Capítulo 5

Protocolos esenciales

Al terminar este capítulo, entenderás:

- Qué función cubre cada uno los protocolos esenciales.
- Cómo es el formato de los mensajes de esos protocolos.
- Qué es y cómo funciona la encapsulación de mensajes.
- Qué es el direccionamiento lógico y físico.

Los protocolos esenciales son las auténticas tripas de Internet, y como tal, no suelen estar a la vista de los usuarios, pero sin ellos, Internet no sería posible, o sería muy diferente. Estos protocolos han condicionado la evolución de Internet para bien y para mal. Su diseño abierto y simple es en buena medida responsable de su gran éxito, rápida adopción, flexibilidad y de la siempre cuestionada «neutralidad de la red».

Sin embargo, ese mismo diseño también es responsable de sus limitaciones y problemas, como la ausencia de mecanismos de seguridad integrados o la escalabilidad limitada en ciertos aspectos. Hay que tener en cuenta que estos protocolos fueron diseñados para una red experimental, sin siquiera plantear el gran y rápido crecimiento que tendría. Aunque la tecnología de Internet ha evolucionado mucho desde sus comienzos, y se han añadido extensiones y protocolos nuevos para cubrir deficiencias y nuevas necesidades, lo cierto es que la parte esencial sigue funcionando sobre la misma base de protocolos que se diseñaron hace más de 40 años, lo que da una idea de la calidad de su diseño.

En este libro vamos a utilizar el modelo híbrido como guía (Figura 5.1). Esta figura incluye también los protocolos más importantes de TCP/IP, que vamos a estudiar a lo largo del libro. En la capa de enlace incluye otros dos protocolos: PPP y Ethernet, que aunque no sean parte de TCP/IP son muy utilizados y parte esencial también del funcionamiento de Internet.

La implementación de estos protocolos es muy variopinta. Los protocolos de enlace se implementan típicamente en el hardware de la NIC, los de red

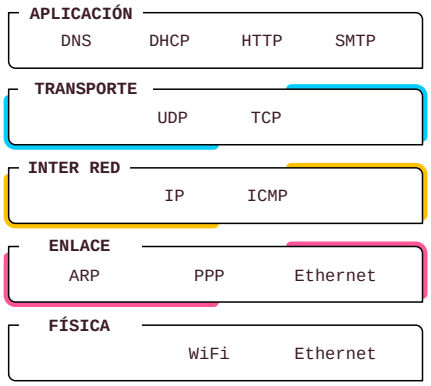

FIGURA 5.1: Modelo híbrido con los protocolos esenciales

y transporte en el subsistema de red, en el núcleo del sistema operativo, mientras que los de aplicación suelen estar disponibles en forma de librerías, o en el código de las propias aplicaciones si son muy específicos. Pero todo eso puede cambiar de un sistema a otro, por ejemplo los routers de gama media/alta implementan la mayoría de su funcionalidad (incluidos los protocolos de red) por medio de ASIC para conseguir mayor rendimiento y menor consumo.

También aprovecharemos esta primera toma de contacto con los protocolos esenciales para explicar cómo se realiza la comprobación de la conectividad en cada capa y protocolo, ya que además de ser técnicamente relevante, ayuda a entender mejor el rol que juega cada protocolo.

La **conectividad** es la capacidad de dos o más (normalmente dos) dispositivos o procesos para intercambiar información entre ellos. Cuando existe un problema de conectividad, no es posible realizar la comunicación. La comprobación de conectividad es una actividad habitual para aislar y localizar problemas en las comunicaciones; y también como forma efectiva de monitorizar el buen funcionamiento del equipamiento y aplicaciones de red.

5.1. Encapsulación

La encapsulación es un mecanismo fundamental de la pila de protocolos. Conceptualmente consiste en colocar el mensaje completo de la capa como «carga útil» del mensaje de la capa inmediatamente inferior. Un ejemplo real sería un datagrama UDP completo que se coloca como carga útil de un

paquete IP, que a su vez se coloca como carga útil de una trama Ethernet. La Figura 5.2 muestra este ejemplo.

Como analogía se puede pensar en un sobre utilizado por el servicio postal. En el exterior del sobre se indica la información del destinatario y en el interior se coloca la carga útil. Esa carga útil podría ser perfectamente otro sobre con indicaciones específicas de la entrega y así sucesivamente. En el último sobre tendríamos el mensaje que el usuario emisor realmente quiere enviar al usuario destino.

En la práctica, los campos de un protocolo, incluido el de carga útil, no son más que una secuencia de bytes, sin embargo, la información que contienen las cabeceras de cada protocolo permite determinar en qué punto de esa secuencia se encuentra cada campo y dónde comienza cada uno de los mensajes encapsulados. Veremos ejemplos concretos de esto más adelante.

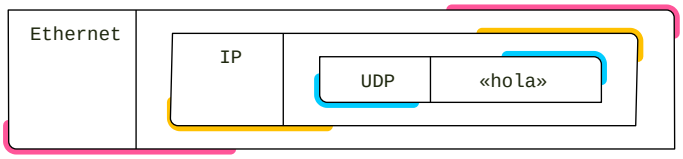

FIGURA 5.2: Ejemplo de encapsulación

Veamos cómo llevar esto a la práctica. Utilizaremos `tshark`, una aplicación de análisis de protocolos (*sniffer*), para capturar el tráfico de red. En un terminal ejecuta el siguiente comando:

```
$ sudo tshark -c 1 -V -f "udp dst port 2000"
```

Para no entrar ahora en demasiados detalles, sirva decir que lo que estamos haciendo es capturar un único datagrama UDP dirigido al puerto 2000. El comando `sudo` es necesario porque la captura de tráfico requiere de permisos especiales[1]. Veremos con detalle cómo capturar y analizar tráfico en el capítulo 18.

Una vez tienes la captura en marcha, en otra consola puedes forzar la generación de un datagrama UDP con `netcat`.

```
$ echo hola | ncat -u example.net 2000
```

[1] aunque `sudo` no es la forma más recomendable de hacerlo, nos sirve de momento. Consulta 18.1.1 para una explicación detallada.

Esto crea un mensaje que reproduce exactametne la Figura 5.2, es decir, coloca el texto «hola» como carga útil de un datagrama UDP, que a su vez es la carga útil de un paquete IP, que a su vez es la carga útil de una trama Ethernet. El resultado de la captura de `tshark` será algo parecido al Listado 5.1. Tanto en este como en posteriores listados, hemos eliminado partes de la captura que no son relevantes para el concepto que abordamos en este momento. Aún así te recomendamos hacer la captura por ti mismo en tu propio computador para ver el resultado completo.

```
1  $ sudo tshark -c 1 -f "udp dst port 2000" -nVx
2  Frame 1: 47 bytes on wire (376 bits) on interface eth0, id 0
3      Interface id: 0 (eth0)
4      Encapsulation type: Ethernet (1)
5      Frame Length: 47 bytes (376 bits)
6      [Protocols in frame: eth:ethertype:ip:udp:data]
7
8  Ethernet II, Src: f8:5f:2a:c0:ff:ee, Dst: fc:99:47:ad:f0:0d
9      Type: IPv4 (0x0800)
10
11 Internet Protocol Version 4, Src: 192.168.0.37, Dst: 93.184.215.14
12     0100 .... = Version: 4
13     .... 0101 = Header Length: 20 bytes (5)
14     Total Length: 33
15     Time to Live: 64
16     Protocol: UDP (17)
17     Header Checksum: 0x49e0 [validation disabled]
18     Source Address: 192.168.0.37
19     Destination Address: 93.184.215.14
20
21 User Datagram Protocol, Src Port: 33262, Dst Port: 2000
22     Source Port: 33262
23     Destination Port: 2000
24     Length: 13
25     UDP payload (5 bytes)
26
27 0000  68 6f 6c 61 0a                                    hola.
28     Data: 686f6c610a
29
30 0000  fc 99 47 ad f0 0d f8 5f 2a c0 ff ee 08 00 45 00   ......,.]!xl..E.
31 0010  00 21 8c 61 40 00 40 11 b0 83 c0 a8 00 25 5d b8   .!.a@.@......%].
32 0020  d7 0e cd e5 07 d0 00 0d 4d 66 68 6f 6c 61 0a      ........Mfhola.
```

LISTADO 5.1: Captura de un datagrama UDP con `tshark`

Se pueden apreciar 6 secciones en la captura, que corresponden respectivamente a cada uno de los protocolos involucrados y por último al mensaje completo representado en hexadecimal. Los vamos a analizar someramente ahora y después los iremos desgranando en las siguientes secciones de este mismo capítulo.

- **Línea 2**: muestra un resumen del mensaje completo. Indica la cantidad de bytes y bits capturados, la interfaz de red, el tipo de trama y

la cadena de protocolos encapsulados: `eth:ethertype:ip:udp:data`, es decir, una trama Ethernet, que contiene un paquete IP, que contiene un datagrama UDP, que contiene datos.

