Tema 3. Nombrado, acceso, archivos

Ingeniería de Servicios

2023-2024

T3.1 Nombrado

DNS (*Domain Name Server*)

- Problema: convertir nombres de máquinas en sus correspondientes IPs
- Solución inicial: un archivo (`hosts.txt`) contenía la información. Nombres "planos"

Problemas

- Actualización manual del archivo. Distribución "manual" por FTP.
- Colisiones de nombres.
- Consistencia (múltiples nombres).

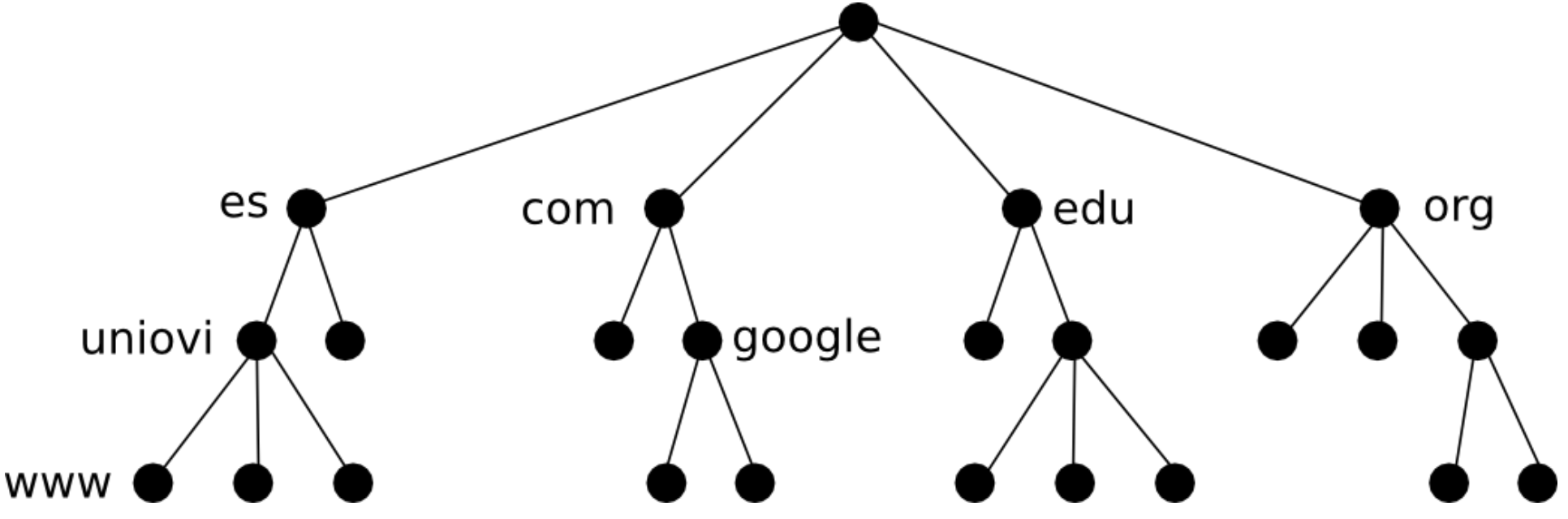
Solución Sistema de nombres jerárquico. Base de datos distribuida ⇒ DNS

Arquitectura básica

- Arquitectura cliente/servidor:
  - Clientes: se denominan *resolvers*
  - Servidores: se denominan *nameservers*
- Espacio de nombre jerárquico. Se representa en un árbol.
- Base de datos distribuida entre muchos servidores
- Protocolos para cliente/servidor o entre servidores

El espacio de nombres

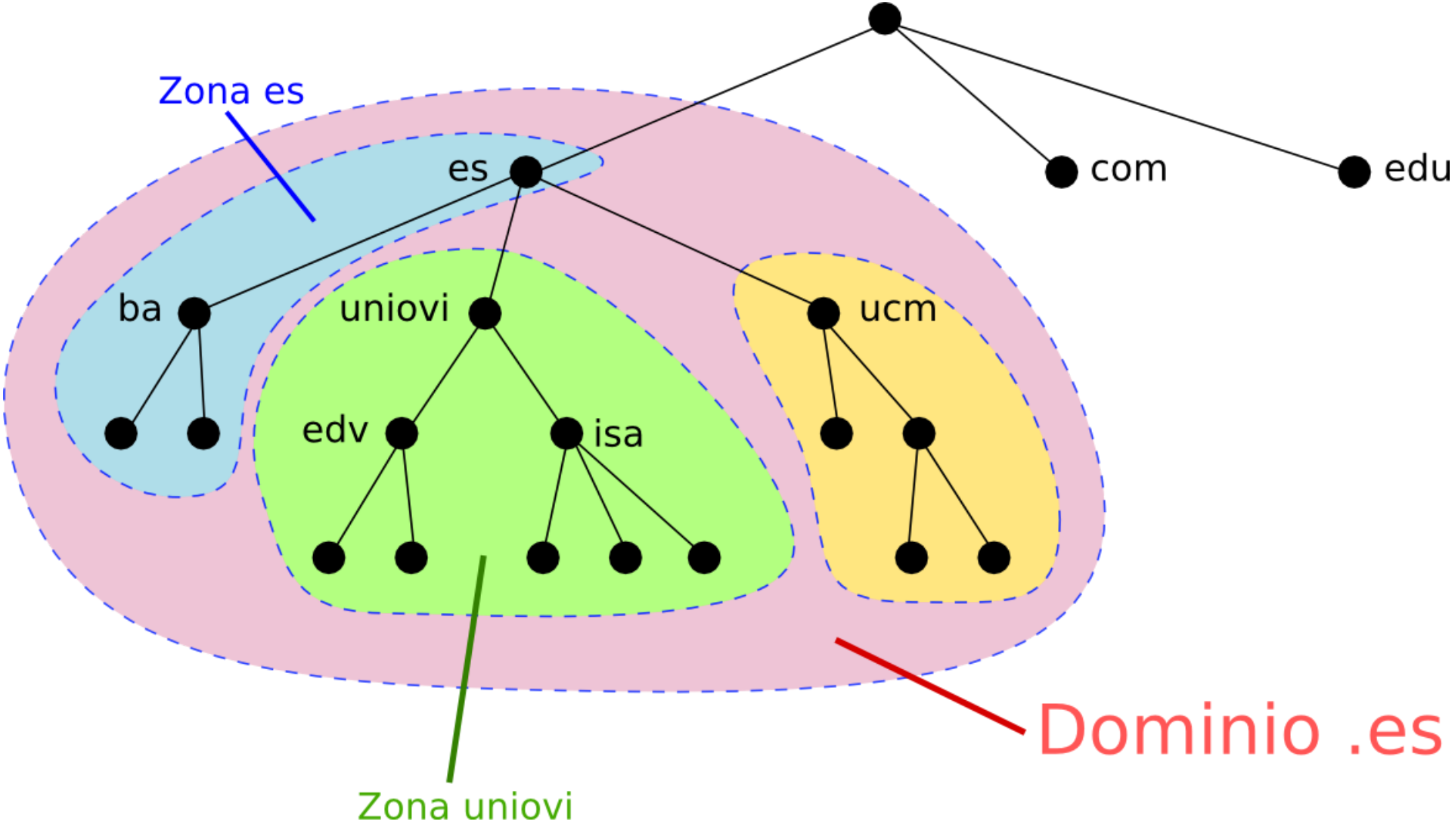
- Se puede representar por un árbol
- Cada nodo es a su vez raíz de otro árbol llamado **dominio**
- Los nodos terminales (hojas) representan *hosts*. Los intermedios pueden respresentar *hosts*, dominios, o ambos.
- El nombre del dominio se construye "leyendo hacia arriba", concatenando con puntos.



Zonas

- Una zona es una partición de nodos conectados en ese árbol.
- Puede ser un sub-dominio completo, o sólo parte de él.
- Las zonas permiten *delegar* la gestión de una parte del espacio de nombres.
- Cada zona es controlada y manejada por una organización (y unos servidores) diferentes.

