Capa de aplicación

Existen varios modelos de servicios en la capa de transporte, los más importantes son el modelo **cliente-servidor**, el modelo **Peer-to-peer (P2P)** y un modelo **híbrido entre ambas P2P y cliente-servidor**. Para poder enviar paquetes se utilizan diferentes protocolos tales como **HTTP**, **FTP**, **SMTP**, **POP3**, **IMAP**, **DNS** entre otros.

Las aplicaciones se comunican a través de la red, estas deben ejecutarse en varios sistemas finales (sistemas con la aplicación) y el software solo debe ser escrito en los elementos que usen la aplicación, no en los elementos intermedios ya que estos no ejecutan aplicaciones.

En la arquitectura cliente-servidor tenemos dos elementos principales que son el **cliente** y el **servidor**.

- El cliente se comunica con el servidor, este se puede conectar con el servidor de manera intermitente, puede tener una IP dinámica y los clientes no se comunican entre sí.
- El servidor esta siempre en un host, su dirección IP es permanente y para escalar el sistema se utilizan granjas de servidores.

La arquitectura P2P es:

- Descentralizada, ya que no existe un servidor central para el control, los participantes se comunican entre sí y todos los nodos actúan como clientes y servidores.
- Distribuida, ya que la información no está alojada en un solo sitio.
- La carga esta balanceada entre todos los participantes.
- La información es redundante para que sea más accesible.
- La disponibilidad es muy alta porque la caída de un host no bloquea un servicio.
- Los recursos están optimizados.

Como ejemplos de arquitecturas híbridas tenemos a Skype y los servicios de mensajería instantánea.

Las aplicaciones que se comunican en la red comunican sus procesos. Estos procesos son programas en ejecución de un sistema informático. Para comunicar diferentes sistemas informáticos se utilizan mensajes. En la arquitectura cliente servidor, el **proceso cliente** inicia la comunicación y el **proceso servidor** espera peticiones de los clientes.

Los procesos envían y reciben mensajes por la red a través de sus sockets. Los sockets son análogos a “puertas” (puertos). Podemos definir a un socket como la interfaz software por la que se envían paquetes, en un socket suele haber dos procesos principales (puerta de una casa):

- Proceso transmisor, envía los mensajes por el puerto. Este confía en la infraestructura de transporte al otro lado del puerto, la cual lleva los mensajes al socket en el proceso receptor del destino.
- Proceso receptor, recibe el mensaje y actúa sobre este.

Un socket es la interfaz entre la capa de aplicación y la capa de transporte de un host. También se conoce como **Interfaz de programación de aplicaciones (API, Application Programming Interface)** que opera entre la aplicación y la red.

Para que un proceso reciba un mensaje, éste debe tener un identificador único. Un host tiene una dirección IP única, pero ¿es suficiente la dirección IP? No, es necesario un número de puerto para identificar a un proceso dentro de un host. El servidor HTTP usa el puerto 80, el servidor e-mail usa el puerto 25.

Hemos dicho que para enviar mensajes hacen falta protocolos, hay diferentes tipos de protocolos, **de dominio público (abiertos)**, permiten interoperabilidad y están definidos en RCFs, y **propietarios**, en general no permiten interoperabilidad.

Las aplicaciones pueden tener o no una transferencia de datos confiable, la tasa de transferencia en aplicaciones implica que pueda haber aplicaciones con un ancho de banda sensitivo o aplicaciones

estáticas, el retardo de algunas aplicaciones debe de ser bajo para que estas puedan ser efectivas, por último, un aspecto importante es la seguridad.

Los dos principales protocolos son:

- TCP. Está orientado a la conexión ya que requiere de un acuerdo entre cliente y servidor. El transporte es confiable, sin pérdidas. El control de flujo impide que se sobrecargue el receptor. El control de congestión frena al transmisor cuando la red esta sobrecargada. No provee garantías de retardo ni ancho de banda mínimo.
- UDP. Transferencia de datos no confiable entre los procesos transmisor y receptor. **No** provee acuerdo entre los procesos, confiabilidad, control de flujo, control de congestión, ni garantías de retardo o ancho de banda.

Programación de sockets

Como mencionamos anteriormente, se usan APIs que permiten desarrollar aplicaciones para usar los servicios TCP.

Se definen los siguientes tipos de sockets en función de los servicios proporcionados:

- Stream sockets: Confiables, orientados a conexión. Utilizan los servicios de TCP. Los datos se envían sin errores y en el mismo orden que se envían. Hay control de flujo y no se imponen límites a los datos.
- Datagram Sockets: No confiables ni orientados a conexión. Utilizan los servicios de UDP. No hay 'handshake' antes de enviar los datos. Los datagramas se envían como paquetes independientes, el emisor añade explícitamente la IP y el puerto de destino a cada paquete, el receptor extrae la IP y puerto de destino del paquete. No hay garantías ni de que lleguen los paquetes ni de que lo hagan ordenados. No hay segmentación ni reconstrucción de los paquetes.
- Raw Sockets: Acceso niveles más bajos del protocolo TCP/IP. Accede a los niveles más bajos de la pila de protocolos (IP, ICMP). Se utiliza para probar nuevos protocolos.

Para que se produzca una conexión entre cliente y servidor:

1. El cliente debe contactar al servidor. El servidor debe estar ejecutándose previamente y también debe de haber creado un socket que espere el contacto de un cliente.
   El cliente contacta al servidor. Crea un socket TCP especificando la IP y puerto del proceso servidor. Cuando el cliente crea el socket el cliente establece conexión con el servidor TCP.
2. Cuando el servidor es contactado, crea un nuevo socket para que el proceso servidor se comunique con ese cliente en particular, aunque el servidor puede comunicarse con otros clientes.

