



## Fundamentos de Redes

# Tema 5

# Capa de Aplicación

Antonio M. Mora García



# Bibliografía

## Básica

- ◎ James F. Kurose, Keith W. Ross. Redes de computadoras. Un enfoque descendente. 7º Edición. Editorial Pearson S.A., 2017.

**CAPÍTULO 2 (2.1, 2.2, 2.4, 2.5)**

- ◎ P. García-Teodoro, J.E. Díaz-Verdejo, J.M. López-Soler. Transmisión de datos y redes de computadores, 2ª Edición. Editorial Pearson, 2014. **CAPÍTULO 11**



## Complementaria

- ◎ James F. Kurose, Keith W. Ross. Redes de computadoras. Un enfoque descendente. 7º Edición. Editorial Pearson S.A., 2017.

**CAPÍTULOS 7 y 8**



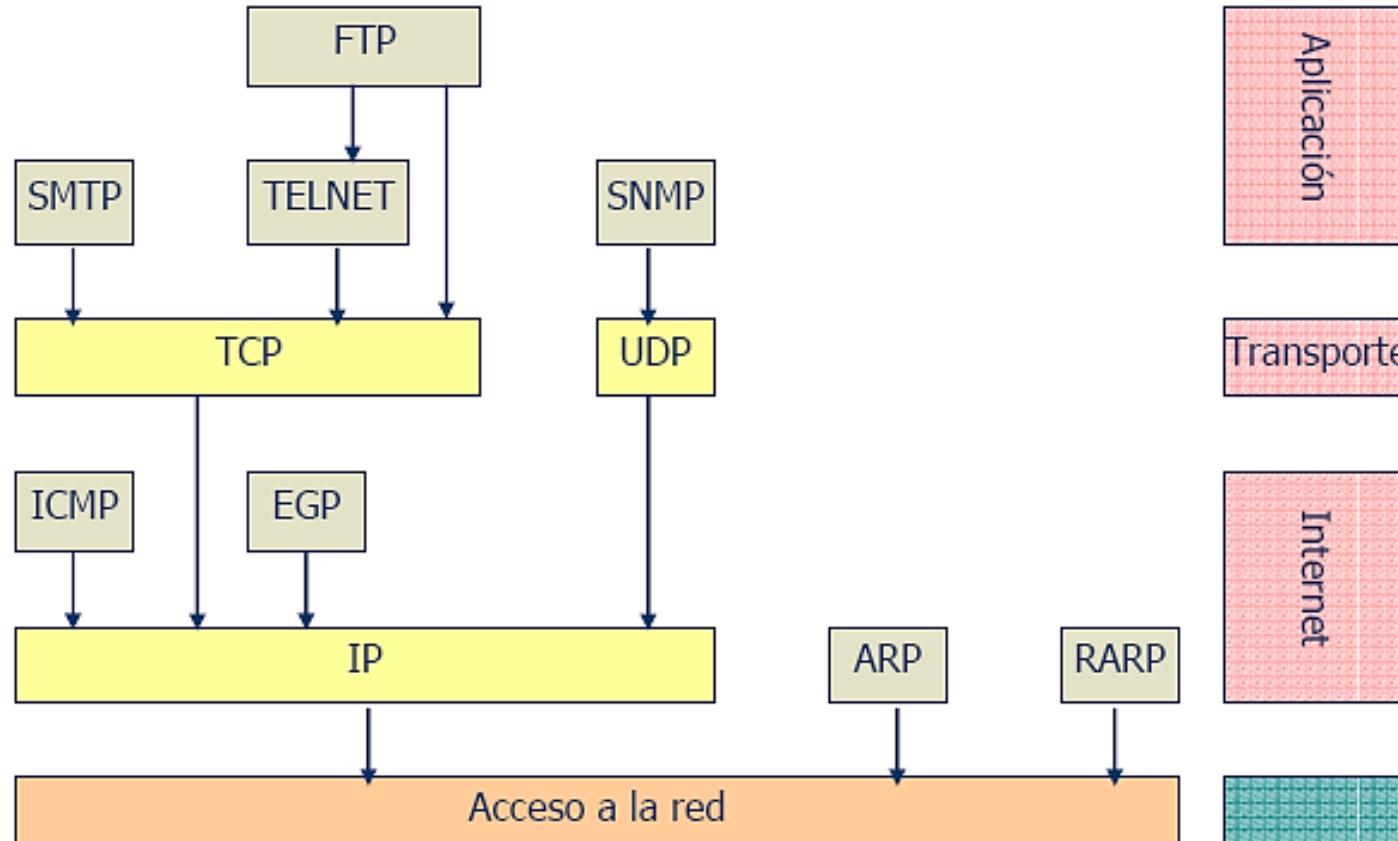
# Índice

- ◎ **5.1.** Introducción a las aplicaciones de red
- ◎ **5.2.** Servicio de Nombres de Dominio (DNS)
- ◎ **5.3.** Navegación web
- ◎ **5.4.** Correo electrónico
- ◎ **5.5.** Aplicaciones multimedia
- ◎ **5.6.** Cuestiones y ejercicios

# TEMA 5. Capa de Aplicación

- 5.1. Introducción a las aplicaciones de red**
- 5.2. Servicio de Nombres de Dominio (DNS)**
- 5.3. Navegación web**
- 5.4. Correo electrónico**
- 5.5. Aplicaciones multimedia**
- 5.6. Cuestiones y ejercicios**

# Estructura de protocolos



# Protocolos TCP/IP

- El **origen de esta familia** de protocolos fue la red ARPANET (en ella se desarrollaron los conceptos fundamentales de diseño y gestión de redes), para la que se definió el precursor de TCP/IP: **NCP (*Network Control Program*)**.
- Los **niveles** más bajos (**enlace y físico**) **no están implementados** ya que TCP/IP se diseñó para **no depender de una red física** concreta:
  - Los protocolos ARP (*Address Resolution Protocol*) y RARP (*Reverse Address Resolution Protocol*) se encargan de enlazar los sistemas de direccionamiento IP y el de la red física utilizada.

# Protocolos TCP/IP

## CAPA DE RED

- La **base** de la familia de protocolos es el **nivel de Red (IP, Internet Protocol)**.
- Es un **protocolo de conmutación de paquetes** muy sencillo, de tipo **datagrama**, de forma que se pueda implementar en cualquier tipo de máquina.
- Existen actualmente dos versiones **IPv4, IPv6**.
- Protocolos de apoyo:
  - **ICMP** (*Internet Control Message Protocol*): comunicación de mensajes entre nodos de la red
  - **IGMP** (*Internet Group Management Protocol*): envío de mensajes a grupos de usuarios

# Protocolos TCP/IP

## CAPA DE TRANSPORTE

- Implementa dos protocolos extremo a extremo (entre nodo origen y nodo destino).
- **TCP (Transmission Control Protocol):**
  - Es un protocolo orientado a la conexión.
  - Con control de errores.
  - Se encarga también del control de flujo.
  - Fragmentado y reensamblado de segmentos (garantiza el secuenciamiento).
- **UDP (User Datagram Protocol):**
  - Es un protocolo no orientado a conexión (datagrama).
  - No realiza control de errores.
  - No garantiza el secuenciamiento de la información.
  - Es muy rápido.

Útil para peticiones  
aisladas, o  
transmisión de  
audio o vídeo

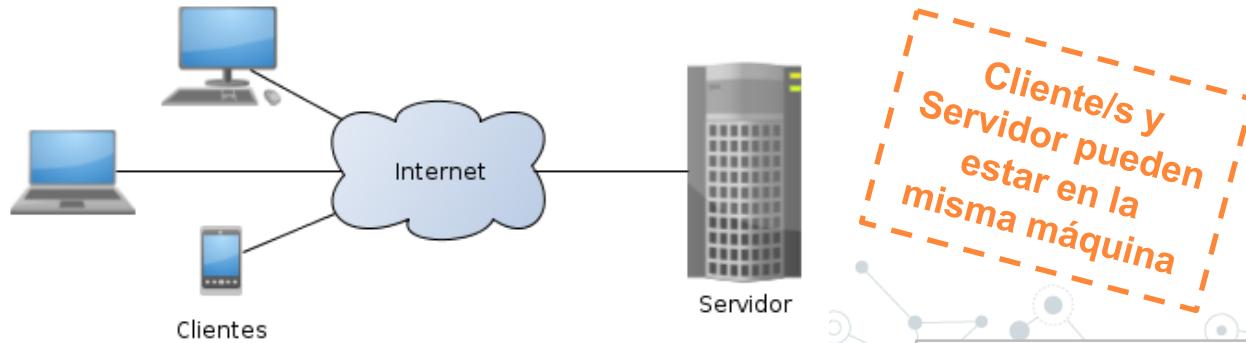
# Protocolos TCP/IP

## CAPA DE APLICACIÓN

- Protocolos basados en **ICMP**:
  - **PING**: solicitud de eco (comprobación de conectividad)
- Protocolos basados en **TCP**:
  - **TELNET**: terminal remoto
  - **FTP**(*File Transfer Protocol*): transmisión de ficheros
  - **SMTP**(*Simple Mail Transfer Protocol*): correo electrónico
  - **HTTP**(*HyperText Transfer Protocol*): páginas web
  - **RPC** (*Remote Procedure Call*): ejecución de procesos remotos
- Protocolos basados en **UDP**:
  - **SNMP**(*Simple Network Management Protocol*): gestión de red
  - **BOOTP**: arranque remoto
  - **DNS**(*Domain Name System*)
  - **NFS** (*Network File System*): gestión de ficheros en red

# Arquitectura Cliente/Servidor

- La **arquitectura** (o modelo) **cliente/servidor** es una **forma específica de diseño de aplicaciones**, aunque también se conoce con este nombre a las computadoras en las que estas aplicaciones son ejecutadas.
- El **cliente** es la computadora que se encarga de efectuar una **petición o solicitar un servicio** y recibir una respuesta. El cliente no posee control sobre los recursos.
- El **servidor** es una computadora (remota normalmente) que **evalúa la petición del cliente** y la acepta o la rechaza. Si es aceptada **ejecuta el servicio y transmite la información resultante** al cliente que efectuó la petición.



# Arquitectura Cliente/Servidor

## VENTAJAS

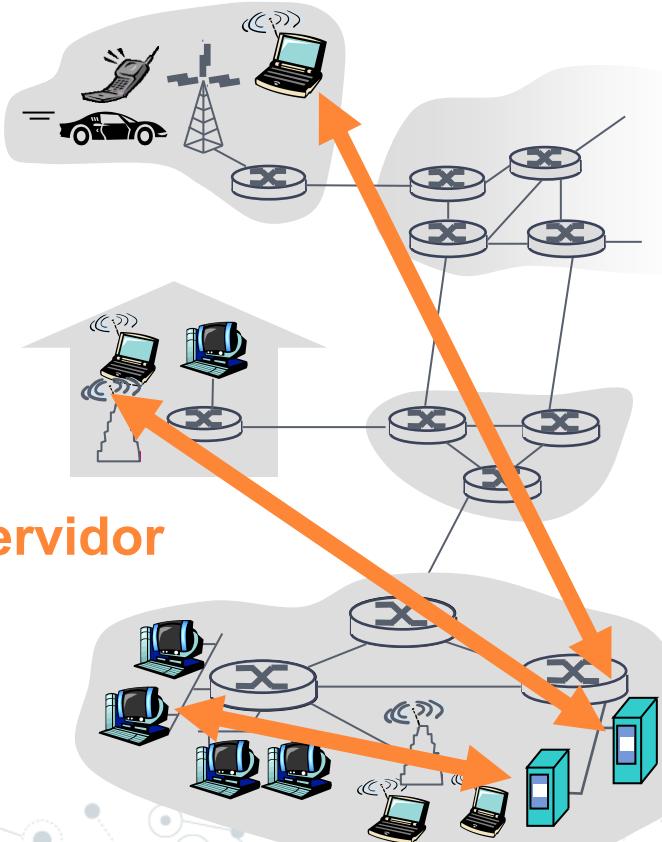
- **Recursos centralizados:** debido a que el servidor suele ser el centro de la red, puede administrar los recursos que son comunes a todos los usuarios (clientes).
- **Seguridad mejorada:** la cantidad de puntos de entrada que permite el acceso a los datos no es importante. El servidor garantizará la seguridad en dicho acceso.
- **Administración al nivel del servidor:** ya que los clientes no juegan un papel importante en este modelo, requieren menos administración.
- **Red escalable:** es posible quitar o agregar clientes (hasta un límite) sin que afecte demasiado al funcionamiento de la red y sin la necesidad de realizar mayores modificaciones.

# Arquitectura Cliente/Servidor

## DESVENTAJAS

- **Costo elevado:** debido a la complejidad técnica del servidor y a su gestión, seguridad y mantenimiento.
- **Servidor es el eslabón débil:** debido a que toda la red del servicio está construida en torno a él. Afortunadamente, el servidor suele ser altamente tolerante a los fallos (replicación, discos espejo, copia de seguridad, virtualización).

# Arquitectura Cliente/Servidor



## Servidor

- Siempre en funcionamiento
- IP permanente y pública
- Agrupados en “granjas”
- Pueden comunicarse entre sí para optimizar el servicio

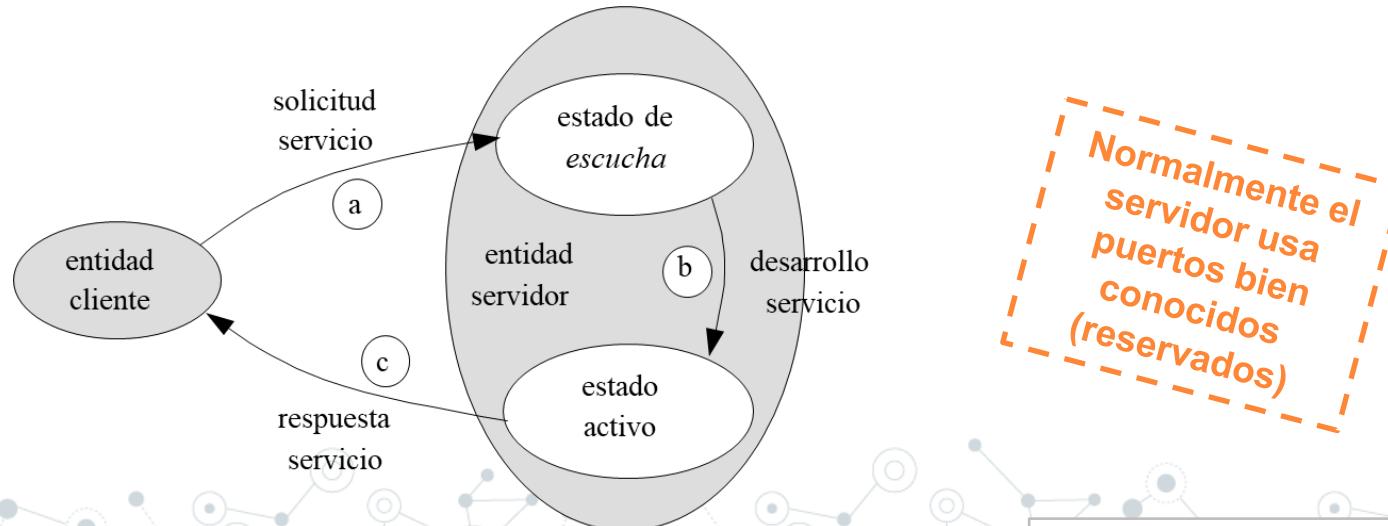
## Clients

- Funcionando intermitentemente
- Pueden tener IP dinámica y privada
- Se comunican con el servidor
- No se comunican entre sí en relación a un servicio

# Arquitectura Cliente/Servidor

La mayoría de las transacciones entre aplicaciones se basan en el paradigma cliente-servidor:

- **Servidor:** programa que ofrece un servicio accesible a través de la red.
- **Cliente:** programa que envía peticiones y espera respuestas del servidor a través de la red.



# Arquitectura Cliente/Servidor

## Tipos de servidores:

- **Servidor Iterativo:**

- Sólo puede atender a un cliente a la vez.
- Se encolarán los clientes que realicen peticiones mientras el servidor está dando servicio a otro cliente.

- **Servidor Concurrente:**

- Puede dar servicio a varios clientes a la vez.
- Ejecuta un proceso (o hebra) por cada cliente, para procesar concurrentemente las peticiones (conurrencia real o simulada)

# Procesos cliente y servidor

**Proceso Cliente:** proceso que inicia la comunicación

**Proceso Servidor:** proceso que espera ser contactado

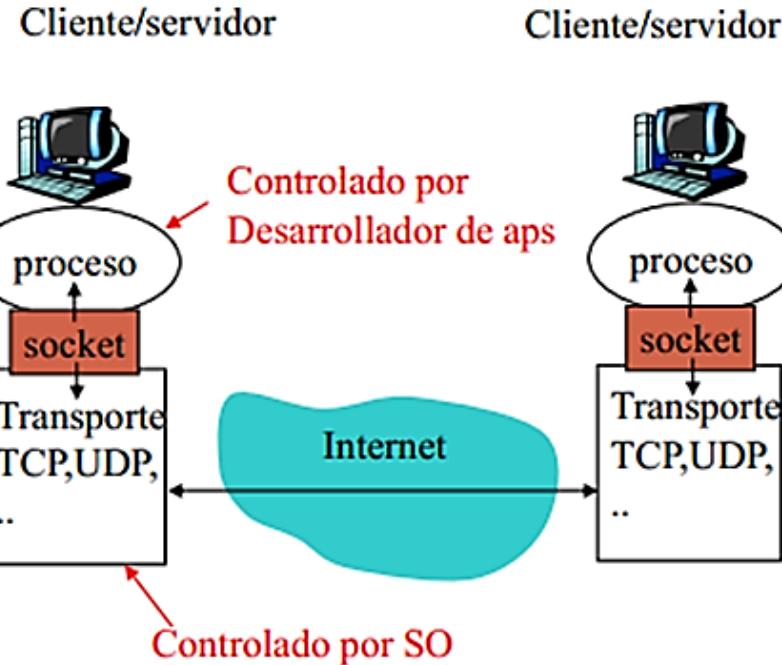
- Un proceso envía/recibe mensajes a/desde un **socket**.
- Cada proceso debe tener un **identificador** compuesto por su **dirección IP + número de puerto**

Ejemplo:

servidor web *gaia.cs.umass.edu*

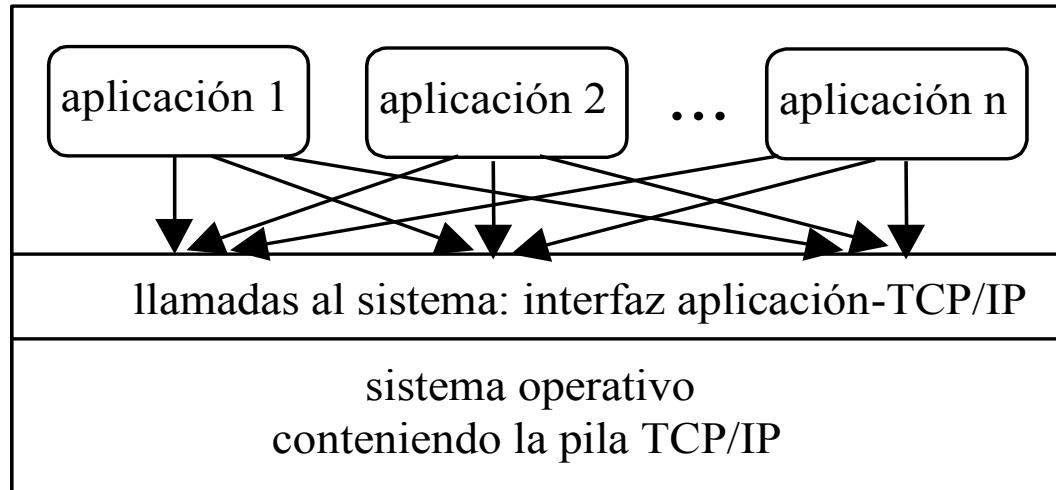
Dirección IP: 128.119.245.12

Núm. Puerto: 80



# Interfaz Socket

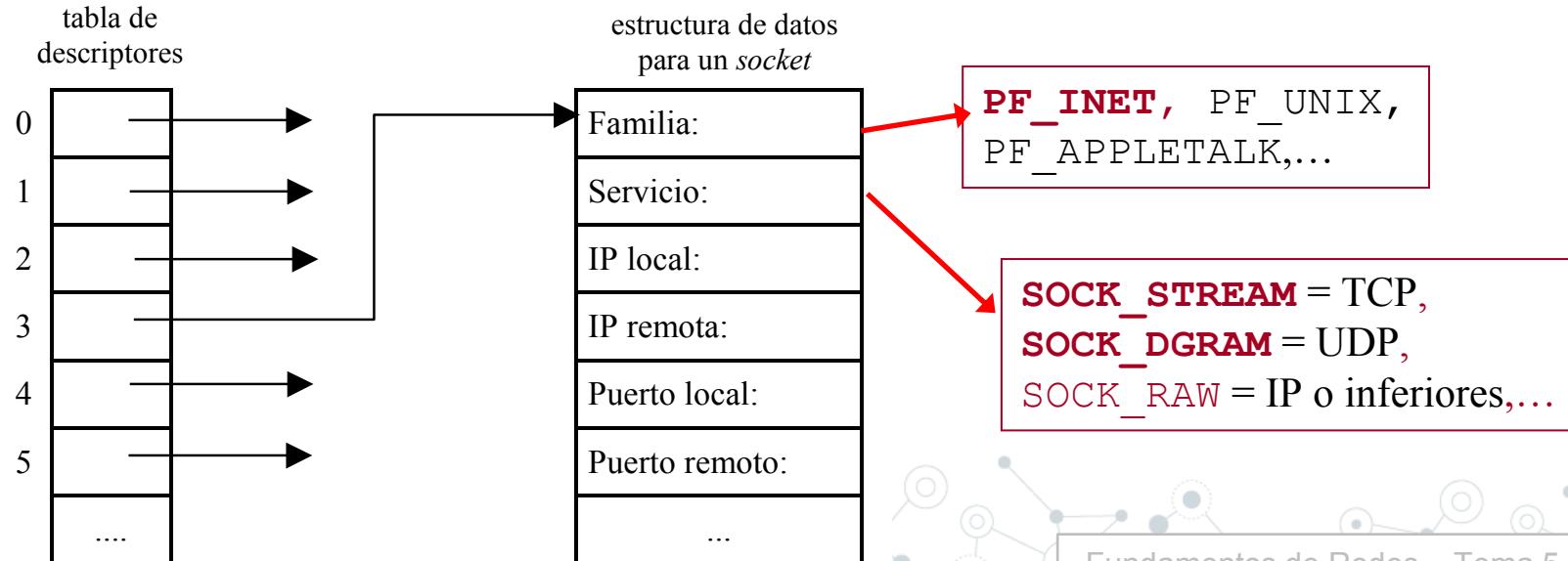
- El software TCP/IP es parte del S.O.
- Las aplicaciones lo usan a través de una **API** consistente en **llamadas al sistema**.  
Es la llamada “**Interfaz socket**”.



- La definición de la **interfaz socket** no es parte de ningún protocolo.
- Distintas implementaciones:
  - Berkeley Socket Distribution,
  - Winsock,
  - Transport Layer Interface, etc.

