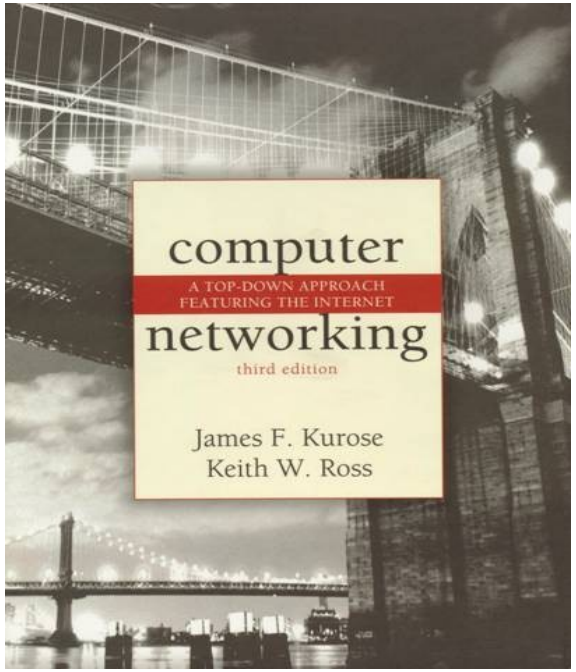
Chapter 2
Application Layer



Computer Networking: A Top Down Approach Featuring the Internet, 3rd edition.
Jim Kurose, Keith Ross
Addison-Wesley, July 2004.

A note on the use of these ppt slides:

We're making these slides freely available to all (faculty, students, readers). They're in PowerPoint form so you can add, modify, and delete slides (including this one) and slide content to suit your needs. They obviously represent a *lot* of work on our part. In return for use, we only ask the following:

- If you use these slides (e.g., in a class) in substantially unaltered form, that you mention their source (after all, we'd like people to use our book!)
- If you post any slides in substantially unaltered form on a www site, that you note that they are adapted from (or perhaps identical to) our slides, and note our copyright of this material.

Thanks and enjoy! JFK/KWR

Capa de Aplicación

2.1 Principios de aplicaciones de red

2.2 Web y HTTP

2.3 FTP

2.4 Correo electrónico

SMTP, POP3, IMAP

2.5 DNS

2.6 Archivos compartidos con P2P (*peer-to-peer*)

2.7 Programación con Sockets TCP

2.8 Programación con Sockets UDP

2.9 Creación de un Web server

Capa de Aplicación

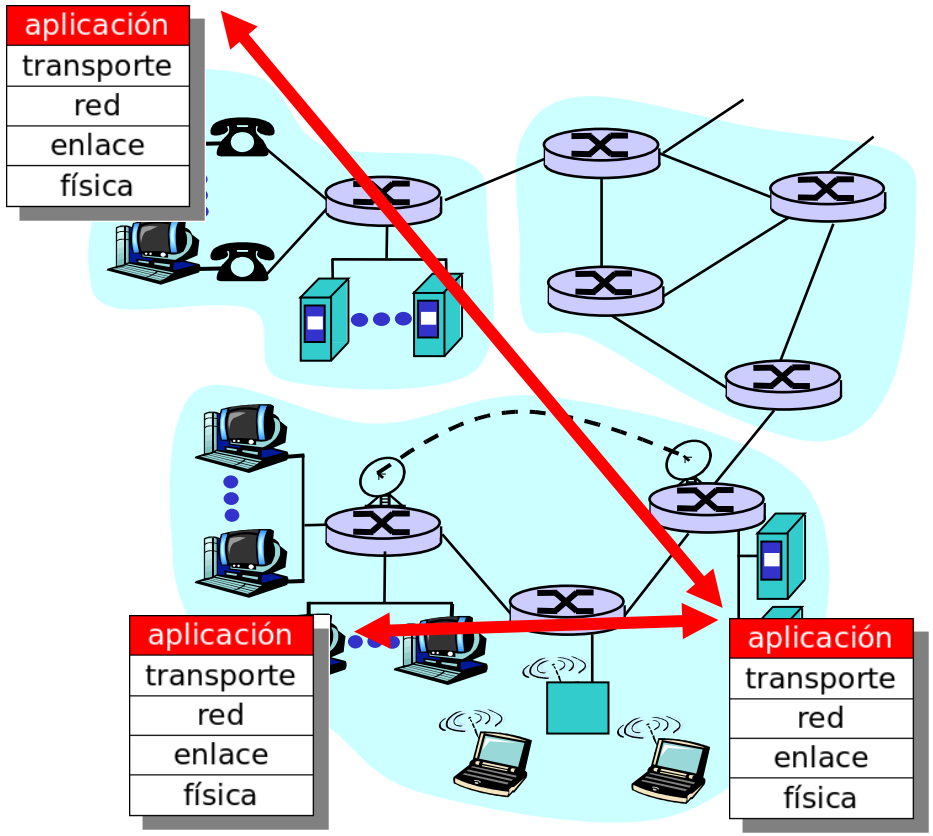
- Metas
  - Aspectos conceptuales y de implementación de protocolos de aplicación de redes
    - Modelos de servicio de la capa de transporte
    - Paradigma client-server
    - Paradigma peer-to-peer
- Aprender sobre protocolos examinando algunos conocidos, de nivel de aplicación
  - HTTP
  - FTP
  - SMTP / POP3 / IMAP
  - DNS
- Programación de aplicaciones de red
  - API de Sockets

