

Práctica 1.2. Conceptos Avanzados de TCP

Objetivos

En esta práctica estudiaremos el funcionamiento del protocolo TCP. Además veremos algunos parámetros que permiten ajustar el comportamiento de las aplicaciones TCP. Finalmente se consideran algunas aplicaciones del filtrado de paquetes mediante iptables.



Para cada ejercicio, se tienen que proporcionar los **comandos utilizados con sus correspondientes salidas**, las **capturas de pantalla de Wireshark realizadas**, y la **información requerida de manera específica**.

Activar el portapapeles bidireccional en las máquinas (menú Dispositivos) para copiar la salida de los comandos. Las capturas de pantalla se realizarán usando también Virtualbox (menú Ver).

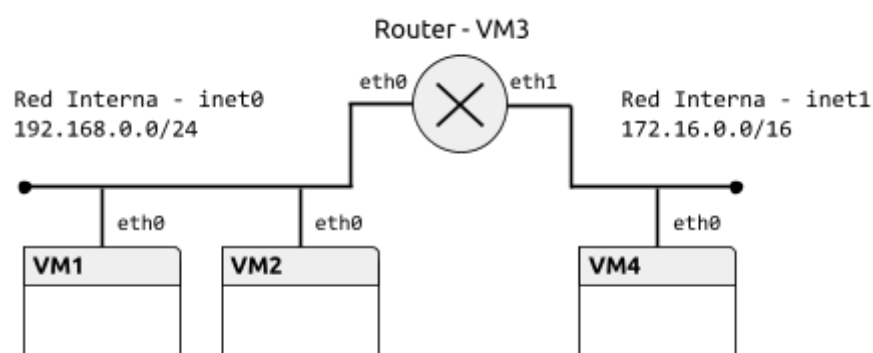
Las **credenciales de la máquina virtual** son: usuario cursoredes, con contraseña cursoredes.

Contenidos

- Preparación del entorno para la práctica
- Estados de una conexión TCP
- Introducción a la seguridad en el protocolo TCP
- Opciones y parámetros TCP
- Traducción de direcciones (NAT) y reenvío de puertos (*port forwarding*)

Preparación del entorno para la práctica

Configuraremos la topología de red que se muestra en la siguiente figura, igual a la empleada en la práctica anterior.



El contenido del fichero de configuración de la topología debe ser el siguiente:

```
netprefix inet
machine 1 0 0
machine 2 0 0
machine 3 0 0 1 1
machine 4 0 1
```

Finalmente, configurar la red de todas las máquinas de la red según la siguiente tabla. Después de configurar todas las máquinas, comprobar la conectividad con la orden ping.

Máquina	Dirección IPv4	Comentarios
VM1	192.168.0.1/24	Añadir Router como encaminador por defecto
VM2	192.168.0.2/24	Añadir Router como encaminador por defecto
Router - VM3	192.168.0.3/24 (eth0) 172.16.0.3/16 (eth1)	Activar el <i>forwarding</i> de paquetes
VM4	172.16.0.4/16	Añadir Router como encaminador por defecto

Estados de una conexión TCP

En esta parte usaremos la herramienta Netcat, que permite leer y escribir en conexiones de red. Netcat es muy útil para investigar y depurar el comportamiento de la red en la capa de transporte, ya que permite especificar un gran número de los parámetros de la conexión. Además para ver el estado de las conexiones de red usaremos el comando ss (similar a netstat, pero más moderno y completo).

Ejercicio 1. Consultar las páginas de manual de nc y ss. En particular, consultar las siguientes opciones de ss: -a, -l, -n, -t y -o. Probar algunas de las opciones para ambos programas para familiarizarse con su comportamiento.

ss -a, --all -> Muestra tanto los sockets donde se está escuchando (servidor esperando conexión) como en los que no.

ss -l, --listening -> Muestra solo los sockets que están escuchando (esperando un cliente que se conecte)

ss -n, --numeric -> Muestra el puerto del servidor en lugar de tratar de poner su nombre.

ss -t, --tcp -> Muestra los sockets TCP

ss -o, --options -> Muestra información sobre temporizadores

Ejercicio 2. (LISTEN) Abrir un servidor TCP en el puerto 7777 en VM1 usando el comando nc -l 7777. Comprobar el estado de la conexión en el servidor con el comando ss -tln. Abrir otro servidor en el puerto 7776 en VM1 usando el comando nc -l 192.168.0.1 7776. Observar la diferencia entre ambos servidores usando ss. Comprobar que no es posible la conexión desde VM1 con localhost como dirección destino usando el comando nc localhost 7776.

Adjuntar la salida del comando ss correspondiente a los servidores.

