Departament d'Enginyeria Telemàtica

Universitat Politècnica de Catalunya (UPC)

Manual de Pràctiques de Laboratori

Introducció a les Xarxes de Telecomunicació

Xarxes IP

Autors (per ordre alfabètic):

Mónica Aguilar Igartua

Juanjo Alins Delgado

Óscar Esparza Martín

Jose L. Muñoz Tapia

Departament d'Enginyeria Telemàtica. Universitat Politècnica de Catalunya

Barcelona, Juny 2010

Índice de la práctica

PRÁCTICA

Internet Protocol (IP)

1. Introducción

Los protocolos de red se diseñan en forma de capas o niveles, donde cada nivel es responsable de una faceta diferente de las comunicaciones. Una torre de protocolos es la combinación de diversos protocolos de varios niveles. TCP/IP se considera un sistema de 4 niveles (o 4 capas) tal y como se muestra en la figura 1.

TCP e IP son los dos protocolos que dan nombre a una arquitectura de red. Sin embargo, una red TCP/IP necesita otros protocolos para tener todas las funcionalidades. La figura 2 muestra los protocolos que forman la familia TCP/IP.

El objetivo del protocolo IP es convertir redes físicamente diferentes (como pueden ser *Ethernet*, *Token Ring*, X.25, *Frame Relay*, ATM...) en una red aparentemente homogénea, lo que se conoce como interconexión de redes. De esta forma IP oculta las redes físicas subyacentes creando por encima de éstas una única red "virtual" o red de redes, donde:

- Hay un esquema de identificación (o direccionamiento) de todos los sistemas, uniforme y universal. Este esquema de direccionamiento ha de ser independiente del *hardware*. Esto se consigue asignando a cada nodo un número único de 32 bits (en IPv4, *Internet Protocol* versión 4) llamado dirección IP.
- Los datos viajan por la red virtual IP agrupados en paquetes denominados datagramas. La manipulación de estos datagramas por parte de la red sigue un método uniforme llamado "encaminamiento de datagramas", independiente de la red en particular donde se encuentren estos datagramas.
- La entrega de datagramas de un extremo a otro por la red IP es:
  - No fiable (no hay control de errores, ni control de flujo).
  - No orientado a conexión.
  - La red se esfuerza al máximo para entregar los paquetes, pero no asegura la entrega (*best effort*).

Por tanto los datagramas se pueden perder, desordenar o incluso duplicar, e IP no tendrá en cuenta estas situaciones (lo controlarán protocolos superiores). El protocolo IP asume pocas cosas de las capas inferiores, sólo que los datagramas "probablemente" serán transportados al *host* de destino.

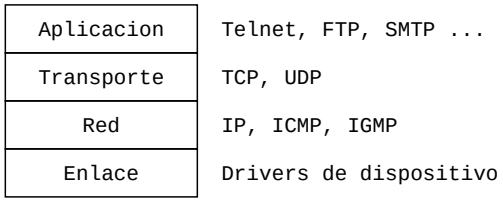


Figura 1: Las 4 capas de TCP/IP

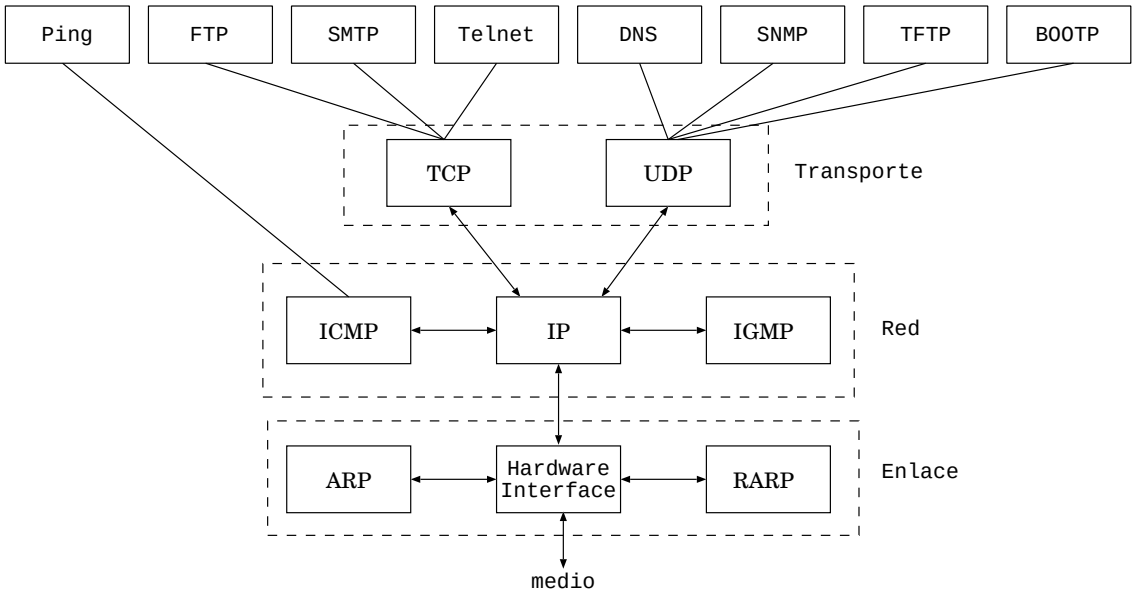


Figura 2: Protocolos de la familia TCP/IP

2. Las direcciones de Internet

- Las direcciones simbólicas son las que usualmente utilizamos los humanos porque son más fáciles de recordar que las numéricas. Estas direcciones son cadenas de caracteres separadas por puntos. Por ejemplo: **www.upc.es**.
- Las direcciones numéricas son las que usualmente utiliza el *software* IP. La forma numérica es una secuencia de 32 bits. Para que sea más inteligible, habitualmente se expresa en forma de números decimales separados por puntos (*dotted-decimal* en inglés), donde cada decimal representa 8 bits (rango 0-255). Por ejemplo, la dirección binaria **10010011. 01010011. 00010100. 00000010** se puede expresar en notación decimal como **147.83.20.2** (que a su vez se corresponde con la dirección simbólica **www.upc.es**).

Las funciones de mapeo entre las direcciones simbólicas y las direcciones numéricas las realiza el DNS (*Domain Name System*).

Una dirección no identifica una máquina en Internet, sino a un determinado interfaz de red de la máquina. Dicho de otra forma, la dirección IP identifica a una determinada máquina en una determinada red física. Cuando la máquina está conectada a más de una red se la denomina "*multi-homed*" y tiene una dirección por cada interfaz de red.

En realidad, para interpretar correctamente una determinada dirección IP se necesita una segunda secuencia de 32 bits denominada “máscara” (*netmask*). La máscara se utiliza para dividir la dirección en dos partes:

- La parte de red, denominada formalmente “número de red” (RFC [1] 1166), aunque en ocasiones se usan los términos dirección de red y netID. Los números de red están administrados centralmente por el INTERNIC (*INTErnet Network Information Center*) y son únicos en toda Internet. Cuando se solicita al INTERNIC una dirección IP, no se asigna una dirección a cada máquina individual, sino que se da un número de red, permitiendo asignar según las necesidades todas las direcciones IP válidas dentro de este rango.
- La parte de *host*, denominada formalmente “número de *host*”, aunque ocasionalmente se usan los términos dirección de *host* y hostID.

Para dividir una dirección IP en número de red y número de *host* se utiliza la máscara, de manera que aquellos bits de la dirección donde la máscara tiene un "1" forman parte del número de red, mientras que las posiciones donde la máscara tiene “0” forman parte del número de *host*. Por ejemplo si tenemos la dirección 10010011. 01010011. 00010100.

[1] Los RFC (Request For Comment) son los estándares y protocolos publicados por la IETF (Internet Engineering Task Force).

00000010 con una máscara 11111111. 11111111. 00000000. 00000000 tenemos un número de red 10010011. 01010011. y un número de *host* 00010100. 00000010.

Por convenio el número de red se encuentra en los bits más significativos (los bits de la izquierda de la dirección) mientras que el número de *host* se suele encontrar en los bits menos significativos (los bits de la derecha de la dirección). Además los números de red y *host* suelen estar formados por bits consecutivos de la dirección IP (es muy extraño ver máscaras con ceros y unos no consecutivos).

La máscara de red también se suele representar en formato decimal, de manera que para el ejemplo anterior tenemos la dirección 147.83.20.2 y la máscara 255.255.0.0. Otra manera de representar una dirección IP es poniendo la dirección en formato decimal y la máscara como un número decimal que indica cuántos unos consecutivos forman el número de red. Así para nuestro ejemplo, la dirección 147.83.20.2 con máscara 255.255.0.0 se puede escribir como 147.83.20.2/16.

El tamaño de la parte dedicada al *host* depende del tamaño de la red. Para satisfacer múltiples necesidades se han definido varias clases de redes, fijando diferentes puntos donde dividir la dirección IP. De esta manera se dispone las siguientes clases:

Clase A

La clase A comprende redes desde 1.0.0.0 hasta 127.0.0.0. El identificador de red está contenido en el primer octeto, quedando para el *host* los 24 bits restantes y permitiendo aproximadamente 1.6 millones de máquinas por red. Por lo tanto, la máscara de red resultante es 255.0.0.0 (11111111. 00000000. 00000000. 00000000 en formato binario). Dentro de esta clase, el rango 10.0.0.0 hasta 10.255.255.255 es un rango reservado para uso privado. Una dirección de *host* reservada posible sería 10.0.0.1/8.

Clase B

La clase B comprende las redes desde 128.0.0.0 hasta 191.255.0.0, donde el *netid* está en los dos primeros octetos. Esta clase permite 16320 redes con 65024 lugares cada una. La máscara de esta clase es 255.255.0.0 (11111111. 11111111. 00000000. 00000000 en formato binario). En esta clase el rango 172.16.0.0 hasta 172.31.0.0 se reserva para uso privado. Una dirección de *host* posible reservada sería 172.31.0.1/16.

Clase C

Las redes de clase C van desde 192.0.0.0 hasta 223.255.255.0, donde el *netid* está en los tres primeros octetos. Esta clase permite cerca de 2 millones de redes con 254 lugares. La máscara de esta clase es de la forma 255.255.255.0 (11111111. 11111111. 11111111. 00000000 en formato binario). En esta clase se reserva para uso privado el rango 192.168.0.0 hasta 192.168.255.0. Una posible dirección de *host* privada puede ser 192.168.128.1/24.

Clase D

Son direcciones dentro del rango 224.0.0.0 hasta 239.255.255.255 y se las llama direcciones *multicast* o de multidifusión.

Clase E

Las direcciones que están en el rango 240.0.0.0 hasta 247.255.255.255 son experimentales o están reservadas para futuras aplicaciones.

Direcciones Especiales

Cabe destacar que en la parte reservada a *host*, se reservan los siguientes valores:

- Una dirección IP donde todos los bits de la parte de *host* son cero se refiere a la dirección de la red misma.
- Una dirección donde todos los bits de la parte de *host* son uno se denomina dirección de difusión (o *broadcast*), ya que hace referencia a todas las máquinas de la red específica. Así 147.83.20.255 no es un *hostid* válido, pero se refiere a todos los *hostid* de la red 147.83.20.0.

Las direcciones de red 0.0.0.0 y 127.0.0.0 están reservadas. La primera dirección se llama encaminamiento por defecto (por donde IP encamina los datagramas), y la segunda es la dirección de *loopback*.

La red 127.0.0.0 está reservada para el tráfico local IP de la máquina. Normalmente la dirección 127.0.0.1 se asigna a una interfaz de la máquina (la interfaz de *loopback*) que actúa como un circuito cerrado. Cualquier paquete IP enviado a este interfaz será devuelto como si hubiera vuelto desde alguna red. Esto permite por ejemplo instalar *software* de red o probar tarjetas *ethernet*, aunque no se disponga de una red real.

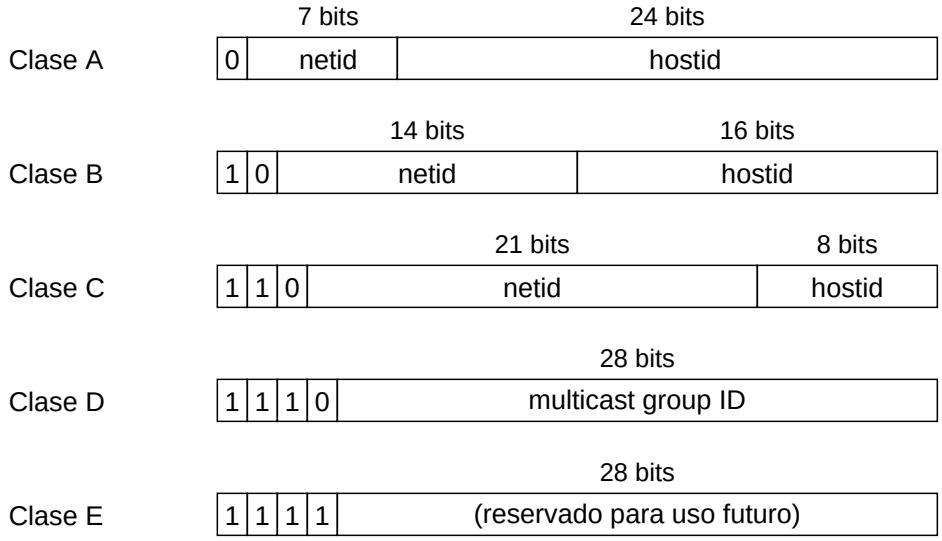


Figura 3: Clases de direcciones IP

3. El datagrama IP

El datagrama IP es la unidad de transferencia en las redes IP. Básicamente consiste en una cabecera IP y un campo de datos para protocolos superiores. El datagrama IP está encapsulado en la trama de nivel de enlace, que suele tener una longitud máxima (MTU, *Maximum Transfer Unit*), dependiendo del *hardware* de red usado. Para Ethernet, esta es típicamente de 1500 bytes. En vez de limitar el datagrama a un tamaño máximo, IP puede tratar la fragmentación y el reensamblado de sus datagramas. En particular, IP no impone un tamaño máximo, pero establece que todas las redes deberían ser capaces de manejar al menos 576 bytes. Los fragmentos de datagramas tienen todos una cabecera, copiada básicamente del datagrama original, y de los datos que la siguen. Los fragmentos se tratan como datagramas

normales mientras son transportados a su destino. Nótese, sin embargo, que si uno de los fragmentos se pierde, todo el datagrama se considerará perdido, y los restantes fragmentos también se considerarán perdidos.

3.1. Formato del datagrama IP

La cabecera del datagrama IP está formada por los campos que se muestran en la figura 4.

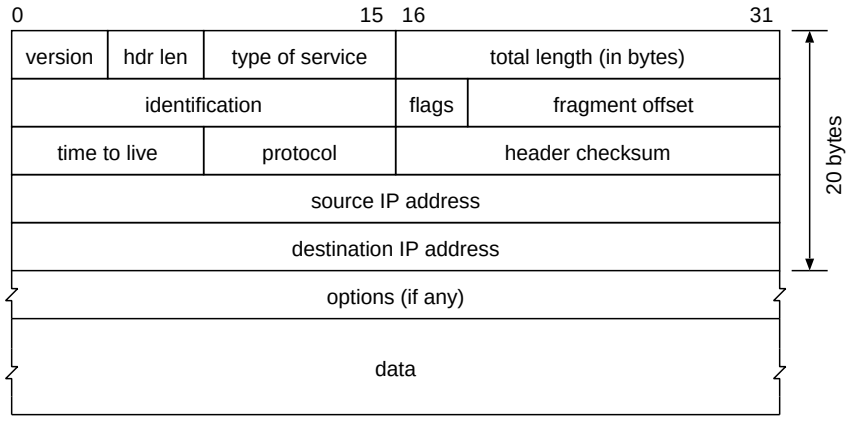


Figura 4: Formato del datagrama IP

Donde:

Version es la versión del protocolo IP. La versión actual es la 4. La 5 es experimental y la 6 es IPng.

Hdr Len es la longitud de la cabecera IP contada en cantidades de 32 bits. Esto no incluye el campo de datos.

Type Of Service es el tipo de servicio es una indicación de la calidad del servicio solicitado para este datagrama IP. Una descripción detallada de este campo se puede encontrar en el RFC 1349.

Total Length es la longitud total del datagrama, cabecera y datos, especificada en bytes.

Identification es un número único que asigna el emisor para ayudar a reensamblar un datagrama fragmentado. Los fragmentos de un datagrama tendrán el mismo número de identificación.

Flags son flags para el control de fragmentación.

Donde:

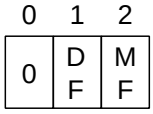


Figura 5: Detalle del campo *Flags*

El bit 0 está reservado y debe ser 0.
El bit **DF** significa no fragmentar (***D**o not **F**ragment*). Con 0 se permite fragmentación y con 1 no.
El bit **MF** significa que hay más fragmentos (***M**ore **F**ragments*). Con 0 significa que se trata del último fragmento del datagrama y con 1 que hay más fragmentos.