Diferencia entre zona y sub-dominio



Dominios de nivel superior (TLD)

- Son los hijos del nodo raíz (en inglés: *top-level domains*)
- Dos variedades:
  - Geográficos (ccTLDs): hay uno para cada país (`es`, `uk`, `jp`, etc.)
  - Genéricos (gTLDs) (`com`, `org`, `edu`, etc.)
- Extensiones recientes:
  - [2000] nuevos gTLDs: `aero`, `biz`, `info`, `name`, `pro`, etc.
  - [2009] Nombres de dominio internacionalizados (con caracteres unicode codificados de forma especial, punycode)
  - [2014] Cientos de nuevos gTLDs: `music`, `free`, `shop`, `doctor`, etc. etc. (y cientos de marcas comerciales), e internacionalizados además (ej: dominio .商店)

La base de datos

- Un nombre de dominio es el *índice* de la BD
- El contenido de la BD para cada índice es un *Resource Record* (RR) con información como:
  - Si es un nodo final (*host*) o una sub-zona
  - Si ese nodo es a su vez un servidor de nombres
  - La IP de la máquina
  - Otros nombres para la máquina (y su nombre *canónico*)
  - Quién es la persona de contacto, su email
  - etc (por ejemplo, geolocalización)

Implementación

- DNS es una especificación, puede haber diferentes implementaciones
- La más usada es `BIND` (*Berkeley Internet Name Domain*)

En la implementación BIND se tienen:

- Servidores de nombres (proceso `named`), complejos
- Clientes *resolver*, muy simples. No son un proceso sino una biblioteca que implementan `gethostbyname()`

Servidor de nombres

- Es un programa que tiene información completa sobre una *zona*
- Sobre esa *zona* tiene *autoridad*. Sus respuestas son "definitivas" (*authoritative*)
- Al delegar parte de la zona:
  - El SN original "borra" la información referente a la nueva-subzona
  - El SN de la nueva zona es la nueva autoridad para ella.
- Una zona puede tener varios NS (redundancia para seguridad y equilibrado de carga).
  - *Primario*: Obtiene los datos de un fichero local
  - *Secundario*: Obtiene los datos de un NS primario
  - La única diferencia es el origen de los datos. Ambos pueden ser *autoridad*.

Resolvers

- Es el cliente que consulta al servidor de nombres.
- Su misión es:
  - A petición de otro programa, conecta con el SN y hace una consulta
  - Interpreta la respuesta
  - Si es necesario, hace más consultas
  - Devuelve la información solicitada

Resolución

- Es el proceso por el cual se llega a una respuesta:
  - Se encuentra la IP asociada con el nombre solicitado
  - O se determina que ese nombre no existe

Ya que cada NS almacena sólo parte de la BD distribuida una consulta a un NS puede no encontrar la respuesta.

En ese caso hay dos alternativas:

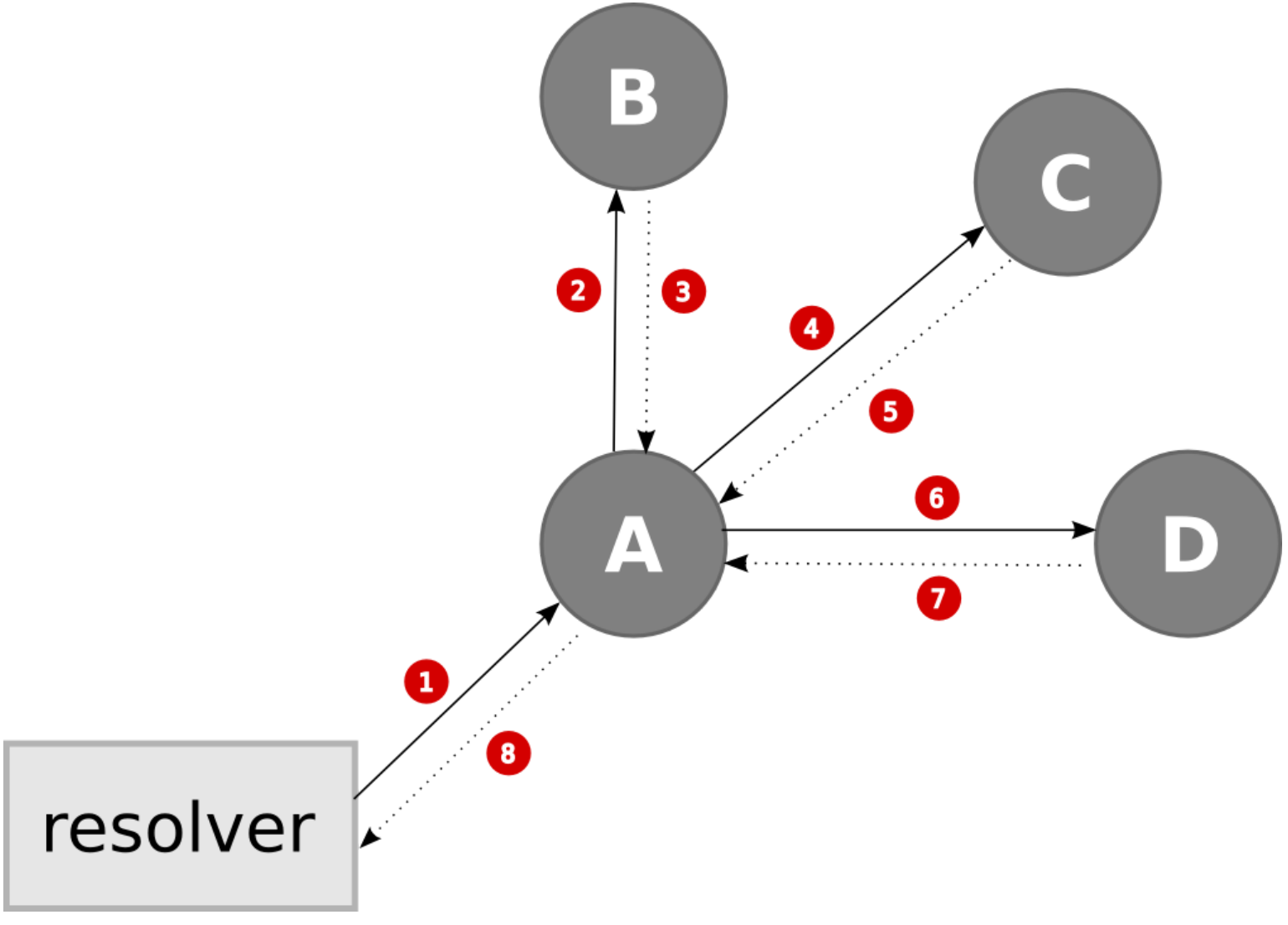
1. El servidor contactará con otros servidores hasta encontrar la respuesta (*consulta recursiva*)
2. El servidor retornará una referencia a otros servidores para que quien hizo la consulta prosiga la resolución (*consulta iterativa*)

Implementación en BIND

- En BIND la consulta del *resolver* es siempre de tipo *recursivo*.
  - Esto simplifica el código del *resolver*
  - Que es una mera biblioteca
- El *Servidor de Nombres* consultado:
  - Si tiene la respuesta, la retorna.
  - Si no la tiene usa consulta de tipo *iterativo* para hallarla.

> La carga de la resolución recae en el primer servidor consultado.

Ejemplo



(explicación de los números en transparencia siguiente)

Ejemplo (explicación)

1. El *resolver* hace una consulta recursiva al NS *A*, para la IP de `sirio.atc.uniovi.es`
2. *A* no sabe la respuesta y tampoco sabe quién es el NS que se la podría dar. Hace una consulta iterativa a *B* que es un NS-raíz
3. *B* conoce al NS que gestiona el TLD `.es` (*C*). Retorna una referencia a la IP de ese NS
4. *A* hace una consulta iterativa a *C*
5. *C* desconoce la respuesta, pero conoce al NS que gestiona la zona `.uniovi.es` (*D*). Retorna una referencia a la IP de ese NS.
6. *A* hace una consulta iterativa a *D*
7. *D* conoce la IP de `sirio.atc.uniovi.es` (no hay más subzonas) y retorna la **respuesta**. Además *D* es la *autoridad* para esa zona, la respuesta es "definitiva".
8. *A* retorna la respuesta que ha averiguado al *resolver*.

