REDES DE COMPUTADORES

Tema 2: Nivel de Aplicación

2.1 Principios de las aplicaciones en red

2.2 DNS

2.3 Web y HTTP

2.4 Programación de la interfaz de acceso al servicio de transporte fiable de Internet en JAVA: Sockets con TCP

2.5 Programación de la interfaz de acceso al servicio de transporte no fiable de Internet en JAVA: Sockets con UDP

DTE
Departamento de
Tecnología Electrónica

Tema 2: Nivel de aplicación

Objetivos:

- Conceptos, aspectos de implementación de protocolos de aplicación en red
  - modelos de servicio del nivel de transporte
  - paradigma cliente-servidor
  - paradigma P2P
- Aprender más sobre protocolos estudiando dos protocolos de la capa de aplicación de internet
  - HTTP
  - DNS
- Programando aplicaciones en red
  - socket API

Algunas aplicaciones en red

- e-mail
- web
- mensajería instantánea
- Acceso remoto
- Compartición de archivos P2P
- Juegos online multiusuario
- Streaming de video (ej. YouTube)
- Voz sobre IP (VoIP)
- Videoconferencia
- Computación en la nube (cloud)
- …
- …
-

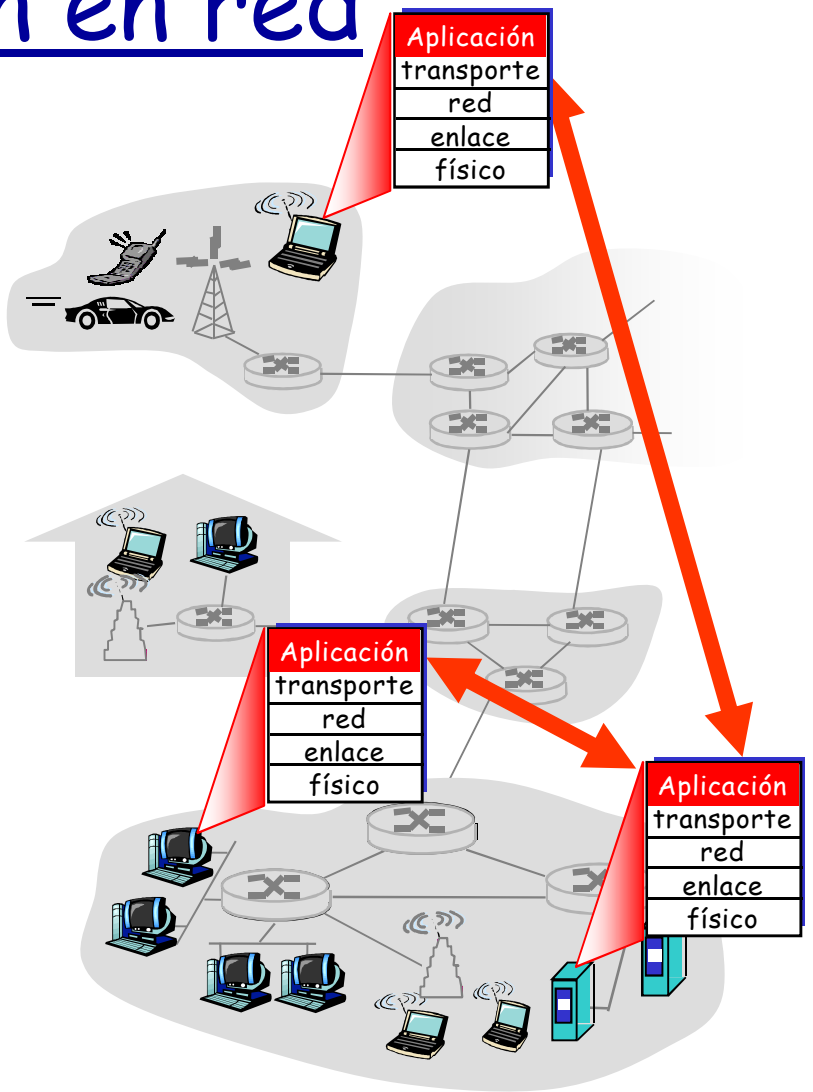
Creando una aplicación en red

escribir programas que

- Corran en (diferentes) *sistemas finales*
- Se comuniquen a través de la red
- Ej: software de un servidor web qué se comunica con el software navegador web

no hay que hacer programas para el núcleo de la red

- Los dispositivos del núcleo no corren las aplicaciones de los usuarios
- Al hacerlo en los sistemas finales se acelera el tiempo de desarrollo y propagación de la aplicación



Tema 2: Nivel de aplicación

2.1 Principios de las aplicaciones en red

2.2 DNS

2.3 Web y HTTP

2.4 Programación de la interfaz de acceso al servicio de transporte fiable de Internet en JAVA: Sockets con TCP

2.5 Programación de la interfaz de acceso al servicio de transporte no fiable de Internet en JAVA: Sockets con UDP

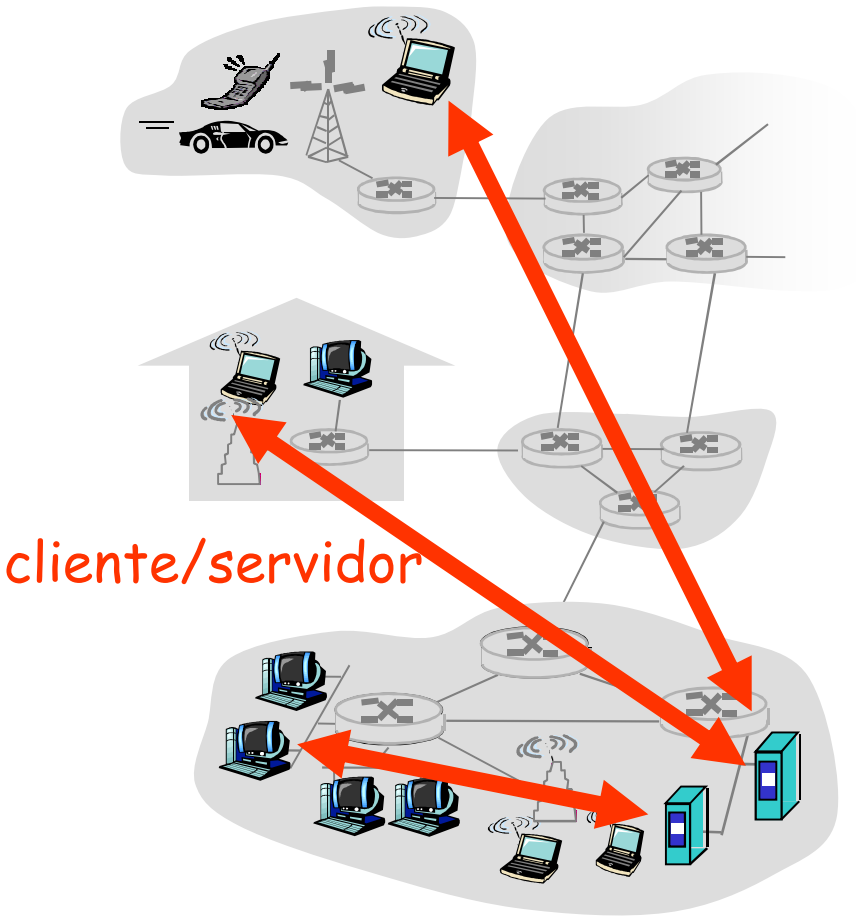
Arquitectura de las aplicaciones

- cliente-servidor
- peer-to-peer (P2P)
- híbrido ente cliente-servidor y P2P

Nota

Dirección IP: identifica de forma única a los equipos (sistemas finales, routers,...) conectados a una red TCP/IP. La asigna el ISP de forma estática (fija) o dinámica (variable). Más en breve...

Arquitectura Cliente-servidor



Servidor:

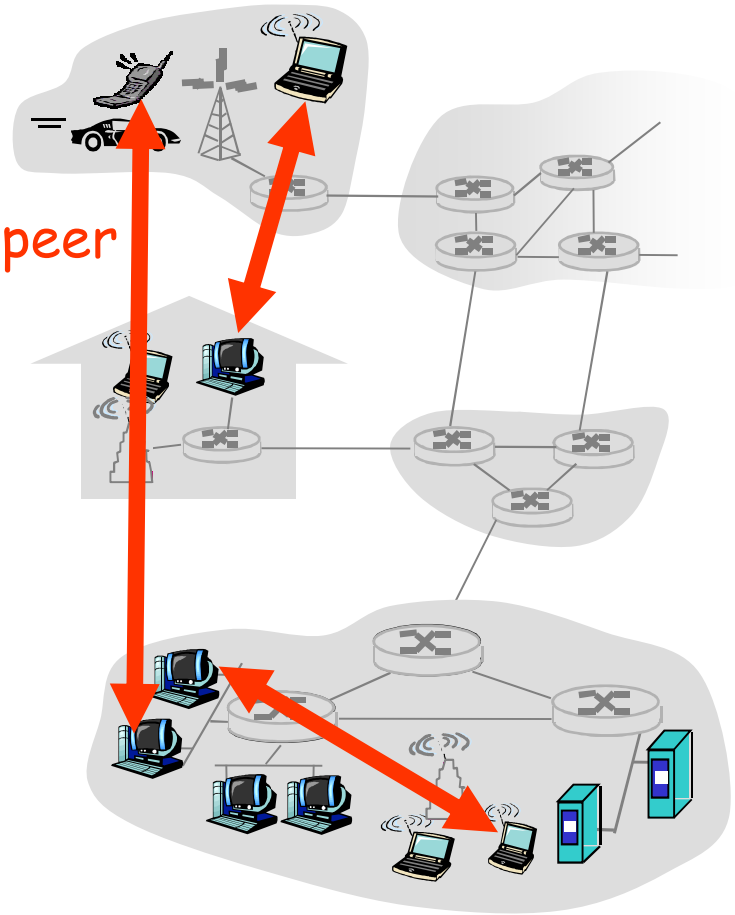
- Equipo siempre-ON
- Dirección IP fija
- Granjas de servidores por escalabilidad

Clientes:

- Se comunican con el servidor
- De manera intermitente
- Con IPs dinámicas o fijas
- No se comunican directamente entre ellos

Arquitectura P2P

- El servidor no está siempre-ON
- Los *sistemas finales* se comunican entre si de manera arbitraria
- Los peers se comunican de manera intermintente y con direcciones IP distintas en cada ocasión



muy escalable pero difícil de gestionar

Híbrido cliente-servidor y P2P

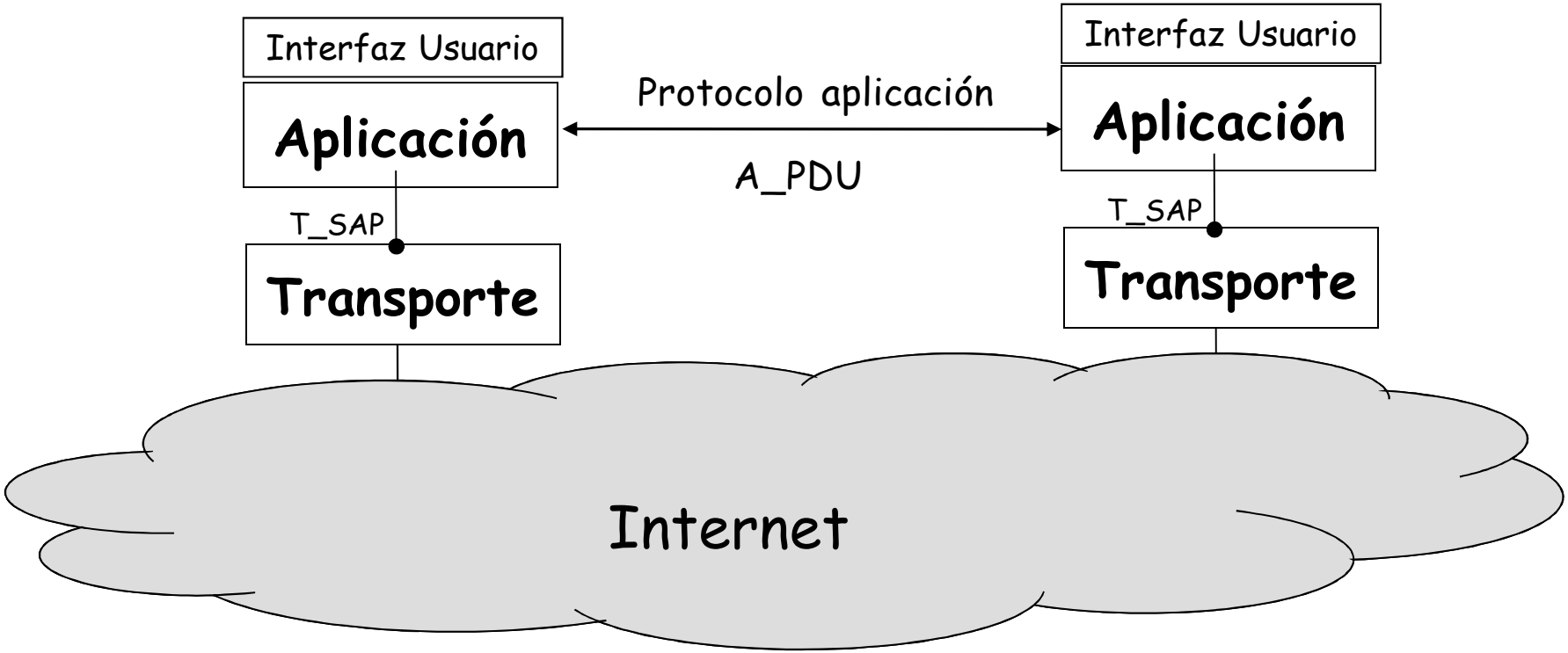
Skype

- Aplicación voz-sobre-IP arquitectura P2P
- Servidor centralizado: encontrar dirección IP del interlocutor remoto
- Conexión cliente-cliente: directa (sin pasar por el servidor)

Mensajería instantánea

- La charla entre 2 usuarios es P2P
- Servidor centralizado: detecta presencia y localización de los clientes
  - Los usuarios registran su IP con el servidor central al conectarse
  - Los usuarios dialogan con el servidor central en busca de la IP de su contacto

¿Cómo se implementa la capa de aplicación?



Navegadores Web, p.e: Mozilla firefox, Internet Explore, Safari,…

El protocolo de nivel de aplicación define:

- Tipo de mensaje a intercambiar,
  - e.g., petición, respuesta
- Sintaxis del mensaje
  - Número de campos y delimitación entre ellos
- Semántica del mensaje
  - Significado de los campos
- Reglas de cómo y cuándo los procesos envían y responden a los mensajes

Protocolos de dominio público:

- definidos en RFCs
- Permiten la inter-operatibilidad
- Ej: HTTP, SMTP

Protocolos propietarios:

- Ej: Skype

Comunicación entre procesos

proceso: programa que corre en un equipo (en nuestro caso implementa un determinado protocolo de aplicación).

