# 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).

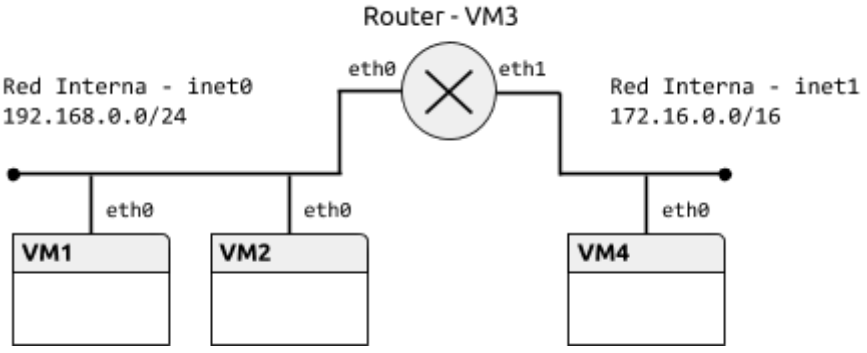
Las **credenciales de la máquina virtual** son: usuario `cursoredes`, con contraseña `cursoredes`.

### 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)`<br>`172.16.0.3/16 (eth1)` | Activar el *forwarding* de paquetes |
| VM4 | `172.16.0.4/16` | Añadir Router como encaminador por defecto |

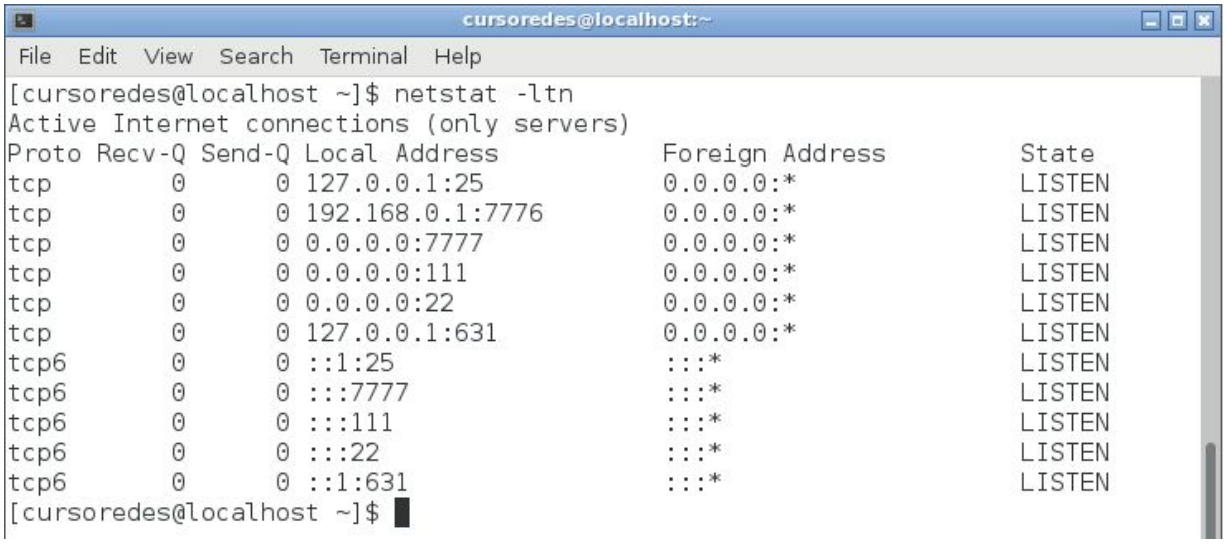
## Estados de una conexión TCP

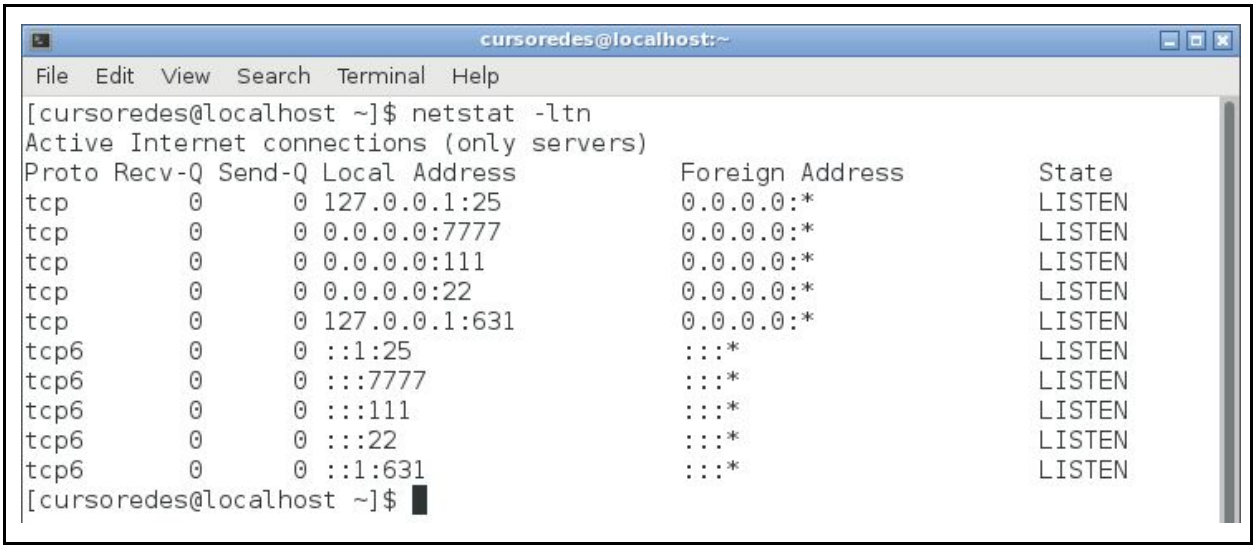
En esta parte usaremos la herramienta Netcat, que permite leer y escribir en conexiones de red. Netcat es muy útil para investigar y depurar el comportamiento de la red en la capa de transporte, ya que permite especificar un gran número de los parámetros de la conexión. Además para ver el estado de las 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.