Fragment Offset (FO) se usa en datagramas fragmentados para ayudar al reensamblado de todo el datagrama. El valor es el número de partes de 64 bits (no se cuentan los bytes de la cabecera) contenidas en fragmentos anteriores. En el primer (o único) fragmento el valor es siempre cero.

Time To Live especifica el tiempo (en segundos) que se le permite viajar a este datagrama. Cada "*router*" por el que pase este datagrama ha de sustraer de este campo el tiempo tardado en procesarlo. En la realidad un "*router*" es capaz de procesar un datagrama en menos de 1 segundo; por ello restará uno de este campo y el TTL se convierte más en una cuenta de saltos que en una métrica del tiempo. Cuando el valor alcanza cero, se asume que este datagrama ha estado viajando en un bucle y se desecha. El valor inicial lo debería fijar el protocolo de alto nivel que crea el datagrama.

Protocol indica el número oficial del protocolo de alto nivel al que IP debería entregar los datos del datagrama. Algunos valores importantes se muestran en la Tabla 1.

Protocolo	Número
Reservado	0
ICMP	1
IGMP	2
IP encapsulado	4
TCP	6
UDP	17
OSPF	89

Tabla 1: Algunos *Protocol Numbers*

Header Checksum es el *checksum* de la cabecera. Se calcula como el complemento a uno de la suma de los complementos a uno de todas las palabras de 16 bits de la cabecera. Si el *checksum* de la cabecera no se corresponde con los contenidos, el datagrama se desecha, ya que al menos un bit de la cabecera está corrupto, y el datagrama podría haber llegado a un destino equivocado.

Source IP Address es la dirección IP de 32 bits del *host* emisor.

Destination IP Address es la dirección IP de 32 bits del *host* receptor.

Options es un campo de longitud variable. Las opciones se incluyen en principio para pruebas de red o depuración, por tanto no se requiere que toda implementación de IP sea capaz de generar opciones en los datagramas que crea, pero sí que sea capaz de procesar datagramas que contengan opciones.
El campo **Options** tiene longitud variable en función de la opción seleccionada. Algunas opciones tienen una longitud de un solo byte y otras tienen longitudes variables. El primer byte de cualquier opción se denomina *type byte* y su estructura se muestra en la figura 6.

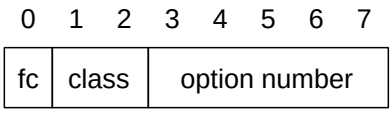


Figura 6: Detalle del *type byte*

Donde:

FC es el flag de copia (***F**lag **C**opy*), e indica si el campo de opciones se ha de copiar (1) o no (0) cuando el datagrama está fragmentado.

class es un entero sin signo de 2 bits, donde:

number es un entero sin signo de 5 bits que indica el número de opción dentro de cada clase.

Option class	significado
0	Control de red o datagrama
1	Reservado para uso futuro
2	Depuración y medición
3	Reservado para uso futuro

Tabla 2: Option classes

3.2. La fragmentación de datagramas IP

Cuando un datagrama IP viaja de un *host* a otro puede cruzar distintas redes físicas. Las redes físicas imponen un tamaño máximo de trama, llamado MTU (*Maximum Transmission Unit*), que limita la longitud de un datagrama.

Por ello, existe un mecanismo para fragmentar los datagramas IP grandes en otros más pequeños, y luego reensamblarlos en el *host* de destino. IP requiere que cada enlace tenga un MTU de al menos 68 bytes, de forma que si cualquier red proporciona un valor inferior, la fragmentación y el reensamblado tendrán que implementarse en la capa de la interfaz de red de forma transparente a IP. 68 es la suma de la mayor cabecera IP, de 60 bytes, y del tamaño mínimo posible de los datos en un fragmento (8 bytes).

Las implementaciones de IP no están obligadas a manejar datagramas mayores de 576 bytes, pero la mayoría podrá manipular valores más grandes, normalmente mayores de 1500 bytes.

Un datagrama sin fragmentar tiene a cero toda la información de fragmentación. Es decir, el flag **MF** está a 0.

Cuando se ha de realizar la fragmentación, se ejecutan los siguientes pasos:

- Se chequea el bit de flag **DF** para ver si se permite fragmentación. Si está a uno, el datagrama se desecha y se devuelve un error al emisor usando ICMP [2] (*Internet Control Message Protocol*).
- Basándose en el valor MTU, el campo de datos se divide en partes donde cada parte, excepto la última, debe ser múltiplo de 8 bytes.
- Todas las porciones de datos se colocan en datagramas IP.
- Se copian las cabeceras de la cabecera original, con algunas modificaciones:
  - El bit de flag **MF** se pone a uno en todos los fragmentos, excepto en el último.
  - El campo **FO** se pone al valor de la localización de la porción de datos correspondiente.
  - Si se incluyeron opciones en el datagrama original, el bit **FC** del type byte determina si se copiaran o no en todos los fragmentos o sólo en el primero[3].
  - Se inicializa el campo **Hdr Len** (longitud de la cabecera).
  - Se inicializa el campo **Total Length** (longitud total).
  - Se recalcula el **Header Checksum** de la cabecera.

Cada uno de estos datagramas se envía como un datagrama IP normal. IP maneja cada fragmento de forma independiente, es decir, los fragmentos pueden atravesar diversas rutas hacia su destino, y pueden estar sujetos a nuevas fragmentaciones si pasan por redes con MTUs inferiores.

En el *host* de destino, los datos se tienen que reensamblar. Para ello se siguen los siguientes pasos:

- Con el fin de reensamblar los fragmentos, el receptor destina un buffer de almacenamiento en cuanto llega el primer fragmento.
- Una vez ha llegado el primer fragmento se inicia un contador temporal.

[2] Véase el apartado 7, dedicado a este protocolo.

[3] Por ejemplo, las opciones de encaminamiento de la fuente se tendrán que copiar en todos los fragmentos y por tanto tendrán a uno este bit.

- Cada fragmento se identifica mediante el campo **Identification** que es un número único dentro de los límites impuestos por el uso de un número de 16 bits. Como la fragmentación no altera este campo, los fragmentos que van llegando al destino pueden ser identificados gracias a este identificador y a las direcciones IP fuente y destino del datagrama. Además el campo **Protocol** también se chequea.
- Los fragmentos que van llegando se copian en el *buffer* en la localización indicada por el campo **FO**.
- Cuando han llegado todos los fragmentos, se restaura el datagrama original y se continúa con su proceso.
- Si vence el contador temporal y no se han recibido todos los fragmentos, el datagrama en fase de reensamblado se desecha.

Para la plena comprensión de las opciones que se detallan a continuación es recomendable la lectura previa del apartado 6.

*Registro de ruta o RR (*Record Route*)*

Esta opción proporciona un medio para almacenar las direcciones IP de los *routers* por los que es encaminado el datagrama. El *host* fuente es el encargado de configurar la opción dejando espacio suficiente para que se puedan almacenar las direcciones IP de los *routers* por los que va pasando el datagrama. Si el espacio para almacenar la ruta se llena antes de que el datagrama llegue a su destino, el datagrama se retransmitirá, pero se dejará de grabar la ruta.

En la figura 7 se puede observar el formato de esta opción:

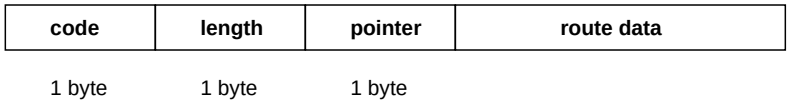


Figura 7: Formato de la opción RR

Donde:

Code contiene la clase y el número de opción para RR (**Code** tiene un valor decimal 7 para esta opción).

Lenght indica la longitud total en bytes de la opción.

Pointer se utiliza para apuntar a la siguiente ranura disponible para almacenar una dirección IP. Este campo está en bytes, por lo que cada *router* introduce su dirección IP en la ranura correspondiente e incrementa **Pointer** en 4. Si en un determinado *router* sucede que **Pointer >Lenght,** la ruta deja de grabarse desde ese *router*.

Route data se incluyen las direcciones registradas por los *routers.*

*Ruta fuente o SR (*Source Route*)*

Esta opción proporciona al emisor una forma de controlar de forma más fina el proceso de encaminamiento que usualmente se basa en una única dirección destino. Mediante esta opción el emisor puede proporcionar información sobre la ruta que deben seguir los datagramas[4].

IP proporciona dos formas de enrutado de fuente:

- Enrutado estricto de fuente o SSR (*Strict Source Routing*).
  La fuente proporciona la ruta "estricta" por la que los datagramas deben encaminarse hasta el destino. Cuando decimos que la ruta es estricta nos referimos a que la ruta entre dos direcciones sucesivas de la lista debe consistir en una sola red física. Si un *router* no puede encaminar directamente el datagrama a la siguiente dirección de la lista se produce un error y se genera un mensaje ICMP indicando destino inalcanzable (ver apartado 7).

[4]Con esta opción se pueden encaminar datagramas a través de una determinada red física y de esta forma poder comprobar su estado, realizar tareas de mantenimiento etc.

- Enrutado no estricto de fuente o LSR (*Loose Source Routing*).
  En este caso la lista que especifica el emisor no es estricta, es decir, se permiten múltiples saltos de redes físicas entre direcciones sucesivas de la lista.

El formato es equivalente al de la opción RR (figura 7) pero el funcionamiento es un poco diferente:

- Para SSR y LSR el campo **Code** vale 137 y 131 respectivamente.
- El *host* origen pone en el campo de IP Destino la dirección del primer *router* de la ruta.
- Los *n* campos **IP Address** almacenan las direcciones IP de los *routers* de la ruta.
- Cuando un datagrama llega a su siguiente destino y la ruta fuente no está vacía (**pointer <lenght**) el receptor:
  - Toma la dirección del campo **IP Address** indicada por **pointer** y la coloca como nueva dirección destino del datagrama.
  - Substituye la dirección del campo **IP Address** indicada por **pointer** por su dirección IP local correspondiente a la red por la que se enviará el datagrama.
  - Incrementa **pointer** en 4 (es decir 32 bits).
  - Transmite el datagrama a la nueva dirección IP de destino.
  - Este procedimiento asegura que la ruta de retorno se graba en orden inverso de modo que el receptor puede usar los datos de la opción para construir la ruta de vuelta.

*Sello temporal o IT (*Internet Timestamp*)*

El *timestamp* o sello de tiempo es una opción para forzar a algunos (o a todos) los *routers* en la ruta hacia el destino a poner un sello temporal (*timestamp*) en los datos de la opción. Los *timestamps* se miden en segundos y se pueden usar para depuración. No se pueden emplear para medir el rendimiento de la red por dos razones:

a. No son lo bastante precisos porque la mayoría de los datagramas se envían en menos de un segundo.

b. No son lo bastante precisos porque los *routers* no suelen tener los relojes sincronizados.

4. Resolución de direcciones (ARP)

En una sola red física, los *hosts* individuales se conocen en la red a través de sus direcciones físicas o direcciones de nivel 2. Los protocolos de alto nivel encaminan los datagramas al destino mediante direcciones globales o direcciones de nivel 3 (para el caso de redes IP la dirección es una dirección IP) de la red. Los *drivers* del dispositivo de red de la red física (por ejemplo los *drivers* de las tarjetas Ethernet de los PCs del laboratorio) no entienden las direcciones de nivel 3.

Por tanto, es necesario realizar una traducción de direcciones de nivel 3 a direcciones de nivel 2. En el caso de redes IP sobre Ethernet, la traducción se realiza de dirección IP a dirección Ethernet (que es una secuencia de 48 bits). Esta traducción se realiza en el *host* mediante una tabla usualmente denominada "caché ARP"[5]. Cuando la dirección no se encuentra en la caché ARP el *host* utiliza el protocolo ARP (*Address Resolution Protocol*) para averiguarla.

ARP es un protocolo de petición/respuesta. La petición se envía en forma de *broadcast* de nivel 2 a la red. Si una de las máquinas de la red reconoce su propia dirección IP en la petición, devuelve una respuesta ARP al *host* que la solicitó. La respuesta contendrá la dirección física correspondiente a la dirección IP solicitada. Esta dirección se almacenará en la caché ARP del *host* solicitante. Todos los posteriores datagramas enviados a esta dirección IP se podrán asociar a la dirección física correspondiente, que será la que utilice el *driver* del dispositivo de red para mandar el datagrama. La figura 8 muestra el formato de la trama ARP.

[5]Para consultar y manipular la cache ARP de un Linux se utiliza el comando `arp`. Para más información sobre este comando consulte la página de manual del arp.

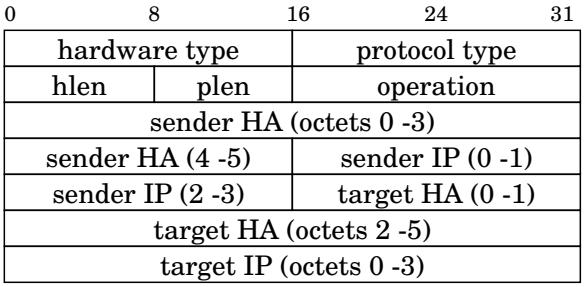
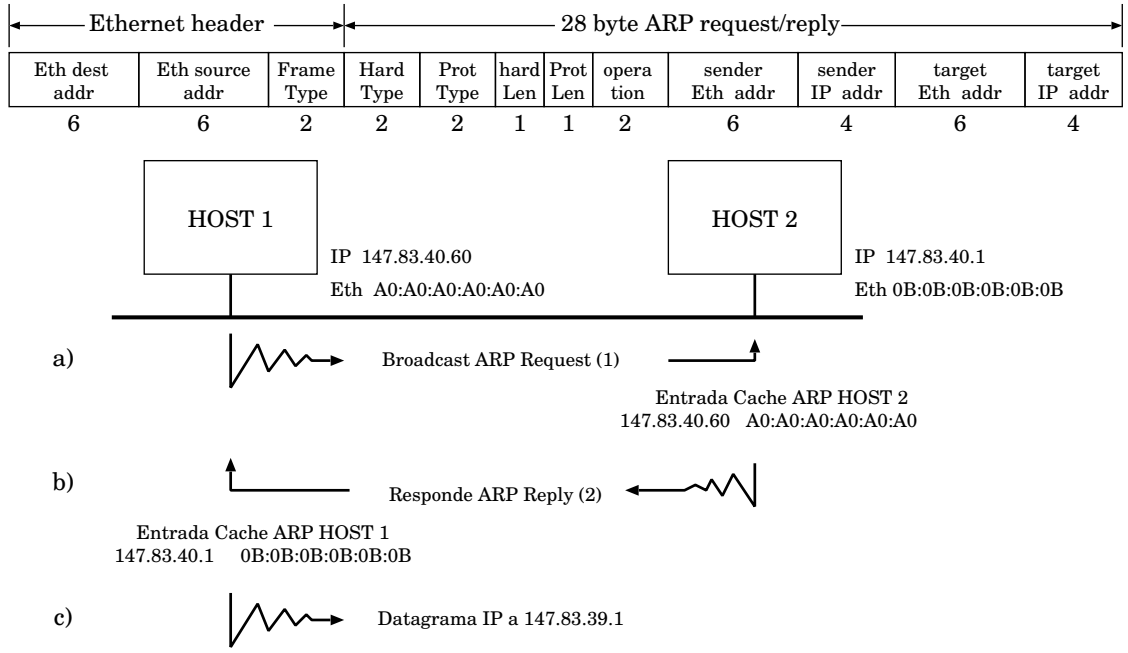


Figura 8: Formato de la trama ARP

Si una aplicación desea enviar datos a una determinada dirección IP de destino, el *software* de encaminamiento IP determina en primer lugar la dirección IP del siguiente salto (que puede ser el propio *host* de destino o un *router*) y en segundo lugar el dispositivo de red al que se debería enviar. A continuación se consulta el módulo ARP para hallar la dirección física.

El módulo ARP intenta hallar la dirección en su caché. Si la encuentra, devuelve la correspondiente dirección física. Si no la encuentra, descarta el paquete (se asume que al ser un protocolo de alto nivel volverá a transmitirlo) y genera un mensaje de broadcast de red para una solicitud ARP de dicha dirección física.

Cuando un *host* recibe un paquete ARP (bien un broadcast o una respuesta punto a punto), el dispositivo receptor pasa el paquete al módulo ARP, que lo añadirá a la caché ARP. La próxima vez que un protocolo de nivel superior quiera enviar un paquete a ese *host*, el módulo de ARP encontrará la dirección *hardware*, a la que se enviará el paquete.



