Práctica 2

Configuración avanzada de IP

En esta práctica estudiaremos algunos de los parámetros de IP referentes a la fragmentación y el reensamblado de datagramas, el protocolo de mensajes de control ICMP (*Internet Control Messages Protocol*) y cómo modificar el comportamiento de IP mediante la orden `sysctl`.

2.1. Ajuste de parámetros del kernel mediante `sysctl`

La orden `sysctl` se emplea para modificar parámetros del núcleo (*kernel*) en tiempo de ejecución. Los parámetros que pueden modificarse son los que aparecen en el sistema de ficheros virtual `/proc/sys`.

Si ejecutamos la orden `sysctl -a` podemos ver todos los parámetros disponibles:

```
/sbin/sysctl -a
kernel.panic = 0
kernel.core_uses_pid = 0
kernel.core_pattern = core
kernel.tainted = 0
error: permission denied on key 'kernel.cap-bound'
kernel.real-root-dev = 0
kernel.ctrl-alt-del = 0
kernel.printk = 7       4       1       7
kernel.modprobe = /sbin/modprobe
kernel.hotplug =
...
vm.stat_interval = 1
vm.vdso_enabled = 2
net.core.wmem_max = 131071
net.core.rmem_max = 131071
net.core.wmem_default = 110592
```

```
net.core.rmem_default = 110592
net.core.dev_weight = 64
net.core.netdev_max_backlog = 1000
net.core.message_cost = 5
net.core.message_burst = 10
net.core.optmem_max = 10240
net.core.xfrm_aevent_etime = 10
net.core.xfrm_aevent_rseqth = 2
...
net.ipv4.tcp_timestamps = 1
net.ipv4.tcp_window_scaling = 1
net.ipv4.tcp_sack = 1
net.ipv4.tcp_retrans_collapse = 1
net.ipv4.ip_forward = 0
net.ipv4.ip_default_ttl = 64
net.ipv4.ip_no_pmtu_disc = 0
...
```

En esta práctica estaremos interesados sólo en los parámetros relativos a la configuración de red. Podemos verlos con la orden `sysctl -a | grep ^net`. Para examinar el valor de un parámetro concreto, por ejemplo, el de `net.ipv4.ip_default_ttl`, se puede usar la orden:

```
$ /sbin/sysctl net.ipv4.ip_default_ttl
net.ipv4.ip_default_ttl = 64
```

Para modificar el valor de un parámetro en concreto, se debe especificar la opción `-w` y el parámetro que deseamos modificar seguido de un signo '=' y el nuevo valor:

```
# sysctl -w net.ipv4.ip_default_ttl=128
net.ipv4.ip_default_ttl = 128
```

Ejercicio: Comprobar el valor del parámetro `net.ipv4.ip_default_ttl`. Iniciar la aplicación `wireshark` y configurar dos máquinas virtuales, `uml1` y `uml2`. Hacer un `ping` desde una máquina a la otra y comprobar que el campo `TTL` de la cabecera IP de los datagramas enviados coincide con el valor de dicho parámetro.

2.2. Protocolo ICMP

El protocolo ICMP (*Internet Control Message Protocol*, Protocolo de Mensajes de Control de Internet) es un protocolo complementario a IP que se emplea para informar de errores y gestión de la red. Una peculiaridad de este protocolo es que sólo informa de la situación, pero no indica qué debe hacerse para corregirla.

Las principales funciones asociadas a ICMP son las siguientes:

- comprobación de equipo alcanzable,
- informe de errores,
- control de congestión,
- solicitud de cambio de ruta.

En esta práctica estudiaremos la comprobación de equipo alcanzable, el informe de errores relacionados con la fragmentación y el reensamblado de datagramas, y los mensajes asociados a la solicitud de cambio de ruta.