Estado de la conexión tras abrir el servidor en 7777:

State	Recv-Q	Send-Q	Local Address:Port	Peer Address:Port
LISTEN	0	100	127.0.0.1:25	*.*
LISTEN	0	10	*:7777	*.*
LISTEN	0	128	*:111	*.*
LISTEN	0	128	*:22	*.*
LISTEN	0	128	127.0.0.1:631	*.*

```
LISTEN 0 100      ::1:25          :::*
LISTEN 0 10      :::7777         :::*
LISTEN 0 128     :::111          :::*
LISTEN 0 128     :::22           :::*
LISTEN 0 128     ::1:631         :::*
```

Estado de la conexión tras abrir también el servidor en 7776:

State	Recv-Q	Send-Q	Local Address:Port	Peer Address:Port
LISTEN	0	100	127.0.0.1:25	::*
LISTEN	0	10	192.168.0.1:7776	::*
LISTEN	0	10	:::7777	::*
LISTEN	0	128	:::111	::*
LISTEN	0	128	:::22	::*
LISTEN	0	128	127.0.0.1:631	::*
LISTEN	0	100	::1:25	::*
LISTEN	0	10	:::7777	::*
LISTEN	0	128	:::111	::*
LISTEN	0	128	:::22	::*
LISTEN	0	128	::1:631	::*

La diferencia es que para el servidor en 7776 pone la dirección ip que le hemos dado como Local Address, mientras que en el 7777 un asterisco (con este sí nos pondremos conectar con la dirección ip localaddress).

Al intentar conectar un cliente al servidor de 7776 poniendo dirección ip localhost:

Ncat: Connection refused.

Ejercicio 3. (ESTABLISHED) En VM2, iniciar una conexión cliente al primer servidor arrancado en el ejercicio anterior usando el comando `nc 192.168.0.1 7777`.

- Comprobar el estado de la conexión e identificar los parámetros (dirección IP y puerto) con el comando `ss -tn`.
- Iniciar una captura con Wireshark. Intercambiar un único carácter con el cliente y observar los mensajes intercambiados (especialmente los números de secuencia, confirmación y flags TCP) y determinar cuántos bytes (y número de mensajes) han sido necesarios.

Adjuntar la salida del comando ss correspondiente a la conexión y una captura de pantalla de Wireshark.

[VM1]

State	Recv-Q	Send-Q	Local Address:Port	Peer Address:Port
ESTAB	0	0	192.168.0.1:7777	192.168.0.2:48826

El servidor está en la dirección ip 192.168.0.1 en el puerto 7777

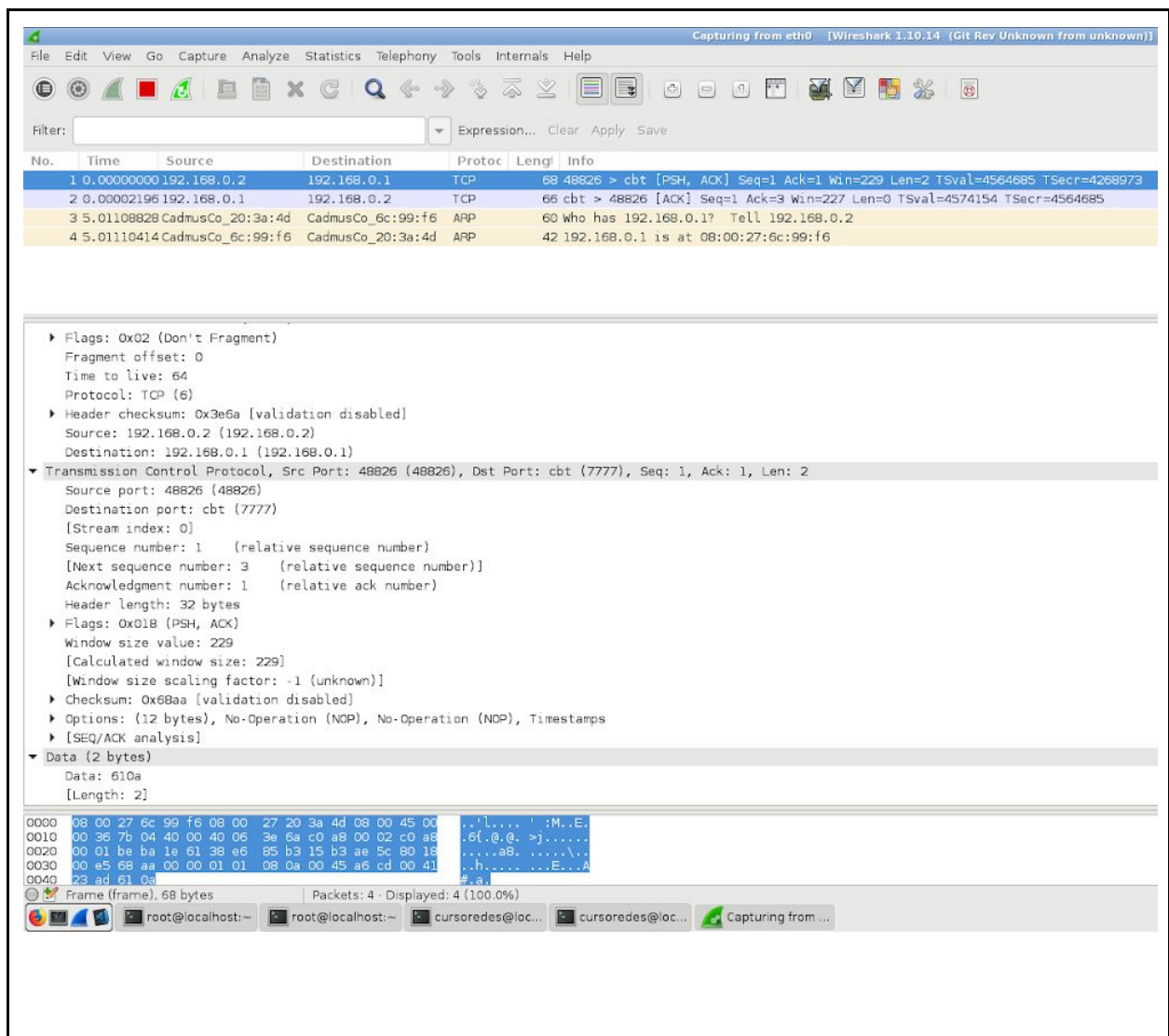
El cliente está en la dirección ip 192.168.0.2 en el puerto 48826

Mandamos el carácter a del cliente al servidor:

Intercambian dos mensajes.