***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.*



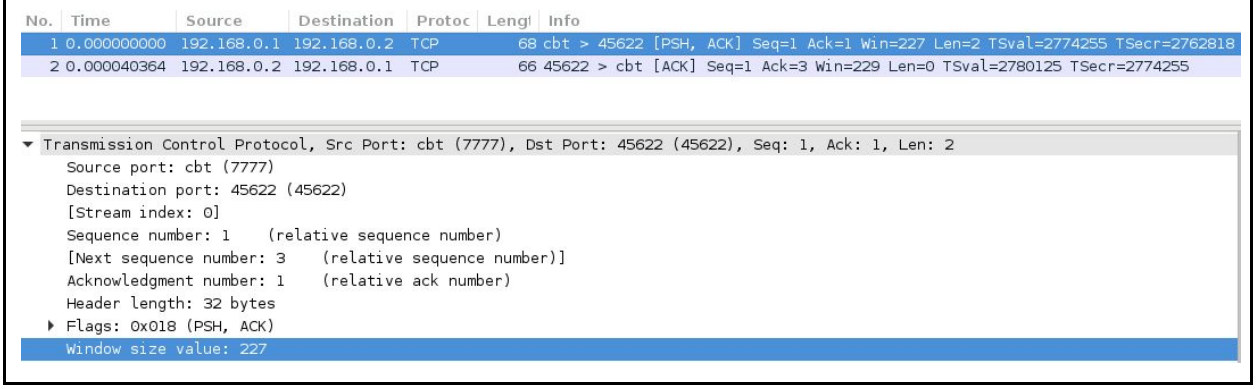



***Ejercicio 3.*** (ESTABLISHED) En VM2, iniciar una conexión cliente al primer servidor arrancado en el ejercicio anterior usando el comando `nc 192.168.0.1 7777`.

- 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.*


*La captura de pantalla de Wireshark.*




***Ejercicio 4.*** (TIME-WAIT) Cerrar la conexión en el cliente (con `Ctrl+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.*

```
[cursoredes@localhost ~]$ netstat -tano
Active Internet connections (servers and established)
Proto Recv-Q Send-Q Local Address           Foreign Address         State       Timer
tcp        0      0 127.0.0.1:25            0.0.0.0:*               LISTEN      off (0.00/0/0)
tcp        0      0 0.0.0.0:111             0.0.0.0:*               LISTEN      off (0.00/0/0)
tcp        0      0 0.0.0.0:22              0.0.0.0:*               LISTEN      off (0.00/0/0)
tcp        0      0 127.0.0.1:631           0.0.0.0:*               LISTEN      off (0.00/0/0)
tcp        0      0 192.168.0.2:45622       192.168.0.1:7777        TIME_WAIT   timewait (13.67/0/0)
tcp6       0      0 ::1:25                  :::*                    LISTEN      off (0.00/0/0)
tcp6       0      0 :::111                  :::*                    LISTEN      off (0.00/0/0)
tcp6       0      0 :::22                   :::*                    LISTEN      off (0.00/0/0)
tcp6       0      0 ::1:631                 :::*                    LISTEN      off (0.00/0/0)
```

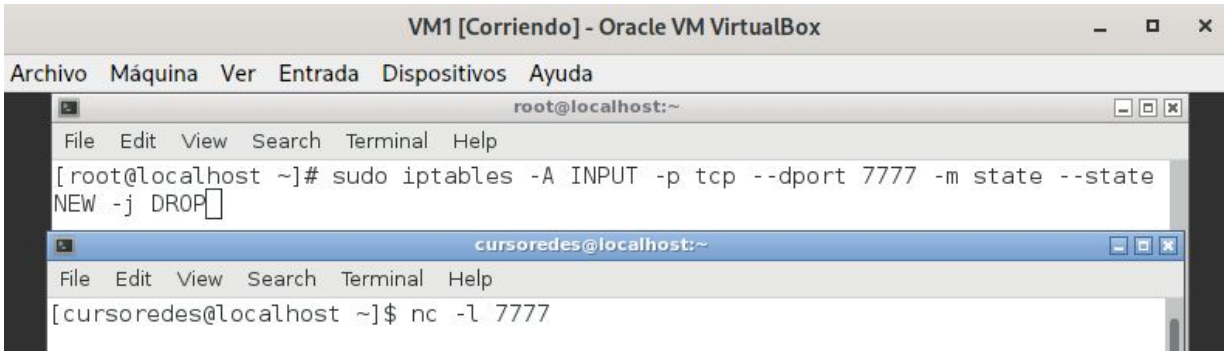
***Ejercicio 5.*** (SYN-SENT y SYN-RCVD) El comando `iptables` permite filtrar paquetes según los flags TCP del segmento con la opción `--tcp-flags` (consultar la página de manual `iptables-extensions`). Usando esta opción:

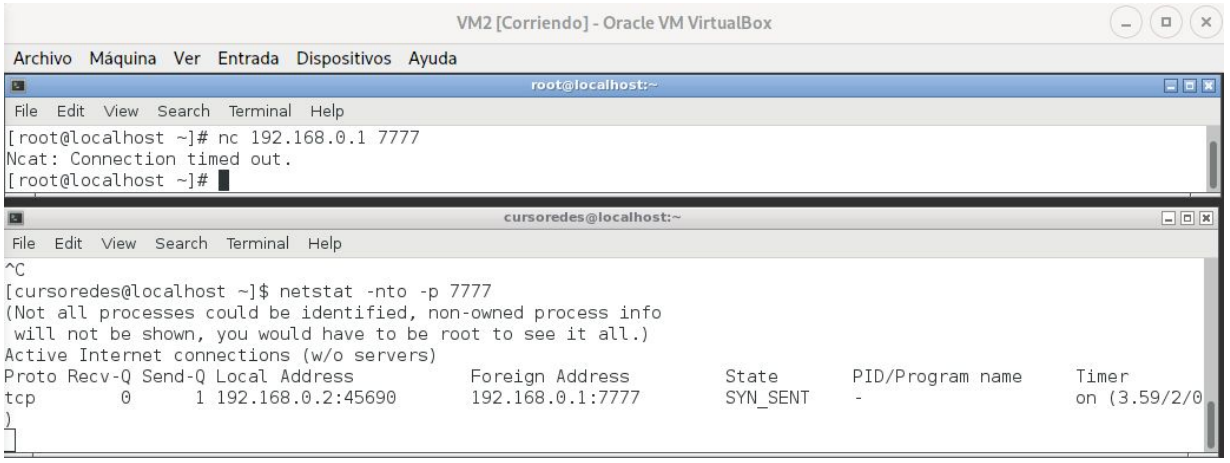
- Fijar una regla en el servidor (VM1) que bloquee un mensaje del acuerdo TCP de forma que el cliente (VM2) se quede en el estado SYN-SENT. Comprobar el resultado con `ss -tan` en el cliente.
- Borrar la regla anterior y fijar otra en el cliente que bloquee un mensaje del acuerdo TCP de forma que el 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.*

---

En el **primer apartado** tenemos que aplicar la regla con “iptables” a MV1 y a continuación crear la escucha con el puerto 7777 tanto en MV1 como en MV2. Finalmente hago un “netstat” en MV2 para ver que el “state” es “**SYN_SENT**”.