- **Línea 8**: aparece información de la cabecera Ethernet, con sus direcciones MAC origen y destino, y tipo de carga.
- **Línea 11**: muestra información de la cabecera IP, con las direcciones IP origen y destino y otros campos relevantes.
- **Línea 21**: ofrece información de la cabecera UDP.
- **Línea 28**: representa la carga útil del datagrama UDP, que en este caso es el texto «hola» codificado en ASCII, es decir, la secuencia hexadecimal `686f6c610a`, aunque `tshark` también lo ofrece como texto a la derecha. El punto al final de «hola.» corresponde al byte `0a` que es un salto de línea `\n`.
- Por último, aparece el mensaje completo. En el margen izquierdo muestra el offset de cada fila con un valor de 4 dígitos. En la parte central representa el contenido organizado en filas de 16 bytes, y a la derecha el mismo contenido, pero codificado como texto ASCII, siempre que corresponda con un símbolo del código; en otro caso muestra puntos.

Esto que hemos hecho enviando un texto directamente como carga útil no es lo más habitual. Normalmente el datagrama UDP contiene un mensaje de algún protocolo de aplicación (*p. ej.* DNS), pero lo hemos hecho así para que el ejemplo sea ultra simple, y de todos modos, esto es algo factible, y no contradice ninguna norma o principio, hay protocolos sencillos y reales que hacen cosas similares.

Otro detalle que merece la pena destacar es que `example.net` aunque existe, no tiene ningún servidor UDP escuchando en el puerto 2000, pero por el modo en que funciona UDP, eso no impide que podamos crear el datagrama y enviarlo. Obviamente no va a llegar a ninguna parte y esto es buena prueba de las garantías que ofrece UDP: ninguna.

5.2. Protocolos de enlace

Los protocolos de la capa de enlace de datos se encargan de mover mensajes dentro del enlace local —por ejemplo, en la LAN a la que está conectado nuestro computador. Por tanto lleva mensajes desde un computador o equipo de comunicaciones hasta sus vecinos[2].

[2] dispositivos cercanos que pueden ser alcanzados sin la intervención de intermediarios

Para el modelo TCP/IP, la capa de enlace (incluida en lo que llama «host a red») simplemente se encarga —por medio del protocolo correspondiente— de llevar paquetes IP entre vecinos de la misma red, obviando todos los detalles que ello implica. El modelo OSI por otra parte, sí que entra a estudiar todos los problemas que implica la comunicación en un enlace local.

Dependiendo del medio físico se distingue entre dos tipos de medios:

Punto a punto que permite enviar mensajes únicamente entre dos dispositivos. Un ejemplo sencillo de esto puede ser un enlace de infrarrojos.

Difusión (o *broadcasting*) se caracterizan por el uso de un medio físico compartido por más de dos dispositivos. Esto permite que un mismo mensaje pueda llegar a todos o a un subconjunto determinado de vecinos. El aire en una comunicación WIFI es un ejemplo de medio físico compartido.

Los medios de difusión son típicos de las redes LAN aunque se da en otros ámbitos, como por ejemplo, las comunicaciones satelitales. En función del tipo de medio físico, se pueden clasificar también los protocolos y tecnologías de enlace. Así, por ejemplo, Ethernet es un protocolo de enlace de difusión, mientras que PPP es un protocolo punto a punto.

En los medios de difusión se requiere una dirección física para determinar para qué nodo es el mensaje, mientras que en los enlaces punto a punto no es necesario, porque los mensajes solo tienen un sitio a dónde ir: el otro extremo.

Los mensajes que circulan por un enlace se llaman «tramas» (*frames*) y son construidos por las NIC a partir de los mensajes que les llegan desde la capa superior. Normalmente, las NIC también se encargan de convertir estas tramas en señales eléctricas, ópticas o de radiofrecuencia que se transmiten por el medio físico.

Cada tecnología de enlace tiene sus limitaciones y características específicas como puede ser la velocidad de transmisión (también llamado «ancho de banda»), la distancia máxima que puede haber entre los dispositivos conectados o la cantidad máxima de dispositivos que pueden estar conectados.

5.3. Ethernet

Con mucha diferencia sobre el resto, las tecnologías LAN más utilizadas en la actualidad son Ethernet IEEE 802 y WiFi que, salvando las distancias, podríamos ver como una especie de «Ethernet inalámbrica».

Ethernet cubre los detalles tecnológicos de la transmisión de señales (la capa física), pero también la funcionalidad, codificación y formato de los mensajes (la capa de enlace) que es en lo que nos vamos a centrar en este punto.

5.3.1. Interfaces de red

Para conectar nuestro computador a una LAN Ethernet o cualquier otra tecnología de enlace como WiFi, Bluetooth, GSM, LTE o incluso infrarrojos, necesitamos hardware específico, es decir, una interfaz de red (NIC) que soporte Ethernet, WiFi, o la tecnología correspondiente, y también un controlador (*driver*) de dispositivo que se encargue de la gestión y transferencia de datos con el SO.

Hoy en día en el ámbito del PC o *smartphone*, muchos de estos dispositivos suelen estar integrados en la placa base (especialmente WiFi y Bluetooth), pero también están disponibles como periféricos externos, normalmente mediante conexión USB.

FIGURA 5.3: Interfaz de red Ethernet a USB

La forma más sencilla de ver con qué interfaces de red cuenta el nodo es el comando `ip addr`[3] (del paquete `iproute2`) como en el ejemplo del Listado 5.2. Se omite la información no relevante en este momento. Puedes probar a ejecutarlo en tu propio computador para ver la salida completa en tu caso.

```
1 $ ip addr
2 1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN
3     link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
4     inet 127.0.0.1/8 scope host lo
5 2: eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc fq_codel state UP
6     link/ether f8:5f:2a:c0:ff:ee brd ff:ff:ff:ff:ff:ff
7     inet 192.168.0.37/24 brd 192.168.0.255 scope global dynamic noprefixroute eth0
```

LISTADO 5.2: Salida del comando `ip addr`

En la salida puedes ver dos secciones diferenciadas, una especie de «fichas», para cada una de las dos interfaces de red que ha encontrado, en este caso: `lo` y `eth0`. La interfaz `eth0` corresponde a una NIC de tipo Ethernet (**línea 7**) con la dirección física: `f8:5f:2a:c0:ff:ee`.

Si hubiera una segunda NIC Ethernet se llamaría `eth1`, y del mismo modo las NIC WiFi tienen nombres como `wlan0`, aunque dependiendo de la versión del SO y su configuración puede tener otros nombres como `eno1`, `enp0s32d5`, `wlp1s0`, etc.

5.3.2. Trama Ethernet

El formato de la trama Ethernet (Figura 5.4) es bastante simple.

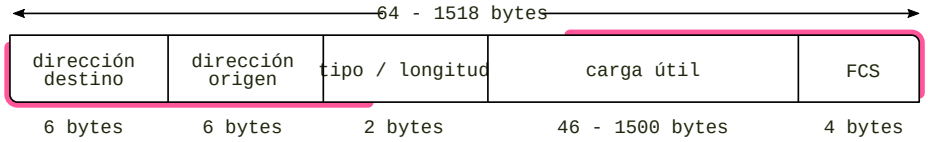

FIGURA 5.4: Formato de la trama Ethernet

Un pequeño resumen de los campos que forman la cabecera:

- Las direcciones MAC de las NIC origen y destino.
- Tipo/longitud puede indicar el tamaño en bytes del campo *payload* o bien un código que indica de que tipo (qué protocolo) es la carga útil.

[3] Se puede abreviar como `ip a`.

- La carga útil (*payload*) que es de tamaño variable (mínimo 46 bytes y máximo 1500). En este campo se coloca un mensaje de la capa superior, en Internet un paquete IP, aunque puede transportar otras cosas, como ARP.
- FCS, una especie de suma de comprobación que permite al receptor verificar que la trama no ha sufrido cambios.

Observa de nuevo en la parte de la captura 5.1 que corresponde a la cabecera Ethernet.

```
Ethernet II, Src: f8:5f:2a:c0:ff:ee, Dst: fc:99:47:ad:f0:0d
  Type: IPv4 (0x0800)
```

Aparecen claramente las direcciones origen y destino y el tipo, que en este caso es `0x0800` e indica que la carga útil es un paquete IP. Este campo se conoce comúnmente como *EtherType*[4]. El tamaño de la cabecera Ethernet es por tanto de 14 bytes (6 para dirección origen, 6 para dirección destino y 2 para tipo).

Obviando todos los detalles del funcionamiento de Ethernet, que no son pocos, el servicio que ofrece es simple: lleva la trama hasta la NIC con la dirección destino que aparece en la cabecera, siempre que sea un vecino de la misma LAN. Las tramas Ethernet no pueden salir nunca de la LAN en la que se crearon.