(1) ARP Request

FF:FF:FF:FF:FF:FF-A0:A0:A0:A0:A0:A0-0806-0001-0800-06-04-01-A0:A0:A0:A0:A0:A0-147.83.40.60-00:00:00:00:00:00-147.83.40.1

(2) ARP Reply

A0:A0:A0:A0:A0:A0-0B:0B:0B:0B:0B:0B-0806-0001-0800-06-04-02-0B:0B:0B:0B:0B:0B-147.83.40.1-A0:A0:A0:A0:A0:A0-147.83.40.60

Figura 9: Ejemplo de ARP

5. Las subredes

Es obvio que una dirección de clase A sólo se asignará a redes con un elevado número de *hosts*, y que las direcciones de clase C son adecuadas para redes con pocos *hosts*. Sin embargo, esto significa que las redes de tamaño medio (aquellas con más de 254 *hosts* o en las que se espera que en el futuro haya más de 254 *hosts*) deben usar direcciones de clase B. El número de redes de tamaño pequeño y medio ha ido creciendo muy rápidamente en los últimos años y se temía que, de haber permitido que se mantuviera este crecimiento, todas las direcciones de clase B se habrían usado para mediados de los 90. Esto es lo que se conoce como "el problema del agotamiento de las direcciones IP".

Como se comentó en la sección 2 cada parte de la dirección IP (número de red y número de *host*) es responsabilidad de entidades diferentes: el número de red es asignado por INTERNIC, mientras que la asignación de los números de *host* es responsabilidad de la autoridad que controla la red. Como veremos a continuación el número de *host* puede dividirse aún más: esta división también es controlada por la autoridad propietaria de la red, y no por INTERNIC.

Debido al crecimiento explosivo de Internet, el uso de direcciones IP asignadas mediante las clases estándar se volvió demasiado rígido para permitir cambiar con facilidad la configuración de redes locales. Estos cambios podían ser necesarios cuando:

- Se instala una nueva red física.
- El crecimiento del número de *hosts* requiere dividir la red local en dos o más redes.

Para evitar tener que solicitar direcciones IP adicionales en estos casos, se introdujo el concepto de subred, donde el número de *host* de la dirección IP se subdivide de nuevo en un número de red y de *host*. Esta segunda red se denomina subred. La red principal consiste ahora en un conjunto de subredes y la dirección IP se interpreta como

<número de red, número de subred, número de *host*>

La combinación del número de subred y del *host* suele denominarse "dirección local" o "parte local". La creación de subredes se implementa de forma que es transparente a redes remotas. Un *host* dentro de una red con subredes es consciente de la existencia de éstas, pero un *host* de una red distinta no lo es y sigue considerando la parte local de la dirección IP como un número de *host*.

La división de la parte local de la dirección IP en números de subred y de *host* queda a libre elección del administrador local. Cualquier serie de bits de la parte local se puede tomar para la subred requerida. Por ejemplo, el *subnetting* es muy utilizado por los ISPs (*Internet Service Provider*) que se encargan de dar servicio de acceso a la Internet, éstos suelen tener asignado un rango de direcciones públicas de Internet, que dividen en subredes para después asignarlas a sus clientes.

La división se efectúa empleando una máscara de red diferente a las que dividen las direcciones IP en clases estándar. El funcionamiento de estas máscaras es equivalente al que se describe en la sección 2 para las clases estándar, es decir, los bits a cero en esta máscara indican posiciones de bits correspondientes al número de *host*, y los que están a uno, indican posiciones de bits correspondientes al número de subred. Las posiciones de la máscara pertenecientes al número de red se ponen a uno pero no se usan.

El tratamiento especial de "todos los bits a cero" y "todos los bits a uno" se aplica a cada una de las tres partes de dirección IP con subredes del mismo modo que a una dirección IP que no las tenga. Notar que al hacer *subnetting*, el número de máquinas sobre las que se hace *broadcast* (difusión) se reduce, optimizando el tráfico útil dentro de la red.

Como ejemplo supongamos que tenemos la red privada de clase C 192.168.1.0/24. En este caso tendremos un total de 254 (es decir $2^8 - 2$, por tener 8 bits de *hostid*) direcciones posibles de *host*, de la 192.168.1.1/24 hasta la 192.168.1.254/24. Observe que las direcciones 192.168.1.0 y 192.168.1.255 se reservan, la primera para identificar la red y la segunda como dirección de *broadcast*. Supongamos que hemos de repartir este direccionamiento en dos subredes de la misma dimensión, ya que tenemos dos oficinas en poblaciones diferentes y montaremos dos LANs unidas por la WAN (*Wide Area Network* o red de transporte que alquilaremos a algún operador). Así pues, tenemos que la parte de *hostid* ha de ceder terreno para la parte de subred. Como necesitamos hacer dos subredes, con un bit tendremos suficiente para identificar la subred en la que nos encontramos, quedándonos dos subredes: la 192.168.1.0/25 y la 192.168.1.128/25, con 126 direcciones posibles de *host* en cada subred (es decir $2^7 - 2$, con 7 bits de *hostid*). A continuación resumimos las direcciones de red, de máscara y de *hosts* implicadas en el *subnetting* de este ejemplo.

Red básica original: 11000000. 10101000. 00000001. 00000000 = 192.168.1.0/24

Máscara: 11111111. 11111111. 11111111. 00000000 = 255.255.255.0 = /24

Hay 254 direcciones posibles para asignar a la red:

host # 0 11000000. 10101000. 00000001. 00000001 = 192.168.1.1/24 (*netmask* 255.255.255.0)

host # 1 11000000. 10101000. 00000001. 00000010 = 192.168.1.2/24 (*netmask* 255.255.255.0)

host # 253 11000000. 10101000. 00000001. 11111110 = 192.168.1.254/24 (*netmask* 255.255.255.0)

Si hacemos *subnetting* nos quedarán estas dos subredes:

Subnet # 0 11000000. 10101000. 00000001. 00000000 = 192.168.1.0/25 (*netmask* 255.255.255.128)

Subnet # 1 11000000. 10101000. 00000001. 10000000 = 192.168.1.128/25 (*netmask* 255.255.255.128)

Con las siguientes direcciones de *host* disponibles en cada oficina:

Subnet # 0 de una oficina:

host # 0 11000000. 10101000. 00000001. 00000001 = 192.168.1.1/25 (*netmask* 255.255.255.128)

host # 1 11000000. 10101000. 00000001. 00000010 = 192.168.1.2/25 (*netmask* 255.255.255.128)

host # 125 11000000. 10101000. 00000001. 01111110 = 192.168.1.126 / 25 (*netmask* 255.255.255.128)

Subnet # 1 de la otra oficina:

host # 0 11000000. 10101000. 00000001. 10000001 = 192.168.1.129/25 (*netmask* 255.255.255.128)

host # 1 11000000. 10101000. 00000001. 10000010 = 192.168.1.130/25 (*netmask* 255.255.255.128)

host # 125 11000000. 10101000. 00000001. 11111110 = 192.168.1.254/25 (*netmask* 255.255.255.128)

6. Encaminamiento IP

6.1. Introducción

La función más importante de la capa IP es el encaminamiento de datagramas extremo a extremo a través de la red virtual. Por tanto esta función proporciona los mecanismos necesarios para interconectar distintas redes físicas. La formación de la red virtual que conecta múltiples redes se consigue por medio de unos *hosts* especiales denominados "*routers*".

Es importante distinguir entre un *hub*, un puente, un *router* y una pasarela.

El *hub* interconecta segmentos de LAN a nivel de interfaz de red y envía tramas entre ellos. El *hub* sirve como prolongación del cable físico que conecta las máquinas de la LAN y su única función es difundir la señal que llega por un cierto puerto (entrada) al resto de puertos. Los hubs pueden ser pasivos (si no amplifican las señales recibidas por sus puertos) o activos (si las amplifican).

El conmutador (*switch*) es un dispositivo parecido al *hub* pero en el que se realiza conmutación entre sus diferentes puertos, es decir, conmuta los paquetes observando sus direcciones físicas origen/destino.

Un puente (*bridge*) interconecta segmentos de LAN a nivel de interfaz de red y envía tramas entre ellos. Un puente realiza la función de retransmisión MAC (*Medium Access Control*) y es independiente de cualquier capa superior (incluyendo LLC). Proporciona, si se necesita, conversión de protocolos a nivel MAC. Un puente es transparente para IP. Es decir, cuando un *host* envía un datagrama a otro *host* en una red con el que se conecta a través de un puente, envía el datagrama al *host* y el datagrama cruza el puente sin que el emisor se dé cuenta. El puente

es capaz de aprender las direcciones *hardware* de las máquinas que tiene en cada puerto y aislar el tráfico y las colisiones de cada tramo LAN.

Un *router* interconecta redes físicas diferentes a nivel de la capa de red y encamina paquetes entre ellas. El *router* debe comprender la estructura de direccionamiento asociada con los protocolos que soporta (IP en nuestro caso) y debe elegir las mejores rutas de transmisión así como tamaños óptimos para los datagramas realizando fragmentación si lo considera oportuno.

La pasarela (*gateway*) interconecta redes a niveles superiores que los puentes y los *routers*. Una pasarela suele soportar el mapeado de direcciones de una red a otra, así como la transformación de datos entre distintos entornos para conseguir conectividad entre los extremos de la comunicación. Las pasarelas típicamente proporcionan conectividad de dos redes para un subconjunto de protocolos de aplicación soportados en cada una de ellas. Una pasarela es opaca para IP. Es decir, un *host* no puede enviar un datagrama IP a través de una pasarela: sólo puede enviarlo a la pasarela. La pasarela se ocupa de transmitirlo a la otra red con la información de los protocolos de alto nivel que vaya en él.
Estrechamente ligado al concepto de pasarela, está el de **cortafuegos** (*firewall*), que se usa para restringir la circulación de datagramas entre redes por motivos de seguridad.

6.2. Entrega directa e indirecta

El encaminamiento IP se puede dividir en dos partes: entrega directa y entrega indirecta:

- La entrega directa es la transmisión de un datagrama desde el *host* origen hasta el *host* destino a través de una sola red física, dicho de otra forma, dos *hosts* sólo pueden comunicarse mediante entrega directa si ambos están conectados directamente a la misma red física (por ejemplo, una sola red Ethernet).
  Básicamente en la entrega directa el emisor encapsula el datagrama dentro de una trama física, transforma la dirección IP destino en una dirección física y envía la trama resultante al destino a través del *driver* del dispositivo *hardware* correspondiente.
- La entrega indirecta es necesaria cuando el *host* destino no está conectado directamente a la red del origen, lo que implica que el datagrama deberá atravesar varias redes físicas, y para ello es necesario atravesar *routers*.
  La entrega indirecta es más compleja ya que el *host* origen ha de identificar al *router* al que debe entregar el datagrama, el primer *router* debe identificar cuál será el siguiente *router* al que debe enviar el datagrama, esto también se denomina identificar el "siguiente salto".
  La comunicación entre dos *routers* consecutivos de la ruta se realiza siempre mediante entrega directa, es decir, un determinado *router* de la ruta y el *router* del siguiente salto deben estar conectados a la misma red física, sino no es posible su comunicación. A su vez, el último *router* de la ruta que sigue el datagrama debe estar conectado a la misma red física que el *host* destino.

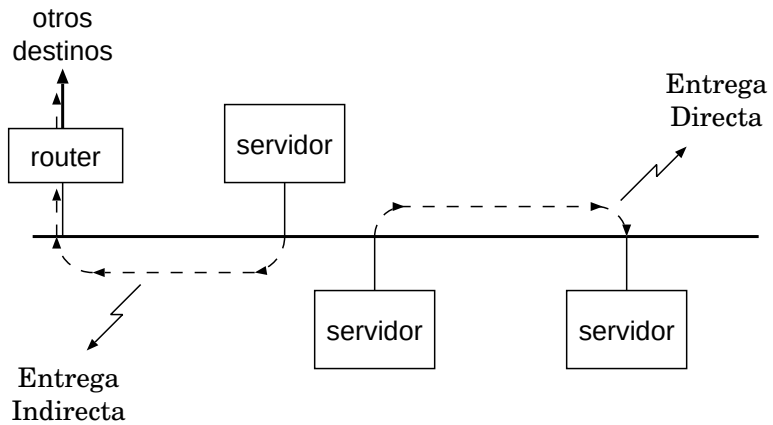


Figura 10: Entrega directa e indirecta

El *host* origen averigua si debe realizar entrega directa o no, es decir, si el *host* destino está conectado o no directamente a su red física mediante el prefijo de red. El *host* origen extrae el número de red de la dirección IP del destino y la compara con el número de red de su propia dirección IP. Si ambas se corresponden significa que se puede utilizar entrega directa, sino se ha de utilizar entrega indirecta.

6.3. El algoritmo de encaminamiento

El método utilizado por un *router* o un *host* para averiguar la siguiente máquina a la que debe enviar un determinado datagrama se denomina genéricamente como el "algoritmo de encaminamiento".

La gran mayoría de algoritmos de encaminamiento utilizan "tablas de encaminamiento". En las tablas de encaminamiento de cada *host* o *router* se almacena información sobre los posibles destinos y sobre cómo alcanzarlos. La información que contienen las tablas de encaminamiento debe ser mínima, ya que si cada tabla de encaminamiento contuviera información sobre cada posible dirección destino sería imposible mantener actualizadas las tablas. Además, las máquinas no tendrían suficiente espacio ni capacidad de proceso para manejarlas.

Se trata de minimizar la información que deben guardar las tablas aplicando un esquema de ocultación de información global, manteniendo sólo la información local mínima necesaria. Afortunadamente el esquema de direccionamiento IP permite realizar esto de forma fácil: como se mencionó en la sección 2, la dirección IP se divide en número de red y en número de *host*. Mediante este esquema es posible almacenar números de red en las tablas de encaminamiento en lugar de direcciones IP completas. De esta forma se ocultan los detalles de qué *hosts* y cómo están conectados a las diferentes redes y se minimiza el tamaño de las tablas de encaminamiento.

El contenido de las tablas de encaminamiento suelen ser pares del tipo <N,R>. Donde N es un número de red y R es la dirección IP del *router* en el siguiente salto para alcanzar dicha red (por tanto el *router* debe estar conectado a la misma red física).

Para simplificar más las tablas de encaminamiento aparece el concepto de "ruta por defecto". La ruta por defecto contiene la dirección del *router* del siguiente salto al que se deben enviar los datagramas (también denominado *router* por defecto) si tras recorrer la tabla de encaminamiento no se encontró ninguna ruta específica para el número de red al que va dirigido el datagrama.

Aunque se ha comentado la conveniencia de encaminar en base al número de red destino, la tabla de encaminamiento permite especificar una ruta especial para un *host* en particular.

De esta forma el algoritmo básico de encaminamiento de un datagrama IP es el siguiente:

a. Extraer la dirección IP destino D.

b. Computar el prefijo de red N con la máscara local.

c. Si N se corresponde con alguna red física a la que estamos conectados se realiza entrega directa (realizando ARP).

d. Si no se puede realizar entrega directa, se comprueba si hay ruta específica para D y en caso afirmativo se envía el datagrama al *router* del salto siguiente especificado en la tabla.

e. Si no hay ruta específica, se comprueba si hay una ruta para la red N y en caso afirmativo se envía el datagrama al *router* del salto siguiente especificado en la tabla.

f. Si no hay ruta para N, se envía el datagrama al *router* por defecto especificado en la tabla.

g. Si no hay ruta por defecto y el *software* de encaminamiento IP ha llegado a este punto se produce un error (que se puede reportar mediante ICMP).

Como conclusiones importantes del algoritmo de encaminamiento IP podemos destacar que:

a. En la mayor parte de implementaciones, el tráfico dirigido a una determinada red desde un *host* origen va a seguir el mismo camino aunque existan diversas posibilidades.

b. Sólo el último *router* de la ruta puede determinar si el *host* destino está disponible (estas situaciones se reportan mediante ICMP). Además también necesitamos reportes de los *routers* intermedios si sucede algún problema.

c. Los datagramas que viajen de A a B pueden seguir rutas diferentes a los datagramas que viajen de B a A.

La figura 11 muestra 4 redes interconectados a través de 2 *routers* y la tabla de rutas del *router* R_1.