El cliente manda uno con el carácter, con los flags de PSH (pasar inmediatamente a la capa de aplicación) y ACK que tiene un tamaño de 68 bytes (SEQ = 1, ACK = 1).

El servidor le contesta con un mensaje de ACK que ocupa 66 bytes con (SEQ = 1, ACK =3) (debido a que ha recibido 2 bytes de datos y por tanto el siguiente que espera es el tercero).



Ejercicio 4. (TIME-WAIT) Cerrar la conexión en el cliente (con Ctr+C) y comprobar el estado de la conexión usando `ss -tan`. Usar la opción `-o` de `ss` para observar el valor del temporizador TIME-WAIT.

Adjuntar la salida del comando `ss` correspondiente a la conexión.

```
State Recv-Q Send-Q Local Address:Port Peer Address:Port
LISTEN 0 100 127.0.0.1:25 *:*
LISTEN 0 128 *:111 *:*
LISTEN 0 128 *:22 *:*
LISTEN 0 128 127.0.0.1:631 *:*
TIME-WAIT 0 0 192.168.0.2:48832 192.168.0.1:7777
timer:(timewait,45sec,0)
LISTEN 0 100 :::1:25 :::*
LISTEN 0 128 :::111 :::*
LISTEN 0 128 :::22 :::*
LISTEN 0 128 :::1:631 :::*
```

El cliente ha iniciado el temporizador TIME-WAIT al que le quedan 45 segundos por si fuese necesario reenviar el ACK final.

Ejercicio 5. (SYN-SENT y SYN-RCVD) El comando `iptables` permite filtrar paquetes según los flags TCP del segmento con la opción `--tcp-flags` (consultar la página de manual `iptables-extensions`). Usando esta opción:

- Fijar una regla en el servidor (VM1) que bloquee un mensaje del acuerdo TCP de forma que el cliente (VM2) se quede en el estado SYN-SENT. Comprobar el resultado con `ss -tan` en el cliente.
- Borrar la regla anterior y fijar otra en el cliente que bloquee un mensaje del acuerdo TCP de forma que el servidor se quede en el estado SYN-RCVD. Comprobar el resultado con `ss -tan` en el servidor. Además, esta regla debe dejar al servidor también en el estado LAST-ACK después de cerrar la conexión (con `Ctrl+C`) en el cliente. Usar la opción `-o` de `ss` para determinar cuántas retransmisiones se realizan y con qué frecuencia.

Adjuntar los comandos iptables utilizados y la salida del comando ss correspondiente a las conexiones.

Se ha utilizado el comando iptables:

`iptables -A INPUT -p tcp --tcp-flags ALL SYN -j DROP`

Y se ha obtenido salida para ss -tan en [VM2]:

```
LISTEN 0 100 127.0.0.1:25 *:*
LISTEN 0 128 *:111 *:*
LISTEN 0 128 *:22 *:*
LISTEN 0 128 127.0.0.1:631 *:*
SYN-SENT 0 1 192.168.0.2:48836 192.168.0.1:7777
LISTEN 0 100 ::1:25 :::*
LISTEN 0 128 :::111 :::*
LISTEN 0 128 :::22 :::*
LISTEN 0 128 ::1:631 :::*
```

Borramos con iptables -D INPUT 1

Ahora bloqueamos los mensajes del cliente que contenga un ACK mediante:

`iptables -A OUTPUT -p tcp --tcp-flags ALL ACK -j DROP`

Si hacemos ss -tano en el servidor (VM1):

```
State Recv-Q Send-Q Local Address:Port Peer Address:Port
LISTEN 0 100 127.0.0.1:25 *:*
LISTEN 0 10 *:7777 *:* timer:(keepalive,124ms,0)
SYN-RCV 0 0 192.168.0.1:7777 192.168.0.2:48874 timer:(on,724ms,1)
LISTEN 0 128 *:111 *:*
LISTEN 0 128 *:22 *:*
LISTEN 0 128 127.0.0.1:631 *:*
LISTEN 0 100 ::1:25 :::*
LISTEN 0 10 :::7777 :::*
LISTEN 0 128 :::111 :::*
LISTEN 0 128 :::22 :::*
LISTEN 0 128 ::1:631 :::*
```