Podemos ver la trama Ethernet como un contenedor en el que colocar una secuencia de bytes arbitraria (el campo *payload*) y Ethernet se encargará de llevar esa trama hasta el computador destino. La NIC emisora calcula el FCS y lo añade al final. Al llegar a su destino, la NIC receptora realiza el mismo cálculo; si corresponde con el valor FCS, la trama es correcta y será entregada a la capa superior, probablemente IP. Si el resultado de ese cálculo no coincide, la trama será descartada...y eso es todo. Ethernet no informa del error a nadie, no pide una retransmisión, no hay temporizadores ni confirmaciones. Si una trama se corrompe durante su viaje, el receptor la descartará sin más. Obviamente habrá muchas aplicaciones que no puedan tolerar ese comportamiento, pero resolver ese problema no es responsabilidad de Ethernet.

5.3.3. Direcciones MAC

En los protocolos de enlace de medio compartido (como Ethernet) se identifica cada dispositivo con una dirección física o hardware. En el caso de

[4]Puedes encontrar la lista oficial de tipos Ethernet en `https://www.iana.org/assignments/ieee-802-numbers/ieee-802-numbers.xhtml`

Ethernet, también se le llama «dirección MAC» o simplemente «dirección Ethernet».

Se trata de un número grabado por el fabricante en la memoria ROM de la NIC. La dirección es única, no existen dos tarjetas Ethernet con la misma dirección en todo el mundo, o al menos no deberían. Es lo que se llama un «identificador único global» o UUID. A pesar de eso, se utiliza únicamente para identificar esta tarjeta solo en el ámbito de la red LAN a la que está físicamente conectada. Todo esto es igualmente aplicable a una NIC WiFi y otros protocolos y tecnologías similares.

En concreto, la dirección MAC Ethernet es un número de 6 bytes que se representa como convenio como una lista de dígitos en hexadecimal separados por el carácter ':', por ejemplo: `f8:5f:2a:c0:ff:ee`. Puedes ver la dirección MAC de una interfaz de tu computador con el comando `ip`.

```
$ ip link show eth0
2: eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc fq_codel state UP
    link/ether f8:5f:2a:c0:ff:ee brd ff:ff:ff:ff:ff:ff
```

La dirección MAC está formada por dos partes:

OUI (3 bytes) Identifica al fabricante de la tarjeta. Este prefijo lo otorga la IANA. Puedes consultar la lista de todos los OUI asignados actualmente en su web[5].

NIC (3 bytes) Es un número de serie que el fabricante garantiza que es único en su producción.

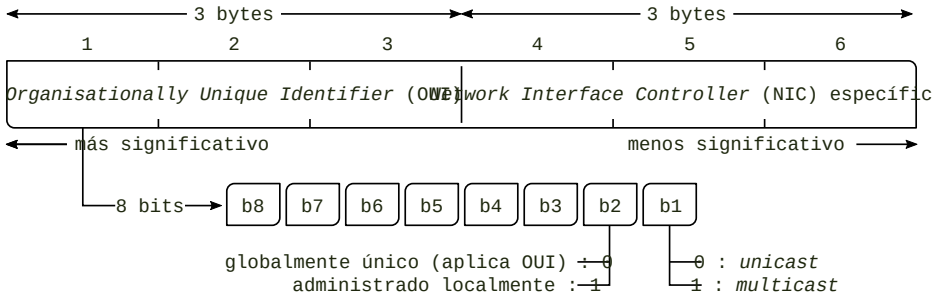

FIGURA 5.5: Formato de la dirección MAC Ethernet

Aparte de la entrega para un único destinatario (*unicast*), Ethernet proporciona otros dos métodos de entrega:

[5] `http://www.iana.org/assignments/ethernet-numbers`

broadcast Permite enviar un mensaje a todos los vecinos conectados a la misma LAN. Para ello se utiliza como destino una dirección especial que tiene todos sus bits a uno; en hexadecimal `FF:FF:FF:FF:FF:FF`.

multicast Permite enviar un mensaje a un subconjunto de los vecinos. Para ello se utilizan direcciones que tienen prefijos reservados.

5.3.4. Conectividad Ethernet

La comprobación de conectividad en el caso de Ethernet consiste en verificar que la NIC está en disposición de comunicarse con un vecino de la LAN por sí mismo, sin la participación de ningún otro protocolo.

Si tu computador dispone de una NIC Ethernet normalmente estará conectada a un conmutador (*switch*), ya sea directamente o a través de una roseta en la pared. Físicamente el conector 8P8C[6] hembra suele presentar dos LEDs (ver Figura 5.6), que ofrecen información interesante sobre el estado del enlace.

El LED de la izquierda aparece encendido si se ha podido establecer el enlace de datos con el vecino (el conmutador). Es habitual también que su color indique el modo (velocidad) a la que ha realizado la negociación: verde para 10 Mbps, naranja para 100 Mbps y amarillo para 1 Gbps.

El LED de la derecha (de color ámbar) parpadea cada vez que se transmite o recibe una trama.

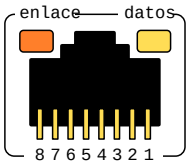

FIGURA 5.6: Conector 8P8C (RJ45) hembra de un equipo Ethernet

Aparte de la comprobación visual anterior, es posible preguntar al controlador del dispositivo con un programa específico. Uno de esos programas es `ethtool`. Lo siguiente es la salida de dicho programa para una interfaz Ethernet de un PC conectado a un conmutador.

```
1 # ethtool eth0
2 Settings for eth0:
3 Supported ports: [ TP ]
4 Supported link modes:   10baseT/Half 10baseT/Full
```

[6]El conector 8P8C se conoce más común, aunque erroneamente, como RJ45

```
5                          100baseT/Half 100baseT/Full
6                          1000baseT/Half 1000baseT/Full
7   Supports auto-negotiation: Yes
8   Advertised link modes:  10baseT/Half 10baseT/Full
9                          100baseT/Half 100baseT/Full
10                         1000baseT/Half 1000baseT/Full
11  Advertised pause frame use: No
12  Advertised auto-negotiation: Yes
13  Speed: 100Mb/s
14  Duplex: Full
15  Port: Twisted Pair
16  PHYAD: 1
17  Transceiver: internal
18  Auto-negotiation: on
19  MDI-X: Unknown
20  Supports Wake-on: g
21  Wake-on: g
22  Current message level: 0x000000ff (255)
23  Link detected: yes
```

Como puedes ver, se trata de un controlador Gigabit (**línea 6**) conectado a un conmutador (**línea 14**)[7] de 100 Mbps (**línea 14**) y el enlace de datos está correctamente establecido (**línea 23**).

5.4. PPP

PPP (Point to Point Protocol) [8] es un protocolo de enlace, pero a diferencia de Ethernet, se utiliza en enlaces punto a punto, como por ejemplo, una conexión directa entre dos routers, o entre un router doméstico y el ISP (la llamada «ultima milla»). También se utiliza de forma habitual sobre enlaces virtuales, túneles y VPNs.

Es un protocolo muy versátil, lo que explica sus usos tan variados:

- Tiene la capacidad de encapsular distintos protocolos de red.
- Puede configurar los detalles de esos protocolos de red (*p. ej.* la dirección IP del suscriptor) mediante los llamados NCP.
- Permite establecer, configurar, probar y cerrar el enlace, con el protocolo LCP.
- Ofrece mecanismos para autenticar al suscriptor mediante distintos protocolos de autenticación como PAP o CHAP.

El formato básico de la trama PPP es relativamente sencillo (Figura 5.7), aunque puede variar en función del enlace, opciones o protocolos que encapsule.

Por compatibilidad con HDLC los campos *Dirección* y *Control* tienen un valor fijo de `0xFF` y `0x03` respectivamente. El campo *Proto* es similar al

[7] Sólo se puede establecer un enlace *full duplex* es Ethernet conmutada, es decir, el PC está conectado a un conmutador (*switch*).

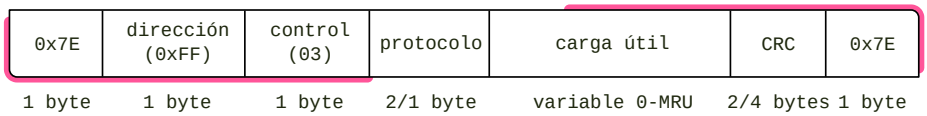

FIGURA 5.7: Formato de la trama PPP

campo *Type* de Ethernet e indica el tipo de carga útil: IP, IPX, AppleTalk, etc. El campo *Carga útil* es el mensaje encapsulado y *CRC* es un código de verificación. El flag `0x7E` indica el inicio y fin de cada mensaje.

5.5. IP

IP[8] es el protocolo de interred. Su tarea es llevar paquetes paquete IP desde un punto del planeta a cualquier otro mediante comunicación 'extremo a extremo' (*end-to-end*). Sin embargo, no ofrece garantías de entrega: IP es un protocolo *best-effort*. Eso significa que hará lo posible por llevar el paquete a su destino, pero bajo determinadas condiciones, lo «mejor posible» puede ser absolutamente nada.