2.2.1. Comprobación de equipo alcanzable

Cuando un destino específico no responde, se puede comprobar si el equipo es alcanzable en la capa IP. La máquina origen envía un ICMP de **solicitud de eco** (*echo request*) al destino; cuando éste lo recibe, responde con una **respuesta de eco** (*echo reply*). En la figura 2.1 se puede ver el formato de estos mensajes ICMP. El campo de datos es opcional y puede tener una longitud variable (por defecto,

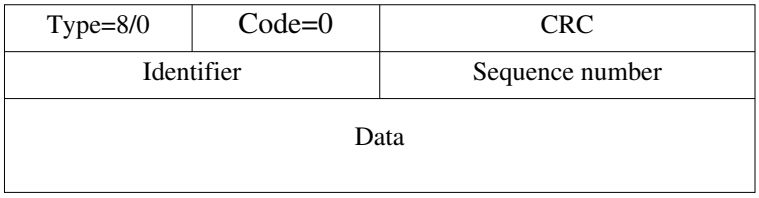

FIGURA 2.1: Formato de los mensajes ICMP para *echo request* (`Type=8`) y *echo reply* (`Type=0`)

56 bytes). Cuando la solicitud de eco llega al destino, el protocolo ICMP cambia el tipo a respuesta y devuelve el mensaje al origen, dejando intactos los demás campos. Si el origen recibe esta respuesta, sabe que existe una ruta hasta el destino y que al menos la capa IP de éste está configurada y funcionando.

Podemos generar mensajes de solicitud de eco con la orden `ping`. Por ejemplo:

```
$ ping -c 3 -s 30 192.168.0.21
PING 192.168.0.21 (192.168.0.21) 30(58) bytes of data.
38 bytes from 192.168.0.21: icmp_seq=1 ttl=255 time=0.392 ms
38 bytes from 192.168.0.21: icmp_seq=2 ttl=255 time=0.310 ms
38 bytes from 192.168.0.21: icmp_seq=3 ttl=255 time=0.414 ms

--- 192.168.0.21 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 1999ms
rtt min/avg/max/mdev = 0.310/0.372/0.414/0.044 ms
```

El parámetro `-c` indica el número de mensajes que se deben enviar y con `-s`, la longitud del campo de datos. El resultado de la orden es una estadística del número de mensajes devueltos y tiempo de ida y vuelta.

Existe la posibilidad de realizar una solicitud de eco a toda una subred. Para ello es preciso especificar el parámetro `-b`. Podemos controlar la respuesta a los mensajes de petición de eco mediante dos parámetros del núcleo:

1. net.ipv4.icmp_echo_ignore_all=0

2. net.ipv4.icmp_echo_ignore_broadcasts=1

Con estos valores por defecto, la primera indica si deben ignorarse o no todas las peticiones de eco. La segunda determina si deben ignorarse sólo las solicitudes de eco dirigidas a direcciones de difusión.

Ejercicio: Configurar tres máquinas virtuales, `uml1`, `uml2` y `uml3` para que estén en la misma red local, por ejemplo, en la `192.168.1.0/24`. Probar a hacer solicitudes de eco desde `uml1` a toda la subred local, activando y desactivando `net.ipv4.icmp_echo_ignore_broadcasts` mediante `sysctl` en `uml2` y `uml3`, y comprobar los resultados.

2.3. Redirección ICMP

En esta práctica comprobaremos el envío de mensajes ICMP de tipo *redirect* por parte de un encaminador cuando detecta que existe una ruta mejor accesible desde la misma red. Crearemos una configuración de máquinas virtuales como la mostrada en la figura 2.2. Los encaminadores `uml2` y `uml3` pueden utilizar