# Interfaz Socket

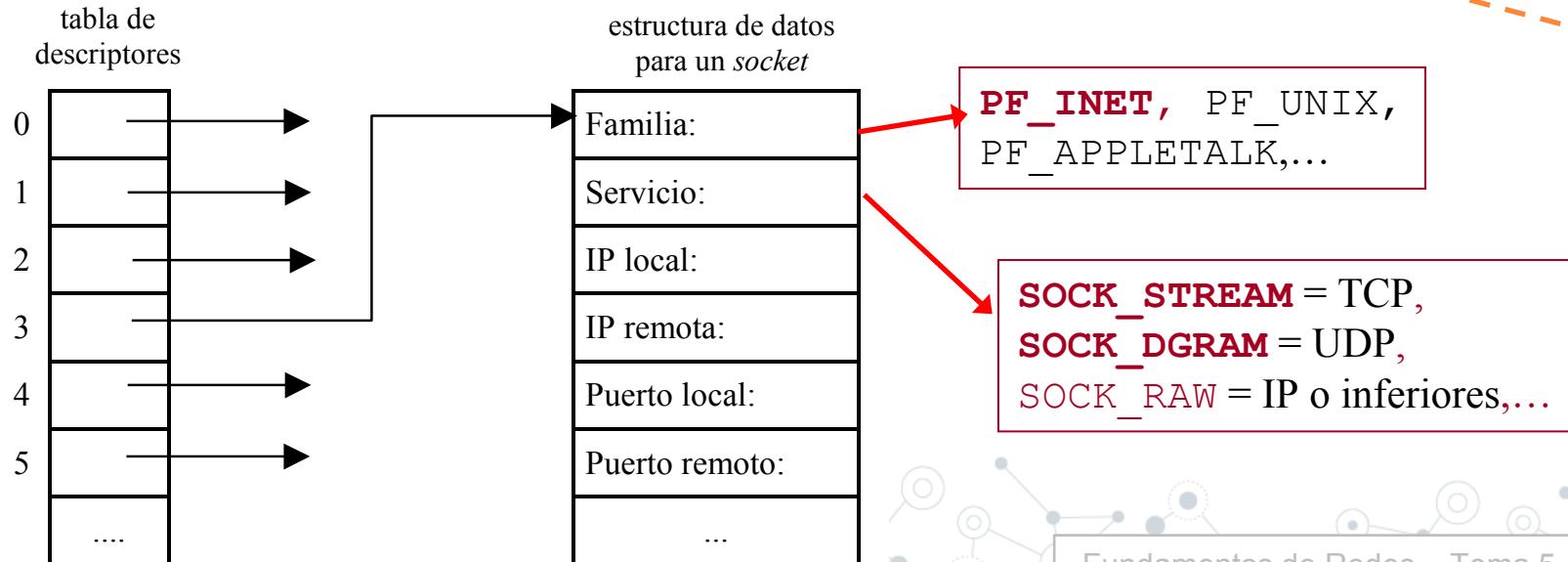
- Definimos **socket** como un **descriptor** de una transmisión **a través del cual** la aplicación puede **enviar y/o recibir información hacia y/o desde otro proceso** de aplicación.
- Es una “**puerta**” de acceso entre la **aplicación** y los servicios de **transporte**.
- En la práctica un **socket** es un **puntero** a una estructura del tipo:



# Interfaz Socket

- La gestión de I/O del S.O. se basa en:  
*abrir socket – leer/escribir – cerrar socket*
- Implementado mediante llamadas al sistema:  
*open – read/write – close*

*Es necesario añadir identificación de los puntos finales:  
IP local, IP remota  
Puerto local, Puerto remoto*



# Interfaz Socket

## PROPIEDADES (Según el tipo de socket)

- 1) La **fiabilidad de la transmisión**: ningún dato transmitido se pierde.
- 2) La conservación del **orden de los datos**: los datos llegan en el orden en el que han sido emitidos.
- 3) La **no duplicación de datos**: sólo llega a destino un ejemplar de cada dato emitido.
- 4) La comunicación en **modo conectado**: se establece una conexión entre dos puntos antes del principio de la comunicación (es decir, se establece un circuito virtual). A partir de entonces, una emisión desde un extremo está implícitamente destinada al otro extremo conectado.
- 5) **Delimitación de los mensajes**: los límites de los mensajes emitidos se pueden encontrar en el destino.
- 6) El envío de **mensajes (urgentes)**: posibilidad de enviar datos fuera del flujo normal, accesibles inmediatamente.

# Interfaz Socket

## TIPOS DE SOCKETS

- **SOCK\_STREAM:** Los sockets de este tipo permiten **comunicaciones fiables en modo conectado** (propiedades 1, 2, 3 y 4) y eventualmente autorizan, según el protocolo aplicado los mensajes fuera de flujo (propiedad 6). **[SOCKET TCP]**
- **SOCK\_DGRAM:** Corresponde a los sockets destinados a la comunicación en modo **no conectado** para el envío de datagramas de tamaño limitado. Los datagramas no trabajan con conexiones permanentes (protocolo UDP). La transmisión por los datagramas se hace a nivel de paquetes, donde cada paquete puede seguir una ruta distinta, **no garantizándose una recepción secuencial** de la información. **[SOCKET UDP]**
- **SOCK\_RAW:** Permite el acceso a los protocolos de más bajo nivel (por ejemplo, el protocolo IP en el dominio Internet). Su uso está reservado al superusuario.
- **SOCK\_SEQPACKET:** Corresponde a comunicaciones que poseen las propiedades 1, 2, 3, 4 y 5.

# Sockets en Java

## Socket TCP - Servidor

- Son orientados a conexión (deben conectarse cliente y servidor antes de hacer la transmisión de datos).
- Al **crear el socket** se indicará el puerto del servicio (en el que ‘escuchará’ peticiones):

```
ServerSocket socketServidor;  
int puerto=888;  
try {  
    socketServidor = new ServerSocket(puerto);  
} catch (IOException e) {  
    System.out.println("Error: no se pudo atender en el puerto "+puerto);  
}
```

# Sockets en Java

## Socket TCP - Servidor

- **Esperar una conexión** (bloqueante) y recibir un socket (tubería) correspondiente a una conexión entre el cliente y el servidor:

```
Socket socketConexion = null;
try {
    socketConexion = socketServidor.accept();
} catch (IOException e) {
    System.out.println("Error: no se pudo aceptar la conexión solicitada");
}
```

- El servidor podrá **atender (aceptar) más conexiones** con el método accept.

# Sockets en Java

## Socket TCP - Cliente

- Son orientados a conexión (deben conectarse cliente y servidor antes de hacer la transmisión de datos).
- Al **crear el socket** se indicará la IP del servidor y el puerto del servicio:

```
Socket socket;
int puerto=8888;
String anfitrion="ei150142.ugr.es"

try {
    socket=new Socket (anfitrion,puerto);
} catch (UnknownHostException e) {
    System.err.println("Error: equipo desconocido");
} catch (IOException e) {
    System.err.println("Error: no se pudo establecer la conexión");
}
```

# Sockets en Java

## Socket TCP - Cliente

- Obtener **flujos de lectura y escritura** del socket (como flujos de bits):

```
InputStream inputStream = socketServicio.getInputStream();
OutputStream outputStream = socketServicio.getOutputStream();
```

- Para **enviar los datos** se usa el método `write`:

```
byte [] buferEnvio="Hola caracola".getBytes();
outputStream.write(buferEnvio,0,buferEnvio.length);
```

- Para **recibir datos** se usa el método `read`:

```
// Hay que reservar memoria para almacenar lo leido
byte []buferRecepcion=new byte[256];
// Intenta leer tantos bytes como posiciones tiene el
// array de bytes, aunque es posible que no haya
// tantos datos, y sólo se lean bytesLeidos:
int bytesLeidos = inputStream.read(buferRecepcion);
```

Los datos se leerán sobre un array de bytes

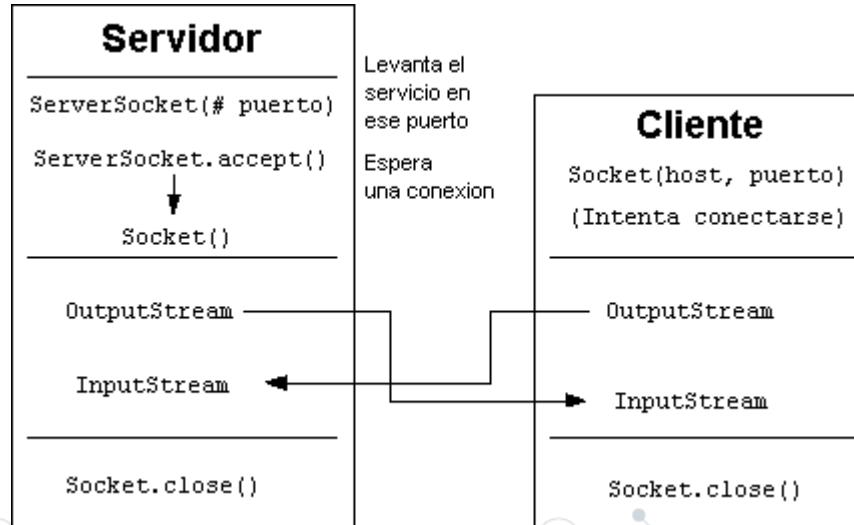
# Sockets en Java

## Socket TCP cliente

- Finalmente se **cerrará el socket** de forma segura (se espera si hay bytes en tránsito):

```
socketServicio.close();
```

- Funcionamiento:



## CLIENTE TCP – JAVA

```
/*
 * <p>Title: Mínimo cliente TCP</p>
 * <p>Description:</p>
 * <p>Copyright: Copyright (c) 2007</p>
 * <p>Company: UGR</p>
 * @author not attributable
 * @version 1.0
 */

import java.net.*;
import java.io.*;

public class MinimoClienteTCP {

    // Atributos de la clase:
    static Socket socket_datos;
    static String direccionServidor; // Nombre o dirección IP
    static int puerto;
    static PrintWriter out;
    static BufferedReader in;

    public MinimoClienteTCP() {
    }

    public static void main (String args[]) {
        boolean error=false;
        String mensajeSolicitud;
        String mensajeRespuesta = "";
        
```

```
// Se piden 3 argumentos: dirección del servidor, puerto y mensaje a enviar.
if (args.length<3) {
    System.err.println("Sintaxis: MinimoClienteTCP <direccion-servidor>
                      <puerto> <mensaje a enviar>");
    System.exit(-1);
}

// Dirección (IP o nombre) del servidor
direccionServidor = args[0];
// Puerto
puerto = Integer.parseInt(args[1]);
// Mensaje a enviar
mensajeSolicitud = args[2];

// 1 - Se abre el socket y se conecta a la dirección y puerto del servidor.
try {
    socket_datos = new Socket (direccionServidor, puerto);

    // Se obtienen los flujos de lectura y escritura para recibir y enviar
    // mensajes.
    out = new PrintWriter (socket_datos.getOutputStream(), true);
    in = new BufferedReader (new
                           InputStreamReader(socket_datos.getInputStream()));
} catch (UnknownHostException e) {
    System.err.println ("Error: no se pudo encontrar al servidor " +
                       direccionServidor);
    System.exit(-2);
} catch (IOException e) {
    System.err.println("Error: no se pudo establecer la conexión con el
                      servidor");
}

// (Continua en la siguiente transparencia ...)
```

## CLIENTE TCP – JAVA

```
// (... continuación de MinimoClienteTCP)

// 2 - Se escribe el mensaje de solicitud y leemos la respuesta:
out.println(mensajeSolicitud);
try {
    mensajeRespuesta = in.readLine();
} catch (IOException e) {
    System.err.println ("Error: no se pudo leer la respuesta.");
}

// 3 - Se cierra la conexión.
try {
    in.close();
    out.close();
    socket_datos.close();
} catch (IOException e) {
    System.err.println ("Error: no se pudo cerrar la conexión.");
}

// Se muestra la respuesta:
System.out.println("El mensaje enviado fue: " + mensajeSolicitud);
System.out.println("El mensaje recibido fue: " + mensajeRespuesta);

}
```

# SERVIDOR ITERATIVO TCP – JAVA

```
/*
 * <p>Title: Mínimo servidor iterativo TCP</p>
 */

import java.net.*;
import java.io.*;

public class MinimoServidorTCP {

    // Atributos de la clase:
    static ServerSocket socket_control;
    static Socket socket_datos;
    static int puerto;
    static PrintWriter out;
    static BufferedReader in;

    public MinimoServidorTCP() {
    }

    public static void main (String args[]){
        boolean salir = false;
        boolean error = false;
        String mensajeSolicitud;
        String mensajeRespuesta;

        // El argumento es el puerto donde se
        // atenderá el servicio.
        if (args.length<1) {
            System.err.println("Sintaxis:
                MinimoServidorTCP <puerto>");
            System.exit(-1);
        }

        // Se obtiene el puerto:
        puerto = Integer.parseInt(args[0]);
    }
}
```

```
// 1 - Se abre el socket en modo "escucha"
try {
    socket_control = new ServerSocket (puerto);
} catch (IOException e) {
    System.err.println("Error: no se puede abrir el puerto
        indicado.");
    System.exit(-2);
}

// Es un servidor iterativo: acepta una conexión, la
// procesa y la cierra. Después se acepta otra conexión
// y así sucesivamente.

do {

    // 2 - Se bloquea la hebra actual en "accept", y se
    //     devuelve la conexión establecida con el cliente.

    try {
        socket_datos = socket_control.accept();

        // Se obtienen los flujos de entrada y salida para
        // recibir y enviar mensajes.
        try {
            out = new PrintWriter
                (socket_datos.getOutputStream(), true);
            in = new BufferedReader (new InputStreamReader
                (socket_datos.getInputStream()));
        } catch (IOException e) {
            System.err.println("Error: no se pudo obtener un
                canal para los flujos");
            error = true;
        }
    }
}
```

```
// 3 - Código del servicio ofrecido.

// 3a) Se lee una línea del cliente
mensajeSolicitud = in.readLine();

// 3b) Se aplica el servicio:
mensajeRespuesta =
    procesaServicio(mensajeSolicitud);

// 3c) Se envía la respuesta:
out.println(mensajeRespuesta);

} catch (IOException e) {
    System.err.println("Error: no se pudo aceptar la
        solicitud de una conexión");
    error = true;
}

} while (!salir);

}

static String procesaServicio (String mensaje) {

    // Aquí se podría poner el código que procesara el
    // mensaje recibido. Actualmente sólo lo devuelve tal
    // cual vino.

    return mensaje;
}
}
```

# SERVIDOR CONCURRENTE TCP – JAVA

## 5.1. Introducción a las aplicaciones de red

```
/**  
 * <p>Title: Mínimo servidor concurrente TCP</p>  
 * <p>Description: </p>  
 * <p>Copyright: Copyright (c) 2007</p>  
 * <p>Company: UGR</p>  
 * @author not attributable  
 * @version 1.0  
 */
```

```
import java.net.*;  
import java.io.*;  
  
public class MinimoServidorConcurrenteTCP {  
    // Atributos de la clase  
    static ServerSocket socket_control;  
    static Socket socket_datos;  
    static int puerto;  
    static Servicio servicio;  
    // Hebras para que el servicio sea concurrente
```

```
    public MinimoServidorConcurrenteTCP() {  
    }
```

```
    static public void main (String args[]) {  
        boolean salir = false;  
        boolean error = false;
```

```
        // Al menos un argumento, el puerto.  
        if (args.length<1) {  
            System.err.println ("Sintaxis:  
                MinimoServidorConcurrenteTCP <punto>");  
            System.exit(-1);  
        }
```

```
        // Se obtiene el puerto.  
        puerto = Integer.parseInt(args[0]);
```

```
        // 1 - Se abre el socket en modo "escucha".  
        try {  
            socket_control = new ServerSocket (puerto);  
        } catch (IOException e) {  
            System.err.println ("Error: no se puede abrir el  
                puerto indicado.");  
            System.exit(-2);  
        }
```

```
        // Bucle para aceptar conexiones. Por cada conexión  
        // aceptada se creará una hebra a la que se le pasará el  
        // socket para cursar el servicio.  
        do {
```

```
            // 2 - Se bloquea la hebra actual en "accept"  
            // esperando una solicitud de conexión. Se devuelve  
            // un nuevo socket con la conexión establecida.  
            try {  
                socket_datos = socket_control.accept();  
  
                // Se lanza una hebra para que sirva a este cliente  
                // por "socket_datos".  
                new Servicio(socket_datos).start();  
            } catch (IOException e) {  
                System.err.println("Error: no se pudo aceptar la  
                    solicitud de una conexión.");  
                error = true;  
            }  
  
            } while (!salir);  
        }
```

```
        class Servicio extends Thread {  
            // Atributos de la clase  
            Socket socket_datos;  
            PrintWriter out;  
            BufferedReader in;
```

```
            // El constructor recibirá como argumentos el socket  
            // (abierto) que debe utilizar (se lo pasa la hebra  
            // principal del servidor).  
            public Servicio (Socket socket_datos_) {  
                socket_datos= socket_datos_;
```

```
                // Se obtienen los flujos de lectura y de escritura para  
                // enviar y recibir mensajes.  
                try {  
                    out = new PrintWriter  
                        (socket_datos.getOutputStream(), true);  
                    in = new BufferedReader (new InputStreamReader  
                        (socket_datos.getInputStream()));
```

```
                } catch (IOException e) {  
                    System.err.println(this.getName() + " Error: no  
                        se pudo obtener un canal para los flujos.");  
                }  
            }
```

```
            public void run() {  
                String mensajeSolicitud = "";  
                String mensajeRespuesta = "";
```

```
                try {  
                    // 3 - Código del servicio ofrecido.  
                    // 3a - Se lee el mensaje del cliente.  
                    mensajeSolicitud = in.readLine();  
                } catch (IOException e) {  
                    System.err.println(this.getName() + " Error: no  
                        se pudo leer el mensaje.");  
                }  
            }
```

```
            // 3b - Se aplica el servicio.  
            mensajeRespuesta = procesaServicio  
                (mensajeSolicitud);
```

```
            // 3c - Se envía la respuesta.  
            out.println(mensajeRespuesta);
```

```
            // 4 - Se cierra la conexión establecida con  
            // el cliente.  
            try {  
                in.close();  
                out.close();  
                socket_datos.close();  
            } catch (IOException e) {  
                System.err.println(this.getName() + " Error: no  
                    se pudo cerrar la conexión.");  
            }  
        }
```

```
        static String procesaServicio (String mensaje) {  
            // Aquí se podría poner el código que procesara  
            // el mensaje recibido.  
            // Actualmente sólo lo devuelve tal cual vino.  
            return mensaje;  
        }
```

# Protocolos

## ¿QUÉ DEFINE UN PROTOCOLO?

- **El tipo de servicio**

- Orientado o no orientado a conexión
- Confirmado o no

- **El tipo de mensaje**

- Request (solicitud), Response (respuesta), etc

- **La sintaxis**

- Definición y estructura de “campos” en el mensaje
- Hay protocolos orientados a texto (HTTP)
- Y otros en binario (DNS)
- Tendencia para otras capas: usar formato Type-Length-Value

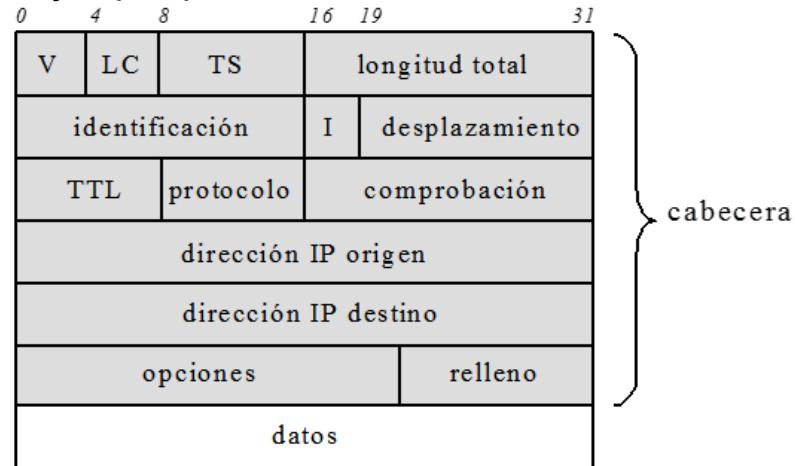
- **La semántica**

- Significado de los “campos”

- **Las reglas**

- Cuándo los procesos envían mensajes/responden a mensajes

### Ejemplo: protocolo IP



# Protocolos

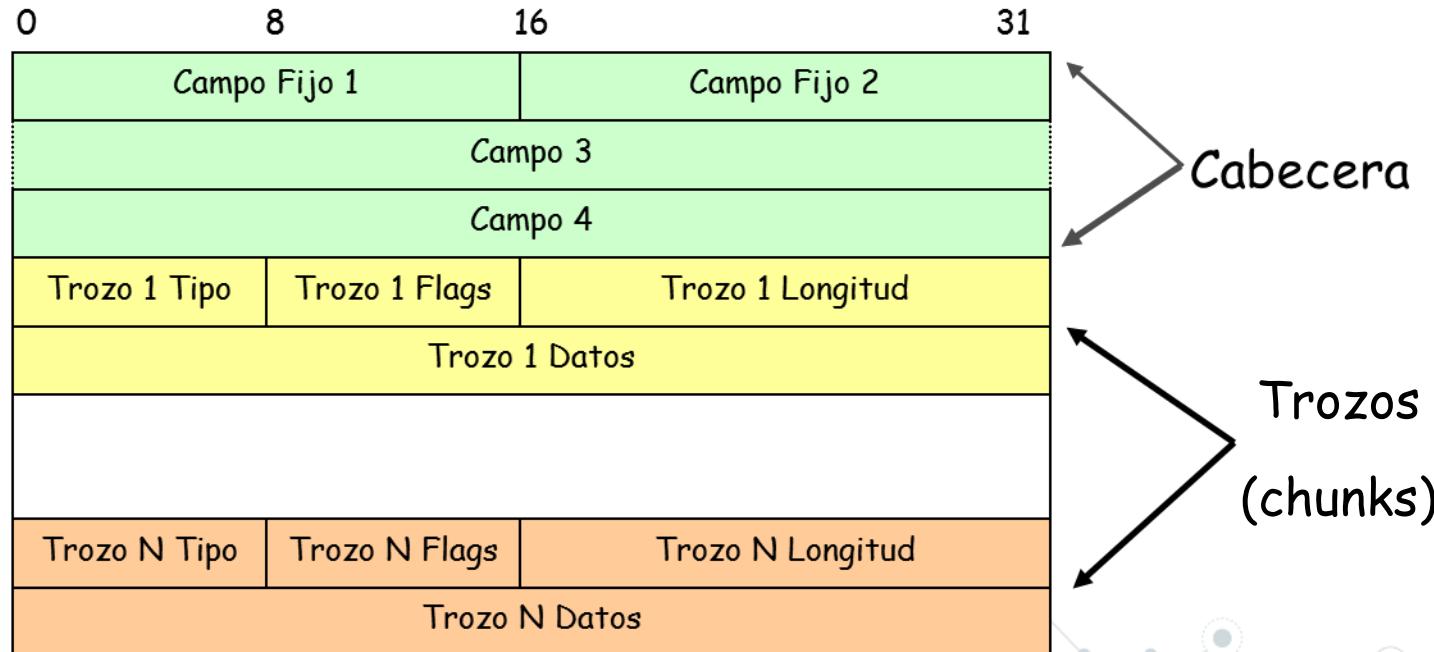
## TIPOS DE PROTOCOLOS

- Protocolos de **dominio público** (Definidos en RFCs (ej., HTTP, SMTP)) vs. **propietarios** (Ej: Skype, IGRP)
- Protocolos **in-band** vs. **out-of-band**
  - **In-band:** protocolos de red con la que se regula el control de datos.
  - **Out-of-band:** (Llamado “Urgent Data” en TCP) Útil para la separación de dos tipos diferentes de datos. No significa que será entregado más rápido o con mayor prioridad que los datos en el flujo de datos in-band.  
(Ej: Protocolo FTP, puertos 20 (Datos) y 21 (Control))
- Protocolos **stateless** vs. **stateful**
  - **stateless:** protocolo que trata cada petición como una transacción independiente que no tiene relación con cualquier solicitud anterior, la comunicación se compone de pares independientes de solicitud/ respuesta.
  - **stateful:** un protocolo que requiere el mantenimiento del estado interno en el servidor.
- Protocolos **persistentes** vs. **no persistentes:** En una conexión persistente solo se hará una conexión TCP, mientras que en una conexión no persistente se utilizarán múltiples conexiones TCP, una por cada objeto solicitado.

# Protocolos

## TENDENCIA: PROTOCOLOS FLEXIBLES

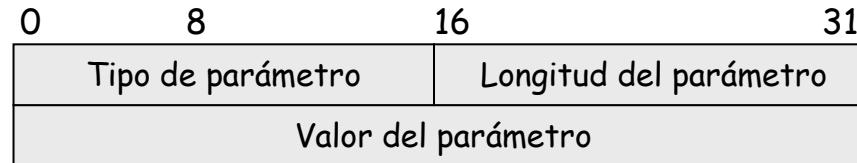
- Una cabecera fija.
- Uno o varios “trozos” (obligatorios u opcionales).