IP es el acrónimo de 'Internet Protocol', pero cuidado, no significa 'el protocolo de la Internet', sino 'protocolo de interred', es decir, el que hace posible la interconexión de redes y la creación de interredes.

5.5.1. Interfaces de red

Las interfaces de red tienen una dirección lógica (una dirección IP), aparte de la dirección hardware que ya hemos visto. La dirección IP, en el caso de un dispositivo doméstico (PC, smartphone, etc.), se obtiene habitualmente de forma automática solicitándola a un servidor DHCP. Desde el punto de vista de su ámbito, se distingue entre 2 tipos de direcciones:

- **Privadas**: Se asignan a dispositivos inaccesibles desde Internet. Se usan en redes privadas y solo sirven para comunicarse con computadores dentro de la misma red o interred privada[9].
- **Públicas**: Se asignan a dispositivos accesibles desde cualquier parte. Se asignan a computadores directamente conectados a Internet, como servidores web, correo, etc.

[8] Cuando veas escrito «IP » estamos hablando de IPv4 (IP versión 4), que sigue siendo la más utilizada hoy en día. Para referirnos a 'IP versión 6' siempre lo escribiremos como IPv6.

[9] Existe un mecanismo llamado NAT que permite superar parcialmente esa limitación.

Como veremos más adelante, se han definido unos rangos de direcciones específicos para cada tipo de dirección.

En el caso de una red doméstica, el proveedor de servicios (el ISP) puede asignar una única dirección IP pública al *router* doméstico, y el *router* a su vez, asigna direcciones privadas a los dispositivos de la red doméstica.

En el caso del listado 5.2, la dirección IP de la interfaz `eth0` es `192.168.0.37`, que es privada.

5.5.1.1. Interfaz *loopback*

Cualquier SO que implemente IP ofrece una interfaz de red llamada *loopback*[10]. Es una interfaz de red bastante especial. Se dice que es una interfaz «virtual» porque no está ligada a ninguna NIC. A pesar de ello, se puede utilizar para cualquier servicio basado en TCP/IP teniendo en cuenta que su ámbito de trabajo está limitado al propio nodo, tal como indica la **línea 4**: `scope host` del Listado 5.2.

Como no hay una NIC asociada, la interfaz *loopback* no tiene dirección física y tiene siempre la dirección IP especial `127.0.0.1`, independiente del tipo de computador o sistema operativo. El nombre simbólico para esa dirección IP es «localhost».

La **línea 2** muestra otros datos interesantes:

- Es de tipo bucle: `LOOPBACK`.
- Tiene una MTU de 16 436 bytes.

5.5.2. Paquete IP

El formato del paquete IP [9] se muestra en la Figura 5.8.

Veamos brevemente el significado y utilidad de cada campo:

`versión`
: Versión del protocolo. Siempre contiene el valor `4` porque estamos describiendo el protocolo IP versión 4, que es la que se utiliza desde el nacimiento de Internet y sigue siendo la de mayor uso en la actualidad, aunque convive con la versión 6.

IHL (Internet Header Length), es decir, el tamaño de la cabecera, pero expresado en palabras de 4 bytes. Eso significa que el valor mínimo posible es `5`, es decir, una cabecera estándar de 20 bytes, y el máximo es 15 que sería una cabecera de 60 bytes.

ToS (Type of Service) contiene información útil para priorizar el tráfico en función de su tipo.

[10] que podríamos traducir como «bucle local»

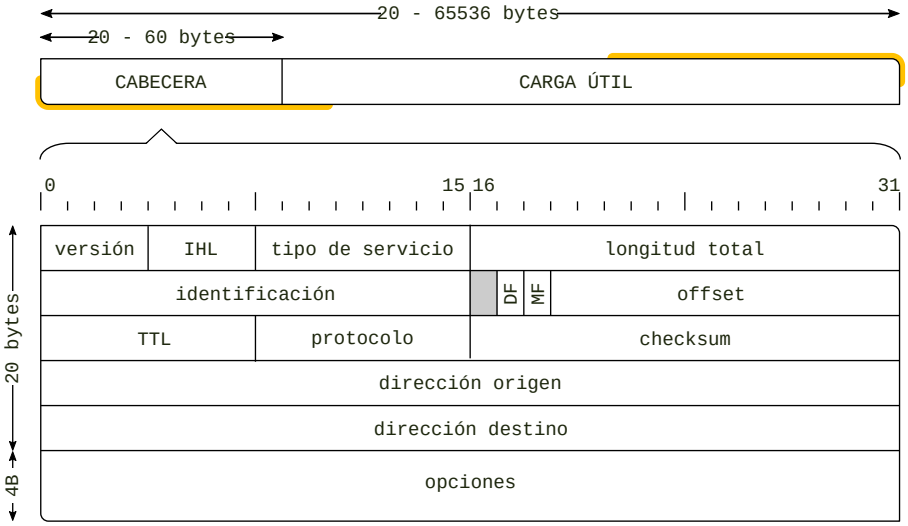

FIGURA 5.8: Formato del paquete IP

longitud total
: La longitud total del paquete incluyendo cabecera y carga útil. Como es un entero de 16 bits implica que el tamaño máximo de un paquete IP es 65 635 bytes.

identificación
: Un número que identifica a todos los fragmentos que proceden de un mismo paquete IP original.

DF (Don't Fragment), le indica a los routers que no deben fragmentar este paquete.

MF (More Fragments), indica que éste no es el último fragmento.

offset
: Indica qué posición ocupa este fragmento en el paquete original.

TTL Indica cuántos routers puede atravesar este paquete (saltos) antes de ser descartado.

protocolo
: Indica el protocolo al que corresponde la carga útil del paquete.

checksum
: Una suma de comprobación para verificar que la cabecera no ha sufrido cambios inesperados.

dirección origen
: La dirección IP del host que crea el paquete.

dirección destino
: La dirección IP del destinatario del paquete.

Analicemos la parte de la captura 5.1 que corresponde con la cabecera IP (esta vez incluyendo más detalles).

```
Internet Protocol Version 4, Src: 192.168.0.37, Dst: 93.184.215.14
    0100 .... = Version: 4
    .... 0101 = Header Length: 20 bytes (5)
    Identification: 0xcf0a (53002)
    010. .... = Flags: 0x2, Don't fragment
    ...0 0000 0000 0000 = Fragment Offset: 0
    Total Length: 33
    Time to Live: 64
    Protocol: UDP (17)
    Header Checksum: 0x49e0 [validation disabled]
    Source Address: 192.168.0.37
    Destination Address: 93.184.215.14
```

Muestra la versión (4), la longitud de la cabecera (5=20 bytes), el identificador (`0xcf0a`), el flag DF (no fragmentar), el offset del fragmento (0), la longitud total del paquete (33 bytes), el TTL (64), el protocolo de la carga útil (17=UDP), la suma de comprobación de la cabecera (`0x49e0`), la dirección origen (`192.168.0.37`) y, por último, la dirección destino (`93.184.215.14`).

5.5.3. Direcciones IP

A diferencia de la dirección MAC, la dirección IP no viene grabada en el dispositivo sino que es asignada cada vez que el computador se conecta a una red[11]. La dirección IP es un número de 32 bits y tiene una estructura jerárquica que la relaciona con esa red y se utiliza para identificar al computador en toda la interred, y no solo en ese enlace.

Como acabamos de ver, se representan como 4 cifras en base decimal separadas por puntos. Puedes averiguar la dirección IP de una interfaz de tu computador también con el comando `ip`.

```
$ ip addr show eth0
2: eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc fq_codel state UP
    link/ether f8:5f:2a:c0:ff:ee brd ff:ff:ff:ff:ff:ff
    inet 192.168.0.37/24 brd 192.168.0.255 scope global dynamic noprefixroute eth0
```

Como en Ethernet, aparte del direccionamiento *unicast*, hay otros modos:

broadcast Es posible direccionar, al menos en teoría, todos los hosts de una red.

[11] Por esto se las denomina «direcciones lógicas»

multicast Se puede direccionar un grupo arbitrario de hosts utilizando direcciones de grupo (las que empiezan por 240). Este mecanismo no es trivial ya que implica la suscripción activa de los hosts a grupos y requiere soporte en los *routers* que deben propagar la información de membresía. Para ello se utiliza un protocolo específico llamado IGMP [10] y protocolos de encaminamiento específicos.

En el capítulo 7 veremos todos los detalles del direccionamiento IP.

5.5.4. Conectividad IP

Una vez has verificado que el computador puede establecer un enlace de datos con sus vecinos puedes comprobar si es posible enviar (y recibir) tráfico IP hacia o desde otro PC, independientemente si es un vecino o está al otro lado del planeta. Las herramientas habituales para comprobar la conectividad IP se basan en el protocolo ICMP, así que las vamos a ver más adelante, en §5.7.1, cuando hayamos aprendido un poco sobre ese protocolo.

