

# Prácticas de Laboratorio de Redes de Computadores

Natalia Ayuso Escuer · Juan Segarra Flor · Jesús Alastruey Benedé



**Departamento de  
Informática e Ingeniería  
de Sistemas**  
**Universidad Zaragoza**

Curso 2018-19





# Índice general



# Práctica 1

## Análisis de tráfico y encapsulación de protocolos

### 1.1. Objetivos

Aprendizaje básico de la arquitectura de red y la encapsulación de protocolos usando el analizador de tráfico *Wireshark*. Analizar la topología básica de una red. Comprender las diferencias entre conexión punto-a-punto y extremo-a-extremo. Uso básico del manual del sistema.

### 1.2. Entorno de trabajo

Las prácticas de la asignatura se realizarán en el sistema *CentOS* (GNU/Linux) de los equipos del laboratorio L1.02, al que se accede con el nombre de usuario y contraseña de *Hendrix*. En cualquier momento puedes cambiar la contraseña desde <https://diis.unizar.es/WebEstudiantes/>. Aparte de los equipos de dicho laboratorio, dispones del equipo *lab000.cps.unizar.es*, al que se puede acceder de forma remota, que es equivalente a los equipos de los laboratorios. Puedes usarlo para completar las prácticas fuera de horarios o para realizarlas de forma no presencial.

### 1.3. Análisis de tráfico

Para analizar el tráfico de red se dispone del analizador *Wireshark*. Se puede lanzar bien desde el menú o bien desde la línea de comandos. A continuación el sistema pedirá la contraseña de administrador, pero está configurado para que al pulsar «Ejecutar sin privilegios» funcione como si lo ejecutara el administrador. Al iniciar el programa, dependiendo de la versión instalada, aparece una ventana similar a la de la Figura ???. En la parte central izquierda se puede seleccionar de un listado el interfaz de red por el que capturar el tráfico. En la práctica hay que usar «eth0» (interfaz de *ethernet*) o «any» (cualquier interfaz) para capturar el tráfico que nos interesa. Tanto la selección del interfaz como el resto de opciones de la pantalla de inicio se pueden cambiar posteriormente a través de los menús de la aplicación.

El analizador captura las tramas (*frames*) que circulan por el medio de transmisión, que en este caso es el cable de par trenzado, e interpreta su formato, en este caso formato Ethernet. Cada trama puede incluir distintos protocolos, y el analizador visualiza el contenido de los campos de sus cabeceras. Para ello, la pantalla del programa se divide en cuatro áreas de visualización (Figura ??):

1. Área de definición de filtros. En ella se pueden especificar filtros, de forma que el resto de las áreas sólo muestren entradas que coincidan con el filtro (e.g. tcp). El botón «Expression» facilita la creación de filtros mediante selección de elementos desplegables.
2. Área de visualización de tramas. En ella aparece el listado de tramas capturadas, con su información básica: número de trama capturada, instante de captura, direcciones origen y destino, protocolo de

## PRÁCTICA 1. ANÁLISIS DE TRÁFICO Y ENCAPSULACIÓN DE PROTOCOLOS

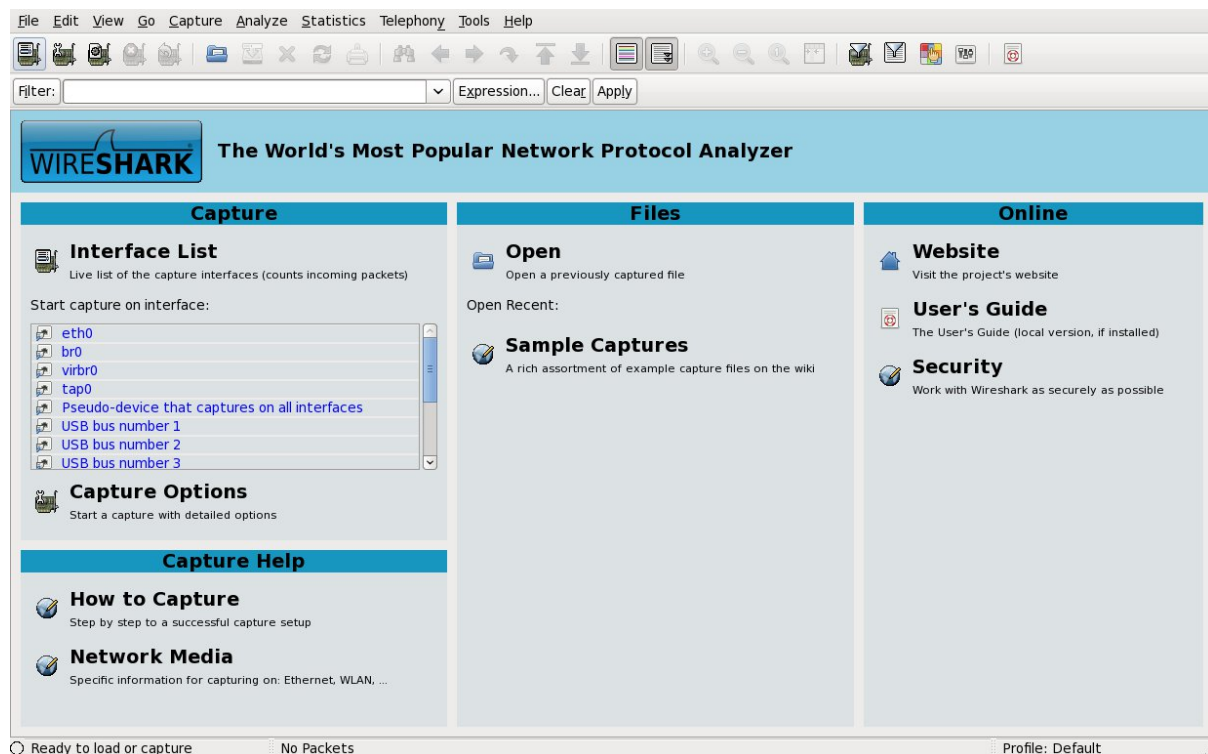


Figura 1.1: Pantalla de inicio del programa *Wireshark*

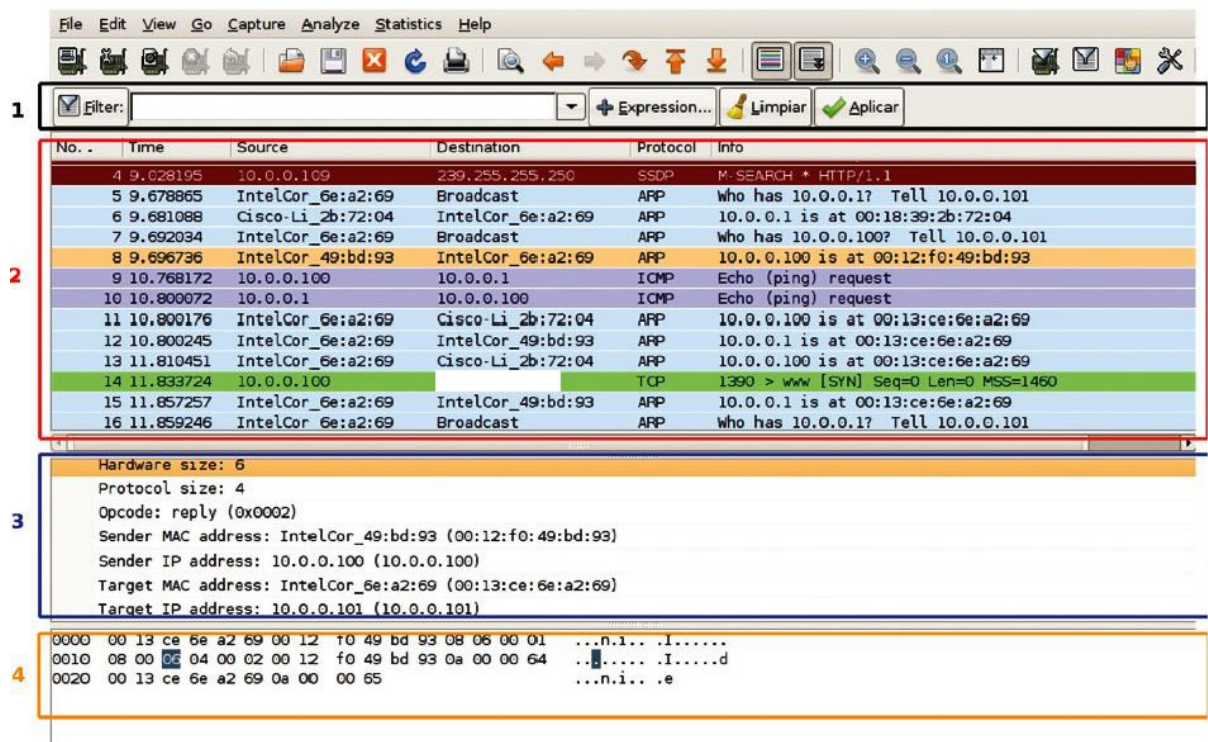


Figura 1.2: Áreas de visualización del programa *Wireshark*

la capa más alta en la trama, e información abreviada de su contenido.

3. Área de detalle de una trama. Seleccionando una de las tramas del área de visualización, aquí se muestra toda su información mediante detalles desplegados. Los detalles incluyen la información de los campos de cada protocolo, que se estudiarán más adelante en la asignatura.
4. Área de información en bruto. En esta área se muestra la trama (y texto para caracteres imprimibles), es decir, los unos y ceros que realmente forman la trama. Por ejemplo, los datos de aplicación transmitidos se puede visualizar en esta parte.

Además, en la zona superior se encuentran los menús y botones, desde los que se puede iniciar/parar una captura, guardarla, etc.

### 1.3.1. Pila de protocolos

En clase de teoría hemos visto las distintas capas en la arquitectura de red, que se implementan mediante distintos protocolos que proporcionan servicios específicos. Inicia una captura de tráfico mientras accedes a Internet con un navegador. Al seleccionar una trama, en la parte desplegable se pueden ver sus detalles. La lista ordenada de protocolos siguiendo la estructura de capas (*pila de protocolos*) que intervienen en la trama seleccionada se puede ver en la parte desplegable de trama, en la línea [Protocols in frame: ]. Aunque suele haber un protocolo por capa, ten en cuenta que no todas las comunicaciones necesitan usar todas las capas.

Busca tramas etiquetadas con los siguientes protocolos y anota qué pila de protocolos usan dichas tramas.

1. ARP (*Address Resolution Protocol*). Protocolo de apoyo a la capa de red para preguntar qué máquina tiene cierto identificador ethernet/MAC:
2. DNS (*Domain Name Service*). Protocolo de la capa de aplicación para preguntar por nombres de equipos:
3. HTTP (*HyperText Transfer Protocol*). Protocolo de la capa de aplicación de comunicación web:
4. STP (*Spanning-Tree Protocol*). Protocolo de la capa de enlace para detectar (y eliminar) bucles en una red local:
5. TCP (*Transmission Control Protocol*). Protocolo fiable de la capa de transporte:
6. ¿Qué otros protocolos aparecen en el listado?
7. Teniendo en cuenta el encapsulado en una trama con TCP y HTTP, ¿la parte TCP está dentro de los datos de HTTP, o es la parte HTTP la que está dentro de los datos de TCP?
8. Dibuja el esquema de encapsulado (zona de cabeceras y datos de protocolos para cada capa, como en la transparencia de *encapsulación* vista en clase, sin indicar los bytes) de una trama que contenga el protocolo HTTP.

Como habrás podido observar en la captura, existen muchos protocolos y no siempre se usa un protocolo concreto en una capa concreta de la arquitectura. Dispones de un listado con los protocolos de *sistema* más relevantes en el fichero<sup>1</sup> `/etc/protocols`. Observa que no aparecen los protocolos de la capa de aplicación (e.g. HTTP), ya que éstos se implementan en las aplicaciones y no en el sistema.

9. ¿Cuál es la descripción que aparece asociada al protocolo UDP en ese fichero?
10. ¿Cuál es la descripción que aparece asociada al protocolo ICMP en ese fichero?

---

<sup>1</sup>Puedes explorar el sistema de ficheros del ordenador desde el menú: *Places* → *Computer* → *Filesystem*.

### 1.3.2. Direcciones de red

El protocolo de la capa de red que sustenta Internet es el protocolo IP. La cabecera de este protocolo incluye varios campos donde se encuentran codificadas las direcciones de los equipos en Internet. Accede a varios sitios de Internet (preferentemente usando conexiones *http* sin cifrar, y no *https*) y responde a las siguientes preguntas.

11. Comparando la información de varias tramas, ¿cuál puedes deducir que es la dirección IP de tu equipo? ¿Coincide con la de la pegatina identificativa que «debería» llevar tu equipo? Anota la IP en la parte correspondiente a «mi Host» en la Figura ???. Anota también la IP de dos de los servidores Web consultados.
12. Pregunta a tus compañeros qué dirección IP tienen sus equipos. ¿Se parecen a la de tu equipo? ¿En qué se parece? Anota la IP de alguno de los equipos dispuestos en tu misma bancada en la parte correspondiente a «Host junto al mío» en la Figura ??.
13. Con la información anterior, ¿se podría deducir que equipos «cercaños» deben tener direcciones IP «cercañas»?

### 1.3.3. Identificadores de acceso al medio

Además de las direcciones «lógicas» anteriores, cada tarjeta de red también tiene un identificador «físico» (también llamado dirección MAC, *media access control*, o dirección *ethernet*). Este identificador se codifica en campos del protocolo *ethernet* (o del protocolo de acceso al medio que corresponda). El identificador MAC está asociado a la tarjeta de red, es único, viene establecido por el fabricante, y no es posible modificarlo.

14. Al mostrar los identificadores MAC, en algunos casos la parte de mayor peso del identificador se muestra como un nombre. ¿Qué puede indicar ese nombre? ¿Cómo puede saber ese nombre el analizador?
15. Conociendo la dirección IP de tu equipo y viendo el detalle de las tramas, ¿cuál es el identificador ethernet/MAC de tu equipo? ¿Coincide con la de la pegatina identificativa que «debería» llevar tu equipo?
16. Busca varias tramas ARP y observa los identificadores MAC *destino*. ¿Hay alguno que resulte especialmente curioso? ¿A qué crees que corresponde dicho identificador?
17. Haz una captura accediendo a distintos lugares de Internet y anota los identificadores MAC de las tramas correspondientes. ¿Se parecen? ¿Por qué (revisa la transparencia «Retransmisores» vista en teoría)? ¿A qué equipo corresponde el identificador MAC que no es el de tu propio equipo?
18. Compara el identificador MAC de tu equipo con el de la pregunta anterior. ¿Se parecen?
19. Con la información anterior, ¿se podría deducir que los identificadores MAC dependen del fabricante y no de la «cercaña» de los equipos?

