

# REDES Y SO

Diciembre 2021

## Índice

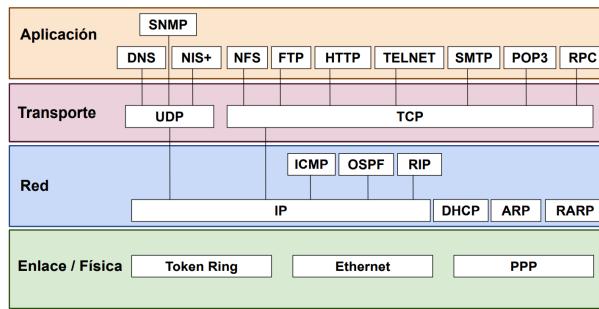
|   |           |
|---|-----------|
| <b>1. REDES</b>   | <b>3</b>  |
| 1.1. IPv4. Protocolo DHCP . . . . .                       | 3         |
| 1.1.1. Direcciones y Formato del Datagrama IP . . . . .   | 3         |
| 1.1.2. Reenvío de paquetes . . . . .                      | 5         |
| 1.1.3. Protocolo DHCP . . . . .                           | 6         |
| 1.2. Protocolo TCP . . . . .                              | 6         |
| 1.2.1. Ventanas y Formato del Segmento TCP . . . . .      | 6         |
| 1.2.2. Fases de la conexión . . . . .                     | 8         |
| 1.2.3. Control de errores . . . . .                       | 10        |
| 1.2.4. Control del flujo . . . . .                        | 11        |
| 1.2.5. Control de la congestión . . . . .                 | 12        |
| 1.3. Servicios de Red . . . . .                           | 13        |
| 1.3.1. Filtrado de paquetes . . . . .                     | 13        |
| 1.3.2. NAT . . . . .                                      | 14        |
| 1.3.3. DNS . . . . .                                      | 15        |
| 1.4. IPv6 . . . . .                                       | 17        |
| 1.4.1. Direcciones . . . . .                              | 18        |
| 1.4.2. Datagrama . . . . .                                | 20        |
| 1.4.3. ICMPv6 . . . . .                                   | 21        |
| 1.5. Encaminamiento en Internet . . . . .                 | 22        |
| 1.5.1. RIP . . . . .                                      | 22        |
| 1.5.2. OSPF . . . . .                                     | 24        |
| 1.5.3. BGP . . . . .                                      | 25        |
| 1.5.4. Arquitectura de Internet . . . . .                 | 26        |
| <b>2. SISTEMA OPERATIVO</b>                               | <b>27</b> |
| 2.1. Introducción a la Programación de Sistemas . . . . . | 27        |
| 2.2. Sistemas de Ficheros . . . . .                       | 29        |
| 2.3. Gestión de Procesos . . . . .                        | 32        |
| 2.4. Señales . . . . .                                    | 34        |
| 2.5. Tuberías . . . . .                                   | 35        |
| 2.6. Programación con Sockets . . . . .                   | 36        |

|        |                                  |    |
|--------|----------------------------------|----|
| 2.6.1. | Direcciones de Sockets . . . . . | 37 |
| 2.6.2. | Socket UDP . . . . .             | 38 |
| 2.6.3. | Socket TCP . . . . .             | 39 |
| 2.6.4. | Otras opciones . . . . .         | 39 |

# 1. REDES

## 1.1. IPv4. Protocolo DHCP

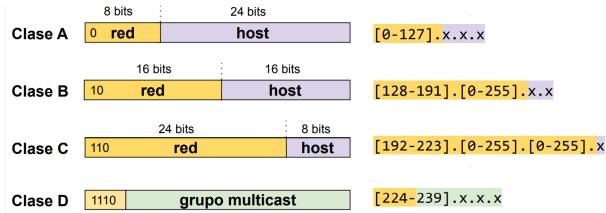
En cursos anteriores hemos visto en profundidad la capa física, con los diferentes tipos de redes y sus topologías. En estos apuntes nos centraremos en la capa de red y de transporte. El protocolo de Internet IP proporciona un servicio básico de entrega de paquetes. Es un protocolo no orientado a conexión, es decir, no verifica que los paquetes lleguen en orden, contengan errores o se pierdan. IP ofrece un espacio de direcciones, encapsulado de datos, fragmentación y reenvío de paquetes.



### 1.1.1. Direcciones y Formato del Datagrama IP

Las direcciones IPv4 son de 32 bits (4 bytes) y se escriben en notación de punto :  $X_1.X_2.X_3.X_4$  con cada  $X_i$  representando 8 bits (0-255). La dirección se divide en 2 partes: el **prefijo de red** y el **identificador del host**. La propia red tiene la dirección con el identificador de host a 0 y se usa en las tablas de rutas, nunca como dirección de destino ni se asigna a un host concreto. Si el identificador de host es 11...1, la dirección es de **broadcast** y el paquete se envía a todos los hosts de la red. Existen 2 tipos de direccionamientos diferentes:

- **De clases:** Las direcciones se agrupan en clases de red. Según el prefijo de red pertenece a una u otra. Con este sistema aparece el problema de agotamiento de direcciones.

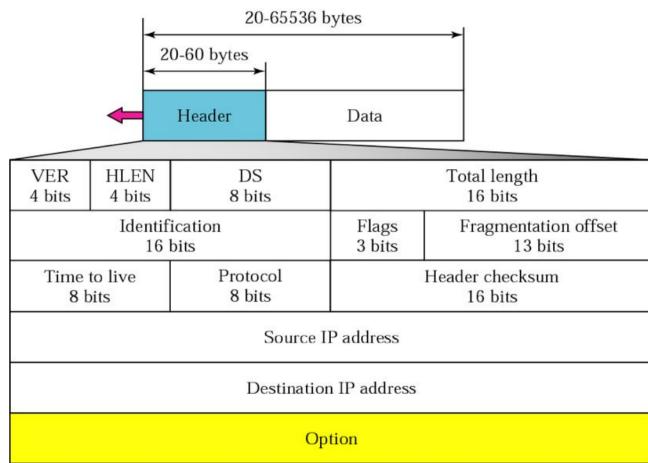


- **CIDR:** No hay clases prefijadas en función del prefijo de red.  $X_1.X_2.X_3.X_4/Y$

donde  $Y$  indica la longitud del prefijo. Para conocer el prefijo de red de una dirección IP se utiliza la **máscara de red**, con  $Y$  1's y el resto 0.

Otras direcciones especiales a parte de la de red y la broadcast son: loopback o de bucle interno, 127.x.y.z, normalmente 127.0.0.1; privadas, sin acceso a Internet, 1 de clase A (10.), 16 de clase B (172.16. – 172.31.) y 256 de clase C (192.168.); y multicast para referirse a un grupo de hosts en un segmento de red, 224.0.0.1 para todos los hosts o 224.0.0.2 para todos los routers.

En cuanto al datagrama IP, este puede ser de 20-65536 bytes. La cabecera ocupa entre 20-60 bytes. Los primeros 20 bytes están fijados en diferentes campos, mientras que los últimos 40 son del campo Options, muy poco usado.



La segunda fila hace referencia a la fragmentación de paquetes. El campo Identification guarda el identificador del grupo al que pertenece el fragmento. Los 3 bits de Flags: 1º reservado, 2º para indicar que no hay fragmentación (DF) y 3º para indicar más fragmentación (MF). El campo de la fila anterior, DS, está relacionado con el tipo de servicio que se ofrece (Differentiated Service).

Para traducir las direcciones IP a las direcciones MAC dentro de una red, se utiliza el protocolo **ARP**. Cada host o máquina mantiene una tabla ARP con las traducciones de las últimas conexiones. Recordamos que el datagrama IP va contenido en los datos del datagrama Ethernet (en realidad se dice trama Ethernet), de la capa física. Tener en cuenta que en la cabecera del datagrama ARP, para utilizar IP se introducen los siguientes valores: HW type = 1 (Ethernet, WiFi), Protocol type = 0x0800 (IP), HW length = 6, Protocol length = 4.

### 1.1.2. Reenvío de paquetes

Los hosts tienen dos formas de enviar un paquete a otra dirección: por dirección destino, mediante la **tabla de rutas** o por etiqueta, en la que los paquetes van etiquetados y siguen el mismo circuito. Esta última forma está orientada a conexión, mientras que la de dirección destino no. Nos vamos a centrar en la primera forma.

Las entradas de la tabla de rutas están compuestas por: dirección, que puede ser de red, un host o default(0.0.0.0/0); la máscara de red por si la dirección es CIDR; la interfaz y/o el siguiente salto en caso de enlace indirecto; y la métrica. Para decidir a qué dirección de la tabla de rutas se envía el paquete, se escoge la más específica (mayor longitud del prefijo) que coincida con la red de la dirección destino. En caso de empate, gana la de menor métrica. Además, en la tabla de rutas hay 3 tipos de entradas:

- Rutas por defecto: se establece en el proceso de autoconfiguración (ver DHCP) o de forma manual (ip route add default)
- Rutas directas: redes configuradas en cada interfaz
- Rutas específicas: configuradas de forma manual (ip route add *dest*;

```
$ ip route
default via 192.168.0.1 dev enp0s31f6 proto dhcp src 192.168.0.249 metric 202
default via 192.168.0.1 dev wlp3s0 proto dhcp src 192.168.0.223 metric 303
10.3.0.0/16 dev enp0s31f6 proto kernel scope link src 10.0.0.24
192.168.0.0/24 dev enp0s31f6 proto dhcp scope link src 192.168.0.249 metric 202
192.168.0.0/24 dev wlp3s0 proto dhcp scope link src 192.168.0.223 metric 303
```

Ruta por defecto  
Ruta instalada por DHCP (servidor 192.168.0.249)  
Ruta directa al configurar el interfaz  
Dos rutas a la red 192.168.0.0/24 se **prefiere la de menor métrica** (LAN > WiFi)

Para enviar mensajes de control de la red se utiliza el protocolo **ICMP**. Estos pueden ser de error o informativos. En el campo Tipo ICMP de la cabecera del datagrama se especifica de qué tipo son:

| 0                              |                         | 8      |              | 16                             |                     | 32                      |                      |  |  |  |  |
|--------------------------------|-------------------------|--------|--------------|--------------------------------|---------------------|-------------------------|----------------------|--|--|--|--|
| Tipo ICMP                      |                         | Código |              | Checksum                       |                     | ICMP Datos (Opcionales) |                      |  |  |  |  |
| <b>Mensajes Error</b>          |                         |        |              | <b>Mensajes Informativos</b>   |                     |                         |                      |  |  |  |  |
| <b>Tipo</b> <b>Significado</b> |                         |        |              | <b>Tipo</b> <b>Significado</b> |                     |                         |                      |  |  |  |  |
| 3                              | Destination Unreachable | 0      | Echo Reply   | 4                              | Source Quench       | 5                       | Redirect             |  |  |  |  |
| 11                             | Time Exceeded           | 8      | Echo Request | 9                              | Router Solicitation | 10                      | Router Advertisement |  |  |  |  |
| 12                             | Parameter Problem       |        |              |                                |                     |                         |                      |  |  |  |  |

Un caso especial es la orden ping, que envía Echo Request y espera a Echo Reply. Se utiliza para ver si un host es alcanzable. Un Request y su correspondiente Reply tienen el mismo identificador.

### 1.1.3. Protocolo DHCP

Este protocolo proporciona configuración automática de los parámetros de red: dirección IP y máscara, Router predeterminado, Servidor DNS y otros parámetros de red. Actúa como un protocolo de cliente/servidor, en el que el cliente solicita a múltiples servidores posibles configuraciones, para acabar escogiendo la definitiva. El servidor utiliza el puerto 67 y el cliente el 68. Además, en este protocolo se realizan comprobaciones de errores mediante Checksums.

El proceso de obtener una configuración es el siguiente:

1. El cliente descubre los posibles servidores mediante **DHCPDISCOVER**. Es un broadcast.
2. Los servidores responden enviando ofertas de configuraciones mediante **DHCPOFFER**.
3. El cliente solicita una de esas ofertas mediante **DHCPREQUEST**. Debe ser broadcast para que se enteren todos los servidores.
4. El servidor escogido confirma la configuración con los parámetros definitivos, mediante **DHCPACK**. Este mensaje también es broadcast.
5. Para extender el tiempo de cesión de los parámetros, se utiliza **DHCPREQUEST Y DHCPACK**.
6. Cuando el cliente finaliza, envía un mensaje **DHCPRELEASE**.