5.6. ARP

ARP (Address Resolution Protocol) es un protocolo auxiliar de IP necesario cuando un nodo conectado a un enlace de difusión (asumamos Ethernet) necesita comunicarse con sus vecinos. En este caso, el nodo emisor debe construir una trama, en la que debe colocar la dirección MAC del vecino destino. Sin embargo, el nodo emisor no conoce ese dato, pero lo que sí conoce es la dirección IP del destino.

ARP resuelve este problema. El nodo emisor envía una petición ARP a todos los vecinos (es un mensaje *broadcast*) preguntando si hay alguno que tenga asignada la IP destino. Si lo hay, el vecino que la tenga, construirá una respuesta ARP con su dirección MAC y la enviará específicamente al nodo solicitante (este mensaje es *unicast*). El vecino conoce la dirección MAC del solicitante porque aparece en la petición ARP.

El formato del mensaje ARP se muestra en la Figura 5.9. Es un mensaje de tamaño variable porque se puede utilizar con distintos protocolos de enlace y/o red, que pueden tener direcciones (físicas y lógicas) de tamaños distintos.

los campos de un mensaje ARP se listan a continuación. Para cada campo, se indica entre paréntesis el valor que tendría para el caso en que el protocolo de enlace sea Ethernet o WiFi y el de red sea IP, que es el caso más habitual.

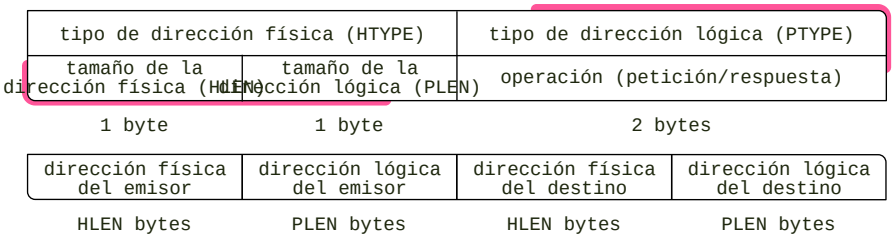

FIGURA 5.9: Formato del mensaje ARP

***hardware type* (HTYPE)**
: Es un código que indica el tipo de dirección física (Ethernet=1).

***protocol type* (PTYPE)**
: Tipo de dirección lógica (IP =0x0800).

***hardware length* (HLEN)**
: Número de bytes de la dirección física (en bytes) (Ethernet=6).

***protocol length* (PLEN)**
: Número de bytes de la dirección lógica (en bytes) (IP =4).

operation
: Operación a realizar: 1=petición, 2=respuesta.

sender hardware address
: Dirección física del emisor (6 bytes para Ethernet).

sender protocol address
: Dirección lógica del emisor (4 bytes para IP).

target hardware address
: Dirección física del destinatario.

target protocol address
: Dirección lógica del destinatario.

Veamos la captura de una petición y una respuesta ARP. Lo podemos conseguir con el siguiente comando y probablemente solo hay que esperar unos segundos para que aparezcan algunas tramas. Como en las capturas anteriores, se muestran solo las partes relevantes en este momento:

```
$ sudo tshark -f arp -nVx
Frame 1: 60 bytes on wire (480 bits) on interface eth0, id 0
    [Protocols in frame: eth:ethertype:arp]
Ethernet II, Src: fc:99:47:ad:f0:0d, Dst: ff:ff:ff:ff:ff:ff
    Type: ARP (0x0806)
    Padding: 00000000000000000000000000000000000000
Address Resolution Protocol (request)
    Hardware type: Ethernet (1)
    Protocol type: IPv4 (0x0800)
    Hardware size: 6
    Protocol size: 4
    Opcode: request (1)
    Sender MAC address: fc:99:47:ad:f0:0d
```

```
    Sender IP address: 192.168.8.193
    Target MAC address: 00:00:00:00:00:00
    Target IP address: 192.168.8.235

0000  ff ff ff ff ff ff fc 99 47 ad f0 0d 08 06 00 01   ...vd.....7....
0010  08 00 06 04 00 01 fc 99 47 ad f0 0d c0 a8 08 c1   ...........7....
0020  00 00 00 00 00 00 c0 a8 08 eb 00 00 00 00 00 00   ................
0030  00 00 00 00 00 00 00 00 00 00 00 00               ............

Frame 2: 42 bytes on wire (336 bits) on interface eth0, id 0
    [Protocols in frame: eth:ethertype:arp]
Ethernet II, Src: f8:5f:2a:c0:ff:ee, Dst: fc:99:47:ad:f0:0d
    Type: ARP (0x0806)
Address Resolution Protocol (reply)
    Hardware type: Ethernet (1)
    Protocol type: IPv4 (0x0800)
    Hardware size: 6
    Protocol size: 4
    Opcode: reply (2)
    Sender MAC address: f8:5f:2a:c0:ff:ee
    Sender IP address: 192.168.8.235
    Target MAC address: fc:99:47:ad:f0:0d
    Target IP address: 192.168.8.193

0000  fc 99 47 ad f0 0d f8 5f 2a c0 ff ee 08 06 00 01   .....7...vd.....
0010  08 00 06 04 00 02 f8 5f 2a c0 ff ee c0 a8 08 eb   .........vd.....
0020  fc 99 47 ad f0 0d c0 a8 08 c1                     .....7....
```

Lo que vemos aquí es una petición ARP (frame 1) enviada por el router local (`192.168.0.1`) y la respuesta ARP correspondiente (frame 2), que la envía el mismo computador que está haciendo la captura (`192.168.0.37`). Claramente el formato de ambos mensajes es el mismo, solo cambia el valor del campo *Opcode* que indica precisamente si es una petición o una respuesta. Observa que la petición va sobre una trama broadcast (dirección destino `ff:ff:ff:ff:ff:ff`), mientras que la respuesta va en una trama unicast (dirigida a la dirección MAC del emisor de la petición). Quizá lo más relevante es que el campo *Target MAC address* de la petición tiene un valor cero, porque ese es precisamente el dato que desconoce y pretende averiguar. Observa también que mientras la petición incluye un relleno (*padding*) para lograr que la trama Ethernet llegue al mínimo de 64 bytes, en la respuesta no aparece. Esto es porque esta respuesta ha sido capturada antes de salir del computador. Cuando la trama se envíe, la NIC añadirá el relleno.

ARP es lo que se llama un protocolo de descubrimiento de vecinos (*neighbor discovery*) o de resolución de direcciones, ya que en la práctica permite averiguar la dirección MAC de un nodo a partir de su dirección IP. De hecho, los programas de captura de tráfico suelen resumir un mensaje de petición ARP como «Who has `192.168.0.37`? Tell `192.168.0.1`» y la respuesta como «`192.168.8.37` is `f8:5f:2a:c0:ff:ee`».

Cabe señalar que los nodos no preguntan por las MAC de los vecinos cada vez que tienen que enviar una trama. El nodo dispone de una caché dónde se guardan las últimas respuestas, y así puede reutilizarlas evitando tráfico innecesario.

Puedes ver la caché ARP de tu computador con el comando `arp`.

```
$ sudo arp -n
Address           HWtype  HWaddress           Flags Mask   Iface
192.168.0.200     ether   44:09:17:e0:b8:5c   C            eth0
192.168.0.1       ether   fc:99:47:ad:f0:0d   C            eth0
192.168.0.198     ether   7c:83:cb:a4:34:b7   C            eth0
192.168.0.184     ether   00:21:66:58:5c:c8   C            eth0
```

Ahora que conoces ARP puedes utilizarlo como forma de comprobar la conectividad Ethernet. Para eso puedes generar una consulta ARP con el programa `arping`. Si el vecino responde, significa que está conectado y operativo. Eso es lo que hace el siguiente comando:

```
~$ sudo arping -c2 192.168.0.1
ARPING 192.168.0.1
60 bytes from 94:83:c4:02:ec:09 (192.168.0.1): index=0 time=2.279 msec
60 bytes from 94:83:c4:02:ec:09 (192.168.0.1): index=1 time=10.207 msec

--- 192.168.0.1 statistics ---
2 packets transmitted, 2 packets received,   0% unanswered (0 extra)
rtt min/avg/max/std-dev = 2.279/6.243/10.207/3.964 ms
```

Por supuesto, esto también ocurre cuando se envía un ping ICMP (lo hace el SO), pero con `arping` ni IP ni ICMP entran en juego, de modo que se puede aislar cualquier problema adicional no relacionado con Ethernet.

5.7. ICMP

ICMP [11] también es un protocolo auxiliar de IP, que sirve para:

- Enviar notificaciones sobre problemas o errores en la entrega de paquetes.
- Realizar diagnósticos e informar sobre el estado de la red.

Los mensajes ICMP se encapsulan sobre datagramas IP, pero a pesar de ello lo consideramos un protocolo de la capa de interred[12].

El formato del mensaje ICMP es sencillo (Figura 5.10). Sin embargo, dependiendo del tipo de mensaje, puede incluir campos con significados específicos y distinta carga útil.

[12] Para muchos autores (y también para nosotros) la capa en la que se ubica un protocolo tiene que ver más con su función que con el protocolo sobre el que se encapsula.