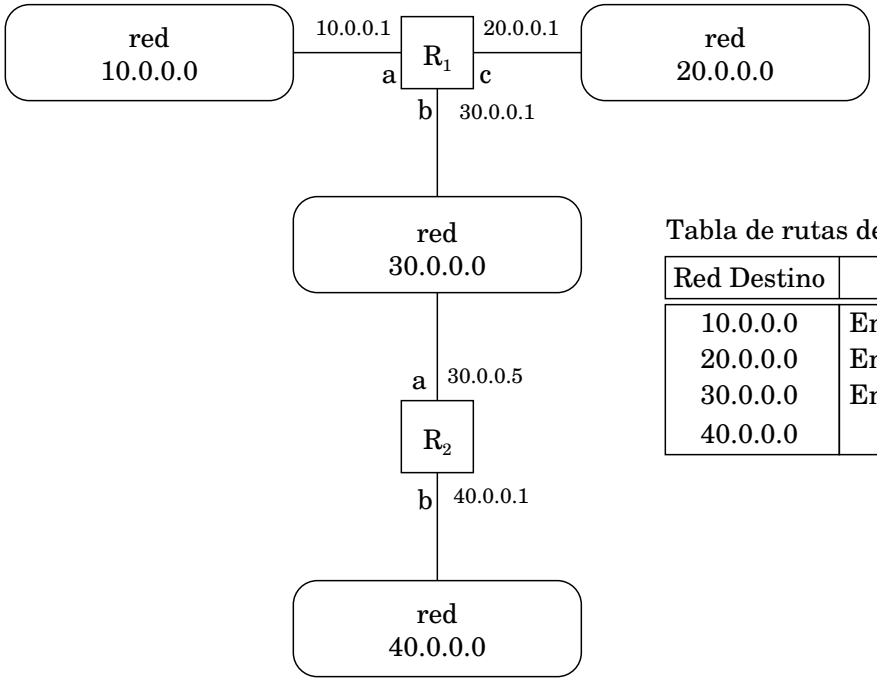


Tabla de rutas de R_1

Red Destino	Enrutar a:	iF
10.0.0.0	Entrega directa	a
20.0.0.0	Entrega directa	c
30.0.0.0	Entrega directa	b
40.0.0.0	30.0.0.5	b

Figura 11: Ejemplo de encaminamiento

Es importante entender que a excepción de la disminución del campo TTL el *software* de encaminamiento no modifica la cabecera del datagrama original. En particular, las direcciones IP origen y destino permanecen inalteradas durante toda la ruta[6].

Por lo que respecta a los datagramas entrantes:

- Cuando un datagrama llega a un *host* el *driver* del dispositivo de red lo entrega al *software* IP para su procesamiento. El *software* IP determina si el datagrama es para el propio *host* en cuyo caso lo pasa al *software* del protocolo de nivel alto apropiado. El datagrama se descarta si no es para el propio *host*.
- En el caso de los *routers,* estos deben encaminar el datagrama si no va dirigido hacia ellos.

Decidir si una máquina es o no la destinataria de un datagrama no es una tarea tan trivial como a simple vista pueda parecer. En primer lugar pueden haber muchos interfaces de red cada uno de ellos con su correspondiente dirección IP, se debe comprobar la correspondiente identificación de subred (si la red está dividida) y además se deben reconocer los mensajes de *broadcast* y los de *multicast*.

7. ICMP- Internet Control Message Protocol

ICMP (*Internet Control Message Protocol*), es un protocolo que se utiliza para señalizar errores u otro tipo de condiciones que se pueden producir durante una comunicación. Estos mensajes pueden ser generados por cualquier dispositivo de red de nivel 3 que necesite informar de una condición de error a un dispositivo emisor cuya transmisión ha causado el error. Esta condición implica que los mensajes ICMP deberán ir encapsulados en datagramas IP, para que puedan

[6]Excepto en la opción SR comentada en la sección 3.2

type	*code*	*Description*	*Query*	*Error*
0	0	echo reply	•	
3		destination unreachable		
	0	network unreachable		•
	1	host unreachable		•
	2	protocol unreachable		•
	3	port unreachable		•
	4	fragmentation needed but fg not set		•
	5	source route failed		•
	6	destination network unknown		•
	7	destination host unknown		•
	8	source host isolated (obsolete)		•
	9	destination network adimistrativly prohibited		•
	10	destination host administratively prohibited		•
	11	network unreachable for TOS		•
	12	host unreachable for TOS		•
	13	communication administratively prohibited by filtering		•
	14	host precedence violation		•
	15	precedence cut-off in effect		•
4	0	source quench (elementary flow control)		•
5		redirect		•
	0	redirect for network		•
	1	redirect for host		•
	2	redirect for TOS and network		•
	3	redirect for TOS and host		•
8	0	echo request	•	
9	0	router advertisement	•	
10	0	router solicitation	•	
11		time exceeded		
	0	time-to-live equals 0 during transit		•
	1	time-to-live equals 0 during reassembly		•
12		parameter problem		
	0	IP header bad		•
	1	required option missing		•
13	0	timestamp request	•	
14	0	timestamp reply	•	
15	0	information request (obsolete)	•	
16	0	information reply (obsolete)	•	
17	0	address mask request	•	
18	0	address mask reply	•	

Tabla 4: Mensajes ICMP

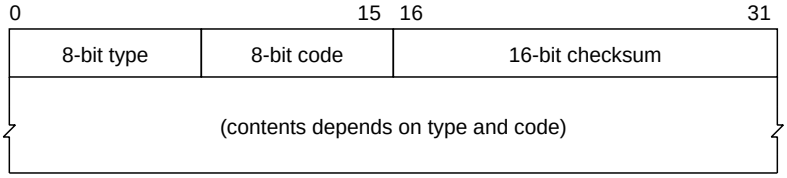


Figura 12: Mensaje ICMP

ser encaminados desde cualquier punto de la red hasta el receptor del mensaje ICMP. Sin embargo, como puede verse en la figura 2, ICMP no utiliza ningún protocolo de transporte ya que el mensaje ICMP no va dirigido a la aplicación de comunicaciones que generó el error, sino a la torre de comunicaciones (TCP/IP) gestionada por el Sistema Operativo que será quien trate el mensaje ICMP y actúe en consecuencia (por ejemplo indicando la condición de error a la aplicación).

Un mensaje ICMP tiene la estructura mostrada en la figura 12. Tanto el campo de tipo (*type*) como el campo de código (*code*), determinan los diferentes mensajes ICMP, así como formato del contenido del mensaje. La tabla 4 muestra los diferentes tipos de mensajes ICMP clasificados según el campo *type* y el campo *code*.

Es importante destacar que muchos de los mensajes ICMP son obsoletos, o bien algunos de ellos estan implementados en *hosts* y otros en *routers*, o bien algunos de ellos están específicamente deshabilitados porque han demostrado ser puntos débiles de seguridad.

La tabla 4 muestra los diferentes tipos de mensajes ICMP. Las dos últimas columnas de la tabla indican que hay dos tipos de mensajes ICMP, peticiones (*query*) y errores (*error*). Los mensajes ICMP de error son mensajes que se envían al dispositivo cuyo paquete ha generado el error. En este caso, los mensajes ICMP de error incluyen la cabecera IP del datagrama que generó el error (20 bytes) y los primeros 8 bytes del datagrama (incluiría el inicio de la cabecera TCP ó UDP). De esta forma, el dispositivo que generó el error sabe qué aplicación es la responsable del error generado.

Los mensajes ICMP de petición se utilizan para obtener información específica (según el tipo y código del mensaje ICMP) de un dispositivo de red (*hosts*, *routers*, . . .). El mensaje de petición corresponde a los etiquetados con *request* en la tabla, mientras que los mensajes de respuesta son los *reply*.

7.1. ICMP *echo* y el programa `ping`

Los mensajes ICMP de *echo* han sido utilizados, tradicionalmente, para verificar el estado de las conexiones entre dos dispositivos de red (*hosts*, *routers*, . . .).

En el mensaje ICMP de *echo*, el campo *type* tiene los valores 0 u 8 (*reply*, *request*) y el campo *code* tiene el valor 0. El *host* que envía un mensaje de petición (*request*) ICMP de *echo*, establece los campos *identifier* y *sequence number* a unos valores que él escoge, así como el contenido de los datos. El *host* que recibe la petición (*request*) responde con un mensaje de respuesta (*reply*) con los mismos valores de los campos *identifier*, *sequence number* y contenido que recibió (realiza un “eco” del mensaje recibido).

El programa `ping` se utiliza para enviar una petición de mensaje ICMP de *echo*. La respuesta (*reply*) se maneja directamente en el *kernel* (núcleo) del sistema y automáticamente envía el mensaje de respuesta (*reply*). En las implementaciones de Unix, el campo *identifier* se establece con el valor del PID del proceso que envía los mensajes, lo que permite a `ping` identificar las respuestas recibidas en el caso de que hayan más procesos `ping` ejecutándose.

El valor del campo *sequence number* se inicializa a 0 y se incrementa en cada mensaje nuevo que se envía, permitiendo detectar si se ha perdido algún paquete.

El programa `ping` calcula el RTT (*round trip time*), tiempo transcurrido entre que se envía una petición de *echo* y se recibe la respuesta correspondiente.

8. Desarrollo de la práctica

8.1. Virtualización

Uno de los principales problemas que presenta esta asignatura de laboratorio es la necesidad de tener a disposición del alumno toda una red con la posibilidad de cambiar la topología. Ello implica disponer de elementos de interconexión físicos tales como hubs, switches, routers, etc. Una solución a este problema es utilizar la virtualización de estos elementos.

Los entornos virtualizados tienen dos elementos básicos:

- **El "host" de virtualización.** La virtualización se lleva a cabo en una máquina física (hardware) en la que estará instalado un cierto sistema operativo y herramientas de virtualización. A este conjunto de elementos es al que llamamos "host" de virtualización[7].
- **El "guest" o máquina virtual.** El sistema guest es la máquina virtual que se ejecuta sobre el sistema host. El guest suele ser un sistema operativo tradicional que se ejecuta exactamente igual que si estuviera instalado en un hardware real. Típicamente muchas máquinas virtuales guest se pueden ejecutar sobre un solo host o máquina física. El sistema host por tanto, será la plataforma subyacente que si es necesario, emulará el hardware al que los sistemas guest tendrán acceso (implementando las interfaces de los drivers de los dispositivos correspondientes).

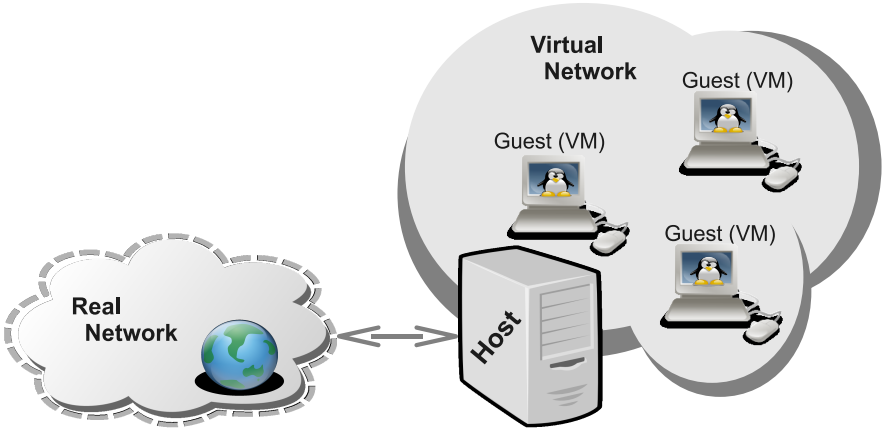


Figura 13: Virtualización: un host y varios guest o máquinas virtuales

En la Figura 13 se puede observar de forma gráfica lo anteriormente comentado.

Respecto al software de virtualización disponible, comentar que hay diferentes tipos de implementaciones dependiendo del grado de complejidad en cuanto a la cantidad de hardware virtualizado que se hace accesible a los guests y a la capacidad de gestión de las máquinas virtuales guest.

En este sentido, hay disponible software de virtualización que implementa una gran cantidad de drivers de dispositivos físicos tales como tarjetas de red, tarjetas USB, dispositivos de sonido etc. Este tipo de software hace totalmente transparente la virtualización a los sistemas operativos guest. En este caso, para los sistemas operativos guest será totalmente equivalente funcionar sobre un host que proporciona virtualización que funcionar sobre una máquina hardware real.

En nuestro caso, vamos a utilizar un tipo de virtualización llamado User Mode Linux (UML) que es útil para entornos donde tanto el host como el guest son sistemas operativos Linux. UML fue creado inicialmente como una herramienta para agilizar el desarrollo del núcleo de Linux. La idea era que un programador pudiera probar sus contribuciones al núcleo Linux haciéndolo arrancar como un proceso más. De esta forma, si los desarrollos de un programador inestabilizaban el núcleo UML y lo "colgaban", sólo era necesario matar el proceso kernel UML y no reiniciar la máquina hardware.

[7] No confundir con el término "host" que también utilizamos en redes para denominar a una máquina de usuario final.

Si disponemos de un núcleo Linux UML llamado “linux“, éste podría ser ejecutado en el host como si fuera un proceso cualquiera con el siguiente comando:

```
host$ linux ubd0=Debian-3.0r0.ext2 mem=32M
```

En este caso, a través de la línea de comandos, le decimos al núcleo UML que el sistema de archivos raíz que debe utilizar está en el archivo Debian-3.0r0.ext2 (imagen ISO de un sistema de ficheros estándar) y que puede utilizar hasta 32 MB de memoria. Utilizando la línea de comandos podemos pasar una gran cantidad de parámetros al núcleo UML (sistema guest).

Respecto a UML utilizado para virtualización, comentar que este sistema es conceptualmente un poco diferente a los sistemas de virtualización hardware anteriormente comentados (donde el host implementa los drivers para el hardware disponible y la virtualización es transparente para los guests).

En nuestro caso, el guest es un sistema operativo especialmente diseñado para ser virtualizado o dicho de otra forma, para ser ejecutado sobre una plataforma Linux. Así pues, en UML es el guest el que ha sido modificado en mayor medida para poder funcionar sobre el host. Por su parte el host requiere herramientas de virtualización en general no demasiado complejas.

En general, podemos decir que este tipo de virtualización es menos compleja que la virtualización hardware aunque menos flexible: el guest debe ser un kernel Linux compilado con arquitectura UML y el host debe ser también un sistema Linux (eso sí, puede ser una distribución convencional).

En este laboratorio, la parte que más nos interesa es la interconexión de máquinas virtuales. La interconexión de máquinas virtuales es un requisito indispensable en cualquier entorno de simulación así que todas las implementaciones de virtualización (incluida UML) permiten crear redes de máquinas virtuales conectadas entre ellas[8].

Obviamente, la interconexión de máquinas virtuales no se realiza utilizando transmisiones por medios físicos como cables o tecnología inhalámbrica, sino que se emula mediante un programa software. En nuestro caso, utilizamos el programa ”uml_switch“ que ha sido desarrollado por los creadores de UML. El funcionamiento del programa uml_switch a ”grosso modo” es el siguiente:

- Cuando el host arranca una máquina virtual ”registra” cada una de sus interfaces de red en ciertas instancias del programa uml_switch.
- Cuando una máquina virtual envía una trama por un cierto interfaz (Ethernet en nuestro caso), en realidad se la envía a un proceso uml_switch.
- El uml_switch se puede comportar de dos formas:
  - Como hub, es decir, restransmitiendo los bits leidos (trama) al resto de interfaces registrados.
  - Como switch, es decir, analizando la dirección MAC destino y enviando la trama al interfaz correspondiente.

Montar una virtualización con varios guests y diferentes topologías de red no resulta sencillo. Por esta razón, el Departamento de Ingeniería Telemática (DIT) de la Universidad Politécnica de Madrid (UPM) desarrolló un sistema para definir y ejecutar redes de máquinas virtuales usando UML. Este proyecto se llama VNUML (Virtual Network User Mode Linux) y es el que utilizaremos para configurar los escenarios de red de las prácticas.

Una forma sencilla de explicar lo que hace VNUML, es la siguiente:

- VNUML define un lenguaje basado en XML, que permite definir redes de máquinas virtuales UML. Mediante este lenguaje se pueden definir máquinas virtuales UML, configuraciones (configuración de interfaces de red, direcciones Ethernet, direcciones IP, etc.), así como la topología de la interconexión (mediante switches, hubs y lineas punto a punto).
- VNUML también proporciona un script o programa llamado “vnumlparser.pl“ para arrancar escenarios descritos con el lenguaje XML anteriormente comentado.

[8] Además, generalmente también es posible conectar las máquinas guest con el sistema host.

Utilizando VNUML, en el departamento de ingeniería telemática (ENTEL) de la UPC hemos desarrollado dos scripts o programas para arrancar nuestros escenarios de prácticas y para podernos conectar a las máquinas virtuales.

En primer lugar, el script "simctl" es el programa que se utiliza para arrancar los escenarios de red que utilizaremos en el laboratorio. Si ejecuta el script sin parámetros obtendrá la lista de posibles escenarios de prácticas.

```
host$ simctl

simctl ( icmp | routing | subnetting ) (OPTIONS)

OPTIONS
    start                  Starts scenario
    stop                   Stops scenario
    vms                    Shows vm's from simulation
    labels (vm | "ALL")    Shows sequence labels for one vm or ALL for all vm's
    exec (vm | "ALL" | "group") label  Exec label in one vm or in ALL vm's or in a group
                           of vm's
```

De la salida del script simctl anterior nos interesa la primera línea, la cual nos muestra los escenarios disponibles. En este caso, los escenarios disponibles son icmp, routing y subnetting.