En cuanto al datagrama DHCP, destacar que el código de request es 0x01 y el de pack 0x02, Trans ID es la correspondencia entre solicitud y respuesta, y Your IP es la IP ofrecida por el servidor. En el Campo Options se almacenan otras informaciones de configuración como los servidores DNS o el tiempo de cesión.

## 1.2. Protocolo TCP

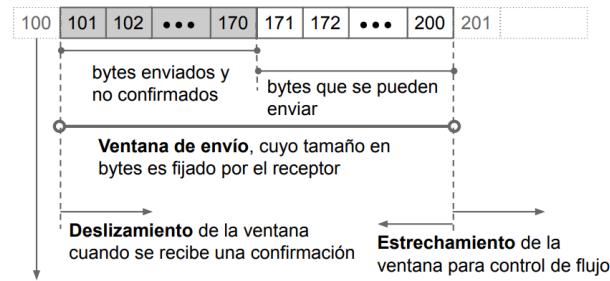
En la capa de transporte, se utiliza el protocolo TCP. Ofrece conexión lógica entre 2 procesos, a través de puertos. Por lo tanto, está orientado a conexión. Existen 3 fases en la transmisión: establecimiento, transferencia de bytes y cierre. La unidad de transferencia es el segmento TCP y al ser orientado a conexión, es un protocolo fiable. Esto significa que incluye mecanismos de control de errores como checksum o numeración de segmentos y confirmación de estos.

### 1.2.1. Ventanas y Formato del Segmento TCP

El número del primer byte del segmento se denomina **SEQ** y sirve para identificar el segmento. Una vez que el receptor reciba dicho segmento, ha de confirmar al emisor que lo ha recibido. Para ello utiliza el **ACK**, que es el número del siguiente byte que espera recibir. Se envía junto con datos y es

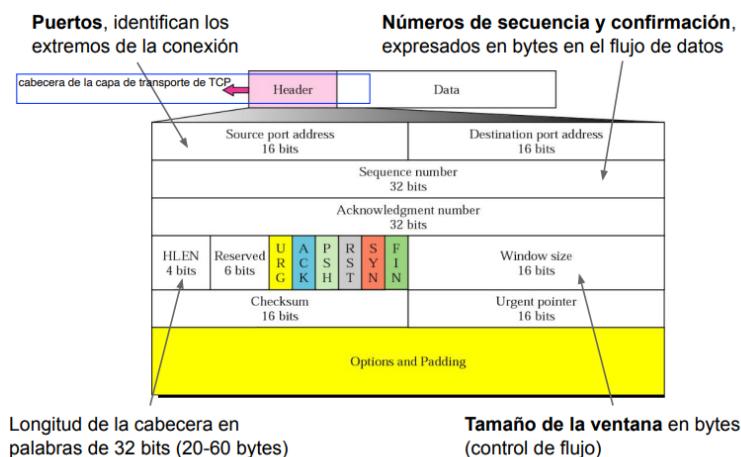
acumulativo, es decir, confirma todos los anteriores a él. El receptor no ha de saturarse, por lo que existe un máximo de bytes que se pueden enviar sin ser confirmados. Dichos bytes vienen dados por 2 ventanas deslizantes.

- **Ventana de Envío:** La ventana se desliza cuando se confirman bytes. Está en el emisor. El tamaño de la ventana es fijado por el receptor.



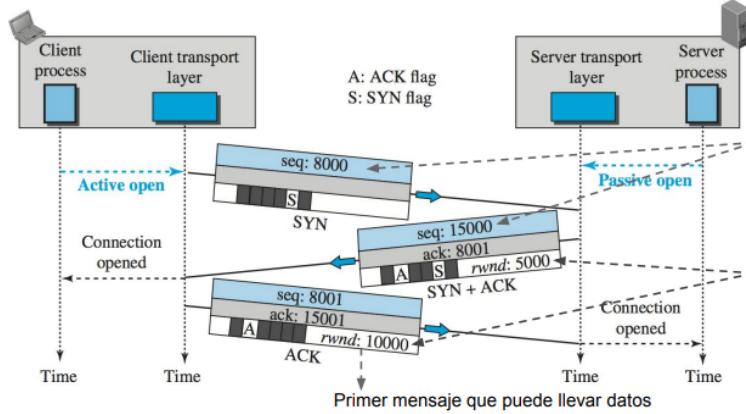
- **Ventana de Recepción:** La ventana se desliza cuando se consumen bytes ya confirmados. Está en el receptor. Marca el número de bytes que se pueden recibir. Se utiliza en el control de flujo.

En cuanto a la cabecera del segmento TCP, destacan los siguientes flags: **SYN**, **FIN** y **RST** para iniciar, cerrar y abortar conexión, **ACK** si el número de confirmación es válido (todos menos el primer segmento de SYN lo llevan), **PSH** si deben ser entregados directamente a la aplicación, y **URG** si transporta datos urgentes hasta el nº de bytes marcado por Urgent pointer.

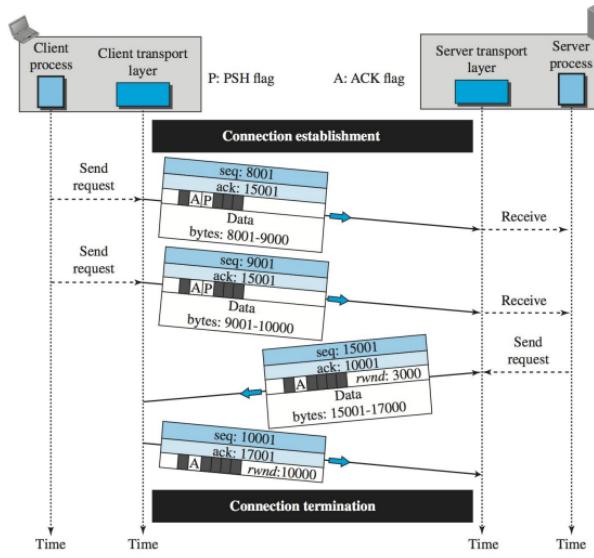


### 1.2.2. Fases de la conexión

1. **Establecimiento:** cliente envía SYN, servidor responde con SYN + ACK y cliente devuelve ACK. El ACK del cliente puede contener datos. En los ACK se fijan los tamaños de las ventanas de recepción y de envío del otro. El ataque SYN Flooding se basa en enviar muchos mensajes con el flag de SYN activo.

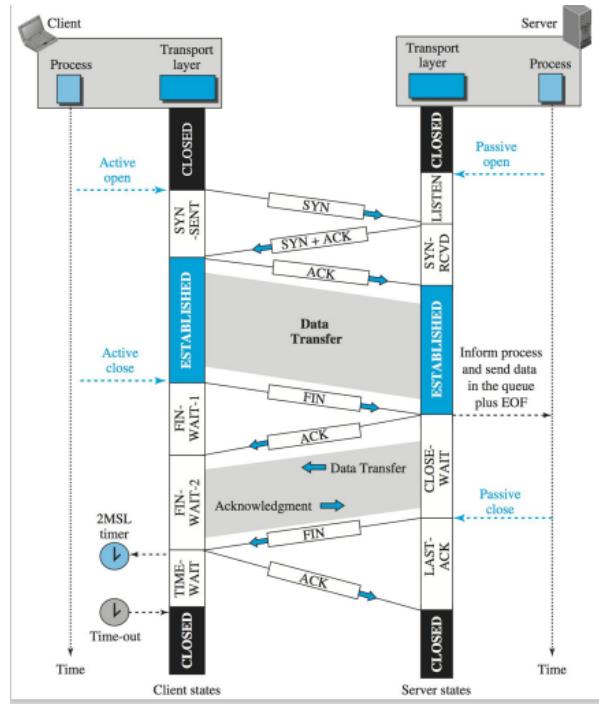
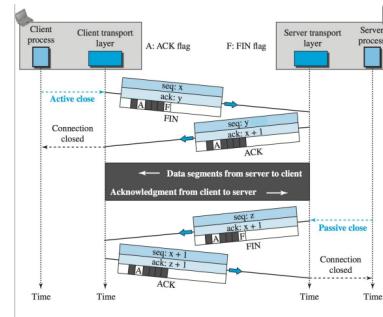


2. **Transferencia:** El campo del tamaño de la ventana debe reflejar la perdida de espacio, es decir, los bytes confirmados pero no consumidos. El tamaño máximo del segmento **MSS** lo fija cada extremo de la conexión en el campo Options.



### 3. Cierre: Puede ser de 2 formas

- 3-way: análogo al establecimiento pero con FIN. El último ACK no lleva datos, pero los de FIN sí pueden contener datos.
- 4-way: el servidor confirma el FIN pero no finaliza hasta que termine de enviar todos los datos. En ese momento lanza FIN y cliente confirma.

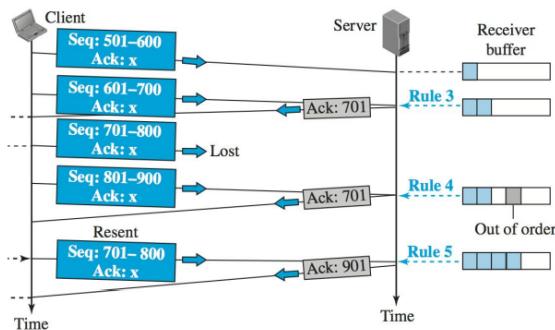


### 1.2.3. Control de errores

Para controlar los errores se utilizan las ventanas deslizantes y los ACK. Existen 6 reglas para las confirmaciones:

- Todo segmento debe llevar nº de confirmación indicando el siguiente byte esperado. Confirma todos los anteriores.
- Se puede retrasar una confirmación 500 ms en caso de que no tenga datos para enviar.
- Solo puede retrasarse la confirmación de un segmento.
- Segmentos fuera de orden, que llenan huecos y duplicados se confirman inmediatamente con el byte que esperábamos en realidad.
- Segmentos que llenan huecos y duplicados se confirman inmediatamente

Se pueden introducir **SACK** para confirmar segmentos fuera de orden. Son informativos y no sustituyen a los ACK.

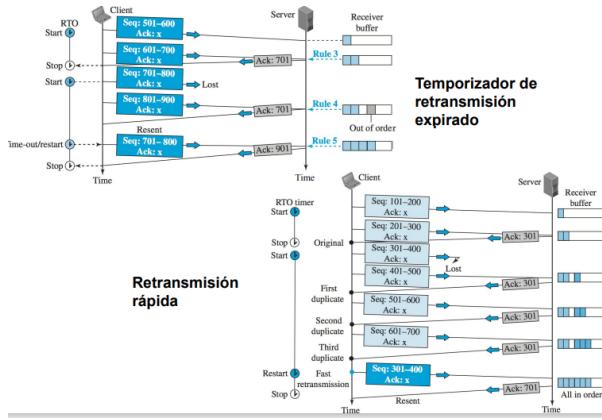


Una vez se ha detectado un error por la falta de concordancia en los ACK, se retransmite el primer segmento sin confirmar de la ventana de envío. Existen 2 mecanismos que lanzan la retransmisión:

- **Temporizador de retransmisión RTO:** se inicia cuando se envía un segmento y se para al recibir la confirmación. Si expira se lanza la retransmisión. Cada mensaje utiliza un RTO propio. Se calcula a partir de diferentes medidas del retardo de red RTT:
  - **Medido o RTTm:** tiempo desde que se envía hasta que se confirma. Puede sufrir grandes fluctuaciones.
  - **Suavizado o RTTs:** para evitar las fluctuaciones es una media ponderada del último RTTs y de RTTm.
  - **Desviación del RTT o RTTd:** considera la variación del RTT. Combinación de las 3.

El algoritmo de Jacobson utiliza RTTs, el de Jacobson/Karels utiliza RTTs y RTTd y el de Karn duplica el RTO cuando se retransmite, para prevenir ambigüedades de RTTm.

- **Retransmisión rápida:** tras recibir 3 ACK's duplicados.