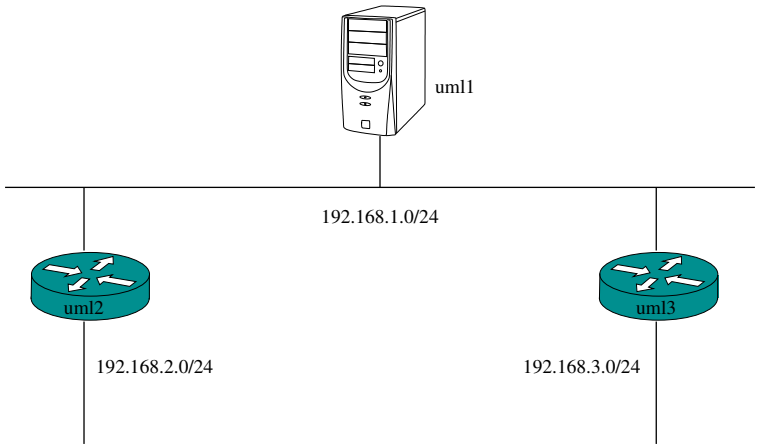

FIGURA 2.2: Red virtual para la prueba de ICMP

RIP para adquirir la tabla de encaminamiento. Para ello hay que realizar la configuración de `quagga`. Si no, se deben incluir las rutas de manera estática:

```
uml2 # ip route add 192.168.3.0/24 via 192.168.1.3
uml3 # ip route add 192.168.2.0/24 via 192.168.1.2
```

La máquina `uml1` debe fijar como encaminador predeterminado a `uml2`:

```
uml1 # ip route add default via 192.168.1.2
```

Iniciar `wireshark` en las máquinas para inspeccionar el tráfico.

Los encaminadores `uml2` y `uml3` deben tener activado el *forwarding*. Desde la máquina `uml1` se hace un `traceroute` a la dirección IP 192.168.3.3. Observar el tráfico que se genera y la tabla de rutas *cache* de la máquina `uml1`. Como podemos ver, el encaminador `uml2` envió un ICMP de tipo *Redirect* a la máquina `uml1`, pero ésta no incluyó ninguna entrada en su tabla de rutas, por lo que siguió enviando el tráfico a través de `uml2`. Existen dos variables `sysctl` que determinan el comportamiento en estos casos:

- `net.ipv4.conf.all.accept_redirects=1`
  `net.ipv4.conf.eth0.accept_redirects=1`
- `net.ipv4.conf.eth0.secure_redirects=0`

La primera especifica si deben aceptarse o no los mensajes ICMP de tipo *Redirect* de manera global; la segunda, si deben aceptarse sólo los mensajes de redirección que provengan del encaminador predeterminado a través del interfaz `eth0`. Fijémoslas a los valores indicados, y repitamos el `traceroute`. Veremos cómo ahora sí se ha incluido una entrada nueva en la tabla de encaminamiento *cache* de `uml1`:

```
uml1:~# ip route show table cache
192.168.3.3 via 192.168.1.3 dev eth0  src 192.168.1.1
    cache <redirected>  mtu 1500 advmss 1460 hoplimit 64
local 192.168.1.1 from 192.168.1.2 dev lo  src 192.168.1.1
    cache <local,src-direct>  iif eth0
192.168.3.3 from 192.168.1.1 via 192.168.1.3 dev eth0
    cache <redirected>  ipid 0xe408 mtu 1500 advmss 1460 hoplimit 64
local 192.168.1.1 from 192.168.3.3 dev lo  src 192.168.1.1
    cache <local>  iif eth0
local 192.168.1.1 from 192.168.1.3 dev lo  src 192.168.1.1
    cache <local,src-direct>  iif eth0
local 192.168.1.1 from 192.168.3.3 dev lo  src 192.168.1.1
    cache <local>  iif eth1
```

2.4. Fragmentación y reensamblado

Sabemos que un datagrama IP puede ocupar hasta 65535 bytes (el límite del campo Longitud de 16 bits). Los datagramas deben atravesar las distintas redes involucradas, cada una con su respectiva MTU (*Maximum Transmission Unit*). Cuando la longitud de datagrama excede la longitud de la MTU,

debe fragmentarse. En el destino se recibirán los fragmentos y deberá reconstruirse el datagrama original (proceso de reensamblado).