### 1.3.4. Topología de red

La topología física y lógica del laboratorio de prácticas no es sencilla. Sin embargo, vamos a determinar si los equipos del laboratorio configuran una red en bus o estrella observando el tráfico capturado con *Wireshark*. Para ello:

20. Teniendo en cuenta que *Wireshark* captura *todo* lo que pasa por el cable de red, ¿en qué podrías fijarte para saber si la topología es en bus o estrella? Haz un experimento para comprobarlo.
21. ¿Es una topología en bus o en estrella?



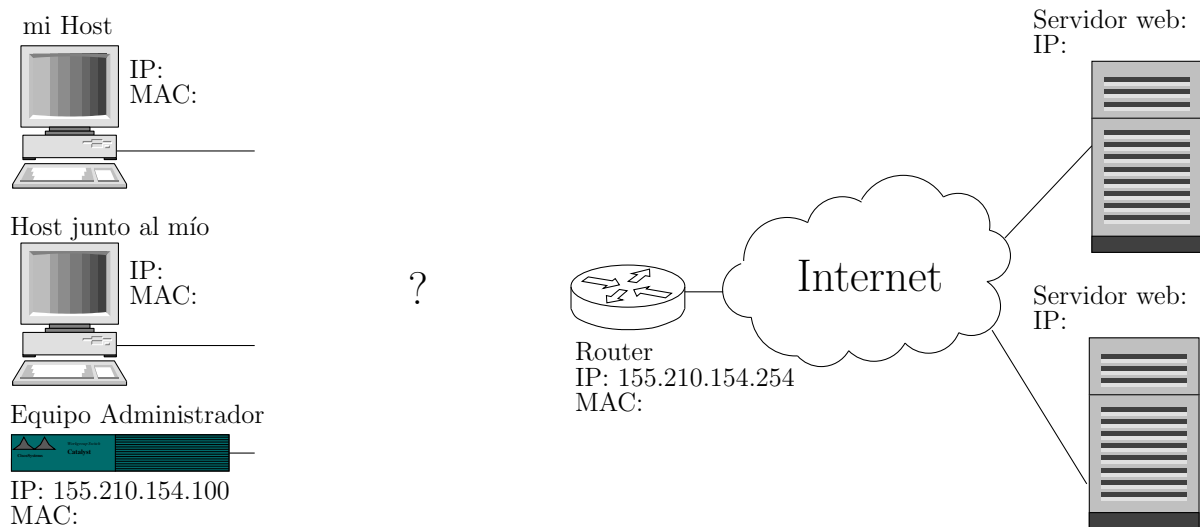


Figura 1.3: Acceso de los equipos de usuario a dos servidores web de Internet a través de la LAN del laboratorio de prácticas y equipo administrador. La conexión entre los host y el encaminador está por determinar.

Recuerda que existen dispositivos de interconexión *transparentes*, es decir, que no modifican las tramas que los atraviesan, y por lo tanto no dejan pistas para poder descubrirlos simplemente observando el tráfico.

22. ¿Es transparente el encaminador (*router*)?
23. Captura las tramas correspondientes al acceder a <http://155.210.154.100> y anota los identificadores MAC de la captura. ¿Coinciden con los del encaminador? ¿Han pasado esas tramas por el encaminador?
24. Teniendo en cuenta la topología deducida anteriormente, y que no todo el tráfico pasa por el encaminador, ¿qué podemos deducir que hay entre los equipos y el encaminador? Dibuja la conexión entre los equipos y el encaminador de la Figura ??.

### 1.3.5. Conexión punto-a-punto y extremo-a-extremo

Responde a las siguientes preguntas *sin tener en cuenta la posible presencia de equipos transparentes*:

25. Si te conectas con un servidor web de google, ¿es una conexión punto-a-punto, extremo-a-extremo o ambas?
26. Si te conectas con un servidor web de google, ¿la comunicación *entre tu equipo y el encaminador* es punto-a-punto, extremo-a-extremo o ambas?
27. Si te conectas con el «Equipo Administrador», ¿es una conexión punto-a-punto, extremo-a-extremo o ambas?
28. ¿Cuál es la diferencia entre una conexión punto-a-punto y otra extremo-a-extremo?

### 1.3.6. Identificadores de la capa de transporte: puertos software

Cada una de las tramas que llega a un equipo va dirigida a un proceso específico en ejecución en el equipo. Igual que hay interfaces (puertos/conectores) hardware, la arquitectura de red TCP/IP define puertos software que asocian una transmisión (capa de transporte) y un proceso. Estos puertos software

podrían considerarse los identificadores de la capa de transporte. Cuando un cliente (por ejemplo un navegador web) desea ser atendido por cierto proceso servidor en otro equipo (por ejemplo un servidor web), debe conocer qué protocolo de transporte usa (generalmente TCP o UDP) y qué puerto software tiene asociado. Aunque el proceso servidor se puede asociar a cualquier puerto, cada servicio suele tener un protocolo/puerto «recomendado», especificado en el fichero `/etc/services`. De esta forma, el proceso cliente sabe a qué puerto software dirigir su transmisión, dependiendo del servicio deseado.

29. ¿Qué puertos usan los servicios *http*, *imap3* y *ssh* según el fichero anterior?
30. Realiza varias peticiones web desde un navegador con múltiples pestañas y observa con el *wireshark* los puertos implicados. ¿A qué puerto enviamos las peticiones web? ¿Siempre a ese puerto?
31. ¿Desde qué puerto nos responden? ¿Coincide con el anterior?
32. ¿Desde qué puerto enviamos las peticiones? ¿Siempre desde ese puerto?
33. ¿A qué puerto nos responden? ¿Coincide con el anterior?

### 1.3.7. Capa de aplicación

Para esta parte, recuerda que puedes utilizar filtros. Por ejemplo, el filtro «http» visualiza sólo las tramas que contengan el protocolo HTTP, y el filtro «http.request» visualiza sólo las tramas que contengan peticiones HTTP.

34. Accede a `http://diis.unizar.es` y localiza las tramas de respuesta HTTP. La parte inferior del analizador muestra la información transmitida en hexadecimal y en texto. ¿Qué información se ve?
35. Accediendo a `https://moodle2.unizar.es` y observa que el navegador indica mediante algún símbolo que está utilizando cifrado. Introduce tu usuario/contraseña y captura el tráfico que se genera al hacer clic en *Log in*. Puedes localizar las tramas correspondientes filtrando por número de puerto 443 en TCP o por protocolo (dependiendo de la versión de Wireshark, podría mostrarse como protocolo SSL, TLS o HTTPS). ¿Puedes ver el usuario/contraseña?
36. ¿Se verá el usuario/contraseña de servicios (web, correo, mensajería, etc.) que no vayan cifrados?
37. Aunque el analizador está configurado para que funcione sin ser administrador en el laboratorio, ¿tiene sentido que necesite permisos de administrador en un sistema multiusuario?
38. El analizador captura el tráfico que «pasa» por el medio de transmisión al que estamos conectados. Asumiendo que el medio de transmisión (cable de red) de nuestro equipo *no se comparte* con otros usuarios, ¿qué otros equipos verían el tráfico que hemos generado para acceder a Google?
39. Si estamos usando un medio compartido, como por ejemplo el aire en una conexión inalámbrica, ¿qué otros equipos verían el tráfico además de los anteriores?

Vamos a crear ahora nuestra propia aplicación: un *chat* entre dos personas. Elige un compañero que esté en otro equipo, con quien entablar la comunicación. Abre en ambos equipos un terminal. En uno de los equipos ejecuta el comando `netcat -l -p 32005` para *escuchar* por el puerto 32005. En el otro equipo ejecuta el comando `netcat <ipdestino> 32005` donde habrá que sustituir `<ipdestino>` por la dirección IP del otro equipo, que habréis obtenido en una de las preguntas anteriores.

40. Una vez esté establecida la comunicación, escribe algo en cualquiera de los dos lados. ¿Qué sucede?
41. Usa el analizador para capturar especificando el filtro «tcp.port==32005» para mostrar solamente mensajes que usen el puerto 32005. ¿Aparecen los mensajes que estas generando/recibiendo? ¿Cuál es la pila de protocolos que se está usando?

De forma similar, podemos utilizar netcat para interactuar con protocolos de aplicación basados en mensajes de texto. Activa la captura de tráfico, esta vez filtrando el puerto 80.

42. Lanza `nc -C webdiis.unizar.es 80` y escribe exactamente, respetando mayúsculas y minúsculas, y sin olvidar la línea en blanco final:

```
GET / HTTP/1.1
Host: webdiis.unizar.es
```

¿Qué ha respondido webdiis a tu mensaje? ¿Puedes verlo en el analizador?

En las secciones anteriores hemos visto identificadores usados en protocolos concretos, situados en distintas capas. Para protocolos en la capa de aplicación también pueden existir identificadores, aunque es menos común. Dado que con los identificadores lógicos anteriores (dirección IP y puerto software) ya estamos especificando un servicio, sólo sería necesario un identificador en capa de aplicación si dentro de dicho servicio estuviéramos realmente ofreciendo varios servicios.






43. Usando el navegador, accede a `http://www.fechadehoy.com/`. ¿Cuál es su dirección IP y número de puerto?
44. Accede ahora a `http://li431-124.members.linode.com/`. ¿Cuál es su dirección IP y número de puerto?
45. ¿Qué muestra el navegador al acceder a `http://50.116.5.124/`?
46. ¿Qué puede usar como identificador de aplicación el protocolo HTTP?

## 1.4. Uso básico del manual del sistema (man)

Todos los sistemas \*NIX tienen su documentación detallada en las páginas del manual, que se pueden visualizar mediante el comando `man`. Si no conoces las convenciones de sintaxis para describir parámetros de comandos puedes consultar la sección *Utility Argument Syntax*<sup>2</sup> de *The Open Group Base Specifications Issue 7*<sup>3</sup>. Para visualizar una página concreta, hay que usar como parámetro el nombre de la página deseada: `man <página>`. Por ejemplo, el comando `man man` mostrará la página del comando `man`. El manual está dividido en *secciones* (código entre paréntesis que aparece en la primera línea al lado de la página del manual que se esté consultando). Aunque no es muy frecuente, puede haber páginas con el mismo nombre en distintas secciones. Por ejemplo, `printf` es un comando del sistema y una función de C. Para listar las páginas (y su sección) cuya descripción contenga una palabra clave se puede usar `man -k <palabra.clave>`, y para especificar una sección concreta: `man <sección> <página>` (en Hendrix `man -s <sección> <página>`).

En ocasiones puede ser conveniente consultar el manual en inglés. Por ejemplo, es posible que las versiones en inglés tengan una versión más reciente que las de otros idiomas. Además, para buscar algo concreto suele ser más fácil localizarlo en inglés. Por ejemplo, si deseamos encontrar información sobre el *buffer* de cierta llamada al sistema, es fácil buscar esa palabra en el manual en inglés, pero para localizarla en el manual en español habría que pensar en todas las posibles traducciones (búfer, báfer, buffer, cola, tampón, etc.). Consulta el manual para ver la forma de cambiar el idioma.

47. Prueba a consultar el manual con un idioma diferente al que aparece por defecto. ¿Qué comando utilizas?

Cuando el sistema muestra la página del manual, normalmente lo hace a través del paginador `less`. La mayoría de los sistemas responden a las teclas , , ,  y , pero es muy útil conocer algunos comandos más, por ejemplo:

- `/<texto>` busca `<texto>` (resalta en pantalla todas las apariciones y va directamente a la siguiente aparición). Si no se pone `<texto>` repite la búsqueda anterior.
- `?` funciona igual que `/` pero busca la aparición previa.

<sup>2</sup>[http://pubs.opengroup.org/onlinepubs/9699919799/basedefs/V1\\_chap12.html](http://pubs.opengroup.org/onlinepubs/9699919799/basedefs/V1_chap12.html)

<sup>3</sup>The Open Group Base Specifications Issue 7, IEEE Std 1003.1, 2013 Edition

- `<núm>G` va directamente a la línea `<núm>`. Para ir al final, donde a veces hay ejemplos y funciones relacionadas, `OG`
- `q` sale del paginador (y de la página del manual).

Para más información: `man less`

48. ¿Qué contenido incluyen las 8 secciones básicas (numeradas 1–8) del manual?
49. ¿Cuál es el comando para mostrar el manual de la función (no del comando) `printf`?

### 1.5. ¿Sabías que...?

- *Wireshark* es un analizador bajo licencia pública general (GPL) (código fuente libre), que puedes descargar, modificar, copiar, etc. La mayoría de distribuciones GNU/Linux ofrecen el *wireshark* en forma de paquete, con lo que no es necesario descargarlo manualmente.
- *Wireshark* permite guardar y cargar capturas, incluso de otros programas. Por ejemplo, para analizar el tráfico en un encaminador (que en principio no contaría con aplicaciones con interfaz gráfico) podrías usar el comando `tcpdump`, guardar la captura (o transmitirla con `netcat`) y después abrirla con *wireshark* en otro equipo.
- Un alumno de informática la Universidad de Zaragoza fue detenido por suplantar la página web de conexión inalámbrica Wiuz, en la que se pide usuario y contraseña. Seguro que haría bien esta práctica, pero... *un gran poder conlleva una gran responsabilidad*.

## Práctica 2

# Interfaz *socket* y programación en red sobre TCP/IP

### 2.1. Objetivos

Introducción a las estructuras de direcciones, a la abstracción *socket* y al modelo cliente-servidor. Programación de una aplicación sencilla TCP. Uso de *netcat* como herramienta de depuración.

### 2.2. Introducción a los *sockets*

La capa de transporte tanto en el modelo OSI como en el modelo TCP/IP se encarga del transporte de los datos extremo-a-extremo (*end-to-end*). En esta capa se sitúan los protocolos TCP (*Transport Control Protocol*) y UDP (*User Datagram Protocol*). Dependiendo del tipo de transmisión deseada, algunas aplicaciones usan TCP, UDP o ambos. Así, TCP y UDP se encargan de comunicar dos procesos concretos, cada uno en un extremo de la comunicación, identificados mediante dos números de puerto.