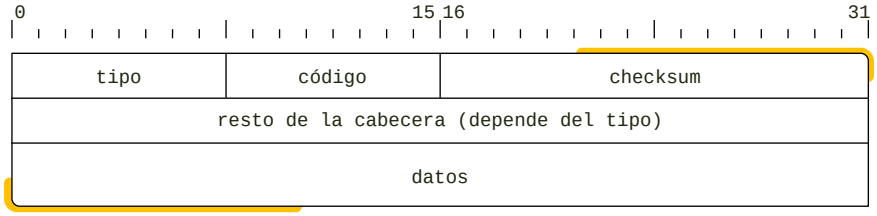

FIGURA 5.10: Formato del mensaje ICMP

Los mensajes ICMP de error se envían siempre al nodo que creó el paquete que provocó el problema. Por ejemplo, si un *router* no sabe cómo llevar un paquete a su destino final, creará un mensaje ICMP de tipo 3 (destino inalcanzable) y lo enviará al nodo que lo envió, es decir, colocará el mensaje ICMP en un paquete IP cuya dirección destino es la dirección origen del paquete problemático, y colocará su propia dirección IP (la del *router*) como dirección origen. Eso significa que el nodo sabe quién le está informando del problema. En este caso además, el mensaje ICMP incluye una copia de la cabecera del paquete IP problemático más 8 bytes de su carga útil. Así el SO puede determinar cuál de sus procesos fue el responsable del paquete problemático y, a su vez, ese proceso podría informar al usuario mediante una excepción u otro mecanismo de reporte de error disponible.

Para ilustrar el formato de la cabecera ICMP veamos la captura de una petición *Echo Request* —más conocido como `ping`— y su correspondiente respuesta. Este tipo de mensaje se puede enviar fácilmente con el programa del mismo nombre (`ping`). Aunque sencilla, resulta ser una herramienta muy útil y común para comprobar la conectividad IP, como veremos enseguida.

Por un lado, vamos a poner a `tshark` a capturar mensajes ICMP:

```
$ sudo tshark -f icmp -nV
```

Y por otro, vamos a enviar un único mensaje `ping` al nodo `example.net`:

```
$ ping -c 1 example.net
PING example.net (93.184.215.14) 56(84) bytes of data.
64 bytes from 93.184.215.14: icmp_seq=1 ttl=52 time=159 ms

--- example.net ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 159.448/159.448/159.448/0.000 ms
```

En la captura de `tshark` vemos algo parecido a esto:

```
Frame 1: 98 bytes on wire (784 bits), on interface wlp1s0, id 0
  [Protocols in frame: eth:ethertype:ip:icmp:data]
Ethernet II, Src: 0c:96:e6:76:64:e1, Dst: 0a:4a:d7:39:89:ba
  Type: IPv4 (0x0800)
Internet Protocol Version 4, Src: 192.168.10.67, Dst: 93.184.215.14
  Total Length: 84
  Protocol: ICMP (1)
Internet Control Message Protocol
  Type: 8 (Echo (ping) request)
  Code: 0
  Checksum: 0x8d86 [correct]
  Identifier (BE): 65286 (0xff06)
  Identifier (LE): 1791 (0x06ff)
  Sequence Number (BE): 1 (0x0001)
  Sequence Number (LE): 256 (0x0100)
  Timestamp from icmp data: May  6, 2024 21:54:10.393995000 CEST
  Data (40 bytes)
    101112131415161718191a1b1c1d1e1f202122232425262728292a2b2c2d2e2f3031323334353637

Frame 2: 98 bytes on wire (784 bits), on interface wlp1s0, id 0
  [Protocols in frame: eth:ethertype:ip:icmp:data]
Ethernet II, Src: 0a:4a:d7:39:89:ba, Dst: 0c:96:e6:76:64:e1
  Type: IPv4 (0x0800)
Internet Protocol Version 4, Src: 93.184.215.14, Dst: 192.168.10.67
  Total Length: 84
  Protocol: ICMP (1)
Internet Control Message Protocol
  Type: 0 (Echo (ping) reply)
  Code: 0
  Checksum: 0x9586 [correct]
  Identifier (BE): 65286 (0xff06)
  Identifier (LE): 1791 (0x06ff)
  Sequence Number (BE): 1 (0x0001)
  Sequence Number (LE): 256 (0x0100)
  Timestamp from icmp data: May  6, 2024 21:54:10.393995000 CEST
    101112131415161718191a1b1c1d1e1f202122232425262728292a2b2c2d2e2f3031323334353637
```

Puedes comprobar fácilmente que los valores de los campos de la cabecera ICMP corresponden con lo que hemos comentado. Puedes ver los tipos de mensaje ICMP en detalle en la sección 8.6.

5.7.1. Conectividad IP (ahora sí)

Como acabamos de ver, `ping` permite verificar que el nodo consultado está activo, accesible, y es capaz de recibir y enviar paquetes IP hacia y desde el nodo desde el que hemos lanzado el comando, es decir, verifica conectividad simétrica. En el caso del ejemplo anterior, está verificando que existe conectividad entre el nodo local y nodo `example.net`, pero eso no significa necesariamente que tenga la misma conectividad con otros nodos de la misma red.

Cabe destacar que `ping` no sirve para demostrar lo contrario, es decir, un nodo perfectamente funcional y con plena conectividad IP podría no respon-

der a *ping* por distintos motivos. Es posible configurar un cortafuegos para ignorar mensajes ICMP o específicamente mensajes *Echo Request*. Tampoco se puede usar `ping` para demostrar conectividad asimétrica, es decir, que el tráfico IP es posible en un sentido, pero no en el contrario.

En cualquier caso, se puede utilizar ICMP para comprobar la conectividad IP de otras formas y con otras herramientas. A continuación vamos a ver algunas de estas herramientas, incluyendo `ping` con más detalle, y algunas de las funciones que pueden ofrecer.

5.7.1.1. ping

Aparte de la verificación de conectividad IP, que obviamente es la funcionalidad principal, una consecuencia interesante del funcionamiento del programa `ping` es que puede calcular el RTT (Round-Trip Time) o «tiempo de vuelo», es decir, el tiempo transcurrido desde que se envió la petición hasta que se recibe la respuesta.

Prueba a ejecutar `ping` hacia tu *router* local y estudiemos con más detalle la información que ofrece.

```
1  $ ping 192.168.0.1
2  PING 192.168.0.1 (192.168.0.1) 56(84) bytes of data.
3  64 bytes from 192.168.0.1: icmp_req=1 ttl=255 time=2.02 ms
4  64 bytes from 192.168.0.1: icmp_req=2 ttl=255 time=0.730 ms
5  64 bytes from 192.168.0.1: icmp_req=3 ttl=255 time=1.66 ms
6  ^C
7  --- 192.168.0.1 ping statistics ---
8  3 packets transmitted, 3 received, 0% packet loss, time 2000ms
9  rtt min/avg/max/mdev = 0.730/1.471/2.023/0.546 ms
```

Ejecutado de esta forma (sin argumentos), el programa enviará mensajes *ping* de forma indefinida (uno por segundo) hasta que pulses `Control-C` (lo que ha pasado en la **línea 6**).

Las líneas 3 a 5 aparecen al recibirse cada respuesta. Para cada una se muestra el número de secuencia (icmp_req), el TTL (Time To Live) del paquete IP y el RTT (en milisegundos). Como el nodo destino (el *router* local en este caso) es un vecino, ese tiempo ronda los 2 ms.

Cuando el programa termina, muestra unas estadísticas sobre lo ocurrido (**línea 8**). Aparece el número de peticiones enviadas, respuestas recibidas, porcentaje de peticiones perdidas (sin respuesta) y el tiempo total que ha requerido. En la última línea aparecen los RTT mínimo, medio, máximo y la desviación estándar, que son datos interesantes para tener una idea de la latencia entre estos dos nodos y también de cómo varía con el tiempo. Por cierto, esta variación se conoce como *jitter* y es importante para cierto

tipo de aplicaciones como la telefonía IP, el streaming multimedia o los videojuegos en línea.

El programa `ping` es una de las formas más fáciles y habituales para establecer un proceso de monitorización automática para una serie de nodos que formen parte de la infraestructura de red de una organización, y de hecho prácticamente todos los sistemas de monitorización lo proporcionan de un modo u otro.

Como todos los comandos, `ping` también tiene opciones. Aquí puedes ver algunas que resultan interesantes para modificar su comportamiento:

-c Envía la cantidad de peticiones indicada y termina.

-i Permite fijar el intervalo entre envíos. Se especifica en segundos y admite decimales.

-A Modo adaptativo, calcula el intervalo en función del RTT medido.

-f Envía la siguiente petición tan pronto como vuelve la respuesta anterior.

5.7.1.2. ping «extendido»

Algunas implementaciones de `ping` proporcionan opciones para características avanzadas que, en realidad, se corresponden con campos y parámetros de los mensajes: TTL, tamaño de la carga del mensaje ICMP, etc. Veamos algunos:

-p Permite indicar el contenido de los bytes de relleno de la carga del paquete ICMP.