Los servidores pueden ser **iterativos** y procesar peticiones de una en una, o pueden ser **concurrentes**, por lo que pueden atender a varios clientes a la vez.

Llamadas básicas a la biblioteca de sockets

Su función es iniciar un socket

Función

int sockfd = **socket**(int family, int type, int protocol)

family: Indica la familia de direccionamiento: AF_UNIX, AF_INET, AF_NS, AF_OS2, y AF_IUCV.

type: Indica el tipo de interfaz de sockets que se quiere utilizar: SOCK_STREAM, SOCK_DGRAM, SOCK_RAW y SOCK_SEQPACKET.

protocol: Protocolo, en el caso de INET: UDP, TCP, IP, o ICMP.

sockfd: Entero (similar a un descriptor de ficheros) devuelto por la llamada a socket(). Deberá pasarse como parámetro en las siguientes llamadas.

Su función es registrar un socket en un puerto

Función

int **bind**(int sockfd, struct sockaddr *localaddr, int addrlen)

sokcfd: Parámetro devuelto por la llamada socket()

localaddr, addrlen: Parámetros para recibir la dirección del socket.

Su función es indicar la disposición a recibir llamadas por un puerto

Función

int **listen**(int sockfd, int queue-size)

sockfd: Parámetro devuelto por socket()

queue-size: Entero que indica el número de peticiones de conexión que pueden ser encoladas por el sistema antes de que el proceso local ejecute un accept.

Su función es registrar un socket en un puerto

Función

int **accept**(int sockfd, struct sockaddr *foreign-address, int addrlen)

sokcfd: Parámetro devuelto por la llamada socket()

foreign-address: Estructura devuelta por la función con la información de dirección remota.

addrlen: Tamaño de la estructura devuelta por la función.

Su función es la de realizar la conexión a un servidor. Es la llamada típica de un proceso cliente

Función

int **connect**(int sockfd, struct sockaddr *foreign-address, int addrlen)

sokcfd: Parámetro devuelto por la llamada socket()

foreign-address: Estructura devuelta por la función con la información de dirección remota.

addrlen: Tamaño de la estructura devuelta por la función.

Su función es la de recibir y enviar datos a través del socket

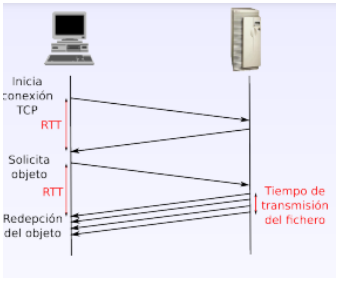
Funciones

ssize_t **readv**(int s, void *buf, size_t lon, int flags)
ssize_t **recv**(int s, void *buf, size_t lon, int flags)
ssize_t **readfrom**(int s, void *buf, size_t lon, int flags, struct sockaddr *desde, socklen_t *londesde)
ssize_t **recvmsg**(int s, struct msghdr *msg, int flags)
ssize_t **read**(int fd, void *buf, size_t nbytes)
ssize_t **send**(int s, const void *msg, size_t len, int flags)
ssize_t **sendto**(int s, const void *msg, size_t len, int flags, const struct sockaddr *to, socklen_t tolen)
ssize_t **sendmsg**(int s, const struct msghdr *msg, int flags) ssize_t **write**(int fd, void *buf, size_t nbytes)

HTTP (HyperText Transfer Protocol)

Normalmente una página web está formada por **objetos** que pueden ser un fichero HTML, una imagen… Cada objeto es direccionable por una URL. HTTP es el protocolo de la capa de aplicación para la web, este protocolo sigue un modelo cliente-servidor. El **cliente** es un "browser" que solicita, recibe y muestra los objetos web, el **servidor** recibe peticiones y este envía objetos.

HTTP utiliza **TCP**, el cliente inicia la conexión TCP con el servidor, el servidor acepta la conexión, se intercambian mensajes HTTP y se cierra la conexión.



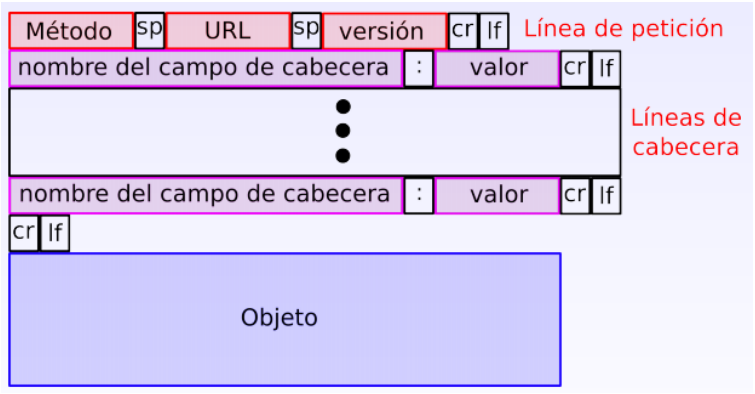
En los intercambios de mensajes existe un **RTT** que es el tiempo ocupado en enviar un paquete pequeño desde el cliente al servidor y su regreso (tiempo entre que se envía y se recibe).