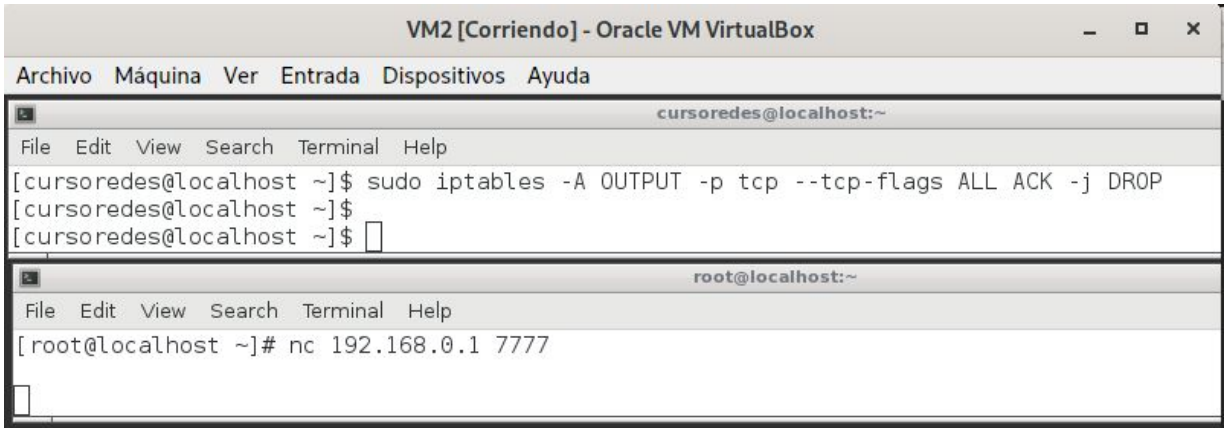

---

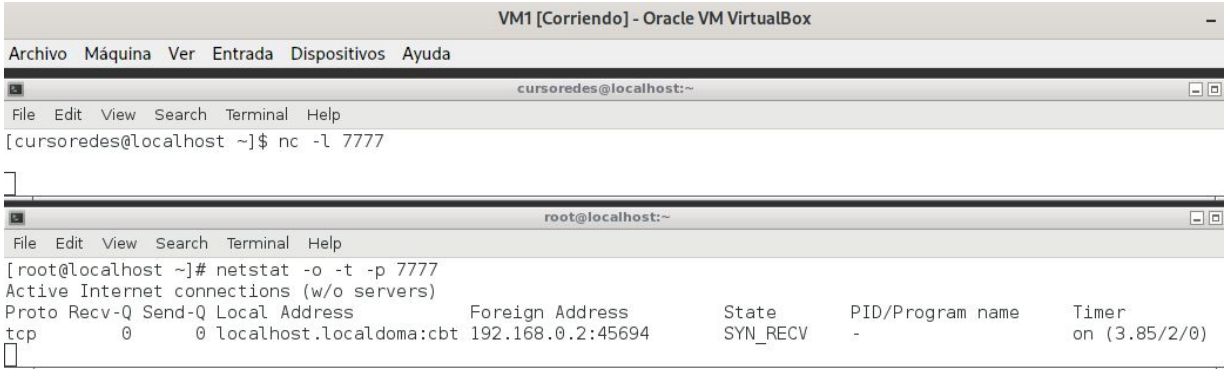
Para **borrar la regla anterior** hago:

- sudo iptables -L --line-numbers (Listo las reglas)
- sudo iptables -D INPUT 1 (Elimino la que me interesa)

---

En el segundo apartado, se aplica la regla en MV2 y a continuación se inicia la escucha en el puerto 7777. Finalmente en la MV1 aplico el comando "netstat -o -t -p 7777" y observo como el "state" es "**SYN_RECV**".



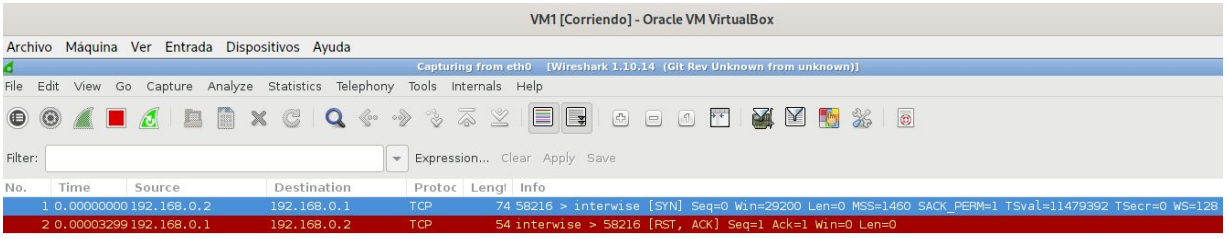
A continuación cierro la conexión en el **cliente** (**MV2**) y vuelvo a aplicar el comando **"netstat -o -t -p 7777"** observando esta vez como el **"state"** cambia a "**LAST_ACK**"

```
[root@localhost ~]# netstat -o -t -p 7777
Active Internet connections (w/o servers)
Proto Recv-Q Send-Q Local Address           Foreign Address         State       PID/Program name     Timer
tcp        0      1 localhost.localdoma:cbt 192.168.0.2:45694       LAST_ACK    -                    on (77.62/0/0)
```

***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.*

Desde **MV1** abrir el **Wireshark** y ver qué sucede cuando se aplica **"nc 192.168.0.1 7778"** desde **MV2**.




Observamos que el cliente envía un [SYN] y el Servidor responde con un [RST, ACK].

## 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.*

Desde VM2 aplicar siguiente regla para bloquear los mensajes SYN-ACK que provenga del servidor:

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

Desde VM2 aplicamos el siguiente comando (hping3):

- **hping3 -S --flood 192.168.0.1 -p 22**
  *( -S sirve para activar el flag TCP SYN)*

Desde VM1 aplicando el comando **"netstat -o -t -p 7777"** observamos como salen muchos SYN_RECV lo que significa que te están haciendo un **ataque** "TCP SYN flood" como se puede ver en la imagen posterior.

```
[root@localhost ~]# netstat -o -t -p 7777
Active Internet connections (w/o servers)
Proto Recv-Q Send-Q Local Address           Foreign Address         State       PID/Program name     Timer
tcp        0      0 localhost.localdoma:ssh 192.168.0.2:pqsflows    SYN_RECV    -                    on (1.53/4/0
tcp        0      0 localhost.localdoma:ssh 192.168.0.2:9601        SYN_RECV    -                    on (1.53/4/0
tcp        0      0 localhost.localdoma:ssh 192.168.0.2:9671        SYN_RECV    -                    on (1.53/4/0
tcp        0      0 localhost.localdoma:ssh 192.168.0.2:9567        SYN_RECV    -                    on (1.53/4/0
tcp        0      0 localhost.localdoma:ssh 192.168.0.2:9520        SYN_RECV    -                    on (1.53/4/0
```

Para evitar este ataque desde el servidor (VM1) aplicar el siguiente comando para desactivar el mecanismo SYN cookies:

- **sysctl net.ipv4.tcp_syncookies=1**

***Ejercicio 8.*** (Técnica CONNECT) Netcat 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.*

En **VM1** se aplica el siguiente comando para abrir un servidor en el puerto 7777:

- "**nc -l 7777"**.