Para arrancar un escenario en particular, deberá ejecutar el script simctl desde el host con el nombre del escenario y la opción start.

```
host$ simctl subnetting start
.......
.......
.......
Total time elapsed: 77 seconds
host$
```

Ejecutando el comando anterior arrancamos el escenario subnetting. Tendrá que esperar un poco y verá varias decenas de lineas de texto. Finalmente, el comando acaba indicándonos el tiempo que ha tardado en iniciar el escenario y nos devuelve el prompt o control sobre la consola de comandos. En este instante, las máquinas virtuales así como sus interconexiones mediante hubs o switches han sido creadas. Puede parar la simulación mediante el comando anterior substituyendo start por stop. En general, si no hay errores solo se necesita ejecutar start y stop de los escenarios deseados.

Una vez arrancada la simulación, podemos utilizar el comando "simterm" para obtener información de la simulación y también para obtener terminales conectados a las máquinas virtuales. Siguiendo con el ejemplo del escenario de subnetting puede observar el estado de las máquinas virtuales de la siguiente forma:

```
host$ simterm subnetting info
        virt1           Running         ---------
        virt2           Running         ---------
        virt3           Running         ---------
        virt4           Running         ---------
host$
```

Finalmente, puede obtener una consola de comandos para una máquina virtual mendiante el siguiente comando:

```
host$ simterm subnetting get virt1
```

En este caso, desde el host obtendremos una consola de comandos para el guest o máquina virtual virt1. Para entrar en el sistema virt1 (o en cualquier otro), presione enter, introduzca **root** como usuario y **xxxx** como password. Ahora usted está en la consola de comandos de la máquina virt1. Si por despiste o cualquier otra causa cierra la consola que tiene con una máquina virtual, siempre la podrá recuperar dicha consola volviendo a ejecutar el comando anterior en el host.

Finalmente, comentar que el script simctl que hemos desarrollado crea unos interfaces de red especiales en el sistema host denominados "taps". Estos interfaces serán utilizados como "sondas" para capturar el tráfico intercambiado entre

los guests en las diferentes redes virtuales. En particular, el script crea cuatro interfaces llamados: tap0, tap1, tap2 y tap3. En cada escenario, algunos de estos interfaces, o todos ellos, estarán conectados a las diferentes instancias de uml_switch. Ello permitirá que en su host pueda utilizar su analizador de protocolos para observar el tráfico intercambiado en las diferentes redes virtuales. En cada práctica se detalla el esquema de red virtualizado, así como la ubicación de los interfaces sonda.

8.2. Comandos básicos

ifconfig

El comando `ifconfig` (**i**nter**f**ace **config**uration) se utiliza para la configuración de los interfaces de red, por ejemplo:

- Consultar la asignación de direcciones IP de los interfaces de una máquina
- Establecer la dirección IP y la máscara de red de un interfaz
- Limitar el valor de la MTU de un interfaz
- Establecer la dirección física del interfaz, si el interfaz soporta esta operación
- etc, etc

La sintaxis que utiliza el comando `ifconfig` es variada en función de la operación a realizar, pero para la asignación de direcciones IP, la sintaxis es:

`ifconfig` *interface #IP* `netmask` *#mask* `broadcast` *IP_broadcast*

donde, interface es el nombre del interfaz que se quiere configurar (por ejemplo eth1 o eth2), #IP es la dirección IP que se asignará a ese interfaz (por ejemplo 192.168.1.1), `netmask` forma parte de la sintaxis e indica que el siguiente argumento es la máscara de red que se va a utilizar (*#mask*), por ejemplo 255.255.255.0. Finalmente `broadcast` es una palabra reservada que indica que el siguiente argumento es la dirección broadcast IP que se utilizará en ese interfaz (por ejemplo 192.168.1.255). A modo de ejemplo, el siguiente comando:

```
$ifconfig eth0 192.168.1.1 netmask 255.255.255.0 broadcast 192.168.1.255
```

Nota. El símbolo $ es el "prompt", no se debe teclear en la consola.

configura el interfaz eth0 con la dirección IP 192.168.1.1 con máscara 255.255.255.0 y dirección IP broadcast 192.168.1.255.

El hecho de asignar una dirección IP a un interfaz, provoca que el sistema operativo actualice su tabla de rutas, añadiendo automáticamente la ruta de entrega directa relacionada con la asignación realizada.

El comando `ifconfig` también permite deshabilitar un interfaz si no se desea hacer uso de él, por ejemplo:

```
$ifconfig eth0 down
```

En este caso, la ruta de entrega directa relacionada con la configuración de eth0 queda borrada de la tabla de rutas, así como todas la rutas de entrega indirecta que utilizasen eth0 como interfaz de salida.

Para volver a "levantar" un interfaz con la configuración que tenía préviamente se debe introducir el siguiente comando:

```
$ifconfig eth0 up
```

Para consultar cómo están configurados los interfaces de una máquina, se utiliza la sintaxis:

`ifconfig` [interface]

donde [interface] es opcional. Si no se utiliza el nombre de ningún interfaz, el comando nos retorna la configuración de todos los interfaces "levantados", tengan o no tengan una IP asignada. Si proporcionamos un nombre de interfaz, el comando nos retorna la configuración del interfaz. Para consultar la configuración de todos los interfaces de una máquina independientemente de si están o no "up":

```
$ifconfig -a
```

Una máquina puede contestar en relación a la dirección IP de un interfaz, aunque éste, esté en estado down. A veces este comportamiento trae problemas en el laboratorio, por ello, lo más conveniente cuando una interfaz de red no es necesaria es desasignarle la dirección IP y a continuación ponerla en estado down. Para eliminar la dirección IP a un interfaz debemos en realidad asignar la dirección 0.0.0.0. Por ejemplo, para eliminar la dirección IP del interfaz eth1 ejecutaríamos el siguiente comando:

```
$ifconfig eth1 0.0.0.0
```

Finalmente, mediante el comando ifconfig también puede asignar más de una dirección IP a un interfaz de la siguiente forma:

```
$ ifconfig eth1 10.0.0.1/24
$ ifconfig eth1:0 10.0.0.2/24
$ ifconfig eth1:1 10.0.0.3/24
```

En el ejemplo anterior, el primer comando asigna la primera dirección (10.0.0.1) a la tarjeta de red eth1. El segundo comando asigna una segunda dirección (10.0.0.2), denominada alias "0", a la tarjeta de red eth1. Finalmente, el tercer comando asigna una tercera dirección (10.0.0.3), denominada alias "1", a la tarjeta de red eth1. De esta forma, se podrían ir añadiendo direcciones IP sucesivamente a un interfaz de red.

arp

El comando `arp` se utiliza para consultar y manipular la tabla arp del ordenador. En las prácticas utilizaremos este comando para saber el contenido de la tabla arp de la siguiente manera:

```
$arp -an
```

Si deseamos borrar una entrada de la tabla arp de nuestro ordenador, utilizamos la sintaxis:

```
$arp -d hostname
```

donde hostname es la IP (ej. 147.83.40.106) o el nombre FQDN (Fully Quality Domain Name) (ej. huffman6.upc.edu) del ordenador que queremos borrar de nuestra tabla arp.

route

El comando `route` sirve para consultar y manipular las tablas de rutas del sistema. Para consultar las tablas de rutas, el comando que se utiliza es:

```
$route -n
```

Para explicar la salida de este comando, lo haremos sobre un ejemplo (solo representamos los campos que son de utilidad para las prácticas):

```
router1: # route -n
Kernel IP routing table
Destination       Gateway           Genmask           Iface
10.0.0.0          0.0.0.0           255.255.255.0   eth1
192.168.1.0       0.0.0.0           255.255.255.0   eth2
0.0.0.0           192.168.1.2     0.0.0.0           eth2
```

El campo `Destination` indica la dirección de red destino. El campo `Gateway` indica a través de quien se debe llevar a cabo la entrega de un paquete. Si el valor del campo `Gateway` es 0.0.0.0 indica que la entrega es directa, y si el valor de `Gateway` es una dirección IP de un host, indica que el paquete se debe entregar a este host (entrega indirecta) para llegar a la red destino "`Destination`". El campo `Genmask` es el valor de la máscara que se utilizará para calcular las direcciones IP que pertenecen a una cierta red. Por ejemplo, para la red 10.0.0.0, se utiliza una `Genmask` de 255.255.255.0; lo cual nos indica que el rango de direcciones IP que va desde la 10.0.0.0 a la 10.0.0.255 son direcciones que pertenecen a esa red destino. Finalmente el campo `Iface` nos indica a través de qué interfaz se llevará a cabo la transmisión del paquete.

El comando `route` se utiliza para añadir y borrar rutas, utilizando la siguiente sintaxis:

```
route (add | del) -net #IP_net netmask #mask [gw #IP_router]
```

Por ejemplo, suponiendo que tenemos la tabla de rutas anterior y queremos añadir una ruta de entrega indirecta a la red 192.168.100.0/24 a través del router 10.0.0.5:

```
$route add -net 192.168.100.0 netmask 255.255.255.0 gw 10.0.0.5
```

Si posteriormente ejecutamos el comando `route -n` obtenemos:

```
router1: # route -n
Kernel IP routing table
Destination       Gateway           Genmask           Iface
10.0.0.0          0.0.0.0           255.255.255.0   eth1
192.168.1.0       0.0.0.0           255.255.255.0   eth2
192.168.100.0   10.0.0.5            255.255.255.0   eth1
0.0.0.0           192.168.1.2     0.0.0.0           eth2
```

Si deseamos volver a borrar la ruta, ejecutamos el comando:

```
$route del -net 192.168.100.0 netmask 255.255.255.0 gw 10.0.0.5
```

Ejercicio 1

En este ejercicio se estudia el funcionamiento básico del envío de datagramas IP en una red Ethernet. Para ello se utilizará la red virtualizada que se muestra en la figura 14.

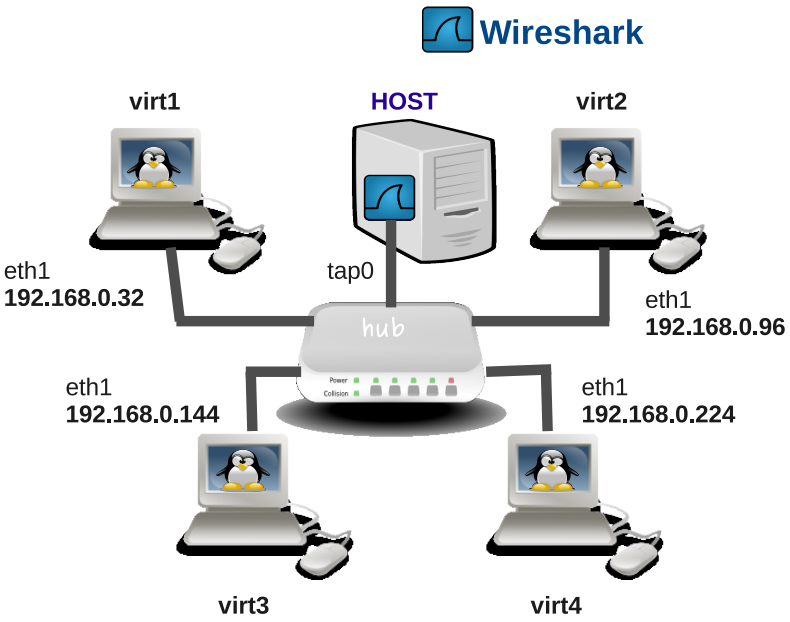


Figura 14: Red virtual para el ejercicio 1

Para iniciar la red la figura 14 en su plataforma host teclee el siguiente comando:

```
host$ simctl subnetting start
```

> *Nota. host$ es el "prompt", no se debe teclear en la consola.*

Una vez arrancada la red virtual obtenga cuatro consolas de comandos, una para cada uno de los *guests* o máquinas virtuales. Estas máquinas virtuales se llaman respectivamente virt1, virt2, virt3 y virt4 (podrá observar estos nombres en el título de la ventana y en los prompts).

Estas máquinas se configurarán de acuerdo con los datos de la tabla 5.

Guest	Network Interface	IP Address
virt1	eth1	192.168.0.32
virt2	eth1	192.168.0.96
virt3	eth1	192.168.0.144
virt4	eth1	192.168.0.224

Tabla 5: Direcciones IP del ejercicio 1

a. Analizando las direcciones IP de las máquinas virtuales de la figura 14, averigüe cuál es la máscara más grande (mayor número de unos) que hace que todas las máquinas pertenezcan a la misma red IP.

b. Mediante el comando `ifconfig` (descrito en la sección 8.2) configure la dirección IP con la máscara anteriormente hallada en cada máquina virtual.

c. Vamos a realizar capturas de tráfico desde el sistema host. Desde una consola de comandos en el host, ejecute el analizador de protocolos.

```
host$ wireshark &
```

Una vez abierto el analizador de protocolos, seleccione tap0 como interfaz de captura.

d. Visualice la cache ARP de la máquina virtual1:

```
virtual1$ arp -n
```

e. Desde la máquina virtual1, ejecute el comando ping con las opciones necesarias para que únicamente se envíe un mensaje *icmp-request* a la máquina virtual2.

f. Espere unos segundos y vuelva a ejecutar el comando ping anterior.

g. Comente el estado de las caches ARP de todas las máquinas virtuales.

h. Comente las diferencias entre las tramas capturadas para el primer ping y para el segundo.

i. En la cache ARP de la máquina virtual1, borre la entrada ARP de la dirección IP de la interfaz eth1 de la máquina virtual2. Vuelva a ejecutar el comando ping anterior y comente los resultados.

j. Ahora vamos a introducir una traducción errónea en virtual1 para la dirección IP de la interfaz eth1 máquina virtual2. Para ello ejecute el siguiente comando:

```
virtual1$ arp -s 192.168.0.96 00:70:48:29:5c:99 temp
```

Vuelva a ejecutar el comando ping anterior y comente los resultados.

k. Ahora necesitamos "limpiar" la cache arp de virtual1, así que ejecute el siguiente comando para eliminar todas las entradas:

```
virtual1$ ip neigh flush all
```

A continuación averigüe cuál es la máscara necesaria para dividir la red en dos subredes de tal forma que las máquinas virtual1 y virtual2 queden en una subred y las máquinas virtual3 y virtual4 queden en otra subred. Configure la máscara elegida y comente cómo comprobar que efectivamente a nivel IP ahora las máquinas están divididas en dos redes.

l. Averigüe, configure y compruebe cuál es la máscara más pequeña (mínimo número de unos) que hace que cada una de las máquinas virtuales quede aislada en una red IP diferente.

m. Ahora vamos a probar qué sucede cuando tenemos máscaras de diferentes valores en las diferentes máquinas. Para ello configure la máscara /24 en las máquinas virtual1 y virtual3 y configure la máscara /25 en las máquinas virtual2 y virtual4. Comente en detalle qué sucede cuando se hacen pings desde la máquina virtual1 al resto de máquinas.

n. De igual forma, comente qué sucede cuando se hacen pings desde la máquina virtual2 al resto de máquinas.

Ejercicio 2

En este ejercicio se pretende realizar una configuración básica de red para una empresa pequeña. Supondremos que vamos a realizar la configuración de una empresa ficticia llamada Acme SA que está formada por tres departamentos: marketing, ventas y producción.

Cada uno de estos departamentos tiene una red IP que está representada por su mínima expresión: una máquina de usuario o host (host1, host2 y host3) y un router. Además de las redes de los departamentos, tenemos otra red IP para la interconexión de routers (red backbone).