Existen 2 tipos de HTTP, persistente y no persistente. En el no persistente como mucho se envía un objeto por conexión TCP, mientras que en el persistente pueden enviarse múltiples objetos entre el cliente y el servidor.

El HTTP no persistente tiene problemas, uno de ellos es que cada vez que quiere enviar un nuevo objeto debe repetir todo el proceso para enviar, esto incluye que el servidor espere, el cliente inicie la conexión, el servidor la acepte, el cliente envíe una solicitud al socket con la url, el servidor responda, después cierre la conexión y por último el cliente reciba el mensaje.

Para el HTTP persistente existen 2 tipos, **con pipelining** y **sin pipelining**. Sin pipelining el cliente envía un nuevo requerimiento solo cuando el previo se ha recibido, mientras que con pipelining el cliente envía requerimientos tan pronto este encuentre objetos referenciados.

PETICIÓN HTTP



Los métodos HTTP/1.0 son GET, POST y HEAD.

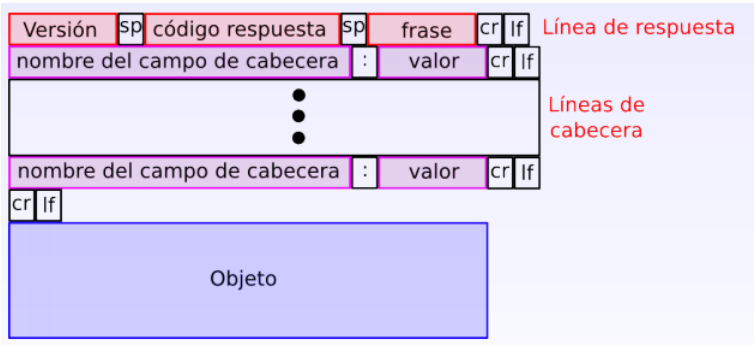
Los métodos HTTP/1.1 son GET, POST, HEAD, PUT y DELETE.

En el método GET, la url del objeto está en la cabecera del mensaje. Es el método más común, pero es más inseguro que POST.

En el método POST, el servidor dispone de un formulario de petición y la petición se realiza. según ese formulario como objeto del mensaje. Se usa típicamente para el envío de datos de formularios web, los datos se codifican en el cuerpo, y suelen ser procesados por un script en el servidor.

El método HEAD, es igual que el método GET pero el servidor solo devuelve cabeceras, es útil para determinar si un recurso existe, ha sido modificado, su tipo, longitud…

RESPUESTA HTTP



200 OK

301 Moved permanently

400 Bad request

404 Not Found

505 HTTP version not supported

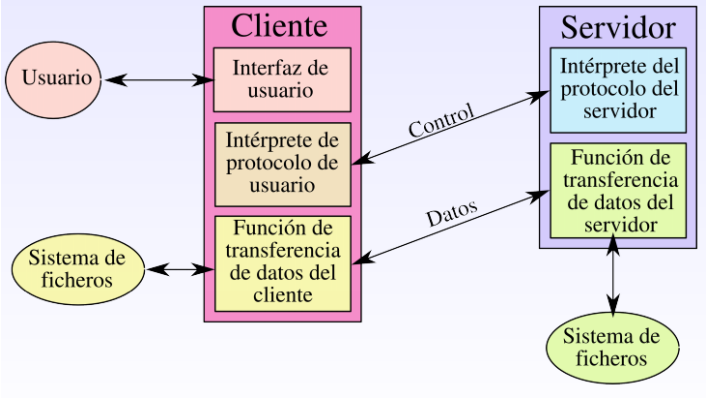
Las **cookies** siguen el protocolo RFC 2965, son datos que se guardan en el cliente y los utiliza el servidor para dar una mejor experiencia al usuario. Las cookies contienen, carritos, sugerencias, estado de la sesión. Las cookies permiten aprender sobre el usuario, se puede proveer nombre y correo.

Para hacer más amena la conexión existen **servidores proxy**/**web cache**. Estos actúan como clientes y servidores. Estos reducen el tiempo de respuesta de las peticiones del cliente, reducen el tráfico de los enlaces de internet y permiten a proveedores de contenido "pobre" entregar contenido de forma eficiente.

FTP (File Transfer Protocol)

FTP es un protocolo para transferir archivos entre dos sistemas. Usa una arquitectura cliente servidor en la que:

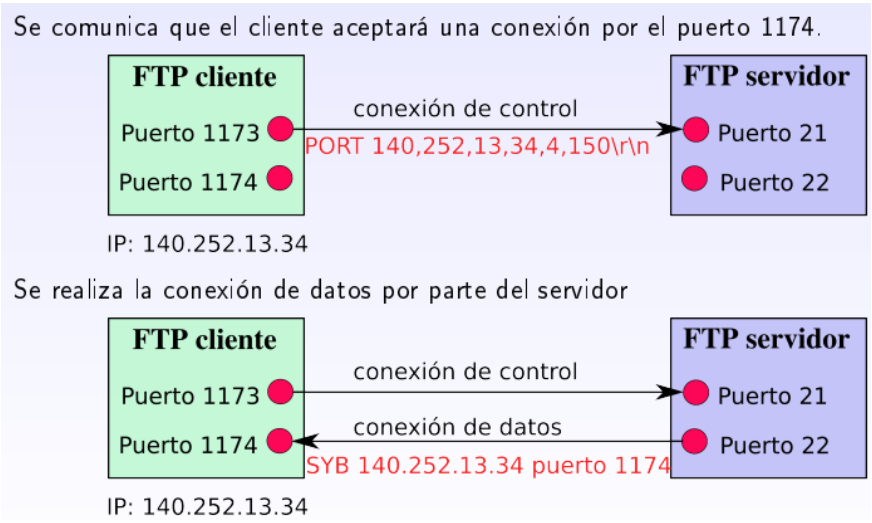
- El cliente tiene una interfaz de usuario, un intérprete de protocolo y una función de transferencia de datos del cliente.
- El servidor tiene un intérprete de protocolo y una función de transferencia de datos.