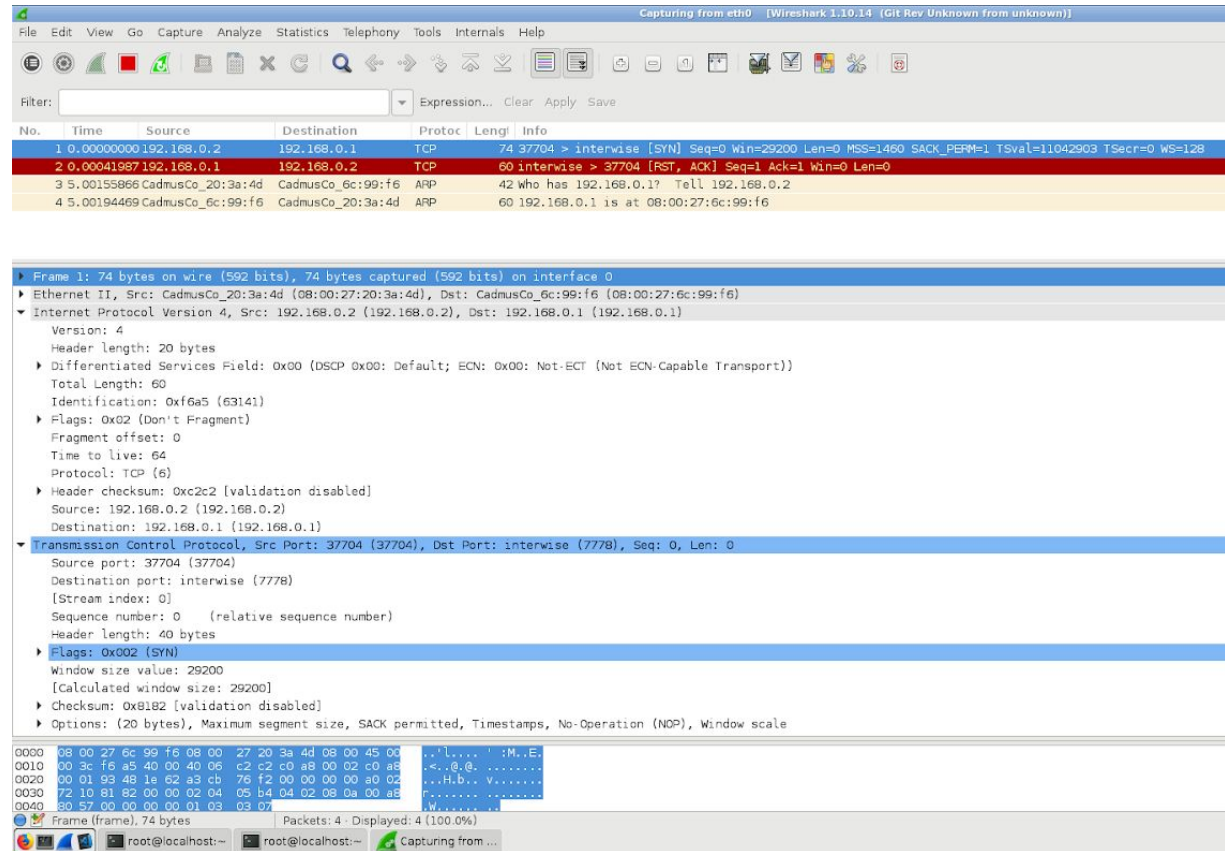
Ahora hacemos Ctrl+C en el cliente (VM2) y de nuevo ss -tano en el servidor (VM1):

```
tate Recv-Q Send-Q Local Address:Port Peer Address:Port
LISTEN 0 100 127.0.0.1:25 *:*
LISTEN 0 128 *:111 *:*
LISTEN 0 128 *:22 *:*
LISTEN 0 128 127.0.0.1:631 *:*
LAST-ACK 0 1 192.168.0.1:7777 192.168.0.2:48874 timer:(on,16sec,0)
LISTEN 0 100 ::1:25 :::*
LISTEN 0 128 :::111 :::*
LISTEN 0 128 :::22 :::*
LISTEN 0 128 ::1:631 :::*
```

Mediante `ss -tano`, haciéndolo repetidamente se ha descubierto que se hacen 6 retransmisiones. Cada una tarda más que la anterior

Ejercicio 6. Iniciar una captura con Wireshark. Intentar una conexión a un puerto cerrado del servidor (ej. 7778) y observar los mensajes TCP intercambiados, especialmente los flags TCP.

Adjuntar una captura de pantalla de Wireshark.



El cliente manda un SYN para establecer la conexión pero como el puerto está cerrado recibe un RST+ACK de respuesta.

Introducción a la seguridad en el protocolo TCP

Diferentes aspectos del protocolo TCP pueden aprovecharse para comprometer la seguridad del sistema. En este apartado vamos a estudiar dos: ataques DoS basados en TCP SYN *flood* y técnicas de exploración de puertos.

Ejercicio 7. El ataque TCP SYN *flood* consiste en saturar un servidor mediante el envío masivo de mensajes SYN.

- (Cliente VM2) Para evitar que el atacante responda al mensaje SYN+ACK del servidor con un mensaje RST que liberaría los recursos, bloquear los mensajes SYN+ACK en el atacante con iptables.
- (Cliente VM2) Para enviar paquetes TCP con los datos de interés usaremos el comando `hping3` (estudiar la página de manual). En este caso, enviar mensajes SYN al puerto 22 del servidor (ssh) lo más rápido posible (*flood*).
- (Servidor VM1) Estudiar el comportamiento de la máquina, en términos del número de paquetes recibidos. Comprobar si es posible la conexión al servicio ssh.