La entrada/salida sobre red en la actualidad se basa en una abstracción llamada *socket*. Un *socket* (enchufe) representa uno de los extremos de una conexión bidireccional entre dos procesos. Cuando lo necesitan, los programas piden al sistema operativo la creación de un *socket*. El sistema devuelve un «descriptor» que el programa deberá usar para referirse al nuevo *socket*. Dependiendo de las características deseadas para la comunicación (en general TCP o UDP) se solicitara el tipo de *socket* correspondiente. Los dos extremos de la comunicación deben tener *el mismo tipo de socket*.

Para poder crear un *socket*, hay que especificar direcciones, números de puerto y tipo de comunicación. Aunque esta idea es sencilla, como cada protocolo tiene un formato de direcciones distinto y da soporte a ciertos tipos de comunicaciones, especificarlo de forma homogénea e independiente del protocolo no es trivial. Por ejemplo, en la familia de protocolos de Internet las direcciones IPv4 son de 32 bits, mientras que las IPv6 son de 128 bits.

#### 2.2.1. Tipos de *socket*

En la pila de protocolos TCP/IP en general se usan dos tipos de *socket*:

**SOCK\_STREAM:** Proporciona una transmisión bidireccional continua y fiable (los datos se reciben ordenados, sin errores, sin pérdidas y sin duplicados) de bytes *con conexión* mediante el protocolo TCP (*Transport Control Protocol*).

**SOCK\_DGRAM:** Proporciona una transmisión bidireccional no fiable, de longitud máxima prefijada, *sin conexión* mediante el protocolo UDP (*User Datagram Protocol*).

### 2.2.2. Números de puerto

Los puertos se identifican por un número entero sin signo de 16 bits (rango de 0 a 65535). Los puertos 0 a 1023 están reservados para los servicios «bien conocidos» (*well-known ports*). Por ejemplo, el puerto 80 está reservado para el servicio web (protocolo HTTP). En la práctica anterior ya viste que puedes consultar el puerto que ocupa cada servicio en el archivo `/etc/services`.

Es importante tener en cuenta que cada arquitectura puede definir un orden de almacenamiento distinto para los bytes de una variable (*big/little-endian*). Los 4 bytes de un entero por ejemplo pueden ser almacenados comenzando por el byte más significativo o al contrario. Dado que en el modelo TCP/IP no existe capa de representación de datos, si una máquina envía un entero a otra con distinto orden de almacenamiento, el entero será interpretado de forma errónea. Para evitar este problema, por convención se ha establecido un orden específico para transmitir enteros, y todas las aplicaciones que envíen o reciban enteros deben hacerlo en este formato de red. Al especificar los números de puerto del socket también hay que hacerlo en formato de red (`man htonl`).

## 2.3. Estructuras de direcciones

A continuación se describen las principales estructuras del API de sockets. Tienes más detalles en el capítulo 3 de la *Guía de programación en red utilizando sockets*<sup>1</sup> disponible en Moodle.

```
struct addrinfo {
    int          ai_flags;           // AI_PASSIVE, ALCANONNAME, etc.
    int          ai_family;         // AF_INET, AF_INET6, AF_UNSPEC
    int          ai_socktype;       // SOCK_STREAM, SOCK_DGRAM
    int          ai_protocol;       // use 0 for "any"
    size_t       ai_addrlen;        // size of ai_addr in bytes
    struct sockaddr *ai_addr;       // struct sockaddr_in or _in6
    char         *ai_canonname;     // full canonical hostname
    struct addrinfo *ai_next;       // linked list, next node
};
```

Figura 2.1: Estructura de (lista de) direcciones `addrinfo`

La estructura `addrinfo` (Fig. ??) contiene una lista de direcciones. Cada una de las direcciones que contiene, que puede corresponder a un protocolo distinto y por tanto tener detalles específicos, se almacena en el campo `ai_addr`. Es decir, el campo `ai_addr`, que formalmente es del tipo genérico `struct sockaddr` será en realidad de un tipo específico dependiendo de la familia de direcciones de que se trate. Las dos familias de direcciones que vamos a estudiar son las de Internet: `struct sockaddr_in` (Fig. ??) para IPv4 (con direcciones de 32 bits) y `struct sockaddr_in6` (Fig. ??) para IPv6 (con direcciones de 128 bits).

```
struct sockaddr_in {
    short int     sin_family;        // Address family, AF_INET
    unsigned short sin_port;         // Port number
    struct in_addr sin_addr;         // Internet address
    unsigned char sin_zero[8];       // Same size as struct sockaddr
};
```

Figura 2.2: Estructura de dirección `sockaddr_in` (IPv4)

Como en general no se conoce la familia de direcciones que usa el equipo remoto, todas las estructuras de direcciones usan los 16 primeros bits para indicar la familia a la que pertenecen. Así, se puede leer esa

---

<sup>1</sup>Beej's Guide to Network Programming Using Internet Sockets

```

struct sockaddr_in6 {
    u_int16_t      sin6_family;    // address family, AF_INET6
    u_int16_t      sin6_port;      // port number, Network Byte Order
    u_int32_t      sin6_flowinfo;  // IPv6 flow information
    struct in6_addr sin6_addr;     // IPv6 address
    u_int32_t      sin6_scope_id;  // Scope ID
};

```

Figura 2.3: Estructura de dirección `sockaddr_in6` (IPv6)

información independientemente del tipo de dirección y después interpretar el resto de la estructura de acuerdo a la familia de direcciones indicada. Como la estructura genérica `struct sockaddr` puede estar definida con un tamaño en el que no quepa una dirección IPv6, existe también la estructura genérica `sockaddr_storage`, que cumple la misma función pero con un tamaño mayor. En el caso de `struct sockaddr_storage`, los 16 primeros bits corresponden a un campo llamado `ss_family`. Así, suele ser útil declarar una estructura `sockaddr_storage` y después usarla mediante interpretación explícita de tipos (*type casting*) como la estructura que interese.

### 2.3.1. Implementación de `migetaddrinfo`

Para facilitar la tarea de construir la estructura de direcciones se usa la función `getaddrinfo()`. A esta función se le pasa el nombre del equipo (o dirección IP) y el servicio (o número de puerto) deseado, y proporciona la estructura de direcciones con la información necesaria para crear un socket. Además, en la llamada se especifica (con tantos detalles como se desee) el tipo de dirección que deseamos obtener.

En esta parte de la práctica hay que completar el código del apartado ?? (`migetaddrinfo.c`), que realiza una llamada a `getaddrinfo()` e imprime la estructura de direcciones obtenida. Te será muy útil la información sobre la función `getaddrinfo()`, que puedes encontrar en el manual (`man getaddrinfo`) y en la sección 5.1 de la guía de programación en red utilizando sockets, disponible en Moodle.

Para completar el código, se recomienda seguir los siguientes pasos:

- Completa los huecos numerados del código en el apartado ??
- Descarga el código de Moodle y realiza los cambios anteriores
- Compila el código en un equipo del laboratorio y corrige posibles errores y *warnings* de compilación

Puedes compilar el código manualmente (`gcc -Wall -o migetaddrinfo migetaddrinfo.c`) o vía menús si usas un entorno de desarrollo. El parámetro `-Wall` (*warnings: all*) en la compilación es recomendable para que muestre todos los avisos, incluso los más triviales. Si trabajas en *hendrix*, hay que especificar al compilador que vamos a usar las bibliotecas *socket* y *nsd*: `gcc -Wall -o migetaddrinfo -lsocket -lnsl migetaddrinfo.c`.

Una vez el programa funcione correctamente, contesta a las siguientes preguntas:

- ¿Qué *flag* hay que especificar en las pistas (*hints*) de la llamada a `getaddrinfo()` cuando vamos a solicitar una estructura de direcciones para lanzar un servidor?
- Lanza `migetaddrinfo www.unizar.es 80` ¿Cuál es su dirección IP? Verifica que en la salida del programa se muestra el 80 como puerto en formato local.
- Desde tu equipo local, lanza `migetaddrinfo moodle.unizar.es https` ¿Cuál es su dirección IP? ¿Coincide el servicio https con el número de puerto que aparece en `/etc/services`?
- ¿Qué ocurre al lanzar el programa especificando la dirección IP anterior y el número de puerto anterior?
- Lanza ahora desde Hendrix `migetaddrinfo moodle.unizar.es https` ¿Qué sucede? ¿Está definido el servicio https en Hendrix en `/etc/services`?

55. Lanza `migetaddrinfo www.v6.facebook.com http` ¿En qué se diferencia la respuesta con respecto a los casos anteriores?
56. Lanza ahora `migetaddrinfo hendrix-ssh.cps.unizar.es ssh` ¿Qué puedes deducir?
57. Lanza ahora `migetaddrinfo www.google.com http` ¿Qué puedes deducir?
58. Lanza ahora `migetaddrinfo http` y observa que la dirección obtenida no es válida. En la versión en inglés de la Wikipedia<sup>2</sup> aparecen 5 posibles usos para esta dirección. ¿Cuál de ellos corresponde a este caso?

## 2.4. El modelo cliente-servidor

El modelo más utilizado para el desarrollo de aplicaciones en red es el de cliente-servidor:

- a) El proceso servidor se pone en ejecución en algún computador y se queda a la espera de que algún cliente requiera sus servicios.
- b) Un proceso cliente es puesto en ejecución en el mismo o en otro computador de la red. En algún momento este proceso envía una petición de servicio a través de la red hacia el servidor y se queda esperando respuesta.
- c) El servidor atiende la petición, responde al cliente y se queda de nuevo esperando otro cliente.

De esta manera, un proceso de comunicación típico siguiendo este modelo sería el de un servidor web, que espera a que un cliente (navegador) le solicite una página web concreta, como puede verse en la figura ??.

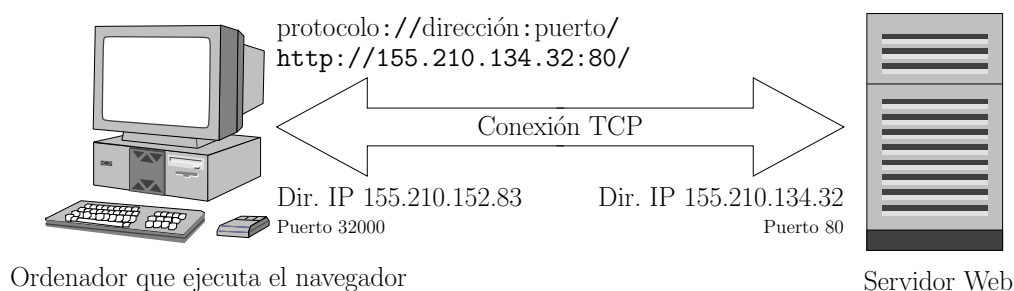


Figura 2.4: Conexión de un cliente a un servidor web

Para una comunicación *con* conexión (TCP), hay que seguir una serie de pasos, tal y como se muestra en la figura ???. El servidor creará inicialmente un extremo de la conexión pidiendo un socket (`socket()`) y asociándolo a una dirección local (`bind()`). En TCP/IP una dirección local es la dirección IP de la máquina más un número de puerto sobre un protocolo de transporte (TCP o UDP). El servidor puede en algún momento recibir varias peticiones de conexión simultáneas por lo que se debe especificar el número máximo de conexiones en espera (`listen()`). A continuación atenderá una de las conexiones pendientes (`accept()`) si las hay. Si no las hay, se quedará bloqueado hasta que las haya. Por otro lado el cliente también debe crear un socket. Es importante destacar que *se desaconseja* el uso de `bind()` en el cliente, puesto que el cliente puede usar cualquier puerto libre y no es necesario especificar uno concreto. Una vez creado el socket, lanzará una petición de conexión al servidor (`connect()`). Si el servidor está disponible (ha ejecutado `accept()` y no hay peticiones anteriores en cola) inmediatamente se desbloquean tanto cliente como servidor. En la parte del servidor, `accept()` habrá devuelto un nuevo identificador de socket que es el que realmente está conectado con el cliente. El identificador de socket original sigue sirviendo para atender nuevas peticiones de conexión a medida que se vayan realizando llamadas `accept()`. Cliente

<sup>2</sup><http://en.wikipedia.org/wiki/0.0.0.0>



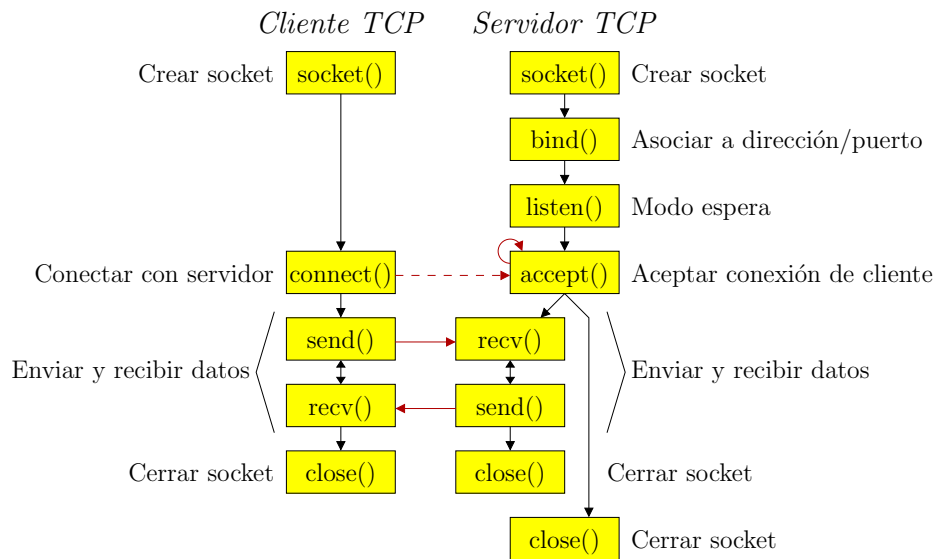


Figura 2.5: Llamadas al sistema para sockets en un protocolo orientado a conexión.

y servidor se intercambiarán datos mediante `send()` y `recv()` pudiendo finalmente cerrar la conexión mediante la llamada `shutdown()` o `close()`. En los sistemas *\*nix*, también es posible usar las llamadas `read()` y `write()` para leer y escribir de un socket.

### 2.4.1. Preguntas de comprensión

Responde a las siguientes preguntas. No olvides que puedes consultar cualquier detalle de las llamadas en el manual (e.g. `man socket`).