La comunicación entre el servidor y el cliente se hace mediante 2 conexiones, 1 para el control y otra para los datos. El cliente envía comandos que son contestados por el servidor. Esta conexión utiliza el protocolo NVT de telnet. Si el cliente pide un archivo se establece una nueva conexión para enviarlo.

Los archivos tienen 3 tipos de estructuras: **Estructura de fichero**, cadena de bytes, **estructura de registro**, archivo compuesto por una secuencia de registros, y **estructura de página**, cada página se transmite con un número que es almacenado por el receptor.

Este protocolo tiene 3 tipos de transmisión: **Modo stream**, los datos se envían tal y como están en el fichero, **modo bloque**, se incluyen cabeceras con cada bloque, y **modo comprimido**, se realizan operaciones de compresión y descompresión de los datos antes y después de la trasferencia.



Las conexiones tienen 2 modos, **modo activo** y el **modo pasivo**.

El modo activo, es el modo por defecto. El servidor inicia la conexión de datos, lo que puede generar problemas con cortafuegos en el cliente.

El modo pasivo, el cliente inicia las conexiones de control y datos. Hay que abrir un rango de puertos en el servidor para soportar este modo.

El protocolo FTP también tiene un servicio de reinicio, aunque no todos los servidores lo implementan. En este servicio el emisor transmite bloques con marcadores (cadenas de texto) de reinicio en puntos adecuados. Cuando el receptor recibe un marcador, almacena los datos en disco con la posición del marcador. Si el receptor es el cliente, notifica la recepción a la aplicación FTP, si el receptor es el servidor notifica al cliente por la conexión de control. En caso de fallo se reinicia indicando el valor del marcador adecuado.

SMTP (Simple Mail Transfer Protocol)

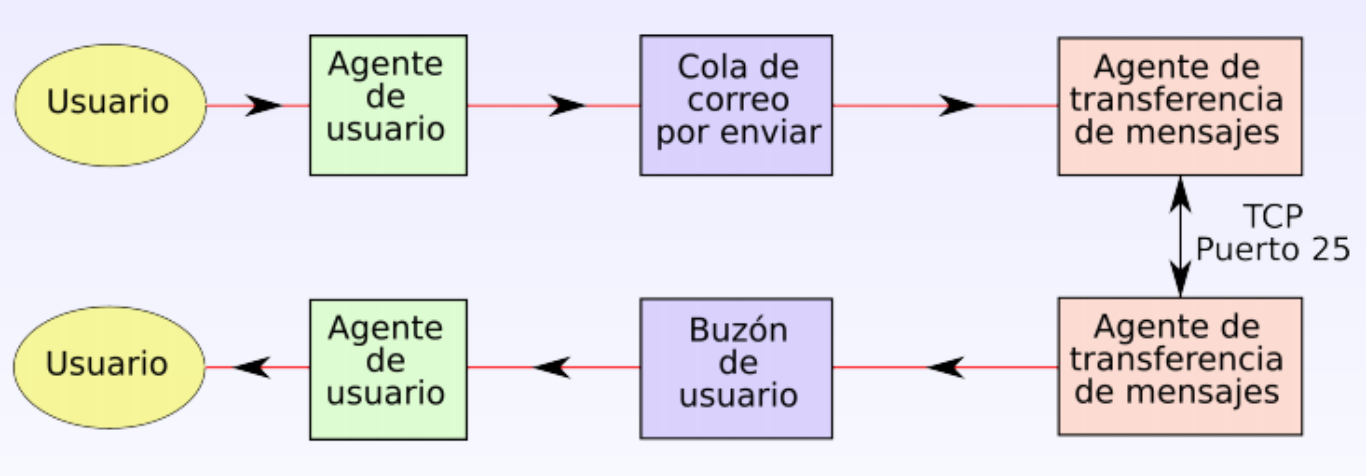
Los 3 principales componentes son: **Agente usuario**, **Servidor de correo** y **protocolo SMTP**.

El agente usuario, también conocido como "lector de correo", es aquel que escribe, edita y lee los correos. Los mensajes de salida y entrada son almacenados en el servidor.

El servidor de correo contiene un **casillero** el cual tiene mensajes de entrada para el usuario. También contiene una **cola de mensajes** de los correos de salida.

El protocolo SMTP:

- Usa **TCP** (**puerto 25**) para transferir confiablemente mensajes e-mail desde el cliente al servidor.
- La transferencia es **directa** ya que el servidor envía correos al servidor receptor.
- La transferencia está compuesta de **3 fases**: Handshaking, transferencia de mensajes y cierre.
- La interacción es por **comandos** (texto ASCII) y **respuestas** (código de estado y frase).
- Los mensajes deben ser enviados en ASCII de 7-bits.



SMTP es un protcolo push ya que se pone contenido en el servidor, también múltiples objetos son enviados en un mensaje multiparte.