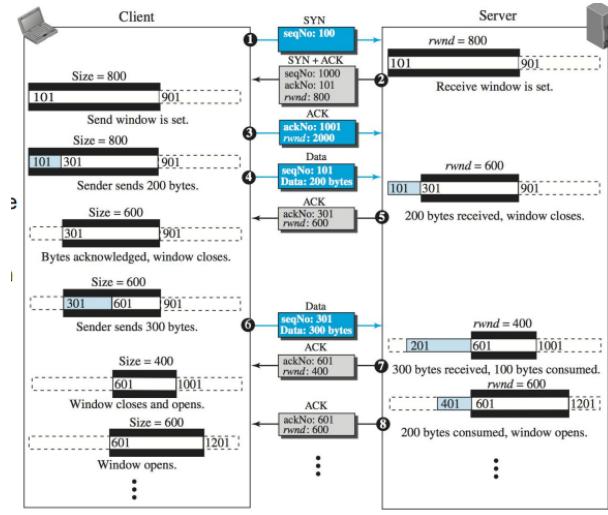


Se utilizan otros temporizadores:

- **Keepalive:** evitar conexiones que perduren indefinidamente.
- **TIME-WAIT:** es el doble de la vida máxima del segmento **MSL** y se aplica en el cliente en el cierre 4-way, tras el último ACK. Impide conexión con los mismos parámetros y permite que expiren los segmentos duplicados.
- **De Persistencia:** asociado a tamaño de ventana 0. Recuperar pérdida de ACK tras cambio de tamaño de ventana.

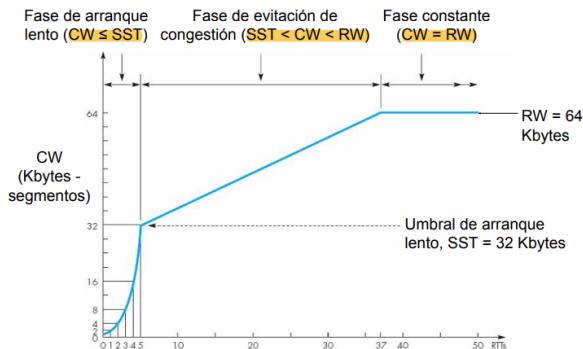
#### 1.2.4. Control del flujo

Evitar sobrecargar al receptor. Se utiliza la ventana de recepción, anunciada en cada ACK. Cuando se consumen datos, el tamaño de la ventana aumenta y se notifica al emisor con un ACK. Si el emisor envía datos muy lento o el receptor los consume igual de lento, este problema se denomina el **síndrome de la ventana trivial**. El algoritmo de Nagle propone una solución para el emisor: envía el primer mensaje y espera hasta que reciba un ACK del receptor, se acumule un segmento completo o expire el RTO. Si se da en el receptor, significa que se anuncian ventanas de tamaño muy reducido. Para solucionarlo, el algoritmo de Clark anuncia tamaño 0 hasta que haya espacio para segmento completo o se haya liberado mitad del buffer de recepción. Otra posibilidad es retrasar ACK, pero puede provocar retransmisiones innecesarias.



### 1.2.5. Control de la congestión

Cuando se pierden paquetes, suele ser por una congestión en algún punto de la red. El emisor regula el envío de segmentos con el ritmo de llegada de confirmaciones. Utiliza una **ventana de congestión (CW)**. Es complementaria a la de recepción RW, siempre se da  $CW \leq RW$  y los bytes que se pueden enviar es el mínimo de ambas. Cuando hay congestión disminuye y cuando desaparece aumenta. Experimenta las siguientes fases:



- **Fase de arranque lento:**  $CW = 1$  y va aumentando por cada segmento enviado y confirmado, experimentando un crecimiento exponencial. Cuando llega al **umbral de arranque lento (SST)** pasa a la siguiente fase.
- **Fase de evitación de congestión:**  $CW$  aumenta cada vez que se envía y confirma una ventana completa, es decir,  $CW$  segmentos. Crecimiento

lineal. Cuando  $CW = RW$  pasa a la última fase.

- **Fase constante:**  $CW = RW$  hasta que no se confirme algún mensaje.

Cuando llegan 3 ACK's duplicados o expira el RTO la red sufre una congestión. En el caso de los ACK's duplicados, el nivel de congestión es leve porque siguen llegando confirmaciones. Se activa el método de recuperación rápida, en el que se reducen CW y SST a la mitad de CW. En el caso del RTO, el nivel de congestión es elevado y se ejecuta el arranque lento con SST reducido a la mitad del CW.

### 1.3. Servicios de Red

#### 1.3.1. Filtrado de paquetes

El **Firewall** es un programa del kernel del SO que analiza el tránsito de datos y decide que paquetes se entregan o reciben, entre otras cosas. Pueden ser stateless o stateful en función de si tienen en cuenta el estado de la conexión, así como de red o de aplicación si checkean cabeceras de los datagramas de red o de aplicación. La estructura del firewall está basada en tablas:

- **Reglas:** definen qué hacer (ej. descartar o aceptar) con un paquete que cumple unos determinados criterios (ej. puerto origen, dirección IP destino...)
- **Cadenas:** listas de reglas que se aplican en orden a un paquete. Una regla puede enviar el paquete a otra cadena, siempre ha de aplicarse al menos una cadena a cada paquete y si este no encaja en ninguna regla se aplica la política de la cadena.
- **Tablas:** listas de cadenas. 3 ejemplos de tablas:
  - **Tabla Filter:** bloquea o permite tránsito de paquetes. Todo paquete atraviesa esta tabla.
    - CADENA INPUT: paquetes de entrada.
    - CADENA OUTPUT: paquetes que salen del sistema.
    - CADENA FORWARD: paquetes que atraviesan el sistema (encaminados).
  - **Tabla NAT:** traducir direcciones (origen y destino), puertos...
    - CADENA PREROUTING: paquetes de entrada antes de decisión de encaminamiento. Usada en DNAT
    - CADENA POSTROUTING: paquetes de salida después de decisión de encaminamiento. Usada en SNAT.
    - CADENA OUTPUT: paquetes de salida generados por el sistema.
  - **Tabla Mangle:** modificar otros campos del paquete como TOS o MSS.

Algunos ejemplos de reglas definidas con el comando iptables son

| Opción/Ejemplo                              | Significado   |
|---|---|
| -A INPUT<br>-A OUTPUT<br>-A FORWARD         | Añade regla a cadena de entrada<br>Añade regla a cadena de salida<br>Añade regla a la cadena forward (sólo en caso de routers)                      |
| -s 192.168.1.1<br>-d 140.10.15.1            | Filtrado por dirección IP origen<br>Filtrado por dirección IP destino   |
| -p tcp<br>-p udp<br>-p icmp                 | Filtrado de paquetes TCP<br>Filtrado de paquetes UDP<br>Filtrado de paquetes ICMP   |
| --sport 3000<br>--dport 80<br>--icmp_type 8 | Filtrado por nº de puerto origen (para TCP o UDP)<br>Filtrado por nº de puerto destino (para TCP o UDP)<br>Filtrado por tipo de mensaje (para ICMP) |
| -i eth0<br>-o eth1                          | Filtrado por interfaz de red de entrada<br>Filtrado por interfaz de red de salida   |

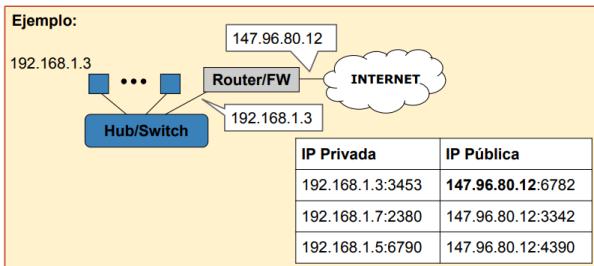
  

| Opción                       | Significado   |
|------------------------------|---|
| -m state --state NEW         | Filtrado de paquetes correspondientes a conexiones nuevas (el primer paquete) |
| -m state --state ESTABLISHED | Filtrado de paquetes correspondientes a conexiones ya establecidas            |
| -m state --state RELATED     | Filtrado de paquetes relacionados con otras conexiones existentes             |
| -m state --state INVALID     | Filtrado de paquetes que no pertenecen a ninguno de los estados anteriores    |

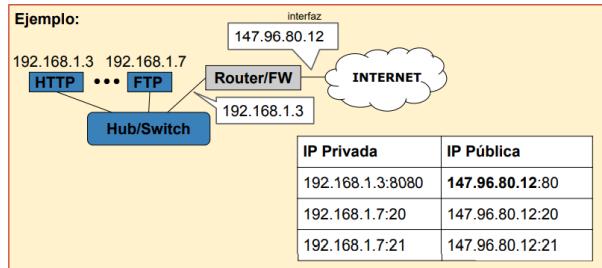
### 1.3.2. NAT

Con la tabla NAT ya hemos visto que se traducen direcciones. Esto se utiliza para aliviar el problema del nº limitado de direcciones IPv4. La idea es tener direcciones privadas que no puedan conectarse a Internet, a no ser que se traduzcan a una dirección pública. Las traducciones pueden ser estáticas, es decir, una biyección entre IP privada y IP pública o pueden ser dinámicas, cuando hay menos IP públicas y algunas privadas no tienen acceso a Internet.

Otra opción es el **NAPT**, que consiste en traducir N IP privadas a una pública con N puertos diferentes. Cada máquina con IP privada se conecta a un puerto diferente del router. La conexión ha de iniciarse desde estas máquinas y en el router debe aplicarse iptables con SNAT en la cadena POSTROUTING para cambiar dirección origen (NAPT con dirección fija). Otra opción es aplicar MASQUERADE, que permite traducción dinámica en vez de fija (usa dirección IP de interfaz como IP origen).



Desde el exterior, todas las máquinas tienen la misma dirección pública. El router se encarga de traducir y reenviar los paquetes a las N direcciones privadas. En la cadena PREROUTING se aplica DNAT para la traducción.



### 1.3.3. DNS

DNS es un servicio de red que permite hacer traducciones entre nombres de dominio e IP. Gracias a esto podemos escribir en Internet nombres de páginas webs en vez de su dirección IP, que sería mucho más lioso. DNS consiste en una BD distribuida a modo de árbol y un protocolo de consultas e intercambio de información. La BD es distribuida porque cada sitio guarda información únicamente de sus sistemas. Además, es una estructura jerárquica. Tenemos un dominio raíz y 13 servidores de nombres de 1 nivel.

Antes de visualizar como es la BD y su funcionamiento, debemos conocer algunos conceptos:

- **Zonas y Dominios:** Un dominio es un subárbol del conjunto de todos los nombres de dominio posibles. Dentro de un dominio hay varias zonas, que son las organizaciones encargadas de gestionar un subconjunto del dominio. Las zonas guardan los servidores autoritarios y los servidores de subdominios delegados.
- **Nombres de dominio:** lista de nombres o etiquetas que representan la jerarquía desde el nivel más bajo hasta la raíz, separadas por un punto. Un ejemplo sería: www.ucm.es. Se suele omitir la raíz y en **FQDN** se escribe el último punto. EL FQDN puede ocupar 256 caracteres y cada sección un máximo de 63. No hay diferencias entre mayúsculas y minúsculas y deben ser caracteres alfanuméricos y guiones.

Para hacer la traducción inversa (IP -> nombre dominio), existe un dominio **in-addr.arpa.** (IPv4). La dirección IP se invierte para que la parte más significativa esté a la derecha. Por ejemplo: 63.173.189.1 -> 1.189.173.63.in-addr.arpa.

- **Servidores de Nombres:** servidores encargados de generar todo el servicio red DNS. Existen varios tipos:

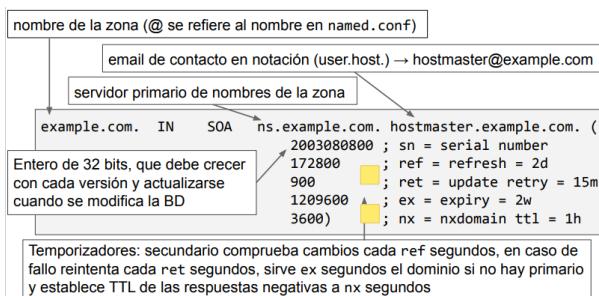
- **Autoritativos:** representan oficialmente a la zona. Los primarios guardan la copia de la BD y los secundarios obtienen la BD de los primarios. Siempre debe haber un primario y al menos un secundario.
- **De cache:** guardan resultados de consultas para mejorar en eficiencia. No son autoritativos ni guardan ningún registro DNS propio.
- **Recursivos y no recursivos:** estos están relacionados con las consultas. Los recursivos resuelven las referencias devueltas por los no recursivos hasta obtener la respuesta. Los no recursivos devuelven referencias a otros servidores en caso de no tener la respuesta. Los servidores autoritativos suelen ser no recursivos. Los clientes deben usar servidores recursivos.