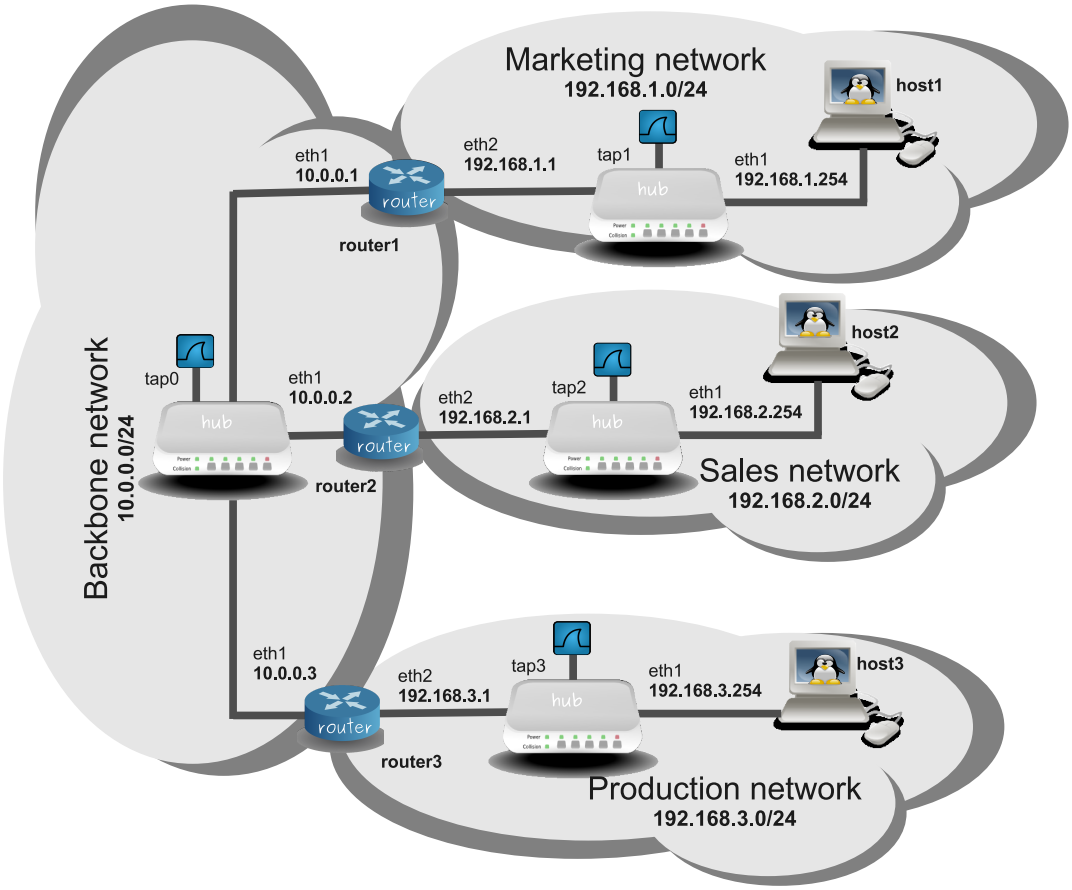


Figura 15: Estructura de la empresa Acme SA.

En la figura 15 se observa la red de la empresa. Para iniciar la red virtualizada de la figura 15 en su plataforma host teclee el siguiente comando:

```
host$ simctl routing start
```

> *Nota. No confundir la "plataforma host", con las máquinas virtuales ("guests") que simulan hosts o máquinas de usuario (host1, host2 y host3).*

Una vez arrancada la red virtual le aparecerán cuatro consolas de comandos que corresponden a las máquinas virtuales host1, router1, host2 y router2. Las máquinas host3 y router3 ya están configuradas con los parámetros necesarios.

Estas máquinas se configurarán de acuerdo con los datos de la tabla 6.

a. Justifique cuál sería la máscara de red adecuada para las redes de cada uno de los departamentos.

b. Configure las tarjetas de red del host1, host2, router1 y router2 que forman las redes de los departamentos utilizando la máscara elegida anteriormente.

c. Compruebe la conectividad entre el host1-router1 y entre host2-router2.

d. Ahora configure las tarjetas de red eth1 de los routers a los que tiene acceso.

e. Compruebe la conectividad en la red backbone y averigüe la dirección MAC de la tarjeta eth1 del router3.

Virtual Machine	Network Interface	IP Address/Mask
router1	eth1	10.0.0.1/24
router2	eth1	10.0.0.2/24
router1	eth2	192.168.1.1
router2	eth2	192.168.2.1
host1	eth1	192.168.1.254
host2	eth1	192.168.2.254

Tabla 6: Datos de configuración para el ejercicio 2

A continuación se va a realizar la configuración mínima necesaria para posibilitar la comunicación entre los hosts de los departamentos de marketing y ventas. Deliberadamente, el host de producción no será accesible (no tendrá ruta) desde los otros hosts.

f. Configure el host1 para que éste envíe los datagramas destinados a la red 192.168.2.0/24 a través del router1.

g. Capturando en tap1 y enviando un ping desde el host1 al host2, compruebe que los datagramas IP que contienen el mensaje ICMP echo-request llegan al router1. Comente los campos principales de nivel de enlace y red en los que va encapsulado el mensaje ICMP (direcciones MAC, direcciones IP, etc.)

h. Compruebe que el router1 tiene activado el forwarding.

i. Añada la entrada necesaria en la tabla de rutas del router1 para alcanzar la red de ventas. Envíe nuevamente el ping desde el host1 al host2. Abra tres analizadores de protocolos para capturar el trafico en tap0, tap1 y tap2. Comente nuevamente qué direcciones IP observa, qué direcciones MAC, cómo se decrementa el campo TTL, etc. Analice las cache ARP de las diferentes máquinas virtuales.

j. Acabe la configuración en el otro sentido para que el host1 pueda recibir el echo-replay del host2. Compruebe que el ping ahora funciona correctamente.

k. Envíe un ping desde el router1 al host2. Averigüe porqué no funciona y proponga dos posibles formas de cambiar la tabla de rutas en los hosts para arreglar el problema.

Como habrá podido observar, la configuración del backbone realizada no permite la comunicación de ningún host de marketing o ventas con el host de producción. Esta es la configuración impuesta y deseada por los administradores de la red backbone. Sin embargo, como va a poder comprobar, un usuario avanzado podría bajo determinadas condiciones enviar y recibir mensajes ICMP. Para ilustrar esto, vamos a ver varios comandos ping que ejecutados en el host1 permiten obtener respuesta del host3.

l. Ejecute el siguiente comando y deduzca con todo detalle cómo funciona.

```
host1$ ping -r 192.168.1.1 10.0.0.3 192.168.3.254
```

m. Ejecute el siguiente comando y deduzca con todo detalle cómo funciona.

```
host1$ ping 10.0.0.3 192.168.3.254
```

n. Ejecute el siguiente comando y deduzca con todo detalle cómo funciona.

```
host1$ ping -r 10.0.0.3 192.168.3.254
```

Ejercicio 3

El objetivo de este ejercicio es ver el proceso de fragmentación de datagramas IP y el funcionamiento de diferentes mensajes ICMP ante diferentes condiciones de error. La red que se utilizará para hacer las pruebas está representada en la figura 16.

Para arrancar la simulación ejecutar el siguiente comando:

```
host$ simctl icmp start
```

Esta figura muestra tres redes (*Net0*, *Net1* y *Net2*), conectadas mediante tres máquinas virtuales que actúan como routers (*router1*, *router2* y *router3*). La *MTU* de las redes es diferente. Las interfaces de red de cada máquina son *ethernet*, con una MTU máxima de 1500 bytes, pero con el comando `ifconfig` podemos acortar esta MTU (ver la entrada al manual de este comando). Con esta acción, informamos a la capa IP de que el sistema físico puede transportar, como máximo, MTU bytes de datos. De esta manera se obliga al nivel IP a entregar paquetes de un tamaño máximo de MTU bytes al nivel físico.

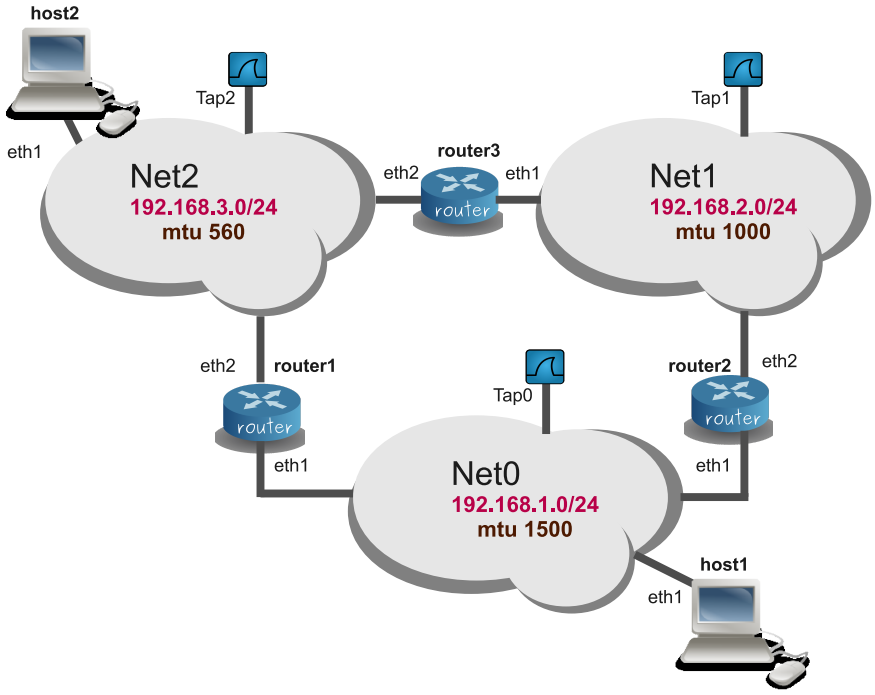


Figura 16: Red de pruebas del ejercicio 3 .

En primer lugar, se deben configurar cada una de las máquinas de la red. Los routers deben configurarse según las siguientes tablas:

router1	IP	MTU
eth1	192.168.1.1/24	1500
eth2	192.168.3.2/24	560

router2	IP	MTU
eth1	192.168.1.2/24	1500
eth2	192.168.2.1/24	1000

router3	IP	MTU
eth1	192.168.2.2/24	1000
eth2	192.168.3.1/24	560

Las máquinas **host1** y **host2** se configuran de la siguiente manera:

host1	IP	MTU
eth1	192.168.1.3/24	1500

host2	IP	MTU
eth1	192.168.3.3/24	560

Finalmente, se establecerán las rutas de entrega indirecta a cada una de las máquinas y routers:

- El router **router1** tiene la ruta por defecto a través de **router2**

- La máquina **host1** tiene la ruta por defecto a través de **router2**
- El router **router2** tiene la ruta por defecto a través de **router3**
- El router **router3** tiene la ruta por defecto a través de **router1**
- La máquina **host2** tiene la ruta por defecto a través de **router1**

a-. Fragmentación en origen

a. En primer lugar y utilizando la información de configuración de las máquinas y routers, determine cuál es el camino que seguirá un paquete que salga desde **host1** con destino **host2**, indicando las redes y los routers que atravesará.

b. En segundo lugar, determine cuál es el camino que seguirá un paquete que salga desde **host2** con destino **host1**, indicando las redes y los routers que atravesará.

c. corrobore las respuestas de los puntos anteriores haciendo un "`ping` " desde **host1** hasta **host2**, y capturando el tráfico en los dispositivos *tap0*, *tap1* y *tap2*. En **host1** ejecute el comando:
`ping -c 1 192.168.3.3`

   - Determine el tamaño de los paquetes IP de *echo-request* y del *echo-reply*. ¿Ha sido necesario fragmentar el paquete ICMP en algún punto del camino entre **host1** y **host2**?.
   - Compruebe el valor del *flag* DF que se encuentra en la cabecera IP. ¿Para qué sirve este *flag*?

d. Ahora, se debe enviar un mensaje *echo-request* con 900 bytes de información desde **host1** hasta **host2**, capturando el tráfico en los tres dispositivos *tap*. Es importante que esta prueba se haga siempre con las mismas condiciones para que el resultado siempre sea el mismo, por esta razón debe ejecutar en **host1** el comando:
`ip route flush cache`
Una vez borrada la memoria temporal de encaminamiento de la máquina **host1**, podemos pasar a hacer las pruebas. Ejecute en **host1** el comando:
`ping -c2 -s 900 192.168.3.3`
Analizando las capturas realizadas en los dispositivos *tap* responda a las siguientes preguntas:

   - ¿Qué tamaño tiene el primer paquete ICMP capturado en *tap0*? Averigüe las medidas de cada una de las cabeceras de los protocolos que hay en este paquete *ethernet*. Identifique dónde viajan los 900 bytes que hemos indicado como argumento del comando.
   - Consultando las capturas realizadas en los otros dispositivos *tap*, determine el trayecto de este paquete en las otras redes, identificando que la medida sea la misma.
   - Analice el paquete ICMP "*Destination unreachable*". Este mensaje ICMP nos está diciendo que el destino no es alcanzable, pero ¿por qué? (Analice la cabecera ICMP de este mensaje). ¿Cuál es el paquete IP que ha provocado el error y la transmisión de este mensaje ICMP? ¿Quién es el remitente de este mensaje?. ¿Quién es el destinatario?. ¿Qué camino ha seguido este mensaje ICMP desde el origen hasta el destino?.
   - Se ha visto que el primer mensaje ICMP de tipo *echo-request* de 900 bytes no ha llegado al destino y, por lo tanto, no habrá respuesta *echo-reply*. Analice la captura del dispositivo *tap*, el segundo mensaje ICMP *echo-request*. Vea el tamaño de este paquete *ethernet* e identifique el tamaño de cada una de las cabeceras que hay dentro del paquete. Vea el valor de los *flags* "Don't Fragment" (DF) y "More Fragments" (MF) de la cabecera IP de este mensaje. ¿Para qué sirve el *flag* MF?. Anote los valores del campo de la cabecera IP " identification" y "fragment offset".
     Analice el siguiente paquete ethernet que transporte un datagrama IP con un valor del campo "identification" de la cabecera igual al del paquete anterior. Analice en la cabecera IP el *flag* MF y los campos "identification" y "fragment offset". ¿Cuál es el tamaño de este paquete? Intente correlar toda esta información con el hecho de que ha ejecutado un comando para enviar un paquete ICMP *echo-request* con 900 bytes de información y que hay un segmento de la red con una MTU de 560 bytes.

- Identifique el camino que sigue el mensaje ICMP *echo-request* fragmentado desde el origen hasta el destino y también el mensaje de respuesta ICMP *echo-reply* desde el origen hasta el destino.

e. ¿Qué máquina ha realizado la fragmentación del paquete?

b-. Fragmentación en tránsito

En este caso, se debe enviar un mensaje *echo-request* desde **host1** hasta **host2**, de tamaño 900 bytes (de información ICMP) con el *flag* DF a 0 (consultar en la página de manual de `ping` la opción -M). Se debe capturar el tráfico por las tres interfaces *tap0 .. 3*. Analizando el tráfico capturado debe averiguar dónde se está produciendo la fragmentación.

¿Qué es lo que pasa si se envía un mensaje *echo-request* desde **host1** hasta **host2**, de tamaño 1200 bytes (de información ICMP) con el *flag* DF a 0?

c-. El mensaje ICMP *Time To Live exceeded*

El objetivo de esta prueba es generar la condición de error que provoca la transmisión de un mensaje ICMP *Time To Live exceeded*. Cuando un datagrama IP llega a un *router* para ser encaminado hacia su destino, algunos campos de la cabecera IP del datagrama IP son modificados por el router:

- El campo TTL Time To Live se decrementa en una unidad.
- El *router* debe calcular el nuevo valor del "checksum" de la cabecera IP, ya que ha cambiado el TTL.

Si el valor del campo TTL calculado es cero, el *router* destruye el paquete y se transmite un mensaje ICMP de tipo *Time To Live exceeded* hacia el remitente del datagrama IP que ha generado el error.

Para comprobar de forma práctica el funcionamiento descrito, enviaremos un mensaje ICMP *echo-request* con un valor del campo TTL de la cabecera IP de 8, desde **host1** a una IP destino 10.0.0.1. Véase en la página de manual de `ping` la opción -t. Antes de iniciar las pruebas conteste las siguientes preguntas:

a. Teniendo en cuenta la configuración de los routers router1, router2 y router3 y de host1, si se envía un datagrama IP desde **host1** con destino 10.0.0.1, qué camino seguirá este datagrama?

b. Si el valor del campo TTL del datagrama inicial es 8, ¿ en qué *router* se producirá la condición de error?

c. Si el *router* que produce la condición de error, envía el mensaje ICMP *Time To Live exceeded* hacia `host1` , ¿por qué camino irá? ¿cuál será la IP de origen de este datagrama?

Ejecute el `ping` desde **host1** y verifique de forma práctica las respuestas que ha dado, capturando el tráfico en las interfaces *tap0 ... 3* y analizando las capturas.

Anexo: Manual del analizador de protocolos *Wireshark*

El analizador de tráfico/protocolos disponible en el laboratorio para esta práctica es el *Wireshark* (anteriormente conocido como *Ethereal*).

Es un analizador de protocols "software" sin hardware *ad-hoc*, es decir que utiliza como plataforma de hardware (captura de bits) y software (procesado y visualización) un equipo no específicamente diseñado, por ejemplo un PC con un una tarjeta de red estándar y sistema operativo común: *Windows*, *Linux*, etc. (existen equipos más específicos, normalmente portátiles, enteramente diseñados por sus fabricantes para actuar como analizadores de protocolos exclusivamente; en algunos casos ofrecen alguna prestación superior, única; eso si, a precios mucho más elevados)

El software *Wireshark* es de licencia general pública (GNU), disponible para varias plataformas hardware y sistemas operativos. Analiza una gran cantidad de familias (*stacks*) de protocolos (no sólo Ethernet, TCP/IP), es potente y de notable calidad.