Repetir el ejercicio desactivando el mecanismo SYN cookies en el servidor con el comando `sysctl` (parámetro `net.ipv4.tcp_syncookies`).

Adjuntar los comandos iptables y hping3 utilizados. Describir el comportamiento de la máquina con y sin el mecanismo SYN cookies.

Introducimos una regla en la tabla INPUT de VM2 para bloquear el SYN+ACK que venga del puerto 22:

`iptables -A INPUT -p tcp --tcp-flags ALL SYN,ACK -j DROP`

Si hacemos `sysctl net.ipv4.tcp_syncookies` (en VM1) comprobamos que está a 1 (activado).

Mandamos paquetes al puerto 22 de la dirección 192.168.0.1 con la opción flood (lo más rápido posible):

`hping3 --flood -p 22 -S 192.168.0.1`

Desde, por ejemplo VM3, hacemos `nc 192.168.0.1 22` para conectarnos a ssh y obtenemos:

`SSH-2.0-OpenSSH_7.4`

(Nos conseguimos conectar al servicio)

Si lo desactivamos con `sysctl net.ipv4.tcp_syncookies=0` y volvemos a hacer `nc 192.168.0.1 22` en VM3 nos sale:

`Ncat: Connection timed out.`

Ejercicio 8. (Técnica CONNECT) Ncat permite explorar puertos usando la técnica CONNECT que intenta establecer una conexión a un puerto determinado. En función de la respuesta (SYN+ACK o RST), es posible determinar si hay un proceso escuchando.

- (Servidor VM1) Abrir un servidor en el puerto 7777.
- (Cliente VM2) Explorar, de uno en uno, el rango de puertos 7775-7780 usando nc, en este caso usar las opciones de exploración (-z) y de salida detallada (-v). **Nota:** La versión de nc instalada no soporta rangos de puertos.
- Con ayuda de Wireshark, observar los paquetes intercambiados.

Adjuntar los comandos nc utilizados y su salida.

Salida para el comando `nc -z -v 192.168.0.1 7777`:

`Ncat: Version 7.50 (https://nmap.org/ncat)`

`Ncat: Connected to 192.168.0.1:7777.`

`Ncat: 0 bytes sent, 0 bytes received in 0.01 seconds.`

Salida para todos los demás (p.e `nc -z -v 192.168.0.1 7775`):

`Ncat: Version 7.50 (https://nmap.org/ncat)`

`Ncat: Connection refused.`

(No se pide, pero se adjunta una captura de Wireshark)

1	0.00000000	192.168.0.2	192.168.0.1	TCP	74 42700 > 7775 [SYN] Seq=0 Win=29200 Len=0 MSS=1460 SACK_PERM=1 TSval=18789790 TSecr=0
2	0.00002043	192.168.0.1	192.168.0.2	TCP	54 7775 > 42700 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0
3	2.20955411	192.168.0.2	192.168.0.1	TCP	74 55134 > 7776 [SYN] Seq=0 Win=29200 Len=0 MSS=1460 SACK_PERM=1 TSval=18792000 TSecr=0
4	2.20957486	192.168.0.1	192.168.0.2	TCP	54 7776 > 55134 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0
5	4.40941132	192.168.0.2	192.168.0.1	TCP	74 48920 > cbt [SYN] Seq=0 Win=29200 Len=0 MSS=1460 SACK_PERM=1 TSval=18794199 TSecr=0
6	4.40943510	192.168.0.1	192.168.0.2	TCP	74 cbt > 48920 [SYN, ACK] Seq=0 Ack=1 Win=28960 Len=0 MSS=1460 SACK_PERM=1 TSval=1885508 TSecr=0
7	4.40982809	192.168.0.2	192.168.0.1	TCP	66 48920 > cbt [ACK] Seq=1 Ack=1 Win=29312 Len=0 TSval=18794200 TSecr=18855086
8	4.40994773	192.168.0.2	192.168.0.1	TCP	66 48920 > cbt [FIN, ACK] Seq=1 Ack=1 Win=29312 Len=0 TSval=18794200 TSecr=18855086
9	4.41002158	192.168.0.1	192.168.0.2	TCP	66 cbt > 48920 [FIN, ACK] Seq=1 Ack=2 Win=29056 Len=0 TSval=18855087 TSecr=18794200
10	4.41026023	192.168.0.2	192.168.0.1	TCP	66 48920 > cbt [ACK] Seq=2 Ack=2 Win=29312 Len=0 TSval=18794201 TSecr=18855087
11	8.08542122	192.168.0.2	192.168.0.1	TCP	74 37750 > interwise [SYN] Seq=0 Win=29200 Len=0 MSS=1460 SACK_PERM=1 TSval=18797875 TSecr=0
12	8.08545771	192.168.0.1	192.168.0.2	TCP	54 interwise > 37750 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0
13	10.1241920	192.168.0.2	192.168.0.1	TCP	74 38996 > vstat [SYN] Seq=0 Win=29200 Len=0 MSS=1460 SACK_PERM=1 TSval=18799912 TSecr=0
14	10.1242125	192.168.0.1	192.168.0.2	TCP	54 vstat > 38996 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0
15	13.8011601	192.168.0.2	192.168.0.1	TCP	74 36580 > 7780 [SYN] Seq=0 Win=29200 Len=0 MSS=1460 SACK_PERM=1 TSval=18803588 TSecr=0
16	13.8011805	192.168.0.1	192.168.0.2	TCP	54 7780 > 36580 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0

Opcional. La herramienta Nmap permite realizar diferentes tipos de exploración de puertos, que emplean estrategias más eficientes. Estas estrategias (*SYN stealth*, *ACK stealth*, *FIN-ACK stealth*...) se basan en el funcionamiento del protocolo TCP. Estudiar la página de manual de nmap (PORT SCANNING TECHNIQUES) y emplearlas para explorar los puertos del servidor. Comprobar con Wireshark los mensajes intercambiados.