La BD está organizada en registros, **RR**. Estos están almacenados en los ficheros de zona, en los servidores primarios. Son los que se intercambian y cachean. El formato es

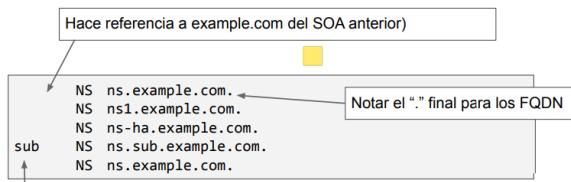
*[nombre] [ttl] [clase] tipo datos*

donde el nombre identifica el registro y suele ser el nombre de dominio o host, ttl el tiempo cacheable, clase IN de Internet, tipo y datos depende del registro. Hay de varios tipos, clasificados en 4 grupos (Zona, Básicos, Seguridad y Opcionales):

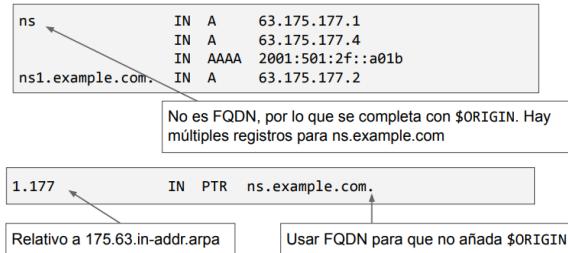
- **SOA:** marcar el comienzo de definición de una zona. Especificar el servidor primario de la zona y temporizadores. Los servidores suelen tener 2 zonas, una directa para traducción de nombres a IP y la inversa.



- **NS:** especificar los servidores secundarios de una zona, así como los servidores de nombre de subdominios delegados (para que funcione la delegación). Estos registros suelen ir tras el SOA.

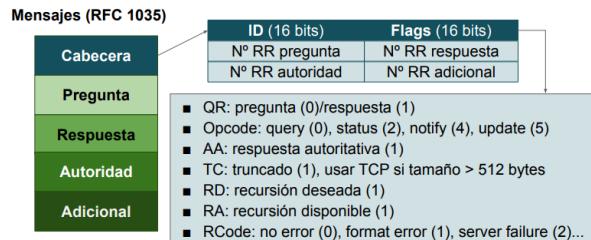


- **A,PTR**: A o AAAA (IPv6) incluye la traducción directa. Todos los servidores especificados deben tener traducción. El registro PTR es para la traducción inversa y se almacena en otra zona distinta.



- **MX**: establecer mail de un servidor. Encaminar los mensajes eficientemente. Se asigna una prioridad decreciente.
- **CNAME**: otorgar una alias a un nombre de dominio. Por lo tanto, siempre deben apuntar a un dominio. Dicho alias no puede tener otros registros y MX, NS no pueden apuntar al alias.

Para llevar a cabo consultas, se utiliza un datagrama especial, propio de la capa de transporte. Si la respuesta contiene más de 512 bytes o son transferencias de zona, se aplica TCP, en caso contrario UDP.



El funcionamiento está basado en la delegación de los servidores no recursivos y en la resolución de los recursivos. Para mejorar la eficiencia se introducen los servidores de caché, que almacenan los resultados durante ttl. La cache puede dar una respuesta negativa cuando la búsqueda falla (el servidor no responde o no encaja el nombre de dominio), o puede devolver varios resultados.

BIND es una implementación open source del protocolo DNS. El archivo named.conf especifica la configuración del servidor.

#### 1.4. IPv6

El protocolo de direcciones IPv4 tiene varias limitaciones: escasez de direcciones, que trataron de solucionar incluyendo CIDR, direcciones privadas NAT

y el protocolo DHCP para configuraciones dinámicas. Sin embargo, es posible que en un futuro no muy lejano sea necesario un espacio de direcciones mayor. Además, el formato de la cabecera es complejo debido a su longitud variable. Tanto la seguridad como los soportes para la prioridad de tráfico son limitados.

Es por ello que surge otro protocolo IPv6, compatible con IPv4, con el que se trata de solucionar estos problemas: direcciones de 128 bits, sin clases, longitud de cabecera fija y las opciones se codifican en el cuerpo del paquete a modo de cabeceras extensibles, permite la autoconfiguración y la seguridad y el tráfico mejoran.

#### 1.4.1. Direcciones

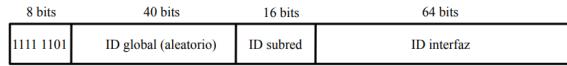
Direcciones de 128 bits, sin clases. Se agrupan en 16 bits, 8 grupos separados por ::. Cada grupo se expresa en notación hexadecimal. Las cadenas de 0's se reducen a ::, aunque solo se puede utilizar una vez dentro de la misma dirección, para evitar ambigüedades. Los 0 a la izquierda en cada grupo se pueden omitir. Se utiliza notación CIDR para indicar la longitud del prefijo.

El **ámbito** de las direcciones es la región en la que es visible. Puede ser de **enlace local**, de **sitio local** visible para las subredes del sitio pero no para Internet y **global**. La **zona** es la parte de red conexa. Los paquetes no se redirigen a zonas distintas. Tan solo se asegura la unicidad dentro de la zona, no del ámbito. Es por eso que para evitar la ambigüedad, a veces a la dirección se le añade % id\_zona.

| Tipo de dirección                      | FP (binario) | FP (hexadecimal) |
|--|--------------|------------------|
| Reserved Address                       | 0000 0000    | ::/8             |
| Global Unicast Address                 | 001          | 2000::/3         |
| Link-Local Unicast Address             | 1111 1110 10 | FE80::/10        |
| Site-Local Unicast Address (en desuso) | 1111 1110 11 | FEC0::/10        |
| Unique Local Address (ULA)             | 1111 110     | FC00::/7         |
| Multicast Address                      | 1111 1111    | FF00::/8         |

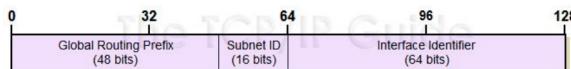
Existen 3 tipos de direcciones en función de lo que representan:

- **Unicast**: asignada a un único nodo o interfaz. Los últimos 68 bits son el ID de la interfaz, que se obtienen a partir de la dirección física (48 bits). Según el ámbito en el que se aplique, pueden ser de diferentes formas:
  - **Link-Local Unicast**: direcciones para redes locales. FE80::/64
  - **Site-Local Unicast**: direcciones para sitio local. En desuso. FEC0::/10 es como comienza.
  - **ULA**: sustituye al site-local. Los primeros 7 bits son 1111 110. El bit 8 indica si se asigna el prefijo de forma local o está indefinido. Los siguientes 40 bits son el ID global, otorgados de forma pseudoaleatoria

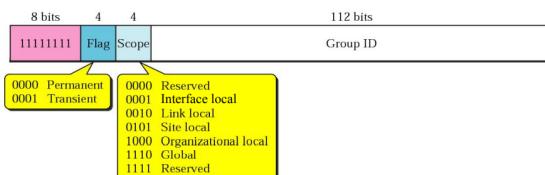


y los últimos 16 bits del prefijo indican la subred. Esto significa que cada sitio puede tener  $2^{16}$  subredes.

- **Global Unicast:** direcciones unicast globales. Comienza por 001. Los siguientes 45 bits representan el sitio y los últimos 16 la subred. Esto significa que existen  $2^4 \cdot 2^{16} = 2^{20}$  sitios diferentes, gestionados por la IANA.



- **Multicast:** asignada a un grupo de nodos. Un paquete dirigido a esta dirección se entrega a todos los nodos del grupo. No existe dirección de broadcast. Comienzan por 1111 1111. Los 4 bits siguientes indican si el grupo es permanente (0000) o temporal (0001). Los 4 siguientes indican el ámbito del grupo y los 112 siguientes representan el ID del grupo.



Las direcciones FFXY::1 son destinadas a nodos, mientras que las FFXY::2 a encaminadores. Recordemos que Y marca el ámbito, es decir, si Y=2 estamos en ámbito de enlace local y el paquete se enviaría a todos los nodos o encaminadores de la red local. Otro ejemplo es FF02::9 que lo envía a los encaminadores RIP de red local.

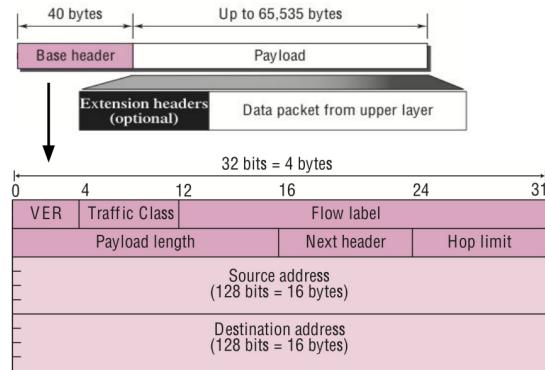
Por último, destaca la dirección de **nodo solicitado**, usado para obtener dirección física de una IP en el protocolo de descubrimiento de vecinos (red local). La dirección es FF02::1:FFXY:UVWZ, siendo los 24 bits del final los de la dirección unicast.

- **Anycast:** asignada a un grupo de nodos, pero el paquete se envía al nodo más cercano según el encaminamiento.

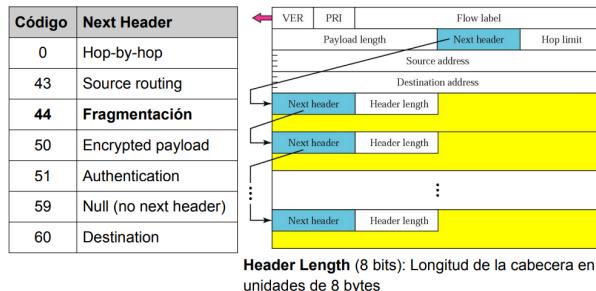
Otras direcciones especiales son: :: para indicar que la interfaz no tiene dirección asignada, ::1 de loopback y ::FFFF:IPv4 para las traducciones de IPv4 a IPv6.

### 1.4.2. Datagrama

La cabecera del datagrama tiene un tamaño fijo de 40 bytes. 32 bytes para las direcciones origen y destino y 8 para los campos. El campo Traffic Class es el equivalente a DS en IPv4, distingue requisitos y con ECN detecta situaciones de congestión de la red. El campo Hop Limit es el equivalente a TTL. El campo Flow Label etiqueta el paquete para mejorar el procesamiento de los encaminadores.



A diferencia de IPv4, ni la información de fragmentación, ni el Checksum, ni el tamaño de la cabecera aparecen en la propia cabecera. En cambio aparece un nuevo campo **Next Header** de 8 bits. Hace referencia a la próxima cabecera extensible. En función del código que contenga puede ser una extensión o la cabecera del protocolo superior (TCP = 6, UDP = 17, ICMPv6 = 58).

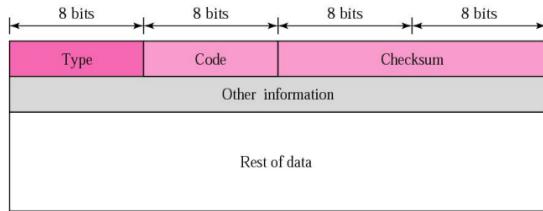


Si Next Header = 44, tenemos cabecera de fragmentación. Esta se produce en el nodo origen, no en los encaminadores. Por ello, los nodos deben conocer el MTU del camino. Sin los datos de la capa superior superan el MTU, ha de fragmentarse (en cada paquete siempre debe ir la cabecera del protocolo superior). El flag M indica si hay más fragmentos.

|                         |               |                      |       |
|-------------------------|---------------|----------------------|-------|
| Next header             | Header length | Fragmentation offset | 0   M |
| Fragment identification |               |                      |       |

#### 1.4.3. ICMPv6

Este protocolo de IPv6 asume el papel de ICMP y ARP en IPv4. Al igual que en ICMP, se envían mensajes de error (0=*tipo*=127) o de información (128=*tipo*=255), que incluye el protocolo de descubrimiento de vecinos o el de gestión de grupos multicast. También se incluye el Checksum para comprobación de errores. Un ejemplo de error sería destino inalcanzable con tipo = 1 y código entre 0 y 4: 0-*i*, sin ruta, 1-*i*, comunicación no permitida, 2-*i*, fuera de ámbito, 3-*i*, dirección inalcanzable, 4-*i*, puerto inalcanzable. Otros ejemplos de errores son: datagrama demasiado grande (2), tiempo excedido (3) y problemas de parámetros (4).