-R Graba y muestra la ruta seguida por la petición y su respuesta.

-s Indica el tamaño (en bytes) de la carga útil del mensaje de petición. El tamaño por defecto es 56 bytes.

-t Fija el valor del TTL de la cabecera IP en los mensajes de petición.

5.8. UDP

UDP es un protocolo de transporte. Eso significa que se encarga de mover mensajes entre procesos, que lógicamente pueden estar ejecutándose en nodos diferentes. Los mensajes UDP se llaman «datagramas» y se encapsulan sobre paquetes IP. Tal como hemos visto en la sección anterior, para determinar a qué nodo se debe enviar el datagrama se utiliza la dirección IP destino; para determinar el proceso dentro de ese nodo se utiliza el puerto

destino. A esto se le llama «multiplexación» y es la que permite que distintos procesos, incluso de la misma pareja de nodos, puedan mantener flujos de tráfico simultáneos e independientes.

El puerto es un simple entero de 16 bits sin estructura. Su función es permitir la comunicación entre procesos: el proceso servidor solicita al SO vincularse a un número específico para permanecer en estado de *escucha* pasiva, es decir, ese proceso queda inactivo en espera de una petición proveniente de un cliente. Cada puerto solo puede estar vinculado a un proceso, aunque un proceso puede vincularse a múltiples puertos.

Para organizar los distintos servicios los puertos están organizados en tres rangos:

Bien conocidos Desde el 1 hasta el 1 023 están reservados para servicios comunes y muy habituales. Se requieren permisos específicos para arrancar un servidor vinculado a un puerto de este rango.

Registrados Desde el 1 024 hasta el 49 151 están reservados para servicios registrados.

Dinámicos y efímeros Desde el 49 152 hasta el 65 535 están reservados para servicios dinámicos y puertos efímeros, que son los que el SO asigna de forma automática cuando el programador no especifica uno concreto.

UDP es un protocolo muy simple. De hecho, prácticamente la única funcionalidad que ofrece sobre IP es la citada multiplexación. No tiene corrección de errores, ni control de flujo, congestión, o duplicados... Simplemente, se coloca una secuencia de bytes como carga útil del datagrama y se envía al destino como una unidad individual e independiente, y sin ninguna garantía de que vaya a llegar. Tampoco proporciona conexión o desconexión, y por eso se dice que es un «protocolo sin conexión» (*connectionless*).

A pesar de que a primera vista parezca un protocolo de poca utilidad o interés, el hecho es que UDP es un protocolo ligero y eficiente, con mínima sobrecarga. Eso lo hace adecuado para situaciones en las que no es crítico que todos los mensajes lleguen a su destino, o cuando los recursos son limitados o la sobrecarga de un protocolo más complejo es inaceptable; para transmisiones en tiempo real, transmisión de audio o vídeo en tiempo real, actualización de estado en videojuegos, monitorización o telemetría.

5.8.1. Datagrama UDP

La Figura 5.11 muestra el formato del datagrama UDP. Por supuesto la cabecera UDP también es muy simple:

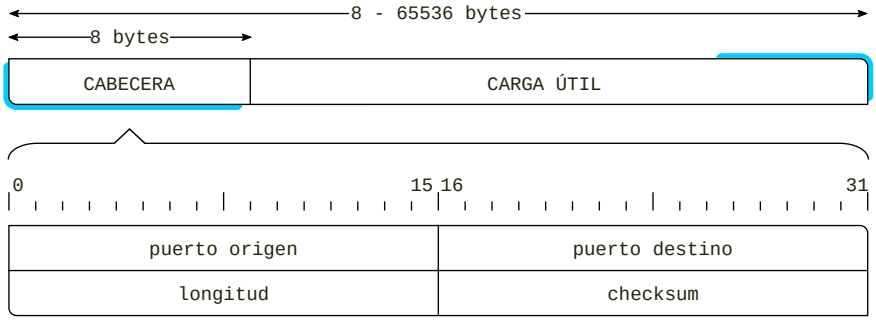

FIGURA 5.11: Formato del datagrama UDP

Puerto origen
: Puerto asociado al proceso que envía el datagrama. Como no tiene por qué haber respuesta, este campo es opcional (en cuyo caso contendrá un 0).

Puerto destino
: Puerto asociado al proceso al que va dirigido el datagrama.

Longitud
: Longitud total del datagrama UDP incluyendo cabecera y carga útil.

Checksum
: Suma de comprobación, que sirve para verificar si el datagrama ha sufrido corrupción en los datos durante la transmisión. Esto ofrece una detección básica de errores, aunque también es opcional.

Veamos una cabecera UDP que corresponde a la captura 5.1:

```
User Datagram Protocol, Src Port: 33262, Dst Port: 2000
  Source Port: 33262
  Destination Port: 2000
  Length: 13
  Checksum: 0x995d [unverified]
  UDP payload (5 bytes)

0000  68 6f 6c 61 0a                                    hola.
  Data: 686f6c610a
```

Y puedes apreciar claramente los cinco campos de los que hemos hablado. En este caso, el datagrama UDP indica que el puerto destino es 2000, el origen el 33 262, tiene una longitud de 13 bytes (incluyendo la carga útil), y la suma de comprobación es `0x995d`. La carga útil es la palabra «hola» seguida de un salto de línea, que codificado en ASCII y hexadecimal son los 5 bytes `686f6c610a`, y más los 8 bytes de la cabecera, hacen el total de 13 bytes.

5.8.2. Conectividad UDP

La conectividad UDP implica la posibilidad de que un cliente sea capaz de enviar un datagrama UDP y el proceso servidor al que va dirigido lo pueda recibir adecuadamente.

El mensaje que acabamos de analizar corresponde con los comandos de la sección 5.1, pero no sirve para verificar conectividad UDP, ya que solo es un mensaje enviado por un cliente y capturado desde el mismo computador, ni siquiera hay servidor.

Para comprobar la conectividad vamos a usar `ncat` creando un servidor UDP en un nodo y también con `ncat` un cliente UDP en el otro nodo. El servidor se crea con:

```
alice@node1$ ncat -u -l 2000
```

Y en otra consola ejecutamos el cliente con:

```
bob@node2$ ncat -u node1 2000
```

Ahora escribe algo de texto en la consola del cliente, pulsa `ENTER` y debería aparecer en la del servidor. Si quieres comprobar la conectividad en el sentido opuesto escribe algo de texto en la del servidor, pulsa `ENTER` y debería aparecer en la del cliente.

Es importante destacar que, como en el caso de IP, es perfectamente posible tener conectividad en uno de los sentidos, pero no en el otro. Un problema de encaminamiento o un cortafuegos en uno de los dos nodos o en cualquier de los routers del camino podría descartar los datagramas UDP solo en uno de los sentidos.

5.9. TCP

TCP es «el otro» protocolo de transporte de TCP/IP. Podríamos decir que TCP es lo contrario a UDP en casi todo. Para empezar, TCP está orientado a conexión, lo que implica que antes de enviar o recibir datos, cliente y servidor obligatoriamente tienen que intercambiar información de sincronización —durante el proceso de conexión— y, de forma similar, también se aplica un procedimiento de desconexión que asegura que ambos extremos saben que el otro da por terminada la comunicación.

La conexión asegura que solo cuando ambas partes tienen la información necesaria sobre el otro puede comenzar la transmisión de datos. A diferencia de UDP, TCP garantiza la integridad de los datos, es decir, que se no se producen cambios, no hay partes ausentes ni duplicadas. Para lograr esto,

TCP utiliza un mecanismo de confirmación, temporizadores y retransmisión de los segmentos. «Segmento» es el nombre que se da a los mensajes que se intercambian en TCP. Además, incluye mecanismos de control de flujo para evitar saturar al receptor, y de congestión para evitar colapsar la red.

La comunicación TCP se comporta como un flujo continuo y bidireccional de bytes. Ambos procesos pueden enviar datos adicionales en cualquier momento y de forma indefinida. Sin embargo, los procesos no controlan cuándo se envían realmente los datos a través de la red. Es el SO el que se encarga de agrupar esos bytes en segmentos y enviarlos en el momento más adecuado para maximizar la eficiencia al mismo tiempo se aplican los mecanismos de control de flujo y congestión.

5.9.1. Segmento TCP

TCP es un protocolo bastante complejo, de modo que comprender el funcionamiento de estos mecanismos, incluso de una forma somera, requiere abordarlos cada uno por separado, y eso lo haremos en sucesivos capítulos. Por supuesto esta complejidad se ve reflejada en el formato de su cabecera, que se muestra en la Figura 5.12.