Para la recepcion del correo existen 3 protocolos más:

- **POP**: Post office protocol. POP3 no mantiene el estado de una sesión a otra, pero mantiene una copia de los mensajes en diferentes clientes.
- **IMAP**: Internet Mail Access Protocol. Mantiene todos los mensajes en el servidor, permite que el usuario organice sus correos en carpetas. Permite tener acceso al correo electronico desde cualquier equipo con internet. Permite visualizar los mensajes sin descargarlos. Mantiene el estado de un usuario de una sesión a otra.
- **HTTP**: Analiza el servidor localmente y presenta los resultados al usuario via web

Todos estos protocolos permiten el uso de la capa SSL (Secure sockets layer) usando otros puertos.

DNS (Domain Name System)

Utilizado por aplicaciones TCP-IP para obtener la IP a partir del nombre del host. La aplicación cliente encargada de la resolución de nombres se denomina **resolver**. El *resolver* obtiene la información consultando un fichero *hosts* en la maquina, o contactando con un servidor de nombres. Este servicio se opera tambien en UDP, en ambos casos (TCP y UDP) operan en el **puerto 53**.

Las organizaciones deben disponer de un servidor local de nombres. Las consultas desde un cliente a una maquina interna serán resueltas en un servidor local. Las consultas desde un cliente a una maquina externa serán resueltas por el servidor local que contactará con el servidor host externo, pero antes de contactar mirara su propia cache.

La estructura de dns es en forma de **árbol**. Cada nodo tiene asignada una etiqueta (63 caractéres máximo) execpto el root. El nombre del dominio se forma añadiendo las etiquetas separadas por un punto, existe un FQDN (full qualified domain name) el cual se forma añadiendo también el nombre de la maquina a la que pertenece.

El nivel superior del arbol contiene:

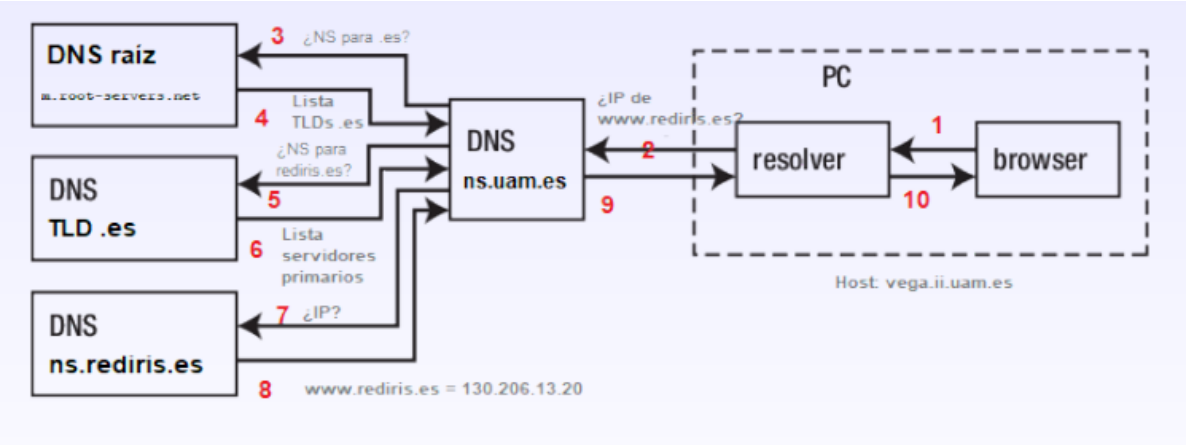
- Nodo raíz, que resuelve direcciones a nombres.
- Dominios genéricos asociados a actividades, com, edu, gov…
- Dominios geográficos de 2 caracteres, es, fr…

Los **TLDs** son dominios de primer nivel, y pueden ser genéricos (gTLD – generic TLD), geográficos o de código de país (ccTLD – country code TLD).

Cada nodo en el arbol tiene una **autoridad**, esta autoridad puede **delegar** la organización de los niveles más bajos de su nombre de dominio.

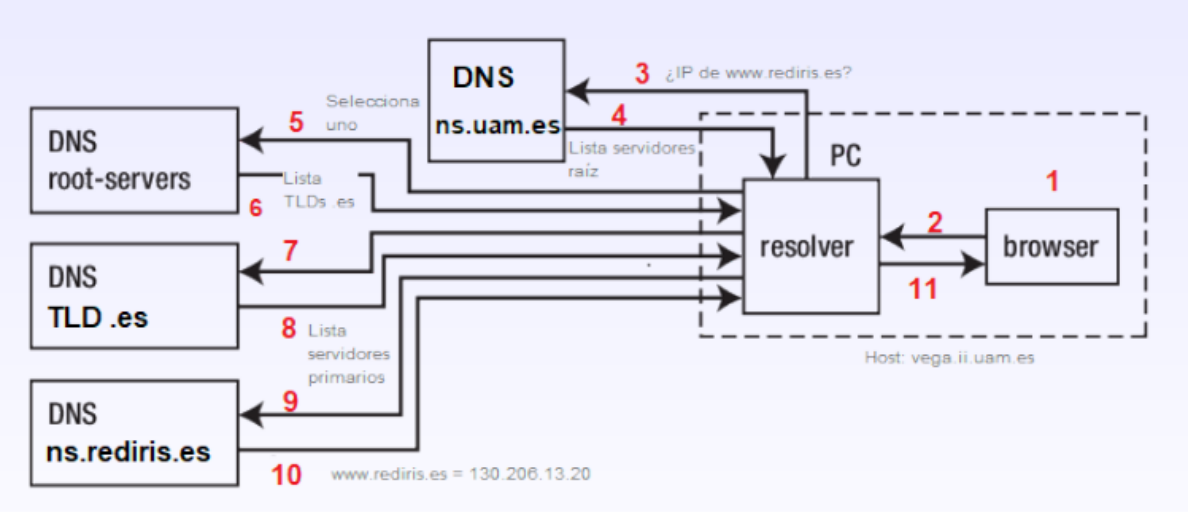
Existen 3 tipos de consultas que pueden hacerse a un servidor DNS:

- Recursiva: El servidor de nombres dará la respuesta completa, haciendo todas las subconsultas que sean necesarias.