Opciones y parámetros de TCP

El comportamiento de la conexión TCP se puede controlar con varias opciones que se incluyen en la cabecera en los mensajes SYN y que son configurables en el sistema operativo por medio de parámetros del kernel.

Ejercicio 9. Con ayuda del comando `sysctl` y la bibliografía recomendada, completar la siguiente tabla con parámetros que permiten modificar algunas opciones de TCP:

Parámetro del kernel	Propósito	Valor por defecto
<code>net.ipv4.tcp_window_scaling</code>	Para aumentar el tamaño de la ventana de recepción por encima de 64K.	Activado
<code>net.ipv4.tcp_timestamps</code>	Guardar marcas de tiempo de cuando se envió el mensaje	Activadas
<code>net.ipv4.tcp_sack</code>	Permite confirmar un segmento fuera de orden y solo ese. Se implementa como una opción de ACK.	Activado

Ejercicio 10. Iniciar una captura de Wireshark. Abrir el servidor en el puerto 7777 y realizar una conexión desde la VM cliente. Estudiar el valor de las opciones que se intercambian durante la conexión. Variar algunos de los parámetros anteriores (ej. no usar ACKs selectivos) y observar el resultado en una nueva conexión.

Adjuntar una captura de pantalla de Wireshark donde se muestren las opciones TCP.

Establecemos conexión con el servidor desde VM2 sin modificar nada y nos fijamos en las opciones mencionadas:

Aparecen los 3 parámetros entre las opciones, por ejemplo en timestamps podemos ver que el primer mensaje tiene timestamp = 21478882. TSecr denota la última marca de tiempo del mensaje justamente anterior. También podemos ver una escala en la ventana de recepción de $2^7 = 128$

No.	Time	Source	Destination	Protoc	Length	Info
1	0.00000000	192.168.0.2	192.168.0.1	TCP	74	48938 > cbt [SYN] Seq=0 Win=29200 Len=0 MSS=1460 SACK_PERM=1 TSval=21478882 TSecr=0 WS=128
2	0.00009304	192.168.0.1	192.168.0.2	TCP	74	cbt > 48938 [SYN, ACK] Seq=0 Ack=1 Win=28960 Len=0 MSS=1460 SACK_PERM=1 TSval=21539781 TSecr=
3	0.00034426	192.168.0.2	192.168.0.1	TCP	66	48938 > cbt [ACK] Seq=1 Ack=1 Win=29312 Len=0 TSval=21478883 TSecr=21539781

Coloring Rule String: tcp.flags & 0x02 tcp.flags.fin == 1
Ethernet II, Src: CadmusCo_20:3a:4d (08:00:27:20:3a:4d), Dst: CadmusCo_6c:99:f6 (08:00:27:6c:99:f6)
Internet Protocol Version 4, Src: 192.168.0.2 (192.168.0.2), Dst: 192.168.0.1 (192.168.0.1)
Transmission Control Protocol, Src Port: 48938 (48938), Dst Port: cbt (7777), Seq: 0, Len: 0
Source port: 48938 (48938)
Destination port: cbt (7777)
[Stream index: 0]
Sequence number: 0 (relative sequence number)
Header length: 40 bytes
Flags: 0x002 (SYN)
Window size value: 29200
[Calculated window size: 29200]
Checksum: 0x0978 [validation disabled]
Options: (20 bytes), Maximum segment size, SACK permitted, Timestamps, No-Operation (NOP), Window scale
Maximum segment size: 1460 bytes
TCP SACK Permitted Option: True
Kind: SACK Permission (4)
Length: 2
Timestamps: TSval 21478882, TSecr 0
Kind: Timestamp (8)
Length: 10
Timestamp value: 21478882
Timestamp echo reply: 0
No-Operation (NOP)
Window scale: 7 (multiply by 128)
Kind: Window Scale (3)
Length: 3
Shift count: 7
[Multiplier: 128]

Desactivamos los 3 parámetros en VM2 y todos menos windows scaling en VM1, desaparecen todas estas opciones si nos fijamos, por ejemplo, en el primer mensaje:

No.	Time	Source	Destination	Protoc	Length	Info
1	0.00000000	192.168.0.2	192.168.0.1	TCP	60	48946 > cbt [SYN] Seq=0 Win=29200 Len=0 MSS=1460
2	0.00002490	192.168.0.1	192.168.0.2	TCP	58	cbt > 48946 [SYN, ACK] Seq=0 Ack=1 Win=29200 Len=0 MSS=1460
3	0.00025250	192.168.0.2	192.168.0.1	TCP	60	48946 > cbt [ACK] Seq=1 Ack=1 Win=29200 Len=0
4	5.00179704	CadmusCo_20:3a:4d	CadmusCo_6c:99:f6	ARP	42	who has 192.168.0.2? Tell 192.168.0.1
5	5.00218077	CadmusCo_20:3a:4d	CadmusCo_6c:99:f6	ARP	60	192.168.0.2 is at 08:00:27:20:3a:4d