Los mensajes de echo request (128) y reply (129) son análogos a los de IPv4. Los mensajes de información se utilizan en el protocolo de **descubrimiento de vecinos**. Este protocolo opera sobre nodos y encaminadores de la misma red local.

- **Descubrimiento de vecinos:** se utilizan dos mensajes, el de solicitud (135) y el de anuncio. El de **solicitud** se puede emplear para averiguar la dirección física de una IP con la dirección multicast de nodo solicitado como destino; para ver si un vecino sigue siendo alcanzable con la dirección unicast como destino; o detectar si la IP del proceso de autoconfiguración está duplicada, usando tb la IP de nodo solicitado. El de **anuncio** para responder a solicitud de vecino con dirección unicast como destino; o anunciar un cambio en la dirección física con la dirección multicast FF02::1 como destino. En este mensaje, tenemos 3 flags: R si el emisor es encaminador, S si se responde solicitud y 0 si es de reemplazo de dirección.
- **Descubrimiento de encaminadores:** se utilizan dos mensajes, el de solicitud (133) y el de anuncio (134). El de solicitud se utiliza para detectar encaminadores y realizar autoconfiguración de la interfaz, con FF02::2 como destino. El de anuncio los envían los encaminadores para anunciar su presencia, con FF02::1 si el mensaje se dirige a todos los nodos o la unicast del solicitante como destino. Tiene flag M si se usa DHCPv6 para

configurar y O si DHCPv6 proporciona otra información a parte de la dirección (ej. servidores DNS...).

- **Redirección:** mensaje de tipo 137, que notifica a un nodo una ruta más adecuada para alcanzar un determinado destino.

Otra función del protocolo ICMPv6 es la autoconfiguración, donde el prefijo es anunciado por el encaminador y el identificador de interfaz generado igual que para Unicast.

## 1.5. Encaminamiento en Internet

En el encaminamiento es esencial es coger cuál es el mejor camino posible. Existen varias métricas para medir la distancia del encaminamiento: nº de saltos, distancia física, retardos promedio, ancho de banda (velocidad de las líneas) o nivel de tráfico. El **plano de control** decide el mejor camino y el **plano de datos** reenvía los paquetes. Los encaminamientos siguen el principio de optimabilidad de Bellman, que dice que si el camino más corto entre A y B pasa por C, entonces el camino entre C y B es el mínimo tb. Además, para que la tabla de rutas de los encaminadores no sea excesivamente grande, se utiliza CIDR para agrupar direcciones. Existen dos tipos de encaminamientos: estáticos, las tablas de rutas se fijan manualmente; y dinámicos, las tablas de encaminamiento se actualizan de forma automática con el intercambio periódico de información entre los encaminadores.

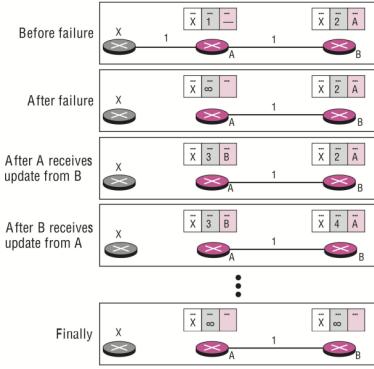
Internet está organizado en **sistemas autónomos AS**, que son conjuntos de redes y encaminadores gestionados por la misma autoridad. Por lo tanto, existen 2 tipos de encaminadores, los **internos** que solo guardan información de su propio AS y los **externos o border routers**, que conocen rutas entre AS. El protocolo para internos es IGP y el de externos es EGP. RIP y OSPF son IGP, mientras que BGP es EGP.

### 1.5.1. RIP

Cada encaminador guarda un **vector de distancias**, en el que las entradas consisten en: posible destino; distancia al destino, normalmente medida en nº de saltos; y la dirección del siguiente salto. Para calcular este vector, todos los encaminadores comparten su propio vector con el resto y las entradas se actualizan si se encuentra un camino mejor. Iterando este proceso se converge a los caminos óptimos (idealmente). Este algoritmo se conoce como **Bellman-Ford**. Sin embargo, las actualizaciones para comunicar un enlace caído pueden no converger. Este es el problema de la **Cuenta al infinito**.

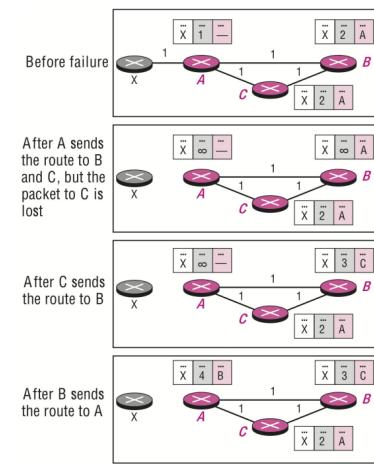
Algunas soluciones existentes:

- **Infinito pequeño:** si el infinito se establece en X saltos, a partir de X se considera inalcanzable el destino. RIP lo fija en 16.



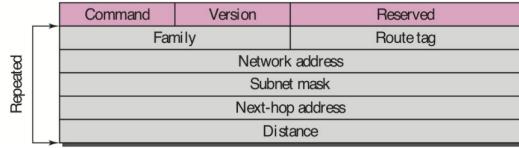
- **Horizonte dividido:** los destinos aprendidos a través de un determinado enlace nunca se difunden a través de dicho enlace
- **Horizonte dividido con ruta inversa envenenada:** los destinos aprendidos a través de un determinado enlace sí se difunden a través de dicho enlace, pero con distancia infinita
- **Actualizaciones forzadas:** Cuando un encaminador detecta una modificación en su tabla de rutas inmediatamente difunde esta información a sus vecinos. De esta forma, los cambios en la topología se propagan de forma rápida.

Aun con estas soluciones, si tenemos bucles en la red, todavía no estaría solucionado el problema de la cuenta al infinito. Las actualizaciones forzadas aceleran la convergencia y en este caso el horizonte dividido no funciona.



En RIP, la distancia se mide como nº de saltos y la listas de destinos son redes. Los mensajes se encapsulan como datagramas UDP al puerto 520. Intro-

duce todos los mecanismos explicados para solucionar la cuenta al infinito y la versión 2 tiene soporte para CIDR, direcciones multicast y autenticación.



El campo Route Tag almacena información extra sobre la ruta o algoritmo de autenticación. El campo Family puede ser 2 de TCP/IP o 0xFFFF de autenticación. Next-hop address suele ser 0.0.0.0 para usar dirección del remitente. La parte gris se repite para cada entrada del vector distancias. Se envían 2 tipos de mensajes:

- **REQUEST:** se envía cuando se conecta a la red con Network address a 0.0.0.0 en todas las entradas, o cuando una entrada de la tabla expira.
- **RESPONSE:** se envía para responder una solicitud, periódicamente con todo el vector de distancias (broadcast) o cuando la distancia a una red cambia (actualización forzada).

RIP maneja 3 temporizadores para controlar los vectores distancias: **periódico** (25-35s), para lanzar mensajes RESPONSE broadcast; **expiración** (180s), marca periodo de validez de una entrada de la tabla sin actualizar; **recolección de basura** (120s), tras expirar una entrada, esta se marca con métrica 16 durante los 120 segundos.

RIP presenta varios problemas: mucho broadcast, no acepta métricas alternativas al nº de saltos, no admite caminos alternativos una vez halladas las tablas y si la red es grande los cambios pueden tardar en propagarse y el infinito quedarse corto.

Existe una versión de RIP compatible con IPv6, **RIPng**. Los mensajes van al puerto 521 y se difunden a la dirección FF02::9. Los mensajes anuncian prefijos de red en vez de direcciones IPv4 y no se incluye el campo Next-Hop.

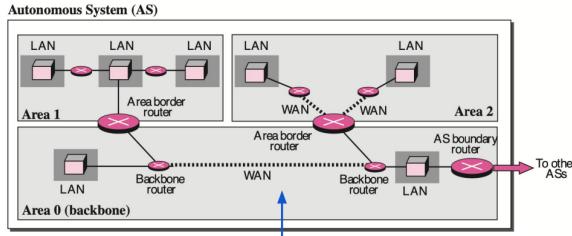
### 1.5.2. OSPF

Este protocolo trata de solucionar los problemas de RIP. En vez de guardar un vector de distancias, cada nodo lleva la topología exacta de la red a modo de grafo. Para construirlo, identifican las distancias a sus vecinos (**estados de enlaces**) y comparten la información por toda la red. Utilizan Dijkstra para hallar los mejores caminos. Los cambios se propagan inmediatamente.

El AS se divide en **áreas**, agrupaciones de encaminadores y redes con un único identificador (área ID) de 32 bits. Destaca el área 0 o **backbone**, que

conecta todas las áreas. Siempre existe en toda red OSPF. Según dónde se encuentren los encaminadores, pueden ser de 3 tipos:

- **Internal Router (IR)**: todos los interfaces del router en un área. Solo guarda información de ella.
- **Area Border Router (ABR)**: conecta 2 o más áreas. Mantiene una BD para cada área.
- **AS Boundary Router (ASBR)**: en la frontera del AS. Transmite rutas externas, que pueden ser aprendidas por otro protocolo como RIP.



Dos encaminadores con un enlace común, de la misma área y con el mismo mecanismo de autenticación son **vecinos**. Si dos encaminadores vecinos son **adyacentes**, sincronizan sus bases de datos y posteriormente comparten con el resto la información. Esto permite limitar la información intercambiada, pues no se comunican todos los vecinos. Si la red es multi-acceso, el encaminador designado (DR) y el de respaldo (BDR) son adyacentes al resto de encaminadores de la red. En este caso los encaminadores envían sus estados de enlaces al DR y BDR. El DR, o en caso de fallo el BDR, envía de vuelta a los encaminadores el grafo completo. Estos mensajes han de confirmarse para asegurar la fiabilidad. El protocolo OSPF Hello permite descubrir vecinos y seleccionar DR y BDR.

Utiliza un protocolo propio de encapsulado. Los mensajes se envían mediante direcciones multicast: 224.0.0.5-224.0.0.6 o FF02::5-FF02::6.

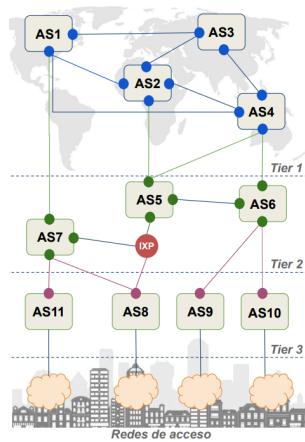
### 1.5.3. BGP

Este protocolo es EGP, es decir, externo a los AS. Las técnicas de los protocolos anteriores no sirven. Basándose en el encaminamiento por vector de distancias, surge el encaminamiento por **vector de rutas**. Cada entrada del vector es un destino alcanzable (red) junto con la ruta completa al destino (lista de AS que han de atravesarse). Permite detectar bucles de forma sencillas y eliminarlos si se desea. Cada AS luego puede aplicar políticas para aceptar o anunciar rutas.

Los encaminadores intercambian sus tablas de rutas cuando establecen la conexión inicial y envían actualizaciones incrementales si las tablas cambian. Los mensajes BGP se envían mediante TCP por el puerto 179:

- **OPEN**: iniciar sesión BGP, fijando el identificador AS y de encaminador.
- **UPDATE**: actualización de la información de encaminamiento. El mensaje incluye redes alcanzables con sus atributos y otra lista de redes retiradas. Los atributos permiten evaluar caminos alternativos. Los tipos de atributos son:
  - **Bien Conocidos**: admitidos por todas las implementaciones BGP. Pueden ser **obligatorios** si se incluyen en cada actualización, o **discretos**. Algunos ejemplos de obligatorios son: **ORIGIN**, origen de la información de la ruta (IGP, EGP O INCOMPLETE); **AS\_PATH**, secuencia de AS que representa la ruta; y **NEXT-HOP**, dirección IP del siguiente salto.
  - **Opcionales**: específicos de cada implementación. Los atributos **transitivos** se deben incluir en todas las actualizaciones, aunque no sean implementados por el encaminador.
- **NOTIFICATION**: se envía a los vecinos cuando se notifica un error. Se cierra la sesión y se invalidan las rutas asociadas.
- **KEEPALIVE**: asegurar que la sesión permanezca activa. En respuesta a OPEN y periódicamente para confirmar presencia de encaminador. Si pasado tiempo de hold no se recibe información, se cierra la sesión.