59. Observa los parámetros que necesita la llamada `socket()` e indica a qué campos del `struct addrinfo` corresponden.
60. Observa los parámetros que necesita la llamada `connect()` e indica a qué campos del `struct addrinfo` corresponden.
61. ¿Es necesario que el servidor esté bloqueado esperando conexión de un cliente para poder crear el socket con la llamada `socket()` en el cliente? ¿Y para iniciar la conexión con la llamada `connect()`?
62. La llamada `bind()` asocia el socket con un puerto y es necesaria en el servidor. ¿Por qué?
63. En el servidor, la llamada `accept()` devuelve un descriptor de socket, con lo que en ese punto del programa disponemos de dos descriptors de socket: el devuelto por `accept()` y el devuelto por `socket()`. ¿Cuál de los dos utilizaremos en las llamadas `send()` y `recv()` del servidor?
64. Teniendo en cuenta los dos descriptors de socket anteriores, ¿cuál es la diferencia entre usar la función `close()` con cada uno de ellos?

### 2.4.2. Herramienta *netcat*

Como ya vimos en la práctica anterior, Netcat es una herramienta para conectar transmisiones con la entrada/salida estándar. Es decir, lo que se introduce por la entrada estándar (teclado) es transmitido hacia donde se le indique, y lo que recibe se muestra en la salida estándar (pantalla). El comando puede funcionar como cliente y como servidor. Como servidor se quedará escuchando (`listen`) (-l) a la espera de conexiones entrantes en el puerto especificado (-p `numpuerto`), y como cliente iniciará una conexión con el servidor y puerto que le especifiquemos como parámetro. Por defecto, netcat realiza conexiones

mediante el protocolo de la capa de transporte TCP, pero se le puede indicar que en su lugar use el protocolo de transporte UDP con `-u`. Hay varios detalles importantes a tener en cuenta. El primero de ellos es que, tanto en *hendrix* como en los equipos del laboratorio, la herramienta que vamos a usar se lanza con el comando `netcat` y *no* con el comando `nc` (en ciertos mensajes de ayuda se muestra incorrectamente que el comando es `nc`). En segundo lugar, para la práctica es recomendable usar siempre el parámetro `-v`, que hará que el comando muestre información adicional. Para obtener información más completa, lanza `netcat -h`.

65. Lanza el netcat en el equipo de prácticas como servidor en el puerto 32005 (`netcat -l -p 32005 -v`) y a continuación lanza en *hendrix* el netcat en modo cliente para que se conecte al servidor netcat de tu equipo (`netcat -v <direccionIP> 32005`). Recuerda que en una pregunta anterior has obtenido la dirección IP de tu equipo. Una vez esté establecida la conexión, escribe algo en cualquiera de los dos lados. ¿Qué sucede?

Como hemos visto en el fichero `/etc/services`, cada servicio tiene asociado un número de puerto por defecto. Ciertos servicios requieren que el usuario se identifique, como por ejemplo el servicio *imap3*. Si ese puerto no está ocupado, en principio cualquier usuario podría lanzar el netcat en ese puerto, y con ello vería las contraseñas de cualquier otro usuario que intentara conectarse a ese servicio. Para evitar este problema, los puertos menores que 1024 están restringidos, y solo el administrador puede asociar procesos a esos puertos.

66. Prueba a lanzar el netcat como servidor en un puerto menor que 1024. ¿Qué error da?
67. En la misma máquina, lanza un servidor netcat que use TCP en cierto puerto (mayor que 1024) y al mismo tiempo (desde otra ventana) lanza otro servidor que también use TCP en el mismo puerto. No olvides poner la opción `-v`. ¿Es posible o da error?
68. Realiza el experimento anterior, pero usando TCP en uno de los servidores netcat y UDP en el otro. ¿Es posible o da error?
69. Teniendo en cuenta los resultados anteriores, ¿puede haber dos servidores usando el mismo protocolo de transporte (TCP o UDP) y el mismo puerto en la misma máquina (asumiendo que tiene una única dirección IP)? ¿Y si usan el mismo puerto pero uno usando TCP y otro usando UDP?

### 2.4.3. Implementación de programas cliente/servidor

En las secciones ?? y ?? se presenta una pareja de programas cliente/servidor *incompletos*, también disponibles en *Moodle*. El cliente lee la entrada estándar y se la envía al servidor. Éste cuenta el número de vocales y devuelve el resultado al cliente, que lo muestra por pantalla. Ten en cuenta que el carácter «fin de fichero» está asociado a la combinación de teclas `[Ctrl] + [d]`. Es decir, cuando se pulsa `[Ctrl] + [d]` se cierra el fichero de entrada y finaliza el envío de datos por parte del cliente.

En este apartado hay que completar los programas para que funcionen correctamente. Para ello, se propone seguir los siguientes pasos, empezando por el *cliente*:

- Completa de forma esquemática los huecos numerados del código en la sección ??
- Traslada al código fuente lo anterior y copia las funciones `obtener_struct_direccion()` y `printsoc_kaddr()` que has completado antes en el código `migetaddrinfo.c`
- Compila el código y corrige los posibles errores y avisos de compilación
- Verifica *parte* del funcionamiento con netcat: Lanza netcat como servidor en un puerto/servicio (e.g. 32000) de tu máquina local y a continuación lanza el cliente en el mismo equipo con los parámetros correspondientes. Lo que escribas en el cliente debería aparecer en el netcat, pero no podrás verificar la respuesta con el número de vocales.

- e) Realiza otra vez la verificación anterior, pero ahora lanzando el cliente en *hendrix*. Recuerda que tendrás que recompilar el código en *hendrix* para generar un ejecutable para dicha máquina:
- ```
gcc -Wall -o clientevocalesTCPPhendrix -lsocket -lnsl clientevocalesTCP.c
```

A continuación, haz lo mismo para el *servidor*:

- a) Completa de forma esquemática los huecos numerados del código en la sección ??
- b) Traslada al código fuente lo anterior y copia las funciones *obtener\_struct\_direccion()* y *printsoc-kaddr()* que has completado antes en el código *migetaddrinfo.c*
- c) Compila el código y corrige los posibles errores *y avisos* de compilación
- d) Verifica el funcionamiento lanzando el servidor en un puerto/servicio (e.g. 32000) de tu máquina local y a continuación el cliente en el mismo equipo con los parámetros correspondientes
- e) Realiza otra verificación lanzando el servidor en el equipo local y el cliente en *hendrix*
- f) Realiza otra verificación lanzando el cliente en el equipo local y el servidor en *hendrix*. Recuerda lo observado en la pregunta 56. Para compilar en *hendrix*:
- ```
gcc -Wall -o servidorvocalesTCPPhendrix -lsocket -lnsl servidorvocalesTCP.c
```

Una de las capas vistas en clase de teoría ha sido la *capa de presentación* de datos. Esta capa se encarga de homogeneizar los datos transmitidos entre distintos equipos. Como esta capa no existe en la arquitectura TCP/IP, este trabajo ha de hacerlo la aplicación. Ya hemos visto ejemplos de ello en las funciones tipo *ntohl()*, pero hay que verificar que cualquier dato sea interpretado correctamente. Por ejemplo, tanto en los equipos del laboratorio como en *Hendrix*, el texto se codifica mediante UTF-8.

70. Prueba a enviar desde el cliente al servidor de contar vocales los siguientes caracteres, cada uno en una línea distinta: línea sin ningún carácter, «a», «ñ», «€». ¿Cuántos bytes ocupa cada envío?

## 2.5. Evaluación de la práctica

El trabajo realizado en esta práctica forma parte de la nota de prácticas de laboratorio de la asignatura. La entrega se realizará vía *Moodle* mediante un cuestionario donde se rellenarán los huecos de los códigos proporcionados. Es muy recomendable que antes de realizar el cuestionario te asegures de que tus códigos compilan correctamente (sin *warnings*) y funcionan como deberían. En la siguiente sesión de prácticas *necesitarás usar estos códigos*. ¡Ten en cuenta la *fecha límite* del cuestionario!

## 2.6. ¿Sabías que...?

- La correspondencia puertos-protocolos se puede consultar en:  
[http://en.wikipedia.org/wiki/List\\_of\\_TCP\\_and\\_UDP\\_port\\_numbers](http://en.wikipedia.org/wiki/List_of_TCP_and_UDP_port_numbers).
- Un API (Application Program Interface) es el conjunto de funciones y procedimientos que ofrece cierta biblioteca para ser utilizado por otro software como una capa de abstracción. El interfaz de programación de aplicaciones de red original fue desarrollado para UNIX BSD (Universidad de Berkely). Para GNU/Linux se denomina API de Sockets BSD o Sockets de Berkeley. Esta interfaz también se ha portado a Windows bajo el nombre Windows Sockets, abreviado como WinSock.
- TCP es uno de los protocolos fundamentales en Internet. Fue creado entre los años 1973 y 1974 por Vinton Cerf y Robert Kahn. Vinton Cerf fue investido Doctor Honoris Causa por la Universidad de Zaragoza en 2008 (ver detalles).

## 2.7. Códigos fuente a utilizar

Para esta práctica y las posteriores necesitarás los siguientes códigos, disponibles en *Moodle*.

### 2.7.1. Código *migetaddrinfo* (migetaddrinfo.c)

```
// importación de funciones, constantes, etc.
// el preprocesador sustituye cada include por contenido del fichero referenciado
#include <stdio.h>
#include <unistd.h>
#include <stdlib.h>
#include <string.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <arpa/inet.h>
#include <netinet/in.h>
#include <netdb.h>

// cabeceras de funciones (importante que aparezcan antes de ser usadas)
struct addrinfo* obtener_struct_direccion(char *nodo, char *servicio, char
    f_verbose);
void printsockaddr(struct sockaddr_storage * saddr);

/***** MAIN *****/
/* argc indica el número de argumentos que se han usado en el la línea de
   comandos. argv es un vector de cadenas de caracteres. El elemento argv[0]
   contiene el nombre del programa y así, sucesivamente.*/
int main(int argc, char * argv[]){
    char f_verbose=1; //flag, 1: imprimir información por pantalla
    struct addrinfo* direccion; // puntero (no inicializado!) a estructura de
        dirección

    //verificación del número de parámetros:
    if ((argc!=2) && (argc!=3)) {
        printf("Número de parámetros incorrecto \n");
        printf("Uso: %s [servidor] <puerto/servicio>\n", argv[0]);
        exit(1); //sale del programa indicando salida incorrecta (1)
    } else if (argc==3) {
        // devuelve la estructura de dirección al equipo y servicio solicitado
        direccion=obtener_struct_direccion(argv[1], argv[2], f_verbose);
    } else if (argc==2) {
        // devuelve la estructura de dirección del servicio solicitado asumiendo
        que vamos a actuar como servidor
        direccion=obtener_struct_direccion(NULL,argv[1], f_verbose);
    }

    // cuando ya no se necesite hay que liberar la memoria dinámica obtenida en
    getaddrinfo() mediante freeaddrinfo()
    if (f_verbose) { printf("Devolviendo al sistema la memoria usada por servinfo
        (ya no se va a usar)... "); fflush(stdout);}
    freeaddrinfo(direccion);
    if (f_verbose) printf("hecho\n");
    direccion=NULL; // como ya no tenemos la memoria, dejamos de apuntarla para
        evitar acceder a ella por error

    // sale del programa indicando salida correcta (0)
    exit(0);
}

/***** obtener_struct_direccion *****/
```

```

/**
 * Función que, en base a ciertos parámetros, devuelve una estructura de
 * direcciones rellena con al menos una dirección que cumpla los parámetros
 * especificados. El último parámetro sirve para que muestre o no los printf
 */
struct addrinfo* obtener_struct_direccion(char *dir_servidor, char *servicio,
char f_verbose){
    struct addrinfo hints, //estructura hints para especificar la solicitud
                      *servinfo; // puntero al addrinfo devuelto
    int status; // indica la finalización correcta o no de la llamada getaddrinfo
    int numdir=1; // contador de estructuras de direcciones en la lista de
                  direcciones de servinfo
    struct addrinfo *direccion; // puntero para recorrer la lista de direcciones
                  de servinfo

    // genera una estructura de dirección con especificaciones de la solicitud
    if (f_verbose) printf("1 - Especificando detalles de la estructura de
        direcciones a solicitar... \n");
    // sobreescribimos con ceros la estructura para borrar cualquier dato que
    // pueda malinterpretarse
    memset(&hints, 0, sizeof hints);

    if (f_verbose) { printf("\tFamilia de direcciones/protocolos: "); fflush(
        stdout);}
    hints.ai_family=1; // sin especificar: AF_UNSPEC; IPv4: AF_INET; IPv6:
        AF_INET6; etc.
    if (f_verbose) {
        switch (hints.ai_family) {
            case AF_UNSPEC: printf("IPv4 e IPv6\n"); break;
            case AF_INET: printf("IPv4\n"); break;
            case AF_INET6: printf("IPv6\n"); break;
            default: printf("No IP (%d)\n", hints.ai_family); break;
        }
    }

    if (f_verbose) { printf("\tTipo de comunicación: "); fflush(stdout);}
    hints.ai_socktype=2; // especificar tipo de socket
    if (f_verbose) {
        switch (hints.ai_socktype) {
            case SOCK_STREAM: printf("flujo (TCP)\n"); break;
            case SOCK_DGRAM: printf("datagrama (UDP)\n"); break;
            default: printf("no convencional (%d)\n", hints.ai_socktype); break;
        }
    }

    // pone flags específicos dependiendo de si queremos la dirección como
    // cliente o como servidor
    if (dir_servidor!=NULL) {
        // si hemos especificado dir_servidor, es que somos el cliente y vamos a
        // conectarnos con dir_servidor
        if (f_verbose) printf("\tNombre/dirección del equipo: %s\n", dir_servidor)
        ;
    } else {
        // si no hemos especificado, es que vamos a ser el servidor
        if (f_verbose) printf("\tNombre/dirección del equipo: ninguno (seremos el
            servidor)\n");
        hints.ai_flags=3; // poner flag para que la IP se rellene con lo
            necesario para hacer bind
    }
}