Algunas aplicaciones de red

- E-mail
- Web
- Mensajería Instantánea (IM)
- Login remoto
- P2P file sharing
- Juegos de red multiusuario
- *Streaming* de video clips almacenados
- Telefonía Internet
- Video conferencia en tiempo real
- Computación paralela masiva

Creando aplicaciones de red

- Escribir programas que
  - Corran en diferentes sistemas finales y se comuniquen a través de una red
  - P. ej. Web: El servidor Web se comunica con el navegador
  - No para objetos del núcleo
  - Los dispositivos del *core* no funcionan a nivel de aplicación
  - Este diseño permite desarrollo rápido de aplicaciones



Capa de Aplicación

2.1 Principios de aplicaciones de red

2.2 Web y HTTP

2.3 FTP

2.4 Correo electrónico

SMTP, POP3, IMAP

2.5 DNS

2.6 Archivos compartidos con P2P (*peer-to-peer*)

2.7 Programación con Sockets TCP

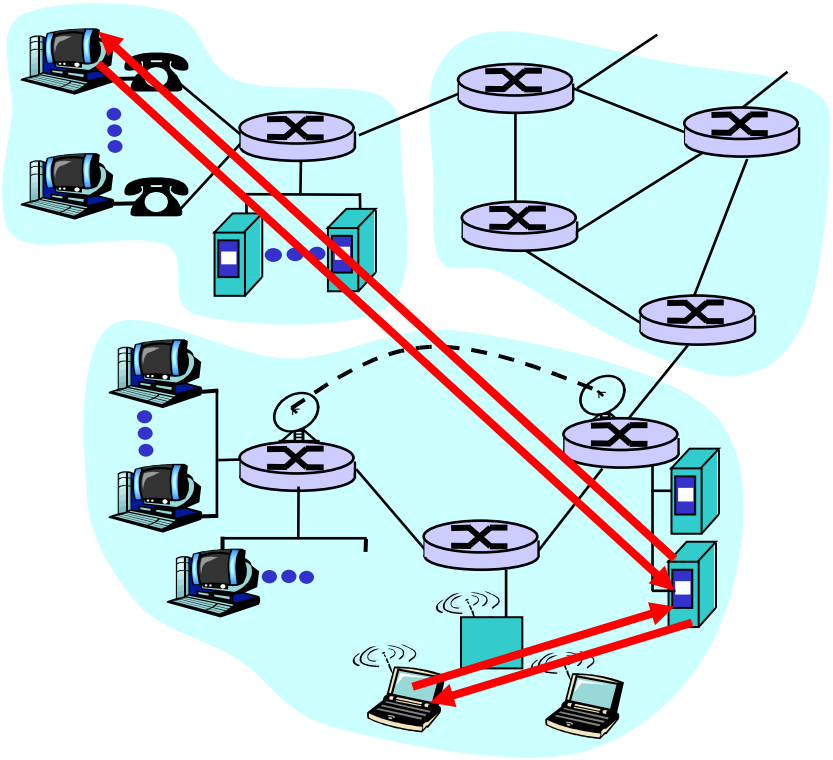
2.8 Programación con Sockets UDP

2.9 Creación de un Web server

Arquitecturas de Aplicación

- Cliente-servidor (*client-server*)
- *Peer-to-peer* (P2P)
- Híbridas de ambas anteriores