# Protocolos

## TENDENCIA: PROTOCOLOS FLEXIBLES

- Una cabecera fija.
- Uno o varios “trozos” (obligatorios u opcionales).
- Los trozos pueden incluir una cabecera específica más una serie de datos en forma de parámetros:
  - Parámetros fijos: en orden
  - Parámetros de longitud variable u opcionales.
  - Para los parámetros se usa Formato TLV (Type-Length-Variable)



# Protocolos

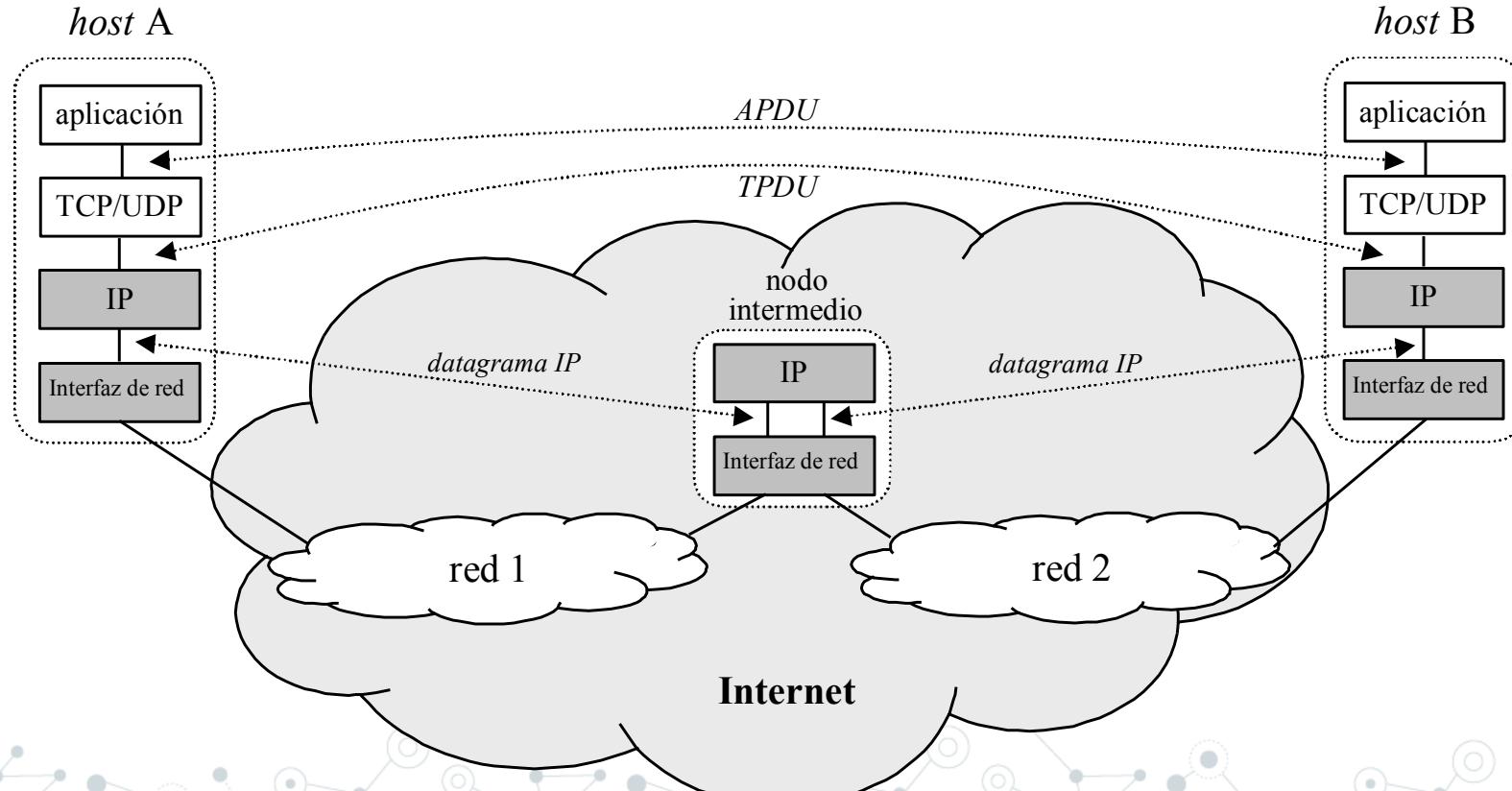
## PROPIEDADES

- **Pérdida de datos (errores)**
  - Algunas aplicaciones (Ej: audio) pueden tolerar algunas pérdida de datos;
  - otras (Ej: FTP, telnet) requieren transferencia 100% fiable
- **Requisitos temporales**
  - Algunas aplicaciones denominadas inelásticas (ej., telefonía Internet, juegos interactivos) requieren retardo acotado (*delay*) para ser efectivas
- **Ancho de banda (tasa de transmisión o *throughput*)**
  - Algunas aplicaciones requieren envío de datos a una tasa determinada
- **Seguridad**
  - Encriptación, autenticación, no repudio, ...

# Protocolos

<b>Application</b>	<b>Data loss</b>	<b>Throughput</b>	<b>Time Sensitive</b>
file transfer	no loss	elastic	no
e-mail	no loss	elastic	no
Web documents	no loss	elastic	no
real-time audio/video	loss-tolerant	audio: 5kbps-1Mbps video:10kbps-5Mbps	yes, 100's ms
stored audio/video	loss-tolerant	same as above	yes, few s
interactive games	loss-tolerant	few kbps up	yes, 100's ms
instant messaging	no loss	elastic	yes and no

# Protocolos



# Protocolos

<b>Application</b>	<b>Application layer protocol</b>	<b>Underlying transport protocol</b>
e-mail	SMTP [RFC 2821]	TCP
remote terminal access	Telnet [RFC 854]	TCP
Web	HTTP [RFC 2616]	TCP
file transfer	FTP [RFC 959]	TCP
streaming multimedia	HTTP (eg Youtube), RTP [RFC 1889]	TCP or UDP
Internet telephony	SIP, RTP, proprietary (e.g., Skype)	typically UDP

# TEMA 5. Servicios y Protocolos en Internet

- 5.1. Introducción a las aplicaciones de red

## **5.2. Servicio de Nombres de Dominio (DNS)**

- 5.3. Navegación web

- 5.4. Correo electrónico

- 5.5. Aplicaciones multimedia

- 5.6. Cuestiones y ejercicios

# Introducción

- La comunicación en Internet **precisa de direcciones IP**.
- Las **direcciones IP son difíciles de memorizar** o recordar
- Necesitamos asignar a dichas direcciones nombres significativos conocidos como nombres de dominio.

**www.ugr.es <-----> 150.214.204.25**

- Los **nombres de dominio**, al igual que las direcciones IP **deben identificar de forma única a una máquina** (una interfaz) en Internet.

# Introducción

## EN LOS COMIENZOS DE INTERNET...

- Se utilizaba una **única tabla centralizada** de traducción de nombres a direcciones.
- En los años 70 la **red ARPANET** estaba formada por unos cientos de máquinas y un sólo archivo, ***HOSTS.TXT*** que  **contenía toda la información** que se necesitaba sobre esas máquinas.
- El centro de información de red del Departamento de defensa americano disponía de la **versión maestra** de la tabla y **otros sistemas** realizaban una **copia** regularmente.

# Introducción

**CON EL CRECIMIENTO DE INTERNET, EL MÉTODO ‘EXPLOTÓ’ POR VARIOS MOTIVOS:**

- El **tráfico de red y la carga para la máquina** que contenía las tablas que hacían posible el mapeo **era desbordante**.
- La **consistencia del archivo** era muy **difícil de mantener**, cuando el `HOST.TXT` llegaba a una maquina muy lejana estaba ya obsoleto.
- **No se podía garantizar la no duplicidad de nombres** (mantener una administración central en una red Internacional era muy complicado).
- El método **no era escalable**.

**CONCLUSIÓN:**

- Este **enfoque quedó obsoleto** debido a su ineficiencia para gestionar una gran cantidad de máquinas.
- Surgió un nuevo sistema de resolución de nombres, **DNS (Domain Name System)**, para solventar los problemas anteriores.

# Sistema de Nombres de Dominio

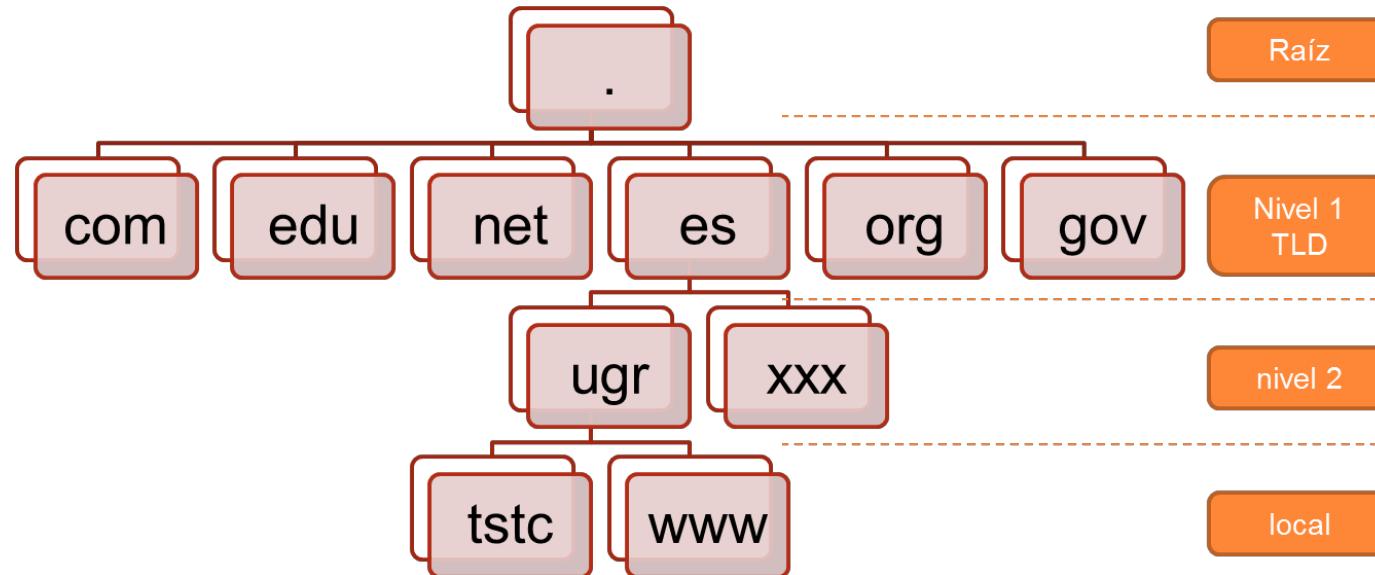


- **Sistema global:** identifica únicamente en todo Internet.
  - ICANN (*Internet Corporation for Assigned Names and Numbers*). Es la autoridad central que controla la entrada de cualquier sistema nuevo. [www.icann.org](http://www.icann.org)
  - NIC (*Network Information Center*). Organizaciones que permiten descentralizar esas tareas, gestionando parte de las numeraciones.
- **Modelo jerárquico:**
  - El dominio se divide en subdominios para facilitar su gestión por los NICs.
  - Se tiene una estructura en árbol.
  - Los dominios de la raíz se denominan Top Level Domain (TLD).
- **Modelo lógico (no físico):** hace falta una traducción.



# Sistema de Nombres de Dominio

## ESTRUCTURA JERÁRQUICA EN ÁRBOL



- Todos los dominios en Internet pueden representarse mediante un árbol.
- Las hojas del árbol serían los dominios que ya no contienen más subdominios.

# Sistema de Nombres de Dominio

## SINTAXIS DEL NOMBRE DE DOMINIO

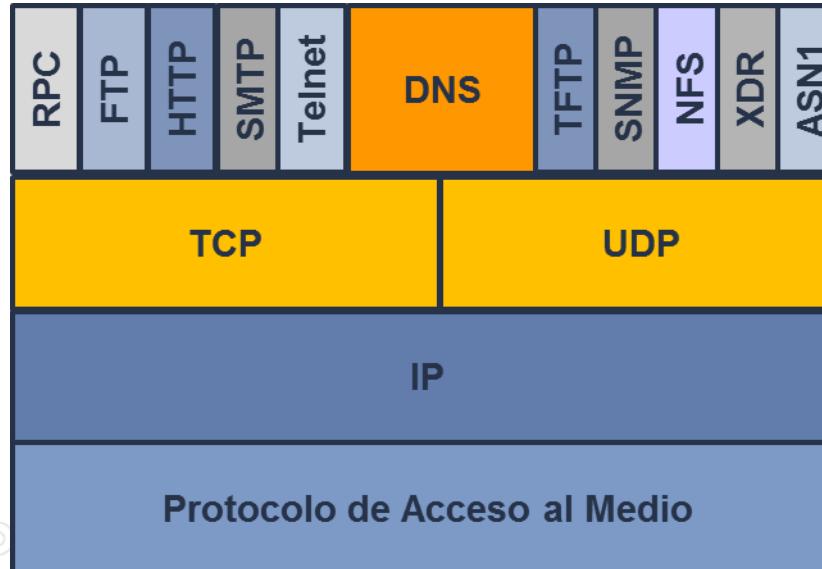
- Cadena de **hasta 255 caracteres**, formada por **etiquetas separadas por puntos** (long. etiqueta < 64 caracteres) que **indican un nivel en la jerarquía**.
- No se distinguen mayúsculas y minúsculas.
- Tipos de nombres de dominio:
  - **Absolutos**: terminados con “.” (Ej: “ugr.es.”)
  - **Relativos**: no terminados con “.”
- En el nivel raíz, los dominios se clasifican en:
  - **Geográficos**: división por países (o regiones).
  - **Genéricos**: en función del tipo de organización.

# Sistema de Nombres de Dominio

- Inicialmente fueron definidos los siguientes **9 dominios genéricos** (RFC 1591):
  - .com** → organizaciones comerciales
  - .edu** → instituciones educativas, como universidades, de EEUU.
  - .gov** → instituciones gubernamentales estadounidenses
  - .mil** → grupos militares de Estados Unidos
  - .net** → proveedores de Internet
  - .org** → organizaciones diversas diferentes de las anteriores
  - .arpa** → propósitos exclusivos de infraestructura de Internet
  - .int** → organizaciones establecidas por tratados internacionales entre gobiernos
  - .xy** → indicativos de la zona geográfica
    - Ej: es (España); pt (Portugal); jp (Japón)...

# Servicio de Nombres de Dominio

- El **servicio DNS** es **transversal** (usado por otros servicios).
- Se sitúa en el esquema de capas de TCP/IP como **protocolo de aplicación**, tanto **sobre UDP** (paquetes de consultas pequeños) como **TCP** (paquetes de consultas grandes).
- Puerto 53 de la capa de transporte.



# Servidores DNS

- El servicio se basa en el uso de una **base de datos descentralizada**, distribuida entre **diversos servidores** (cada uno almacena una parte).
- Cada **servidor** almacenará datos relativos a los **dominios** de los que es **responsable**. Ese grupo de dominios se conoce como **zona**.
- El **servidor** que gestiona una **zona** se dice que tiene **autoridad** sobre ella.
- Hay dos tipos de servidores por zona:
  - **Servidor Autoridad Primario** (o *master*): mantiene una copia principal de la BD. Atiende las peticiones en primera instancia.
  - **Servidor Autoridad Secundario** (o *slave*): almacena una copia de la BD que le transfiere el servidor primario cada cierto tiempo

# Servidores DNS

- Los servidores DNS también almacenan la información sobre los **servidores a consultar** en caso de que se les pregunte por un **dominio sobre el que no tienen autoridad**.
- Además tienen una **caché para almacenar las últimas peticiones** resueltas en caso de que se les soliciten de nuevo.
- Los **servidores raíz** contienen la información de localización de los servidores con autoridad sobre los TLDs.
- Son los **primeros en ser consultados**, por lo que deben estar bien dimensionados, ya que todas las peticiones DNS empiezan en ellos.
- Existen **13 servidores raíz** en el mundo, referenciados con las letras A-M.
- Aunque **cada uno es un servidor distribuido en varias máquinas** ubicadas en múltiples puntos geográficos.

# Servidores Raíz

**(13) Root-Servers** <http://www.root-servers.org/>

Servidor A: Network Solutions, Herndon, Virginia, USA.

Servidor B: Instituto de Ciencias de la Información de la Universidad del Sur de California, USA.

Servidor C: PSINet, Virginia, USA.

Servidor D: Universidad de Maryland, USA.

Servidor E: NASA, en Mountain View, California, USA.

Servidor F: Internet Software Consortium, Palo Alto, California, USA.

Servidor G: Agencia de Sistemas de Información de Defensa, California, USA.

Servidor H: Laboratorio de Investigación del Ejercito, Maryland, USA.

Servidor I: NORDUnet, Estocolmo, Suecia.

Servidor J: (TBD), Virginia, USA.

Servidor K: RIPE-NCC, Londres, Inglaterra.

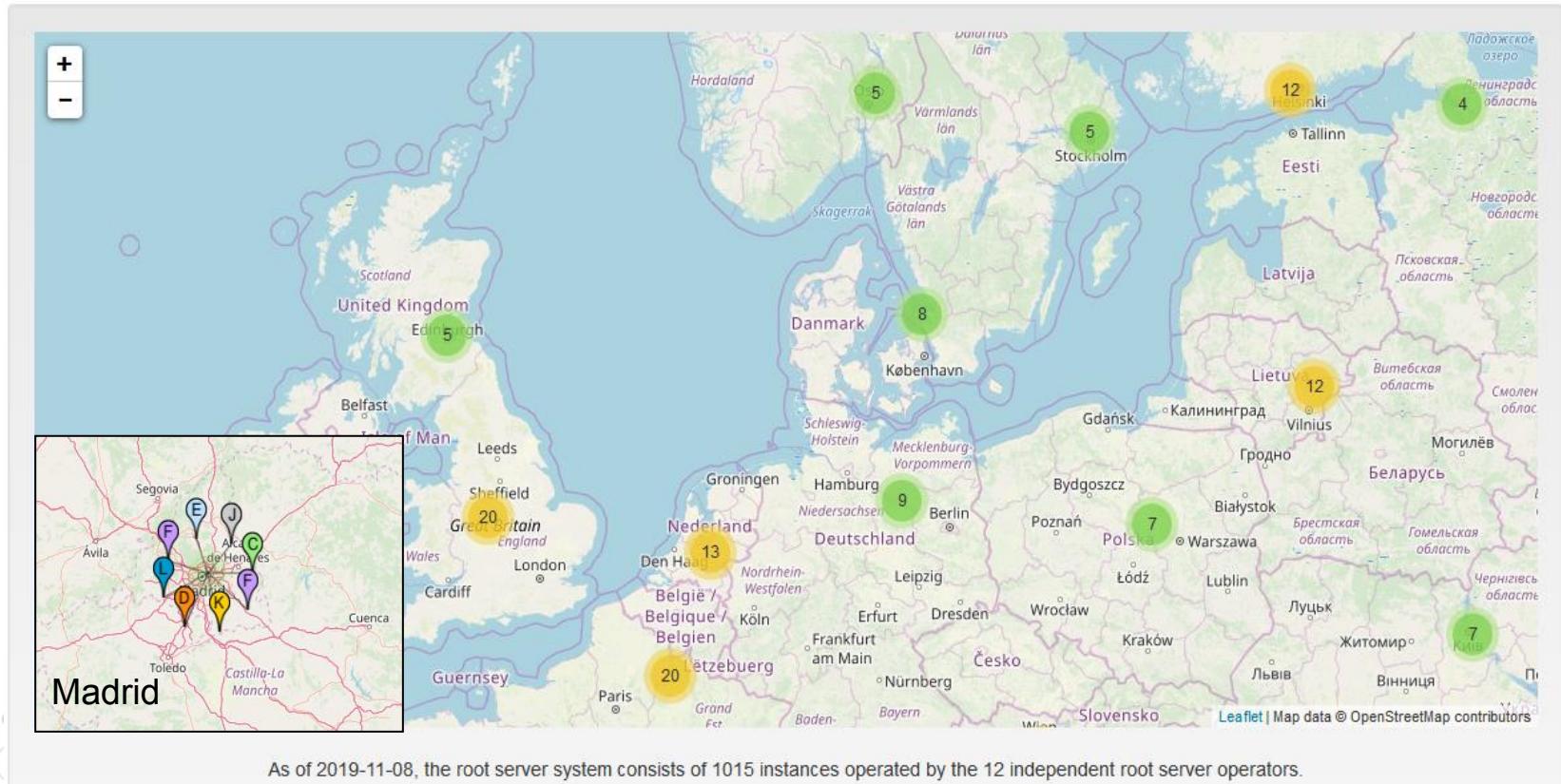
Servidor L: (TBD), California, USA.

Servidor M: Wide Project, Universidad de Tokyo, Japón.



# Servidores Raíz

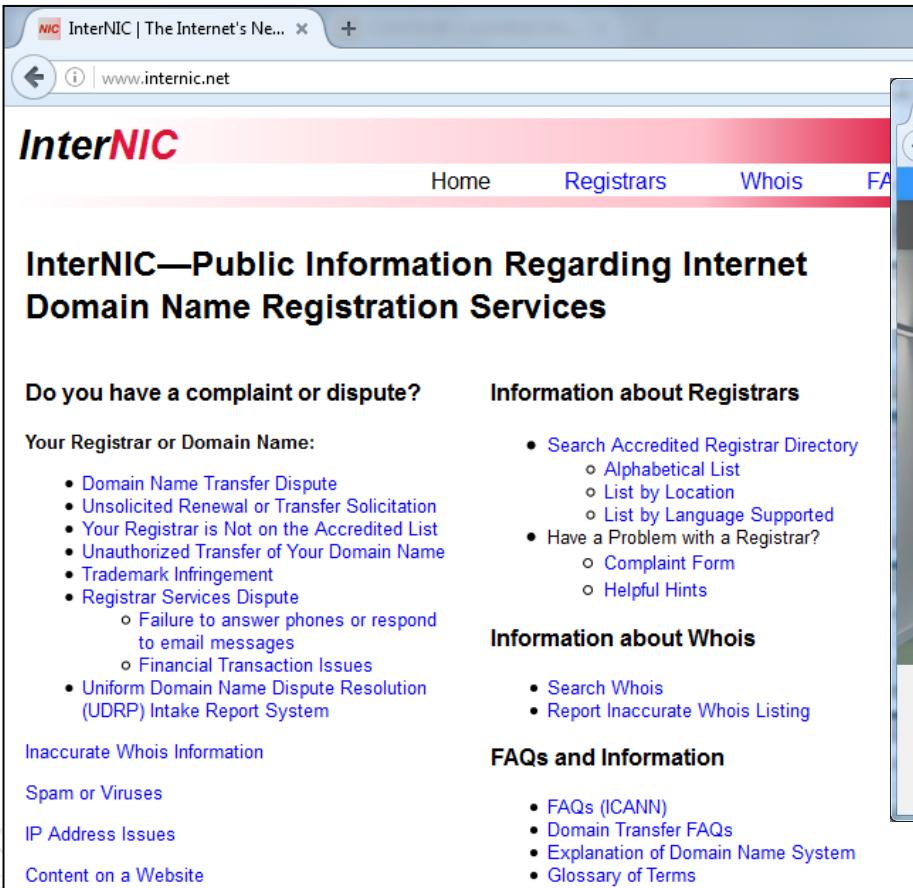
Gestionados por organismos públicos o compañías



# Delegación de la autoridad

- La **organización** que posee un **nombre de dominio**, es **responsable** del funcionamiento y mantenimiento de los **servidores de nombres (zona de autoridad)**.
- La **solicitud de registro de un dominio** se realiza a una autoridad competente, por ejemplo InterNIC (<http://www.internic.net/>) es una **autoridad de registro**.
- Se puede **solicitar** un **dominio** a una **empresa** (Ej: [www.arsys.es](http://www.arsys.es)) y/o **ISP**.
- Cada país dispone de **autoridades de registro**.

# Delegación de la autoridad

A screenshot of the InterNIC website ([www.internic.net](http://www.internic.net)). The page title is "InterNIC—Public Information Regarding Internet Domain Name Registration Services". It features a navigation bar with links for Home, Registrars, Whois, and FAQs. Below the title, there's a section titled "Do you have a complaint or dispute?" with a list of various types of disputes, including Domain Name Transfer Dispute, Unsolicited Renewal or Transfer Solicitation, Your Registrar is Not on the Accredited List, Unauthorized Transfer of Your Domain Name, Trademark Infringement, Registrar Services Dispute (with sub-points about failure to answer phones/respond to messages and financial transaction issues), and Uniform Domain Name Dispute Resolution (UDRP) Intake Report System. There are also links for Inaccurate Whois Information, Spam or Viruses, IP Address Issues, and Content on a Website.