```

```
}
if (f_verbose) printf("\tServicio/puerto: %s\n",servicio);

// llamada a getaddrinfo para obtener la estructura de direcciones solicitada
// getaddrinfo pide memoria dinámica al SO, la rellena con la estructura de
// direcciones, y escribe en servinfo la dirección donde se encuentra dicha
// estructura
// la memoria *dinámica* creada dentro de una función NO se destruye al salir
// de ella. Para liberar esta memoria, usar freeaddrinfo()
if (f_verbose) { printf("2 - Solicitando la estructura de direcciones con
getaddrinfo()... "); fflush(stdout);}
status = getaddrinfo(dir_servidor,servicio,&hints,&servinfo);
if (status!=0) {
    fprintf(stderr,"Error en la llamada getaddrinfo: %s\n",gai_strerror(
        status));
    exit(1);
}
if (f_verbose) { printf("hecho\n"); }

// imprime la estructura de direcciones devuelta por getaddrinfo()
if (f_verbose) {
    printf("3 - Analizando estructura de direcciones devuelta... \n");
    direccion=servinfo;
    while (direccion!=NULL) { // bucle que recorre la lista de direcciones
        printf("    Dirección %d:\n",numdir);
        printsockaddr((struct sockaddr_storage*)direccion->ai_addr);
        // "avanzamos" direccion a la siguiente estructura de direccion
        direccion=direccion->ai_next;
        numdir++;
    }
}

// devuelve la estructura de direcciones devuelta por getaddrinfo()
return servinfo;
}

/***** printsockaddr *****/
/**
 * Imprime una estructura sockaddr_in o sockaddr_in6 almacenada en
 *   sockaddr_storage
 */
void printsockaddr(struct sockaddr_storage * saddr) {
    struct sockaddr_in *saddr_ipv4; // puntero a estructura de dirección IPv4
    // el compilador interpretará lo apuntado como estructura de dirección IPv4
    struct sockaddr_in6 *saddr_ipv6; // puntero a estructura de dirección IPv6
    // el compilador interpretará lo apuntado como estructura de dirección IPv6
    void *addr; // puntero a dirección. Como puede ser tipo IPv4 o IPv6 no
    // queremos que el compilador la interprete de alguna forma particular, por
    // eso void
    char ipstr[INET6_ADDRSTRLEN]; // string para la dirección en formato texto
    int port; // para almacenar el número de puerto al analizar estructura
    // devuelta

    if (saddr==NULL) {
        printf("La dirección está vacía\n");
    }
}
```

```

    } else {
        printf("\tFamilia de direcciones: "); fflush(stdout);
        if (saddr->ss_family == AF_INET6) { //IPv6
            printf("IPv6\n");
            // apuntamos a la estructura con saddr_ipv6 (el typecast evita el
            // warning), así podemos acceder al resto de campos a través de este
            // puntero sin más typecasts
            saddr_ipv6=(struct sockaddr_in6 *)saddr;
            // apuntamos a donde está realmente la dirección dentro de la
            // estructura
            addr = &(saddr_ipv6->sin6_addr);
            // obtenemos el puerto, pasando del formato de red al formato local
            port = ntohs(saddr_ipv6->sin6_port);
        } else if (saddr->ss_family == AF_INET) { //IPv4
            printf("IPv4\n");
            saddr_ipv4 = 

|  |   |
|--|---|
|  | 4 |
|--|---|

;
            addr = 

|  |   |
|--|---|
|  | 5 |
|--|---|

;
            port = 

|  |   |
|--|---|
|  | 6 |
|--|---|

;
        } else {
            fprintf(stderr, "familia desconocida\n");
            exit(1);
        }
        //convierte la dirección ip a string
        inet_ntop(saddr->ss_family, addr, ipstr, sizeof ipstr);
        printf("\tDirección (interpretada según familia): %s\n", ipstr);
        printf("\tPuerto (formato local): %d \n", port);
    }
}

```

### 2.7.2. Código *cliente* de contar vocales (clientevocalesTCP.c)

```

//importación de funciones, constantes, etc.
#include <stdio.h>
#include <unistd.h>
#include <stdlib.h>
#include <string.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <arpa/inet.h>
#include <netinet/in.h>
#include <netdb.h>

//definición de constantes
#define MAX_BUFF_SIZE 1000 //establecemos este valor máximo para el buffer

// cabeceras de funciones
struct addrinfo* obtener_struct_direccion(char *nodo, char *servicio, char
    f_verbose);
void printsockaddr(struct sockaddr_storage * saddr);
int initsocket(struct addrinfo *servinfo, char f_verbose);

/***** MAIN *****/
int main(int argc, char * argv[]) {

```

```

//declaración de variables propias del programa principal (locales a main)
char f_verbose=1; //flag, 1: imprimir por pantalla información extra
const char fin = 4; // carácter ASCII end of transmission (EOT) para indicar
    fin de transmisión
struct addrinfo * servinfo; // puntero a estructura de dirección destino
int sock; // descriptor del socket
char msg[MAX_BUFF_SIZE]; // buffer donde almacenar lo leído y enviarlo
ssize_t len, // tamaño de lo leído por la entrada estándar (size_t con signo)
    sentbytes; // tamaño de lo enviado (size_t con signo)
uint32_t num; // variable donde anotar el número de vocales

//verificación del número de parámetros:
if (argc != 3) {
    printf("Número de parámetros incorrecto \n");
    printf("Uso : %s servidor puerto/servicio\n", argv[0]);
    exit(1); //sale del programa indicando salida incorrecta (1)
}

// obtiene estructura de direccion
servinfo=obtener_struct_direccion(argv[1], argv[2], f_verbose);

// crea un extremo de la comunicación con la primera de las direcciones de
    servinfo e inicia la conexión con el servidor. Devuelve el descriptor del
    socket
sock = initsocket(servinfo, f_verbose);

// cuando ya no se necesite, hay que liberar la memoria dinámica usada para
    la dirección
freeaddrinfo(servinfo);
servinfo=NULL; // como ya no tenemos la memoria, dejamos de apuntarla para
    evitar acceder a ella por error

// bucle que lee texto del teclado y lo envía al servidor
printf("\nTeclea el texto a enviar y pulsa <Enter>, o termina con <Ctrl+d>\n"
);
while ((len = read(0, msg, MAX_BUFF_SIZE)) > 0) {
    // read lee del teclado hasta que se pulsa INTRO, almacena lo leído en
    msg y devuelve la longitud en bytes de los datos leídos
    if (f_verbose)
        printf(" Leídos %zd bytes\n",len);

    if ((sentbytes=send(7, 8, 9,0)) < 0) { //envía
        datos al socket
        perror("Error de escritura en el socket");
        exit(1);
    } else { if (f_verbose)
        printf(" Enviados correctamente %zd bytes \n",sentbytes);
    }
    // en caso de que el socket sea cerrado por el servidor, al llamar a send
        se genera una señal SIGPIPE, que como en este código no se captura,
        hace que finalice el programa SIN mensaje de error
    // Las señales se estudian en la asignatura Sistemas Operativos

    printf("Teclea el texto a enviar y pulsa <Enter>, o termina con <Ctrl+d>\n"
        n");
}

//se envía una marca de finalización:

```



```

if (send([10], [11], [12], 0) < 0) {
    perror("Error de escritura en el socket");
    exit(1);
}
if (f_verbose) {
    printf("Enviada correctamente la marca de finalización.\nEsperando
           respuesta del servidor...");
    fflush(stdout);
}

//recibe del servidor el número de vocales recibidas:
if (recv([13], [14], [15], 0) < 0) {
    perror("Error de lectura en el socket");
    exit(1);
}
printf(" hecho\nEl texto enviado contenía en total %d vocales\n", [16]);
//convierte el entero largo sin signo desde el orden de bytes de la red al
    del host

close([17]); //cierra la conexión del socket:
if(f_verbose)
    printf("Socket cerrado\n");

exit(0); //sale del programa indicando salida correcta (0)
}

/***** initsocket *****/
//función que crea la conexión y se conecta al servidor
int initsocket(struct addrinfo *servinfo, char f_verbose){
    int sock;

    printf("\nSe usará ÚNICAMENTE la primera dirección de la estructura\n");

    //crea un extremo de la comunicación y devuelve un descriptor
    if (f_verbose) { printf("Creando el socket (socket)... "); fflush(stdout); }
    sock = socket([18], [19], [20]);
    if (sock < 0) {
        perror("Error en la llamada socket: No se pudo crear el socket");
        /*muestra por pantalla el valor de la cadena suministrada por el
           programador, dos puntos y un mensaje de error que detalla la causa del
           error cometido */
        exit(1);
    }
    if (f_verbose) { printf("hecho\n"); }

    //inicia una conexión en el socket:
    if (f_verbose) { printf("Estableciendo la comunicación a través del socket (
        connect)... "); fflush(stdout); }
    if (connect([21], [22], [23]) < 0) {
        perror("Error en la llamada connect: No se pudo conectar con el destino")
        ;
        exit(1);
    }
    if(f_verbose){ printf("hecho\n"); }

    return sock;
}

```

```

}

/* copia aquí la función obtener_struct_direccion del programa migetaddrinfo */
24
/* copia aquí la función printsockaddr del programa migetaddrinfo */
25

```

### 2.7.3. Código *servidor* de contar vocales (servidorvocalésTCP.c)

```

//importación de funciones, constantes, variables, etc.
#include <stdio.h>
#include <unistd.h>
#include <stdlib.h>
#include <string.h>
#include <sys/socket.h>
#include <arpa/inet.h>
#include <netdb.h>
#include <netinet/in.h>

//definición de constantes
#define EOT 4 //carácter ASCII end of transmission
#define BUFF_SIZE 1000 //establecemos el tamaño del buffer

//definición de funciones
struct addrinfo* obtener_struct_direccion(char *nodo, char *servicio, char
    f_verbose);
void printsockaddr(struct sockaddr_storage *saddr);
int establecer_servicio(struct addrinfo *servinfo, char f_verbose);
uint32_t countVowels(char msg[], size_t s);

/***** MAIN *****/
int main(int argc, char *argv[]) {

    //declaración de variables propias del programa principal (locales a main)
    char f_verbose=1; //flag, 1: imprimir por pantalla información extra
    struct addrinfo* servinfo; // dirección propia (servidor)
    int sock, conn; // descriptors de socket
    char msg[BUFF_SIZE]; // espacio para almacenar lo recibido
    ssize_t readbytes; // número de bytes recibidos
    uint32_t num, netNum; // contador de vocales en formato local y de red
    struct sockaddr_storage caddr; // dirección del cliente
    socklen_t clen; // longitud de la dirección

    //verificación del número de parámetros:
    if (argc != 2) {
        printf("Número de parámetros incorrecto \n");
        printf("Uso : %s puerto\n", argv[0]);
        exit(1);
    }

    // obtiene estructura de dirección
    servinfo=obtener_struct_direccion(NULL, argv[1], f_verbose);

    // crea un extremo de la comunicación. Devuelve el descriptor del socket

```

```

sock = establecer_servicio(servinfo, f_verbose);

// cuando ya no se necesite, hay que liberar la memoria dinámica usada para
// la dirección
freeaddrinfo(servinfo);
servinfo=NULL; // como ya no tenemos la memoria, dejamos de apuntarla para
// evitar acceder a ella por error

// bucle infinito para atender conexiones una tras otra
while (1) {
    printf("\nEsperando conexión (pulsa <Ctrl+c> para finalizar la ejecución)
    ...\n");

    //acepta la conexión
    clen=sizeof caddr;
    if ((conn = accept([26], (struct sockaddr *)&caddr,&clen)) < 0) {
        perror("Error al aceptar una nueva conexión");
        exit(1);
    }

    // imprime la dirección obtenida
    printf("Aceptada conexión con cliente:\n");
    printsockaddr(&caddr);

    // bucle de contar vocales hasta recibir marca de fin
    num = 0;
    do {
        if ((readbytes = recv([27], [28], BUFF_SIZE, 0)) < 0) {
            perror("Error de lectura en el socket");
            exit(1);
        }
        printf("Mensaje recibido: "); fflush(stdout);
        write(1, msg, readbytes); //muestra en pantalla (salida estándar 1) el
        // mensaje recibido
        // evitamos usar printf por si lo recibido no es texto o no acaba con
        // \0
        num += countVowels(msg, readbytes);
        printf("Vocales contadas hasta el momento: %d\n", num);

        // condición de final: haber recibido al menos un byte y que lo penúltimo
        // sea la marca de fin
        // se comprueba lo penúltimo porque lo último siempre es un <intro>
    } while ((readbytes > 0) && (msg[readbytes - 1] != EOT));

    printf("\nMarca de fin recibida\n");
    printf("Contadas %d vocales\n", num); //muestra las vocales recibidas
    netNum = htonl(num); //convierte el entero largo sin signo hostlong desde
    // el orden de bytes del host al de la red
    //envia al cliente el número de vocales recibidas:
    if (send([29], &netNum, sizeof netNum, 0) < 0) {
        perror("Error de escritura en el socket");
        exit(1);
    }
}
if(f_verbose)
    printf("Enviado el número de vocales contadas\n");

```

```
//cierra la conexión con el cliente
close(30);
if(f_verbose)
    printf("Cerrada la conexión con el cliente\n");

}
exit(0);
}

/***** establecer_servicio *****/
*/
//función que crea la conexión y espera conexiones entrantes
int establecer_servicio(struct addrinfo *servinfo, char f_verbose){
    int sock;

    printf("\nSe usará ÚNICAMENTE la primera dirección de la estructura\n");

    //crea un extremo de la comunicación y devuelve un descriptor
    if (f_verbose) { printf("Creando el socket (socket)... "); fflush(stdout); }
    sock = socket(31, 32, 33);
    if (sock < 0) {
        perror("Error en la llamada socket: No se pudo crear el socket");
        /*muestra por pantalla el valor de la cadena suministrada por el
        programador, dos puntos y un mensaje de error que detalla la causa del
        error cometido */
        exit(1);
    }
    if (f_verbose) { printf("hecho\n"); }

    //asocia el socket con un puerto
    if (f_verbose) { printf("Asociando socket a puerto (bind)... "); fflush(
        stdout); }
    if (bind(34, 35, 36) < 0) {
        perror("Error asociando el socket");
        exit(1);
    }
    if (f_verbose) { printf("hecho\n"); }

    //espera conexiones en un socket
    if (f_verbose) { printf("Permitiendo conexiones entrantes (listen)... ");
        fflush(stdout); }
    listen(37, 5); //5 es el número máximo de conexiones pendientes en
        algunos sistemas
    if (f_verbose) { printf("hecho\n"); }

    return sock;
}

//función que cuenta vocales
uint32_t countVowels(char msg[], size_t s) {
    uint32_t result = 0;
    size_t i;
    for (i = 0; i < s; i++)
        if (msg[i] == 'a' || msg[i] == 'A' ||
            msg[i] == 'e' || msg[i] == 'E' ||
```

```
        msg[i] == 'i' || msg[i] == 'I' ||  
        msg[i] == 'o' || msg[i] == 'O' ||  
        msg[i] == 'u' || msg[i] == 'U') result++;  
    }  
  
    /* copia aquí la función obtener_struct_direccion del programa migetaddrinfo */  
    38  
    /* copia aquí la función printsockaddr del programa migetaddrinfo */  
    39
```



## Práctica 3

# Programación en red sobre UDP

### 3.1. Objetivos

En esta práctica se modificará la pareja cliente/servidor de la práctica anterior para que usen el protocolo de transporte UDP.