El software *Wireshark* configura una tarjeta de red Ethernet convencional, por ejemplo, en un modo especial, llamado *promiscuo,* de manera que capture todas las tramas que circulen en el medio físico (no sólo las dirigidas a la dirección MAC de la estación y la de *broadcast* como sería el modo de operación normal).

El analizador de protocolos (a veces también llamado *datascopio*) es para el ingeniero en telemática como el *osciloscopio* para el ingeniero en electrónica; es decir, un instrumento de elevada impedancia de entrada (no intrusivo) que, insertado en cierto punto de una red (circuito), nos explica con todo detalle, los datos (la señal); lo que allí está sucediendo. Algunos analizadores de protocolos también permiten realizar medidas activas (intrusivas) programables. (no es el caso de *Wireshark*)

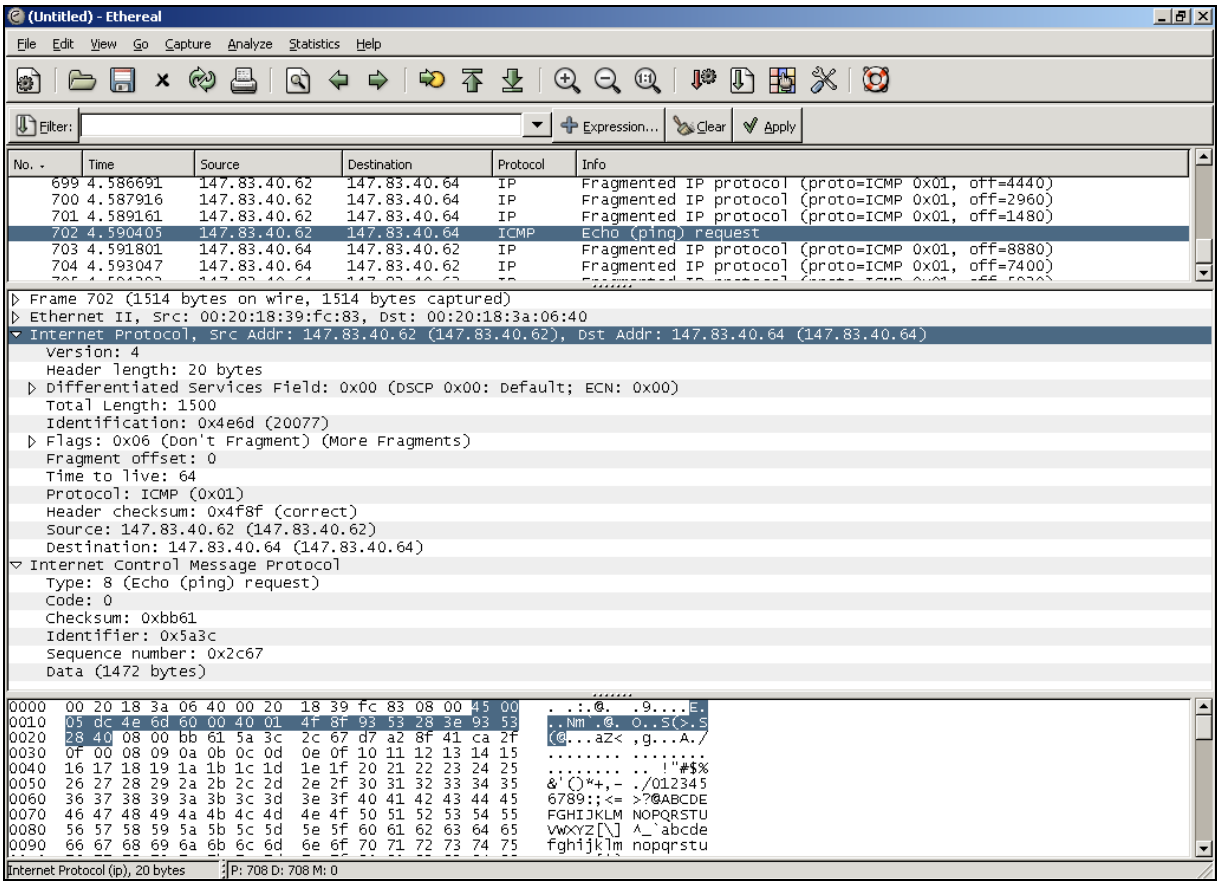
Los tipos de análisis/medidas que permite *Wireshark* pueden clasificarse en 2 grandes grupos:

– Medidas estadísticas: es decir, medidas agregadas (macroscópicas) en base a muchas tramas observadas. Ejemplos serían: estadísticas de longitud de las tramas (porcentaje de tramas cortas, medias, largas; longitud media de trama); tramas por segundo totales, bits por segundo totales; porcentajes de ocupación de la red para distintos tipos de mensajes: de control, por protocolo, por tipo de aplicación, etc; porcentajes de ocupación por orígenes/destinos (las llamadas *conversaciones*): bits por segundo, tramas por segundo, de A a B, de B a A, de C a D, de D a C, etc; análisis inteligentes de conexiones de transporte (evolución de los números de secuencia y del retardo *RTT*: *Round Trip Time* y las pérdidas), del tráfico de navegación Web, de los flujos (*streams*) de una videoconferencia, etc, etc.

– Decodificación de protocolos: es decir, análisis detallado y explicativo de cada paquete que viajaba por la red en el punto donde el analizador de protocolos estuvo capturando. *Wireshark* contiene una inmensa base de datos, para la mayoría de arquitecturas de comunicaciones, que le permite descifrar y presentar de forma clara e inteligible, para nosotros los humanos, desde los niveles más bajos (*enlace de datos*) hasta los superiores (*aplicación*), cualquier mensaje.

La interfaz gráfica de *Wireshark* es relativamente sencilla. Dispone de una barra de menús desplegables, una barra de iconos y una ventana principal subdividida en 3 bandas horizontales (personalizables) donde se muestran: el listado de los paquetes capturados de forma resumida (superior), la decodificación detallada del paquete seleccionado actualmente (centro) y la visualización en hexadecimal y ASCII de dicho paquete (inferior).

A continuación puede verse un ejemplo:



En las 8 opciones desplegables del menú se pueden configurar y ejecutar las funciones del analizador:

1. *File*: Abrir/guardar ficheros, medidas, imprimir, etc.
2. *Edit*: Explorador de paquetes, marcas y preferencias.
3. *View*: Configuración del entorno gráfico.
4. *Go*: Moverse por los paquetes capturados (siguiente, anterior, primero, último, por número de orden)
5. *Capture*: Inicia/detiene la captura de paquetes; especifica los interfaces y filtros con que se lleva a cabo.
6. *Analyze:* Define los filtros de pantalla y las opciones de decodificación.
7. *Statistics*: Estadísticas (resumen, generales, jerarquías de protocolos, conversaciones, más específicas, gráficas, etc.)
8. *Help*: Acceso a la ayuda del programa, protocolos soportados e información sobre la versión.

Alternativamente, toda la funcionalidad del analizador se ofrece en los iconos que aparecen debajo del menú:

 Inicia una captura (con los filtros de captura definidos, si es el caso).

 Abre un archivo donde se haya almacenado una captura previa.

 Guarda la captura actual en un archivo.

 Cierra el archivo de captura actual.

 Vuelve a cargar el archivo de captura entero.

 Imprime, en puerto de impresora o archivo, la captura o selección actual.

 Busca un paquete con los parámetros dados dentro de la captura actual.

 y  Buscan el paquete inmediatamente anterior o posterior, respectivamente, que cumple los parámetros de búsqueda dentro de la captura actual.

 Lleva al paquete del cual se especifica su número de orden.

 y  Llevan al primer y último paquete, respectivamente.

 y  Amplían y reducen el tamaño de la fuente de la vista actual, respectivamente.

 Restaura la vista a la escala por defecto.

 Define los filtros de captura.

 Define los filtros de pantalla.

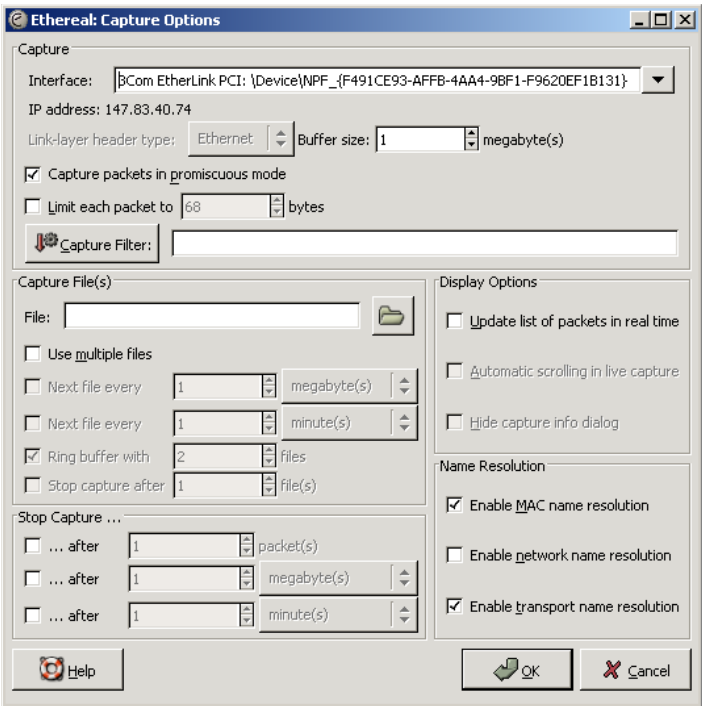
 Permite cambiar la configuración de color actual de las ventanas.

 Accede a la ventana de preferencias.

 Ayuda.

Captura de paquetes:

Al pulsar el icono  se accede a la ventana de opciones de captura (según la siguiente imagen):



Debe esta activado "*capture packets in promiscuous mode*". El "*buffer size*" lo podemos poner a 64 MB o quizá más (usualmente 64 MB es el máximo para la mayoría de máquinas; en todo caso el programa lo ajustará).

La opcción "*Limit each packet to*" es útil si deseamos capturar muchos paquetes sin desbordar el *buffer* de captura. Por ej., si lo ponemos a 100 sólo se almacenarán los 100 primeros bytes de cada paquete; suficiente para almacenar todas las cabeceras hasta la de transporte y una parte, quizás toda, la de aplicación. (nos perdemos el resto: quizás parte de la cabecera de aplicación y los datos, para paquetes grandes)

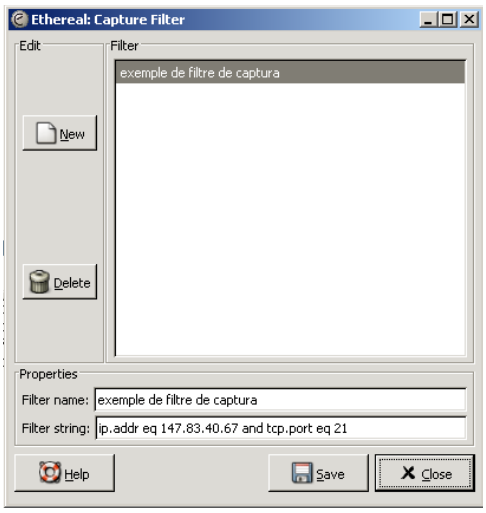
La opción "*Display Options > Udpate list of packets in real time*", puede ser conveniente el desactivarla cuando el tráfico a capturar en la red es extremadamente elevado; ya que *Wireshark*, al ser un analizador "software", podría gastar demasiada CPU en la visualización por pantalla y "perder", por consiguiente, algunos paquetes en la captura.

Filtros de captura y de visualización:

El sistema de filtrado de *Wireshark* es a 2 niveles. De captura y de visualización.

Los filtros de captura deben introducirse en las ventanas de opciones de captura, o con el icono  , y, como puede entenderse, actúan como "pre-filtros", con pérdida definitiva de lo filtrado, sin vuelta atrás; es decir, que actúan eliminando definitivamente una parte de lo que el analizador posteriormente analizará/visualizará. Además, para complicar un poco las cosas, la sintaxis de dichos

filtros es distinta y algo más limitada (ver ayuda) a la de los filtros de visualización (bastante más rica y potente).



Por tanto, excepto para usos muy específicos y avanzados, dejaremos de lado los filtros de captura y nos centraremos en los de visualización, más flexibles, que nos permiten ponerlos y quitarlos en cualquier momento, sin perder nunca los datos capturados.

Los filtros tienen una sintaxis a base de *palabras clave,* compuestas de uno o más nombres enlazados por un punto (derivados de la familia de protocolos que se analiza/filtra), y unos *operadores relacionales/lógicos* (al estilo del lenguaje "C"). A primera vista puede parecer que la creación de un filtro sea muy engorrosa; pero este método también le dota de una extraordinaria flexibilidad y potencia.

Algunos ejemplos de *operadores relacionales/lógicos* para unir *palabras clave* y crear sentencias:

`not` o `!=`

`eq` o `==`

`>, >=, <, =<`

`&&` o `and`

`||` o `or`

`(  )`

Algunos ejemplos de *palabras clave*:

`eth.src`: dirección MAC origen

`eth.dst`: dirección MAC destino

`eth.addr`: dir MAC origen o destino

`eth.type`: campo "tipo" en trama Ethernet; existen valores para identificar por ej.: IP, ARP, PAUSE (flow control IEEE 802.3X), VLAN tag (virtual LAN IEEE 802.1Q)

`ip.src`: dir IP fuente

`ip.dst`: dirección IP destino

`ip.addr`: direccion IP origen o destino (indistintamente)

`tcp.srcport`: puerto TCP origen

`tcp.dstport`: puerto TCP destino

`udp`: contiene mensaje del protocolo UDP

`udp.port`: puerto UDP origen o destino (indistintamente)

`tcp.port`: puerto TCP origen o destino (indistintamente)

`icmp`: contiene un mensaje del protocolo ICMP (per ej.: *Echo Request/Response,* del comando *ping*)

`arp`: trama Ethernet de "tipo" *ARP (broadcast Request, Reply)*

`http`: contiene mensaje de aplicación con protocolo HTTP (Web)

`telnet`: contiene mensaje de aplicación de acceso remoto en modo terminal con protocolo TELNET

La combinación de palabras clave y operadores relacionales da lugar a las *filter strings* (*cadenas filtro*). Algunos ejemplos nos mostrarán que el uso de los filtros no es tan complicado como parece y sí, en cambio, muy útil y potente para separar/aislar el tráfico que nos interesa de entre los miles de paquetes que suele capturar el analizador. Veamos algunos ejemplos:

`eth.dst==56:0A:FC:88:04:6B`: Nos visualiza sólo las tramas cuya dirección destino es la introducida en el filtro. Además si, en ciertos menús de estadísticas, marcamos la opción "*limit to display filte*r", ésas solamente se calcularán en base a los paquetes así "filtrados".

`arp`: Sólo nos muestra las tramas del protocolo ARP (resolución de direcciones IP-MAC); el resto de tráfico permanecerá oculto.

`icmp`: Sólo nos muestra las tramas que encapsulan mensajes del protocolo ICMP (*Internet Message Control Protocol*). Mensajes de diagnósticos de error de red de los hosts/routers o de los "ecos" solicitados con el comando *ping*, por ejemplo.

`ip.addr==147.83.40.105 && http`: Este filtro, por ejemplo, nos muestra solamente las tramas que encapsulan datagramas IP con dirección origen o destino 147.83.40.105 (indistintamente) "y" que encapsulan el protocolo de aplicación HTTP; es decir, el filtro nos muestra sólo el tráfico de navegación Web de la estación *huffman5.upc.es* del laboratorio.

`icmp and (ip.addr eq 10.0.1.108)`: Este filtro nos muestra básicamente los paquetes de *ping* enviados/recibidos a/de la dirección 10.0.1.108 (2a tarjeta de red del host *huffman8*; para ello habremos estado monitorizando, en modo promiscuo, con *Wireshark*, por la 2a tarjeta (*eth1*, no *eth0,* de Linux) de algún otro equipo *huffmanx* del laboratorio)

`tcp.port ==23`: El puerto 23 TCP pertenece al conjunto de puertos "bien conocidos" (*well-known ports*) y está reservado para los servidores de conexiones entrantes TELNET. Esta cadena filtra todas las tramas que encapsulan segmentos de transporte con origen o destino el puerto 23 del

protocolo TCP (conexiones de terminal remoto con un sistema, según el protocolo TELNET, independientemente de las direcciones IP destino u orígen).