Información en la base de datos

Cada registro de la base de datos contiene:

- El índice que es el nombre de dominio (ej: `www.uniovi.es`)
- Varios RR (*Registry Record*) con la información relativa a ese dominio

Ejemplo: parte del RR de `uniovi.es`

```
uniovi.es.  172800  IN  SOA enol.si.uniovi.es. \
                            dnsmaster.si.uniovi.es.\
                            2013092701 7200 7200 2419200 172800
uniovi.es.  172800  IN  A   156.35.33.105
uniovi.es.  172800  IN  TXT "MS=ms34896724"
uniovi.es.  172800  IN  LOC 43 21 13.000 N 5 52 24.000 W 228.00m 1m 10000m 10m
uniovi.es.  172800  IN  MX  10 primera.net.uniovi.es.
uniovi.es.  172800  IN  MX  20 llar.net.uniovi.es.
uniovi.es.  172800  IN  MX  20 xanes.net.uniovi.es.
uniovi.es.  172800  IN  NS  zeus.etsimo.uniovi.es.
uniovi.es.  172800  IN  NS  coruxa.epsig.uniovi.es.
uniovi.es.  172800  IN  NS  vci.uniovi.es.
```

Tipos de registro

- **SOA**: Start of Authority ⇒ Contiene información sobre la zona, quién es el NS con autoridad para ella, etc
- **NS**: Name Server ⇒ Contiene información sobre servidores de nombre para ese dominio
- **A**: Address ⇒ Contiene la IPv4 asociada a ese dominio o host
- **AAAA**: Address IPv6
- **CNAME**: Canonical Name ⇒ Contiene el nombre oficial de ese dominio (el índice es un alias)
- **PTR**: Pointer ⇒ Contiene el nombre oficial de ese dominio, para la resolución inversa
- **MX**: Mail Exchange ⇒ Información para el protocolo SMTP sobre cómo hacer llegar correo a ese dominio
- Otros: qué tipo de máquina es, su operativo, su administrador, su geolocalización, etc...

Resolución inversa

Problema: Dada una IP averiguar el nombre de la máquina.

Es habitual necesitar esta "resolución inversa" para, por ejemplo, mostrar nombres de máquinas en los ficheros *log* de conexiones, etc.

En la BD el índice es el *nombre* de la máquina, la IP es un dato del RR

¿¿Esto implica que para encontrar la IP hay que mirar *uno a uno* **todos** los RR??

Resolución inversa, solución

- Se define un dominio especial: `in-addr.arpa`
- Este dominio tiene sus *Servidores de Nombres* como cualquier otro dominio.
- El dominio puede estar dividido en zonas delegadas, como cualquier otro dominio.

Idea clave

Los nombres de los subdominios de `in-addr.arpa` son números entre 1 y 255, y éstos se subdividen en sub-dominios con nombres numéricos también

Así, podemos encontrar por ejemplo:
`105.33.35.156.in-addr.arpa`

Resolución inversa a través de in-addr.arpa

- El nombre del subdominio es la IP "al revés".
  - Esto simplifica la asignación de NS de zona para resolver este tipo de nombres
  - El NS de la zona `156.in-addr.arpa` estará en la sub-red `156.*`
- El RR correspondiente a `105.33.35.156.in-addr.arpa` contiene el nombre de esa máquina.

Por tanto, para encontrar el nombre de `156.35.33.105`, el *resolver* hace una consulta por el "nombre" `105.33.35.156.in-addr.arpa`.

La consulta se resuelve "normalmente", pero la respuesta no contiene una dirección, sino un PTR

Cacheado

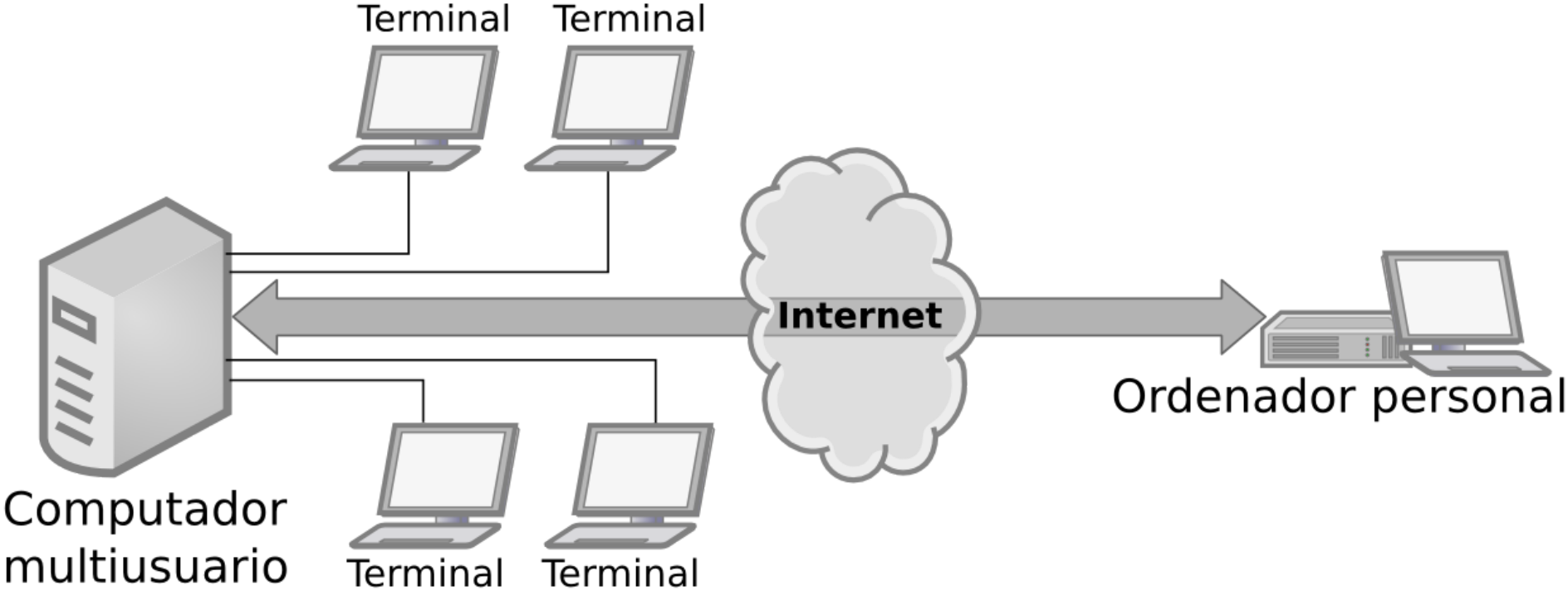
- Cada Servidor de Nombres puede mantener una *cache* con:
  - Las respuestas que ha servido
  - Las respuestas negativas que ha servido
  - Las IPs de los NS de otras zonas
- Esto evita que tenga que acudir cada vez a los NS-raíz

- La cache debe descartarse pasado un cierto tiempo
- De lo contrario los cambios en el DNS no se propagarían
- **TTL**: (*Time to Live*) es un parámetro fijado por el administrador de la zona

3.2 Acceso remoto

El problema

- Un potente servidor multiusuario (ej: Unix)
- Puede tener varias terminales localmente conectadas (vía serie)
- ¿Podrían conectarse terminales remotas vía internet?