#### 1.5.4. Arquitectura de Internet



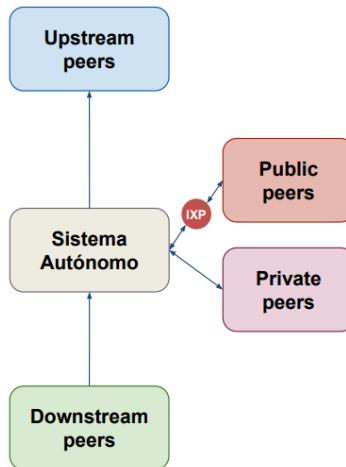
Internet está dividido en 3 tiers:

- 1: Backbone de Internet. Acceso a cualquier red sin pagar.
- 2: Proveedores (carriers) que pagan para llevar su tráfico. Normalmente a salto de backbone y con presencia en un solo continente.

- 3: Proveedores de acceso a Internet (**ISP**) que pagan por el tráfico. Nivel regional o nacional.

Existen diferentes tipos de AS: **Stub**, conectados a otro AS como destino u origen (AS9); **Transit**, conectado a varios AS y permite tráfico de tránsito (tiers 1 y 2); y **Multihomed**, conectado a varios AS sin permitir tráfico de tránsito (AS8). Los AS establecen relación de **peering** cuando se intercambian información. Esta relación puede ser de 3 formas:

- **Upstream** o proveedores: AS consume servicios de tránsito mientras que el proveedor envía información de rutas y redes a las que tiene acceso.
- **Public/Private**: entre AS de igual nivel. Suelen ser relaciones gratuitas, y enlaces directos en caso de ser privado. Intercambian prefijos de redes y de clientes.
- **Downstream** o clientes: distribuye prefijos entre sus clientes. Además proporciona servicios de tránsito a Internet a sus clientes.

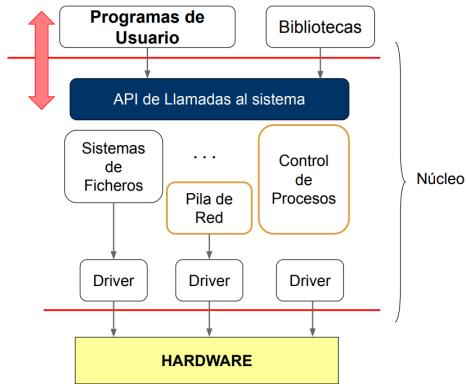


## 2. SISTEMA OPERATIVO

### 2.1. Introducción a la Programación de Sistemas

Este resumen es una continuación de la asignatura de Sistemas Operativos. Por lo tanto, todo lo visto anteriormente se dará por conocido.

Las **llamadas al sistema** son funciones de la biblioteca estándar de C que solicitan un servicio al sistema (trap), que se resuelve en el kernel del SO. No



reserva espacio de memoria para los parámetros. Devuelven -1 en caso de error y modifican la variable `errno` (más adelante veremos qué es). Las **funciones de biblioteca** no interaccionan directamente con el sistema. En su código sí que pueden haber llamadas al sistema. Estas si reservan espacio para los parámetros y pueden devolver NULL en caso de error.

Para rastrear las llamadas al sistema que realiza un programa, existe el comando `strace`. En cada línea muestra la llamada, los argumentos y el valor de retorno. Se pueden añadir algunas opciones: `-c` mucha info, `-T` tiempos, `-f` traza los procesos hijos, `-e trace = call` filtrar el tipo de llamada y `-e write = fd` vuelca los datos escritos en el descriptor `fd`.

A la hora de gestionar los errores, destaca la función `perror(char* s)`. Muestra en la salida de error (`stderr`) el mensaje asociado al último error de una llamada al sistema o función de biblioteca. El código de error se obtiene de la variable `errno`, que no se actualiza cuando la llamada tiene éxito.

En cuanto a obtener información, puede ser:

- Sistema operativo: mediante `uname(struct utsname * b)`. Todo lo relativo al SO (nombre, versión, Hardware,...) En caso de error devuelve `EFAULT`. El comando `uname` proporciona esta funcionalidad. También se puede acceder a esta información vía `sysctl`.
- Configuración del sistema: `sysconf(int name)`. En función de `name` devuelve el valor de configuración del sistema. Devuelve -1 en caso de error, pero no modifica `errno`.

- `_SC_ARG_MAX`: Longitud máxima de argumentos en funciones `exec()`
- `_SC_CLK_TCK`: Número de ticks de reloj por segundo (Hz)
- `_SC_OPEN_MAX`: Número máximo de ficheros abiertos por proceso
- `_SC_PAGESIZE`: Tamaño de página en bytes
- `_SC_CHILD_MAX`: Número máximo de procesos simultáneos por usuario

- Configuración de un fichero: pathconf(char\* path, int name) o fpathconf(int fd, int name). Análogo a sysconf, solo que en este caso sí modifica errno cuando ha encontrado el fichero y hay un error.

```

■ _PC_LINK_MAX: Número máximo de enlaces
■ _PC_NAME_MAX: Longitud máxima del nombre de fichero
■ _PC_PATH_MAX: Longitud máxima de la ruta relativa
■ _PC_CHOWN_RESTRICTED: Devuelve un valor no nulo si el cambio del
    propietario del fichero está restringido
■ _PC_PIPE_BUF: Número máximo de bytes que pueden escribirse
    atómicamente en la tubería

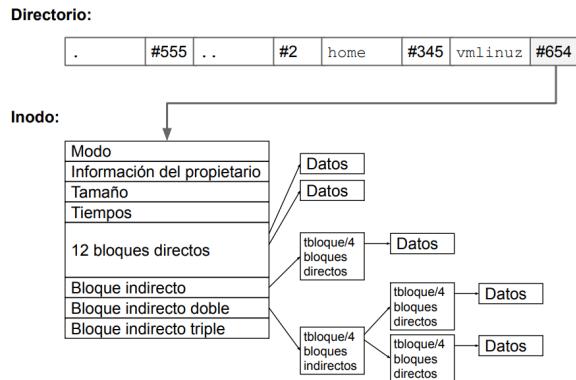
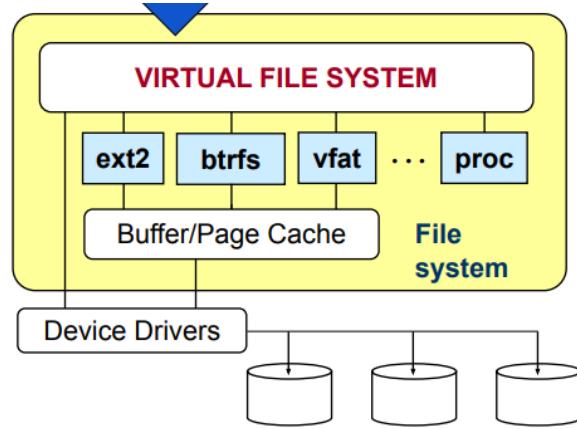
```

- Identificadores de un proceso: Los identificadores (usuario y grupo) pueden ser de 2 tipos: **reales (UID, GID)**, los creadores del proceso o en su defecto heredados del proceso padre; y **efectivos (EUID, EGID)**, los que se comprueban para permisos. Suelen ser los mismos, a no ser que el fichero de programa tenga activo el bit **setuid** o **setgid**. En ese caso EUID y EGID son los del fichero, y no los creadores del proceso. Para obtenerlos, utilizar getuid(), etc.
- Usuario de la base de datos de contraseñas: mediante struct passwd \*getpwnam(char\* name). El struct guarda información del usuario (nombre, contraseña, identificadores...). Devuelve NULL en caso de error o ENOMEM si no hay espacio en memoria para el struct. Con shadow passwords es necesario getspnam.
- Hora: time(time\_t \* t) devuelve los segundos desde el EPOCH. Si t no es NULL, también guarda el resultado. Algunas funciones interesantes son:
  - gettimeofday(struct ..., NULL)/set: obtener o establecer (superusuario) fecha del sistema.
  - struct tm\* gmtime(time\_t\* t): obtener el tiempo desglosado UTC o en zona horaria local (localtime en vez de gmtime).
  - strftime(..., char\* format, struct tm\* tm): formatear fecha y hora.

## 2.2. Sistemas de Ficheros

Desde el punto de vista de los usuarios es una colección de ficheros y directorios, pero desde el S0 son tablas y estructuras. La capa **VFS** permite la conexión entre el kernel y los diferentes sistemas de ficheros. Es una API. Contiene varias caches que mejoran la eficiencia de entrada/salida: de inodos, de entradas de directorios y de páginas.

Los sistemas de ficheros basados en disco evolucionaron a partir de Minix, ext, ext2... En ext2, los bloques de datos están cerca de los inodos, y estos cerca de su directorio. Se separan en n bloques, cada uno con la misma estructura: superbloque, descriptores de grupo, mapa de bloques y de inodos, tabla de inodos y bloques de datos. La estructura del inodo se vio en la asignatura de Sistemas Operativos.



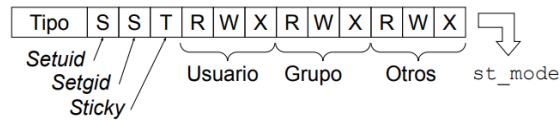
Cuando un sistema de ficheros se cierra de forma abrupta e incorrecta, es necesario revisar la consistencia al abrirlo de nuevo. Esto implica recorrer toda la estructura buscando inodos huérfanos o inconsistentes. Como tiene mucho coste, en ext3 se incorporó un fichero, región o dispositivo especial denominado **journal**. En él se almacenan los metadatos (mapas,inodos...) y en caso de fallo, de forma periódica o si supera un tamaño, se produce la consolidación del sistema de ficheros (recuperar los datos a partir del journal).

A continuación mostramos los conceptos más importantes de los ficheros. Todas las funciones son en realidad llamadas al sistema, por lo que en caso de error devuelven -1 y modifican errno.

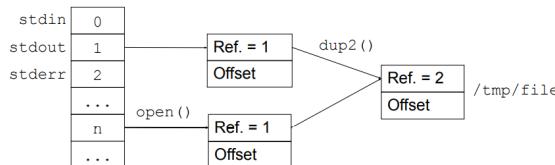
- **Atributos:** la estructura stat almacena múltiples atributos del fichero: dispositivo, inodo, permisos, uid, gid, nº de enlaces rígidos, último acceso (st\_atime), última modificación (st\_mtime) y último cambio en el inodo (st\_ctime). Los atributos se obtienen mediante stat o lstat (no sigue enlaces simbólicos). No se necesitan permisos sobre el fichero, pero sí para

buscar la ruta. fstat requiere de descriptor de fichero abierto.

Para comprobar el tipo de fichero y los permisos se pueden utilizar unas macros. El tipo del fichero no se puede modificar. Los permisos se almacenan en la estructura mode\_t: bits de setuid, setgid, sticky y permisos de usuario, grupos y otros. Para modificar los permisos se utiliza chmod y el EUID del proceso debe coincidir con el del propietario del fichero o el proceso debe ser privilegiado. Para comprobar si un usuario tiene permisos en un fichero se utiliza access. En esta comprobación se tiene en cuenta la ruta completa y se checkearán los identificadores reales, no efectivos.



- **Abrir/Crear:** un fichero abierto se representa mediante un entero, el descriptor de fichero. Para abrir se utiliza open(path,int flags). Los flags indican el tipo de apertura: escritura, lectura o ambas. Si se incluye O\_CREAT y el fichero no existe, se crea uno nuevo. open requiere los permisos como argumento de entrada. Estos se verán afectados por la máscara del proceso **umask**, desactivando las opciones seleccionadas. Para fijar la máscara se utiliza la llamada umask, que siempre se ejecuta correctamente y devuelve la antigua máscara. Al crear un nuevo fichero, UID y GID serán los efectivos del proceso, excepto si el bit de setgid está activo, que en ese caso será el GID del directorio. Además, si no se incluyen permisos, se darán valores arbitrarios de la pila. Open devuelve el menor descriptor de fichero disponible y el puntero de acceso apuntando a la primera posición del fichero.
- **Duplicar:** se duplica un descriptor de fichero abierto y ambos apuntan al mismo (comparten cerrojos, punteros y opciones). Se utiliza dup(old) o dup2(old,new), donde new referirá a old y si estaba abierto, se cerrará.