La fragmentación puede realizarse en dos lugares: en el origen del datagrama o en alguno de los encaminadores intermedios. Cuando se produce en el origen, la máquina debe conocer la mínima MTU hasta el destino, lo que se conoce como **descubrimiento de MTU** (*path MTU discovery*).

2.4.1. Descubrimiento de MTU

La mayor parte de los sistemas modernos activan por defecto el bit DF (*Disable Fragmentation*, Deshabilitar Fragmentación) y utilizan el procedimiento de descubrimiento de MTU (descrito en el RFC 1191). El mecanismo es sencillo:

- La máquina origen envía el datagrama ajustándose a la MTU de su red local y con el bit DF activado.
- Si el datagrama debe atravesar una red con una MTU menor, el encaminador correspondiente se encuentra con que no puede fragmentarlo, al estar el bit DF activado, por lo que descarta el datagrama y envía un mensaje ICMP **destino inalcanzable**, subtipo **fragmentación necesaria**, al origen. Este mensaje ICMP contiene la MTU de la red a la que no se pudo llegar.
- El origen envía un nuevo datagrama, más corto, que se ajusta a la MTU de la red remota. El bit DF se activa de nuevo.
- El proceso se repite hasta que el datagrama llega al destino (no se recibe ningún ICMP de fragmentación).

Se puede controlar el uso o no del algoritmo de descubrimiento de MTU mediante el parámetro del kernel `net.ipv4.ip_no_pmtu_disc`. Cuando dicho parámetro vale 0, que es su valor predeterminado, está activado el descubrimiento de MTU.

```
# sysctl net.ipv4.ip_no_pmtu_disc
net.ipv4.ip_no_pmtu_disc = 0
```

Ejercicio: Utilizar las órdenes `ip` e `ifconfig` para averiguar la MTU de las redes locales.

Ejercicio: Activar 4 máquinas virtuales y configurarlas como indica la figura 2.3. Las máquinas `uml2` y `uml3` son encaminadores. Pueden adquirir sus rutas mediante RIP, o bien configurarlas a mano. La máquina `uml1` debe usar a `uml2` como encaminador predeterminado, y `uml4` usará a `uml3`. La red que une ambos encaminadores tiene una `MTU` de 600 bytes. Se debe utilizar la orden `ifconfig` para configurar la `mtu` correspondiente en los interfaces `eth1` de los encaminadores.

Para realizar esta práctica, por el funcionamiento de los interfaces virtuales en Linux, no nos sirve la configuración básica de la red virtual, por lo que debemos usar la configuración avanzada. Para ello, creamos un archivo, llamado por ejemplo `net.conf`, de manera apropiada.

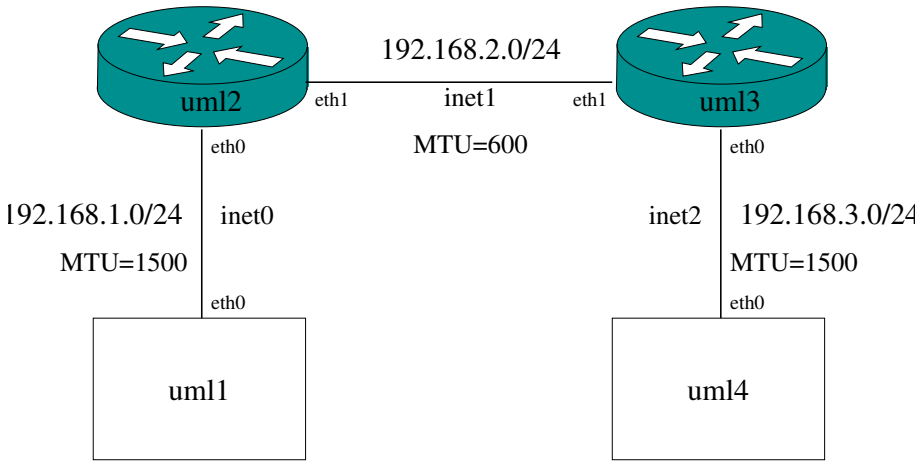

FIGURA 2.3: Configuración para la prueba de fragmentación

Si ahora hacemos `ping` desde la máquina `uml1` hasta `uml4`, debería ser accesible. Iniciamos `wireshark` y lo ponemos a escuchar en el interfaz `eth0` en la máquina `uml1`.