Soluciones

Depende del tipo de terminal y del operativo:

- Terminal de texto, operativo tipo Unix (instalable también en Windows)
  - Telnet (inseguro, sin cifrado)
  - SSH (seguro, cifrado)
- Terminal gráfica:
  - X-Window (sólo en servidores tipo Unix, el cliente puede ser Windows)
  - RDP (*Remote Desktop Protocol*) Protocolo propietario de Microsoft. Servidores Windows. Clientes para diferentes plataformas.
  - RFP (*Remote Framebuffer Protocol*) Protocolo libre. Múltiples plataformas (tanto servidor como cliente)

Terminales de texto

- Telnet y SSH son similares. Ambos:
  - Requieren el concepto de "Terminal remota virtual" (una terminal emulada)
  - Requieren identificación y autenticación del usuario
  - Mantienen una "sesión" en la que el usuario interactúa con un *shell*
- Difieren en:
  - La forma de identificar y autenticar
  - El transporte de la información (SSH va cifrado)
  - La versatilidad (SSH permite también transferir ficheros, crear túneles cifrados, etc)

Telnet

Telnet

Se basa en tres conceptos clave (RFC 854):

- *Terminal de red virtual* (NVT). Una abstracción de las terminales físicas reales.
- Negociación de opciones. La terminal física real puede tener o no todas las características de la NVT.
- Simetría de los extremos. No se trata de forma diferente el cliente o el servidor, ambos se consideran dos NVT.

NVT

Las características de funcionamiento de cada terminal física son diferentes.

> El protocolo telnet no puede tener previstos todos los tipos de terminal en existencia o futuros.

En lugar de eso:

- Define un "denominador común", la NVT
- Cada servidor o cliente telnet debe traducir las características de su terminal a las de esa NVT, y viceversa

Negociación de opciones

Muchas terminales podrán ser más avanzadas que el denominador común NVT

Para cubrir estos casos al inicio de la conexión telnet tiene lugar un proceso de negociación.

- Propuesta de activar o desactivar características
- Aceptar o rechazar la propuesta

> Ambas partes *deben acordar al menos la NVT básica*.

Negociación

El "lenguaje" de negociación se basa en los mensajes (son en realidad binarios)

- `WILL` X: Una parte anuncia que usaría la característica X
  - `DO` X: Respuesta positiva, X será usada
  - `DON'T` X: Respuesta negativa, X no será usada
- `DO` X: Una parte solicita a la otra que use la característica X
  - `WILL` X: Respuesta positiva, X será usada
  - `WON'T` X: Respuesta negativa, X no será usada

Este "lenguaje" puede extenderse para una característica X si ambos la soportan y por tanto entienden el correspondiente lenguaje para ella.

Protocolo general de Telnet

- El protocolo de Telnet es binario
- Los datos comunicados han de ser ASCII de 7 bits (pero puede negociarse de 8 bits)
- El valor 0xFF (255) tiene significado especial:
  - Se denomina `IAC` (*Interpret As Command*)
  - Indica que lo que sigue no forma parte de los datos, sino que es un *comando Telnet*
  - Tras él viene otro byte que es el código del comando
  - Algunos comandos requieren un tercer byte detrás con las opciones del comando.

Ejemplos de comandos telnet

Código	Hex	Nombre	Significado
243	f3	BRK	La tecla Break ha sido pulsada
244	f4	IP	*Interrumpir Proceso* conectado a esta terminal
247	f7	EC	*Erase Character* borra el caracter anteriormente recibido
253	fd	DO	Solicito una característica (la cual se indica en el byte siguiente)
251	fb	WILL	Acepto la característica (que se indica en el byte siguiente)

etc...

Ejemplos de características negociables

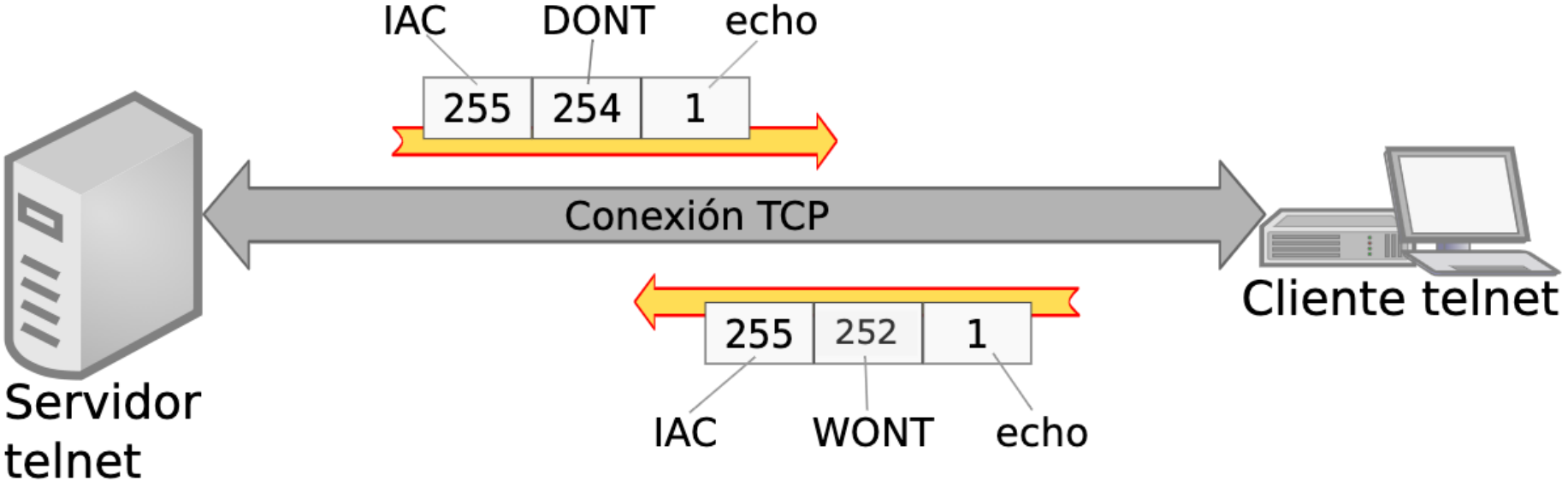
Cada característica se detalla en un RFC aparte, junto con el protocolo de los bytes adicionales que se deben transmitir cuando ésta se negocia.

Código	Nombre	RFC	Significado
1	echo	857	Solicita al otro extremo que haga eco
34	linemode	1184	Si debe haber búfer de línea o no en el cliente
36	environment variables	1408	Se van a transmitir variables de entorno del cliente

etc..

Ejemplo de negociación de una opción

El servidor pide al cliente que no haga "eco" (p.ej. se va a solicitar una contraseña). El cliente lo acepta.



SSH

SSH: *Secure Shell*

En realidad SSH no es tan solo un protocolo de terminal remota.

> SSH-2 introduce un nuevo nivel de multiplexado de comunicaciones, por encima del *puerto* TCP: el ***canal***.

- Cliente y servidor SSH se conectan por TCP
- Cliente y servidor se autentican mutuamente mediante claves públicas
- Una vez establecida, la conexión puede transportar varios "canales"
- Cada paquete enviado entre cliente y servidor por SSH lleva un "identificador de canal"
- Todos los paquetes van cifrados con algoritmos acordados en la fase de negociación

SSH: Funcionamiento

La **sesión SSH** se desarrolla en fases:

1. Establecimiento de una conexión segura entre cliente y servidor
   - Identificación del servidor
   - Negociación de los algoritmos de cifrado y clave a usar
2. Autenticación del cliente
   - Hay varios métodos para ello y se prueban todos hasta que uno funcione
3. Creación de los canales ssh
4. Uso de los canales ssh
5. Cierre de la conexión TCP

SSH2 son varios protocolos

Relacionados con las fases anteriores:

- SSH-TRANS (RFC 4253) se ocupa de la fase 1 y de transportar la información cifrada para el resto de la sesión.
- SSH-USERAUTH (RFC 4252) se ocupa de la fase 2, apoyándose en SSH-TRANS
- SSH-CONNECT (RFC 4254) se ocupa de la fase 3, apoyándose también en SSH-TRANS

SSH: Claves criptográficas usadas

Clave de anfitrión (*host key*)

- Es una clave asimétrica (privada/pública) usada por el servidor
- Sirve para **autenticar al servidor** ante el cliente
- Es creada por el administrador cuando el servidor se instala (o reinstala), usando `ssh-keygen`
- Queda almacenada en disco
- Es la misma en cada sesión (a menos que el servidor se reinstale)

¡Problema! La parte pública debe ser conocida de antemano por el cliente (¿Certificados?)