### 3.2. Programación usando UDP

A partir de los códigos proporcionados para la práctica anterior, rellena otra vez los huecos y realiza los cambios oportunos para que la comunicación entre ellos se haga usando UDP en lugar de TCP. Al igual que el servidor TCP, el servidor UDP debe ser capaz de servir a varios clientes, uno detrás de otro. Para ello sería necesario verificar si los mensajes recibidos son del mismo remitente, pero *no es necesario considerar el caso de varios remitentes simultáneos*. Para el servidor vamos a asumir que todos los mensajes recibidos desde la primera cadena hasta la marca de fin son del mismo remitente, y después atenderemos a nuevos clientes asumiendo lo mismo. Aunque las llamadas al sistema a utilizar son ligeramente distintas (Fig. ??), los conceptos a aplicar son los mismos, con lo que te será útil toda la documentación de la práctica anterior.

### 3.3. Evaluación de la práctica

El trabajo realizado en esta práctica forma parte de la nota de prácticas de laboratorio de la asignatura. La entrega se realizará vía *Moodle* mediante dos tareas: una para entregar el código del cliente y otra para el código del servidor. Para cada uno de ellos se entregará un único fichero .c sin comprimir

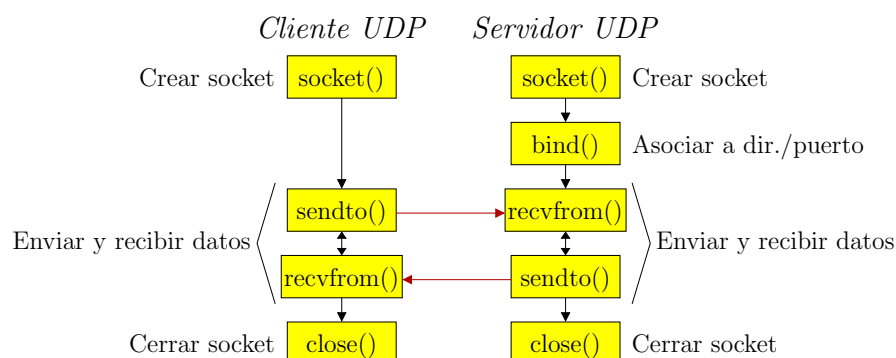


Figura 3.1: Llamadas al sistema para sockets con UDP.

que compile con gcc en los equipos del laboratorio sin errores. Tanto el cliente como el servidor deben funcionar correctamente. *Entregas que no cumplan las especificaciones de la entrega, que no compilen, o que no funcionen obtendrán una nota de 0.* La entrega debe realizarse antes de la fecha establecida en Moodle. En la siguiente sesión de prácticas *necesitarás usar estos códigos.*

### 3.4. ¿Sabías que...?

- El departamento de informática e ingeniería de sistemas dispone del equipo *lab000.cps.unizar.es*, equivalente a los del laboratorio, al que se puede acceder de forma remota. Puedes usar dicho equipo para probar o completar las tareas de prácticas desde fuera de la universidad.



## Práctica 4

# Implementación de protocolos y algoritmos de secuenciación (trabajo práctico)

El enunciado de la práctica y trabajo relacionado estará disponible en la página web de la asignatura con antelación a la práctica 4.



## Práctica 5

# Topología y tráfico en Internet

### 5.1. Objetivos

Exploración de la topología y la organización del tráfico en Internet. Uso de herramientas comunes de exploración de redes.

### 5.2. Topología y tráfico en Internet

La estructura de Internet en cuanto a topología parece un caos de redes y enlaces que crecen de forma continua en todas partes y sin ningún control. Sin embargo, a pesar del desorden aparente, se puede encontrar una cierta estructura jerárquica (Fig. ??). Los proveedores de servicios de Internet (ISP, *Internet Service Providers*) son empresas que proporcionan servicios para acceder, usar o participar en la red Internet. Estas empresas pueden clasificarse de acuerdo a los servicios que ofrecen a sus clientes. Hay ISPs pequeños que tienen en propiedad una red de alcance local o regional, y contratan líneas a otras empresas para poder ofrecer el servicio de acceso a Internet. También pueden encontrarse los ISP de nivel nacional, que cuentan con una infraestructura de red a nivel del país (o grupo de países, como España y Portugal) en que desarrollan su actividad. Estos ISPs de nivel nacional tendrán que contratar líneas o servicios a terceras empresas cuando quieran ofrecer una «salida» internacional a sus clientes, mientras que el tráfico nacional puede ir por su red. Finalmente estarían las grandes compañías de telecomunicaciones que cuentan con líneas propias que conectan sus infraestructuras en diversos países. En este caso, es usual que dichas compañías se presenten como distintos ISPs (distintas marcas o filiales). Entre ellas estarían las empresas que cuentan con las líneas (inter)continentales (los *backbones* de Internet).

En general, una empresa con grandes infraestructuras propias (sea ISP o no) tendría registradas dichas infraestructuras como uno o más *Sistemas Autónomos* (SA/AS), cada uno con su *Autonomous System Number* (ASN). Los SAs están gestionados por los *Registros Regionales de Internet* (RIR). Por ejemplo, en *RIPE NCC* puedes encontrar toda la información sobre los SAs que gestiona. Como dicha información es enorme, es más cómodo filtrarla para las necesidades específicas que se requieran. En este caso nos interesa el buscador RIPEstat (<https://stat.ripe.net>).

71. Escribe «rediris» en el campo de búsqueda y espera que muestre elementos relacionados (*no hagas clic en «Go»*). La red académica y de investigación española RedIRIS es un SA. ¿Cuál es su ASN? ¿Coincide con el que figura en las transparencias de teoría?
72. ¿Cuál es el ASN del SA GEANT? ¿Coincide con el que figura en las transparencias de teoría?
73. La compañía *Vodafone* debería tener al menos un SA de *tránsito*. ¿Cuántos SAs contienen «Vodafone» en su nombre?
74. Grandes compañías que no están dedicadas a tránsito suelen tener SAs *multihomed*, es decir, con conexiones a varios SAs. ¿Cuántos SAs tiene *Nestle*?

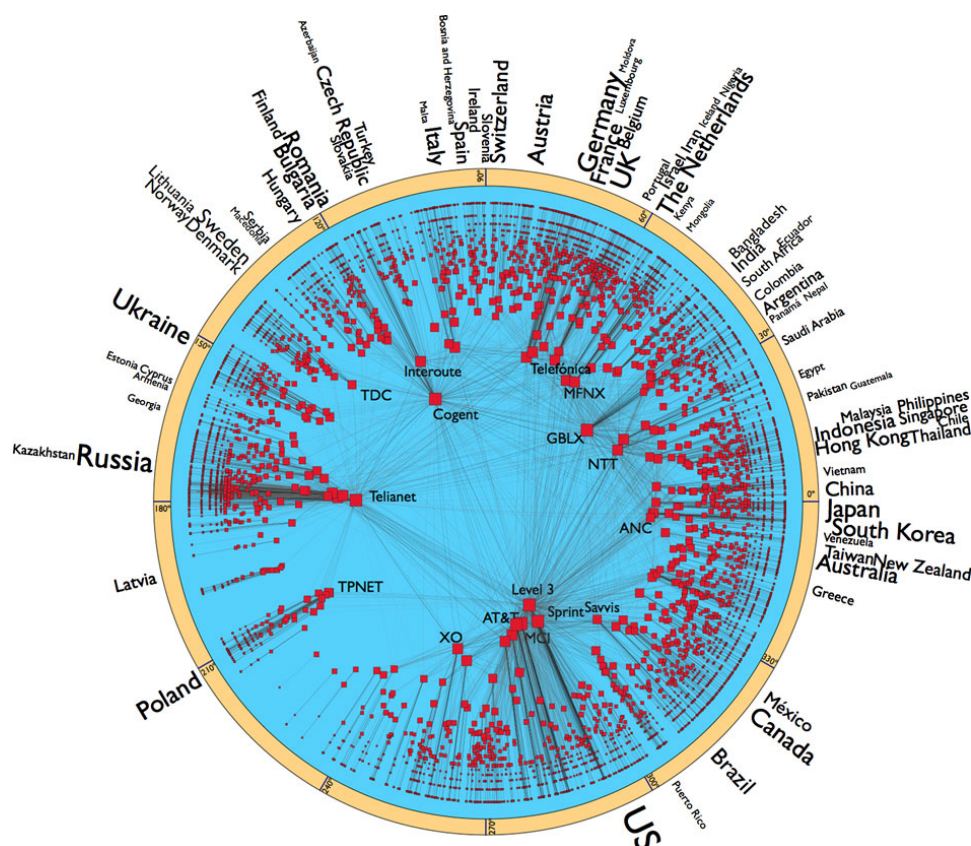


Figura 5.1: Mapa hiperbólico de Internet. Los nodos representan Sistemas Autónomos. El tamaño de letra de los países representa el número de SAs que tiene [B. Marián, P. Fragkiskos, K. Dmitri. Sustaining the Internet with hyperbolic mapping. Nat Commun, 1:62, sep 2010].

75. Empresas o instituciones no tan grandes también pueden llegar a tener un SA de tipo *stub*, es decir, conectado únicamente a SAs de tránsito. ¿Tiene SA el Boletín Oficial del Estado (BOE)?

Vuelve a introducir «rediris», y ahora haz clic en «Go» para obtener información detallada de dicho sistema autónomo. Por ejemplo, en el recuadro «Routing Status» se indica el valor de sus *Observed BGP neighbours*. Puedes hacer clic en ese valor para ver sus vecinos.

76. ¿Cuántos vecinos BGP tiene RedIRIS?
77. ¿Cuántos vecinos BGP tiene el BOE? ¿Cuáles son?

### 5.2.1. Tráfico entre SAs

Como sabes, la red Internet se basa en el modelo de conmutación de paquetes, por lo que la información va de un punto a otro en forma de datagramas. Dado que existen tantos SAs interconectados, cada uno de estos datagramas podría seguir un camino totalmente diferente al seguido por los demás. Sin embargo, el funcionamiento de Internet se rige en gran medida por patrones económicos: la mayoría de redes (SAs) pertenecen a empresas que deciden si cursan o no el tráfico proveniente de otras empresas.

Para que una empresa pueda usar la infraestructuras de otras se debe establecer una relación comercial previa, de forma que la empresa propietaria de las líneas reciba una compensación por el tráfico que cursa («deja pasar»). A esta relación comercial se la denomina «acuerdo de tránsito». Estos acuerdos se reflejan en la configuración BGP de cada SA, tal y como has comprobado en preguntas anteriores. Existe otro tipo de relación entre empresas que se produce cuando las dos empresas son más o menos de

igual tamaño y los tráficos que una originará en la otra son semejantes. En estos casos, se crean acuerdos de intercambio de datos sin que exista contraprestación económica. A estos acuerdos se los denomina «acuerdos de *peering*», ya que se establecen entre empresas que se reconocen como *peer* o iguales.

Debido a estos acuerdos, el camino seguido por un paquete desde su origen hasta su destino es más predecible de lo que se pudiese pensar en un principio y no cambia prácticamente de un paquete a otro a no ser que uno de los encaminadores esté muy congestionado o fuera de servicio.

Sin embargo, esta forma de conexión entre ISPs dio lugar a un problema grave de retardos y de saturación de los enlaces, ya que se daban situaciones en las que, para que dos usuarios conectados con dos ISPs diferentes de la misma región pudiesen intercambiar datos, el flujo de información llegaba a salir por enlaces internacionales antes de llegar a su destino. Por ejemplo, podía darse el siguiente flujo de datos para comunicar a dos usuarios leoneses A y B que usasen los ISPs locales *Nora* y *Lesein*:

Usuario A → Nora Internet (León, España) → Retevisión (España) → BT Ignite (España) → BT Ignite (Reino Unido) → BT Ignite (EE.UU.) → Cable & Wireless (EE.UU.) → Cable & Wireless (Amsterdam, Holanda) → Cable & Wireless (España) → Telefónica Data (España) → Lesein (León, España) → Usuario B

Como puede observarse, los datos se intercambian en Estados Unidos, donde existe un acuerdo de tránsito (o de *peering*) entre *BT Ignite* y *Cable & Wireless*, por lo que, para conectar a dos usuarios locales, los datos viajan hasta Estados Unidos a través de las redes de seis ISPs diferentes. Esta situación, además de causar un mayor retardo en las comunicaciones, saturaba los enlaces transoceánicos (que son menos numerosos que los continentales), por lo que los ISP buscaron una forma de evitar el problema. La solución fue crear los llamados *puntos neutros* de intercambio, normalmente a nivel nacional, de forma que el intercambio de datos entre proveedores del mismo país no salga de las redes existentes en el mismo.

En España, el punto neutro a nivel nacional se denomina *Espanix* y está en Madrid. El punto neutro es un lugar físico con un dispositivo que permite intercambiar los datos entre los ISPs que así lo desean. La dirección del punto neutro español es <http://www.espanix.net> y para poder conectarse al punto neutro, el ISP debe cumplir una serie de normas. En el caso de *Espanix*, se exige que el ISP tenga medios suficientes para dirigir el tráfico internacional por sí mismo, sin que intervenga ninguna de las redes de los otros ISPs conectados. Una vez conectados los ISPs, el intercambio de datos efectivo se debe consensuar entre cada par de ISPs, mediante acuerdos de *peering*.

78. ¿Cuáles son los socios de *Espanix*?
79. ¿Se encuentra RedIRIS entre los socios de *Espanix*?
80. ¿Coincide el ASN de RedIRIS con el hallado anteriormente?

En España existen otros puntos neutros, pero de nivel local, como por ejemplo *Catnix*, que se encarga de mantener el tráfico local dentro de Cataluña. También hay puntos neutros en Europa. La *European Internet Exchange Association* (<http://www.euro-ix.net>) es una asociación de puntos neutros europeos.

81. Observa la lista de ASNs conectados a los puntos neutros europeos (en Euro-IX, IXPDB → Participants → # of IXP connection). ¿Qué empresas están presentes en más puntos neutros?
82. En cuanto a latencia, ¿qué implicaciones tiene estar en muchos puntos neutros?

### 5.2.2. Consultas mediante WHOIS

Además de consultar los datos en los distintos RIRs y puntos neutros, se puede consultar mucha información mediante el protocolo WHOIS. Puedes realizar consultas con este protocolo mediante el comando `whois`. Para especificar la base de datos a consultar, usa la opción `-h`.