[Time since reference or first frame: 0.00000000 seconds]
Frame Number: 1
Frame Length: 60 bytes (480 bits)
Capture Length: 60 bytes (480 bits)
[Frame is marked: False]
[Frame is ignored: False]
[Protocols in frame: eth:ip:tcp]
[Coloring Rule Name: TCP SYN/FIN]
[Coloring Rule String: tcp.flags & 0x02 tcp.flags.fin == 1]
Ethernet II, Src: CadmusCo_20:3a:4d (08:00:27:20:3a:4d), Dst: CadmusCo_6c:99:f6 (08:00:27:6c:99:f6)
Internet Protocol Version 4, Src: 192.168.0.2 (192.168.0.2), Dst: 192.168.0.1 (192.168.0.1)
Transmission Control Protocol, Src Port: 48946 (48946), Dst Port: cbt (7777), Seq: 0, Len: 0
Source port: 48946 (48946)
Destination port: cbt (7777)
[Stream index: 0]
Sequence number: 0 (relative sequence number)
Header length: 24 bytes
Flags: 0x002 (SYN)
Window size value: 29200
[Calculated window size: 29200]
Checksum: 0x2216 [validation disabled]
Options: (4 bytes), Maximum segment size
Maximum segment size: 1460 bytes

Ejercicio 11. Con ayuda del comando `sysctl` y la bibliografía recomendada, completar la siguiente tabla con parámetros que permiten configurar el temporizador *keepalive*:

Parámetro del kernel	Propósito	Valor por defecto
<code>net.ipv4.tcp_keepalive_time</code>	Tiempo a partir del cual se empiezan a enviar señales si no hay tráfico de datos en la conexión para cerrarla y que no se mantenga indefinidamente abierta.	7200 segundos
<code>net.ipv4.tcp_keepalive_probes</code>	Número de sondas que se envían antes de cerrar la conexión esperando un ACK para no hacerlo como respuesta.	9 sondas
<code>net.ipv4.tcp_keepalive_intvl</code>	Tiempo entre sonda y sonda enviada antes de cerrar la conexión.	75 segundos

Traducción de direcciones (NAT) y reenvío de puertos (*port forwarding*)

En esta sección supondremos que la red que conecta Router con VM4 es pública y que no puede encaminar el tráfico `192.168.0.0/24`. Además, asumiremos que la dirección IP de Router es dinámica.

Ejercicio 12. Configurar la traducción de direcciones dinámica en Router:

- (Router) Usando `iptables`, configurar Router para que haga SNAT (*masquerade*) sobre la interfaz `eth1`. Iniciar una captura de Wireshark en los dos interfaces.
- (VM1) Comprobar la conexión entre VM1 y VM4 con la orden `ping`.
- (Router) Analizar con Wireshark el tráfico intercambiado, especialmente los puertos y direcciones IP origen y destino en ambas redes

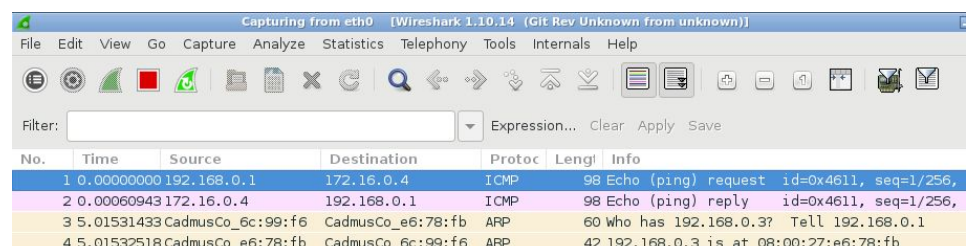
Adjuntar el comando `iptables` utilizado y una captura de pantalla de Wireshark.

[Router]

```
iptables -t nat -A POSTROUTING -o eth1 -j MASQUERADE
```

Hacemos `ping -c 1 172.16.0.4` en VM1

Por la interfaz `eth0` del router aparecen las direcciones tanto privadas como públicas (se ha deshecho bien la traducción):



No.	Time	Source	Destination	Protoc	Lengt	Info
1	0.00000000	192.168.0.1	172.16.0.4	ICMP	98	Echo (ping) request id=0x4611, seq=1/256, t
2	0.00060943	172.16.0.4	192.168.0.1	ICMP	98	Echo (ping) reply id=0x4611, seq=1/256, t
3	5.01531433	CadmusCo_6c:99:f6	CadmusCo_e6:78:fb	ARP	60	Who has 192.168.0.3? Tell 192.168.0.1
4	5.01532518	CadmusCo_e6:78:fb	CadmusCo_6c:99:f6	ARP	42	192.168.0.3 is at 08:00:27:e6:78:fb

Mientras que por la interfaz eth1 son todas públicas, utilizando la dirección Ip del router por esta interfaz donde debería ir la dirección privada 192.168.0.1 (se ha hecho bien la traducción mediante NAPT-Masquerade):