- En un mismo equipo, 2 procesos se comunican usando comunicación entre-procesos (la proporciona SO).
- Procesos en equipos diferentes se comunican intercambiando mensajes (PDU) usando los servicios de comunicación (en general los proporciona SO)

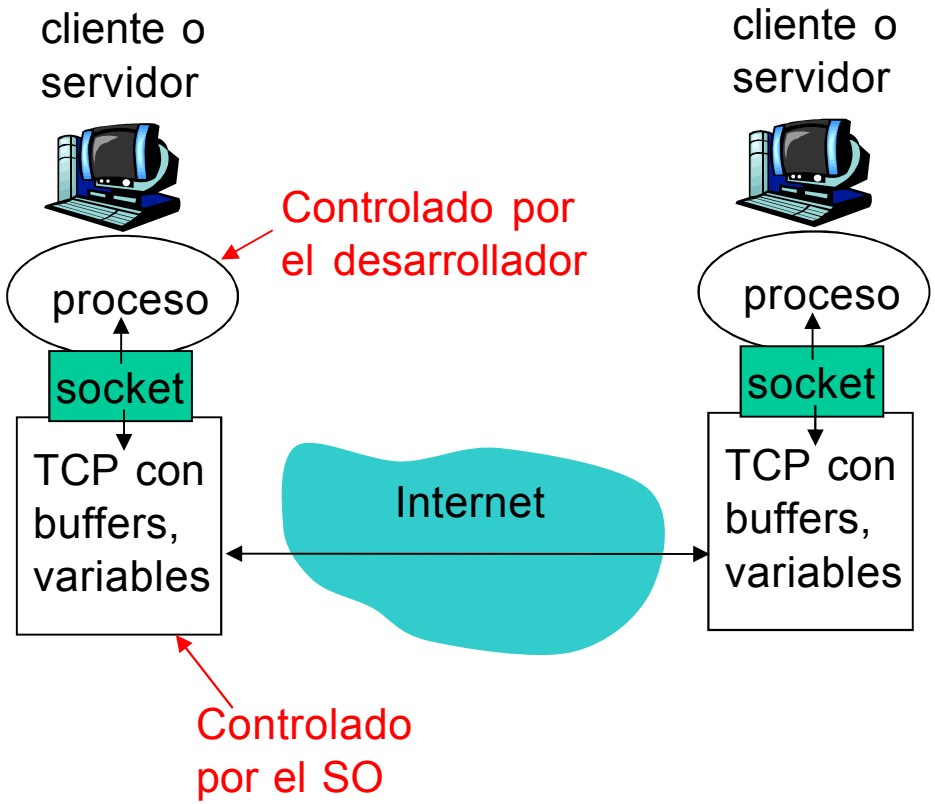
Proceso cliente: proceso que inicia la comunicación

Proceso servidor: proceso que espera a ser contactado

- nota: las aplicaciones P2P combinan ambos procesos, cliente y servidor

Sockets (SAP)

- Un proceso envía/recibe mensajes a/de su socket
- Analogía con una puerta:
  - El proceso emisor envía el mensaje a través de la puerta de salida
  - El proceso emisor confía en la infraestructura de transporte que hay detrás de la puerta, encargada de llevar el mensaje hasta la puerta del receptor



- API: (1) elección del servicio de transporte ; (2) posibilidad de fijar parámetros (a continuación...)

¿Cómo se identifica el socket?

- Para enviar una carta a un amigo es necesario saber su dirección para que llegue al buzón de su casa.
  - Cada sistema final tiene un dirección IP única de 32 bits.

Nota

A las direcciones IP se les asocia un nombre, que es el que se utiliza para identificar a los equipos.

Por ejemplo, www.dte.us.es = 150.214.141.196

Más sobre nombres en el apartado siguiente…

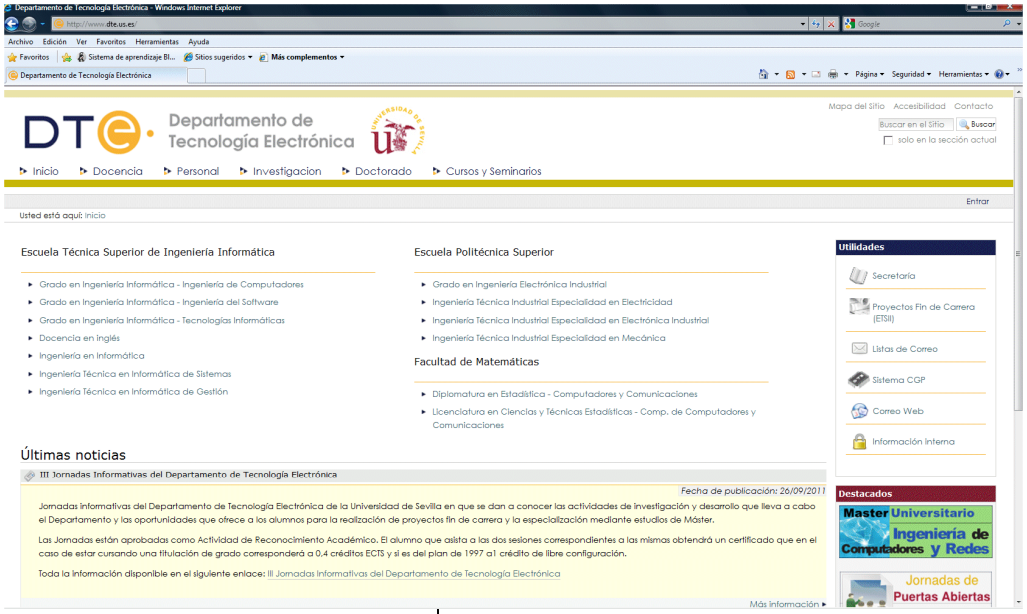
- *P:* ¿es suficiente con la dirección para hacer que llegue la carta a un amigo?
- *R:* No, *varias* personas pueden estar viviendo en el misma casa.
  - Varios protocolos de aplicación pueden estar ejecutándose en un sistema final.
    - Navegador, lector de correo, Skype, …

¿Cómo se identifica el socket?

- Cada protocolo de aplicación se identifica por un número de puerto.
- El número de puerto usado para identificar al proceso cliente y servidor en general no coinciden.
- Ej. de número de puerto:
  - Servidor HTTP: 80
  - Servidor Email: 25
  - Servidor DNS: 53
- *La ICANN (Internet Corporation for Assigned Names and Numbers), se encarga del registro de los puertos de protocolos de aplicación públicos.*

  http://www.iana.org/assignments/port-numbers
  - Existen diferentes tipos de puertos.
- Un socket queda identificado por:
  - Dirección IP.
  - Número de puerto.

Ejemplo



Servidor web DTE

150.214.141.196, 80

Dir IP cliente, puerto

Transporte

Transporte

Internet

Localhost: Conectando 2 procesos del mismo sistema final

- localhost: es un "nombre especial" que está asociado a una dirección IP especial que sirve para identificar al propio sistema final.
- Permite probar aplicaciones en red en un único sistema final sin necesidad de estar conectado a una red.
- En general permite comunicar procesos en un mismo sistema final usando los servicios de comunicaciones de Internet.

Nota

Localhost suele tener asociado la IP 127.0.0.1, aunque puede ser otra. Más en el tema 4…

P: ¿Por qué el Servicio de Comunicación del sistema final es capaz de distinguir a cada proceso?

¿Qué servicios de transporte necesito?

Perdida de datos

- Algunas aplicaciones toleran algo de perdida (ej: audio, video)
- Otras requieren 100% de fiabilidad (ej: login, transferencia de archivos)

Temporización

- Algunas aplicaciones precisan de retardos cortos para ser 'efectivas' (ej: telefonía por internet, juegos interactivos)

Tasa de transferencia

- Algunas requieren una tasa mínima para funcionar adecuadamente (ej: multimedia)
- Otras, conocidas como "aplicaciones elásticas", hacen uso de la tasa disponible en cada momento

Seguridad

- Encriptación, integridad de los datos, …

Requisitos de algunas aplicaciones comunes

Aplicación	Pérdida datos	Tasa transferencia	Sensible temp.
transferencia ficheros	sin pérdidas	elástica	no
e-mail	sin pérdidas	elástica	no
páginas web	sin pérdidas	elástica	no
audio/vídeo en tiempo real	tolerante	audio: 5kbps-1Mbps vídeo:10kbps-5Mbps	Sí, 100's ms
audio/vídeo archivado	tolerante	como la anterior	Sí, pocos segs
juegos interactivos	tolerante	varios kbps	Sí, 100's ms
mensajería instantánea	sin pérdidas	elástica	Sí y no

Servicios de los protocolos de Internet

Servicio TCP:

- *Orientado a conexión:* requiere acuerdo previo entre los procesos cliente y servidor antes de iniciar la transferencia
- *Transporte fiable* entre procesos emisor y receptor
- *Control de flujo:* emisor no saturará al receptor
- *Control de congestión:* uso equitativo del ancho de banda
- *No provee:* temporización, garantizar un ancho de banda, seguridad

Servicio UDP:

- Transporte ligero, no orientado a conexión y no confiable entre procesos emisor y receptor
- *No provee:* acuerdo previo entre procesos, fiabilidad, contol de flujo, control de congestión, temporización, ancho de banda garantizado, ni seguridad.

P: ¿Qué utilidad tiene UDP?

Ejemplos: Protocolos de aplicación y transporte

Aplicación	Protocolo del nivel de aplicación	Protocolo de transporte
e-mail	SMTP [RFC 2821]	TCP
acceso remoto	Telnet [RFC 854]	TCP
web	HTTP [RFC 2616]	TCP
transferencia de ficheros	FTP [RFC 959]	TCP
streaming multimedia	HTTP (ej: YouTube), RTP [RFC 1889]	TCP o UDP
telefonía IP	SIP, RTP, propietario (ej: Skype)	usualmente UDP
Traducción de nombres en direcciones IP	DNS [RFC 1034]	TCP o UDP (usual)

Tema 2: Nivel de aplicación

2.1 Principios de las aplicaciones en red

2.2 DNS

2.3 Web y HTTP

2.4 Programación de la interfaz de acceso al servicio de transporte fiable de Internet en JAVA: Sockets con TCP

2.5 Programación de la interfaz de acceso al servicio de transporte no fiable de Internet en JAVA: Sockets con UDP

DNS: Domain Name System

personas: muchos IDs:

- DNI, nombre, nº seguridad social...

equipos y routers de Internet:

- direcciones IP (32 bit) – sirven para direccionar datagramas
- "nombre", ej: www.google.com – usado por humanos

P: ¿cómo mapeamos entre direcciones IP y nombres y viceversa?

Domain Name System:

- *Base de datos distribuida* implementada con una jerarquía de servidores de nombres
- *Protocolo de nivel de aplicación:* equipos y servidores de nombres se comunican para resolver nombres (traducción de direcciones y de nombres)
  - nota: característica fundamental de Internet, ¡implementada en el nivel de aplicación!

DNS

Servicio DNS

- Traducción de nombre a IP (directa)
- Traducción de IP a nombre (inversa)
- Creación de "alias"
- Alias de email
  - @dominio
- Distribución de carga
  - Servidores web replicados: conjunto de Ips para un único nombre canónico

¿por qué no centralizar el servicio de DNS?

- Único punto de falla
- Volumen de tráfico
- Distancia a la base de datos
- Mantenimiento

en definitiva...

¡no sería escalable!

Usa UDP

- El cliente envía mensajes al puerto 53 del servidor a través del socket.

DNS: caché y actualización entradas

- Una vez que un servidor DNS aprende una traducción, ésta se guarda en una memoria *caché*
  - Las traducciones más habituales suelen estar en caché y no hace falta consultar a otros DNS.
  - Las entradas de la caché "caducan" tras un tiempo determinado (*timeout*)
- Hay mecanismos de actualización y notificación entre DNS propuestos por el IETF standard
  - RFC 2136

DNS: ¿cómo funciona? (simplificación)

- Cuando una aplicación en un sistema final consulta al servicio DNS la dirección IP asociada a un nombre de equipo, o viceversa. El servicio DNS busca en su "caché de DNS" para ver si tiene una entrada para la dirección IP o el nombre, según corresponda, puede ocurrir que…
  - … encuentre la entrada.
    - En este caso le devuelve a la aplicación la IP o el nombre.
  - … no encuentre la entrada.
    - Envía un mensaje (DNS_PDU) de "Solicitud de DNS" al servidor de DNS que tenga configurado y se espera a recibir la "Respuesta de DNS" del servidor con la información solicitada que entregará a la aplicación que solicitó su servicio.
      - En caso de que no sea posible resolver un nombre o una dirección IP avisa a la aplicación que solicitó su servicio

Nota

Un servidor de DNS al recibir una "Solicitud DNS" se comporta como el servicio de DNS en el sistema final, busca en su caché y consulta a otros servidores de DNS si no encuentra la información solicitada.

Tema 2: Nivel de aplicación

2.1 Principios de las aplicaciones en red

2.2 DNS

2.3 Web y HTTP

2.4 Programación de la interfaz de acceso al servicio de transporte fiable de Internet en JAVA: Sockets con TCP

2.5 Programación de la interfaz de acceso al servicio de transporte no fiable de Internet en JAVA: Sockets con UDP

WWW (World-Wide-Web)

Primero, un repaso…

- Una página web contiene una serie de objetos
- Esos objetos pueden ser: fichero HTML, imagen JPEG, applet Java, fichero audio,…
- Consiste en un fichero base HTML que incluye objetos referenciados
- Cada objeto es direccionable a través de su URL
- Ejemplo de URL (Uniform Resource Locator):

`www.informatica.us.es/index.php/organizacion-docente`

host name
(nombre del servidor donde está el objeto)

path name
(nombre de la ruta al objeto en el servidor)

Formato del lenguaje HTML

Sirve para elaborar las páginas web, desde 1991. Se usan elementos con etiquetas entre <>. Cada elemento suele tener 4 campos: una etiq. inicio (<html>) y una etiq. cierre (</html>), unos atributos (en la de inicio) y un contenido (entre ambas).

```
<!DOCTYPE html…>        define inicio documento (opcional)
<html>página</html>     define inicio/fin documento
<head>cabecera</head> el contenido de la cabecera
(información "no visible" al usuario, como título,
estilos, metainformación, etc…)
<body>cuerpo</body>     define el cuerpo, contiene:
de <h1> a <h6> encabezados
    <table>tabla</table>  crea una tabla filas/cols
    <a href="URL">enlace</a>  hipervínculo: al hacer click
      en "enlace" se solicita la página de URL.
    <img src="URL"/>  imagen referenciada, el navegador la
      carga a continuación desde URL para visualizarla.
```

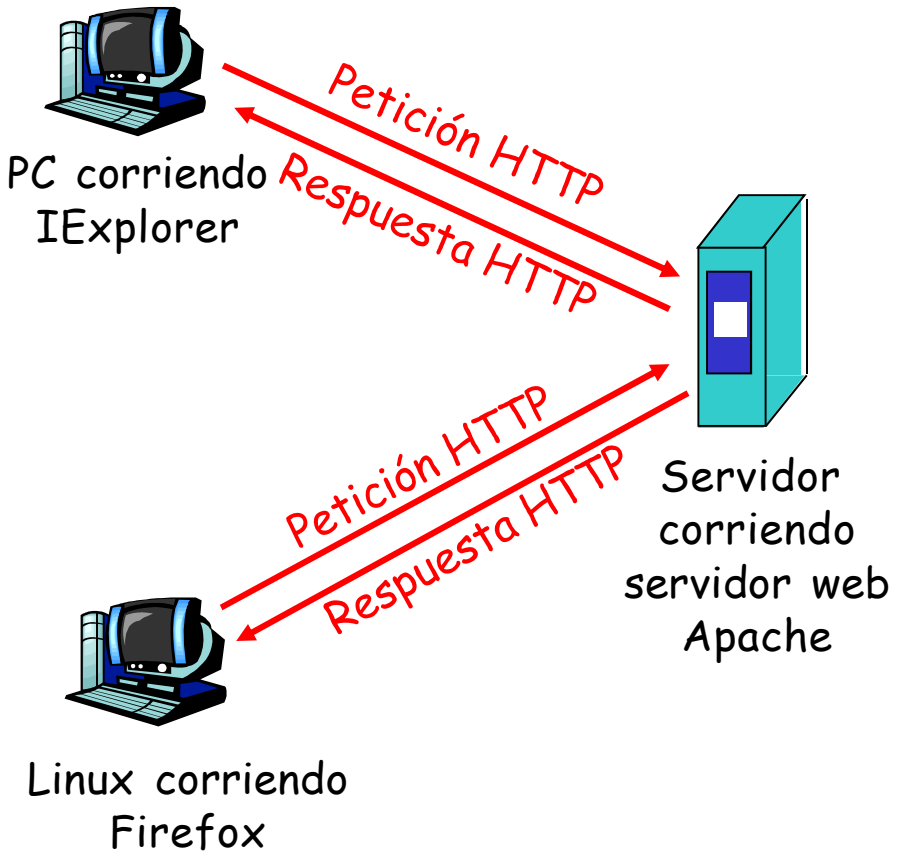
Nota

Se puede ver el código HTML de una página en el navegador, botón derecho -> ver código fuente

Vistazo de HTTP

HTTP: HyperText Transfer Protocol

- Protocolo de nivel de aplicación para la web
- Modelo cliente/servidor
  - *cliente:* navegador que pide, recibe y muestra los objetos web
  - *servidor:* proceso que envía los objetos pedidos por los clientes



Vistazo de HTTP (cont.)