En **VM2** se aplica el siguiente comando para explorar de uno en uno los puertos con **nmap:**

- **"nmap -p 7775-7780 192.168.0.1"**. *(La salida se ve en la siguiente imagen)*

VM2 [Corriendo] - Oracle VM VirtualBox

Archivo Máquina Ver Entrada Dispositivos Ayuda

root@localhost:~

File Edit View Search Terminal Help

```
[root@localhost ~]# nmap -p 7775-7780 192.168.0.1

Starting Nmap 6.40 ( http://nmap.org ) at 2020-10-21 18:16 CEST
Nmap scan report for 192.168.0.1
Host is up (0.00070s latency).
PORT     STATE  SERVICE
7775/tcp closed unknown
7776/tcp closed unknown
7777/tcp open   cbt
7778/tcp closed interwise
7779/tcp closed unknown
7780/tcp closed unknown
MAC Address: 08:00:27:D5:7C:CF (Cadmus Computer Systems)

Nmap done: 1 IP address (1 host up) scanned in 13.08 seconds
```

En **Wireshark** se puede observar lo siguiente:

VM1 [Corriendo] - Oracle VM VirtualBox

Archivo Máquina Ver Entrada Dispositivos Ayuda

Capturing from eth0 [Wireshark 1.10.14 (Git Rev Unknown from unknown)]

File Edit View Go Capture Analyze Statistics Telephony Tools Internals Help

Filter: Expression... Clear Apply Save

| No. | Time | Source | Destination | Protocol | Lengt | Info |
|---|---|---|---|---|---|---|
| 1 | 0.00000000 | CadmusCo_03:f4:d6 | Broadcast | ARP | 60 | Who has 192.168.0.1? Tell 192.168.0.2 |
| 2 | 0.00001724 | CadmusCo_d5:7c:cf | CadmusCo_03:f4:d6 | ARP | 42 | 192.168.0.1 is at 08:00:27:d5:7c:cf |
| 3 | 0.00067643 | 192.168.0.2 | 192.168.0.1 | DNS | 84 | Standard query 0xa696 PTR 1.0.168.192.in-addr.arpa |
| 4 | 0.00071027 | 192.168.0.1 | 192.168.0.2 | ICMP | 112 | Destination unreachable (Port unreachable) |
| 5 | 4.00173243 | 192.168.0.2 | 192.168.0.1 | DNS | 84 | Standard query 0xa696 PTR 1.0.168.192.in-addr.arpa |
| 6 | 4.00180017 | 192.168.0.1 | 192.168.0.2 | ICMP | 112 | Destination unreachable (Port unreachable) |
| 7 | 5.01557553 | CadmusCo_d5:7c:cf | CadmusCo_03:f4:d6 | ARP | 42 | Who has 192.168.0.2? Tell 192.168.0.1 |
| 8 | 5.01649531 | CadmusCo_03:f4:d6 | CadmusCo_d5:7c:cf | ARP | 60 | 192.168.0.2 is at 08:00:27:03:f4:d6 |
| 9 | 8.00180910 | 192.168.0.2 | 192.168.0.1 | DNS | 84 | Standard query 0xa696 PTR 1.0.168.192.in-addr.arpa |
| 10 | 8.00188064 | 192.168.0.1 | 192.168.0.2 | ICMP | 112 | Destination unreachable (Port unreachable) |
| 11 | 13.0032605 | 192.168.0.2 | 192.168.0.1 | TCP | 60 | 51694 > 7775 [SYN] Seq=0 Win=1024 Len=0 MSS=1460 |
| 12 | 13.0033359 | 192.168.0.1 | 192.168.0.2 | TCP | 54 | 7775 > 51694 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0 |
| 13 | 13.0033861 | 192.168.0.2 | 192.168.0.1 | TCP | 60 | 51694 > vstat [SYN] Seq=0 Win=1024 Len=0 MSS=1460 |
| 14 | 13.0034125 | 192.168.0.1 | 192.168.0.2 | TCP | 54 | vstat > 51694 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0 |
| 15 | 13.0034382 | 192.168.0.2 | 192.168.0.1 | TCP | 60 | 51694 > cbt [SYN] Seq=0 Win=1024 Len=0 MSS=1460 |
| 16 | 13.0034757 | 192.168.0.1 | 192.168.0.2 | TCP | 58 | cbt > 51694 [SYN, ACK] Seq=0 Ack=1 Win=29200 Len=0 MSS=1460 |
| 17 | 13.0035129 | 192.168.0.2 | 192.168.0.1 | TCP | 60 | 51694 > 7776 [SYN] Seq=0 Win=1024 Len=0 MSS=1460 |
| 18 | 13.0035400 | 192.168.0.1 | 192.168.0.2 | TCP | 54 | 7776 > 51694 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0 |
| 19 | 13.0035693 | 192.168.0.2 | 192.168.0.1 | TCP | 60 | 51694 > 7780 [SYN] Seq=0 Win=1024 Len=0 MSS=1460 |
| 20 | 13.0035926 | 192.168.0.1 | 192.168.0.2 | TCP | 54 | 7780 > 51694 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0 |
| 21 | 13.0036170 | 192.168.0.2 | 192.168.0.1 | TCP | 60 | 51694 > interwise [SYN] Seq=0 Win=1024 Len=0 MSS=1460 |
| 22 | 13.0036390 | 192.168.0.1 | 192.168.0.2 | TCP | 54 | interwise > 51694 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0 |
| 23 | 14.0034643 | 192.168.0.1 | 192.168.0.2 | TCP | 58 | [TCP Retransmission] cbt > 51694 [SYN, ACK] Seq=0 Ack=1 Win=29200 Len=0 MSS=1460 |
| 24 | 16.0034612 | 192.168.0.1 | 192.168.0.2 | TCP | 58 | [TCP Retransmission] cbt > 51694 [SYN, ACK] Seq=0 Ack=1 Win=29200 Len=0 MSS=1460 |
| 25 | 20.4064702 | 192.168.0.1 | 192.168.0.2 | TCP | 58 | [TCP Retransmission] cbt > 51694 [SYN, ACK] Seq=0 Ack=1 Win=29200 Len=0 MSS=1460 |
| 26 | 28.4125622 | 192.168.0.1 | 192.168.0.2 | TCP | 58 | [TCP Retransmission] cbt > 51694 [SYN, ACK] Seq=0 Ack=1 Win=29200 Len=0 MSS=1460 |
| 27 | 44.4264878 | 192.168.0.1 | 192.168.0.2 | TCP | 58 | [TCP Retransmission] cbt > 51694 [SYN, ACK] Seq=0 Ack=1 Win=29200 Len=0 MSS=1460 |