A screenshot of the Arsys website (<https://www.arsys.es>). The page title is "Arsys - Registra tu dominio...". It features a navigation bar with links for ATENCIÓN 24/7, EMAIL, CHAT, ES, EN, PROFESIONALES, SOPORTE, and ÁREA DE CLIENTE. The main content area has a banner with a smiling woman and the text "20 AÑOS DE INNOVACIÓN EN DOMINIOS, HOSTING Y CLOUD" and "DALE LA VUELTA A TU NEGOCIO". It includes sections for "Registra tu dominio" (with points about protecting your brand, 1,500 new extensions, welcome page, and DNS management), "VER DOMINIOS", and "DOMINIOS". On the right, there are sections for "HOSTING", "SERVIDORES CLOUD", and "SOLUCIONES CLOUD". At the bottom, there's a search bar with "www.indicatudominio.com" and a "BUSCAR" button, along with price boxes for ".com" (10€), ".es" (10€), and ".org" (10€).

# Delegación de la autoridad

- En una **zona** existe un **administrador local** que puede delegar en otros administradores.
- Ejemplo: “ugr.es.” puede delegar en el Dep. de TSTC (“tstc.ugr.es.”) para gestionar este dominio inferior.*
- Un mismo recurso puede tener asignados varios dominios o nombres registrados, formando **servidores virtuales**.

*Ejemplo: <http://web1.ugr.es> y <http://www.universidades.org> son dos servidores de dos dominios diferentes pero que se pueden asociar a la misma IP.*

# Funcionamiento del servicio DNS

Las entidades principales que intervienen en el servicio son:

- **Clientes DNS:**

- Programas en los ordenadores de los usuarios que **hacen peticiones** de resolución de nombres (Ej: un navegador web).

- **Servidores DNS:**

- Máquinas que **responden a las consultas** realizadas por los Clientes DNS.

- Pueden dar la **respuesta** bien **por tener autoridad** sobre el dominio en cuestión **o** bien **por tenerla en su caché**.

- En caso de no tenerla pueden consultar a otros servidores DNS.



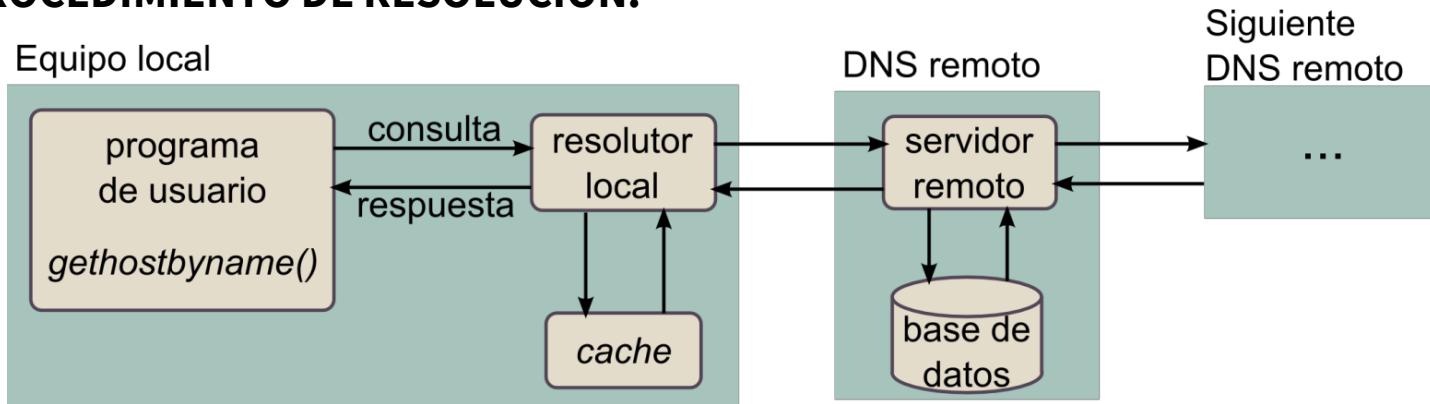
No conoce  
Respuesta



# Funcionamiento del servicio DNS

## PROCEDIMIENTO DE RESOLUCIÓN:

PASO 1

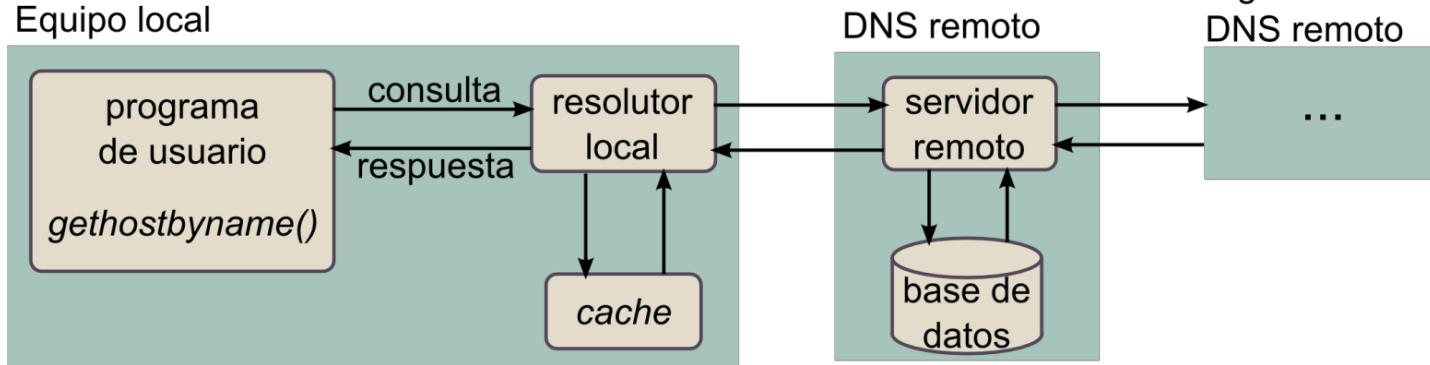


- El **Cliente DNS** (programa de usuario) hace consulta de resolución de nombre de dominio al **resolutor local** (en el mismo equipo). Éste intenta resolver la petición consultando:
  - **Información local de DNS** (ficheros de configuración con asignaciones de IPs).
  - **Información en caché** (memoria con resoluciones recientes).
- Si encuentra la respuesta, se la pasa al programa cliente.

# Funcionamiento del servicio DNS

## PROCEDIMIENTO DE RESOLUCIÓN:

PASO 2

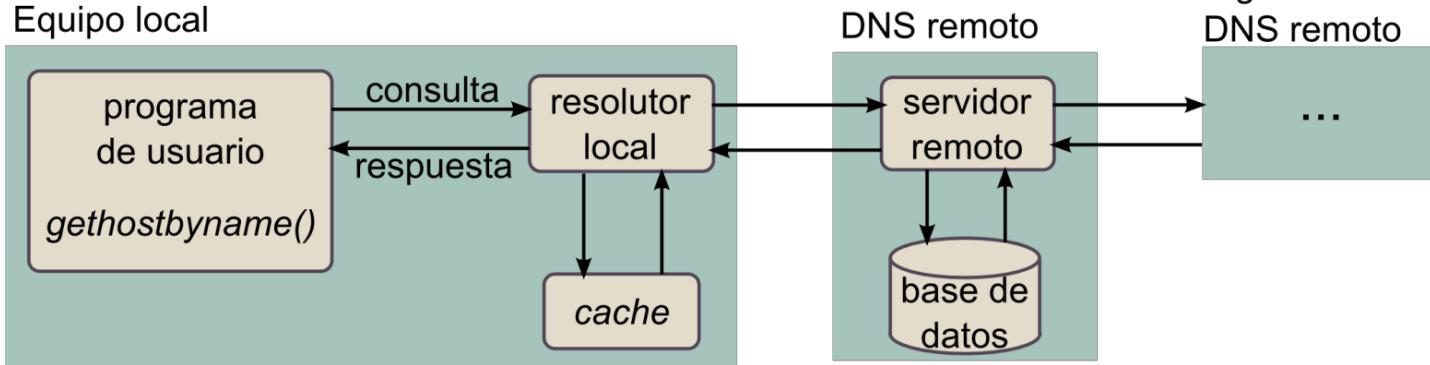


- Si el **resolutor local** no encuentra respuesta, **pasa la consulta** al servidor DNS que tenga asignado el equipo local por defecto (puede haber dos).
- El **servidor DNS comprueba** si dispone de la información para la solicitud en su base de datos, es decir, **si tiene autoridad** sobre ese dominio.

# Funcionamiento del servicio DNS

## PROCEDIMIENTO DE RESOLUCIÓN:

PASO 3

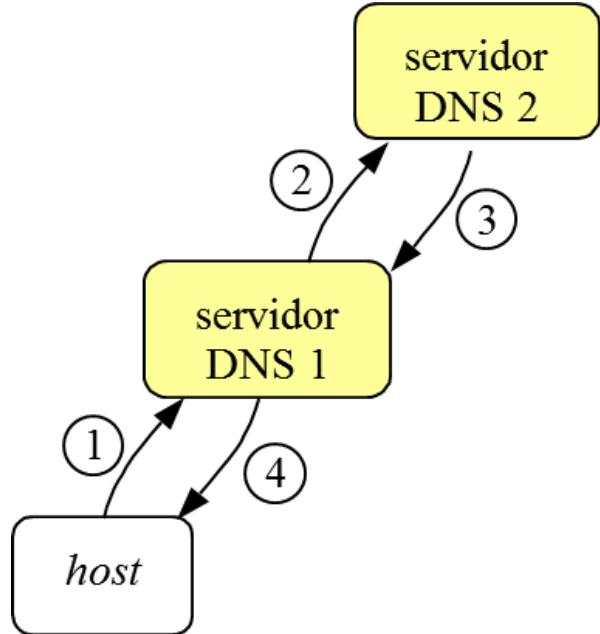


- Si el **servidor DNS no tiene autoridad**, debe encontrar al servidor que la tenga:
  - Consulta a los **servidores raíz** por el servidor con autoridad para el **TLD**.
  - Consulta al **servidor con autoridad** para el **dominio de segundo nivel**.
  - Y así **sucesivamente** hasta llegar al **servidor con autoridad para el dominio local**, que sería el que enviaría la respuesta.

# Funcionamiento del servicio DNS

## RESOLUCIÓN DISTRIBUIDA:

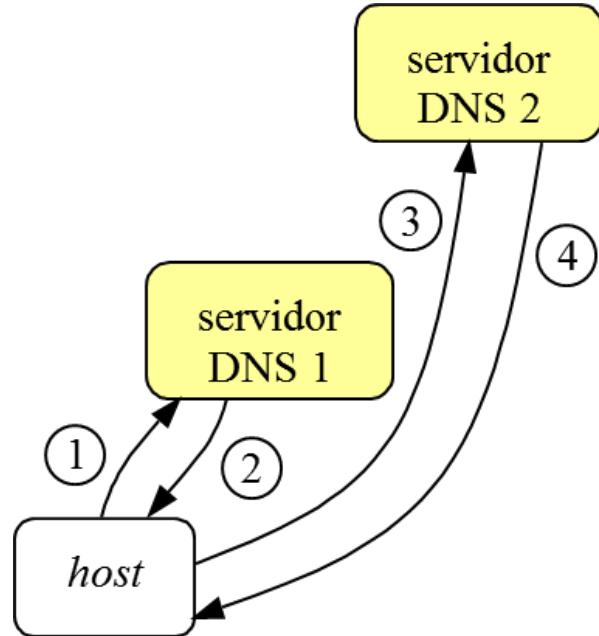
- **Recursiva:** el servidor DNS 1 (servidor por defecto) recibe la consulta del host y determina que es el servidor DNS 2 quien tiene autoridad sobre el dominio. Transfiere la consulta al servidor DNS 2 y recibe la respuesta, que luego pasa al host. Este último no será consciente de los saltos o consultas que haya tenido que realizar DNS 1.



# Funcionamiento del servicio DNS

## RESOLUCIÓN DISTRIBUIDA:

- **Iterativa:** el servidor DNS 1 envía al host la información sobre el siguiente servidor al que tendrá que consultar y será el host quién realice la siguiente consulta a DNS 2.

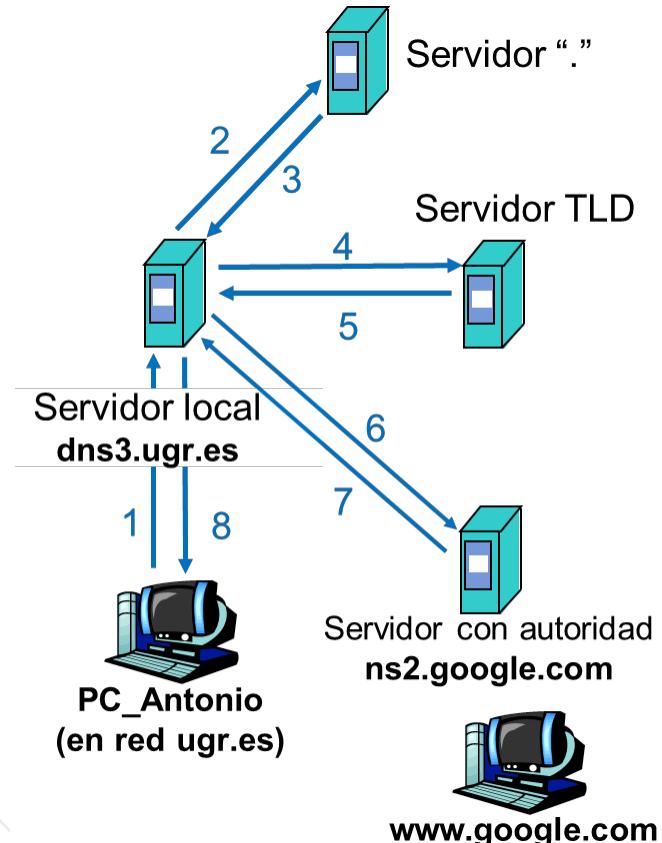


# Funcionamiento del servicio DNS

## PROCEDIMIENTO DE RESOLUCIÓN DE DNS DISTRIBUIDA:

- Ejemplo de funcionamiento del servicio de DNS para una consulta sobre la IP del nombre de dominio “[www.google.com](http://www.google.com)” realizada desde un equipo en la red de la UGR ([ugr.es](http://ugr.es)).

*>>> Resolución de forma recursiva respecto al PC e iterativa respecto a servidor local de DNS <<<*



# Formato de la BD de DNS

- Todo **dominio** está **asociado** al menos a un **Resource Record (Registro de recursos)**.
- El formato de los registros es de la siguiente forma:

*[Nombre\_dominio] [TTL] [Clase] Tipo Valor\_tipo*

- Cuando un cliente (*resolutor*) da un **nombre de dominio al DNS**, lo que **recibe son los RR asociados** a ese nombre.
- Normalmente existen varios RR por dominio.

# Formato de la BD de DNS

- **Nombre\_dominio:** puede haber más de un registro por dominio. Se puede omitir, tomando por defecto el último nombre de dominio indicado.
- **TTL:** tiempo de vida (estabilidad del registro). La información *altamente estable* tiene un valor grande (86400 seg. o un día), mientras que la *volátil* recibe un valor pequeño (60 seg.).
- **Clase:** Actualmente sólo se utiliza *IN*, para información de Internet.

# Formato de la BD de DNS

- **Valor\_tipo:** es un número o texto ASCII dependiendo del tipo.

Tipo de Registro	Descripción
SOA <i>Start Of Authority</i>	Inicio de autoridad, identificando el dominio o la zona. Fija una serie de parámetros para esta zona.
NS <i>Name Server</i>	El nombre de dominio se hace corresponder con el nombre de un servidor con autoridad para dicho dominio.
A <i>Addres</i>	Dirección IP correspondiente al dominio (formato 32 bits). Si este tiene varias direcciones IP, habrá un registro por cada una de ellas.
CNAME	Es un alias que se corresponde con el nombre canónico verdadero.
MX <i>Mail eXchanger</i>	Indica la dirección IP del servidor de e-mail que corresponde a un nombre de dominio.
TXT	Texto, es una forma de añadir comentarios a la Base de Datos. Por ejemplo, para dar la dirección postal del dominio.
PTR <i>Pointer</i>	Puntero, hace corresponder una dirección IP con un nombre de dominio. Se usa en archivos dirección-nombre, la inversa del tipo A.
HINFO	Información del Host, como tipo y modelo de computadora.
WKS	Servicios públicos (Well-Known Services). Puede listar los servicios de las aplicaciones disponibles en el ordenador.

# Formato de la BD de DNS

- Ejemplo de fichero de DNS (servidor BIND)

```
labredes.pri. 94400 IN SOA eihal.labredes.pri. admin.labredes.pri.(
                      2008042401      ; Serial
                      28800            ; Refresh (seconds)
                      14400            ; Retry (seconds)
                     360000           ; Expire (seconds)
                     86400 )          ; Minimum TTL(seconds)
labredes.pri.          IN NS   eihal.labredes.pri.
labredes.pri.          IN MX   10 mailserver.labredes.pri.
controler.labredes.pri. IN CNAME eihal.labredes.pri.

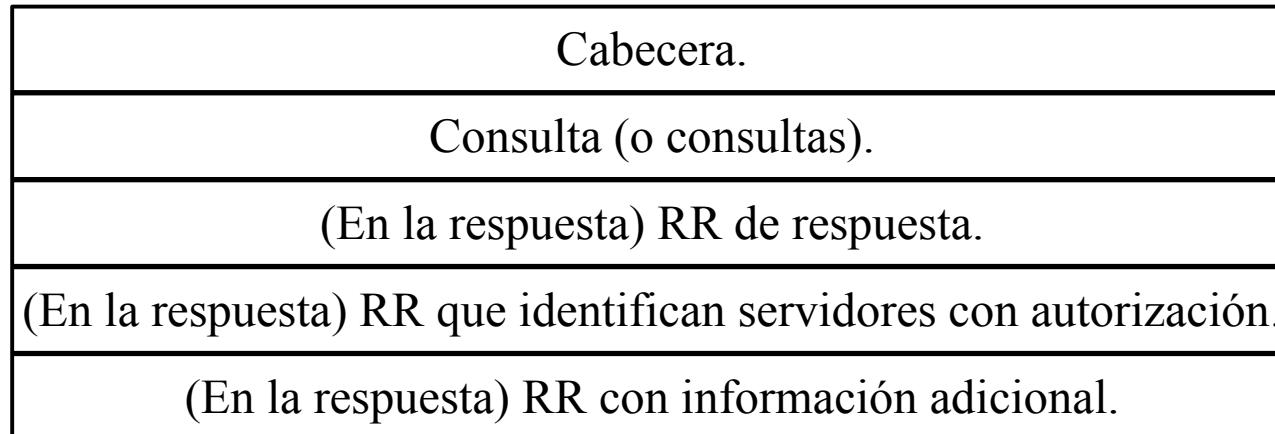
eihal.labredes.pri.    IN A    172.18.140.21
eil40146.labredes.pri. IN A    172.18.140.146
mailserver.labredes.pri. IN A    172.18.140.148
voipserver.labredes.pri. IN A    172.18.140.149
```

# Consultas inversas

- Existen registros en la base de datos para la **resolución inversa** (traducir direcciones IP a nombres de dominio).
- Se refieren al dominio especial “**in-addr.arpa**”.
- El dominio asociado a la **dirección IP w.x.y.z** se almacenará:
  - Campo Nombre\_dominio → z.y.x.w.in-addr.arpa.
  - Campo PTR → nombre del dominio correspondiente a la IP.
- La inversión de los bytes es necesaria debido a que los nombres de dominio son más genéricos por la derecha, al contrario que ocurre con las direcciones IP.

# Formato de los mensajes DNS

- Los mensajes de consulta y respuesta intercambiados entre clientes y servidores DNS tienen un formato sencillo. Un servidor añade la información requerida a la consulta original y la envía de vuelta.



# Formato de los mensajes DNS

- Campos de la **cabecera**:

Campo	Descripción
ID	Identificador para hacer corresponder una respuesta con su petición.
Parámetros	Consulta o respuesta. Consulta normal o inversa. En respuestas, si es de un servidor con autoridad. Recursivo o no. En respuestas, si la recursión está disponible. En respuestas, código de error.
Num. de consultas	Proporcionado en una consulta y en una respuesta.
Num. de respuestas	Proporcionado en una respuesta.
Num. de registros de autoridad	Proporcionado en una respuesta. La información de los registros de autoridad incluye los nombres de los servidores que contienen los datos de confianza.
Num. de registros adicionales	Proporcionado en una respuesta. La información incluye las direcciones de los servidores de confianza.

# Formato de los mensajes DNS

- Campos de una **consulta** (puede haber varias en una petición):

Campo	Descripción
Nombre	Nombre de dominio o dirección IP en el subárbol IN-ADDR.ARPA
Tipo	Tipo de consulta, por ejemplo A o NS
Clase	IN para Internet, se representa como 1

# Formato de los mensajes DNS

- Campos de **respuesta, información de autoridad e información adicional**. Secuencia de RR.

Campo	Descripción
Nombre	Nombre del nodo para este registro.
Tipo	Tipo de registro, como SOA o A, indicado por un código numérico.
Clase	IN, se representa como 1.
TTL	Tiempo de vida, un entero con signo de 32 bits que indica cuánto tiempo puede permanecer el registro en la caché.
RDLENGTH	Tamaño del campo de datos de recursos.
RDATA	La información, por ejemplo, para un registro de direcciones, es la dirección IP. Para un registro SOA, incluye más datos.

# Formato de los mensajes DNS

Ejemplo de  
consulta a  
“tstc.ugr.es”  
(captura Wireshark)



The screenshot shows a Wireshark capture window titled "\*Wi-Fi". The packet list pane shows two DNS packets. The first packet (No. 7) is a query from 192.168.1.48 to 192.168.1.1 with a length of 71 bytes. The second packet (No. 8) is a response from 192.168.1.1 to 192.168.1.48 with a length of 108 bytes. The details pane displays the DNS request for "tstc.ugr.es" with a transaction ID of 0x351d, flags indicating a standard query, and a single query for the "tstc.ugr.es" A record. The bytes pane shows the raw hex and ASCII data of the captured frames.