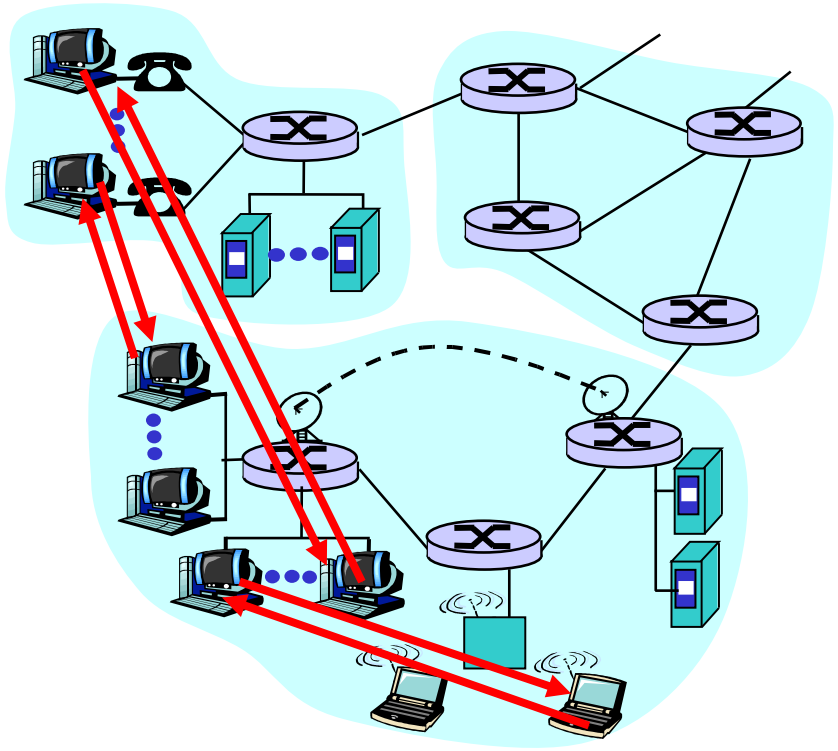
Arquitectura Client-server



- Servidor
  - Host siempre activo
  - Dirección IP permanente
  - Colonias (*farms*) para escalabilidad
- Clientes
  - Se comunican con el servidor
  - Pueden estar conectados intermitentemente
  - Pueden tener direcciones IP dinámicas
  - No se comunican entre sí

Arquitectura P2P pura

- Sin servidor siempre activo
- Sistemas finales arbitrarios se comunican directamente
- Los peers se conectan intermitentemente y cambian sus direcciones IP
- Ejemplo: Gnutella
- Altamente escalable
- Pero difícil de administrar



Híbridos de C-S y P2P

Napster

- Transferencia de archivos P2P
- Búsqueda de archivos centralizada
  - Los peers registran sus contenidos en un servidor central
  - Los peers consultan a ese servidor central para localizar contenidos

Mensajería Instantánea

- El chat entre dos usuarios es P2P
- Detección y localización de usuarios es centralizada
  - El usuario registra su IP con el servidor central al conectarse
  - Consulta al servidor central para encontrar IPs de sus contactos