***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.

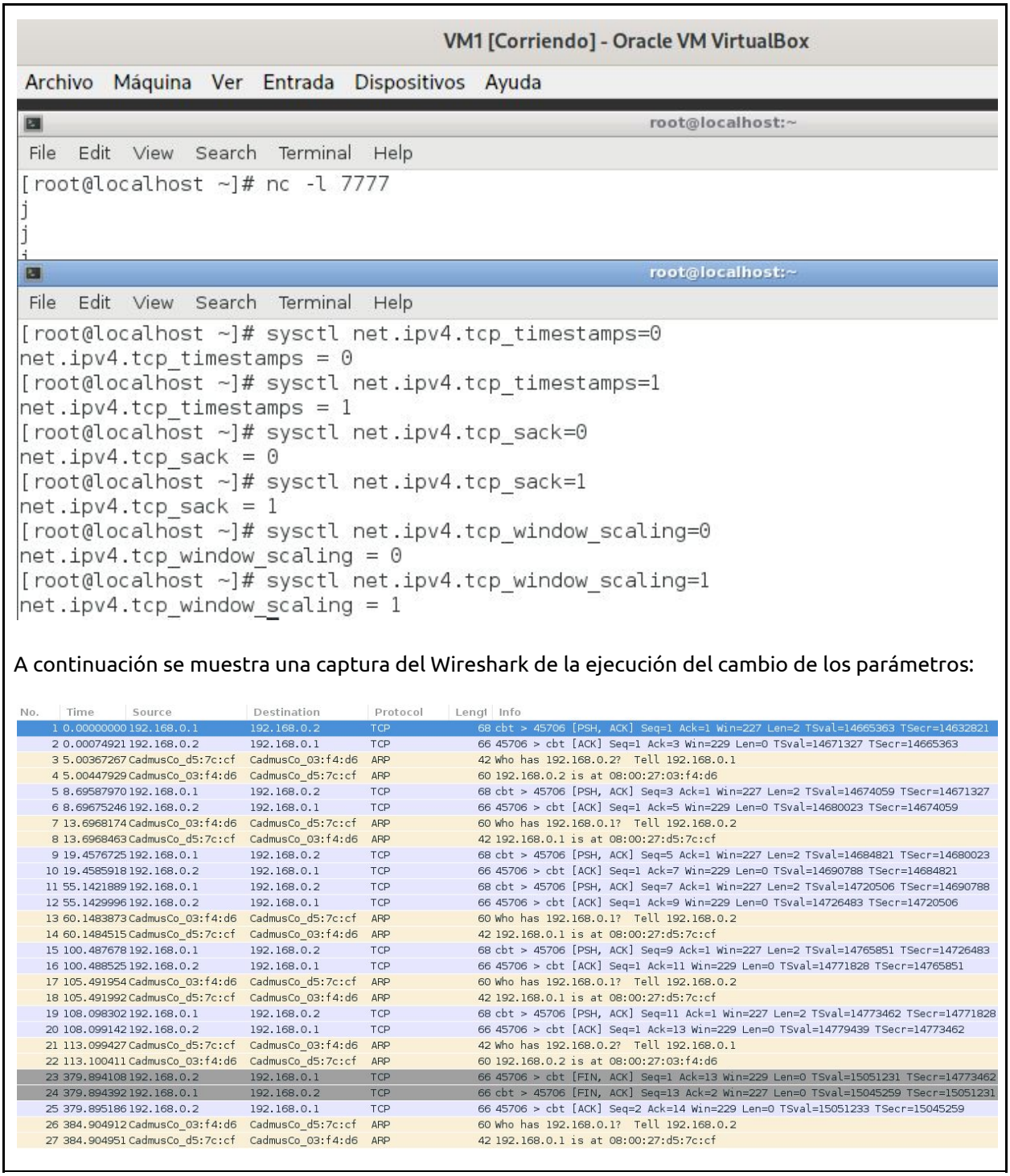
***Ejercicio 9.*** Con ayuda del comando `sysctl` y la bibliografía recomendada, completar la siguiente tabla con parámetros que permiten modificar algunas opciones de TCP:

| Parámetro del kernel | Propósito | Valor por defecto |
|---|---|---|
| `net.ipv4.tcp_window_scaling` | Permite aumentar el tamaño de la RW a mas de 64K, algo necesario para altas velocidades de transmisión con alta latencia. | 1 |
| `net.ipv4.tcp_timestamps` | Sirve para ayudar a TCP a determinar en qué orden se enviaron los paquetes. | 1 |
| `net.ipv4.tcp_sack` | Sirve para activar los SACK: Selective-ACK | 1 |

***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 muestran las opciones TCP.*

Desde **VM1** se pone el servidor a la escucha en el puerto 7777 con **"nc -l 7777"**. A continuación se inicia la escucha desde **VM2** aplicando **"nc 192.168.0.1 7777"** y finalmente se aplican los cambios en los parámetros como se ve en la siguiente imagen.




A continuación se muestra una captura del Wireshark de la ejecución del cambio de los parámetros:



| No. | Time | Source | Destination | Protocol | Lengt | Info |
|---|---|---|---|---|---|---|
| 1 | 0.00000000 | 192.168.0.1 | 192.168.0.2 | TCP | 68 | cbt > 45706 [PSH, ACK] Seq=1 Ack=1 Win=227 Len=2 TSval=14665363 TSecr=14632821 |
| 2 | 0.00074921 | 192.168.0.2 | 192.168.0.1 | TCP | 66 | 45706 > cbt [ACK] Seq=1 Ack=3 Win=229 Len=0 TSval=14671327 TSecr=14665363 |
| 3 | 5.00367267 | CadmusCo_d5:7c:cf | CadmusCo_03:f4:d6 | ARP | 42 | Who has 192.168.0.2? Tell 192.168.0.1 |
| 4 | 5.00447929 | CadmusCo_03:f4:d6 | CadmusCo_d5:7c:cf | ARP | 60 | 192.168.0.2 is at 08:00:27:03:f4:d6 |
| 5 | 8.69587970 | 192.168.0.1 | 192.168.0.2 | TCP | 68 | cbt > 45706 [PSH, ACK] Seq=3 Ack=1 Win=227 Len=2 TSval=14674059 TSecr=14671327 |
| 6 | 8.69675246 | 192.168.0.2 | 192.168.0.1 | TCP | 66 | 45706 > cbt [ACK] Seq=1 Ack=5 Win=229 Len=0 TSval=14680023 TSecr=14674059 |
| 7 | 13.6968174 | CadmusCo_03:f4:d6 | CadmusCo_d5:7c:cf | ARP | 60 | Who has 192.168.0.1? Tell 192.168.0.2 |
| 8 | 13.6968463 | CadmusCo_d5:7c:cf | CadmusCo_03:f4:d6 | ARP | 42 | 192.168.0.1 is at 08:00:27:d5:7c:cf |
| 9 | 19.4576725 | 192.168.0.1 | 192.168.0.2 | TCP | 68 | cbt > 45706 [PSH, ACK] Seq=5 Ack=1 Win=227 Len=2 TSval=14684821 TSecr=14680023 |
| 10 | 19.4585918 | 192.168.0.2 | 192.168.0.1 | TCP | 66 | 45706 > cbt [ACK] Seq=1 Ack=7 Win=229 Len=0 TSval=14690788 TSecr=14684821 |
| 11 | 55.1421889 | 192.168.0.1 | 192.168.0.2 | TCP | 68 | cbt > 45706 [PSH, ACK] Seq=7 Ack=1 Win=227 Len=2 TSval=14720506 TSecr=14690788 |
| 12 | 55.1429996 | 192.168.0.2 | 192.168.0.1 | TCP | 66 | 45706 > cbt [ACK] Seq=1 Ack=9 Win=229 Len=0 TSval=14726483 TSecr=14720506 |
| 13 | 60.1483873 | CadmusCo_03:f4:d6 | CadmusCo_d5:7c:cf | ARP | 60 | Who has 192.168.0.1? Tell 192.168.0.2 |
| 14 | 60.1484515 | CadmusCo_d5:7c:cf | CadmusCo_03:f4:d6 | ARP | 42 | 192.168.0.1 is at 08:00:27:d5:7c:cf |
| 15 | 100.487678 | 192.168.0.1 | 192.168.0.2 | TCP | 68 | cbt > 45706 [PSH, ACK] Seq=9 Ack=1 Win=227 Len=2 TSval=14765851 TSecr=14726483 |
| 16 | 100.488525 | 192.168.0.2 | 192.168.0.1 | TCP | 66 | 45706 > cbt [ACK] Seq=1 Ack=11 Win=229 Len=0 TSval=14771828 TSecr=14765851 |
| 17 | 105.491954 | CadmusCo_03:f4:d6 | CadmusCo_d5:7c:cf | ARP | 60 | Who has 192.168.0.1? Tell 192.168.0.2 |
| 18 | 105.491992 | CadmusCo_d5:7c:cf | CadmusCo_03:f4:d6 | ARP | 42 | 192.168.0.1 is at 08:00:27:d5:7c:cf |
| 19 | 108.098302 | 192.168.0.1 | 192.168.0.2 | TCP | 68 | cbt > 45706 [PSH, ACK] Seq=11 Ack=1 Win=227 Len=2 TSval=14773462 TSecr=14771828 |
| 20 | 108.099142 | 192.168.0.2 | 192.168.0.1 | TCP | 66 | 45706 > cbt [ACK] Seq=1 Ack=13 Win=229 Len=0 TSval=14779439 TSecr=14773462 |
| 21 | 113.099427 | CadmusCo_d5:7c:cf | CadmusCo_03:f4:d6 | ARP | 42 | Who has 192.168.0.2? Tell 192.168.0.1 |
| 22 | 113.100411 | CadmusCo_03:f4:d6 | CadmusCo_d5:7c:cf | ARP | 60 | 192.168.0.2 is at 08:00:27:03:f4:d6 |
| 23 | 379.894108 | 192.168.0.2 | 192.168.0.1 | TCP | 66 | 45706 > cbt [FIN, ACK] Seq=1 Ack=13 Win=229 Len=0 TSval=15051231 TSecr=14773462 |
| 24 | 379.894392 | 192.168.0.1 | 192.168.0.2 | TCP | 66 | cbt > 45706 [FIN, ACK] Seq=13 Ack=2 Win=227 Len=0 TSval=15045259 TSecr=15051231 |
| 25 | 379.895186 | 192.168.0.2 | 192.168.0.1 | TCP | 66 | 45706 > cbt [ACK] Seq=2 Ack=14 Win=229 Len=0 TSval=15051233 TSecr=15045259 |
| 26 | 384.904912 | CadmusCo_03:f4:d6 | CadmusCo_d5:7c:cf | ARP | 60 | Who has 192.168.0.1? Tell 192.168.0.2 |
| 27 | 384.904951 | CadmusCo_d5:7c:cf | CadmusCo_03:f4:d6 | ARP | 42 | 192.168.0.1 is at 08:00:27:d5:7c:cf |