No.	Time	Source	Destination	Protocol	Length	Info
7	8.286785	192.168.1.48	192.168.1.1	DNS	71	Standard query 0x351d A tstc.ugr.es
8	8.312647	192.168.1.1	192.168.1.48	DNS	108	Standard query response 0x351d A tstc.ugr.es

Frame 7: 71 bytes on wire (568 bits), 71 bytes captured (568 bits) on interface 0

Ethernet II, Src: IntelCor\_1b:53:19 (c8:f7:33:1b:53:19), Dst: Arcadyan\_6d:a3:70 (48:8d:36:6d:a3:70)

Internet Protocol Version 4, Src: 192.168.1.48, Dst: 192.168.1.1

User Datagram Protocol, Src Port: 50231, Dst Port: 53

Domain Name System (query)

- Transaction ID: 0x351d
- Flags: 0x0100 Standard query
- Questions: 1
- Answer RRs: 0
- Authority RRs: 0
- Additional RRs: 0

Queries

- tstc.ugr.es: type A, class IN
  - Name: tstc.ugr.es
  - [Name Length: 11]
  - [Label Count: 3]
  - Type: A (Host Address) (1)
  - Class: IN (0x0001)

[Response In: 8]

0000 48 8d 36 6d a3 70 c8 f7 33 1b 53 19 08 00 45 00 H·6m·p··3·S···E·

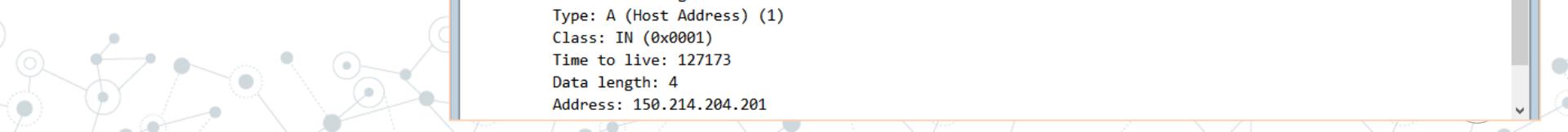
wireshark\_A35C9EEC-9784-404B-8E7E-003A47947909\_20191114121855\_a06004.pcapng

Packets: 44 · Displayed: 2 (4.5%)

Profile: Default

# Formato de los mensajes DNS

Ejemplo de respuesta de “ugr.es” por “tstc” (captura Wireshark)



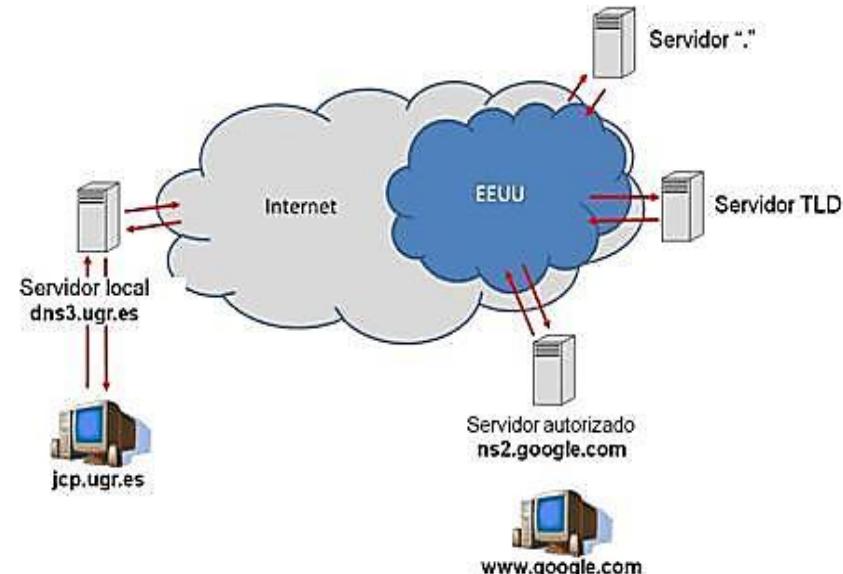
The screenshot shows a Wireshark capture window titled "dns". The packet list pane displays two DNS messages. The first message (No. 7) is a query from 192.168.1.48 to 192.168.1.1 with a transaction ID of 0x351d. The second message (No. 8) is a response from 192.168.1.1 to 192.168.1.48 with a transaction ID of 0x351d. The details pane shows the DNS response structure:

- Transaction ID: 0x351d
- Flags: 0x8180 Standard query response, No error
- Questions: 1
- Answer RRs: 2
- Authority RRs: 0
- Additional RRs: 0
- Queries
- Answers
  - tstc.ugr.es: type CNAME, class IN, cname uniweb.ugr.es
    - Name: tstc.ugr.es
    - Type: CNAME (Canonical NAME for an alias) (5)
    - Class: IN (0x0001)
    - Time to live: 172800
    - Data length: 9
    - CNAME: uniweb.ugr.es
  - uniweb.ugr.es: type A, class IN, addr 150.214.204.201
    - Name: uniweb.ugr.es
    - Type: A (Host Address) (1)
    - Class: IN (0x0001)
    - Time to live: 127173
    - Data length: 4
    - Address: 150.214.204.201

# Ejercicio

## EJERCICIO (RELACIÓN DE PROBLEMAS DEL TEMA 5)

- En la siguiente figura se ilustra un ejemplo de acceso DNS por parte de una máquina (jcp.ugr.es) que quiere acceder a los servicios de www.google.com. Para obtener la dirección IP del servidor, es necesario que la consulta pase por todos los servidores del gráfico. Considerando unos retardos promedio de 8  $\mu$ s dentro de una red LAN, de 12 ms en cada acceso a través de Internet (4 ms si la conexión se restringe a EEUU) y de 1 ms de procesamiento en cada servidor:
  - Calcule el tiempo que se tardaría si la solicitud al servidor local es recursiva, pero el propio servidor local realiza solicitudes iterativas.
  - Especifique una política (recursiva-iterativa) más rápida de solicitudes y el tiempo que tardaría la solicitud en ser respondida. ¿Qué desventaja tiene sobre la solución anterior?



# TEMA 5. Capa de Aplicación

- 5.1. Introducción a las aplicaciones de red
- 5.2. Servicio de Nombres de Dominio (DNS)
- **5.3. Navegación web**
- 5.4. Correo electrónico
- 5.5. Aplicaciones multimedia
- 5.6. Cuestiones y ejercicios

# Introducción

- La **WWW** (World Wide Web) es la **aplicación más importante** en Internet.
- WWW es un sistema de distribución de información **basado en hipertexto** o hipermedios enlazados y accesibles a través de Internet.
- Con un **navegador web**, se accede a **páginas web** que pueden contener texto, imágenes, vídeos u otros contenidos multimedia, y se navega a través de ellas usando hiperenlaces.
- En los últimos años ha crecido enormemente gracias a:
  - Presentación atractiva
  - Fácil de usar
  - Interface unificado para todos los servicios
  - Permite de manera flexible e interactiva acceder a grandes cantidades de información

# Introducción

- Por su flexibilidad, puede dar **soporte a multitud de servicios diferentes** (información, publicación de contenidos, interacción entre usuarios, servicios comerciales, publicidad, cursos, bases de datos, etc).
- Es **muy fácil publicar nueva información** y hacerla accesible a todo el mundo.
- Está en **continua evolución**, y cada día sus capacidades de acceso y representación de información se vuelven más sofisticadas.
- Ha sido la **principal causa** del espectacular **crecimiento** que **Internet** ha tenido en los últimos años, tanto en número de usuarios como en volumen de información disponible.
- También sirve como **soporte para** las denominadas "**Intranets**".

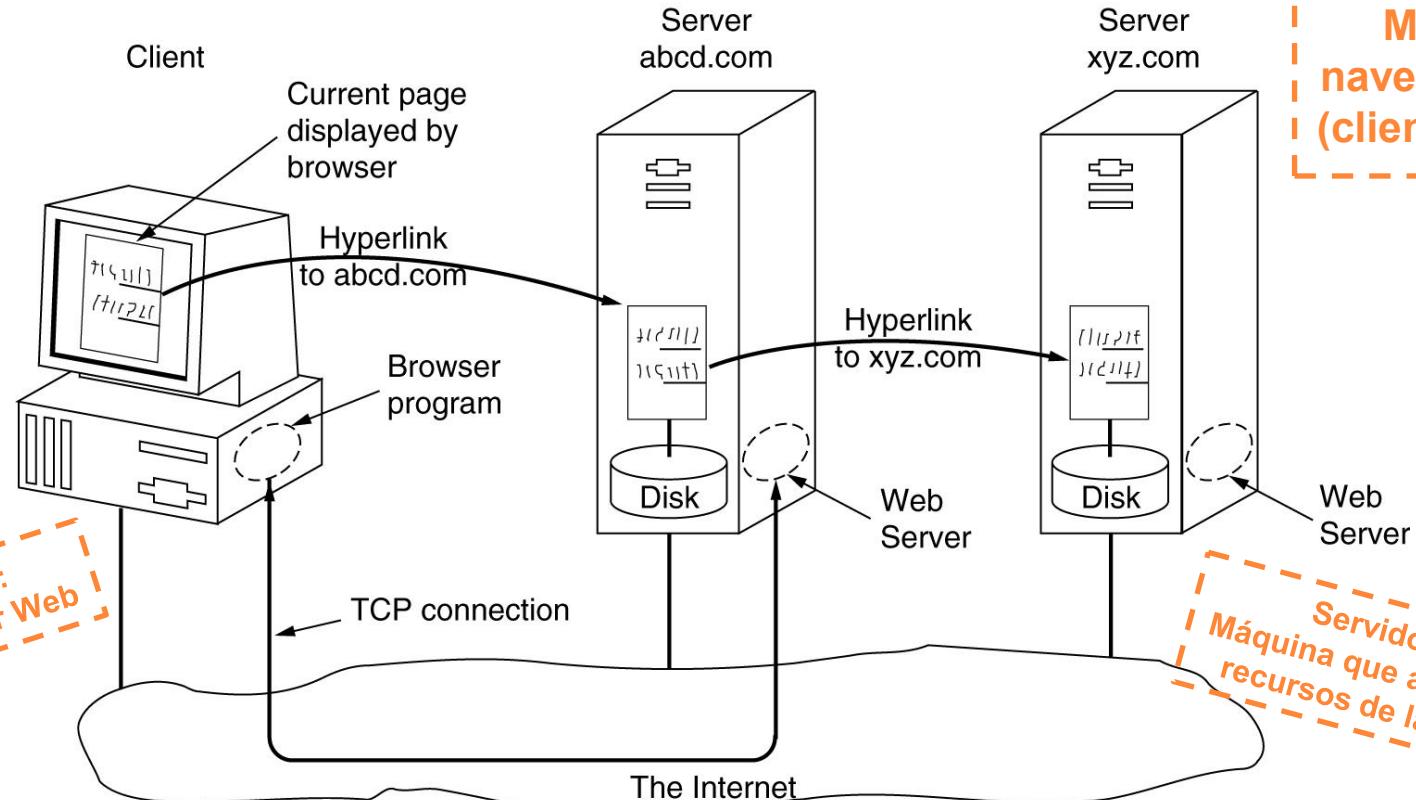
Redes privadas que  
usan tecnología de  
Internet

# Introducción

- Se destacan los siguientes **estándares**:
  - El **Identificador de Recurso Uniforme (URI)**, que es un sistema universal para referenciar recursos en la Web.
  - El **Protocolo de Transferencia de Hipertexto (HTTP)**, que especifica cómo se comunican el navegador y el servidor entre ellos.
  - El **Lenguaje de Marcado de Hipertexto (HTML)**, usado para definir la estructura y contenido de documentos de hipertexto (páginas web).
  - El **Lenguaje de Marcado Extensible (XML)**, usado para describir la estructura de los documentos.
- El World Wide Web Consortium (W3C) desarrolla y mantiene estos y otros estándares que permiten a los ordenadores de la Web almacenar y comunicar efectivamente diferentes formas de información.

Una URL  
es un URI

# Introducción



**Modelo de navegación Web (cliente/servidor)**

# Cliente web

- Un **cliente web** (también llamado navegador o *browser*) es esencialmente un programa que **permite visualizar e interaccionar** con páginas web.
- Sirve para acceder a la www y “navegar” por ella a través de los enlaces.
- El navegador hace **peticiones al servidor** para recibir los ficheros asociados a las páginas web (se resuelve previamente la correspondencia entre el nombre de dominio y la IP del servidor con DNS).
- Tiene soporte para imágenes, sonidos y videos.
- No todas las páginas contienen HTML, las hay que pueden tener un documento PDF, un icono GIF, un vídeo en MPEG... El servidor indica el tipo MIME. Si el tipo MIME no es de los integrados hay dos posibilidades:
  - Plug-in
  - Aplicaciones auxiliares
- Puede utilizar otros protocolos, como ftp o file (ficheros locales).

MIME:  
Multipurpose Internet  
Mail Extensions

# Cliente web

## PROCESAMIENTO

<http://www.epsg.upv.es/historia.php?modo=presentacion&titulo=Hist%F2ria>

1. El navegador determina la URL (de un enlace)
2. Accede al servicio DNS para averiguar la dirección IP de *www.epsg.upv.es*
3. DNS contesta 158.42.144.1
4. El navegador se conecta al puerto TCP 80 de 158.42.144.1
5. Y envía “*GET /historia.php?modo=presentacion&titulo=Hist%F2ria*”
6. El servidor manda el fichero *historia.php*
7. Si existen imágenes u otro contenido asociado, el navegador las solicita y se envían
8. Se cierra la conexión
9. El navegador visualiza el contenido de *historia.php* y los recursos asociados



# Servidor web

- Máquina que aloja los recursos de las páginas web.
- Escucha conexiones de tipo TCP en el puerto 80.
- Entrega los ficheros requeridos a través del protocolo HTTP.

## PROCESAMIENTO (básico)

1. Acepta una conexión TCP de un cliente (navegador)
2. Obtiene el nombre del archivo solicitado por el cliente
3. Recupera el archivo (del disco), así como los dependientes
4. Envía el/los archivo/s al cliente
5. Libera la conexión TCP



# Servidor web

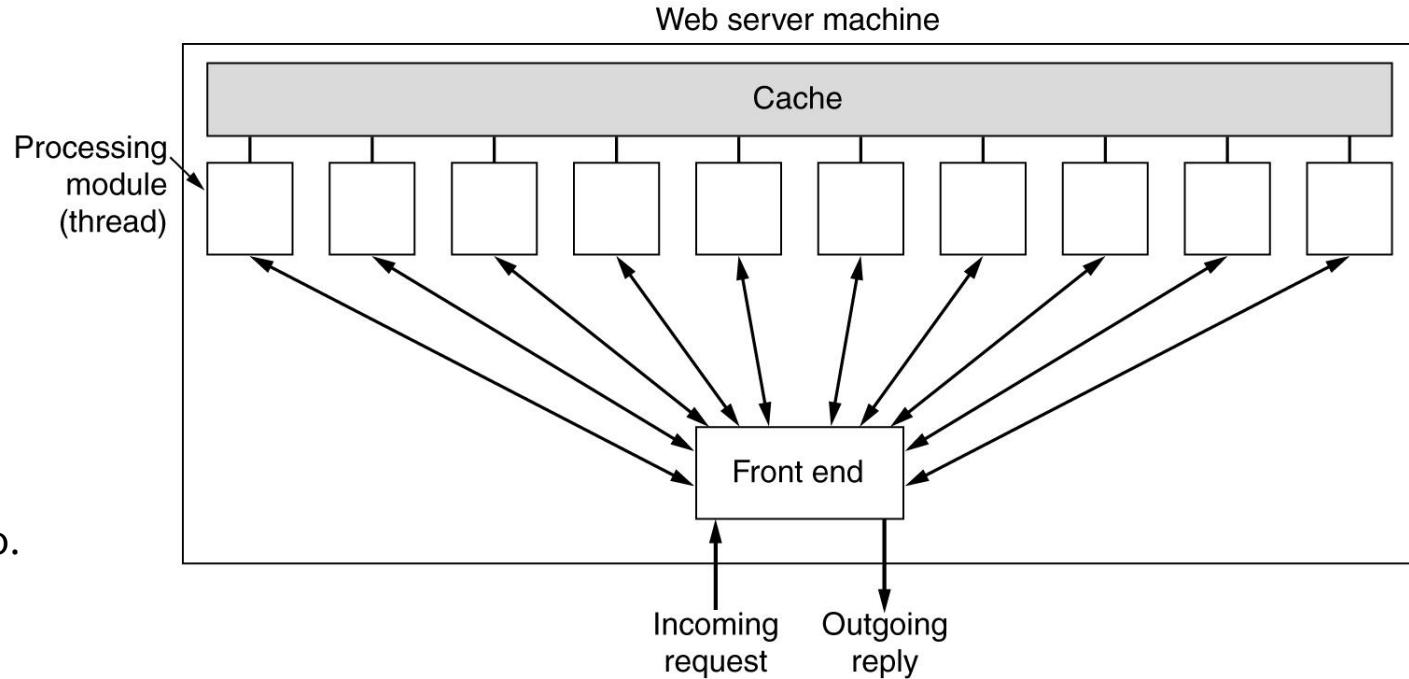
## PROCESAMIENTO (avanzado)

1. Resuelve el nombre de la página web solicitado
2. Autentica al cliente
3. Realiza control de acceso en el cliente
4. Realiza control de acceso en la página web
5. Verifica la caché
6. Obtiene del disco la página solicitada
7. Determina el tipo MIME que se incluirá en la respuesta
8. Devuelve la respuesta al cliente
9. Realiza una entrada en el registro del servidor

# Servidor web

## ARQUITECTURA HABITUAL

Servidor web multihilo con módulos de acceso (Front End) y de procesamiento.



# Protocolo HTTP

- El **Protocolo de Transferencia de HiperTexto** es un sencillo protocolo **cliente/servidor** que articula los intercambios de información entre los clientes y los servidores web.
- Esta soportado sobre los servicios que ofrecen los **protocolos TCP e IP**.
- Un proceso **servidor** escucha en un **puerto** de comunicaciones **TCP (80)** y espera solicitudes de los clientes web.
- Una vez establecida la conexión, **HTTP** se encarga de **mantener la comunicación** y garantizar un intercambio de datos **libre de errores**.
- Se basa en sencillas operaciones de solicitud/respuesta.
  - cliente → envía mensaje con los datos de la solicitud (**request**)
  - servidor → envía mensaje con el estado de la operación y el resultado (**response**)

# Protocolo HTTP

## CARACTERÍSTICAS PRINCIPALES

- Protocolo **basado en ASCII**. De esta forma se **puede transmitir cualquier tipo de documento**: texto, binario, etc, respetando su **formato original**.
- Permite la **transferencia de objetos multimedia**. El contenido de cada objeto intercambiado está identificado por su clasificación MIME.
- Existen **tres funciones básicas** (otras no se utilizan) que un **cliente puede utilizar** en sus solicitudes:
  - **GET** → para recoger un objeto
  - **POST** → para enviar información al servidor
  - **HEAD** → para solicitar las características de un objeto (Ejemplo: fecha de modificación de un documento HTML).

# Protocolo HTTP

## CARACTERÍSTICAS PRINCIPALES

- Protocolo “**stateless**” → El servidor no mantiene información de las peticiones de los clientes.
  - | **Cookie:** información breve (y estructurada) enviada por un sitio web y almacenada en el navegador (cliente).
  - | La puede consultar el sitio web en futuras visitas
- Dos tipos de servicio:
  - **No persistente** → Se envía únicamente un objeto en cada conexión TCP.
  - **Persistente** → Pueden enviarse múltiples objetos sobre una única conexión TCP entre cliente y servidor

# Protocolo HTTP

## FUNCIONAMIENTO

1a. El Cliente HTTP inicia conexión TCP al servidor HTTP (proceso) en www.ugr.es en el puerto 80 (segmento SYNC de TCP)

2. El Cliente HTTP envía ***request message*** para el objeto

1b. El Servidor HTTP acepta la conexión y solicita al cliente abrir la conexión (SYNC+ACK)  
1c. El cliente confirma (ACK)

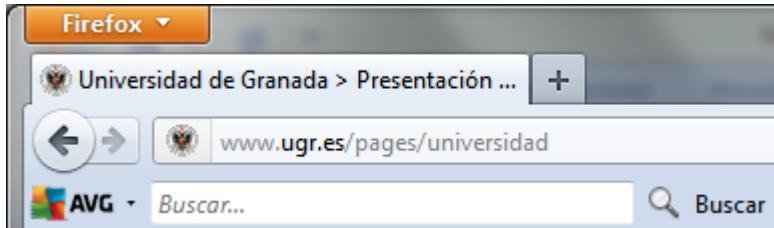
3. El servidor HTTP devuelve la respuesta (***response message***)

4. Si es persistente → Envío de más objetos por la misma conexión TCP

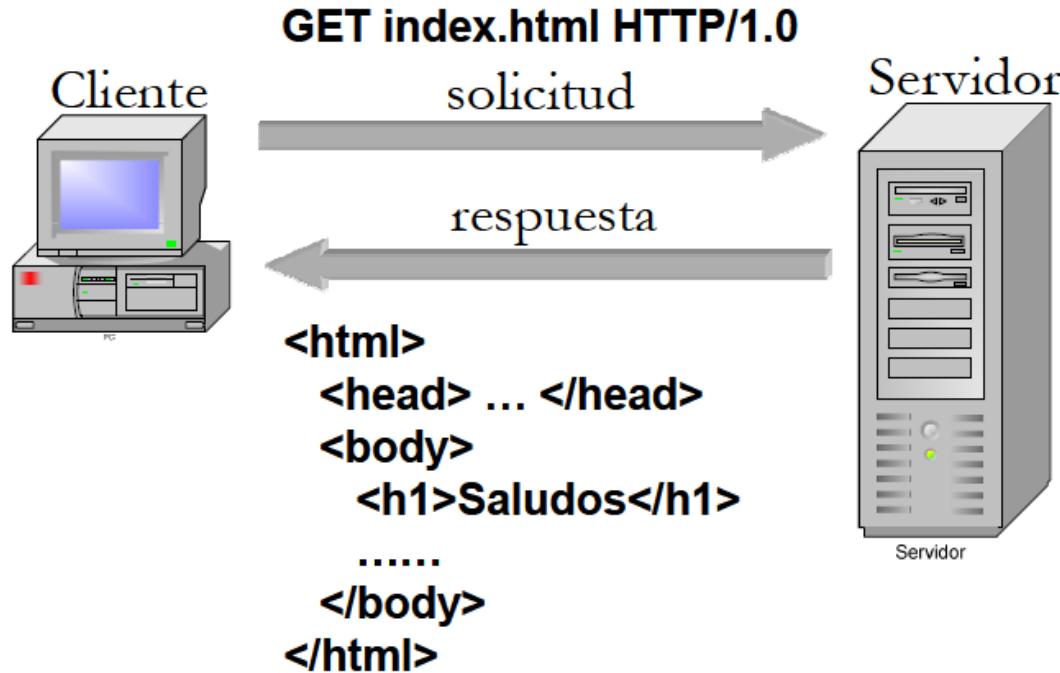
5. Cierre de conexión TCP (liberación de recursos)

6. Nuevas conexiones TCP

tiempo



# Protocolo HTTP



# Ejercicio

## EJERCICIO (RELACIÓN DE PROBLEMAS DEL TEMA 5)

- Compare el rendimiento en términos temporales de HTTP persistente y no persistente considerando los siguientes parámetros:
  - Descarga de una página web con 10 objetos incrustados
  - Tiempo de Establecimiento de conexión TCP → 5 ms
  - Tiempo de Cierre de conexión TCP → 5 ms
  - Tiempo de solicitud HTTP → 2 ms
  - Tiempo de respuesta HTTP (página web u objeto) → 10 ms

# Protocolo HTTP 1.1 (RFC 2616)

## MENSAJE REQUEST

- HTTP request message (solicitudes del cliente al servidor).

Línea de petición  
(GET, POST,  
HEAD)

Líneas de cabecera

Carriage Return (CR) +  
Line Feed (LF)  
Indican fin del mensaje

**GET /somedir/page.html HTTP/1.1**  
**Host: www.someschool.edu**  
**User-agent: Mozilla/4.0**  
**Connection: close**  
**Accept-language:fr**

# Protocolo HTTP 1.1 (RFC 2616)

## MENSAJE RESPONSE

- HTTP response message: (respuestas del servidor al cliente).

Línea de estado

HTTP/1.1 200 OK  
Connection: close  
Date: Thu, 06 Aug 1998 12:00:15 GMT  
Server: Apache/1.3.0 (Unix)  
Last-Modified: Mon, 22 Jun 1998 ....  
Content-Length: 6821  
Content-Type: text/html

200 OK  
301 Moved Permanently  
400 Bad Request  
404 Not Found  
505 HTTP Version Not  
Supported

Líneas de cabecera

Datos,

Ej: fichero html

data data data data data ...

# Protocolo HTTP 1.1 (RFC 2616)

## MÉTODOS (Acciones solicitadas en los *request messages* del cliente)

- **OPTIONS:** solicitud de información sobre las opciones disponibles
- **GET:** solicitud de un recurso (puede ser condicional)  
(Se envía al pulsar sobre un enlace o al teclear una URL directamente en el navegador)
- **HEAD:** igual que GET pero el servidor no devuelve el “cuerpo” sólo cabeceras  
(Utilizado por los gestores de cachés de páginas, para saber cuándo es necesario actualizar la copia que se tiene de un fichero)
- **POST:** solicitud al servidor para que traslade a la URI especificada, los datos incluidos en la solicitud.  
(Ej: Envío de datos de un formulario. El servidor pasará los datos a un proceso encargado de su utilización)
- **PUT:** solicitud de sustituir la URI especificada con los datos incluidos en dicha solicitud.
- **DELETE:** solicitud de borrar la URI especificada.

# Protocolo HTTP 1.1 (RFC 2616)

## CÓDIGOS DE RESPUESTA (para los *response messages* del servidor)

- **1xx** indican mensajes exclusivamente informativos
- **2xx** indican algún tipo de éxito
- **3xx** redirección al cliente a otra URL
- **4xx** indican un error
- **5xx** indican un error

## CABECERAS y CAMPOS (47 *request fields*, 49 *response fields*)

From: User-Agent:, Content-Type:, Content-Length:, ...