Suponiendo que hemos configurado la máquina `uml4` con la dirección IP `192.168.3.4`, ejecutar ahora desde `uml1`:

```
$ ping -c 3 -s 800 192.168.3.4
```

y estudiar el resultado.

Si `net.ipv4.ip_no_pmtu_disc` vale 0 (valor por defecto), se activa el descubrimiento de MTU. Cuando `uml1` hace `ping` con el parámetro `-s 800`, se envía un mensaje ICMP de tipo *echo request* con un campo de datos de 800 bytes. El primer datagrama lleva el bit `DF` activado y una longitud de 828 bytes (800 de datos, 8 de cabecera ICMP y 20 de cabecera IP). La máquina `uml2` lo descarta y envía un ICMP con `Type==3` (*Destination unreachable*), `Code==4` (*Fragmentation needed*) y `MTU of next hop==600`. Ahora `uml1` fragmentará el datagrama **en origen**, apareciendo dos fragmentos en el interfaz `eth0` (figura 2.4).

Nota: Una vez que se ha descubierto la MTU para esa ruta se añade una entrada en la tabla de encaminamiento *cache*. Se puede comprobar con la orden `ip route show table cache`. Si queremos repetir el proceso, previamente debemos borrar la entrada en la tabla. Podemos vaciar la tabla con la orden `ip route flush table cache`.

Cuando `net.ipv4.ip_no_pmtu_disc==1`, los datagramas que parten de `uml1` no llevan activado el bit `DF`, por lo que `uml2` puede fragmentarlos sin problemas (figura 2.5). Podemos comprobarlo inspeccionando con `wireshark` el tráfico en el interfaz `uml2.1`.

Cuando el encaminador `uml2` incluye una regla en su cortafuegos para descartar el segundo fragmento (y sucesivos):

```
iptables -I FORWARD -f -j DROP
```

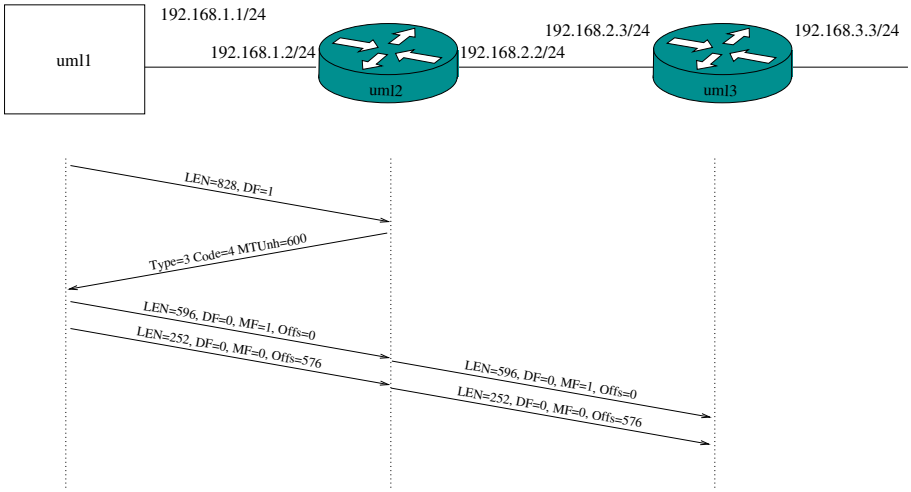

FIGURA 2.4: Descubrimiento de MTU (fragmentación en el origen)