Usa TCP:

- El cliente inicia una conexión TCP ("crea un socket"), con el puerto 80 del servidor
- El servidor acepta la conexión TCP del cliente
- Se intercambian mensajes HTTP (de nivel de aplicación) el navegador web (cliente) y el servidor web (servidor)
- Se cierra la conexión TCP

HTTP es "sin estado"

- El servidor no guarda información acerca de las peticiones anteriores de los clientes

Nota

Los protocolos que recuerdan el estado son "complejos":

- El histórico de estados anteriores se debe mantener
- Si el cliente o el servidor "caen" sus estados pueden ser inconsistentes y tienen que sincronizarse

Tipo de conexiones HTTP

HTTP no-persistente

- Cómo máximo se envía un objeto por cada conexión TCP.

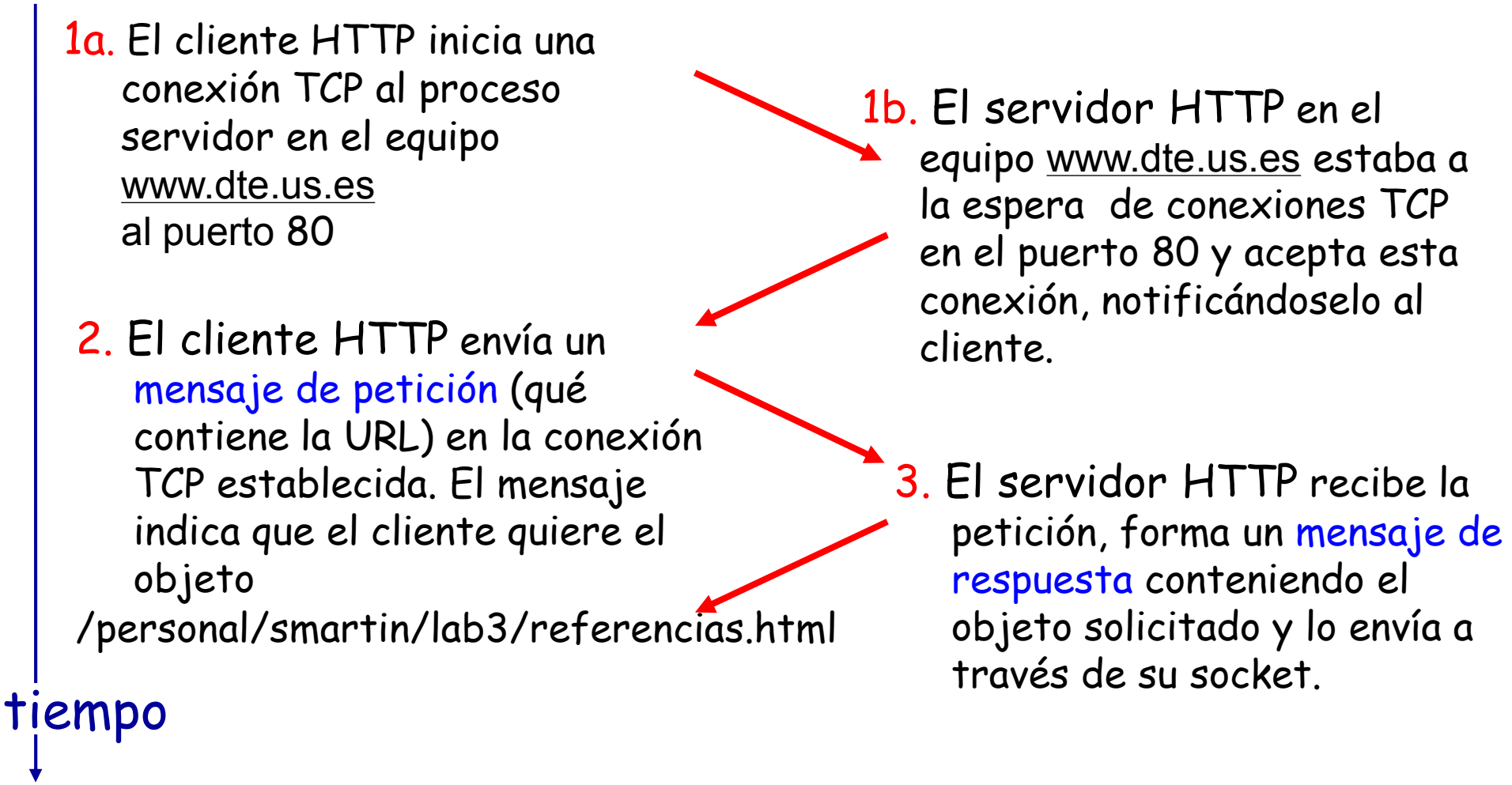
HTTP persistente

- Se pueden enviar multiples objetos por una misma conexión TCP entre cliente y servidor.

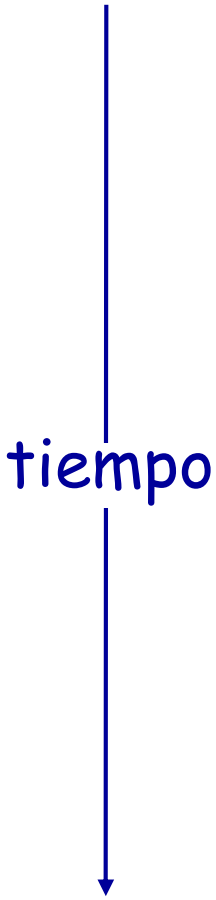
HTTP No-persistente

Supongamos que un usuario introduce esta URL: (contiene texto y referencias a 13 objetos)

http://www.dte.us.es/personal/smartin/lab3/referencias.html



HTTP No-persistente (cont.)



4. El servidor HTTP cierra la conexión TCP.

5. El cliente HTTP recibe el mensaje de respuesta, conteniendo el fichero HTML, muestra el contenido y lo analiza encontrando 13 referencias a otros objetos.

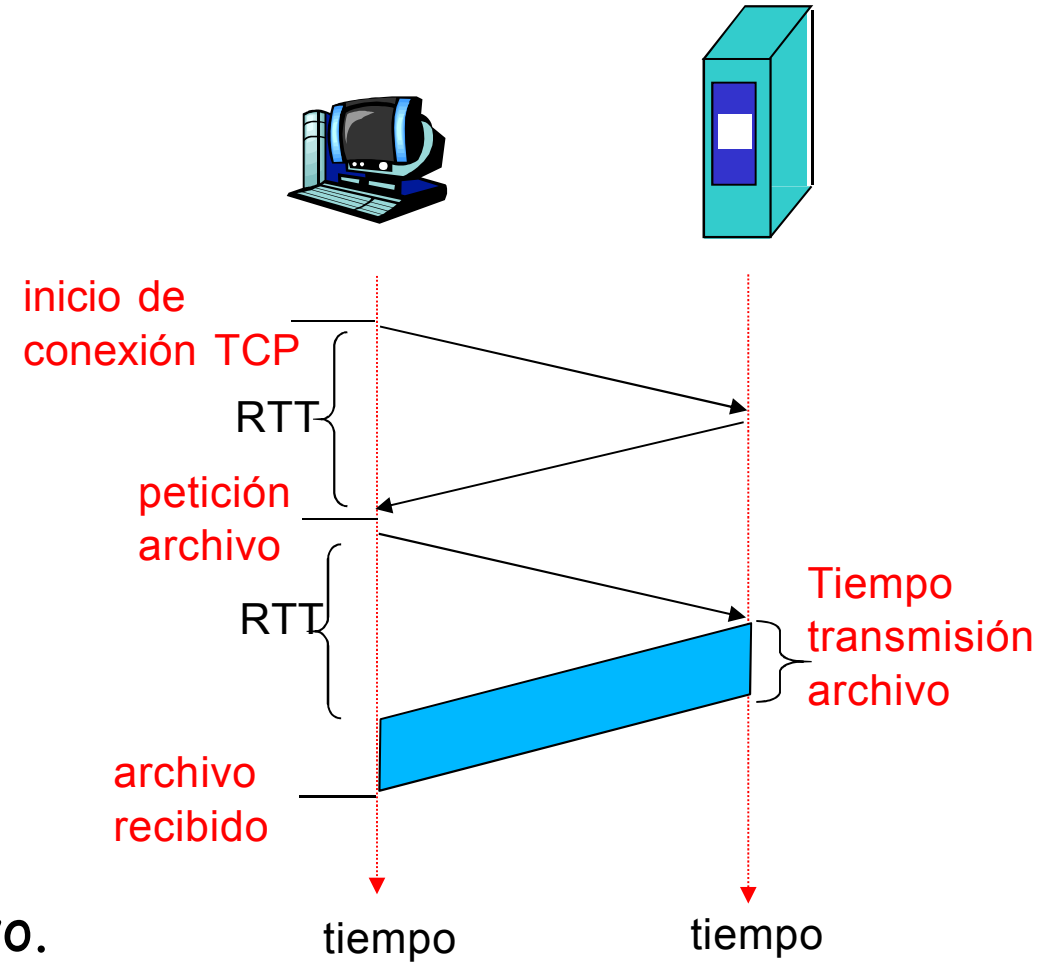
6. Los pasos 1-5 se repiten para cada una de los 13 objetos (4 imágenes y 9 scripts JavaScript) con URLs distintas.

HTTP No-persistente: Tiempo de respuesta

RTT (*Round-Trip Time*, tiempo de ida y vuelta): el tiempo que tarda una PDU con pocos bytes en viajar del cliente al servidor y de vuelta.

Tiempo de respuesta:

- 1 RTT, inicio conexión.
- 1 RTT, petición HTTP y primeros bytes de respuesta HTTP.
- Tiempo transmisión archivo.



Tiempo de respuesta = 2RTT + tiempo transmisión archivo

HTTP Persistente

Inconvenientes del HTTP no-persistente:

- Requiere 2 RTTs por objeto (más lo que tarde en transmitirse dicho objeto).
- Sobrecarga SO con *cada* conexión TCP

Nota

Conexiones HTTP en paralelo:

- Los navegadores a menudo abren varias conexiones TCP en paralelo para obtener los objetos referenciados más rápidamente, los cuales se piden de forma simultánea por cada conexión (sean estas persistentes o no).

HTTP persistente

- El servidor mantiene la conexión abierta tras enviar la respuesta
- Los siguientes mensajes HTTP entre el mismo cliente y el servidor se envían por la conexión abierta
- El cliente envía una nueva petición cuando acaba de recibir el objeto anterior
- Cada objeto referenciado tarda sólo 1 RTT (más lo que tarde en transmitirse dicho objeto).

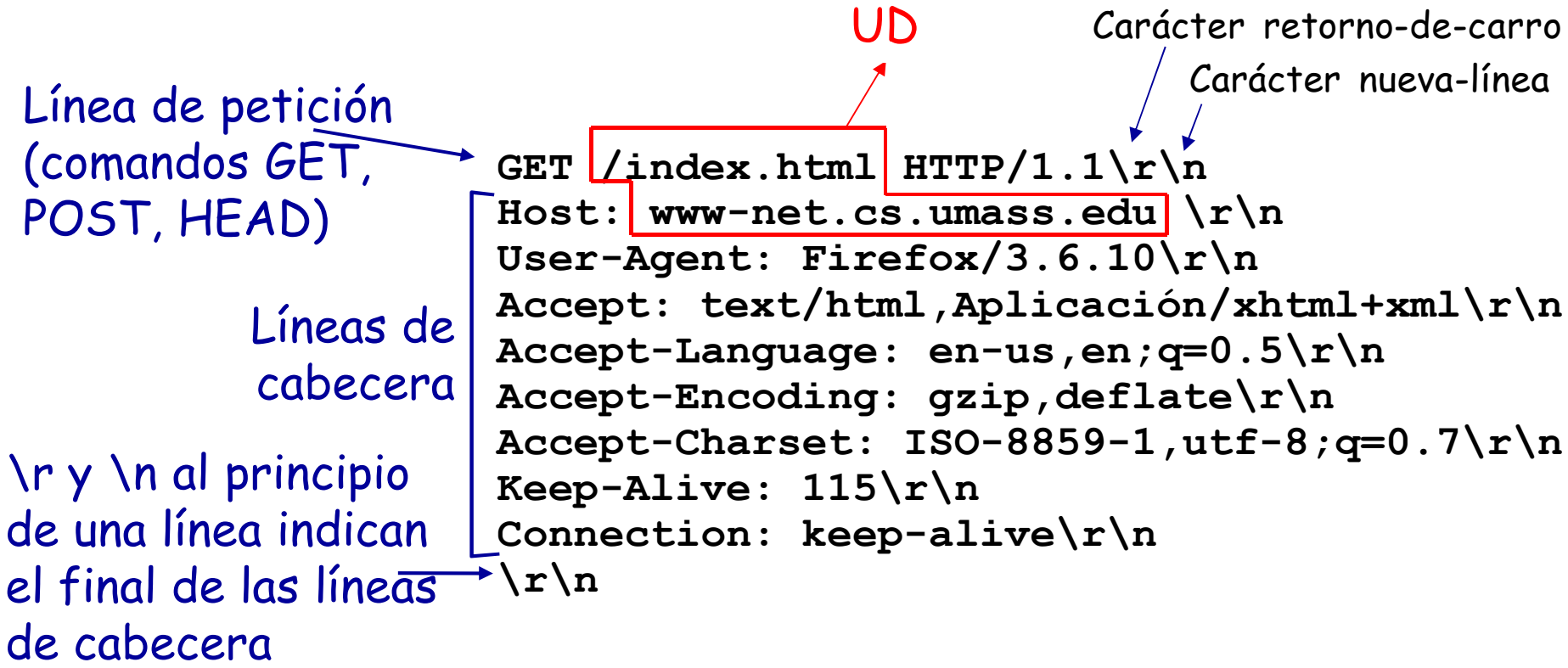
Mensaje de Petición HTTP

- Hay 2 tipos de mensajes HTTP (HTTP_PDU): *petición y respuesta*
- Petición HTTP:
  - ASCII (texto inteligible)