No.	Time	Source	Destination	Protoc	Lengt	Info
1	0.00000000	172.16.0.3	172.16.0.4	ICMP	98	Echo (ping) request id=0x4611, seq=1/256, ttl=6
2	0.00054123	172.16.0.4	172.16.0.3	ICMP	98	Echo (ping) reply id=0x4611, seq=1/256, ttl=6
3	5.00417502	CadmusCo_4c:ab:70	CadmusCo_76:ba:9f	ARP	42	Who has 172.16.0.4? Tell 172.16.0.3
4	5.00458527	CadmusCo_76:ba:9f	CadmusCo_4c:ab:70	ARP	60	172.16.0.4 is at 08:00:27:76:ba:9f

Ejercicio 13. ¿Qué parámetro se utiliza, en lugar del puerto origen, para relacionar las solicitudes con las respuestas? Comprueba la salida del comando `conntrack -L` o, alternativamente, el fichero `/proc/net/nf_conntrack`.

Adjuntar la salida del comando `conntrack` y responder a la pregunta.

Salida del comando:

```
icmp 1 27 src=192.168.0.1 dst=172.16.0.4 type=8 code=0 id=18606 src=172.16.0.4 dst=172.16.0.3
type=0 code=0 id=18606 mark=0 use=1
conntrack v1.4.4 (conntrack-tools): 1 flow entries have been shown.
```

Respuesta:

Utiliza el nº de identificación del ICMP recibido. En nuestro caso 18606.

Ejercicio 14. Acceso a un servidor en la red privada:

- (Router) Usando iptables, reenviar las conexiones (DNAT) del puerto 80 de Router al puerto 7777 de VM1. Iniciar una captura de Wireshark en los dos interfaces.
- (VM1) Arrancar el servidor en el puerto 7777 con nc.
- (VM4) Conectarse al puerto 80 de Router con nc y comprobar el resultado en VM1.
- (Router) Analizar con Wireshark el tráfico intercambiado, especialmente los puertos y direcciones IP origen y destino en ambas redes.

Adjuntar el comando iptables utilizado y una captura de pantalla de Wireshark.

Comando:

```
iptables -t nat -A PREROUTING -d 172.16.0.3 -p tcp --dport 80 -i eth1 -j DNAT --to 192.168.0.1:7777
```

Como podemos comprobar en la captura de wireshark hecha en router llega un SYN con puerto destino 80 con IP destino 172.16.0.3, pero al hacerse la traducción se cambia la IP destino 192.168.0.1 y el puerto a 7777.

No.	Time	Source	Destination	Protoc	Length	Info
1	0.00000000	172.16.0.4	172.16.0.3	TCP	74	42918 > http [SYN] Seq=0 Win=29200 Len=0 MSS=1460 SACK_PERM=1 TSval=28154875 TSecr=0 WS=128
2	0.00003206	172.16.0.4	192.168.0.1	TCP	74	42918 > cbt [SYN] Seq=0 Win=29200 Len=0 MSS=1460 SACK_PERM=1 TSval=28154875 TSecr=0 WS=128
3	0.00072354	192.168.0.1	172.16.0.4	TCP	62	cbt > 42918 [SYN, ACK] Seq=0 Ack=1 Win=29200 Len=0 MSS=1460 SACK_PERM=1
4	0.00073560	172.16.0.3	172.16.0.4	TCP	62	http > 42918 [SYN, ACK] Seq=0 Ack=1 Win=29200 Len=0 MSS=1460 SACK_PERM=1
5	0.00112603	172.16.0.4	192.168.0.1	TCP	54	42918 > cbt [ACK] Seq=1 Ack=1 Win=29200 Len=0
6	0.00108728	172.16.0.4	172.16.0.3	TCP	60	42918 > http [ACK] Seq=1 Ack=1 Win=29200 Len=0
7	5.00984551	CadmusCo_76:ba:9f	CadmusCo_4c:ab:70	ARP	60	Who has 172.16.0.3? Tell 172.16.0.4

Type: IP (0x0800)

- Internet Protocol Version 4, Src: 172.16.0.4 (172.16.0.4), Dst: 192.168.0.1 (192.168.0.1)
 - Version: 4
 - Header length: 20 bytes
 - Differentiated Services Field: 0x00 (DSCP 0x00: Default; ECN: 0x00: Not-ECT (Not ECN-Capable Transport))
 - Total Length: 60
 - Identification: 0xa4b9 (42169)
 - Flags: 0x02 (Don't Fragment)
 - Fragment offset: 0
 - Time to live: 63
 - Protocol: TCP (6)
 - Header checksum: 0x2a45 [validation disabled]
 - Source: 172.16.0.4 (172.16.0.4)
 - Destination: 192.168.0.1 (192.168.0.1)
- Transmission Control Protocol, Src Port: 42918 (42918), Dst Port: cbt (7777), Seq: 0, Len: 0
 - Source port: 42918 (42918)
 - Destination port: cbt (7777)
 - [Stream index: 1]
 - Sequence number: 0 (relative sequence number)
 - Header length: 40 bytes
 - Flags: 0x002 [SYN]
 - Window size value: 29200
 - [Calculated window size: 29200]
 - Checksum: 0x600f [validation disabled]
 - Options: (20 bytes), Maximum segment size, SACK permitted, Timestamps, No-Operation (NOP), Window scale