[http://en.wikipedia.org/wiki/List\\_of\\_HTTP\\_header\\_fields](http://en.wikipedia.org/wiki/List_of_HTTP_header_fields)

# Protocolo HTTP 1.1 (RFC 2616)

## CAMPOS DE CABECERA COMUNES PARA PETICIONES Y RESPUESTAS

- **Content-Type:** descripción MIME de la información contenida en este mensaje.
- **Content-Length:** longitud en bytes de los datos enviados, expresado en base decimal.
- **Content-Encoding:** formato de codificación de los datos enviados en este mensaje. Sirve, por ejemplo, para enviar datos comprimidos o encriptados.
- **Date:** fecha local de la operación. Las fechas deben incluir la zona horaria en que reside el sistema que genera la operación. Por ejemplo: Sunday, 12-Dec-96 12:21:22 GMT+01. No existe un formato único en las fechas.

# Protocolo HTTP 1.1 (RFC 2616)

## CAMPOS DE CABECERA SOLO PARA PETICIONES DEL CLIENTE

- **Accept:** campo opcional que contiene una lista de tipos MIME aceptados por el cliente.
- **Authorization:** clave de acceso que envía un cliente para acceder a un recurso de uso protegido o limitado. La información incluye el formato de autorización empleado, seguido de la clave de acceso propiamente dicha.
- **From:** campo opcional que contiene la dirección de correo electrónico del usuario del cliente Web que realiza el acceso.

# Protocolo HTTP 1.1 (RFC 2616)

## CAMPOS DE CABECERA SOLO PARA PETICIONES DEL CLIENTE

- **If-Modified-Since:** permite realizar operaciones GET condicionales, en función de si la fecha de modificación del objeto requerido es anterior o posterior a la fecha proporcionada. Puede ser utilizada por los sistemas de almacenamiento temporal de páginas. Es equivalente a realizar un HEAD seguido de un GET normal.
- **Referer:** contiene la URL del documento desde donde se ha activado este enlace. De esta forma, un servidor puede informar al creador de ese documento de cambios o actualizaciones en los enlaces que contiene. No todos los clientes lo envían.
- **User-agent:** cadena que identifica el tipo y versión del cliente que realiza la petición. Por ejemplo, los browsers de Netscape envían cadenas del tipo User-Agent: Mozilla/3.0.

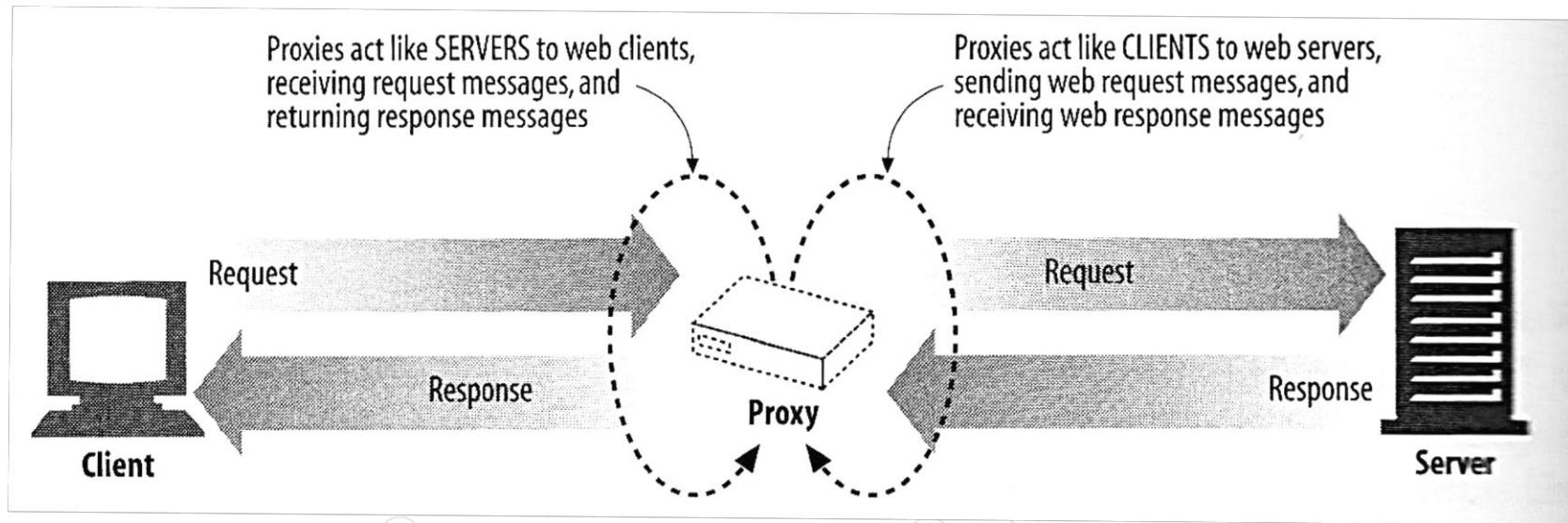
# Protocolo HTTP 1.1 (RFC 2616)

## CAMPOS DE CABECERA SOLO PARA RESPUESTAS DEL SERVIDOR

- **Allow:** informa de los comandos HTTP opcionales que se pueden aplicar sobre el objeto al que se refiere esta respuesta. Por ejemplo, Allow: GET, POST.
- **Expires:** fecha de expiración del objeto enviado. Los sistemas de cache deben descartar las posibles copias del objeto pasada esta fecha. Por ejemplo, Expires: Thu, 12 Jan 97 00:00:00 GMT+1. No todos los sistemas lo envían.
- **Last-modified:** fecha local de modificación del objeto devuelto. Se puede corresponder con la fecha de modificación de un fichero en disco, o, para información generada dinámicamente desde una base de datos, con la fecha de modificación del registro de datos correspondiente.

# Servidor Proxy

- Servidor que se sitúa entre el cliente y el servidor web, que hace papel de servidor de cara al cliente y de cliente de cara al servidor.



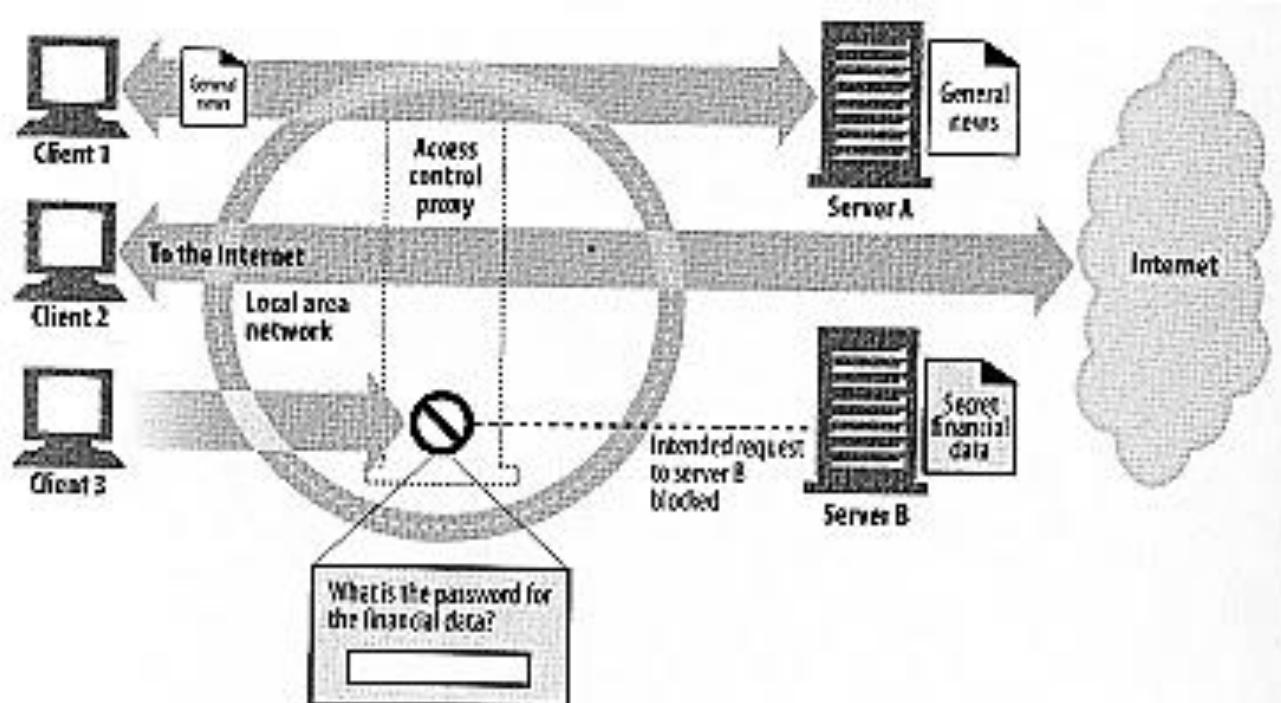
# Servidor Proxy

## VENTAJAS

- **Control:** Sólo el proxy hace la petición real al servidor, por tanto se pueden limitar y restringir los derechos de los usuarios y dar permisos sólo al proxy.
- **Ahorro:** Sólo uno de los usuarios (el proxy) ha de estar equipado para hacer el trabajo real.
- **Velocidad:** Si varios clientes van a pedir el mismo recurso, el proxy puede hacer caché (guardar la respuesta de una petición para darla directamente cuando otro usuario la pida). Así no tiene que volver a contactar con el servidor destino, y se resuelve antes la petición.
- **Filtrado:** El proxy puede negarse a responder algunas peticiones si detecta que están prohibidas.

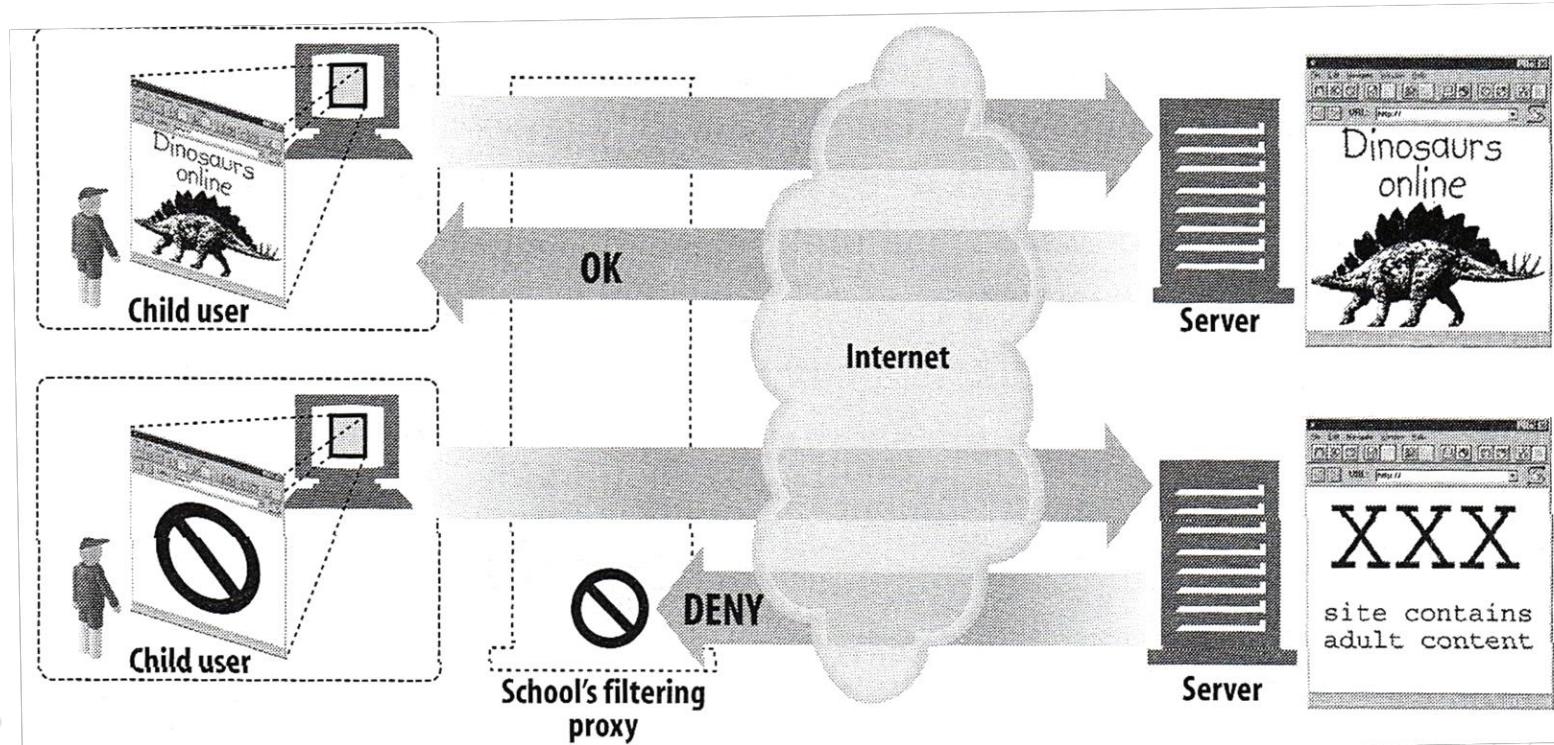
# Servidor Proxy

## EJEMPLO CONTROL DE ACCESO A DOCUMENTOS CENTRALIZADO



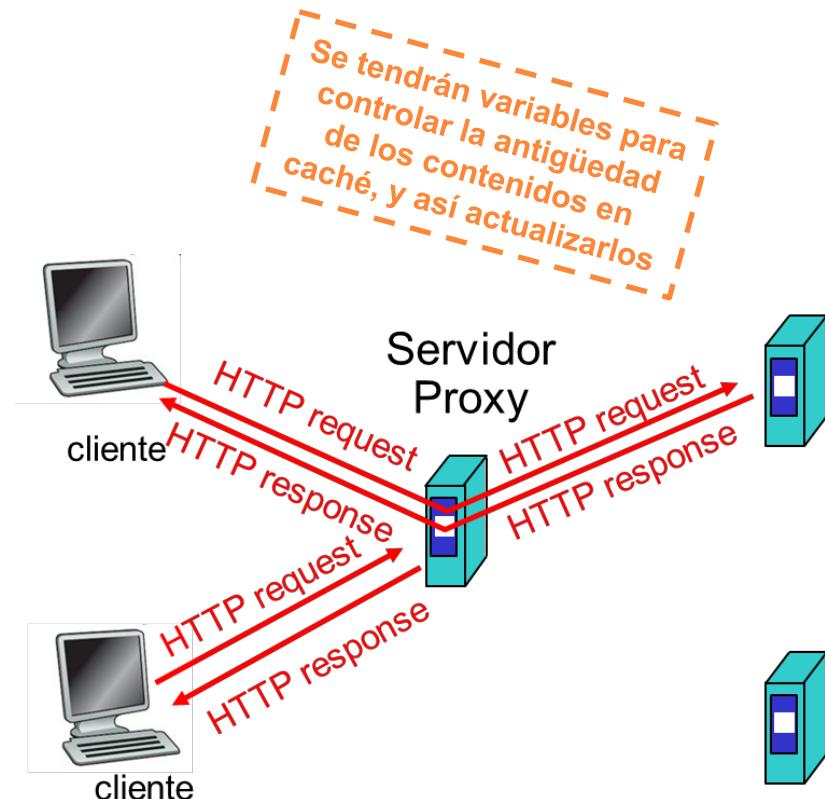
# Servidor Proxy

## EJEMPLO FILTRO DE PROTECCIÓN A MENORES



# Servidor Caché

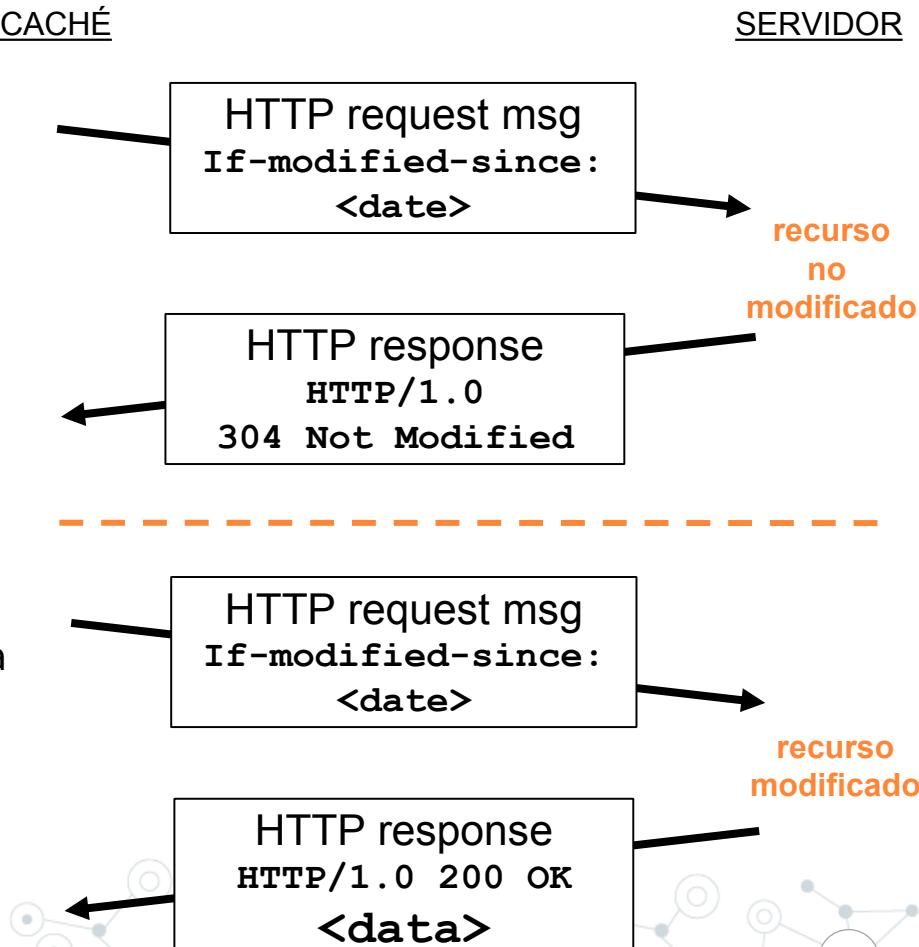
- Servidor **proxy** que **almacena las páginas visitadas** (recientemente) por uno o varios clientes.
- El **navegador/cliente** enviará todas las **peticiones HTTP al servidor caché**:
  - Si recurso está en la caché → se devuelve el objeto local
  - Si no → el servidor caché solicita el recurso al servidor destino, actualiza la caché con el mismo, y lo sirve al cliente
- La caché puede estar en el propio ordenador del usuario (cliente).



# Servidor Caché

- **Objetivo:** reducir el tráfico si la cache tiene una versión actualizada del objeto/recurso.
- La **cache solicita** el recurso condicionado a la fecha de la copia local, usando:  
If-modified-since: <date>
- Si el recurso es más reciente de esa fecha, el servidor lo envía.
- El servidor responde sin el recurso si la copia de la cache está actualizada:

HTTP/1.0 304 Not Modified



# Cookies (RFC 2109)

- Las **cookies** son pequeños **ficheros de texto** que se intercambian **clientes y servidores HTTP**, para solucionar una de las principales deficiencias del protocolo: la falta de información de estado entre dos transacciones (**stateless**). Fueron introducidas por Netscape.
- La primera vez que un **usuario accede** a un determinado **documento** de un **servidor**, éste **proporciona una cookie** que contiene datos que **relacionarán posteriores operaciones**.
- El **cliente almacena** la **cookie** en su sistema para usarla después. En los **futuros accesos** a este **servidor**, el navegador podrá **proporcionar** la **cookie original**, que servirá de **nexo entre este acceso** y los anteriores.
- Todo este proceso se realiza automáticamente, sin intervención del usuario.
- Usos habituales: compra electrónica, recuerdo de contraseñas, preferencias.



# Cookies (RFC 2109)

## USO DE LAS COOKIES

- Una **cookie** es simplemente una serie de **líneas de texto**, con **pares variable/valor**. Existe un conjunto predefinido de **nombres de variable**, necesarias para el correcto funcionamiento.
- Por ejemplo:
  - **Domain**: conjunto de direcciones Internet para el que es válida la cookie. Se puede dar una dirección única (www.mitienda.es) o un rango (.netscape.com).
  - **Path**: fija el subconjunto de URLs para las que sirve esta cookie.
  - **Version**: Permite seleccionar entre diferentes versiones del modelo de cookies.
  - **Expires**: Fecha de expiración de la información. Si no se incluye, los datos son descartados al finalizar la sesión con el cliente Web.

# Cookies (RFC 2109)

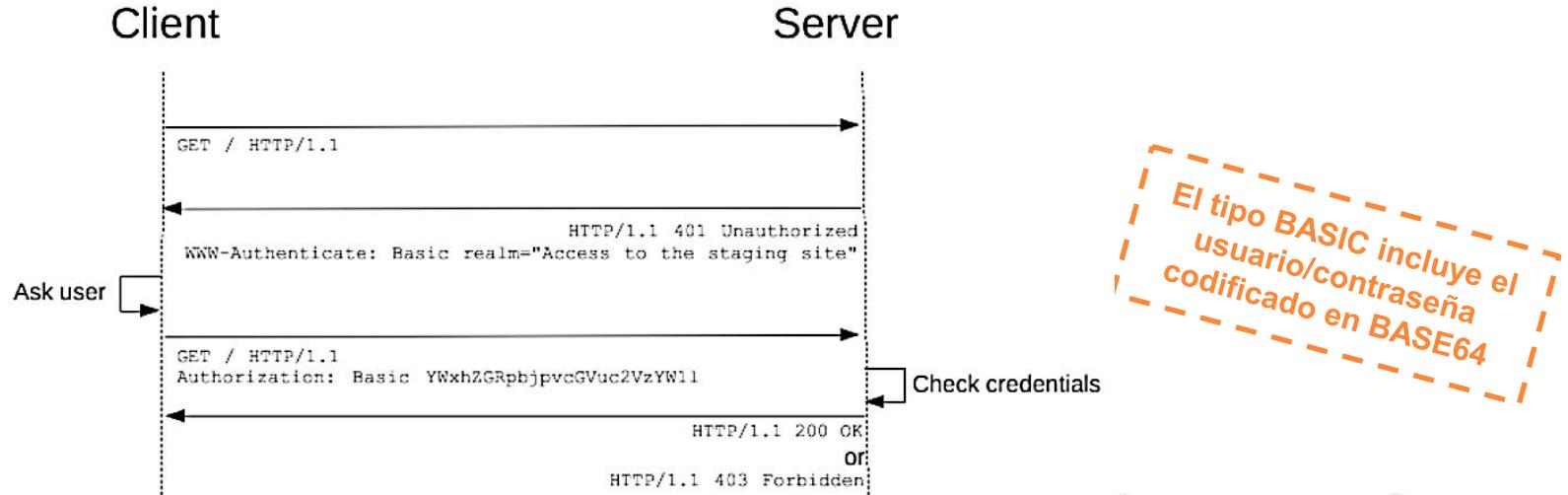
- Un **servidor HTTP envía** los diferentes campos de una **cookie** con la cabecera **HTTP Set-Cookie**: *Set-Cookie: Domain=www.unican.es; Path=/; Nombre=Luis; Expires Fri, 15-Jul-97 12:00:00 GMT*
- Cuando se **accede** a una **URL** que **verifica el par dominio/path registrado**, el **cliente enviará** automáticamente la **información** de los diferentes **campos** de la **cookie** con la cabecera **HTTP Cookie**: *Cookie: Domain=www.unican.es; Path=/; Nombre=Luis*



El servidor crea un ID que el cliente utilizará en conexiones posteriores

# Acceso restringido

- **HTTP no es seguro**, pero incluye cabeceras **WWW-Authenticate** (servidor) y **Authorization** (cliente) para **restringir el acceso** a recursos.
- HTTPS → Versión segura de HTTP. Encripta las transmisiones de peticiones y respuestas.