Nota

<CR>: Carriage-Return : \r

<LF>: Line-Feed: \n



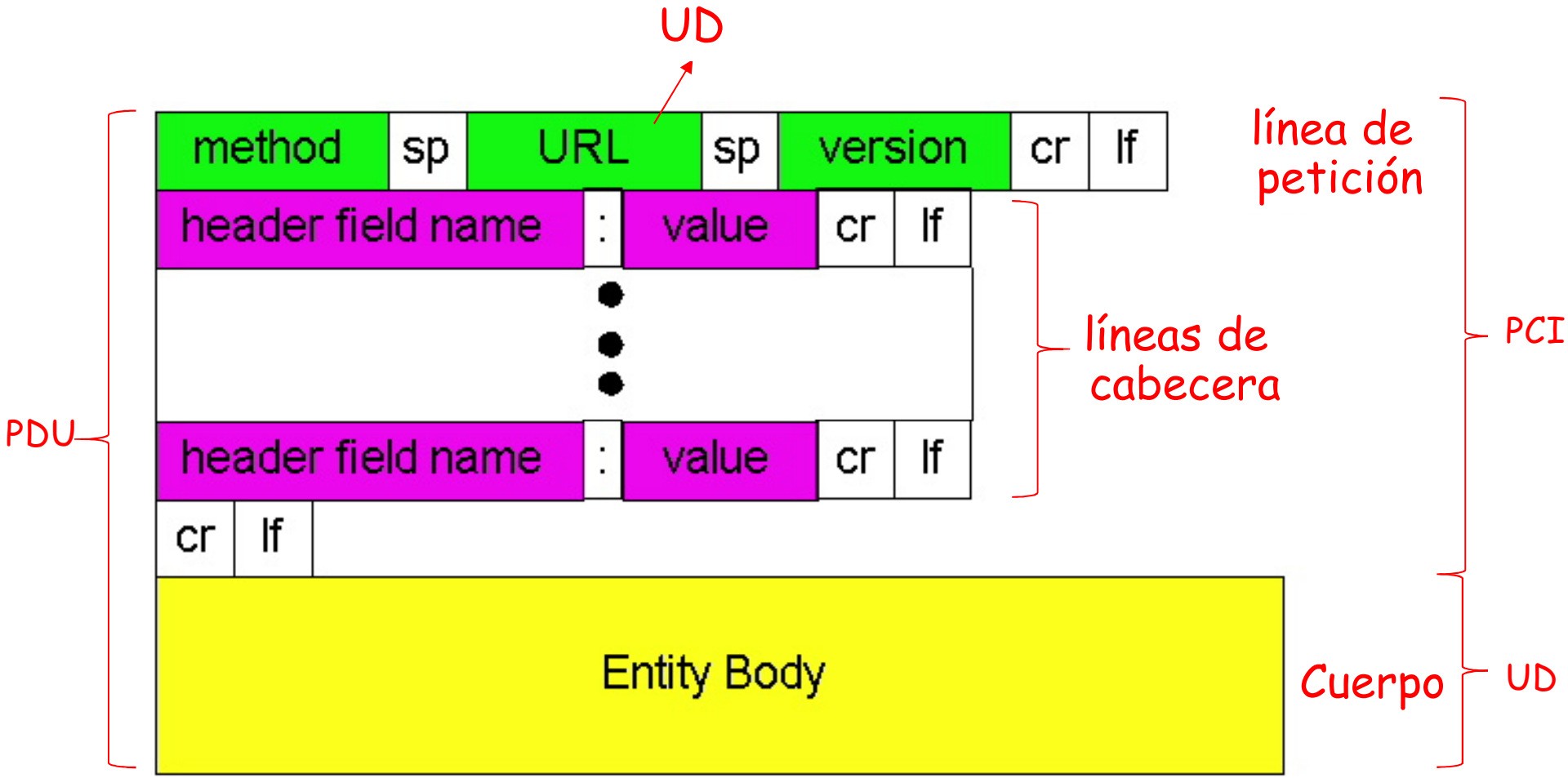
Algunas Cabeceras HTTP que puede enviar el cliente

- Host: ***hostname (nombre del servidor web)***
- User-Agent: *versión_del_navegador*
- Accept-xxx: lista_de_preferencias_para_xxx
- Connection: keep-alive

  El cliente solicita al servidor una conexión persistente al servidor
- Keep-Alive: *nnn*

  El cliente solicita al servidor el tiempo máximo de *nnn* segundos para las conexiones persistentes

Mensaje de Petición HTTP: formato general



Subida de parámetros (de un formulario)

Una página web a menudo incluye un formulario con unos parámetros que se envían al servidor. Hay 2 métodos de envío:

Método POST:

- Los parámetros se suben al servidor en el CUERPO de la petición

Método GET:

- Las entradas se pasan al servidor en la propia URL, con la línea de petición (separado por "?" y "&"):

```
GET /buscar?monos&platanos HTTP/1.1\r\n
Host: www.unbuscador.com\r\n
....
```

Tipos de métodos

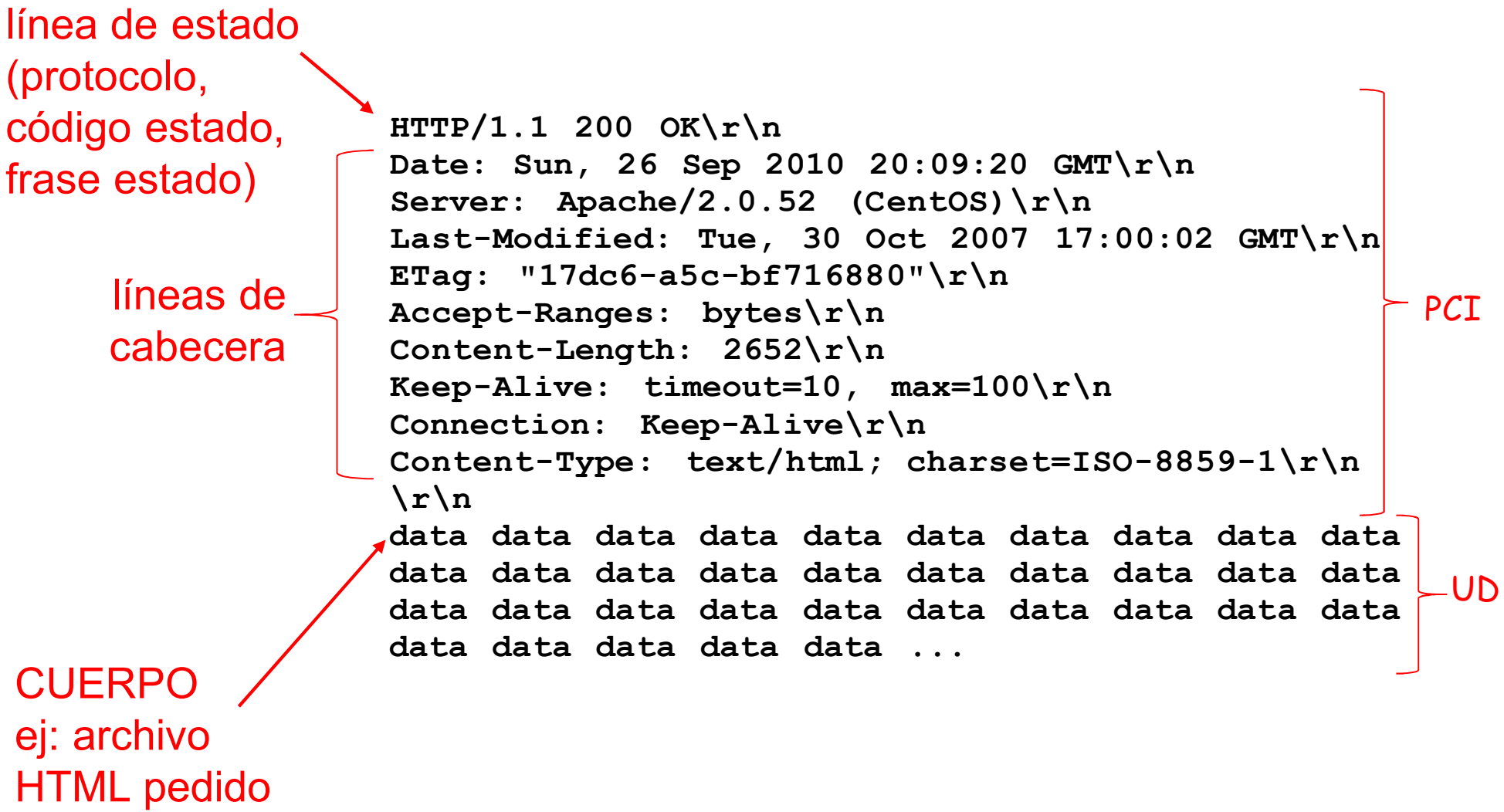
HTTP/1.0 (RFC-1945)

- GET
- POST
- HEAD
  - Idéntico al GET, salvo que no se incluye el objeto en el cuerpo de la respuesta (sólo las cabeceras correspondientes)

HTTP/1.1 (RFC-2616)

- GET, POST, HEAD
- PUT
  - Sube el fichero en el CUERPO de la petición al path especificado en la URL
- DELETE
  - Borra del servidor el fichero especificado en la URL

Mensaje de Respuesta HTTP



Algunas Cabeceras HTTP que puede enviar el servidor

- Date: *fecha (en el que se envía el mensaje)*
- Last-Modified: *fecha (en la que el objeto se modificó por última vez)*
- Server: *versión_del_servidor*
- Content-Type: *tipo_del_objeto (HTML, imagen, …)*
- Content-Length: *tamaño_del_cuerpo (en bytes)*
- Connection: keep-alive

  El servidor confirma al cliente que esa conexión será persistente
- Keep-Alive: timeout=*ttt*, max=*nnn*

  El servidor cerrará la conexión persistente tras *ttt* segundos de inactividad o tras *nnn* segundos en cualquier caso.

Códigos de estado (Respuesta)

- El código de estado aparece en la primera línea de la respuesta del servidor al cliente.
- Algunos códigos habituales:

200 OK

- Petición exitosa, el objeto solicitado va a continuación...

301 Moved Permanently

- El objeto se ha movido permanentemente, se especifica la ubicación nueva (cabecera "Location:")

400 Bad Request

- Mensaje de petición no entendido por el servidor

404 Not Found

- El documento solicitado no se encuentra en el servidor

505 HTTP Version Not Supported

Prueba el HTTP tú mismo

1. Telnet a tu servidor Web favorito:

```
telnet www.dte.us.es 80
```

Abre conexión TCP al puerto 80 (puerto HTTP por defecto) de www.dte.us.es

Todo lo que escribas se envía allí

2. Escribe una petición GET:

```
GET /docencia/ HTTP/1.1
Host: www.dte.us.es
```

Escribe esto (con doble-enter al final) para enviar un GET request reducido a un servidor HTTP

3. Mira el mensaje de respuesta del servidor!
   (o puedes usar Wireshark!)

P. ¿Qué ocurre si envío "hola"?

Cookies: manteniendo "el estado"

Muchos servicios web usan cookies

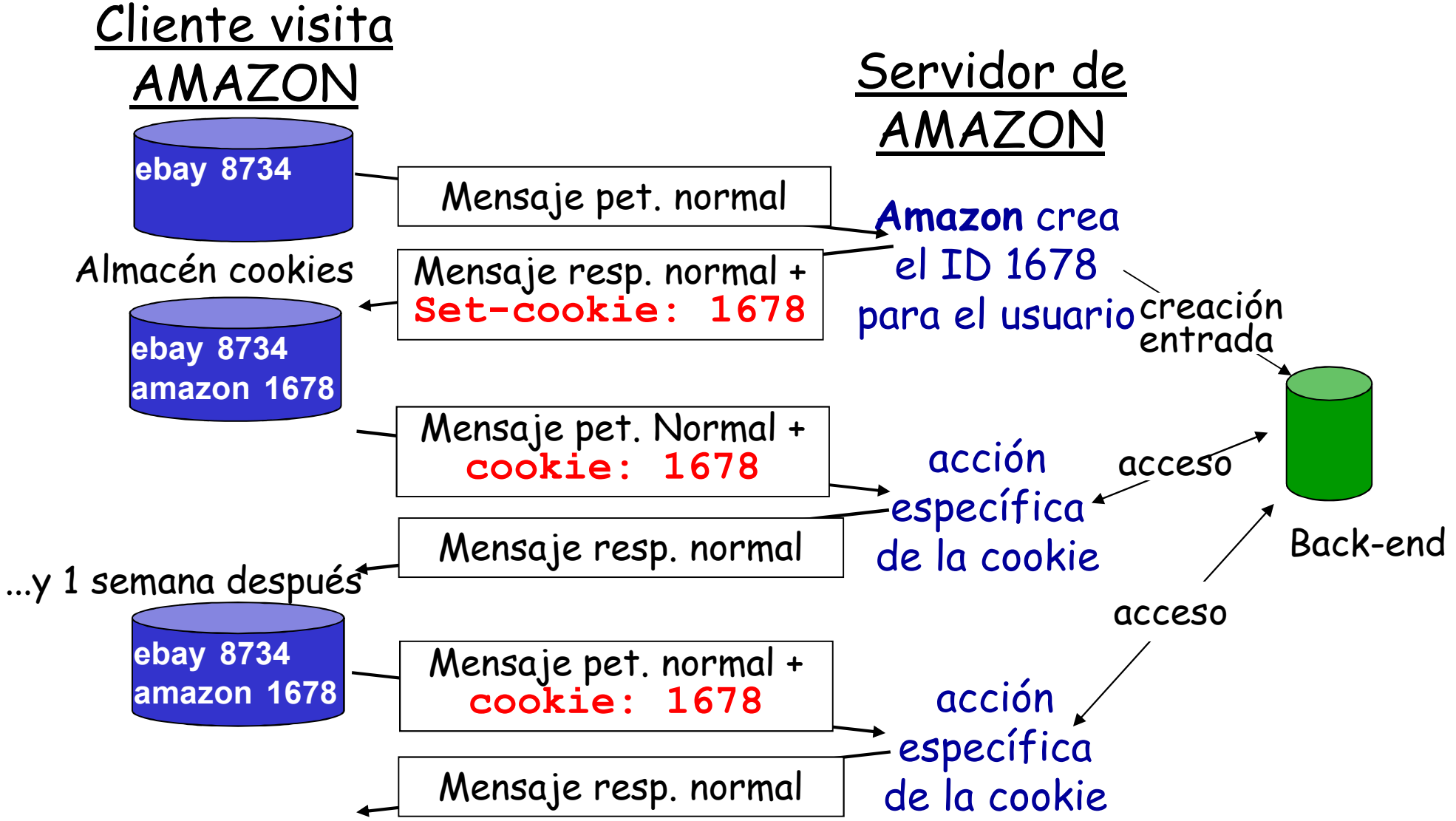
4 componentes:

1) cabecera Set-cookie: en el mensaje de respuesta
2) cabecera Cookie: en el mensaje de petición
3) archivo de cookies almacenado por el equipo del usuario y gestionado por el navegador
4) base de datos back-end en el servidor web

Ejemplo de uso:

- Susana siempre accede a Internet desde su PC
- Visita una tienda online (ej: Amazon) por primera vez
- Al llegar las peticiones, el servidor crea:
  - Un ID único
  - Una entrada para esa ID en la base de datos back-end

Cookies: Ejemplo de uso



Cookies: discusión

Posibles aplicaciones:

- autorización
- carritos de la compra
- recomendaciones
- mantenimiento de sesión de usuario (ej: webmail)

Nota

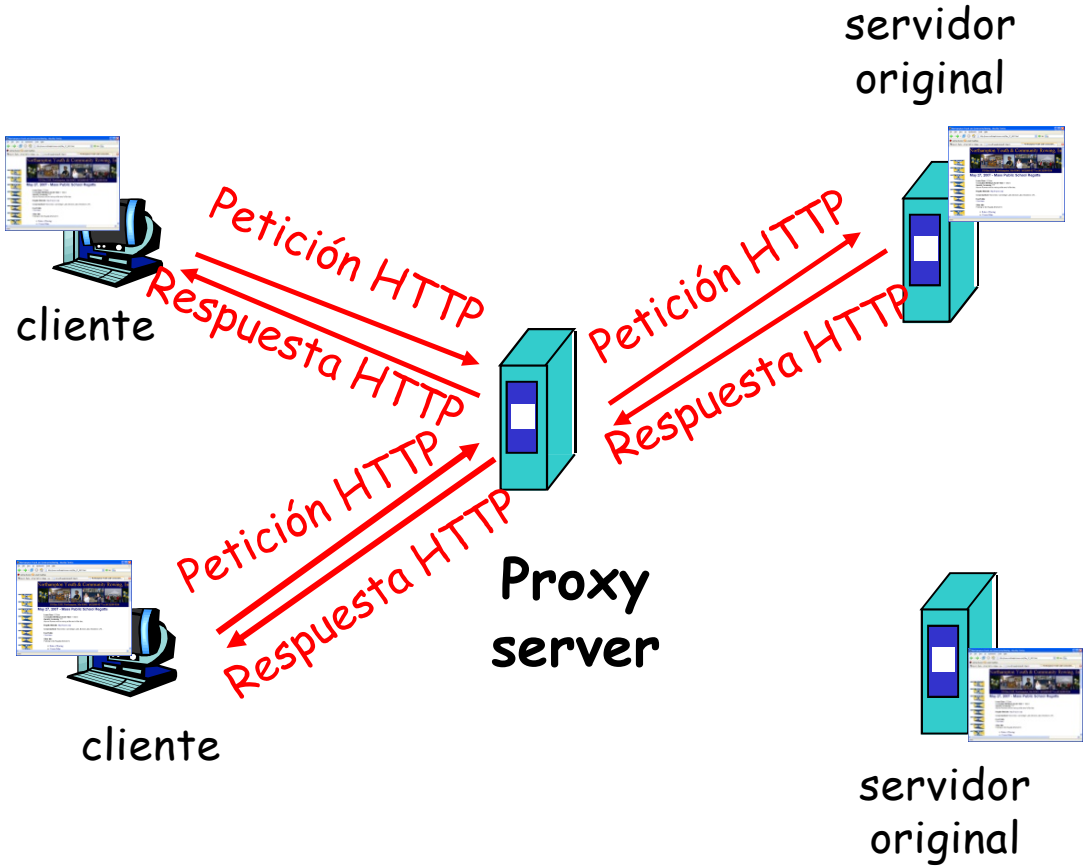
cookies y la intimidad:

- las cookies permiten a los sitios conocer mucho sobre ti
- puedes estar dando información personal a esas páginas: emails, nombres, etc...

Servidor Proxy (Caché de la Web)

Objetivo: satisfacer la petición del cliente sin involucrar al servidor web original

- El navegador se configura para usar el Proxy-Caché.
- Entonces se envían todas las peticiones HTTP al Proxy
  - objeto en la caché: se devuelve el objeto
  - si no: caché solicita el objeto al servidor original y lo devuelve al cliente.



Más acerca del Proxy

- El caché actúa como cliente (del servidor original) y como servidor (del cliente)
- Normalmente se instalan en los ISP (universidades, compañias, ISPs residenciales)

¿por qué es interesante?

- Reducir el tiempo de respuesta de la petición del cliente
- Reducir el tráfico de enlace de datos de una institución
- Permitir a proveedores "pequeños" entregar de forma eficiente los contenidos (algo que también permite P2P)

GET Condicional

Objetivo

Que el servidor web no envíe el objeto si la caché tiene una versión actualizada del mismo

- Proxy (o caché del navegador): especifica la fecha de la copia cacheada en la petición HTTP

```
If-modified-since: <date>
```

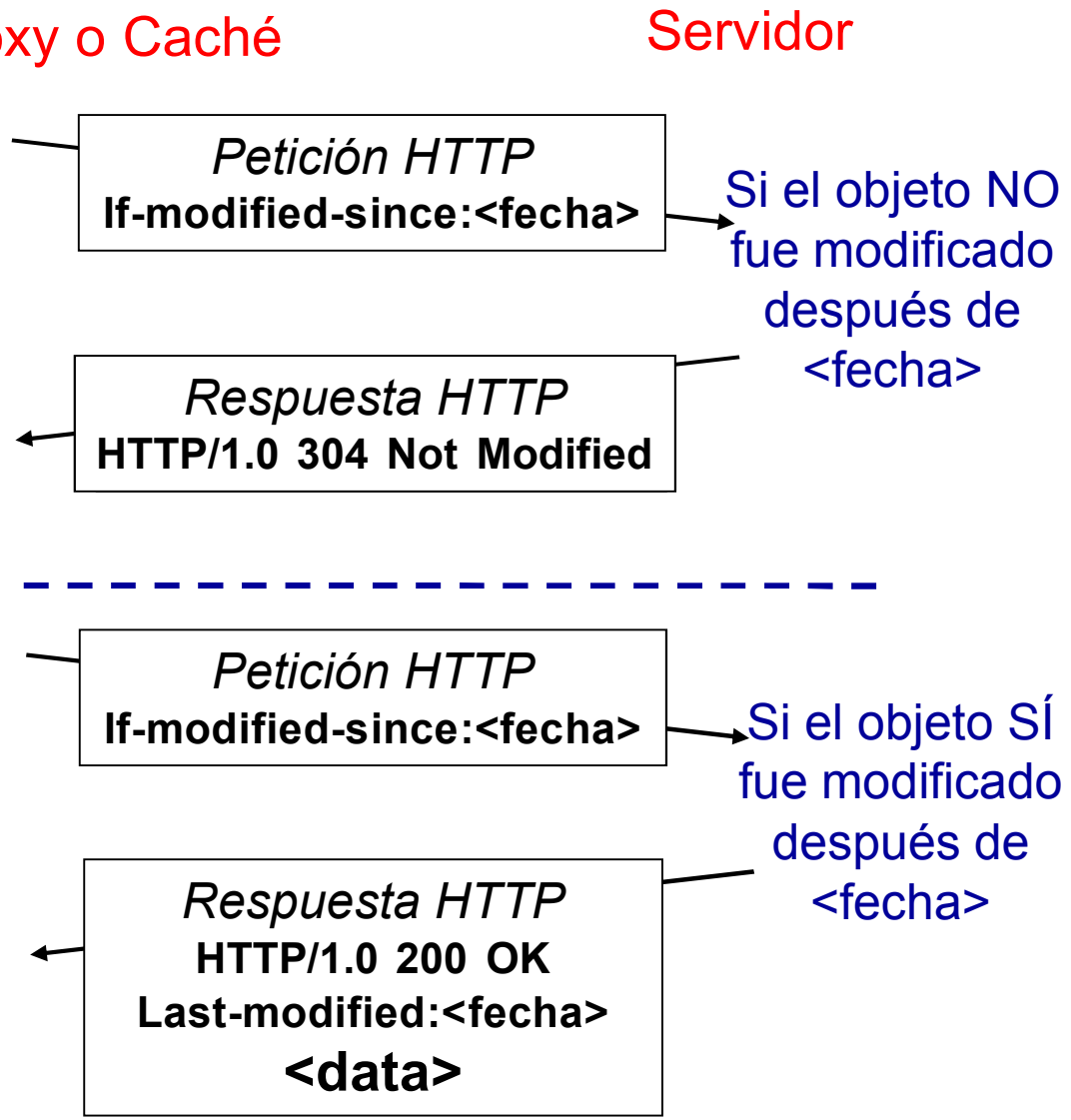
- Servidor: en la respuesta van cabeceras y…

a) no va ningún objeto si la copia no se ha modificado…

```
HTTP/1.0 304 Not Modified
```

b) o bien se envía el objeto si está modificado, junto con la fecha de modificación:

```
Last-modified:<fecha>
```



Tema 2: Nivel de aplicación

2.1 Principios de las aplicaciones en red

2.2 DNS

2.3 Web y HTTP

2.4 Programación de la interfaz de acceso al servicio de transporte fiable de Internet en JAVA: Sockets con TCP

2.5 Programación de la interfaz de acceso al servicio de transporte no fiable de Internet en JAVA: Sockets con UDP

Programación de Sockets

Objetivo: aprender cómo se programa una aplicación cliente/servidor que se comunique usando sockets

Socket API

- Se introdujo en BSD4.1 UNIX, 1981
- Los sockets se crean, usan y liberan de forma explícita por las aplicaciones
- Paradigma cliente/servidor
- 2 tipos de servicios:
  - No fiable, orientado a datagramas (UDP)
  - Fiable, orientado a flujo de bytes (TCP)

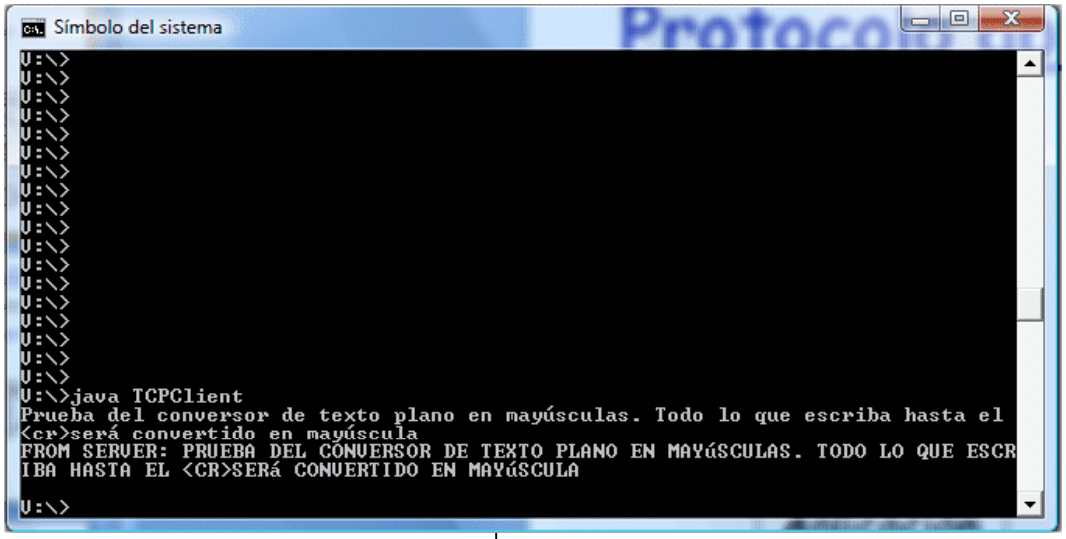
socket

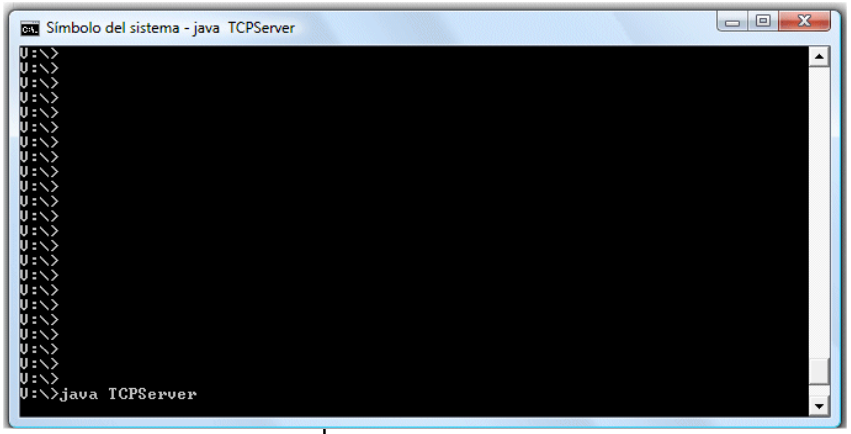
Un interfaz del equipo local, creado por una aplicación y controlado por el SO (una "puerta") por la que el proceso de aplicación puede tanto envíar/recibir mensajes a/desde otros procesos remotos (o incluso locales)

Protocolo aplicación

Cliente

Servidor





Dir IP cliente, puerto

Dir IP servidor, puerto

Transporte

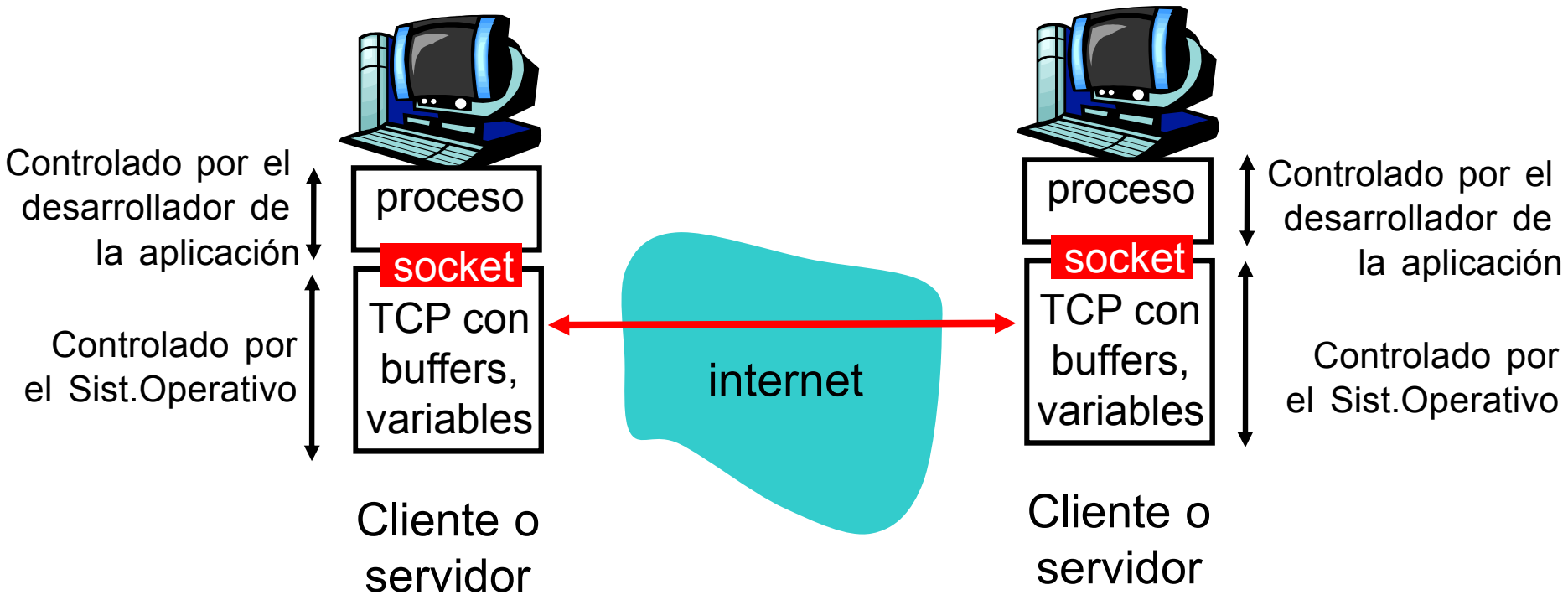
Transporte

Internet

Programación de Sockets con TCP

Socket: la interfaz entre el proceso y el protocolo de transporte extremo a extremo (TCP o UDP)

servicio TCP: transferencia fiable de un *flujo de bytes (stream)* de un proceso a otro



Programación de Sockets con TCP

El cliente debe contactar con el servidor

- Proceso servidor debe estar corriendo primero
- El servidor debe haber creado un socket (una "puerta") que aceptará la solicitud de conexión de cualquier cliente.