- **Enlaces rígidos y simbólicos:** los enlaces rígidos son copias de ficheros. Deben hacerse sobre ficheros del mismo sistema e incrementan el nº de enlaces del inodo. Todas las copias apuntan al mismo inodo. El enlace simbólico es otro inodo, cuyo contenido es la ruta a un fichero cualquiera.

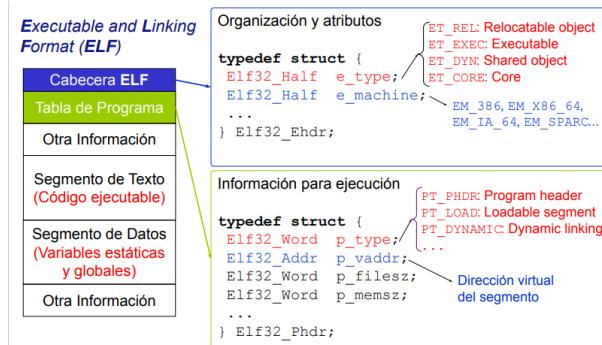
Para crear un enlace rígido se utiliza link y para uno simbólico se utiliza symlink.

- **Borrar un fichero:** Para eliminar un fichero se utiliza unlink. Borra la entrada del directorio y decrementa el nº de enlaces. Si es 0 y ningún proceso tiene abierto el fichero, este se elimina. Si el bit sticky está activado, solo puede ser borrado por el propietario del fichero o del directorio.
- **Cerrojos:** un fichero puede tener asignados cerrojos de POSIX para bloquear o desbloquear regiones de él mismo. Estos son consultivos(read y write no comprueban su existencia), por lo que son útiles entre procesos que cooperan. Para manejar los cerrojos se utiliza lockf.

En cuanto a **directorios**, estos se representan mediante la estructura DIR. Para abrirlo se utiliza opendir y devuelve el puntero de flujo apuntando a la primera entrada en DIR. Para leer entradas se utiliza readdir(DIR). Devuelve una estructura dirent, que apunta a la siguiente entrada en el directorio o NULL si es el final. Para cerrarlo se utiliza closedir(DIR). Para crear un directorio se utiliza mkdir y al igual que open en ficheros, los permisos se ven afectados por umask. UID y GID son los efectivos del proceso a no ser que el bit de setgid del directorio padre esté activado, en cuyo caso GID es el del directorio padre. Para eliminar es rmdir(path) y para renombrar (tb válido para ficheros) es rename(old,new). Ambos han de ser del mismo tipo y estar en el mismo sistema. Si new existe se elimina y si es un enlace simbólico se sobreescribe.

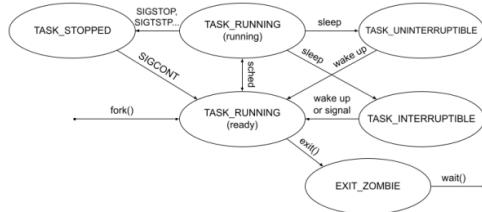
### 2.3. Gestión de Procesos

Un programa es un conjunto de instrucciones y datos almacenados en una imagen ejecutable en disco. Es una entidad pasiva. Para traducir la imagen virtual en páginas de la memoria física, se utilizan las tablas de traducción (MMU).



Un proceso (ejecución de un programa) pasa por diferentes estados a lo largo de su vida: **task\_running**, en ejecución o preparado (R); **task\_uninterruptible**, bloqueado (D); **task\_interruptible**, esperando a evento (S); **task\_stopped**,

parado por una señal (T); y **exit\_zombie**, muerto, aunque deja su entrada en la tabla de procesos para que su padre pueda recoger el estado de salida (Z).



Para organizar la ejecución de las tareas de los procesos, el núcleo cuenta con un planificador expropiativo. Esto significa que una tarea de mayor prioridad le robará el procesador a otra de menor prioridad. Las políticas de planificación determinan el orden entre 2 procesos de igual prioridad. Las 3 más frecuentes son:

- **SCHED\_OTHER:** utiliza el valor de **nice** (-20 i- i 19) a modo de prioridad. Valores menores representan mayor porción de CPU.
- **SCHED\_FIFO:** prioridades entre 1 y 99. Una tarea se ejecutará hasta que se bloquee por E/S, sea expropiada o ceda la CPU.
- **SCHED\_RR:** igual que FIFO pero las tareas con misma prioridad se ejecutan por turnos durante un **cuanto** de tiempo máximo.

Los atributos de planificación se heredan. Para modificar o ver planificador utilizar getscheduler, que aunque requiera del pid, se refiere al hilo actual. Para modificar el valor de nice utilizar setpriority(int which, int who, int prio) donde which puede ser PRIO\_PROCESS, PRIO\_PGRP o PRIO\_USER y who es PID,PGID o UID respectivamente (si es 0 se refiere a proceso actual, grupo de procesos actual o UID real del proceso actual).

Cada proceso tiene su propio identificador único **PID**. También registran el PID del padre (**PPID**). Los procesos se pueden agrupar en **grupos**. Cada grupo tiene un **PGID**, que es el PID del proceso líder. Su principal uso es la distribución de señales y el PGID se puede modificar (si pgid es 0 en la llamada, se toma como pgid el pid de la llamada). Los grupos de procesos se pueden juntar a su vez en **sesiones**, cada una con su **SID**. Un proceso crea una sesión y abre un terminal que comparten todos los procesos de la sesión. Al desconectar, envía la señal de SIGHUP a todos los procesos de la sesión. El proceso que crea la sesión no puede ser un líder de grupo. Al crearla, el SID es el PID del proceso y se crea automáticamente un grupo cuyo PGID es idéntico al SID. Para crear una nueva sesión se utiliza setsid.

El directorio de trabajo es el directorio usado para resolver toda la ruta relativa en el proceso. Para obtener la ruta absoluta se utiliza getcwd y para

cambiar el directorio de trabajo chdir. El entorno de un proyecto es un conjunto de variables y su valor. Se hereda del padre, aunque se puede controlar el entorno que se pasa.

Para crear procesos se utiliza **fork**. Devuelve: 0 si está ejecutando el hijo, >0 ejecutando el padre y -1 error. El proceso hijo ejecuta el mismo código y recibe una copia de los descriptores de ficheros abiertos del padre. Para finalizar un proceso (sin señales) utilizar exit(int status). **Status** es el estado de salida del proceso, menor de 255. 0 es éxito y 1 error. Se cierran los descriptores de fichero abiertos y el proceso padre recibe señal de SIGCHLD. Si el padre finaliza, los hijos quedan huérfanos y son heredados por el proceso init (pid = 1). Para esperar la finalización de los hijos se utiliza wait o waitpid. En el caso de waitpid, si pid es 0 indica que espera a cualquiera del grupo del proceso del padre, si es -1 a cualquiera y si >-1 a cualquiera con PGID = -pid. Ambas devuelven en PID del hijo terminado y si un proceso termina sin ser esperado se convierte en zombi. El status contiene información del estado del proceso esperado: WIFEXITED o WIFSIGNALED. waitpid tiene un argumento options, que puede ser: WNOHANG (no espera a los hijos), WUNTRACED (retorna si el proceso ha sido detenido) o WCONTINUED (retorna si un hijo detenido ha sido reanudado). Para ejecutar un programa se utiliza cualquier función de la familia exec. Sustituye la imagen del proceso actual por una nueva. Si tenemos vector de argumentos, el 1º debe ser el nombre del programa y el último NULL. Para ejecutar un comando de la shell existe la llamada system. Retorna cuando finaliza la ejecución y devuelve el código de finalización.

|                      | Ruta absoluta | Ruta relativa | Nuevo entorno |
|----------------------|---------------|---------------|---------------|
| Lista de argumentos  | exec1()       | exec1p()      | execle()      |
| Vector de argumentos | execv()       | execvp()      | execve()      |

---

Los procesos consumen una serie de recursos de la CPU. Es posible consultar y modificar los recursos y los límites, mediante getrlimit y getrusage. En getrusage hay que especificar de quien nos referimos: RUSAGE\_SELF (todos los hilos del proceso), RUSAGE\_CHILDREN (todos los hijos) y RUSAGE\_THREAD.

## 2.4. Señales

Las señales son interrupciones del software que informan a un proceso de un suceso, de forma **asíncrona**. Las señales se pueden bloquear, ignorar, realizar acción por defecto o capturar la señal con un manejador. Para enviar una señal se utiliza kill y el pid de argumento funciona igual que en el waitpid (con -1 se envía a todos los procesos excepto el 1).

Las señales se agrupan en conjuntos para establecer la **máscara** de señales. Estas son el conjunto de señales bloqueadas. Para modificarla se utiliza sigprocmask(how, set, oldset). Con how permitimos añadir set al conjunto oldset

- **SIGPGRP**: Desconexión de terminal (**F**, terminar proceso)
- **SIGINT**: Interrupción. Se puede generar con **Ctrl+C** (**F**)
- **SIGQUIT**: Finalización. Se puede generar con **Ctrl+\** (**F** y **C**, volcado de mem.)
- **SIGSTOP**: Parar proceso. No se puede capturar, bloquear o ignorar (**P**, parar)
- **SIGSTP**: Parar proceso. Se puede generar con **Ctrl+Z** (**P**)
- **SIGCONT**: Reanudar proceso parado (continuar)
- **SIGILL**: Instrucción ilegal (punteros a funciones mal gestionados) (**F** y **C**)
- **SIGTRAP**: Ejecución paso a paso, enviada después de cada instrucción (**F** y **C**)
- **SIGKILL** (9): Terminación brusca. No se puede capturar, bloquear o ignorar (**F**)
- **SIGBUS**: Error de acceso a memoria (alineación o dirección no válida) (**F** y **C**)
- **SIGSEGV**: Violación de segmento de datos (**F** y **C**)
- **SIGPIPE**: Intento de escritura en un tubería sin lectores (**F**)
- **SIGALRM**: Despertador, contador a 0 (**F**)
- **SIGTERM**: Terminar proceso (**F**)
- **SIGUSR1, SIGUSR2**: Señales de usuario (**F**)
- **SIGCHLD**: Terminación del proceso hijo (**I**, ignorar)
- **SIGURG**: Recepción de datos urgentes en socket (**I**)

**signal (7)**

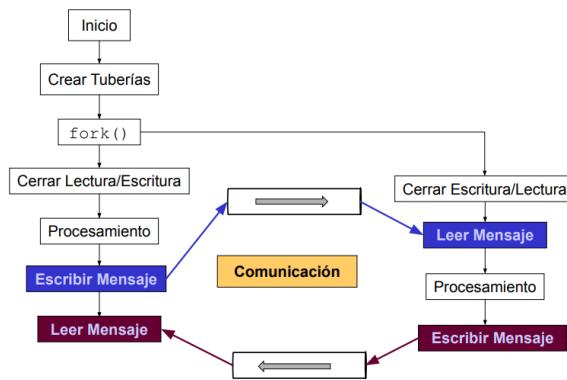
(SIG\_BLOCK), extraer de oldset (SIG\_UNBLOCK) o reemplazar oldset por set (SIG\_SETMASK). Para comprobar las señales pendientes se utiliza sigpending. Por otra parte, para capturarla y tratarla con un manejador propio se utiliza sigaction(signal, struct sigaction act). Todas las señales excepto SIGSTOP y SIGKILL pueden ser tratadas. La estructura contiene 3 campos: el manejador (puntero a función), la máscara de señales que estarán bloqueadas durante el tratamiento (por defecto se bloquea la señal tratada) y un campo de flags. Algunas de las flags más destacadas son: SA\_NODEFER, no bloquea la señal tratada; SA\_RESTART, reinicia llamadas al sistema; SA\_RESETHAND, restaura el manejador por defecto tras tartas la señal; y SA\_SIGINFO. Al finalizar el manejador, se restaura la máscara de señales y se retoma la ejecución en el punto en el que se produjo la señal. En los manejadores no debe usarse funciones no reentrantes, almacenar el valor de errno antes de llamar a función que pueda modificarlo y las variables globales han de ser volatile. Por último, es posible suspender un proceso hasta que llegue una señal mediante sigsuspend(set). La máscara se susituye temporalmente por set y el proceso se suspende hasta que llega una señal que no está en set. Cuando la recibe se ejecuta el manejador y acto seguido se restaura la máscara original. Esta llamada siempre devuelve -1.