1. El usuario solicita la traducción del dominio.
2. El navegador le envía la pregunta al *resolver* del PC.
3. Consulta al servidor DNS local.
4. La respuesta no está en su caché. Al ser una pregunta recursiva, él se encarga del resto del proceso. Selecciona uno de los servidores raíz y pregunta por los NS del dominio ".es".
5. El DNS elige una entrada de la lista recibida, y pide la lista de servidores primarios del subdominio .rediris.es.
6. El DNS selecciona una entrada de la lista recibida y realiza la pregunta final: ¿.rediris.es?
7. El servidor DNS autoritario del dominio devuelve la IP solicitada.

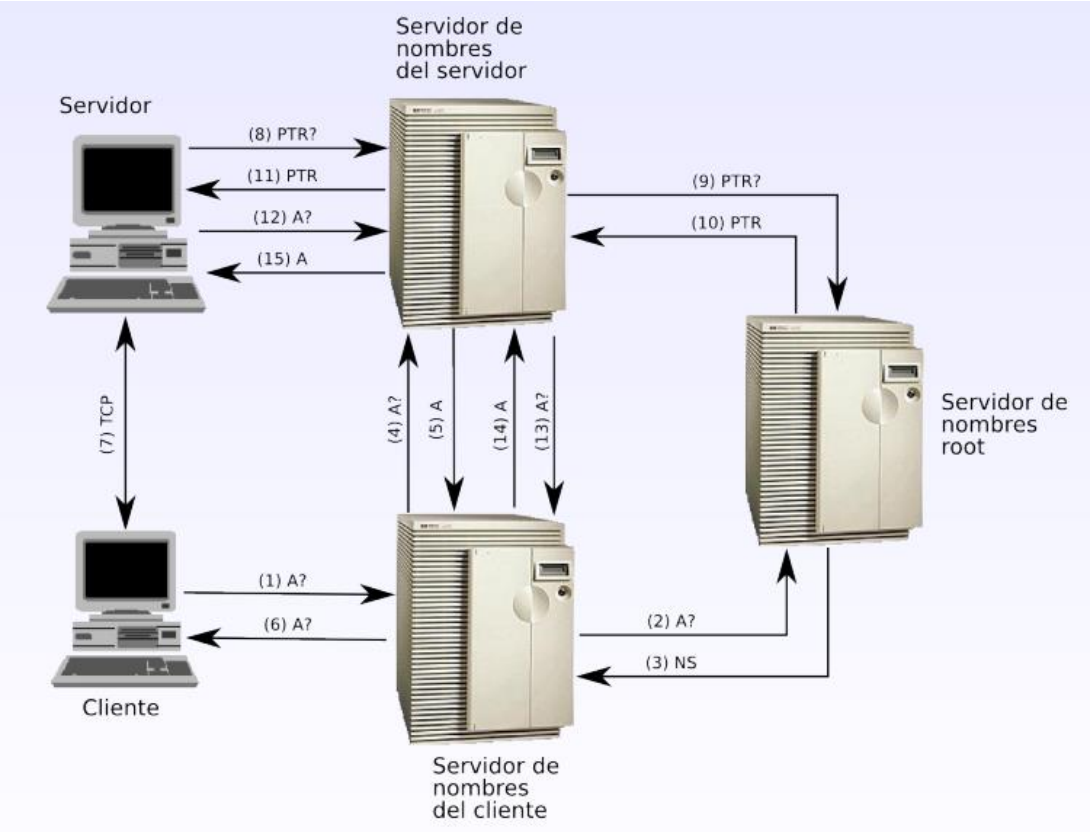
- Iterativa: El servidor de nombres da una respuesta parcial, y es responsabilidad del cliente "seguir preguntando".



1. El usuario solicita la traducción del dominio.
2. El navegador le envía la pregunta al *resolver* del PC.
3. Consulta al servidor DNS local.
4. La respuesta no está en su caché. Al ser pregunta iterativa, devuelve la lista de los servidores raíz.
5. El resolver del PC selecciona uno de los servidores raíz y le envía la pregunta.

6. El servidor raíz solo soporta consultas iterativas y responde con la lista de IPs de los servidores de nombres autoritarios del dominio .es (ccTLD).
7. El resolver del PC elige una IP de la lista recibida, y le envía la pregunta a dicho servidor de nombres.
8. El servidor DNS del ccTLD .es solo soporta consultas iterativas, y responde con la lista de IPs de los servidores DNS autoritarios del siguiente nivel.
9. El resolver del PC elige una IP de la lista recibida, y le envía la pregunta a dicho servidor de nombres.
10. El servidor DNS autoritario del dominio devuelve finalmente la IP solicitada.

- Inversa: Rersponden a *¿qué dominio corresponde a esta IP?*. Se resuelven a través de consultas iterativas o recursivas de dominio especial .IN-ADDR.ARPA.