El cliente para contactar

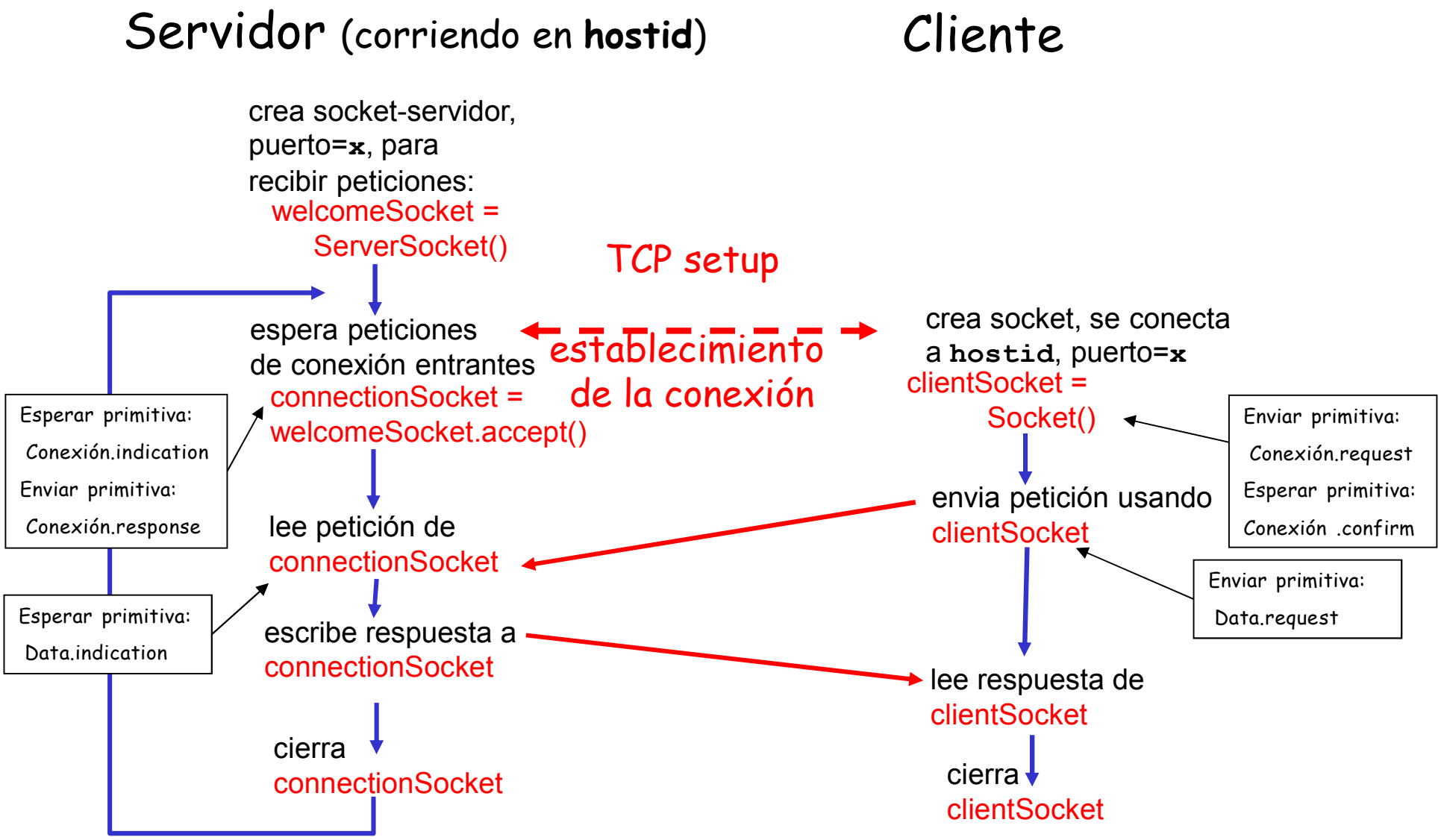
- creará un socket TCP local
- especificará la IP y el nº de puerto del proceso servidor
- Al crearse el socket en el cliente se crea una conexión TCP con el servidor

- Cuando es contactado por un cliente, el servidor TCP crea un nuevo socket para que el proceso servidor se comunique con él
  - Permite al servidor hablar con múltiples clientes
  - El nº de puerto de origen se usa para distinguir a los clientes [más en Tema 3]

Punto de vista de la aplicación

TCP provee una transferencia fiable y ordenada de bytes entre un cliente y un servidor

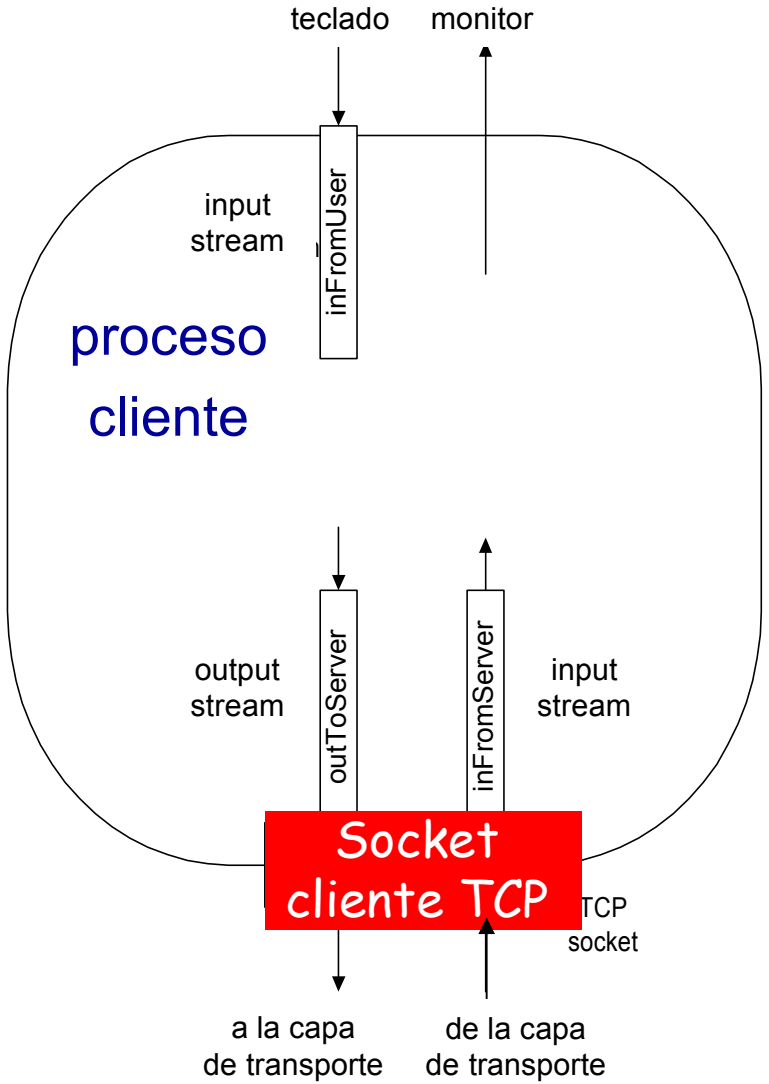
Interacción cliente/servidor con TCP



hostid = IP o nombre

Stream de Java

- stream (flujo) es una secuencia de caracteres/bytes que entran o salen a/de un proceso
- input stream está conectado a una fuente de entrada de datos, como un teclado o un socket
- output stream está conectado a una fuente de salida de datos, como un monitor o un socket
- Ej: en el cliente vamos a usar 3 streams y 1 único socket



Ejemplo: aplicación con sockets TCP

Aplicación de ejemplo cliente/servidor TCP:

1) el cliente lee una línea de entrada standard (**`inFromUser`** stream) , y la envía al servidor por un socket (**`outToServer`** stream)
2) el servidor lee la línea de un socket
3) el servidor convierte la línea a mayúsculas y la envía de vuelta al cliente por el socket
4) el cliente la lee del socket (**`inFromServer`** stream) y la muestra en el monitor

Ejemplo: cliente Java (TCP)

```
import java.io.*;
import java.net.*;
class TCPClient {

    public static void main(String argv[]) throws Exception
    {
        String sentence;
        String modifiedSentence;

        BufferedReader inFromUser =
          new BufferedReader(new InputStreamReader(System.in));

        Socket clientSocket = new Socket("hostname", 6789);

        DataOutputStream outToServer =
          new DataOutputStream(clientSocket.getOutputStream());
```

Este paquete contiene las clases Socket() y ServerSocket()

Nombre del servidor
ej: www.dte.us.es
ICI

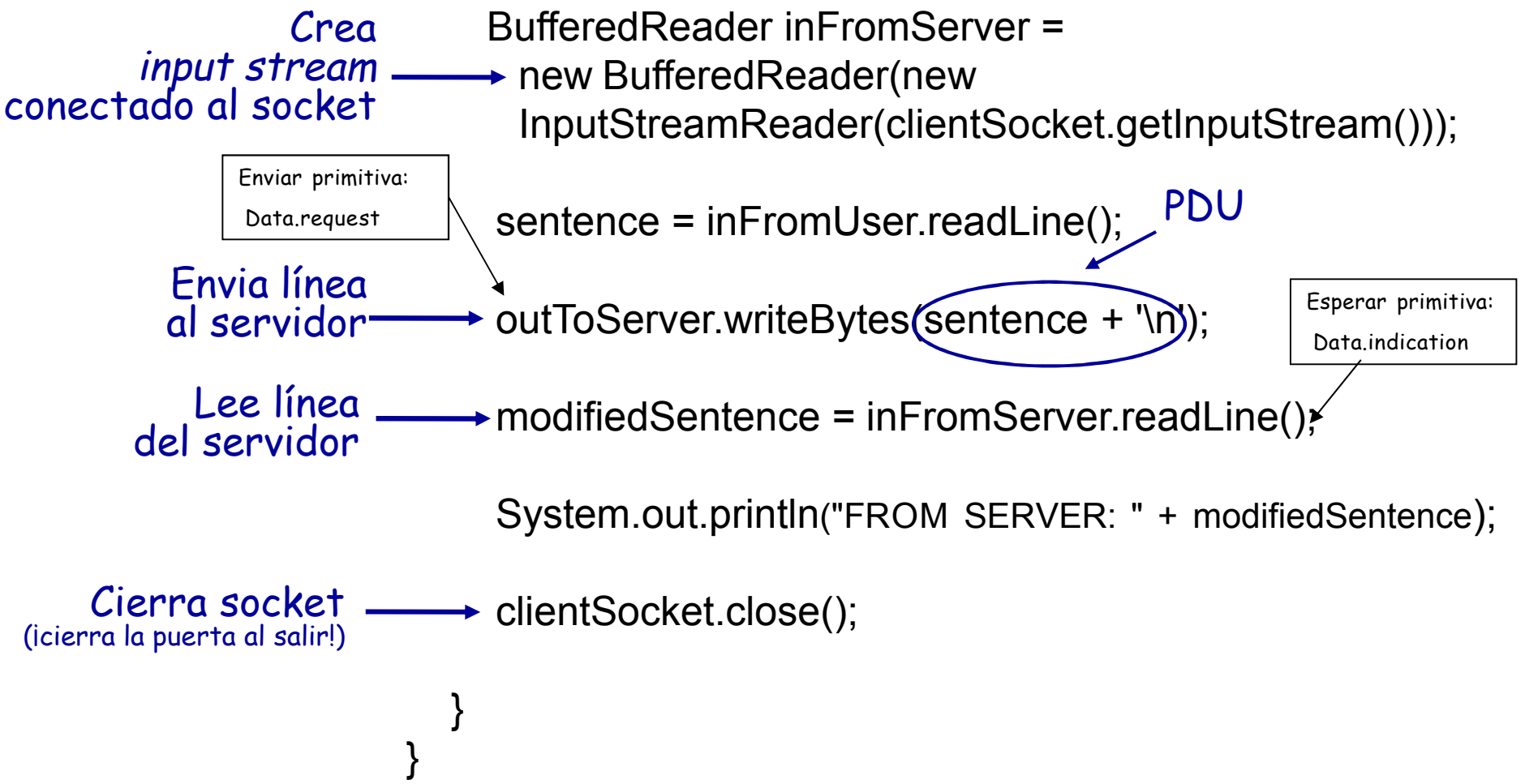
Crea *input stream*

Crea objeto *clientSocket* de tipo *Socket*, conecta al servidor

Nº de puerto del servidor
ICI

Crea *output stream* conectado al socket

Ejemplo: cliente Java (TCP) (cont)



Ejemplo: servidor Java (TCP)