SSH: Claves criptográficas usadas

Clave de usuario (*user key*)

- Es una clave asimétrica (privada/pública)
- Sirve para **autenticar al cliente** ante el servidor (es uno de los posibles mecanismos en el paso 2)
- Es creada por el usuario en la máquina cliente usando `ssh-keygen`
- Queda almacenada en disco
- Es la misma en cada sesión (a menos que el usuario cree otra)

¡Problema! La parte pública debe ser copiada al servidor

SSH: Claves criptográficas usadas

Clave de sesión (*session key*)

- Es una clave **simétrica** generada aleatoriamente para la sesión
- Sirve para cifrar los datos transportados en la sesión
- Es diferente en cada sesión. Se destruye al final.
- No se almacena nunca en disco
- Es compartida por cliente y servidor de forma segura
- (En realidad SSH2 tiene varias claves de sesión, pero las trataremos como una sola)

Desconocida para ambas partes hasta que se establece sesión.

SSH: Claves criptográficas usadas

~~Clave de servidor (*server key*)~~

- Es una clave asimétrica (privada/pública) creada por el servidor
- Sirve para proteger la clave de sesión en SSH-1
- Es diferente en cada sesión. Se re-genera cada hora si la sesión sigue activa
- No se almacena nunca en disco
- Específica de SSH1, en **SSH-2 no se usa**

No confundir con la *clave de anfitrión*

SSH: El problema de las claves públicas

Como vimos:

- El cliente necesita la parte pública de la *host key*
- El servidor necesita la parte pública de la *user key*

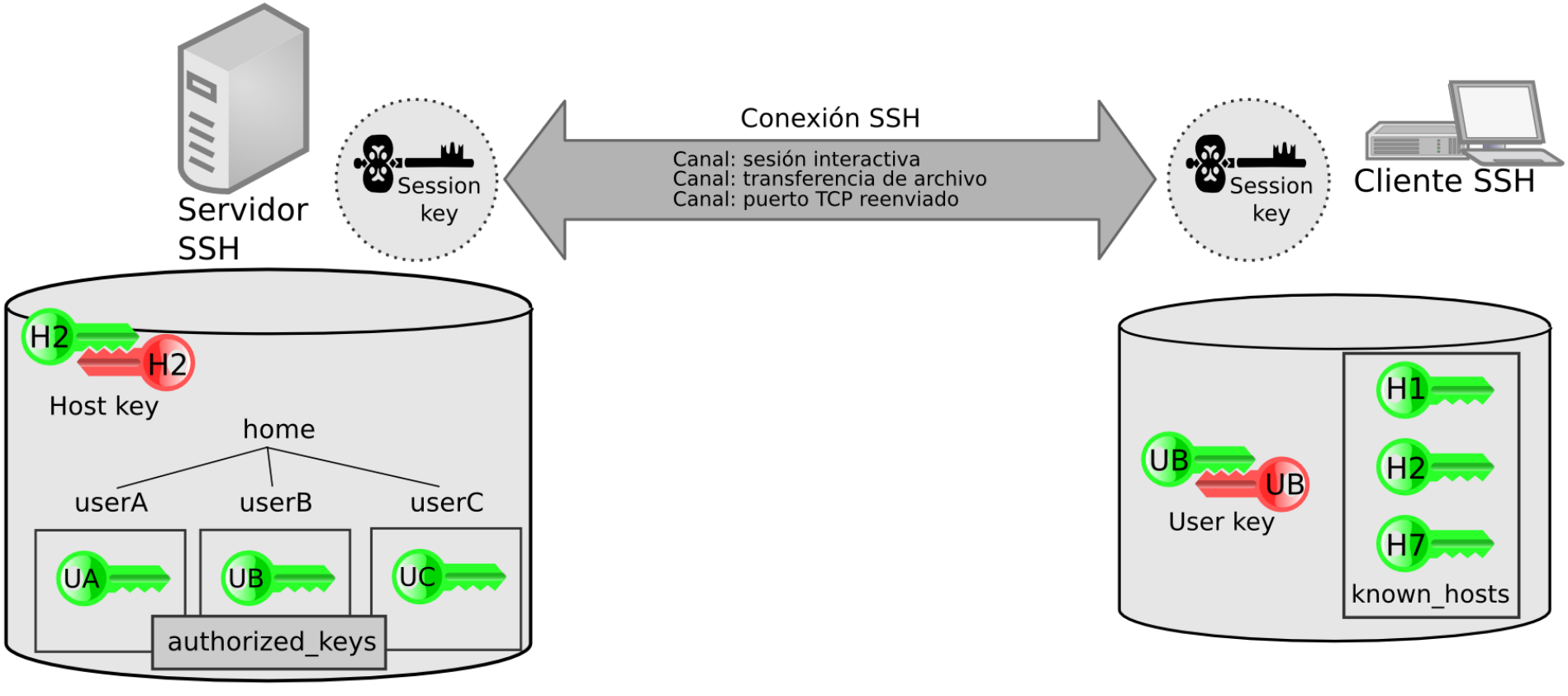
Estas son cadenas ASCII almacenadas en ficheros:

- El cliente las almacena en `~/.ssh/known_hosts`
- El servidor las almacena en `cuenta_de_usuario/.ssh/authorized_keys`

¿Cómo llegan a esos ficheros?

La *user key* se hace **manualmente**. La *host key* suele ser semi-automáticamente en la primera conexión a ese *host*

SSH: Visión general de la arquitectura



Fase 1 (SSH-TRANS): Establecimiento de conexión

- 1) El cliente conecta al puerto 22 del servidor
- 2) Cliente y servidor anuncian qué versión de SSH soportan (cadenas ASCII)
  - Si no son compatibles, desconectan
  - Si son compatibles, el resto de la comunicación es binaria en paquetes de formato definido (longitud-contenido)
- 3) Cada parte envía a la otra una lista de todos los algoritmos que soportan, y se ponen de acuerdo en:
  - Algoritmo de intercambio de claves (ej: *Diffie-Hellman*)
  - Algoritmo de firma por clave pública (ej: RSA)
  - Algoritmo de cifrado simétrico (ej: AES, DES)
  - Algoritmo de *Message Authentication Code* (ej: hmac-sha2)
  - Algoritmo de compresión (ej: zlib)

Fase 1 (SSH-TRANS): Establecimiento de conexión

- 4) Cliente y servidor acuerdan una clave compartida (*session key*). Para ello
  - Utilizan el algoritmo Diffie-Hellman
  - Pero modificado de modo que incluye una firma digital con la *host key*
  - En este paso el servidor envía al cliente su *host key* para que pueda verificar la firma
  - Esto implica que el cliente debe *confiar* en la clave recibida
  - SSH-2 prevé uso de certificados, pero no hay implementaciones que lo hagan
  - Delegan en el usuario la verificación de esa clave
- 5) A partir de este punto todo irá cifrado con la *session key*
  - Y protegida su integridad por hmac

Fase 2 (SSH-USERAUTH): Autenticación del cliente

SSH soporta varios métodos de autenticación. Los dos más importante son:

- Contraseña (*keyboard interactive*)
- Clave pública (*public key*)

Contraseña (*keyboard interactive*).

- El cliente pide al usuario una contraseña y la envía al servidor (el canal ya está cifrado)
- El servidor SSH delega en el operativo servidor la verificación de la clave (ej: `/etc/passwd`)

No requiere ningún *setup* inicial, pero es inseguro que el servidor tenga la contraseña.