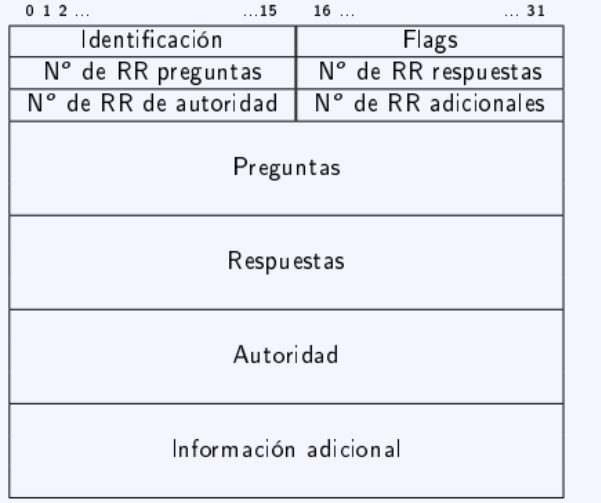


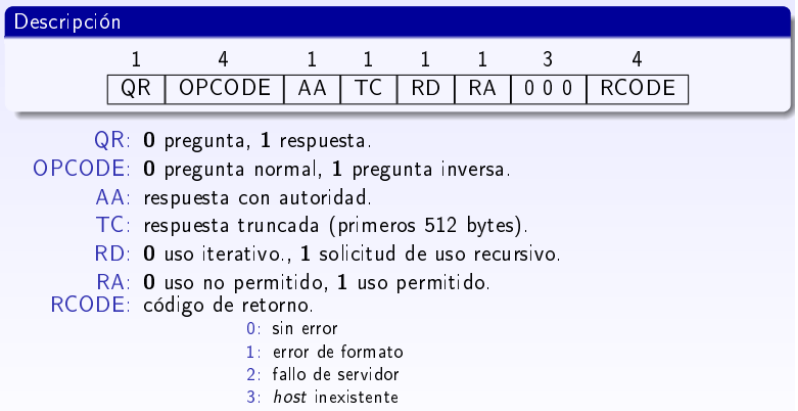
1. El cliente consulta la dirección del ordenador de destino a su servidor de nombres.
2. El servidor de nombres consulta a servidores superiores.
3. Los servidores superiores le devuelven la dirección del servidor de nombres autoridad sobre la dirección a la que se pretende conectar.
4. Se realiza la consulta al servidor de nombres identificado.
5. El servidor de nombres del servidor devuelve la dirección correspondiente.
6. El servidor de nombres del cliente devuelve la dirección del ordenador con el que queire conectar.
7. Se realiza la conexión sin validación.
8. Se inicia la validación por parte del servidor realizando la consulta inversa solicitando el nombre de la máqiema que se ha conectado.
9. Usando ARPa el servidor raíz devuelve la dirección al servidor de nombres del servidor.
10. El servidor de nombres del servidor le pasa la dirección al servidor.
11. El servidor le pregunta a su servidor de nombres por la dirección asociada a ese nombre.
12. Se realiza la consulta al serviodr autoridad del cliente. No es necesario consultar al root pues ya se tiene esa dirección en caché de la conexión anterior.
13. El servidor de nombres autoridad del cliente le devuelve la dirección al servidor de nombres del servidor
14. El servidor de nombres del servidor le devuelve la dirección al servidor.

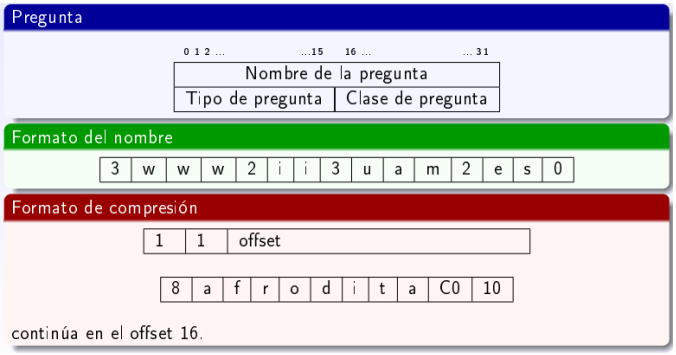
15. Si esta dirección coincide con la de la comunicación inicial se comprueba que la comunicación es correcta y que no hay suplantación.

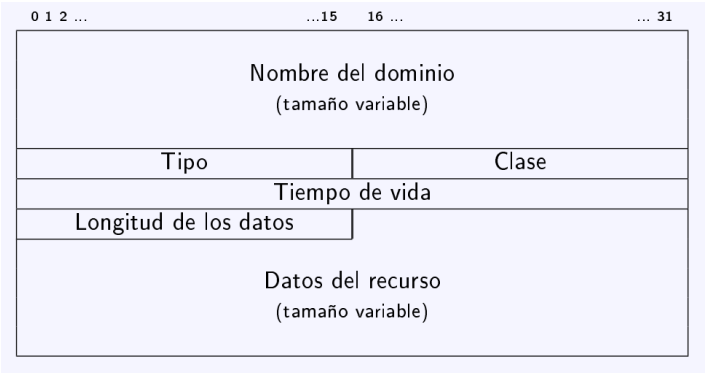
Las respuestas almacenadas en cache tienen un **tiempo de vida** (TTL, time to live), si este pasa y no es utilizada de nuevo, la respuesta se elimina de la cache.

Formato del protocolo.