Usa el comando `whois -h whois.ripe.net` con la dirección IPv4 de tu equipo.

83. ¿Cuál es el nombre de la red a la que pertenece (*netname*)?

84. ¿A qué SA pertenece dicha red (*mnt-by*)?
85. ¿Cuál es su ASN (*origin*)?
86. ¿Qué responde `whois -h whois.arin.net` en *Comment* al usar la dirección IP 127.0.0.1? ¿Coincide con lo visto en clase de teoría?
87. ¿Y al usar la dirección IP 10.0.0.1? ¿Coincide con lo visto en clase de teoría?
88. Consulta ahora con `whois` la dirección IPv6 de tu equipo. ¿A qué ISP pertenece?
89. Averigua mediante `whois` a qué SA pertenece la dirección IP pública que te proporciona el ISP en tu casa.
90. Usando también el comando `whois`, pregunta por el ASN de RedIRIS (`whois as<núm>`). Entre otras cosas, podrás ver su configuración en cuanto a acuerdos de tránsito.
91. Entre los SAs desde los que acepta tráfico (*import/from/accept*), ¿se encuentra EASYNET? ¿Cuál es su ASN? Comprueba si este SA está en Espanix.
92. Entre los SAs hacia los que anuncia sus propias rutas (*export/to/announce*), ¿se encuentra AS15169? ¿A qué empresa pertenece ese SA?
93. Finalmente, habrás notado que las respuestas pueden incluir direcciones postales y personas responsables de las distintas redes. Usa `whois` para preguntar por la persona con identificador MJG8-RIPE. ¿De qué red crees que es responsable?

### 5.2.3. DNS como organizador de tráfico

El servidor de nombres de dominio realiza traducciones de nombres *fácilmente usables por personas* a direcciones IP. Este servicio también puede ser utilizado para organizar el tráfico de un SA a nivel mundial. Por ejemplo, ya has visto que *Google* se encuentra en muchos puntos neutros en Europa.

94. ¿Un usuario en España obtendrá el mismo resultado al traducir el nombre *www.google.com* (mediante comandos como `host` o `dig`) que un usuario en Francia?
95. Realiza dicha consulta DNS y a continuación haz la misma consulta desde `http://centralops.net`. ¿Se obtiene la misma respuesta?
96. En cada punto neutro, Google dispone de varios servidores, cada uno con su dirección IP, asociados al mismo nombre de dominio. Realiza una consulta DNS mediante `host www.google.com` y a continuación repite exactamente el mismo comando varias veces. ¿Son iguales las respuestas? ¿Cuál puede ser la razón?

### 5.2.4. Tráfico en tiempo real

Entre los SAs anteriores habrás visto la empresa *Akamai*. Esta empresa ofrece, entre otros servicios, una plataforma de computación distribuida para el cacheo global de contenidos de Internet y balanceo de aplicaciones. Al ser una empresa con gran presencia internacional y disponer de importantes infraestructuras de información, dispone de información muy interesante y actualizada que se puede consultar en el siguiente enlace: <http://www.akamai.com/html/technology/dataviz1.html>.

97. ¿Qué porcentaje de tráfico se registra en este momento en España?

### 5.3. Traceroute

La herramienta **traceroute** (**tracert** en MS-DOS y derivados) sirve para descubrir el camino que siguen los paquetes desde su origen hasta su destino. Ejemplo:

```
lab102-194:~$ traceroute www.google.com
traceroute to www.google.com (74.125.39.105), 30 hops max, 60 byte packets
 1  155.210.154.254 (155.210.154.254)  0.984 ms  1.272 ms  1.488 ms
 2  155.210.251.9 (155.210.251.9)  0.281 ms  0.329 ms  0.388 ms
 3  155.210.248.41 (155.210.248.41)  0.598 ms  0.801 ms  0.888 ms
 4  155.210.248.66 (155.210.248.66)  1.763 ms  2.073 ms  2.099 ms
 5  193.144.0.169 (193.144.0.169)  1.687 ms  1.963 ms  1.961 ms
 6  GE0-2-0.EB-Zaragoza0.red.rediris.es (130.206.195.13)  2.123 ms  2.025 ms  2.983 ms
 7  XE1-1-2.ciemat.rt1.mad.red.rediris.es (130.206.245.5)  10.727 ms  10.469 ms  10.281 ms
 8  mad-b1-link.telialia.net (213.248.81.25)  11.827 ms  12.053 ms  12.001 ms
 9  prs-bb1-link.telialia.net (213.155.131.152)  39.742 ms prs-bb1-link.telialia.net (80.91.245.58)  40.045 ms
 prs-bb2-link.telialia.net (80.91.245.60)  34.814 ms
10  ffm-bb2-link.telialia.net (80.91.246.184)  43.939 ms ffm-bb2-link.telialia.net (80.91.246.180)  43.901 ms
 ffm-bb2-link.telialia.net (80.91.246.182)  43.910 ms
11  s-bb1-link.telialia.net (80.91.246.211)  76.116 ms s-bb2-link.telialia.net (80.91.251.146)  65.403 ms
 s-bb2-link.telialia.net (80.91.248.58)  120.067 ms
12  s-b3-link.telialia.net (80.91.253.226)  65.712 ms s-b3-link.telialia.net (80.91.249.220)  75.565 ms
 s-b3-link.telialia.net (213.155.131.121)  75.565 ms
13  google-ic-130574-s-b3.c.telialia.net (213.248.93.194)  66.620 ms  66.954 ms  72.173 ms
14  209.85.250.192 (209.85.250.192)  84.284 ms  66.893 ms  74.520 ms
15  * * *
16  * * *
17  209.85.254.114 (209.85.254.114)  73.455 ms  73.464 ms 209.85.254.116 (209.85.254.116)  73.542 ms
18  * 209.85.249.166 (209.85.249.166)  76.278 ms *
19  fx-in-f105.1e100.net (74.125.39.105)  72.922 ms  68.083 ms  68.045 ms
```

Revisa el manual del comando **traceroute**. A continuación responde a las siguientes preguntas.

98. ¿Qué campo del protocolo IP usa este programa para que los paquetes no lleguen más allá de cierta distancia?
99. Si todo va bien, ¿cómo sabe a qué máquinas han llegado?
100. ¿Qué indican los asteriscos que aparecen a veces?
101. ¿Cuántos paquetes sonda se envían a cada distancia específica?
102. Observa que, para ciertas distancias, las respuestas vienen de direcciones distintas. ¿Qué puedes deducir en esos casos?

#### 5.3.1. Traceroute (y otras herramientas) en servidores web

En Internet existen una serie de lugares que proporcionan servidores de herramientas de red (traceroute, ping, etc.). Entre las herramientas usuales, un **traceroute** «especular» nos informa de los resultados de la ejecución de un comando **traceroute** desde un *host* hasta el nuestro. A estos servidores se les suele llamar *Looking Glass*. Por ejemplo, puedes probar <http://centralops.net>, <http://ping.eu>, <http://www.yougetsignal.com> y <https://www.ip2location.com/free/traceroute>. Los dos últimos además muestran en un mapa la ubicación de los encaminadores por los que pasan. Esta información la obtiene de la base de datos de los distintos registros regionales de Internet (RIR).

103. Realiza varias pruebas (con visualización sobre mapa) de rutas desde distintos continentes hasta tu equipo

## 5.4. ¿Sabías que...?

- Desde hace años hay planes para crear un punto neutro en Aragón (*Aragonix*). En concreto, el I Plan Director de Infraestructuras de Telecomunicaciones de Aragón ([http://www.aragon.es/estaticos/ImportFiles/24/docs/Areas/SocInfo/PlnEstrtgAra/PlanDir/I\\_PLAN\\_DIRECTOR\\_INFRAESTRUCTURAS\\_DE\\_TELECOMUNICACIONES\\_ARAGON.pdf](http://www.aragon.es/estaticos/ImportFiles/24/docs/Areas/SocInfo/PlnEstrtgAra/PlanDir/I_PLAN_DIRECTOR_INFRAESTRUCTURAS_DE_TELECOMUNICACIONES_ARAGON.pdf)), presentado en 2006, contemplaba la creación de un punto neutro de Aragón para conectar «tanto los proveedores aragoneses de acceso a Internet, la Universidad y centros de investigación, la Administración y los proveedores de acceso a Internet de carácter nacional». Posteriormente, en 2009, el Plan Director 2.0 para el Desarrollo de la Sociedad de la Información en la Comunidad Autónoma de Aragón, ([http://www.aragon.es/estaticos/ImportFiles/24/docs/Areas/SocInfo/PlnEstrtgAra/IIPlan/II\\_PLAN\\_DIRECTOR\\_SOCIEDAD\\_INFORMACION\\_ARAGON.pdf](http://www.aragon.es/estaticos/ImportFiles/24/docs/Areas/SocInfo/PlnEstrtgAra/IIPlan/II_PLAN_DIRECTOR_SOCIEDAD_INFORMACION_ARAGON.pdf)) incluía entre sus acciones «la implementación de un punto neutro que permitirá la interconexión de todas las Administraciones públicas aragonesas entre sí y con la red SARA<sup>1</sup>».
- Los ISPs proporcionan servicio DNS a sus clientes. Además, hay empresas (por ejemplo, *Google*) que ofrecen servidores DNS de uso público que se pueden añadir a los del proveedor para aumentar la tolerancia a fallos. Ten en cuenta que al usarlos les estás facilitando tu dirección IP y los sitios web que visitas.

---

<sup>1</sup>La red SARA es la infraestructura común de comunicaciones del Estado español.



## Práctica 6

# Herramientas básicas de red

### Objetivos

Uso de herramientas y comandos para visualizar y configurar conexiones a Internet.

### 6.1. Introducción

Para conectarse a Internet, un equipo debe conocer cierta información básica: la dirección IP de su interfaz de red (podría tener más de uno), la dirección de su encaminador por defecto y la dirección de su servidor de nombres de dominio (DNS). Esta información la puede obtener automáticamente a través del protocolo DHCP si en su red hay un servidor DHCP activo. De esta forma, al arrancar, el ordenador pide dicha información y el servidor se la proporciona. Actualmente esto es lo habitual en redes domésticas y en redes inalámbricas. En otros escenarios es necesario configurar manualmente estos parámetros. A continuación se describe el uso de algunas herramientas y comandos para consultar y modificar la configuración de red y para comprobar su correcto funcionamiento.

### 6.2. Herramienta ping

`ping` es quizá la más básica de todas las herramientas de red. Su finalidad es comprobar que un determinado equipo es alcanzable y es capaz de responder a nuestros mensajes. Ejemplo:

```
lab000:~$ ping www.google.com
PING www.l.google.com (74.125.39.106) 56(84) bytes of data.
64 bytes from fx-in-f106.1e100.net (74.125.39.106): icmp_seq=1 ttl=51 time=37.1 ms
64 bytes from fx-in-f106.1e100.net (74.125.39.106): icmp_seq=2 ttl=51 time=40.2 ms
64 bytes from fx-in-f106.1e100.net (74.125.39.106): icmp_seq=3 ttl=51 time=37.1 ms
64 bytes from fx-in-f106.1e100.net (74.125.39.106): icmp_seq=4 ttl=51 time=37.0 ms
64 bytes from fx-in-f106.1e100.net (74.125.39.106): icmp_seq=5 ttl=51 time=40.9 ms
^C
--- www.l.google.com ping statistics ---
5 packets transmitted, 5 received, 0% packet loss, time 4533ms
rtt min/avg/max/mdev = 37.067/38.516/40.959/1.734 ms
```

Revisa el manual del comando `ping` y contesta a las siguientes preguntas.

104. ¿Qué tipo de mensaje es un `ping`? ¿Qué protocolos utiliza?
105. ¿Qué hace la opción `-f`? ¿Por qué para usarla hay que ser superusuario?
106. Haz un `ping` a todos los ordenadores del laboratorio usando *broadcast*.

### 6.3. Interfaces de red

Cada estación conectada a Internet necesita disponer de una dirección IP asociada al interfaz de red. En GNU/Linux, la consulta o modificación de la configuración de interfaces se realiza mediante el comando `ifconfig`, mientras que en Windows se usa el comando `ipconfig`. Para interfaces inalámbricos, el comando `iwconfig` permite visualizar y configurar sus características específicas. Las distribuciones modernas de GNU/Linux están en proceso de sustituir los comandos de configuración originarios de BSD por el comando `ip`.

Este comando engloba los anteriores y además permite configurar elementos como multicast, túneles, control de tráfico, IPsec, etc. Puedes encontrar ejemplos de uso de `ip` en Internet<sup>1</sup>. En general, cada sistema puede tener un comando específico, aunque su funcionamiento suele ser similar.

Ejecuta el comando `ifconfig` (situado en `/sbin/`). Cada interfaz (a la izquierda) muestra detalles sobre su configuración, incluyendo direcciones, datos transmitidos (TX) y datos recibidos (RX) hasta el momento.

107. Viendo la configuración, ¿qué es el interfaz `lo`? ¿Cuál es su MTU? ¿Qué direcciones IPv4 e IPv6 tiene asignadas? ¿Por qué no tiene asociada una dirección hardware (MAC) como otros interfaces?
108. Ejecuta el comando `ip addr` y comprueba que esencialmente aparece la misma información.
109. ¿Cuál de los dos comandos muestra el tiempo de validez de las direcciones IP asignadas? Observa que se va decrementando con el tiempo.

En un equipo de usuario, los interfaces tienen nombres como `eno0` o `enp2s0` para Ethernet (anteriormente <sup>2</sup> `eth0`) o `wls1` para wlan (anteriormente `wlan1`), o tener etiquetas dependientes de la tarjeta de red. Cada interfaz activo tiene una dirección IP asociada. Los equipos del laboratorio, además, tienen configurados puentes (*bridges*) virtuales (`br`, `virbr`). Estos conmutadores virtuales se configuran con el comando `brctl` y permiten conectar máquinas virtuales de forma equivalente a como se conectarían si fueran máquinas reales.