Además cada sesión SSH que se inicie, vuelve a pedir la contraseña (no adecuado para trabajo desatendido)

Autenticación del cliente por clave pública

Requisitos previos

- El cliente tiene una pareja de claves privada/pública (el usuario las ha generado con `ssh-keygen`)
- Están almacenadas en el disco duro del cliente (normalmente en la carpeta local `~/.ssh/`)
- El servidor ha de tener una copia de la parte pública (el usuario la "sube" a un fichero del servidor, que suele ser `~/.ssh/authorized_keys` en el servidor)

Autenticación del cliente por clave pública

Protocolo

- **CLIENTE:** envía una copia de su clave pública.
- **SERVIDOR:** la compara con la(s) que tiene para ese usuario. Si no la encuentra → Fallo de autenticación. Si la encuentra:
  - Genera un "reto" (número de 256 bits).
  - Lo cifra con la Kp del cliente y se lo envía.
- **CLIENTE:** recibe el reto cifrado.
  - Lo descifra con su clave privada.
  - Realiza sobre el resultado una serie de operaciones prefijadas (incluyen un identificador de sesión, y un hash) y envía el resultado al servidor.
- **SERVIDOR:** Compara lo que recibe con lo que ha obtenido él mismo. Si coinciden, *la autenticación ha tenido éxito*.

Ventajas

- Muy segura (nunca se transmite ningún secreto)
- Configurable en el lado del servidor (según la clave pública recibida y el *host* de origen se le puede dar o no acceso, o sólo un *shell* restringido, etc)

Desventajas

- Para descifrar el reto el cliente necesita acceso a su clave privada
- Que está protegida por contraseña (por si la roban)
- De modo que el usuario aún necesita meter una contraseña cada vez

¿Soluciones?

SSH: Agentes

- Un agente es un proceso en ejecución que puede mantener una copia (en memoria) de mis claves privadas.
- Al añadir una clave privada al agente, éste la lee de disco (se nos pide la contraseña para descifrarla) y la almacena en memoria RAM.
- El cliente SSH le pide al agente que resuelva el reto (a través de un socket en *localhost*, seguro)
- El agente lo hace usando la clave privada sin pedirnos de nuevo su contraseña.

> **Resultado**: Mientras el agente esté en ejecución, todas las sesiones SSH que iniciemos no nos pedirán contraseña.

Aún así, plantea riesgos de seguridad dejar la terminal desatendida.

Fase 3 (SSH-CONNECT): Creación de canales

La creación del canal puede iniciarse desde cualquiera de los extremos, mediante un protocolo específico.

- Se elige un número (libre) de canal
- Se envía un mensaje especial solicitando la apertura (incluyendo el número y el tipo del canal)
- El otro extremo responde aceptando o denegando su creación

Todos los canales se multiplexan en la misma conexión TCP

- Cada paquete que SSH intercambia contiene el identificador del canal al que pertenece.
- Todos van cifrados y opcionalmente comprimidos
- Cada uno puede servir para una funcionalidad diferente

Utilidades de los canales:

- Sesión interactiva remota. Funcionalidad análoga a `telnet`
- Ejecución de un comando remoto (no interactivo)
- Transferencia de ficheros
- Redirección de puertos (túnel cifrado entre aplicaciones inseguras)

Según cuántos canales se creen, se puede hacer una sola de las anteriores o todas a la vez

Redirección de puertos (túnel SSH)

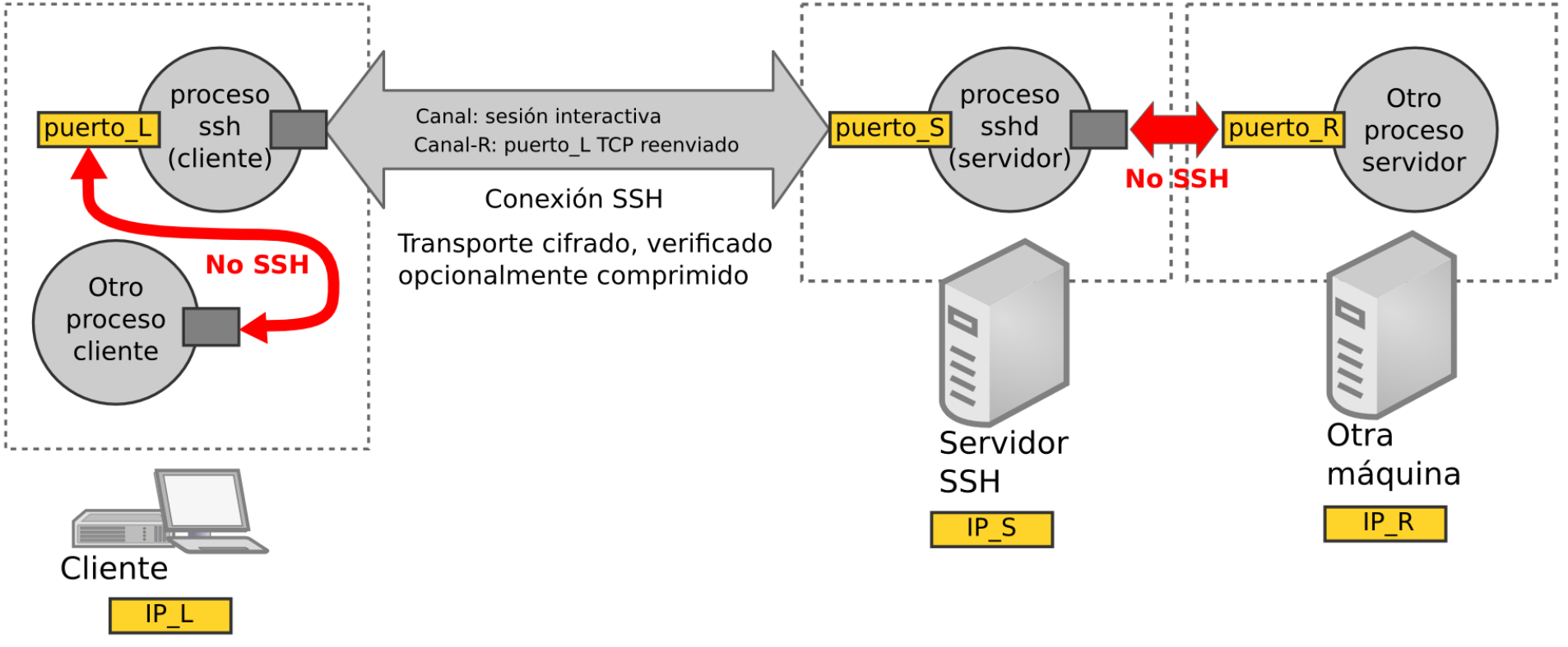
Una conexión SSH puede lanzarse con el comando:

```
$ ssh -L[IP_L:]puerto_L:IP_R:puerto_R [-p puerto_S] user@IP_S
```

Hará lo siguiente:

- El cliente se conectará al `puerto_S` (por defecto 22) de `IP_S`
- Cliente y servidor establecerán un transporte seguro entre `IP_L` y `IP_S`
- El cliente se autenticará para el usuario `user`
- Se creará un canal de reenvío de puertos: Canal-R
- El cliente SSH escuchará en `IP_L` (por defecto `INADDR_ANY`) y `puerto_L`. Cualquier conexión que reciba la enviará por el canal-R
- El servidor SSH reenviará todo lo que le llegue del canal-R hacia `IP_R`, `puerto_R`

Redirección de puertos, figura



3.3 Transferencia de archivos

Transferencia de archivos

Estos servicios permiten:

- Examinar los archivos en una carpeta remota
- Borrar, renombrar dichos archivos
- Transferirlos al ordenador local
- Enviar archivos desde el ordenador local

Dos posibles interfaces:

- Se requiere un cliente específico (línea de comandos o GUI)
- Usa el propio sistema de archivos del cliente

Con cliente específico

- **FTP**: El más antiguo, ya casi en desuso
- **scp**: Copia de archivos sobre ssh1
- **sftp**: Copia de archivos y más opciones sobre ssh2
- **FTPS**: FTP sobre TLS (análogo a https para http)

¡No confundir sftp con FTPS!