DHCP (Dynamic Host Configuration Protocol)

El protocolo DHCP suministra una **dirección IP**, una **máscara de subred** y un **router por defecto**. Este protocolo facilita la administración y reduce el número total de direcciones IP. El único inconveniente es que la información en el DNS no está actualizada, hay que usar Dynamic DNS (DDNS).

El protocolo soporta 3 mecanismos para asignar direcciones:

1. Automática: La dirección es permanente para el cliente.
2. Dinámica: La dirección está limitada a un periodo de tiempo.
3. Manual: La dirección IP se configura manualmente en el cliente.

Opcode

DISCOVER:	Usado en difusión por el cliente.
OFFER:	Respuesta del servidor con la dirección IP y otros parámetros.
REQUEST:	Solicita los parámetros ofrecidos por uno de los servidores. También se usa para extender en el tiempo la validez de una dirección IP.
PACK:	Reconocimiento por el servidor de los parámetros a utilizar.
NPACK:	
:	Reconocimiento negativo. No puede seguir usando la dirección IP o es incorrecta.
DECLINE:	El cliente comunica al servidor que la dirección IP esta en uso.
RELEASE:	El cliente comunica que no seguirá usando la dirección IP asignada.
INFORM:	Mensaje enviado por el cliente que ya conoce su dirección IP, para obtener otros parámetros de configuración.

Tipo HW:	Ethernet (1). IEEE-802 (6).
Longitud:	n^0 de bytes de la dirección física
Saltos:	Puesto por el cliente a 0, se incrementa por cada maquina *relay* que redirige el mensaje
Tiempo:	Segundos transcurridos desde que se inició el intento de arranque de la maquina.
Flags:	Bit 15. Difusión (1). Se utiliza para indicar al servidor que el cliente solo puede recibir mensajes en difusión (no conoce su IP).
Dirección IP cliente:	Puesto por el cliente, puede ser 0.0.0.0 o su dirección IP si la conoce.
Dirección IP asignada:	Dada por el servidor.
Dirección IP del router de reenvío:	Introducido por una maquina relay DHCP, para que el servidor de DHCP le devuelva la contestación.
Nombre del servidor DHCP:	Optativo.
Nombre del archivo de arranque:	Optativo. Indicado por el servidor para indicar al cliente un archivo de arranque en el cliente o en un servidor de configuración.

El modo de operación del protocolo es el siguiente:

1. El cliente manda en difusión un mensaje DISCOVER, que puede incluir como opciones una posible dirección IP y un tiempo de validez.
2. Cada servidor puede responder con un mensaje OFFER que incluye una dirección IP y otras opciones de configuración. Se envía con MAC del cliente y con IP 255. 255. 255. 255. Los servidores recuerdan las direcciones ofertadas evitando ofrecerlas en posteriores mensajes de DISCOVER.
3. El cliente elige una de las direcciones IP y manda en difusión un REQUEST donde índice el servidor y la dirección IP elegida. Los servidores que reciben este mensaje y no son elegidos saben que su oferta ha sido denegada.
4. El servidor elegido manda un PACK confirmando la dirección IP y otros parámetros de configuración, así como dos tiempos T1 y T2 relacionados con el tiempo máximo T en que la dirección es válida.
5. Cuando T1 expira, el cliente manda un REQUEST al servidor solicitando extender la validez. Si el servidor está de acuerdo manda un PACK con nuevos valores T1 y T2.
6. Si el mensaje PACK no llega cuando T2 expira, el cliente manda un mensaje REQUEST (difusión). Esta solicitud puede ser contestada por alguno de los servidores con un mensaje PACK.
7. Si el cliente no recibe PACK y el tiempo y vence el tiempo T, el cliente debe dejar de usar la configuración IP actual.
8. Un cliente puede dejar de usar la configuración IP actual e iniciar el proceso con mensaje DISCOVER.

Existe también un **agente de router de envío**, este se emplea en el caso de el servidor DHCP no este en la misma subred que el cliente. Cuando el agente detecta una solicitud introduce la dirección IP de la red por la que llegó la petición en el campo Router Relay y lo reenvía al servidor remoto DHCP. La respuesta del servidor se hace a través del agente Router de Reenvío.

Aplicaciones P2P

La arquitectura P2P no tiene necesidad de servidores ya que la comunicación se produce entre 2 sistemas finales arbitrarios, los pares se conectan intermitentemente y cambian sus IPs.

En esta arquitectura se tarda lo siguiente para enviar un fichero:

$$t_{p2p} = \max\left(\frac{F}{u_s}, \frac{F}{\min(d_i)}, \frac{NF}{u_s + \sum u_i}\right)$$

- u_s es el ancho de banda de subida del servidor.
- F es el tamaño del archivo a subir.
- d_i es el ancho de banda de bajada de un cliente.

  La primera columna corresponde al tiempo que tardaría el servidor en subir un archivo, la segunda columna corresponde al tiempo que tarda el nodo más lento en descargarlo, y la tercera sería el tiempo que tarda toda la red P2P en transmitir el archivo.

En la arquitectura cliente servidor tardaría esto:

$$t_{cs} = \max\left(\frac{NF}{u_s}, \frac{F}{\min(d_i)}\right)$$

En la arquitectura P2P tenemos **DHT (Distributed Hash Table)** que es la base de datos distribuida de P2P. La base de datos contiene pares **(clave, valor)**, la clave identifica al usuario y el valor identifica al contenido. Los pares (peers) consultan la base de datos con la clave. Los pares (peers) solo pueden insertar pares **(clave, valor)** en la base de datos.