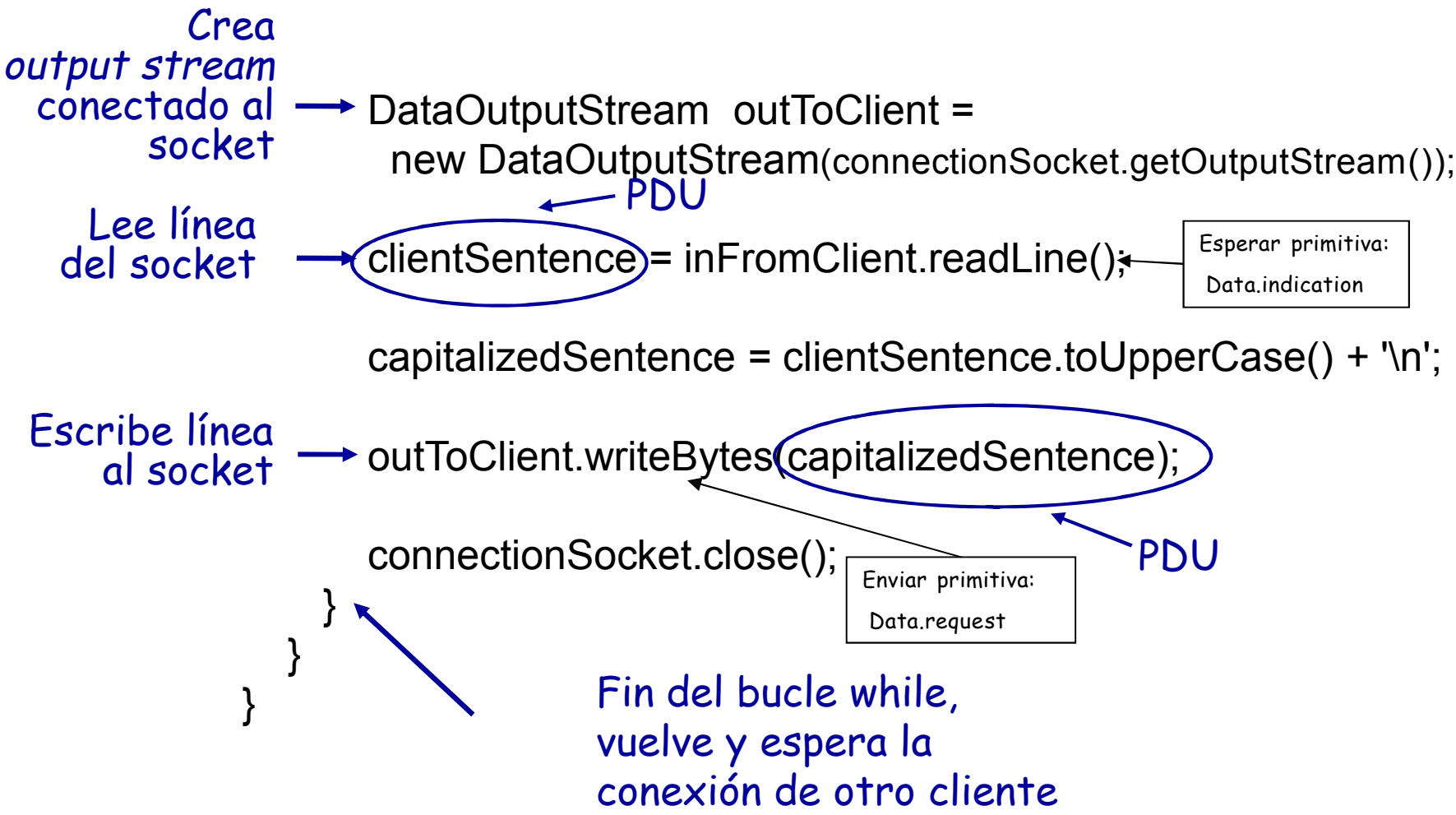
```
import java.io.*;
import java.net.*;

class TCPServer {

  public static void main(String argv[]) throws Exception
   {
     String clientSentence;
     String capitalizedSentence;

     ServerSocket welcomeSocket = new ServerSocket(6789);

     while(true) {

       Socket connectionSocket = welcomeSocket.accept();

       BufferedReader inFromClient =
         new BufferedReader(new
         InputStreamReader(connectionSocket.getInputStream()));
```

Crea socket de acogida en el puerto 6789

El socket de acogida espera con el método *accept*() al contacto de un cliente, retorna un *nuevo* socket

Crea *input stream* conectado al socket

Ejemplo: servidor Java (TCP) (cont)



Tema 2: Nivel de aplicación

2.1 Principios de las aplicaciones en red

2.2 DNS

2.3 Web y HTTP

2.4 Programación de la interfaz de acceso al servicio de transporte fiable de Internet en JAVA: Sockets con TCP

2.5 Programación de la interfaz de acceso al servicio de transporte no fiable de Internet en JAVA: Sockets con UDP

Programación de Sockets con UDP

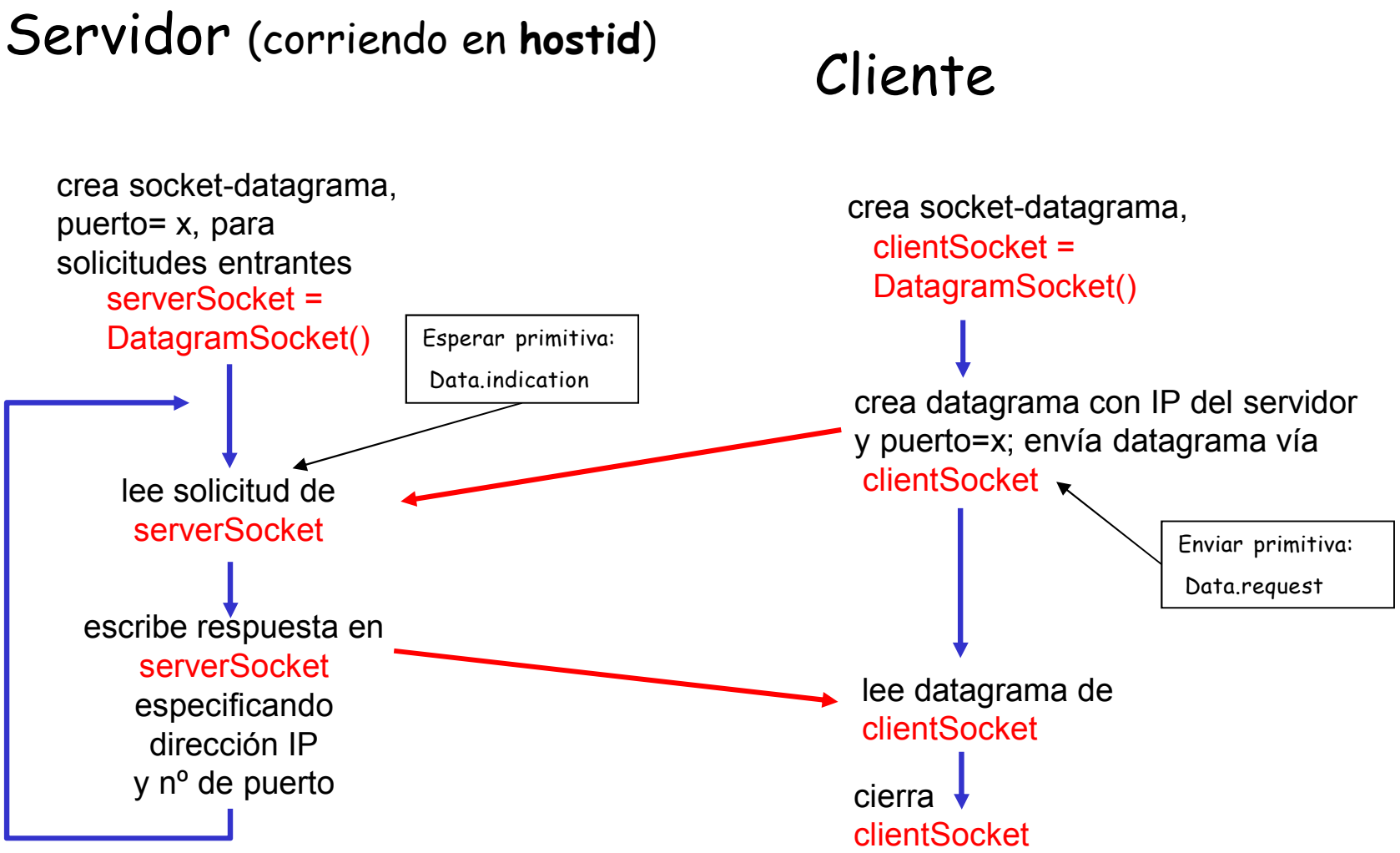
UDP: no hay "conexión" entre cliente y servidor

- Sin negociación
- El cliente explícitamente adjunta la dirección IP y puerto de destino de cada mensaje (PDU) que quiera enviar.
- El servicio de transporte debe pasar al servidor la dirección IP y el puerto del mensaje recibido.

UDP: los datos transmitidos pueden ser recibidos de forma desordenada, o algunos pueden perderse

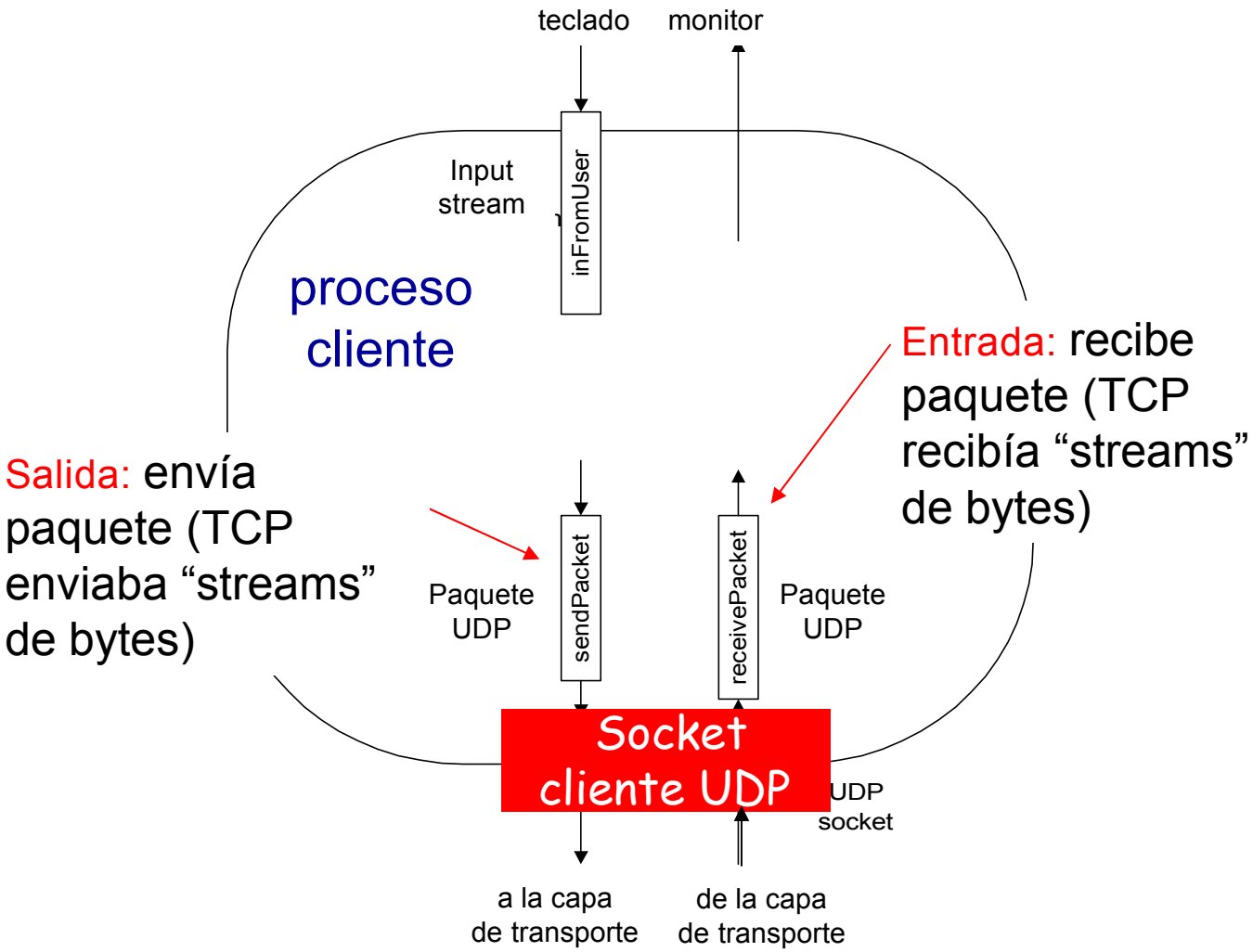
Punto de vista de la aplicación:
UDP proporciona una transferencia no fiable de "datagramas" (grupos de bytes) entre el cliente y el servidor

Interacción cliente/servidor con UDP



hostid = IP o nombre

Ejemplo: cliente (con datagramas UDP)



Ejemplo: cliente Java (UDP)

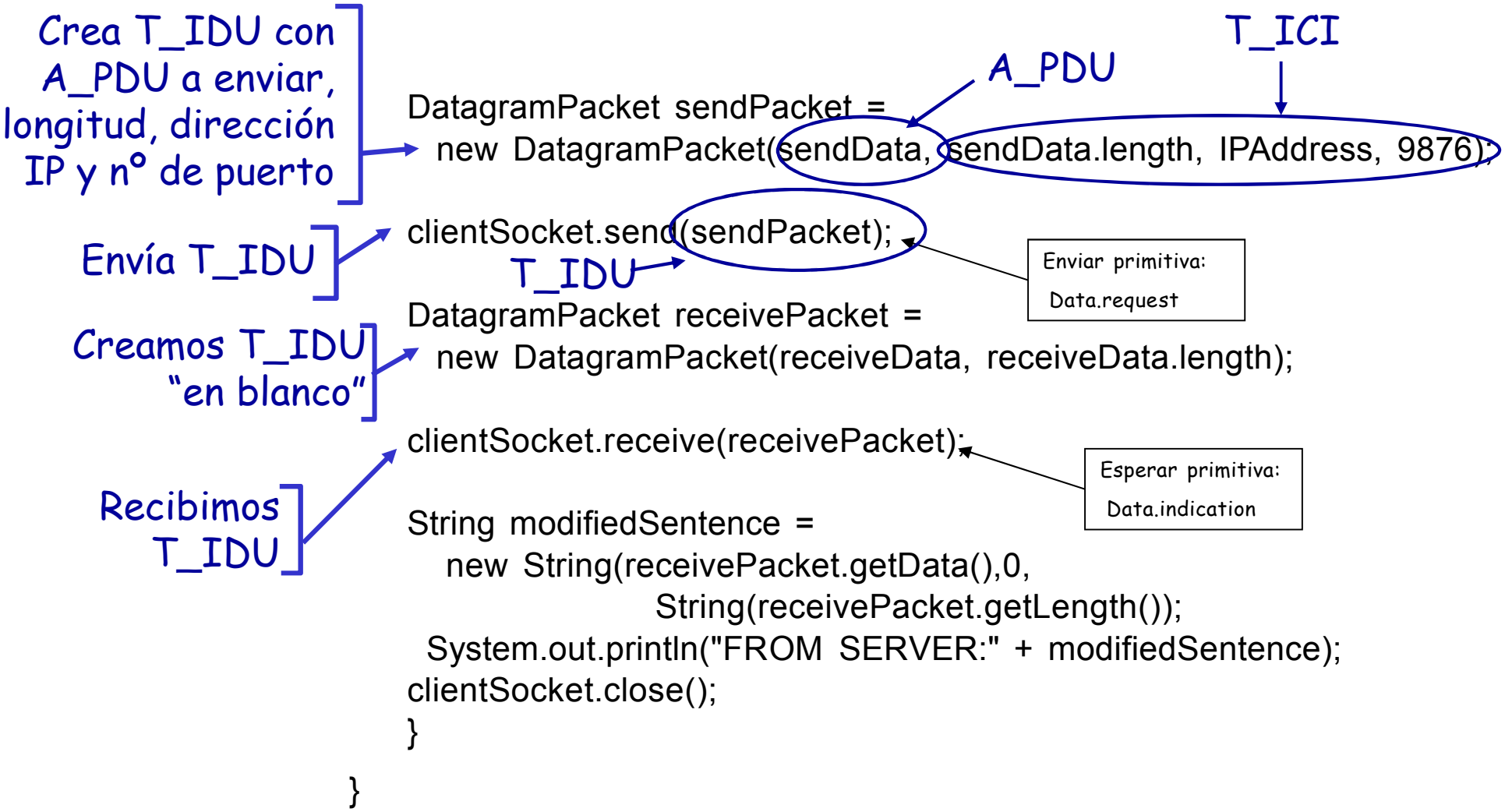
```
import java.io.*;
import java.net.*;

class UDPClient {
   public static void main(String args[]) throws Exception
   {

     BufferedReader inFromUser =
       new BufferedReader(new InputStreamReader(System.in));

     DatagramSocket clientSocket = new DatagramSocket();

     InetAddress IPAddress = InetAddress.getByName("hostname");

     byte[] sendData = new byte[1024];
     byte[] receiveData = new byte[1024];

     String sentence = inFromUser.readLine();

     sendData = sentence.getBytes();
```