# TEMA 5. Capa de Aplicación

- 5.1. Introducción a las aplicaciones de red
- 5.2. Servicio de Nombres de Dominio (DNS)
- 5.3. Navegación web
- **5.4. Correo electrónico**
- 5.5. Aplicaciones multimedia
- 5.6. Cuestiones y ejercicios

# Introducción

- El **correo electrónico (e-mail)** es un **servicio de red** que permite a los usuarios **enviar y recibir mensajes y archivos** rápidamente, mediante sistemas de comunicación electrónicos.
- El correo electrónico nació a principios de los años 60. En este sistema inicial un usuario sólo era capaz de enviar mensajes a usuarios del mismo sistema.
- Una **dirección de correo electrónico** es un conjunto de **palabras** que **identifican** a una **persona** (únivamente) que puede enviar y recibir correo.
- Dicha dirección tiene un acceso restringido mediante un nombre de usuario y una contraseña.



# Introducción

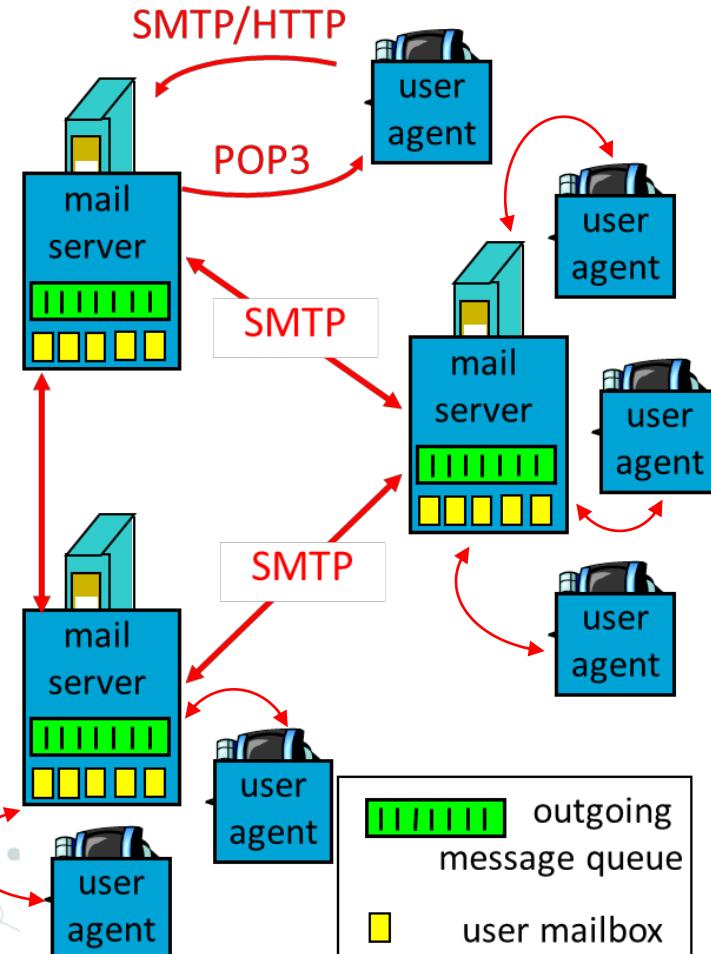
- El uso de la **arroba (@)** se incorporó en 1971, al ser un carácter que no existe en ningún nombre en el mundo.
- La **arroba divide** la dirección entre el **usuario** y el **dominio del proveedor de correo** (máquina en la que se aloja el correo).
- En inglés la arroba se lee “at” (en).
- El **nombre de usuario** (remitente/destinatario) puede incluir **letras, números** y algunos **signos**.
- La **dirección** la tiene que **proporcionar** un **proveedor de correo**, que son quienes **ofrecen el servicio** de envío y recepción.
- Es **indiferente** que las letras que integran la dirección estén escritas en **mayúscula o minúscula**.

# Introducción

- El **correo electrónico** se entrega usando una arquitectura **cliente/servidor**:
  - Un mensaje de correo electrónico se crea usando un programa de correo cliente.
  - Este programa envía el mensaje a un servidor.
  - El servidor lo redirige al servidor de correo del destinatario y allí se le suministra al cliente de correo del destinatario.
- Un e-mail **consta de**: destinatario, asunto y mensaje. Además puede tener ficheros adjuntos.
- Se pueden incluir **además los campos**:
  - CC (Copia de Carbón) → para incluir destinatarios en copia .
  - CCO (Copia de Carbón Oculta) → para incluir destinatarios no visibles por los demás.

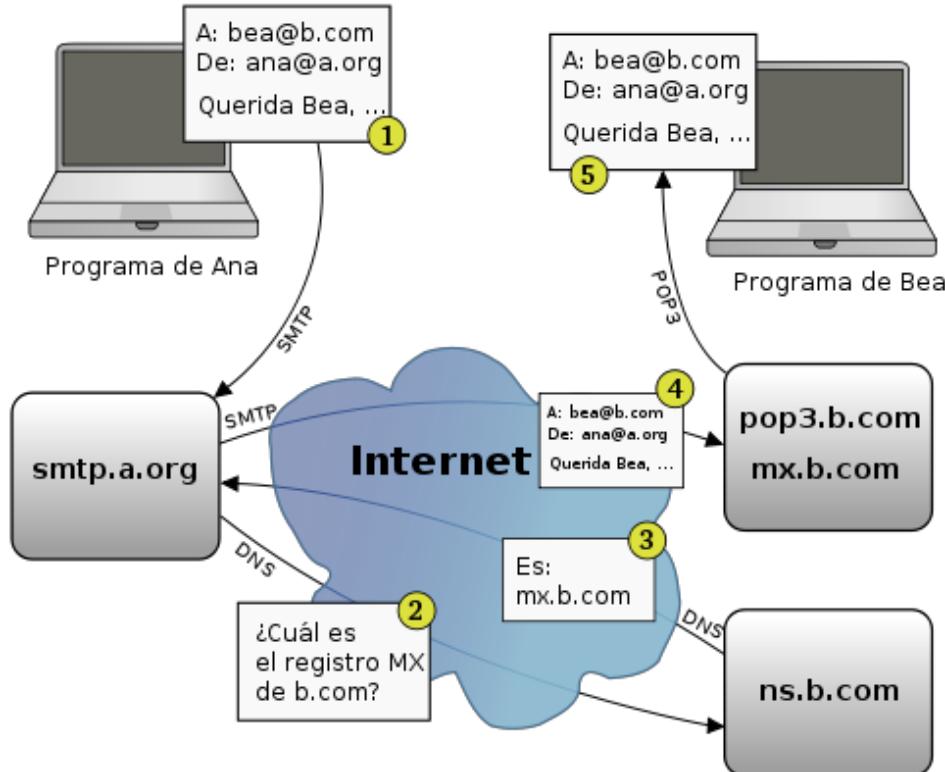
# Introducción

- **Elementos y protocolos principales:**
  - **Cliente de correo (Mail User Agent)**
  - **Servidor de correo (Mail Server o Mail Transfer Agent)**
  - **Protocolo de envío:**  
*Simple Mail Transfer Protocol (**SMTP**)*
  - **Protocolos de descarga** (o lectura):  
*POP3, IMAP, HTTP*
- **Agente de usuario (MUA):**  
Compone, edita y lee mensajes de correo del buzón.  
Ej: Outlook, Thunderbird, etc.
- **Servidor de correo (MTA):**  
Reenvía mensajes salientes y almacena en buzones los mensajes entrantes de cada usuario. Permite desacoplar temporalmente a remitente y destinatario



# Introducción

- Es un **servicio** que **hace uso de DNS**:



MX es el servidor de correo asociado a un dominio en DNS

# SMTP (RFC 5321)

- **Simple Mail Transfer Protocol.**
- Protocolo **cliente/servidor** sencillo.
- Funciona mediante **TCP** a través del **puerto 25** (en el servidor):
  - Handshaking (“saludo”)
  - Transferencia de mensajes
  - Cierre
- **SMTP es un protocolo:**
  - Orientado a texto.
  - Orientado a conexión
  - Statefull
- La interacción entre cliente SMTP y servidor SMTP se realiza mediante comandos/respuesta:
  - Comandos → texto ASCII
  - Respuestas → código de estado y frases explicativas

Inicialmente los mensajes se codificaban con ASCII de 7 bits!!!

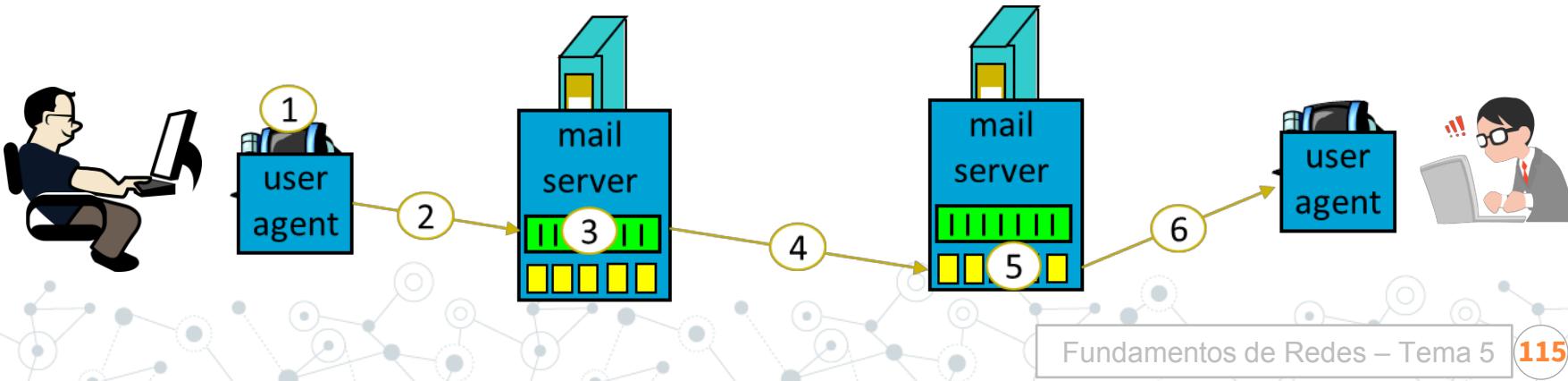
Posteriormente con MIME se pueden enviar ASCII de 8 bits y formatos enriquecidos

# SMTP (RFC 5321)

## PASOS EN EL ENVÍO Y RECEPCIÓN DE CORREO

- 1) El usuario origen compone mediante su Agente de Usuario (MUA) un mensaje dirigido a la dirección de correo del usuario destino.
- 2) Se envía con SMTP (ó HTTP) el mensaje al servidor de correo (MTA) del usuario origen que lo sitúa en la cola de mensajes salientes.
- 3) El cliente SMTP (del servidor del remitente) abre una conexión TCP con el servidor de correo (MTA) (obtenido por DNS) del usuario destino.

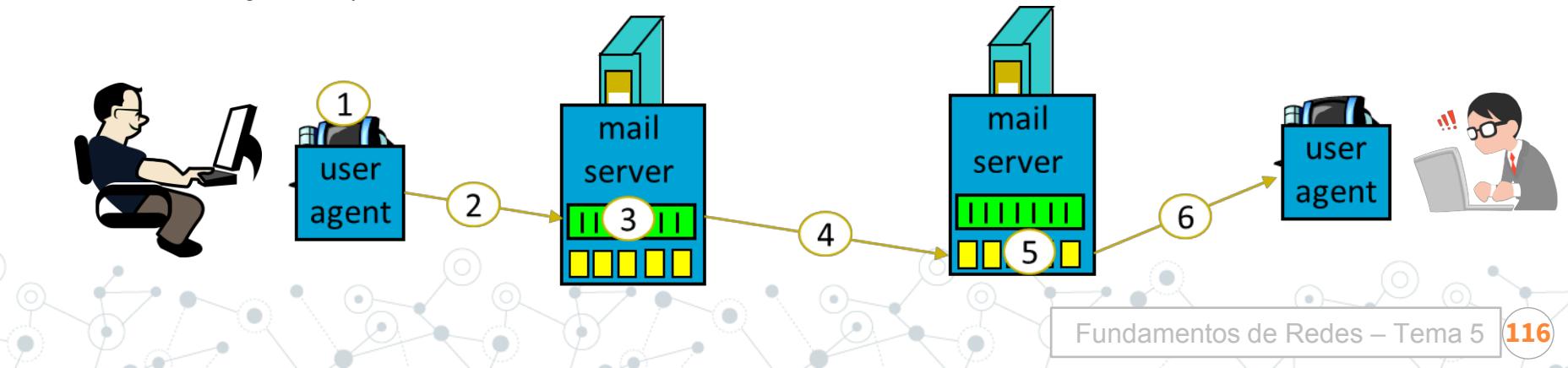
HTTP si se usa  
Webmail



# SMTP (RFC 5321)

## PASOS EN EL ENVÍO Y RECEPCIÓN DE CORREO

- 4) El cliente SMTP (del servidor del remitente) envía el mensaje sobre la conexión TCP al servidor de destino.
- 5) El servidor de correo del usuario destino ubica el mensaje en el *mailbox* del usuario destino.
- 6) El usuario destino invoca su Agente de Usuario (MUA) para leer el mensaje utilizando POP3, IMAP ó HTTP.



# SMTP (RFC 5321)

## COMANDOS SMTP (Cliente)

<u>Comando</u>	<u>Descripción</u>
<b>HELO (ahora EHLO)</b>	Identifica el remitente al destinatario.
<b>MAIL FROM</b>	Identifica una transacción de correo e identifica al emisor.
<b>RCPT TO</b>	Se utiliza para <b>identificar un destinatario individual</b> . Si se necesita identificar múltiples destinatarios es necesario repetir el comando.
<b>DATA</b>	Permite enviar una serie de líneas de texto. El tamaño máximo de una línea es de 1.000 caracteres. Cada línea va seguida de un retorno de carro y avance de línea <CR><LF>. <b>La última línea debe llevar únicamente el carácter punto "."</b> seguido de <CR><LF>.
<b>RSET</b>	Aborta la transacción de correo actual.
<b>NOOP</b>	No operación. <b>Indica al extremo que envíe una respuesta positiva. Keepalives</b>
<b>QUIT</b>	Pide al otro extremo que envíe una respuesta positiva y cierre la conexión.
<b>VRFY</b>	Pide al receptor que confirme que un nombre identifica a un destinatario valido.
<b>EXPN</b>	Pide al receptor la <b>confirmación de una lista de correo</b> y que devuelva los nombres de los usuarios de dicha lista.
<b>HELP</b>	Pide al otro extremo información sobre los comandos disponibles.
<b>TURN</b>	El emisor pide que se <b>inviertan los papeles</b> , para poder actuar como receptor. El receptor puede negarse a dicha petición.
<b>SOML</b>	Si el destinatario está conectado, entrega el mensaje directamente al terminal, en caso contrario lo entrega como correo convencional.
<b>SAML</b>	Entrega del mensaje en el buzón del destinatario. En caso de estar conectado también lo hace al terminal.
<b>SEND</b>	Si el destinatario está conectado, entrega el mensaje directamente al terminal.

# SMTP (RFC 5321)

## RESPUESTAS SMTP (Servidor)

<u>Código</u>	<u>Descripción</u>
211	Estado del sistema.
214	Mensaje de ayuda.
<b>220</b>	<b>Servicio preparado.</b>
<b>221</b>	<b>Servicio cerrando el canal de transmisión.</b>
<b>250</b>	<b>Solicitud completada con éxito.</b>
251	Usuario no local, se enviará a <dirección de reenvío>
<b>354</b>	<b>Introduzca el texto, finalice con &lt;CR&gt;&lt;LF&gt;.&lt;CR&gt;&lt;LF&gt;.</b>
421	Servicio no disponible.
450	Solicitud de correo no ejecutada, servicio no disponible (buzón ocupado).
451	Acción no ejecutada, error local de procesamiento.
452	Acción no ejecutada, insuficiente espacio de almacenamiento en el sistema.
500	Error de sintaxis, comando no reconocido.
<b>501</b>	<b>Error de sintaxis. P.ej contestación de SMTP a ESMTP</b>
502	Comando no implementado.
503	Secuencia de comandos errónea.
504	Parámetro no implementado.
550	Solicitud no ejecutada, buzón no disponible.
<b>551</b>	<b>Usuario no local, pruebe &lt;dirección de reenvío&gt;. Si no se tiene cuenta</b>
552	Acción de correo solicitada abortada.
553	Solicitud no realizada (error de sintaxis).
554	Fallo en la transacción.

# SMTP (RFC 5321)

## EJEMPLO DE COMANDOS Y RESPUESTAS SMTP

```
S: 220 smtp1.ugr.es
C: EHLO ugr.es
S: 250 smtp1.ugr.es
C: MAIL FROM: uno@ugr.es
S: 250 Ok
C: RCPT TO: dos@ugr.es
S: 250 Ok
C: DATA
S: 354 End data with <CR><LF>.<CR><LF>
C: Subject: Correo estúpido
C: Tengo ganas de enviarte un correo...
C: ¿Te importa si lo hago?
C: .
S: 250 Ok: queued as KJSADHFFWDF
C: QUIT
S: 221 Bye
```

EHLO ⇔ HELLO

# SMTP (RFC 5321)

## CAPTURA WIRESHARK

cliente -> serv\_correo TCP 3612 > 25 [SYN] Seq=0 Win=64512 Len=0 MSS=1460  
 serv\_correo -> cliente TCP 25 > 3612 [SYN, ACK] Seq=0 Ack=1 win=65535 Len=0 MSS=1460  
 cliente -> serv\_correo TCP 3612 > 25 [ACK] Seq=1 Ack=1 win=64512 Len=0  
 serv\_correo -> cliente SMTP Response: 220 SERV-CORREO Spectrum Server 1.0 ESMTP ready  
 cliente -> serv\_correo SMTP Command: EHLO alfon  
 serv\_correo -> cliente SMTP Response: 250-SERV-CORREO  
 cliente -> serv\_correo SMTP Command: AUTH LOGIN  
 serv\_correo -> cliente SMTP Response: 334 VXNlcm5hbwu6  
 cliente -> serv\_correo SMTP Command: XXXzdGVtYXN4xxxhbmn1LmVz  
 serv\_correo -> cliente SMTP Response: 334 UGF2c3dvcmQ6  
 cliente -> serv\_correo SMTP Command: XXXETUFGXXXW3Q==  
 serv\_correo -> cliente TCP 25 > 3612 [ACK] Seq=243 Ack=71 win=65465 Len=0  
 serv\_correo -> cliente SMTP Response: 235 2.0.0 Authentication successful  
 cliente -> serv\_correo SMTP Command: MAIL FROM: <alfon@miservidor.com>  
 serv\_correo -> cliente SMTP Response: 250 2.1.0 Sender <alfon@miservidor.com> ok  
 cliente -> serv\_correo SMTP Command: RCPT TO: <destinatario@otroservidor.com>  
 serv\_correo -> cliente SMTP Response: 250 2.1.5 Recipient <alfon@miservidor.com> ok (local)  
 cliente -> serv\_correo SMTP Command: DATA  
 serv\_correo -> cliente SMTP Response: 354 Enter mail, end with CRLF.CRLF  
 cliente -> serv\_correo SMTP DATA fragment, 1460 bytes  
 cliente -> serv\_correo SMTP DATA fragment, 740 bytes  
 serv\_correo -> cliente TCP 25 > 3612 [ACK] Seq=411 Ack=3061 win=65535 Len=0  
 cliente -> serv\_correo SMTP EOM:  
 serv\_correo -> cliente TCP 25 > 3612 [ACK] Seq=411 Ack=4521 win=65535 Len=0  
 serv\_correo -> cliente SMTP Response: 250 2.0.0 48456cd9-0001e6ea Message accepted for delivery  
 cliente -> serv\_correo SMTP Command: QUIT  
 serv\_correo -> cliente SMTP Response: 221 2.0.0 SMTP closing connection  
 serv\_correo -> cliente TCP 25 > 3612 [FIN, ACK] Seq=505 Ack=6732 Win=65524 Len=0  
 cliente -> serv\_correo TCP 3612 > 25 [ACK] Seq=6732 Ack=506 win=64008 Len=0  
 cliente -> serv\_correo TCP 3612 > 25 [FIN, ACK] Seq=6732 Ack=506 win=64008 Len=0  
 serv\_correo -> cliente TCP 25 > 3612 [ACK] Seq=506 Ack=6733 win=65524 Len=0

TCP 3-way Handshake

Servidor: Preparado (220)

Cliente: Inicio de diálogo (EHLO)

Servidor: OK (250)

Srv: Solicita usuario y contraseña  
Cli: Se los envía (cifrados)

Srv: OK (235)

Cli: FROM, Srv: OK  
Cli: TO, Srv: OK

Mensaje

Fin mensaje

Cli: Salir  
Srv: OK

Fin conexión TCP

# MIME (Multipurpose Internet Mail Protocol Extensions)

- MIME está especificado los RFCs: 2045, 2046, 2047, 4288, 4289 y 2077.
- Nada cambia respecto a la arquitectura de correo anterior.
- Las extensiones de MIME van encaminadas a soportar:
  - Texto en conjuntos de caracteres distintos de US-ASCII.
  - Adjuntos que no son de tipo texto.
  - Cuerpos de mensajes con múltiples partes (multi-part).
  - Información de encabezados con conjuntos de caracteres distintos de ASCII.

# MIME (Multipurpose Internet Mail Protocol Extensions)

- **Cabeceras del mensaje MIME:**

Cabecera	Descripción
MIME-Version:	Identifica la versión de MIME. Si no existe se considera que el mensaje es texto normal en inglés.
Content-Description:	Cadena de texto que describe el contenido. Esta cadena es necesaria para que el destinatario sepa si desea descodificar y leer el mensaje o no.
Content-Id:	Identificador único, usa el mismo formato que la cabecera estándar Message-Id.
Content-Transfer-Encoding:	Indica la manera en que está envuelto el cuerpo del mensaje.
Content-Type:	Especifica la naturaleza del cuerpo del mensaje.

- **Content-Transfer Encoding:**

- Indica la manera en que está envuelto el cuerpo para su transmisión, ya que podría haber problemas con la mayoría de los caracteres distintos de letras, números y signos de puntuación.
- Existen 5 tipos de codificación (RFC1521) : *ASCII 7, ASCII 8, codificación binaria, base64 y entrecomillada-imprimible.7.2.*

# MIME (Multipurpose Internet Mail Protocol Extensions)

- **Content-Type:** La lista inicial de tipos y subtipos especificada por el RFC 1521 es:

<u>Tipo</u>	<u>Subtipo</u>	<u>Descripción</u>
Text	Plain	Texto sin formato.
	Richtext	Texto con comandos de formato sencillos.
Image	Gif	Imagen fija en formato GIF.
	Jpeg	Imagen fija en formato JPEG.
Audio	Basic	Sonido.
Video	Mpeg	Película en formato MPEG.
Application	Octet-stream	Secuencia de bytes no interpretada.
	Postscript	Documento imprimible PostScript.
Message	Rfc822	Mensaje MIME RFC 822.
	Partial	Mensaje dividido para su transmisión.
	External-body	El mensaje mismo debe obtenerse de la red.
Multipart	Mixed	Partes independientes en el orden especificado.
	Alternative	Mismo mensaje en diferentes formatos.
	Parallel	Las partes deben verse simultáneamente.
	Digest	Cada parte es un mensaje RFC 822 completo.

# MIME (Multipurpose Internet Mail Protocol Extensions)

- **Content-Type:** Tipo *Application*

- El tipo ***application*** es un tipo general para los formatos que requieren procesamiento externo no cubierto por ninguno de los otros tipos.
- El subtipo ***octet-stream*** simplemente es una secuencia de bytes no interpretados, tal que a su recepción, un agente de usuario debería *presentarla en la pantalla sugiriendo al usuario que se copie en un archivo y solicitando un nombre de archivo*.
- El subtipo ***postscript***, se refiere al lenguaje PostScript de Adobe Systems. Aunque un agente de usuario puede llamar a un intérprete PostScript externo para visualizarlo, hacerlo no está exento de riesgos al ser PostScript un lenguaje de programación completo.

# MIME (Multipurpose Internet Mail Protocol Extensions)

- **Content-Type:** Tipo *Message*

- El tipo ***message*** permite que un mensaje esté encapsulado por completo dentro de otro. Este esquema es útil para reenviar, correo electrónico.
- El subtipo ***rfc822*** se utiliza cuando se encapsula un mensaje RFC 822 completo en un mensaje exterior.
- El subtipo ***partial*** hace posible dividir un mensaje encapsulado en pedazos y enviarlos por separado. **Los parámetros hacen posible ensamblar correctamente todas las partes en el destino.** Ej: 1/3, 2/3, 3/3.
- El subtipo ***external-body*** puede usarse para mensajes muy grandes, por ejemplo películas de vídeo. En lugar de incluir el archivo mpeg en el mensaje, se da una dirección de FTP y el agente de usuario del receptor puede obtenerlo a través de la red cuando se requiera.