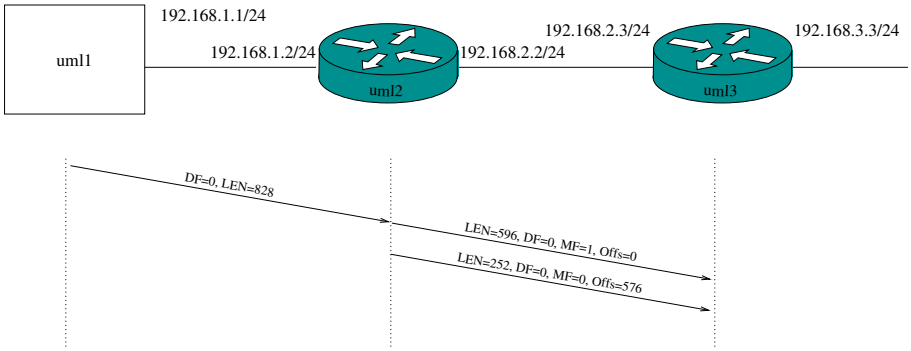

FIGURA 2.5: Fragmentación en el encaminador

sólo dejará pasar el primer fragmento de cada datagrama, simulando de esta manera la pérdida de un fragmento. Si `uml1` no fragmenta en origen, no habrá problema, pero si lo hace, `uml2` no permitirá pasar el segundo fragmento. Cuando `uml4` recibe el primero, queda a la espera de los siguientes (el bit `MF` viene activado). Cuando se cumple el temporizador de ensamblado, descarta el datagrama y envía a `uml1` un mensaje ICMP con `Type==11` (*Time–to–live exceeded*) y `Code==1` (*Fragment reassembly time exceeded*) (figura 2.6).

El temporizador de reensamblado se controla con la variable `net.ipv4.ipfrag_time` (por defecto, 30 segundos). Se puede repetir el proceso anterior modificando el valor de esta variable en la máquina `uml4`.

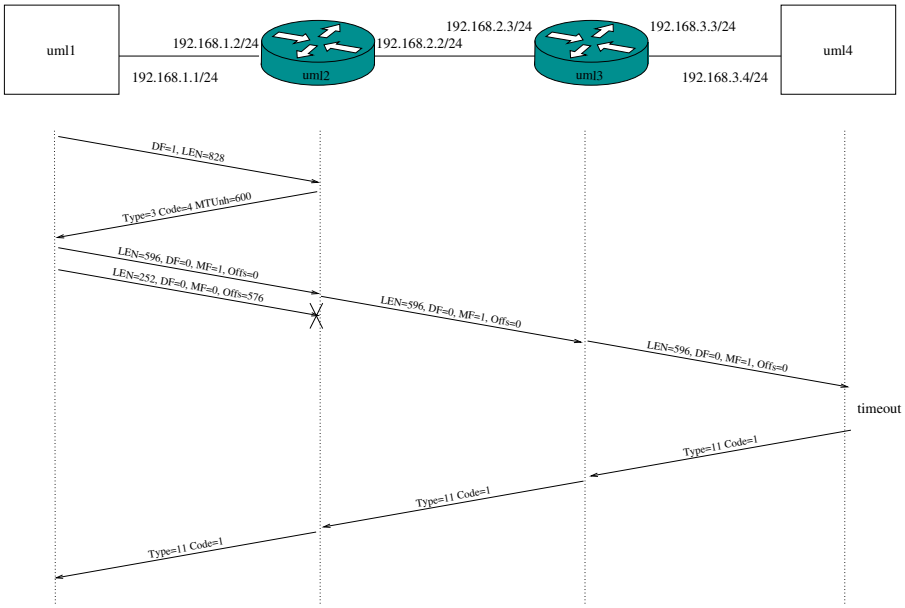

FIGURA 2.6: Pérdida de un fragmento