***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 |
|---|---|---|
| `net.ipv4.tcp_keepalive_time` | Tiempo necesario (en silencio) para revisar si una conexión sigue viva. | 7200 |
| `net.ipv4.tcp_keepalive_probes` | Número de pruebas tras el tiempo anterior superado, para ver si sigue | 9 |

| | | |
|---|---|---|
| | viva la conexión. | |
| `net.ipv4.tcp_keepalive_intvl` | Intervalo entre pruebas | 9 |

## 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:

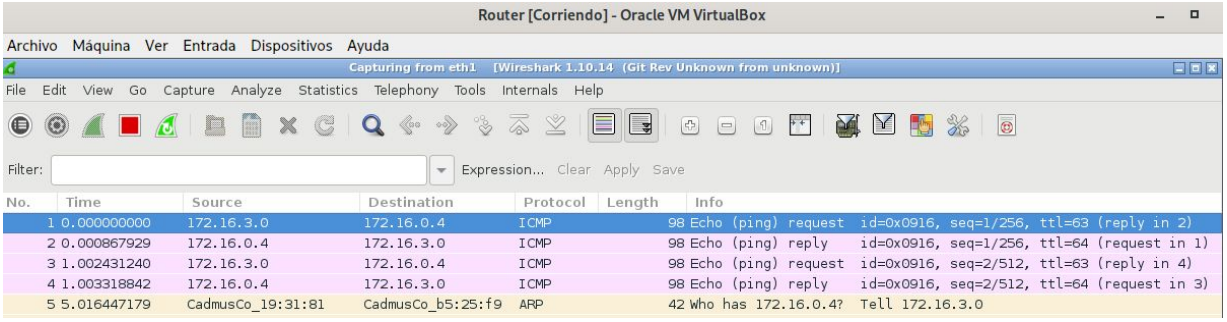
- (Router) Usando `iptables`, configurar Router para que haga SNAT (*masquerade*) sobre la interfaz `eth1`. Iniciar una captura de Wireshark en los dos interfaces.
- (VM1) Comprobar la conexión entre VM1 y VM4 con la orden `ping`.
- (Router) Analizar con Wireshark el tráfico intercambiado, especialmente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

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

Comando:

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

En **eth0** no se ve **ningún** paquete.
En **eth1** se ve lo siguiente:




***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.*

Comandos:

- **conntrack -L**
- **sudo cat /proc/net/nf_conntrack**

El parámetro que se utiliza es: **-q, --reply-dst IP_ADDRESS**

***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.*

- **iptables -t nat -A PREROUTING -p tcp --dport 80 -j DNAT --to 192.168.0.1:7777**
- **"nc -l 7777"** (en VM1)
- **nc -l 80** *(en VM4)*
- ***netstat -ltn*** *(en VM1)*

---

*Captura del Wireshark ...*