### 6.4. Conectividad local

Si estuviéramos en una red aislada (sin encaminador hacia Internet) tendríamos conectividad local, es decir, podríamos comunicarnos con el resto de equipos conectados a esa misma red local. Para ello, el ordenador construirá paquetes cuya dirección IP destino conoce (es con quien se quiere comunicar), pero posiblemente no conozca su identificador MAC, que debe incluir como destino en la cabecera ethernet (o el protocolo físico que corresponda), así que necesita algún medio para obtener esa información. En clase hemos visto que en IPv4 las direcciones lógicas (IP) se asocian a los identificadores físicos (MAC) mediante el protocolo ARP (*Address Resolution Protocol*), mientras que en IPv6 esas asociaciones se gestionan a través de ICMPv6 (*Neighbour Discovery*). Es decir, cualquier equipo tiene una tabla con asociaciones entre direcciones IP y sus correspondientes identificadores MAC. Así, cuando se necesita un identificador MAC se genera un mensaje ARP (o ICMPv6) de difusión total (*broadcast*) del tipo «Quien tenga esta dirección IP, que me diga su identificador MAC». Ese mensaje llegará a todos los equipos de la red local, y el equipo aludido responderá. Con esa información se actualiza la tabla de asociaciones IP-MAC. El comando `arp` permite ver y manipular esta tabla de asociaciones en IPv4, mientras que el comando `ip neigh` funciona tanto para IPv4 como para IPv6.

110. Comprueba qué asociaciones tienes en este momento.
111. En el ping broadcast anterior, ¿cuál es el id. MAC destino del ping? ¿Por qué no está en la tabla de vecinos?
112. Revisa qué equipos han respondido a tu ping broadcast anterior, elige uno y hazle un ping. ¿Ha cambiado la tabla de vecinos? ¿Por qué?

---

<sup>1</sup><http://dougvitale.wordpress.com/2011/12/21/deprecated-linux-networking-commands-and-their-replacements/>

<sup>2</sup>Desde la versión v197 de `systemd/udev` se asignan nombres de interfaz de red predecibles y persistentes a los interfaces de red. Más información en <https://www.freedesktop.org/wiki/Software/systemd/PredictableNetworkInterfaceNames/>

## 6.5. Tablas de encaminamiento

Una función importante del nivel de red es el encaminamiento del tráfico. La tabla de encaminamiento especifica hacia donde hay que mandar un paquete dependiendo de su dirección destino.

### 6.5.1. IPv4

El comando `route` (actualizado por el comando `ip route`) permite consultar las rutas actuales y configurar las rutas de forma estática, es decir, manualmente sin que intervenga ningún protocolo de búsqueda de caminos óptimos. Lanzado sin argumentos, el comando `route` muestra la tabla de encaminamiento de tu equipo. Observa también la información mostrada por el comando `ip route`.

113. ¿Cuál es el encaminador por defecto de tu equipo?
114. ¿A qué redes está conectado tu equipo? ¿Cuál es la máscara de cada una de ellas en formato CIDR? ¿Se corresponden los bits a los mostrados en *Genmask*?
115. ¿Qué indica el que para ciertos destino no haya *vía* para llegar a ellos?
116. ¿Cuál sería el comando para añadir una ruta a la red 145.145.20.0 con máscara 255.255.255.0, pasando por el encaminador (*gateway*/siguiente salto) 145.145.20.1 a través del interfaz eth2?
117. ¿Qué indica *Metric*?

En GNU/Linux, uno de los métodos de comunicación entre el núcleo y el resto del sistema es a través del sistema de ficheros virtual montado en `/proc`, donde cada fichero es una especie de variable con cierto valor (o valores). Por ejemplo, el contenido del fichero `/proc/sys/net/ipv4/ip_forward` indica si el equipo está actuando como encaminador (1) o no (0). Si el equipo no actúa como encaminador, cuando recibe un paquete con una dirección IP que no le pertenece, directamente lo descarta. En cambio, si está actuando como encaminador, reexpedirá ese paquete como corresponda según su tabla.

118. ¿Está actuando tu equipo como encaminador?

Por otro lado, en la asignatura hemos visto muchos algoritmos que dependen de parámetros. En la mayoría de los casos estos parámetros se pueden cambiar, dependiendo del sistema operativo. En GNU/Linux, se puede interactuar con el sistema mediante la función `setsockopt()` (específica para parámetros de red), mediante el directorio `/proc/sys/` o mediante el comando `sysctl` (en `/sbin`). Este comando proporciona (y permite modificar) la misma información que hay en el directorio `/proc/sys/`. Lanza `sysctl -a` para mostrar todos los parámetros (puedes filtrar los relativos a las redes con `grep net`). También puedes buscar el valor de un parámetro si conoces el nombre que tiene.

119. ¿Coincide el valor de `net.ipv4.ip_forward` con el de la pregunta anterior?
120. ¿En qué se diferencia la tabla de un ordenador normal de la de un encaminador?

### 6.5.2. IPv6

La configuración usando IPv6 es muy similar a la de IPv4. La diferencia más importante es que en IPv6 existe la autoconfiguración *sin estado*, en la que los equipos rellenan automáticamente los campos de sus direcciones IP con la información que conocen. De esta forma, al arrancar un equipo se autoconfigura como mínimo con una dirección localmente válida.

Entra en <http://test-ipv6.com/> para comprobar tu conectividad IPv6 y observa los resultados. Revisa los *Tests Run* y la *Technical Info*.

121. ¿Qué puedes deducir?

Revisa las direcciones IPv6 en las transparencias de clase. Busca en el manual de `route` cómo mostrar las rutas del protocolo IPv6 y muéstralas. Haz lo mismo para el nuevo comando `ip route`.

122. ¿Hay alguna red con dirección globalmente única en los destinos? ¿Hay encaminador por defecto? ¿Qué implica eso?
123. ¿Qué simboliza el prefijo `fe80`? ¿Qué implica lo que aparece en sus campos *Next Hop*?
124. ¿Qué simboliza el prefijo `ff`?

## 6.6. Servidores de nombres de dominio

Con todo lo anterior correctamente configurado, el equipo ya tiene conectividad a Internet. Aún así, existe un servicio adicional que se considera básico. Ese servicio es el servidor de nombres, que realiza traducciones de nombres *fácilmente usables por personas* a direcciones IP. Los nombres o *dominios* son jerárquicos. Por ejemplo, todos los nombres dentro del dominio de España acaban en *.es* y todos los equipos de la Universidad de Zaragoza acaban en *.unizar.es*. En general, como mínimo cada red dispone de dos equipos encargados de la traducción de nombres. Para que nuestro equipo los conozca, hay que especificar cómo llegar a ellos, es decir, su dirección IP. En GNU/Linux, esa especificación se encuentra en el fichero `/etc/resolv.conf`, que también incluye el dominio que añadirá por defecto a los nombres que vaya a traducir.

125. ¿Cuáles son los servidores de nombres de tu máquina?
126. ¿Por qué están especificados con su dirección IP y no con su nombre?
127. ¿Qué pasaría si fallaran los dos?
128. Si tienes una conexión TCP activa con *www.google.com*, ¿qué pasaría con esa conexión si todos los servidores de nombres fallaran?

Una forma sencilla de realizar consultas de nombres es a través del comando `host`. Revisa el manual del comando y observa la respuesta al preguntar por los siguientes nombres:

129. *hendrix*
130. *moodle* (¿por qué *hendrix* funciona y *moodle* no?)
131. *moodle.unizar.es*
132. *hendrix*. (no te dejes el `.` final)
133. *moodle.unizar.es*. (no te dejes el `.` final) (¿por qué *moodle.unizar.es* funciona y *hendrix*. no?)
134. *unizar.es*. (no te dejes el `.` final)
135. *unizar.es* (¿por qué *unizar.es* funciona con y sin `.` final?)
136. *www.unizar.es*.
137. Además de traducción de direcciones, ¿qué otras dos informaciones te han aparecido en algunas de las consultas anteriores?

El comando `dig` también permite realizar consultas de nombres de forma similar a `host`, pero proporcionando muchos más detalles. Los principales registros que DNS maneja son:

- A (Address), define la dirección IPv4
- AAAA (Address), define la dirección IPv6
- NS (Name Server), define los servidores DNS
- MX (Mail eXchanger), define los servidores de correo

- CNAME (Canonical Name), permite definir alias de otros nombres
- SOA (Start Of Authority), contiene información sobre el servidor DNS primario
- LOC (LOCation), define la localización
- TXT (TeXT): almacena cualquier información

138. Ejecuta `dig ANY unizar.es` y compara el resultado con la información obtenida con `host`.

En cuanto a los servidores de nombres de dominio, el uso de IPv6 simplemente implica manejar un nuevo *tipo* de información: las direcciones IPv6. Si las direcciones de IPv4 (32 bits) se simbolizan con tipo A (*address*), las direcciones IPv6 (4 veces mayores) se simbolizan con tipo AAAA.

139. Pregunta por tipos AAAA en *google* (`dig AAAA google.com`). ¿Responde con una dirección IPv6?

Obtén la dirección IP de los siguientes nombres:

140. `ipv6.google.com`.

141. `www.v6.facebook.com`.

142. ¿Qué obtienes si haces `ping` de `ipv6.google.com`? ¿Por qué?

143. ¿Y si haces `ping6`? ¿Por qué?

## 6.7. Estado de puertos

La herramienta `netstat` permite visualizar múltiples datos de red del ordenador en el que nos encontramos, incluyendo información de rutas e interfaces. No obstante, se usa particularmente para mostrar el estado de los puertos TCP y UDP. Al igual que otros comandos que has visto en la asignatura, esta herramienta está siendo sustituida, en este caso por el comando `ss` (*socket statistics*) en GNU/Linux.

144. ¿Qué hace `netstat` con el argumento `-t`? ¿Qué muestran las columnas *Local Address* y *Foreign Address*?

145. Prueba ahora `ss -t` ¿Se parece?

146. ¿Qué hace el argumento `-l`? ¿Por qué se muestran asteriscos en la columna *Foreign Address* (*Peer Address* en `ss`)?

147. ¿Qué hace el argumento `-a`? Observa que además de sockets TCP/UDP aparecen también los *UNIX domain sockets* asociados a un fichero, que habrás estudiado en Sistemas Operativos.

148. Lanza un netcat como servidor TCP y en otro terminal obtén un listado de los sockets TCP que estén en modo *listen* (e.g. `ss -l -t`). ¿Puedes localizar el socket del netcat en el listado? ¿Qué aparece en la columna *state*?

149. Lanza ahora un netcat que se conecte con el netcat anterior. Localiza su entrada con `netstat` o `ss` (e.g. `ss -t -a`). ¿Cuántas veces aparece? ¿En qué estado está ahora el socket? ¿Qué puertos se están utilizando en esa conexión?

150. Si pulsas Ctrl+C en uno de los netcat, ¿finaliza la conexión de ese netcat o de los dos? ¿Por qué?

151. Si inmediatamente después de cerrar la conexión ejecutas `ss -a -t` ¿en qué estado aparece la conexión?

152. Lanza ahora un netcat como servidor UDP y en otro terminal obtén un listado de los sockets UDP (`ss -u -l`). ¿Puedes localizar el socket del netcat en el listado?

153. Lanza ahora un netcat UDP para interactuar con el netcat anterior, pero sin enviar ningún texto entre ellos. ¿Cuántas entradas aparecen en el listado referidas a los sockets utilizados (`ss -u -l -a`)? ¿Cuál es su estado?
154. Escribe algo en el netcat *servidor*. ¿Se transmite al cliente? ¿Por qué? ¿Ha cambiado la información que muestra `ss`?
155. Escribe algo *distinto* ahora en el netcat *cliente*. ¿Qué ha pasado? ¿Ha cambiado la información que muestra `ss`?
156. Pulsa Ctrl+C para finalizar el servidor. ¿Ha finalizado el cliente automáticamente? ¿Cuántas entradas aparecen ahora en el listado? ¿En qué estado?
157. Sin cancelar el cliente anterior, lanza un nuevo servidor netcat UDP en el mismo puerto. ¿Si escribes algo en el cliente, lo recibe el nuevo servidor? ¿Por qué?

Al igual que con los parámetros de la capa de red anteriores, puedes mostrar también los valores de parámetros de capa de transporte mediante el comando `sysctl`.

158. ¿Qué valor tiene el factor de escalado (opcional) de la ventana anunciada del protocolo TCP (`sysctl net.ipv4.tcp.window_scaling`)?
159. ¿Qué valores (mínimo, por defecto y máximo) tiene la ventana de recepción del protocolo TCP (`sysctl net.ipv4.tcp.rmem`)?
160. ¿Qué valores (mínimo, por defecto y máximo) tiene la ventana de emisión del protocolo TCP (`sysctl net.ipv4.tcp.wmem`)?

## 6.8. Interacción con protocolos de aplicación

En prácticas anteriores se ha utilizado la herramienta *netcat* para observar y verificar el comportamiento de aplicaciones como la de enviar vocales. Esto es generalizable para cualquier protocolo de aplicación cuyas comunicaciones estén basadas en texto, por ejemplo el protocolo HTTP.

161. Lanza el netcat como servidor TCP en tu equipo, por ejemplo en el puerto 32002. A continuación introduce `http://pon-aquí-tu-direccion-ip:32002/` en un navegador (usando la dirección IP de tu equipo.) ¿Qué es lo que ha aparecido en el netcat?
162. Lanza ahora netcat como cliente, conectando con `www.unizar.es` por el puerto 80. Realiza una solicitud web manualmente a través de netcat, es decir, tecleando el texto que debería entender el servidor web. Observa lo que aparece en pantalla y justifica por qué es diferente a la respuesta anterior.

## 6.9. Herramienta Nmap

Otra herramienta muy interesante es *nmap*, que permite explorar y analizar muchos aspectos de las redes. Muchas exploraciones las realiza usando los sockets de forma «normal». En cambio, para hacer ciertas exploraciones menos convencionales, esta herramienta construye «manualmente» los paquetes, es decir, rellenando los campos de las cabeceras con ciertos valores (correctos o no). En general los sistemas no permiten que cualquier usuario pueda hacer esto, así que para realizar ciertas exploraciones es necesario tener permisos de administrador. Eso sí, ten en cuenta que cualquier exploración implica que la máquina explorada debe responder, con lo que además de saber que está siendo explorada conoce la dirección de quien la está explorando.

163. Revisa el manual del comando `nmap` y haz una exploración de la red del laboratorio mediante pings

## 6.10. ¿Sabías que...?

- Aunque la mayoría de servicios y contenidos alojados en «Internet IPv6» también lo están en «Internet IPv4», hay algunos (cada vez más) que sólo están disponibles usando IPv6. Ciertas empresas proporcionan servicios gratuitos de túnel IPv6 sobre IPv4, para acceder a «Internet IPv6» desde proveedores de servicios que sólo proporcionan IPv4. Por ejemplo puedes configurar un túnel de estas características en <http://www.tunnelbroker.net/>
- La herramienta **nmap** es una de las preferidas por los hackers en el cine desde su aparición en *Matrix Reloaded* (<http://nmap.org/movies.html>).