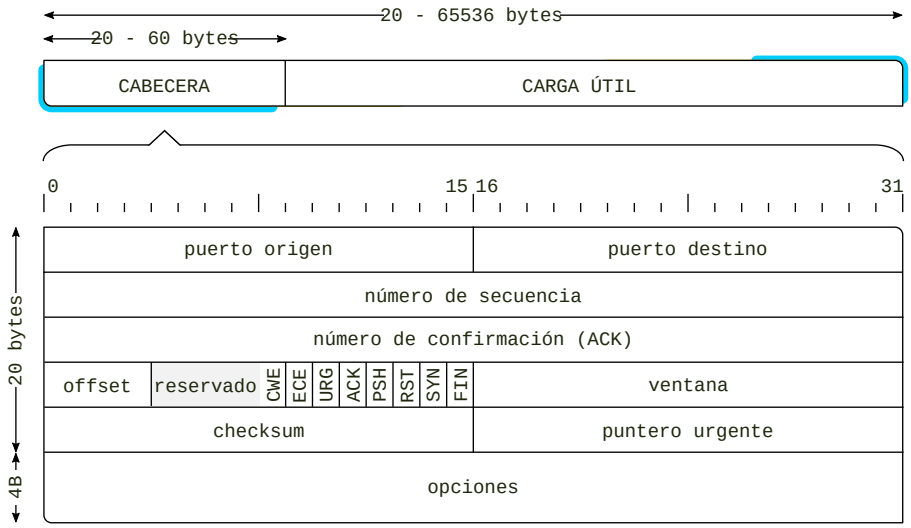

FIGURA 5.12: Formato del segmento TCP

Aunque veremos la función que desempeña cada unos de estos campos a lo largo del libro, a continuación se describen brevemente a modo se referencia. En este contexto, cuando hablemos de «emisor» nos estaremos refiriendo

al nodo que ha enviado específicamente **este** segmento, y cuando hablemos de «destinatario» para referirnos al nodo al que va dirigido.

`puerto origen`
: Puerto asociado al proceso emisor.

`puerto destino`
: Puerto asociado al proceso destinatario.

`número de secuencia`
: Número de secuencia del primer byte de la carga útil del segmento.

`número de confirmación`
: Número de secuencia del siguiente byte que el emisor espera recibir. Dicho de otro modo, es un acuse de recibo de todos los datos hasta el inmediatamente anterior al indicado.

`offset`
: Tamaño de la cabecera expresado en palabras de 32 bits. Indica el lugar dónde comienza la carga útil, por lo que también se llama *data offset*.

`flags`
: Los flags son un conjunto de bits que ofrecen información del segmento o sobre su emisor. Los más importantes son:

Flag	Descripción
CWR	(Congestion Window Reduced) El emisor ha recibido un segmento con el bit ECE activado y ha reducido su tasa de emisión.
ECE	(ECN Echo) El emisor sufre congestión.
URG	El campo *puntero urgente* es significativo.
ACK	El campo *acuse de recibo* es significativo.
PSH	El destino debe pasar los datos al proceso tan pronto como pueda.
RST	El emisor rechaza (o cierra) la conexión.
SYN	El emisor quiere establecer una conexión.
FIN	El emisor ha terminado de enviar datos.

CUADRO 5.1: Descripción de los flags TCP

`ventana`
: Tamaño de la ventana del receptor. El receptor indica al emisor de cuánto espacio libre (en bytes) dispone para aceptar datos nuevos.

`checksum`
: Suma de comprobación.

`puntero urgente`
: Indica la posición del último byte urgente.

`opciones`
: Cabeceras adicionales para propósitos específicos, que tienen su propio formato.

5.9.2. Capturando una conexión TCP

Como en las secciones anteriores, podemos capturar mensajes TCP con `tshark`. Vamos a replicar con TCP una situación equivalente a lo que hemos visto con UDP. En una consola ejecuta:

```
~$ sudo tshark -i lo -f "tcp port 2000"
```

En otra consola ejecuta un servidor TCP:

```
~$ ncat -l -p 2000
```

Y por último, en una tercera consola, ejecuta un cliente TCP, escribe algo y pulsa `Control-D` (que es el carácter de fin de archivo) que cerrará la conexión.

```
~$ ncat 127.0.0.1 2000
hola
C-d
```

Analicemos brevemente lo que aparece en la captura de `tshark`:

```
1  127.0.0.1 → 127.0.0.1  TCP 74 47178 → 2000 [SYN] Seq=0 Win=65495 Len=0 MSS=65495
2  127.0.0.1 → 127.0.0.1  TCP 74 2000 → 47178 [SYN, ACK] Seq=0 Ack=1 Win=65483 Len=0
↪  [...]
3  127.0.0.1 → 127.0.0.1  TCP 66 47178 → 2000 [ACK] Seq=1 Ack=1 Win=65536 Len=0
4  127.0.0.1 → 127.0.0.1  TCP 71 47178 → 2000 [PSH, ACK] Seq=1 Ack=1 Win=65536 Len=5
5  127.0.0.1 → 127.0.0.1  TCP 66 2000 → 47178 [ACK] Seq=1 Ack=6 Win=65536 Len=0
6  127.0.0.1 → 127.0.0.1  TCP 66 47178 → 2000 [FIN, ACK] Seq=6 Ack=1 Win=65536 Len=0
7  127.0.0.1 → 127.0.0.1  TCP 66 2000 → 47178 [FIN, ACK] Seq=1 Ack=7 Win=65536 Len=0
8  127.0.0.1 → 127.0.0.1  TCP 66 47178 → 2000 [ACK] Seq=7 Ack=2 Win=65536 Len=0
```

Esta vez hemos hecho una captura más sencilla. Al no añadir el argumento `-V` a `tshark`, solo muestra un resumen de algunos campos importantes de las cabeceras.

En cada línea se muestran varios datos:

- Las direcciones IP de origen y destino. En este caso es `127.0.0.1` para ambos ya que los estás ejecutando en dos consolas de tu computador y además el cliente ha conectado mediante la interfaz *loopback*.
- Los puertos origen (47178, asignado automáticamente para el cliente) y 2000 (que es dónde has puesto el servidor).
- El protocolo del mensaje más específico. En este ejemplo, el payload del segmento TCP no encapsula un mensaje de un formato que `tshark` pueda reconocer, de modo que indica TCP.
- El tamaño del segmento completo. 74 bytes para el primer segmento.
- Los flags activos. Por ejemplo, en el primero segmento está activo el flag SYN.

- El número de secuencia (Seq) y de confirmación (Ack).
- El tamaño de la ventana de recepción (Win).
- La longitud de la carga útil (Len). Solo el mensaje 4 porta datos (la cadena «hola\n») que corresponde a esos 5 bytes que indica.

Lo primero que llama la atención es que escribir un solo mensaje (con el texto «hola») ha provocado 8 segmentos. Esto es porque, como se ha dicho, TCP realiza un proceso de conexión previo y un proceso de desconexión al final.

- Los mensajes 1 a 3 corresponden al proceso de conexión.
- Los mensajes 4 y 5 corresponden al envío de la carga útil y su correspondiente segmento de acuse de recibo.
- Los mensajes 6 y 7 corresponden a la desconexión.

Para poder entender plenamente toda esta información y también otros campos de las cabeceras que ni se están mostrando aquí, tendrás que esperar a próximos capítulos donde veremos el funcionamiento detallado de TCP. Pero de momento, con lo que hemos visto aquí, ya tienes una idea de la considerable diferencia de complejidad y funcionalidad que ofrece respecto a UDP, incluso para un ejemplo tan simple como ese.

5.9.3. Conectividad TCP

El ejemplo que has realizado con `ncat` en la sección anterior para la captura es también una comprobación de conectividad TCP perfectamente válida, siempre que, lógicamente, ejecutes cliente y servidor en nodos diferentes.

Fíjate que comprobar la conectividad TCP implica obligatoriamente comprobar la simetría, ya que es imposible enviar datos si primero no se estable la conexión, y la conexión requiere segmentos en ambos sentidos. Tener conectividad en un sentido no implica necesariamente que exista en el otro. Un cortafuegos podría filtrar específicamente el segmento de conexión (SYN) de A hacía B impidiendo la conexión, pero no a la inversa. Por eso, si necesitas verificar la posibilidad de establecer la conexión en ambos sentidos, tendrás que repetir la prueba cambiando los roles como cliente y servidor.

Y ¿qué más?

Esto ha sido un vistazo rápido al formato y utilidad de los protocolos principales. Es probable que como lector tengas ahora más preguntas y dudas que antes empezar el capítulo. No te preocupes, es normal. La intención era abrir camino y mostrar la parte más tangible de los protocolos: sus mensajes. Por eso hemos hecho énfasis en las capturas de tráfico real y por también por eso es muy importante que las realices por tu cuenta.

En próximos capítulos veremos con más detalle el funcionamiento de cada protocolo y cómo cada campo de estas cabeceras tiene consecuencias clave en el comportamiento de la red. Nos quedan algunos protocolos importantes como DNS, DHCP, HTTP ... pero esos todavía tendrán que esperar hasta que tengas una bases sólidas sobre direccionamiento, encaminamiento y programación de redes.