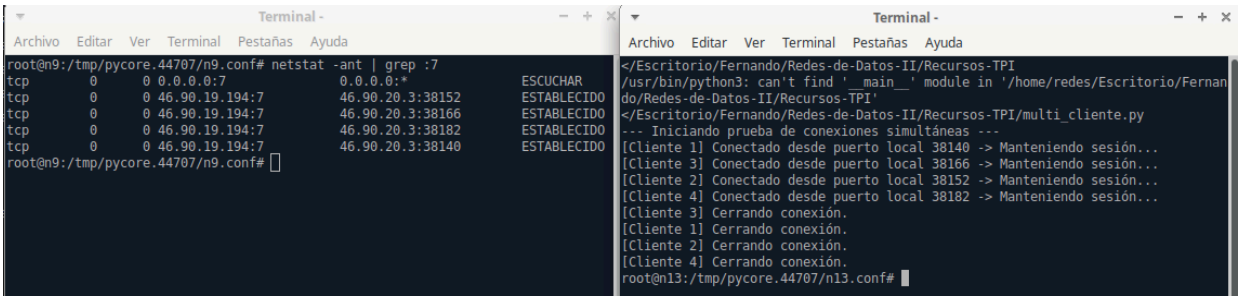


Figura 17. Nuevas conexiones hacia n9

Conexión a un puerto donde no existe un proceso esperando

Desde el nodo n13, se intentó establecer una conexión TCP hacia el puerto 9999 del servidor n9. Dado que en dicho puerto no se encuentra ningún servicio escuchando (estado CLOSED), el sistema operativo del servidor debe rechazar la solicitud. Comando ejecutado: **nc -v 46.90.19.194 9999**

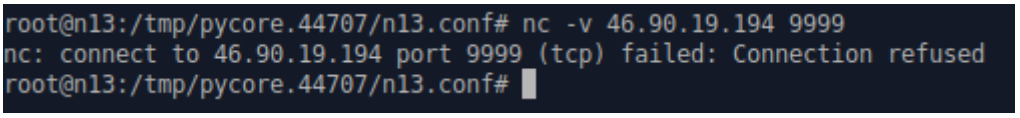


Figura 18. Conexión rechazada

Para un mejor análisis de los flags activados se utiliza la captura de wireshark del archivo **Practica4-ejercicio23-incisoF.pcapng.** Allí observamos lo siguiente:

- **Cliente (n13):** Envía un segmento SYN dirigido al puerto 9999, intentando iniciar el handshake.
- **Servidor (n9):** El stack TCP recibe el segmento y verifica si hay algún socket en estado LISTEN asociado a ese puerto. Al no encontrar ninguno, responde inmediatamente con un segmento donde se activan los flags RST, ACK.
- **Cierre:** Al recibir el flag RST, el cliente aborta inmediatamente el intento de conexión y devuelve el error "Connection refused" al usuario, sin intentar retransmisiones adicionales típicas de un timeout.

Conexiones cerradas

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

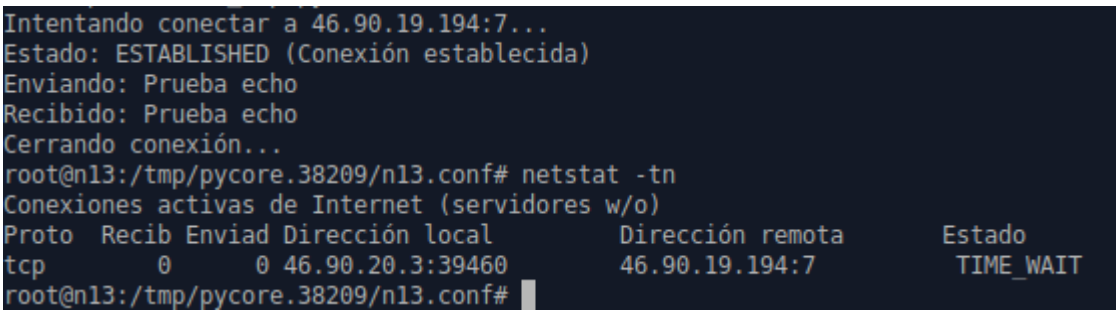


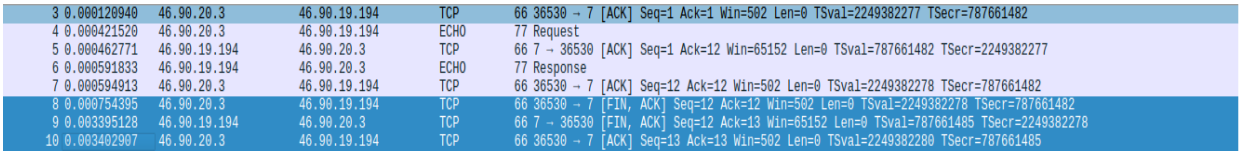
Figura 19. Conexión TIME_WAIT

Segmentos y Flags

En la figura se observan únicamente dos flags TCP: ACK y FIN, ACK (combinados)

En los paquetes con el flag ACK se está realizando el reconocimiento de segmentos previamente recibidos.