# MIME (Multipurpose Internet Mail Protocol Extensions)

- **Content-Type:** Tipo *Multipart*

- El tipo es ***multipart***, que permite que un mensaje contenga más de una parte, con el comienzo y el fin de cada parte claramente delimitados.
- El subtipo ***mixed*** permite que cada parte sea diferente.
- El subtipo ***alternative*** indica que cada parte contiene el mismo mensaje, pero expresado en un medio o codificación diferente.
- El subtipo ***parallel*** se usa cuando todas las partes deben “verse” simultáneamente, por ejemplo, en los canales de audio y vídeo de las películas.
- El subtipo ***digest*** se usa cuando se juntan muchos mensajes en un mensaje compuesto.

# POP3 (RFC 1939)

- **Post Office Protocol.**
- Es un protocolo **utilizado** para la **entrega de mensajes** al usuario **final**.
- Obtiene el **correo electrónico** del **buzón remoto** (en el servidor de correo) y **lo almacena** en la **máquina local del usuario** para su lectura posterior.
- Por defecto, **una vez transferido** el mensaje, éste **se borra automáticamente del servidor** de correo.
- La versión 3 (la actual) utiliza **TCP** sobre el **puerto 110**.
- Se basa en **comandos y respuestas** en texto ASCII, como SMTP.
- Tiene **comandos** para que un **cliente** establezca una sesión (USER y PASS), la termine (QUIT), obtenga mensajes (RETR) y los borre (DELE).

# POP3 (RFC 1939)

- POP3 **se inicia** cuando el usuario arranca **el gestor de correo**. Éste llama al servidor y **establece una conexión TCP con el agente de transferencia de mensajes** en el puerto 110.
- El protocolo **POP3 administra la autenticación** utilizando el nombre de usuario y la contraseña. **Aunque no es seguro** porque la información no va encriptada.
- Para añadir **seguridad a POP3**, es posible **utilizar** la encriptación **Secure Socket Layer (SSL)** para la autenticación del cliente y las sesiones de transferencias de datos.
- **POP3 bloquea las bandejas de entrada** durante el acceso, lo que significa que es imposible que dos usuarios accedan de manera simultánea a la misma bandeja de entrada.

# POP3 (RFC 1939)

- **Comandos POP3:**

Comando	Descripción
USER identification	Este comando permite la autenticación. Debe estar seguido del nombre de usuario, es decir, una cadena de caracteres que identifique al usuario en el servidor. El comando <i>USER</i> debe preceder al comando <i>PASS</i> .
PASS password	El comando <i>PASS</i> permite especificar la contraseña del usuario cuyo nombre ha sido especificado por un comando <i>USER</i> previo.
STAT	Información acerca de los mensajes del servidor
RETR	Número del mensaje que se va a recoger
DELE	Número del mensaje que se va a eliminar
LIST [msg]	Número del mensaje que se va a mostrar
NOOP	Permite mantener la conexión abierta en caso de inactividad
TOP <messageID> <n>	Comando que muestra <i>n</i> líneas del mensaje, cuyo número se da en el argumento. En el caso de una respuesta positiva del servidor, éste enviará de vuelta los encabezados del mensaje, después una línea en blanco y finalmente las primeras <i>n</i> líneas del mensaje.
UIDL [msg]	Solicitud al servidor para que envíe una línea que contenga información sobre el mensaje que eventualmente se dará en el argumento. Esta línea contiene una cadena de caracteres denominada <i>unique identifier listing</i> ( <i>lista de identificadores únicos</i> ) que permite identificar de manera única el mensaje en el servidor, independientemente de la sesión. El argumento opcional es un número relacionado con un mensaje existente en el servidor POP, es decir, un mensaje que no se ha borrado.
QUIT	El comando <i>QUIT</i> solicita la salida del servidor POP3. Lleva a la eliminación de todos los mensajes marcados como eliminados y envía el estado de esta acción.

# POP3 (RFC 1939)

- **Ejemplo POP3:**

## Fase de autorización

Comandos del cliente:

**user**: nombre de usuario

**pass**: contraseña

Respuestas del servidor

+OK

-ERR

## Fase de transacción, cliente:

**list**: lista mensajes por número

**retr**: obtiene mensajes por num.

**dele**: borra

**quit**

## Fase de actualización del servidor

(tras desconexión)

```
S: +OK POP3 server ready
C: user bob
S: +OK
C: pass hungry
S: +OK user successfully logged on
C: list
S: 1 498
S: 2 912
S: .
C: retr 1
S: <message 1 contents>
S: .
C: dele 1
C: retr 2
S: <message 1 contents>
S: .
C: dele 2
C: quit
S: +OK POP3 server signing off
```

# IMAP (RFC 2060)

- **Internet Message Access Protocol.**
- Es un protocolo **utilizado** para la **entrega de mensajes** al usuario **final, alternativo a POP3**.
- La idea en que se basa IMAP es que el **servidor de correo electrónico** mantenga un **depósito central** al que puede **accederse desde cualquier máquina**.
- **No copia el correo electrónico** en la máquina del usuario y **no lo borra del servidor**.
- IMAP supone que todo el **correo electrónico permanecerá en el servidor** de manera indefinida.
- La versión actual es IMAP4 (revisión en RFC 3501)

# IMAP (RFC 2060)

- Permite **organización en carpetas** (o buzones) en el lado del **servidor** (MTA).
- Para ello, **mantiene información** entre sesiones (asociando **flags a los mensajes**).
- Permite la **descarga de partes** de los mensajes.
- Posible **acceder con varios clientes** (POP también, pero en modo descargar y guardar).
- Para **seguridad adicional**, es posible utilizar la encriptación **SSL** para la autenticación de clientes y para las sesiones de transferencia de datos.
- Un proceso cliente IMAP se comunica con el proceso servidor IMAP identificado a través del número de **puerto 143 TCP** (IMAP 4).

# POP3 vs IMAP4

- Operación en línea o fuera de línea.
  - Con POP3 los clientes se conectan brevemente al servidor de correo (sólo para descargar).
  - Con IMAP4, los clientes permanecen conectados el tiempo que su interfaz permanezca activa y descargan los mensajes bajo demanda.
- Conexión única o múltiple.
  - POP3 supone que el cliente conectado es el único dueño de una cuenta de correo.
  - IMAP4 permite accesos simultáneos de múltiples clientes y proporciona ciertos mecanismos a los mismos para que se detecten los cambios hechos a un buzón de correo por otro cliente concurrentemente conectado.
- Recuperación de mensajes completos o parciales.
  - Casi todo el correo electrónico en Internet es transmitido en formato MIME. IMAP4 permite a los clientes obtener separadamente cualquier parte MIME individual, así como obtener porciones de las partes individuales o los mensajes completos.

IMAP es más rápido

# POP3 vs IMAP4

- Información de mensajes y estado del mensaje en el servidor.
  - POP3 elimina los mensajes del servidor.
  - IMAP4 mantiene los mensajes en el servidor. Además, mediante el uso de marcas/señales definidas en el protocolo IMAP4 de los clientes, se puede asignar y conocer el estado de un mensaje (si ha sido o no leído, respondido o eliminado). Estas señales se almacenan en el servidor, de manera que varios clientes conectados al mismo correo en diferente tiempo pueden detectar los cambios hechos por otros clientes.
- Búsqueda y recuperación selectiva de mensajes.
  - POP3 recupera todos los mensajes.
  - IMAP4 permite hacer búsquedas en el servidor para acceder sólo a mensajes determinados.
- Facilidad para incluir extensiones.
  - IMAP se diseñó para incorporar extensiones de manera relativamente sencilla. Por ejemplo IMAP IDLE permite que el servidor envíe una señal al cliente cuando haya nuevos correos, evitando que el cliente tenga que preguntar cada cierto tiempo.

# POP3 vs IMAP4

Características	POP3	IMAP4
RFC	RFC 1939	RFC 2060
Puerto TCP utilizado	110	143
Dónde se almacena el correo electrónico	PC del usuario	Servidor
Tiempo de conexión requerido	Poco	Mucho
Uso de recursos del servidor	Mínimo	Amplio
Quién mantiene los buzones	Usuario	ISP
Bueno para usuarios móviles	No	Si
Descargas parciales de mensajes	No	Si
Sencillo de implementar Soporte amplio	Si	No

# Puertos

- Los puertos utilizados para los servicios relacionados con Correo Electrónico son:
  - POP3 - puerto 110
  - IMAP - puerto 143
  - SMTP - puerto 25
  - HTTP - puerto 80
  - Secure SMTP (SSMTP) - puerto 465
  - Secure IMAP (IMAP4-SSL) - puerto 585
  - IMAP4 over SSL (IMAPS) - puerto 993
  - Secure POP3 (SSL-POP) - puerto 995

# TEMA 5. Capa de Aplicación

- 5.1. Introducción a las aplicaciones de red
- 5.2. Servicio de Nombres de Dominio (DNS)
- 5.3. Navegación web
- 5.4. Correo electrónico
- **5.5. Aplicaciones multimedia**
- **5.6. Cuestiones y ejercicios**

# Definiciones

- Para A. Bartolomé (1994):

“Los sistemas Multimedia, en el sentido que hoy se da al término, son básicamente sistemas interactivos con múltiples códigos”

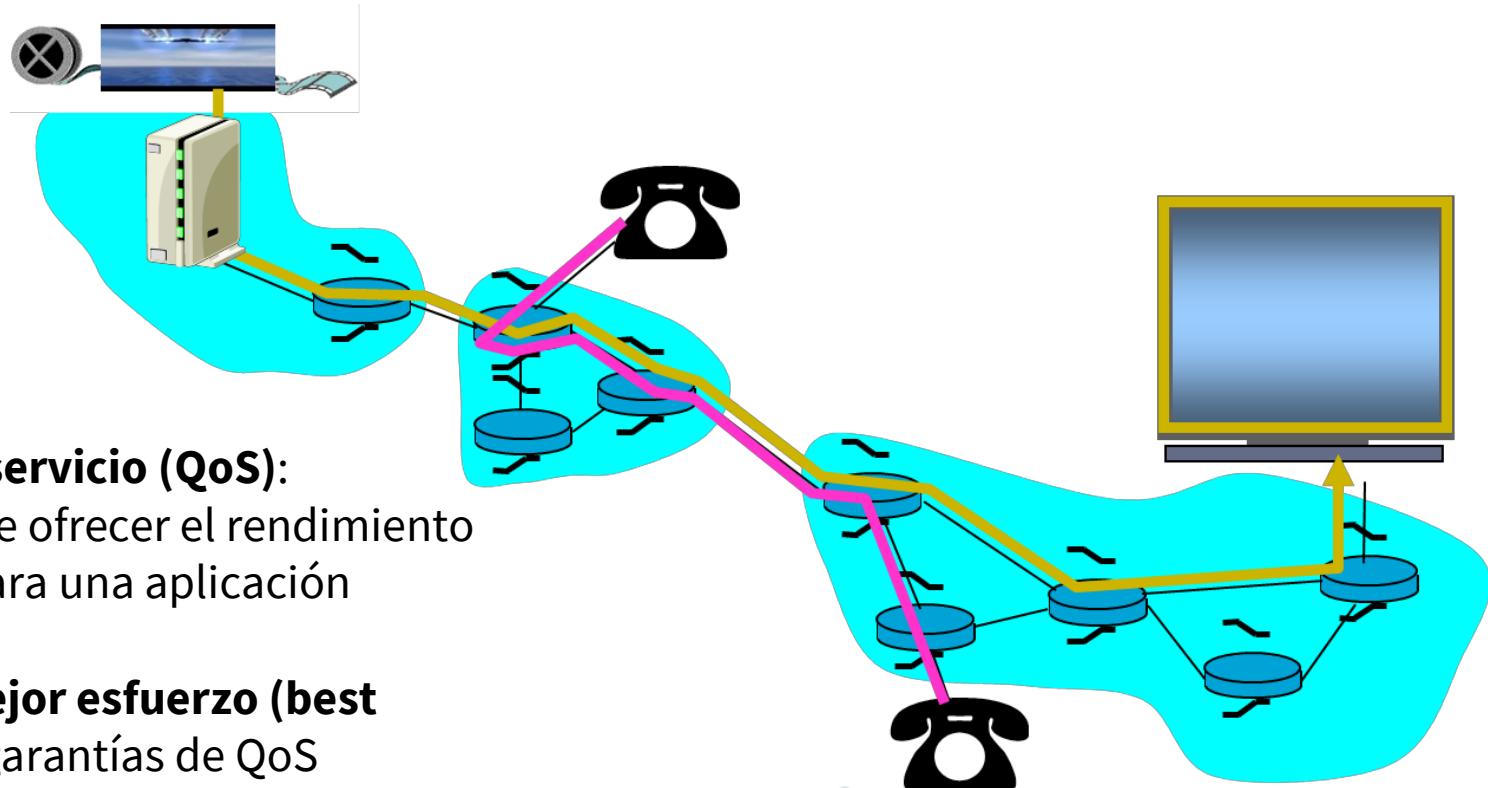
- Según Fred Hoffstetter:

“Multimedia es el uso del ordenador para presentar y combinar: texto, gráficos, audio y vídeo con enlaces que permitan al usuario navegar, interactuar, crear y comunicarse”.

# Concepto de aplicación multimedia

- El término **multimedia** hace referencia al **uso combinado** de **diferentes medios de comunicación**: texto, imagen, sonido, animación y video.
- Los **programas informáticos** que **utilizan de forma combinada** y coherente con sus objetivos **diferentes medios**, y **permiten la interacción** con el usuario son **aplicaciones multimedia interactivas**.
- La evolución producida en los sistemas de comunicación ha dado lugar a este tipo heterogéneo de aplicaciones o programas que **tienen dos características básicas**:
  - **Multimedia**: Uso de múltiples tipos de información (textos, gráficos, sonidos, animaciones, videos, etc.) integrados coheramente.
  - **Hipertexto**: Interactividad basada en los sistemas de hipertexto, que permiten decidir y seleccionar la tarea que deseamos realizar, rompiendo la estructura lineal de la información.

# Conceptos



# Aplicaciones multimedia en la red

## TIPOS DE APLICACIONES

- Flujo de audio y vídeo (*streaming*) almacenado. Ej: YouTube
- Flujo de audio y vídeo en vivo. Ej: emisoras de radio o IPTV
- Audio y vídeo interactivo. Ej: Skype

## CARACTERÍSTICAS PRINCIPALES

- Ocupan un elevado ancho de banda.
- Tolerantes relativamente a la pérdida de datos.
- Exigen retardo (*delay*) acotado.
- Exigen fluctuaciones del retardo (*jitter*) acotado.
- Se pueden beneficiar de usar de multicast (direcciones destino de grupo).

# Problemas habituales

Network Delivery Issues



Low Quality Source or Overly Aggressive Optimization



Stalling

Blurriness

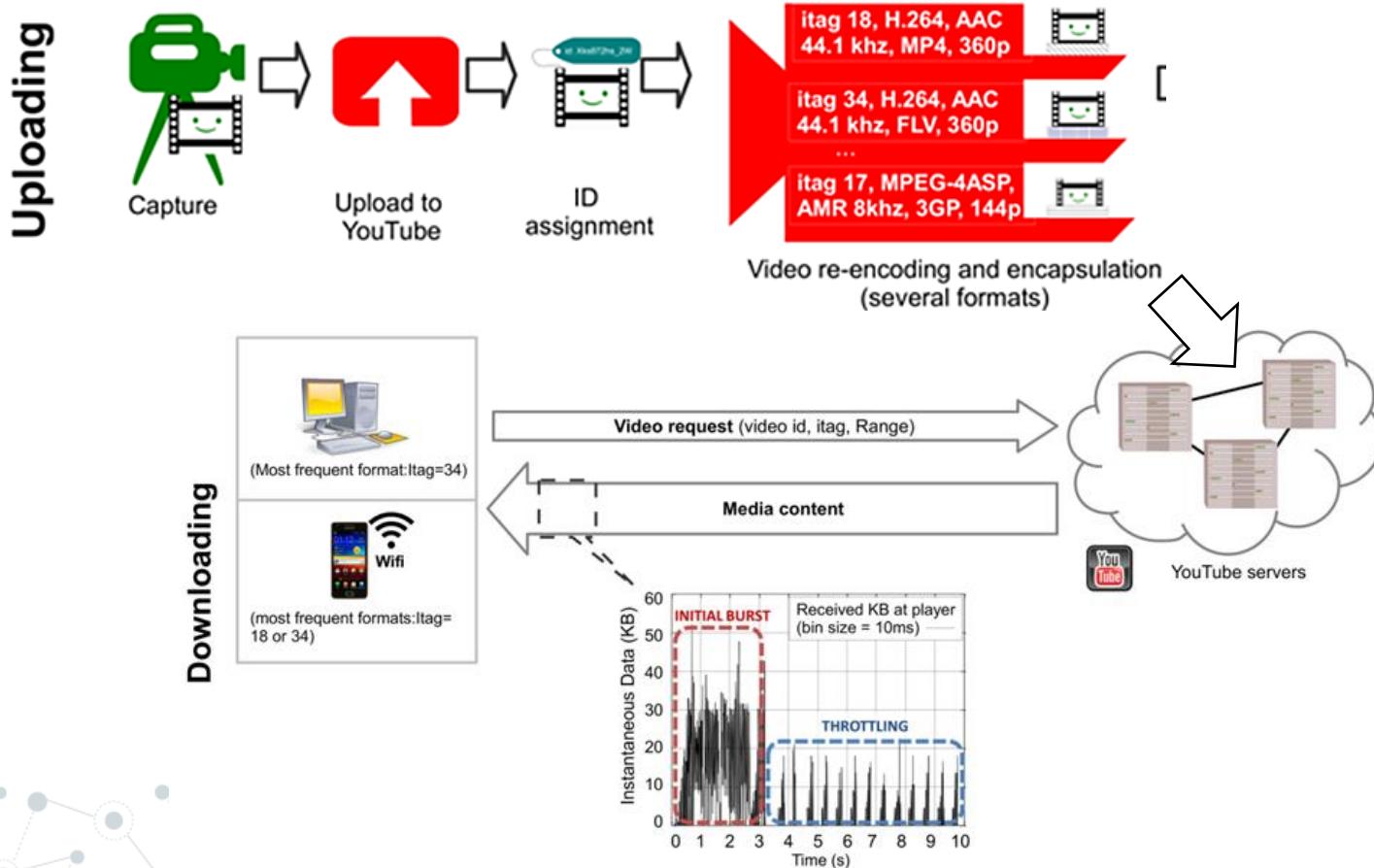


Macroblocking (loss of packets)

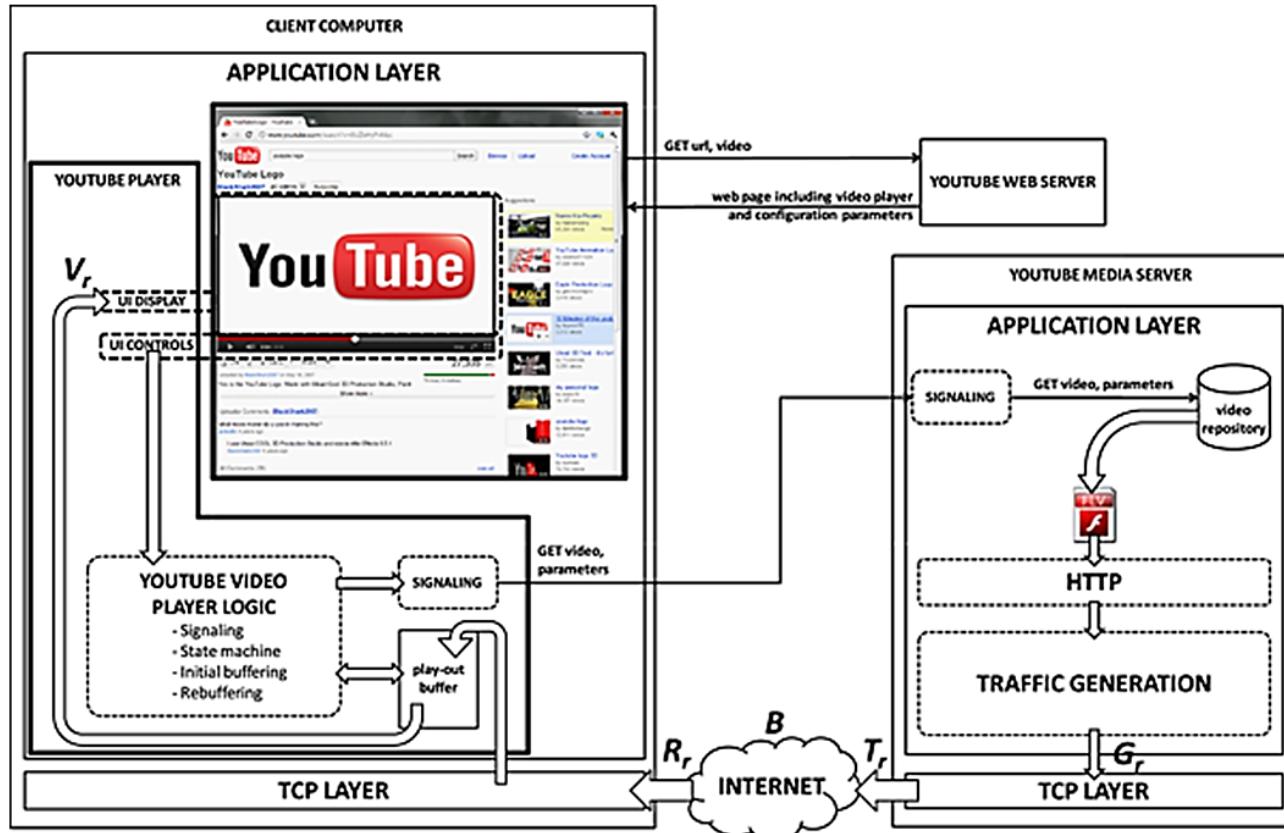


Blockiness

# Ejemplo: Youtube



# Ejemplo: Youtube



# TEMA 2. Servicios y Protocolos en Internet

- 5.1. Introducción a las aplicaciones de red
- 5.2. Servicio de Nombres de Dominio (DNS)
- 5.3. Navegación web
- 5.4. Correo electrónico
- 5.5. Aplicaciones multimedia
- **5.6. Cuestiones y ejercicios**

# Ejercicio

## EJERCICIO - RELACIÓN DE EJERCICIOS DEL TEMA 5

- Discuta las características de las siguientes aplicaciones en términos de su tolerancia a la pérdida de datos, los requisitos temporales, la necesidad de rendimiento mínimo y la seguridad.

Telefonía móvil

WhatsApp

YouTube

Spotify

Comercio electrónico

# Ejercicio

## EJERCICIO - RELACIÓN DE EJERCICIOS DEL TEMA 5

- ¿Es posible que un host tenga varias direcciones IP y un único nombre de dominio? Discuta un caso
- ¿Es posible que un host tenga varios nombres de dominio y una única dirección IP? Discuta un caso
- ¿Es posible que varios host tengan el mismo nombre de dominio, aunque direcciones IP distintas? Discuta un caso.

# Ejercicio

## EJERCICIO - RELACIÓN DE EJERCICIOS DEL TEMA 5

- Una sucursal con 50 empleados en Granada tiene una red interna basada en FastEthernet (100Mbps) que se conecta a Internet con una red de acceso ADSL de 0,5 Mbps de subida y 1,5 Mbps de bajada. Cada empleado, en el desempeño de su trabajo, realiza un promedio de 2000 solicitudes de información a la hora a un servidor de Base de Datos ubicado en la central del banco, en Madrid, donde cada solicitud supone el envío por parte del servidor de un promedio de 10 registros de 1KB cada uno. Adicionalmente, la modificación de datos tras algunas de estas solicitudes supone el envío promedio de 100 actualizaciones, de 10 registros de media, a la hora desde la sucursal al servidor. El resto de los servicios telemáticos se restringe.
- a) Calcule el promedio de la velocidad de transmisión requerida. ¿Es la velocidad del enlace de acceso suficiente?

# ¿Preguntas?

O comentarios, sugerencias, inquietudes