`telnet`: Este filtro, parecido al anterior, sólo muestra los mensajes de nivel de aplicación; el anterior mostraba, además, todos los mensajes de transporte (hacia/desde el puerto 23), incluyendo los segmentos *ACK*s. La diferencia es sutil. Pero esto cae fuera del ámbito de esta práctica.

`not icmp`: Este filtro muestra todas las tramas que "no" encapsulan un mensaje del protocolo ICMP. Puede ser útil para poner de manifiesto el tráfico "de fondo" cuando estás "inundando" mediante el comando *ping* algún otro puerto durante los experimentos de la práctica.

Wireshark nos permite un modo "asistido" de edición de las *cadenas filtro* (mostrándonos todas las posibles palabras clave del "lenguaje", que son muchas, al ser muchos los protocolos y campos de sus cabeceras); debe notarse que las *cadenas filtro* son "*case-sensitive*" (sensibles a mayúsculas-minúsculas).

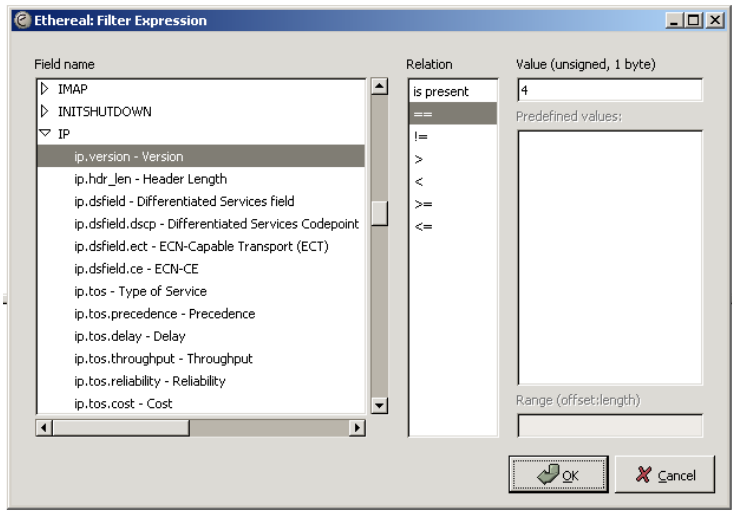
Una facilidad incorporada es que nos indica si una *cadena filtro* es sintácticamente válida o no, mostrándonos un fondo de color verde, en caso afirmativo, o color rojo como fondo para una cadena errónea o incompleta.

Los filtros de pantalla (o visualización), una vez realizada la captura, se aplican directamente escribiéndolos en la barra de filtrado, o pulsando sobre el icono  . Deberá tenerse en cuenta el color del fondo (verde o rojo) de la cadena tecleada para saber si el filtro es sintácticamente válido o no. Se dispone de un glosario de palabras de filtrado pulsando el botón  , al lado de la barra de filtrado. Para aplicar finalmente el filtro a los datos capturados que se tienen en pantalla, debe pulsarse el botón *Apply*. Si, por el contrario, deseamos retirar el filtro, pulsaremos en *Clear*.

A continuación tenemos un ejemplo de un mismo filtro de visualización (*filter string*) definido de las dos formas posibles:

Manualmente:

Usando el asistente del glosario (*Expression*) para definir la *cadena filtro (filter string)*:

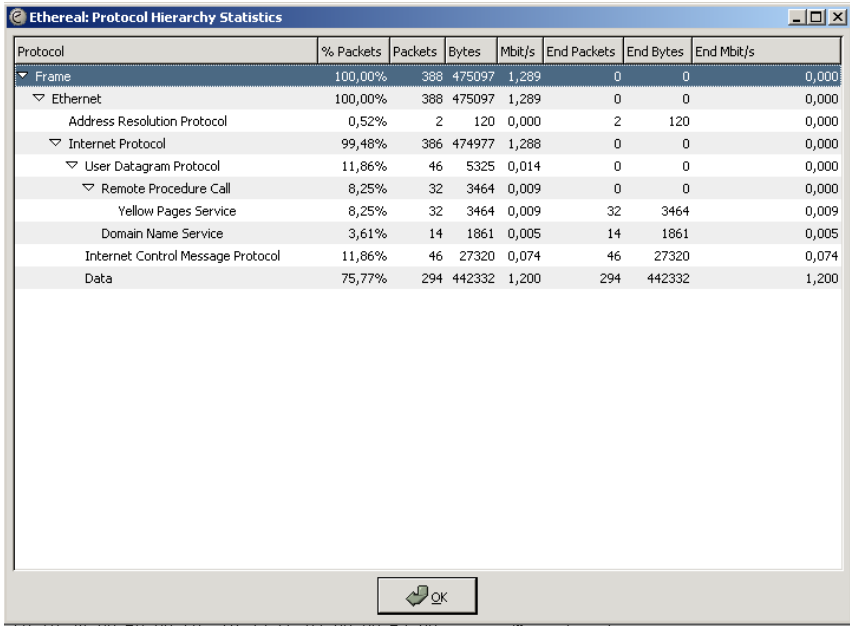


Medidas estadísticas:

Wireshark quizás no dispone de una interfaz gráfica espectacular de cara a la presentación numérica de resultados estadísticos pero si que los calcula y el usuario los tiene disponibles en forma de tablas.

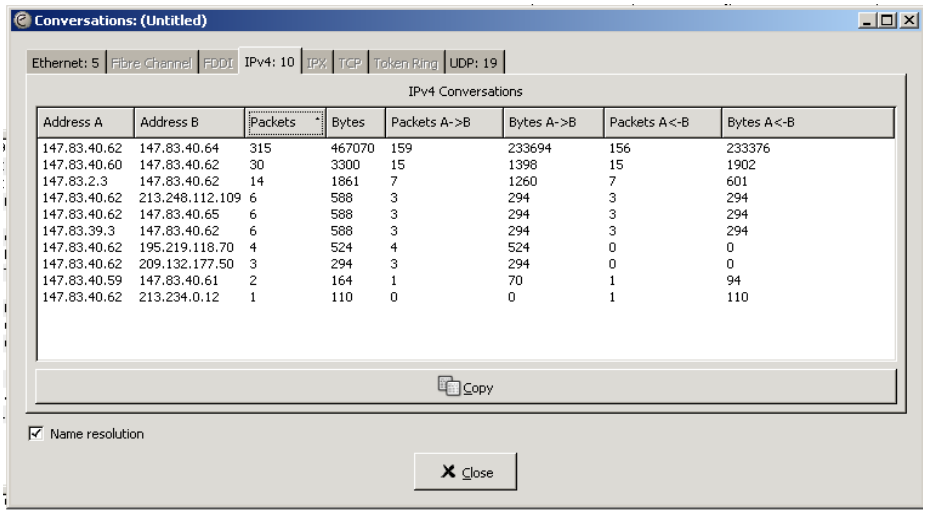
Summary: Consiste en un resumen estadístico muy, muy breve de los datos capturados (paquetes totales, tiempo total de captura, filtros aplicados, paquetes "perdidos" por el analizador, tasas promedio en paquetes, bytes (y Mbit) por seg., y poco más). La siguiente figura muestra un ejemplo:

Protocol Hierarchy: Genera un árbol jerárquico de los protocolos que se tienen capturados en pantalla. Proporciona información sobre los encapsulamientos de las tramas, porcentajes de paquetes para cada protocolo, número de paquetes, bytes y Mbytes/s por protocolo, número de paquetes acabados (*End Packets*) en un determinado protocolo (útil para saber en qué niveles se están transmitiendo paquetes, etc).



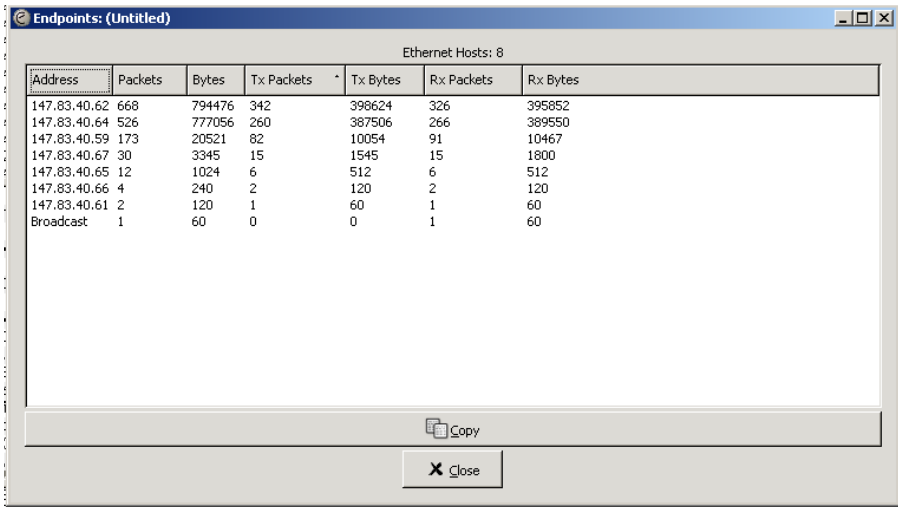
Protocol	% Packets	Packets	Bytes	Mbit/s	End Packets	End Bytes	End Mbit/s
Frame	100,00%	388	475097	1,289	0	0	0,000
Ethernet	100,00%	388	475097	1,289	0	0	0,000
Address Resolution Protocol	0,52%	2	120	0,000	2	120	0,000
Internet Protocol	99,48%	386	474977	1,288	0	0	0,000
User Datagram Protocol	11,86%	46	5325	0,014	0	0	0,000
Remote Procedure Call	8,25%	32	3464	0,009	0	0	0,000
Yellow Pages Service	8,25%	32	3464	0,009	32	3464	0,009
Domain Name Service	3,61%	14	1861	0,005	14	1861	0,005
Internet Control Message Protocol	11,86%	46	27320	0,074	46	27320	0,074
Data	75,77%	294	442332	1,200	294	442332	1,200

Conversations: Genera un "mapeo" de las conversaciones entre máquinas de la red, es decir, sobre los intercambios de datos, dos a dos, y por protocolos/capas (MAC, IP, Transporte, etc.). Aporta información sobre los paquetes/bytes totales transmitidos, y desglosados en cada dirección. La siguiente figura muestra las conversaciones a nivel de red (dir. IP), por ejemplo:



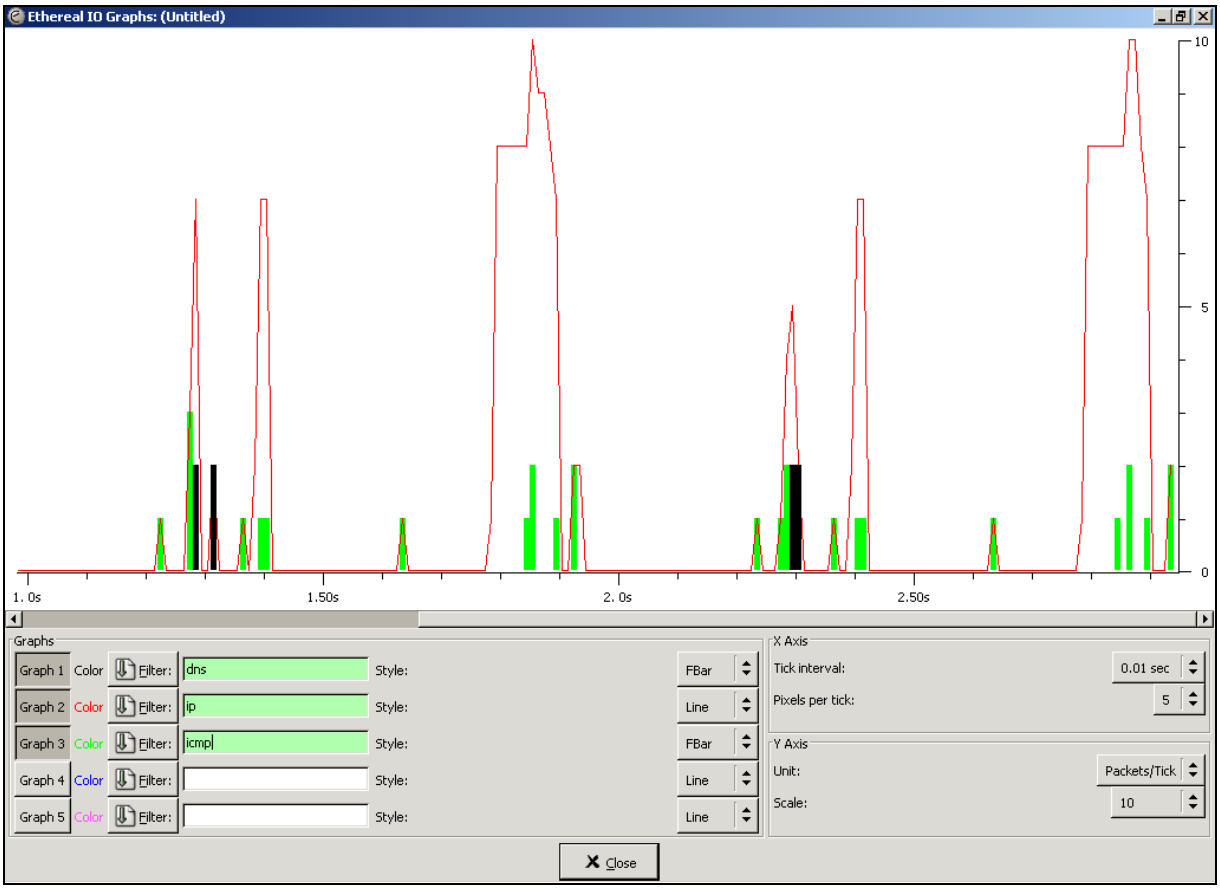
Address A	Address B	Packets	Bytes	Packets A->B	Bytes A->B	Packets A<-B	Bytes A<-B
147.83.40.62	147.83.40.64	315	467070	159	233694	156	233376
147.83.40.60	147.83.40.62	30	3300	15	1398	15	1902
147.83.2.3	147.83.40.62	14	1861	7	1260	7	601
147.83.40.62	213.248.112.109	6	588	3	294	3	294
147.83.40.62	147.83.40.65	6	588	3	294	3	294
147.83.39.3	147.83.40.62	6	588	3	294	3	294
147.83.40.62	195.219.118.70	4	524	4	524	0	0
147.83.40.62	209.132.177.50	3	294	3	294	0	0
147.83.40.59	147.83.40.61	2	164	1	70	1	94
147.83.40.62	213.234.0.12	1	110	0	0	1	110

Endpoints: Esta opción estadística nos muestra el tráfico que genera/recibe cada nodo de la red. (en paquetes/bytes)



IO Graphs: Wireshark permite generar hasta 5 gráficas (en tiempo real, o diferido) relativas al tráfico capturado de la red y superponerlas aplicando filtros de visualización independientes. Se pueden elegir las escalas de los ejes verticales (bytes/s, paquetes/s, lineal, logarítmico, ...) y horizontales (resolución segs/pixel). El color no se puede cambiar. La *cadena filtro* se escribe directamente para cada gráfica (su fondo en verde nos indicará que es sintácticamente correcta). Las gráficas se pueden añadir o quitar pulsando los botones "*Graph 1*"…"*Graph 5*". El estilo del trazado de las gráficas también se puede personalizar (linea, puntos, barras, …)

La figura siguiente nos muestra un ejemplo de todo esto:



Decodificación de protocolos:

La faceta de decodificación de protocolos de *Wireshark* es bastante intuitiva, autoexplicativa.

En la ventana superior tenemos una linea por cada trama captura donde se indica su número de orden, su tiempo de llegada (con precisión del orden de los microsegs.), sus direcciones de origen y destino (de nivel de red) y un resumen informativo, que reúne varios niveles arquitectónicos, del paquete en cuestión.

Cuando seleccionamos una trama en dicha ventana superior, tenemos su análisis detallado/explicado en la ventana central.

El nivel de detalle en la ventana central, para cada nivel de la arquitectura de protocolos, aún puede ampliarse haciendo *clic* en los pequeños triángulos a la derecha: que apuntan horizontalmente, si aún queda información por mostrar; o hacia abajo, si se han desplegado todos los detalles.

Por cada campo de información sobre el que nos situemos en la ventana central, tenemos automáticamente marcados, en la ventana inferior, los bytes (en hexadecimal y ASCII) que lo codifican en la trama.

Wireshark posee, además, muchas funciones inteligentes, como: relacionarnos datagramas IP entre si, cuando éstos son fragmentos de un datagrama IP original grande; mostrarnos números de secuencia "relativos" al inicio de una conexión TCP (en vez de absolutos, mucho más engorrosos), mostrar los datos (en ASCII), en ambos sentidos, de una conexión TCP simplemente "pinchando" un paquete cualquiera de dicha conexión y seleccionando la opción "*Follow TCP stream*", etc, etc. Pero todo esto va más allá del ámbito de esta práctica...