También se pueden utilizar las señales para programar alarmas. La función alarm modifica el temporizador ITIMER\_REAL para generar una señal SIGALRM cuando llegue a 0. Cualquier alarma programada se cancela y no debe mezclarse con ninguna función que utilice ese temporizador como sleep o setitimer. No se heredan pero se mantienen tras execve. Existen otros temporizadores que pueden tener asociados alarmas: ITIMER\_VIRTUAL que mide tiempo de CPU en modo usuario y genera SIGVTALRM o ITIMER\_PROF que mide el tiempo de CPU total y genera SIGPROF.

## 2.5. Tuberías

Las tuberías se utilizan para compartir datos entre procesos del mismo sistema. Pueden ser de 2 tipos:

- **Sin Nombre:** el sistema las trata como ficheros (i-nodo, descriptor, heredadas de padres a hijos...). El núcleo se encarga de la sincronización y residen en memoria principal. Solo se pueden utilizar entre procesos que comparten parentesco. Para crear una tubería sin nombre se utiliza `pipe(int fd[2])`. Read se bloquea cuando no hay bytes que leer y write cuando no hay espacio suficiente. `fd[1]` es el extremo de escritura y `fd[0]` el de lectura. Debe cerrarse el extremo que no se vaya a utilizar, puesto que cuando todos los extremos están cerrados, read indica fin de fichero y write envía señal `SIGPIPE`.



- **Con Nombre:** similar a sin nombre, salvo que se accede como parte del sistema de ficheros (con open). De nuevo es el núcleo el encargado de la sincronización. En este caso varios procesos pueden abrir la tubería para lectura o escritura y no han de ser del mismo parentesco. Deben abrirse ambos extremos antes de poder intercambiar datos. Si no lo especificamos en los flags, la apertura de un extremo se bloquea hasta que se abre el otro. Para crear una tubería se utiliza mknod (como fichero normal) con el tipo S\_IFIFO (comando mknod con tipo p) o mkfifo.

Para sincronizar los canales de E/S de un mismo proceso, se puede emplear la opción no bloqueante, aunque así consume tiempo de CPU innecesario, ya que nunca se bloquea. Otra opción es la sincronización basada en señales (asíncrono). Para gestionar los canales de forma síncrona se recurre a la **multiplexación**. El proceso monitoriza varios descriptores esperando a que estén listos. La función select devuelve el nº de descriptores listos y los conjuntos (de lectura, escritura o excepciones) se modifican con los que está preparados. En caso de error no se modifican. El argumento timeout de la llamada es el tiempo máximo en el que retornará la función. Si es NULL, espera hasta que haya alguno listo.

## 2.6. Programación con Sockets

Cada extremo de la comunicación entre un cliente y un servidor se denomina **socket**. Permite el intercambio de datos bidireccional entre cliente y servidor.

Cada aplicación servidor o cliente se identifica con un número de puerto. Existen 3 tipos de sockets:

- **SOCK\_STREAM**: orientado a conexión, fiable y bidireccional. Se debe establecer la conexión y envía señal SIGPIPE en caso de flujo interrumpido. Los mensajes deben llevar cabecera y final para los límites.
- **SOCK\_DGRAM**: datagramas, sin conexión y no fiable.
- **SOCK\_RAW**: orientado a datagramas, permite acceso directo a protocolos de red y transporte. Permite implementar nuevos protocolos.

Cada tipo de socket se basa en un **dominio** de comunicación. Un dominio es una familia de protocolos que usan el mismo espacio de direccionamiento. Un ejemplo sería **AF\_INET** o **AF\_INET6** para IPv4 y IPv6. Se usa un protocolo particular de la familia para implementar el socket.



#### 2.6.1. Direcciones de Sockets

Cada extremo de la comunicación tiene una dirección asignada. Estas se almacenan en structs. IPv4 y IPv6 tienen structs diferentes. En ambos se almacena la familia, el puerto y la dirección IP. Además en IPv6 se almacena el id del flujo y el índice de zona. Si el puerto es menor que 1024, es un proceso privilegiado. Las direcciones se pueden inicializar a **INADDR\_ANY** (0.0.0.0 o ::) y **INADDR\_LOOPBACK** (127.0.0.1 o ::1).

Dado un host, bien con nombre (obtenido a través de gethostbyname), bien con dirección IPv4 o IPv6, se obtienen las direcciones de sockets disponibles con getaddrinfo.

```
int getaddrinfo(const char *node, const char *service,  
                const struct addrinfo *hints, struct addrinfo **res);
```

Si el host es NULL, se refiere al local. Service hace referencia al puerto, que puede ser un número, un nombre de servicio como http o NULL para no especificar ninguno. Hints establece criterios de búsqueda y result es una lista enlazada de las posibles direcciones de sockets. La estructura **addrinfo** contiene los siguientes campos:

Con hints especificamos la familia (**AF\_UNSPEC** para IPv4 y IPv6), el tipo de socket, el protocolo y los flags puede ser AI\_PASSIVE para devolver 0.0.0.0 o :: cuando node sea NULL (en caso contrario devuelve loopback). La estructura sockaddr contiene la dirección devuelta.

```

struct addrinfo {
    int           ai_flags;   // Opciones para filtrado (hints)
    int           ai_family;
    int           ai_socktype;
    int           ai_protocol;
    socklen_t     ai_addrlen; // Resultado (res)
    struct sockaddr *ai_addr;
    char          *ai_canonname;
    struct addrinfo *ai_next;
};

}

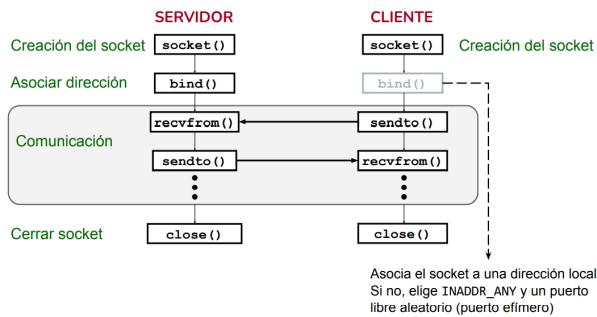
```

Una vez tenemos las direcciones, el siguiente paso es crear el socket. Para ello usar la función socket(int domain, int type, int protocol). El protocolo puede ser IPPROTO\_TCP para SOCK\_STREAM o IPPROTO\_UDP para SOCK\_DGRAM. En ambos casos es 0. La llamada devuelve un descriptor de fichero.

Para obtener el nombre de un host y el servicio a partir de una dirección se utiliza getnameinfo. Además, a veces conviene convertir direcciones de formato binario a texto (inet\_ntop) o viceversa (inet\_pton). Otras ocasiones es necesario convertir valores entre orden de byte de red (big-endian) y de host. Los datos, direcciones y puertos van en orden de byte de red.

### 2.6.2. Socket UDP

Socket de tipo SOCK\_DGRAM para AF\_INET, AF\_INET6 o AF\_UNSPEC. No es necesario establecer conexión entre ambos extremos, aunque el cliente debe conocer a quien le manda el mensaje.



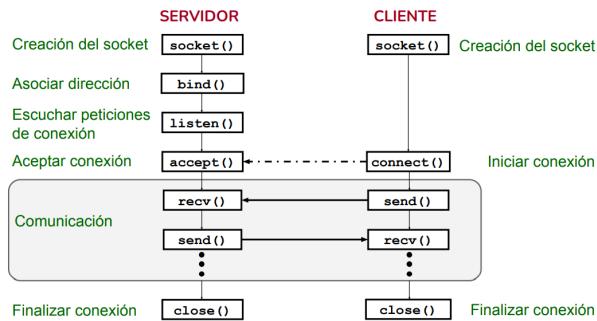
El servidor debe llamar a bind obligatoriamente para asignar una dirección local al socket. La dirección y longitud es la extraída de result. Para enviar datos se utiliza sendto y para recibir recvfrom. recvfrom es bloqueante si no hay mensajes disponibles y actualiza los argumentos de la dirección y longitud. Como el servidor no conoce la dirección a la que enviar datos, esta la obtiene del recvfrom. En SOCK\_STREAM se ignoran las direcciones, pues ya está establecida la conexión.

Para que la aplicación sea compatible con IPv4 y IPv6, a parte de marcar la familia como AF\_UNSPEC, es necesario tener una estructura que elimi-

ne las dependencias en el formato de las direcciones. Dicha estructura es `sockaddr_storage`, que almacena `sockaddr_in` y `sockaddr_in6`. La estructura `sockaddr` solo evita advertencias del compilador.

### 2.6.3. Socket TCP

Orientados a conexión, SOCK\_STREAM. A diferencia del anterior, se establece conexión entre ambos extremos . El cliente envía solicitud de conexión y el servidor la acepta.



El servidor utiliza `listen` para escuchar peticiones. El argumento `backlog` es el tamaño máximo de la cola de conexiones establecidas esperando ser aceptadas. Para aceptar peticiones utiliza `accept(int sd, sockaddr_in* addr, socklen_t* addrlen)`. Es bloqueante y crea un nuevo socket conectado cuando hay conexión pendiente. Devuelve el descriptor de este nuevo socket y en los argumentos se actualiza la dirección del cliente. Para enviar y recibir datos se utiliza `send` y `recv` (no tienen direcciones como argumentos). Ambos son bloqueantes: `send` cuando el mensaje no entra en el buffer de envío y `recv` si no hay mensajes disponibles en el socket. Se pueden utilizar `tb` con `SOCK_DGRAM`, aunque `recv` debe leer el mensaje en una sola operación, y antes ha de utilizarse `connect`.

#### 2.6.4. Otras opciones

Se pueden fijar o consultar otras opciones para los sockets con `setsockopt`. `Level` indica la capa de protocolos donde se aplica la opción, puede ser la API del socket `SOL_SOCKET`.

Una alternativa para aplicaciones que soporten IPv4 y IPv6 es crear un único socket IPv6 para ambas versiones, desactivando la opción IPV6ONLY y asociando con bind a ::. Este método no está soportado en todos los sistemas. También se pueden crear dos sockets, uno para cada versión.

Por último, en la práctica es necesario que el servidor pueda anejar varios clientes a la vez, es decir, tener servidores concurrentes. Como algunas llamadas son bloqueantes (`accept`, `send`, `recv` y `recvfrom`) y los hilos comparten los descriptores (`sockets`), conviene estructurar procesos para ello.

- Nivel de API de sockets (SOCKET):
    - SO\_KEEPALIVE: Activa el mecanismo de *keepalive* en sockets SOCK\_STREAM
    - SO\_BROADCAST: Permite a sockets SOCK\_DGRAM usar direcciones de *broadcast*
    - SO\_REUSEADDR: Activa la reutilización de direcciones locales en TIME\_WAIT
    - SO\_SNDBUF y SO\_RCVBUF: Obtiene o establecen el tamaño de los buffers de envío y recepción (actualmente se autoajusta en función de la latencia y el ancho de banda)
  - Nivel de protocolo TCP (IPPROTO\_TCP):
    - TCP\_NODELAY: Desactiva el algoritmo de Nagle
    - TCP\_QUICKACK: Desactiva los ACKs retrasados
  - Nivel de protocolo IPv4 (IPPROTO\_IP):
    - IP\_ADD\_MEMBERSHIP e IP\_DROP\_MEMBERSHIP: Gestión de grupos multicast
    - IP\_MTU: Obtiene el MTU de la ruta
    - IP\_MTU\_DISCOVER: Activa el algoritmo Path MTU Discovery
    - IP\_OPTIONS, IP\_TTL e IP\_TOS: Obtiene o establecen campos del datagrama
- SOCK\_DGRAM: la concurrencia es a nivel de mensaje, pues múltiples procesos se encargan de procesar los mensajes recibidos con recvfrom.
- SOCK\_STREAM: forks se pueden hacer antes de aceptar la conexión y en ese caso los procesos se encargan de las conexiones. Si primero se acepta, los hilos/procesos solo gestionan el intercambio de mensajes.