Crea *input stream*

Crea socket UDP en puerto libre

Traduce nombre del servidor a dirección IP usando DNS

Ejemplo: cliente Java (UDP) (cont)



Ejemplo: servidor Java (UDP)

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

class UDPServer {
  public static void main(String args[]) throws Exception
   {

    DatagramSocket serverSocket = new DatagramSocket(9876);

    byte[] receiveData = new byte[1024];
    byte[] sendData  = new byte[1024];

    while(true)
     {

      DatagramPacket receivePacket =
        new DatagramPacket(receiveData, receiveData.length);

      serverSocket.receive(receivePacket);
```

Crea socket UDP en puerto 9876 conocido por cliente

Creamos T_IDU "en blanco"

Recibimos T_IDU

Esperar primitiva: Data.indication

Ejemplo: servidor Java (UDP) (cont)

```
String sentence = new String(receivePacket.getData(),0,
                                        receivePacket.getLength());
InetAddress IPAddress = receivePacket.getAddress();

int port = receivePacket.getPort();

        String capitalizedSentence = sentence.toUpperCase();

sendData = capitalizedSentence.getBytes();

DatagramPacket sendPacket =
  new DatagramPacket(sendData, sendData.length, IPAddress,
                     port);

serverSocket.send(sendPacket);
    }
  }
}
```

Obtiene de la T_ICI la dirección IP y nº de puerto del emisor

Crea T_IDU con A_PDU a enviar, longitud, dirección IP y nº de puerto

Envía T_IDU

Fin del bucle *while*, vuelve y espera la llegada de otro datagrama

Departamento de
Tecnología Electrónica

Redes de Computadores -
Tema 2: Nivel de Aplicación

EJERCICIOS

Pr1: ¿Verdadero o Falso?

a) Un usuario solicita una página web que consta de texto y 3 referencias a imágenes. Para obtener esa página, el cliente envía un mensaje de solicitud y recibe cuatro mensajes de respuesta.

b) Dos páginas web diferentes (www.mit.edu/research.html y www.mit.edu/students.html) se pueden enviar a través de la misma conexión persistente.

c) Con las conexiones no persistentes entre un navegador y un servidor de origen, un único segmento TCP puede transportar dos mensajes de solicitud HTTP distintos.

d) La línea de cabecera "Date:" del mensaje de respuesta HTTP indica cuándo el objeto fue modificado por última vez.

e) Los mensajes de respuesta HTTP nunca incluyen un cuerpo de mensaje vacío.

Pr2: Aplicación-Transporte

Un cliente HTTP desea recuperar un documento web que se encuentra en una URL dada. Inicialmente, la dirección IP del servidor HTTP es desconocida.

- ¿Qué protocolos de la capa de aplicación y de la capa de transporte, además de HTTP, son necesarios en este escenario?

Pr3: Cabeceras Cliente HTTP

La siguiente cadena ASCII ha sido capturada cuando el navegador enviaba un mensaje GET HTTP.

NOTA: La anchura de las líneas del recuadro es 60 caracteres

```
GET /cs453/index.html HTTP/1.1←↓Host: gaia.cs.umass.edu←↓Use
r-Agent: Mozilla/5.0 (Windows;U; Windows NT 5.1; en-US; rv:1
.7.2) Gecko/20040804 Netscape/7.2 (ax)←↓Accept: ext/xml, app
lication/xml, application/xhtml+xml, text/html;q=0.9, text/p
lain;q=0.8, image/png, */*;q=0.5←↓Accept-Language: en-us,en;
q=0.5←↓Accept-Encoding: zip,deflate←↓Accept-Charset: ISO-885
9-1,utf-8;q=0.7,*;q=0.7←↓Keep-Alive: 300←↓Connection: keep-a
live←↓←↓
```

NOTA: ← es un retorno de carro y ↓ es un fin de línea.

Responda a las siguientes cuestiones, indicando en que parte del mensaje GET HTTP se encuentra la respuesta a la cuestión:

a) ¿Cuál es la URL del documento solicitado?

b) ¿Qué versión de HTTP se está ejecutando en el navegador?

c) ¿Solicita el navegador una conexión persistente o no?

d) ¿Cuál es la dirección IP del host que corre el navegador?

e) ¿Qué tipo de navegador envía el mensaje? ¿Por qué es necesario indicar el tipo de navegador en el mensaje?

f) ¿Cuántos bytes ocupa la HTTP_PDU enviada por el cliente?

g) ¿Cuántos bytes de HTTP_UD transporta?

Pr4: Cabeceras Servidor HTTP

La siguiente cadena muestra la respuesta devuelta por el servidor web al mensaje del problema anterior.

NOTA: La anchura de las líneas del recuadro es 60 caracteres

```
HTTP/1.1 200 OK←↓Date: Tue, 07 Mar 2008 12:39:45 GMT←↓Server
: Apache/2.0.52 (Fedora)←↓Last-modified: Sat, 10 Dec 2005 18
:27:46 GMT←↓ETag: "526c3-f22-a88a4c80"←↓Accept-Ranges: bytes
←↓Content-Length: 3874←↓Keep-Alive: timeout=max=100←↓Connect
ion: keep-alive←↓Content-Type: text/html; charset=ISO-8859-1
←↓←↓<!doctype html public "-//w3c//dtd html 4.0 transitional
//en">←↓<html>←↓<head>←↓<meta name="GENERATOR" content="Mozi
lla/4.79 [en] (Windows NT 5.0; U) Netscape]">←↓<title>←↓</he
ad>←↓...Aquí seguiría el resto del documento HTML...
```

NOTA: ← es un retorno de carro y ↓ es un fin de línea.

Responda a las siguientes cuestiones, indicando en que parte del mensaje respuesta HTTP se encuentra la respuesta a la cuestión:

a) ¿Ha encontrado el servidor el documento? ¿En qué momento se suministra la respuesta con el doc.?

b) ¿Cuándo fue modificado por última vez el documento?

c) ¿Cuántos bytes contiene el documento devuelto?

d) ¿Cuáles son los primeros 5 bytes del documento devuelto?

e) ¿Ha acordado el servidor emplear una conexión persistente? (si es que sí diga el tiempo máximo de inactividad que se permite)

f) ¿Cuántos bytes de HTTP_UD transporta?

g) ¿Cuántos bytes ocupa la HTTP_PDU enviada por el servidor?

Pr5: Tiempo de transferencia (I)

Suponga que en su navegador hace clic en un vínculo a una página web. La dirección IP correspondiente al URL asociado no está almacenada en la caché de su host local, por lo que es necesario realizar una búsqueda DNS.

Suponga que el tiempo de ida y vuelta (RTT) de la consulta al servidor DNS es RTT_{DNS}

Suponga también que la página web asociada con el vínculo es un pequeño fichero HTML (lo que supone un tiempo de transmisión despreciable) y que no contiene referencias a otros objetos.

Sea RTT_0 el tiempo RTT entre el host local y el servidor web.

¿Cuánto tiempo transcurre desde que el cliente hace clic en el vínculo hasta que recibe el objeto?

Pr6: Tiempo de transferencia (II)

Continuando con el Problema 5, suponga que el archivo base HTML hace referencia a 8 objetos muy pequeños que se encuentran en el mismo servidor.

Despreciando los tiempos de transmisión, para cargar la página web completa, ¿cuánto tiempo transcurre si se utiliza…

a) …HTTP no persistente sin conexiones TCP en paralelo?

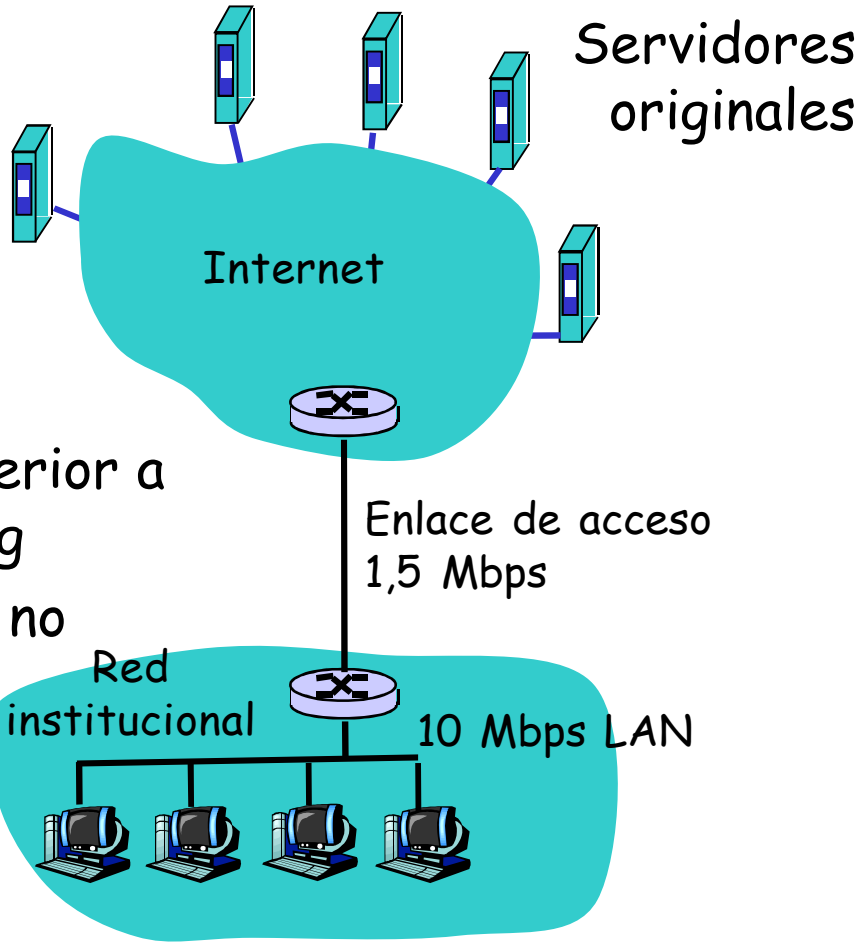
b) …HTTP no persistente con 5 conexiones en paralelo?

c) …1 única conexión HTTP persistente?

Pr7: Proxy (I)

Supongamos una universidad

- Tamaño objetos web = 100 Kbits
- Tasa de peticiones media de los navegadores a los "servidores web originales" = 15 peticiones/seg
- "Retardo Internet" del router superior a cualquier servidor "original" = 2 seg
- Las peticiones web son pequeñas y no generan retardo.



En consecuencia tenemos:

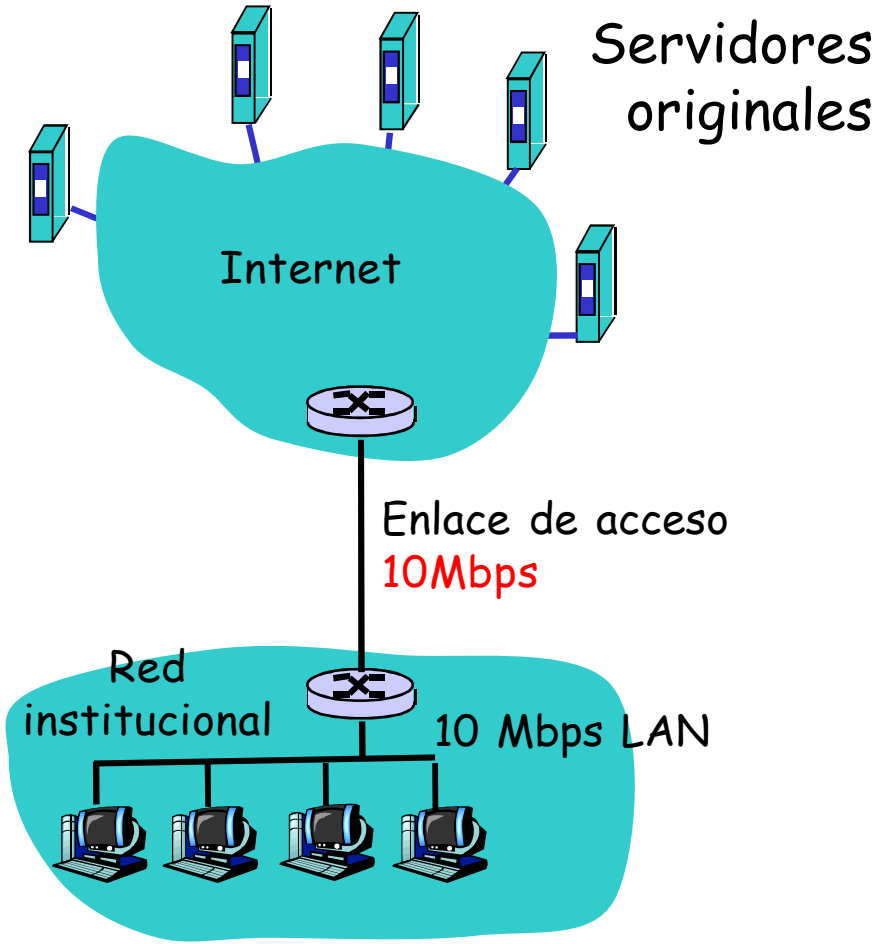
- Uso de la LAN = 15%
- Uso del enlace = 100%
- Al tener un uso del enlace del 100% (La/R ~ 1) las colas pueden crecen indefinidamente (y con ellas el retardo de cola).

Pr7: Proxy (II)

Propuesta de solución:

- Aumentar ancho de banda del enlace a 10 Mbps

- ¿Uso de la LAN ?
- ¿Uso del enlace?
- ¿Retardo total medio?
- ¿Comentarios?



Pr7: Proxy (III)

Otra posible solución:

- Instalar proxy-caché
- Supongamos tasa éxito 0.4:
  - 40% peticiones satisfechas inmediatamente
  - 60% peticiones satisfechas por el servidor original

- ¿Uso de la LAN ?
- ¿Uso del enlace?
- ¿Retardo total medio?
- ¿Comentarios?