Usando sistema de archivos del cliente

- **NFS**: El primer sistema de archivos remoto (Sun)
- **SMB**: "Carpetas compartidas" en Windows (Microsoft)
- **samba**: implementación libre de SMB
- **sshfs**: análogo a NFS pero sobre ssh y en el espacio de usuario

Colaboración, control de versiones, *nube*

- **SVN**: Subversion, sistema de control de versiones centralizado
- **Git**: Sistema de control de versiones distribuido
- **WebDAV**: Almacenamiento, modificación y colaboración en documentos sobre HTTP
- **Otros, comerciales**: iCloud, Dropbox, Google Drive, Microsoft OneDrive (antes SkyDrive), etc.

FTP

FTP: *File Transfer Protocol* (RFC 959)

Este es el más antiguo protocolo para transferencia de archivos:

- Es anterior al HTTP
- Pero los navegadores Web lo soportan también con el esquema `ftp://`
- Permite autenticación del usuario o "modo anónimo" (para archivos públicos)
- Es un protocolo inusual porque requiere dos conexiones TCP en paralelo.

La comunicación va sin cifrar. Por tanto se considera inseguro

FTP: Canales de comunicación

Dos canales de comunicación simultáneos:

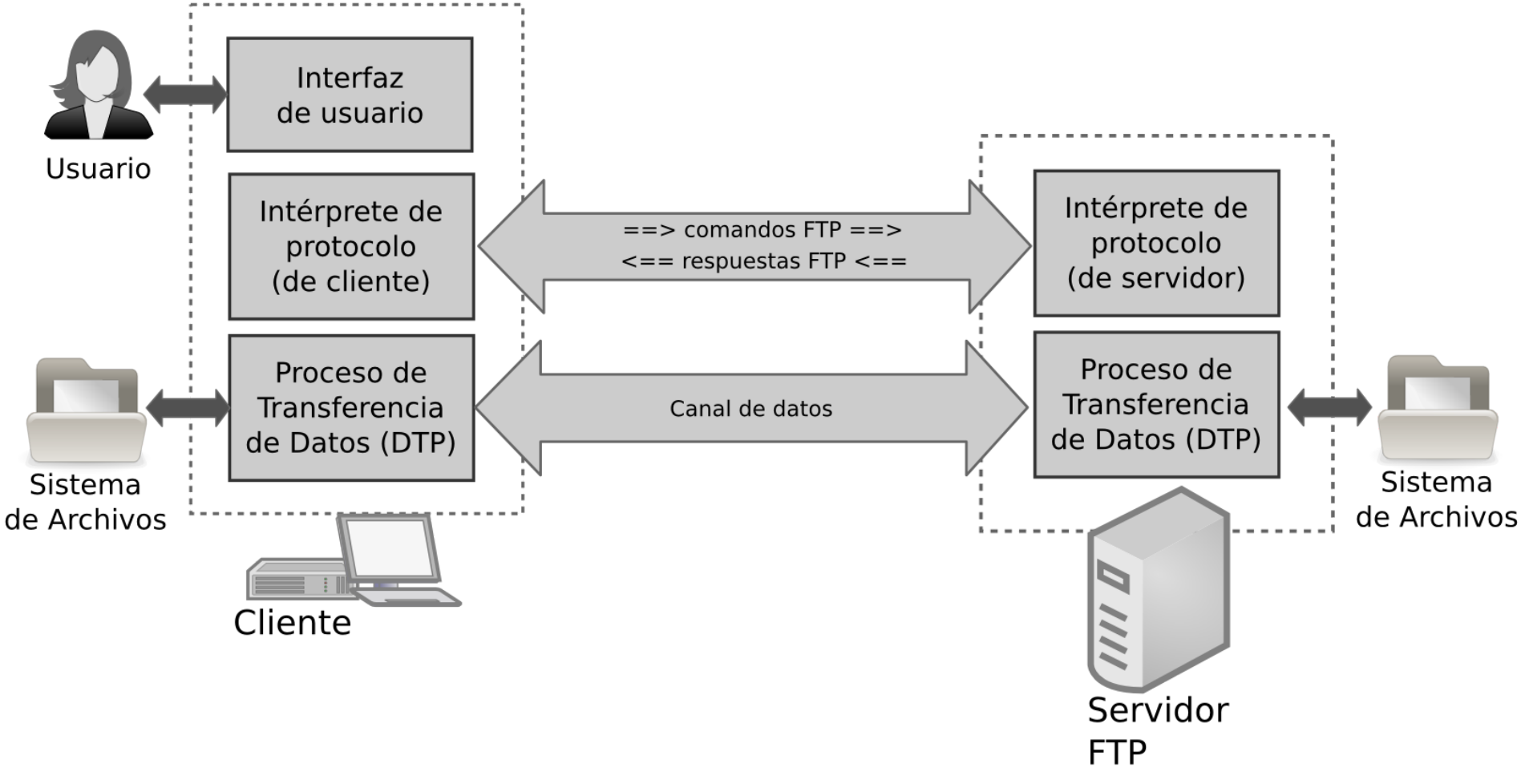
Canal de control

- Lo inicia el cliente conectándose al puerto **21** del servidor
- Transporta comandos que solicita el cliente y las respuestas de estado del servidor.
- *Pero no los archivos* que se quieren transferir
- Permanece activo toda la sesión