El paquete con los flags FIN y ACK indica el inicio del cierre ordenado de la conexión por parte de uno de los extremos, confirmando al mismo tiempo los datos anteriores. No se observan flags SYN, RST, PSH ni URG. Esto implica que la conexión ya se encontraba establecida y que los paquetes mostrados corresponden a la fase final y al cierre de la sesión TCP.



3	0.000120940	46.90.20.3	46.90.19.194	TCP	66 36530 → 7 [ACK] Seq=1 Ack=1 Win=502 Len=0 TSval=2249382277 TSecr=787661482
4	0.000421520	46.90.20.3	46.90.19.194	ECHO	77 Request
5	0.000462771	46.90.19.194	46.90.20.3	TCP	66 7 → 36530 [ACK] Seq=1 Ack=12 Win=65152 Len=0 TSval=787661482 TSecr=2249382277
6	0.000591833	46.90.19.194	46.90.20.3	ECHO	77 Response
7	0.000594913	46.90.20.3	46.90.19.194	TCP	66 36530 → 7 [ACK] Seq=12 Ack=12 Win=502 Len=0 TSval=2249382278 TSecr=787661482
8	0.000754395	46.90.20.3	46.90.19.194	TCP	66 36530 → 7 [FIN, ACK] Seq=12 Ack=12 Win=502 Len=0 TSval=2249382278 TSecr=787661482
9	0.003395128	46.90.19.194	46.90.20.3	TCP	66 7 → 36530 [FIN, ACK] Seq=12 Ack=13 Win=65152 Len=0 TSval=787661485 TSecr=2249382278
10	0.003402907	46.90.20.3	46.90.19.194	TCP	66 36530 → 7 [ACK] Seq=13 Ack=13 Win=502 Len=0 TSval=2249382280 TSecr=787661485

Figura 20. Flags de cierre

Diagrama de los segmentos

El siguiente diagrama fue realizado a partir de la captura de wireshark del archivo **Practica4-ejercicio23-incisoI.pcapng.**

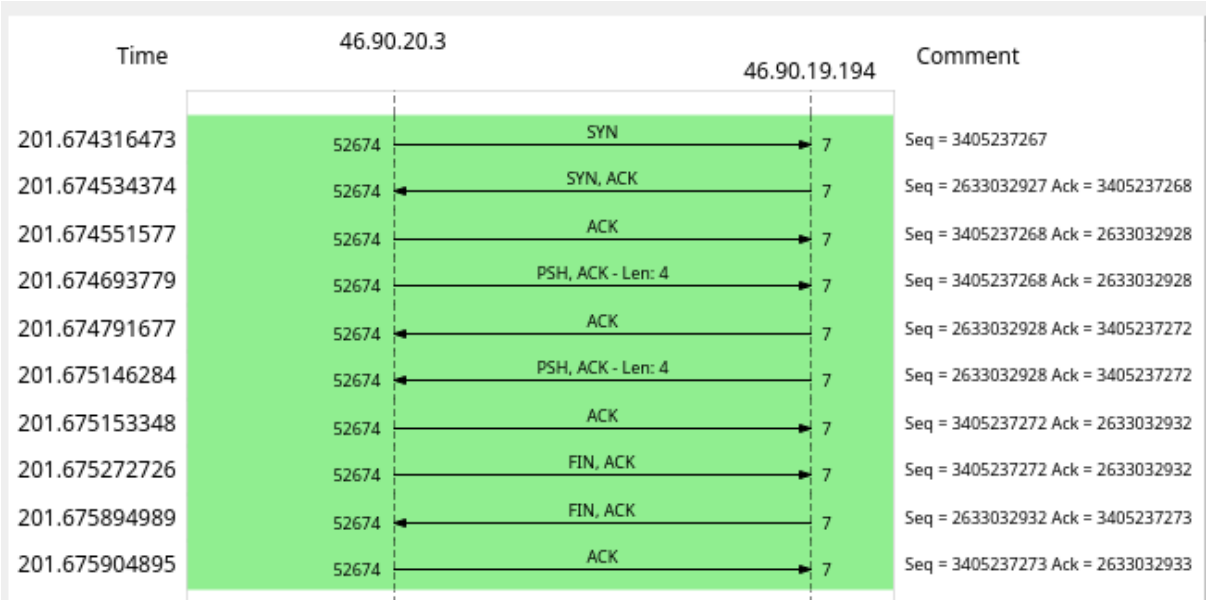


Figura 21. Diagrama de segmentos