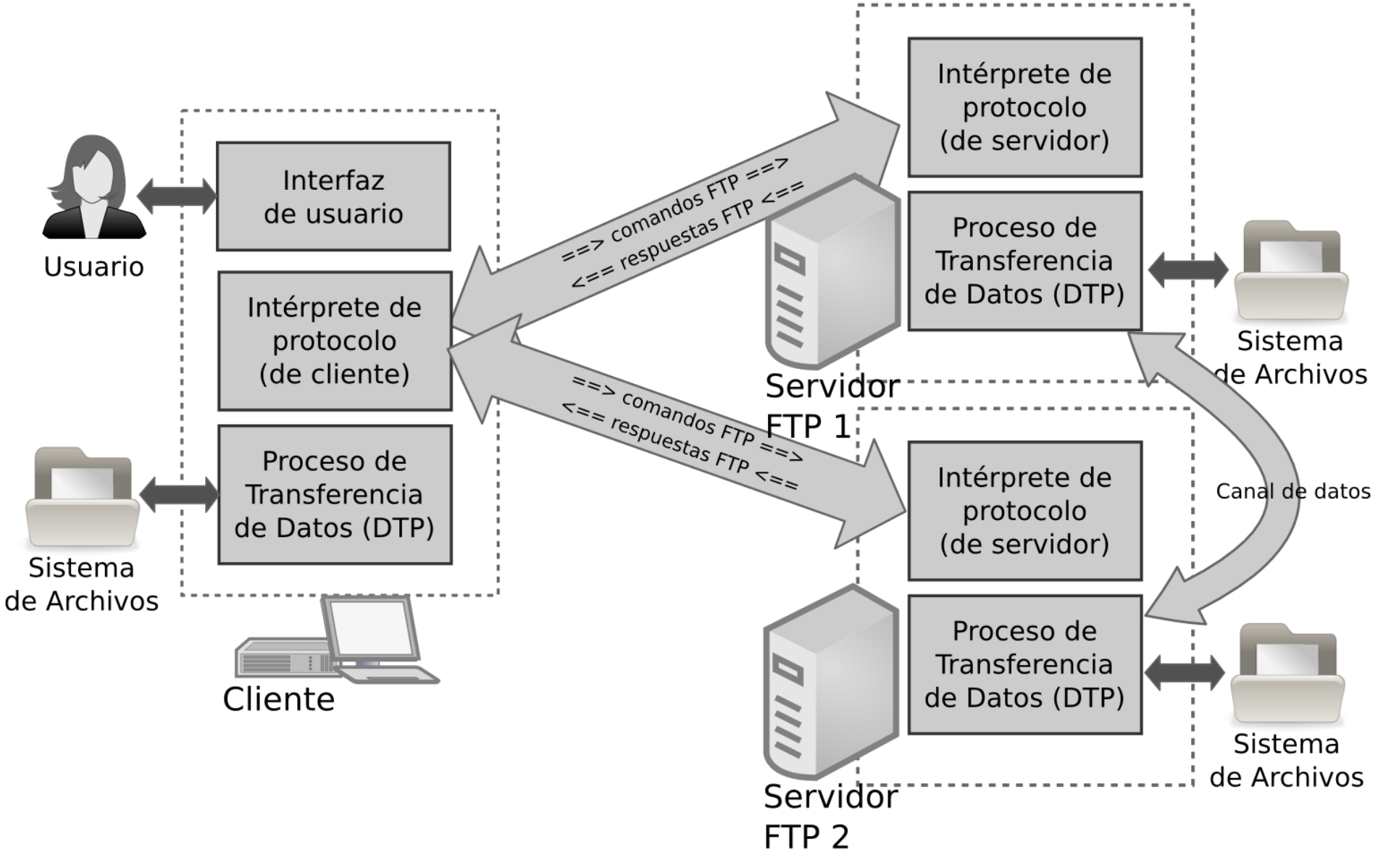
Canal de datos

- Puede iniciarlo el cliente, o el servidor (según el *modo* sea *pasivo* o *activo*)
- Transporta los archivos
- Se establece para cada transferencia
- Puede establecerse incluso entre dos servidores (el cliente sólo los controla)

FTP: Esquema con un solo servidor



FTP: Esquema de transferencia entre dos servidores



FTP: Modo *activo*

- Es el modo por defecto
- El cliente inicia la conexión de control (al puerto 21 del servidor)
- El cliente crea un socket de escucha y comunica al servidor el número de puerto
- Cuando hay que transferir un archivo, el *servidor se conecta al cliente*
- Es **activo** para el servidor.

¿Qué pasa si el cliente está detrás de un NAT?

FTP: modo *pasivo*

- No es el modo por defecto. El usuario debe activarlo con un comando u opción.
- El cliente inicia la conexión de control (al puerto 21 del servidor)
- El cliente solicita al servidor el modo pasivo
- El servidor crea un socket de escucha y comunica al cliente el número de puerto
- Cuando hay que transferir un archivo, el *cliente se conecta al servidor*
- Es **pasivo** para el servidor

Funciona también con el cliente detrás de un NAT o Firewall

FTP: Detalles del protocolo

- El canal de control usa el protocolo `telnet`, pero una versión simplificada
  - Cada comando FTP es una cadena ascii terminada por `\r\n`
  - Cada respuesta FTP es también una cadena ASCII terminada por `\r\n`
  - La respuesta tiene un formato similar a HTTP (código numérico + mensaje explicativo)
  - Algunas respuestas requieren la apertura de una conexión de datos
- Las conexiones de datos pueden ser:
  - ASCII: El carácter `\n` se trata de forma especial
  - Binaria: Ningún carácter se trata de forma especial

Algunos comandos FTP

Mensaje	Descripción
`USER`	Envía el nombre de usuario, para autenticación
`PASS`	Envía la contraseña del usuario, para autenticación
`PORT`	Envía el puerto de escucha del cliente (modo activo)
`PASV`	Solicita modo pasivo
`LIST`	Obtener listado del directorio actual
`CWD`	Cambio de directorio actual
`CDUP`	Subir un directorio
`TYPE`	Establece el modo de la conexión de datos (ascii/binaria)
`RETR`	Solicita descarga de archivo
`STOR`	Solicita subida de archivo

Estados del protocolo FTP

FTP es un protocolo complejo, ya que algunas acciones requieren varios comandos.

> Por ejemplo, la autenticación del usuario son dos comandos: USER y PASS

Esto implica que cliente y servidor deben mantener un *estado* en función del cual sepan cuál es el próximo paso a dar o a esperar.

> Algunos comandos pueden fallar al principio, otros al final, otros "por el medio"

El servidor tiene diferentes códigos de respuesta para cada caso. La máquina de estados evoluciona según esos códigos de respuesta

Respuestas FTP

La respuesta del servidor tiene dos formas:

1. Una sola línea: Código numérico + espacio + texto explicativo

```
502 Command not implemented
```

2. Varias líneas. La primera usa un guión (`-`) en lugar de espacio detrás del código numérico. Las restantes son libres, salvo la última, que repite el código numérico pero esta vez con espacio.

```
214-Help message
Este es un mensaje de ayuda para ser leido por el usuario.
Si alguna linea del mensaje comienza por numero, como
 302 por ejemplo esta debe ser indentada para no confundirla
con otra respuesta del servidor
214 Ultima linea
```

Código numérico

El código numérico se compone de tres dígitos. Cada dígito da información más detallada que el anterior

- **Primer dígito** Indica si el comando ha tenido éxito, fallado, o se halla en un estado intermedio en que aún puede tener éxito o fallar.

  En base a este primer dígito el cliente puede decidir qué acción tomar

- **Segundo dígito** Establece la categoría del mensaje de respuesta. Si es una respuesta meramente informativa, un error de sintaxis, un error relativo a la conexión de datos, etc.

- **Tercer dígito** Detalles más específicos en función de la categoría especificada por el segundo dígito.

Código numérico: primer dígito

Valor	Descripción	Ejemplo
1	Respuesta preliminarmente positiva	`120` Service ready in nnn minutes
2	Respuesta positiva	`200` Command okay
3	Respuesta intermedia positiva	`331` User name okay, need password
4	Respuesta negativa transitoria	`426` Connection closed; transfer aborted
5	Respuesta negativa permanente	`530` Not logged in

Código numérico: segundo dígito

Valor	Significado
0	Errores sintácticos, comandos supérfluos, no implementados, ...
1	Información (ayuda, estado)
2	Respuestas sobre las conexiones (de control o datos)
3	Respuestas sobre autenticación (login, etc.)
4	No usado
5	Respuestas relacionadas con el sistema de ficheros

Interfaz de usuario

El cliente ftp clásico era de línea de comandos. Estos son algunos de los que usa:

Comando	Significado
`open sitio`	Establece conexión con el sitio
`dir`	Solicita listado del directorio remoto
`cd directorio`	Cambia el directorio remoto
`mkdir d`	Crear directorio d remoto
`ascii/binary`	Cambiar tipo de transferencia
`passive`	Cambiar servidor a modo pasivo
`get fichero`	Descarga del fichero
`put fichero`	Subida del fichero
`mget *.txt`	Descarga de varios, según patrón
`mput *.py`	Subida de varios, según patrón
`rename`	Cambia nombre a fichero remoto
`exit`	Finalizar conexión

scp / sftp

scp

El protocolo `scp` permite transferir ficheros entre un cliente y un servidor `ssh`.

```
$ scp fich_local.txt user_remoto@IP_remota:carpeta/remota/remoto.txt
$ scp user_remoto@IP_remota:carpeta/remota/fich_remoto.txt local.txt
```

- Durante la creación del canal se usarán los métodos de autenticación de ssh
- No permite crear directorios remotos (usar `ssh` para ello)
- Ni listar sus contenidos (usar `ssh` para ello)
- Ni "cambiar de directorio" en la máquina remota (aunque se puede especificar la ruta a donde copiarlos)
- La transferencia siempre es "binaria", cifrada y opcionalmente comprimida

sftp

El protocolo `sftp` permite crear una sesión de transferencia de ficheros sobre `ssh`.

- La finalidad y uso es similar a FTP
  - Listar contenidos de carpetas
  - Crear nuevas carpetas
  - Transferir varios ficheros entre cliente y servidor
  - Renombrar, etc.
- Pero es un protocolo completamente nuevo
- Definido como parte de SSH2 (SSH-CONNECT), va sobre SSH-TRANS
- Usa un canal SSH.

NO es FTP sobre SSH