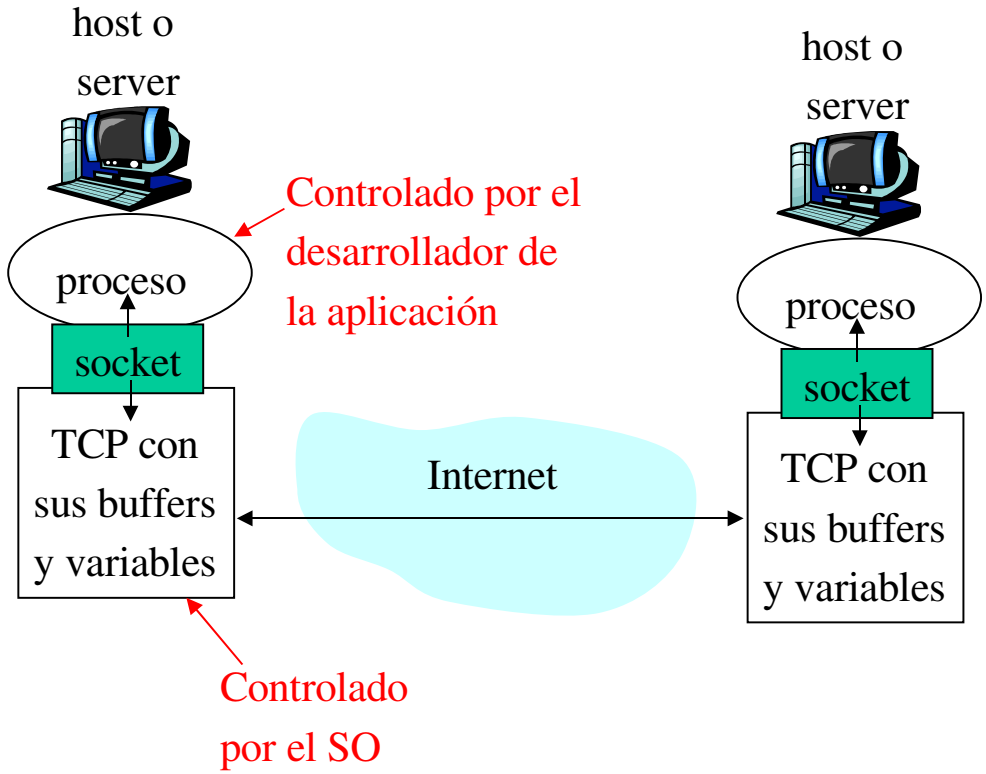
Procesos que se comunican

- Proceso
  - Programa que corre en un host
- Comunicaciones
  - Dentro del mismo host, dos procesos se comunican por IPC (definida por el SO)
  - Procesos en diferentes hosts se comunican intercambiando mensajes

- Proceso cliente
  - Un proceso que inicia la comunicación
- Proceso servidor
  - Proceso que espera ser contactado
- Aplicaciones P2P tienen procesos cliente y procesos servidor

Sockets

- Cada proceso envía/recibe mensajes a/desde su socket
- El socket es análogo a una puerta
  - El proceso emisor pasa el mensaje por la puerta
  - Descansa en la infraestructura de transporte del otro lado de la puerta para que lleve el mensaje al socket del proceso receptor



- La API provee: (1) elección del protocolo de transporte y (2) capacidad de fijar algunos parámetros

Direccionamiento de procesos

- Para que un proceso reciba mensajes debe tener un identificador
- Un host tiene una dirección IP de 32 bits única
  - Preg. ¿Es suficiente la dirección IP del host donde corre el proceso, para identificarlo?
  - Resp. No, puede haber corriendo muchos procesos en el mismo host
- El identificador incluye la dirección IP y un número de puerto (port number) asociado con el proceso en el host
- Ejemplos de números de port:
  - HTTP server: 80
  - Mail server: 25
- Más, luego

Un protocolo de aplicación define

- Tipos de mensajes intercambiados, como requerimientos (request) y respuestas (response)
- Sintaxis de tipos de mensajes
  - Qué campos llevan y cómo se delimitan
- Semántica de los campos
  - Significado de la información en ellos
- Reglas para cuándo y como los procesos envían y responden mensajes

- Protocolos de dominio público
  - Definidos en RFCs
  - Permiten la interoperabilidad
  - HTTP, SMTP
- Protocolos proprietarios
  - P. ej. KaZaA

Servicios de transporte requeridos por las aplicaciones

- Pérdida de datos
  - Algunas aplicaciones (audio) toleran algo de pérdida
  - Otras (file transfer, telnet) requieren transferencias de datos 100% confiables
- Ancho de banda (*bandwidth*)
  - Algunas aplicaciones (multimedia) requieren una cantidad de BW mínimo para ser “efectivas”
  - Otras (*“aplicaciones elásticas”*) usan lo que reciban
- Temporización (*timing*)
  - Algunas aplicaciones (telefonía Internet, juegos interactivos) requieren bajo *delay* para ser “efectivas”

Servicios de transporte requeridos por las aplicaciones

Aplicación	Pérdida	Bandwidth	Sensible al *timing*
transferencia de archivos	sin pérdida	elástica	no
e-mail	sin pérdida	elástica	no
documentos Web	sin pérdida	elástica	no
audio/video en tiempo real	tolerante	audio: 5kbps-1Mbps video:10kbps-5Mbps	sí, 100's ms
audio/video almacenado	tolerante	igual que anterior	sí, pocos s
juegos interactivos	tolerante	desde algunos kbps	sí, 100's ms
mensajería instantánea	sin pérdida	elástica	sí y no

Protocolos de transporte de Internet y servicios

Servicio TCP

- Orientado a conexión
  - Se requiere acuerdo previo entre procesos cliente y servidor
- Transporte confiable entre procesos emisor y receptor
- Control de flujo: el emisor no desbordará al receptor
- Control de congestión: el emisor para cuando la red está sobrecargada
- No provee *timing* ni garantías de acho de banda

Servicio UDP

- Transferencia de datos no confiable entre procesos emisor y receptor
- No provee establecimiento de conexión, confiabilidad, control de flujo, control de congestión, *timing* ni garantías de ancho de banda
- ¿Y entonces? ¿Para qué sirve UDP?

Aplicaciones de Internet y protocolos de transporte

Aplicación	Protocolo de aplicación	Prot. de transporte subyacente
e-mail	SMTP [RFC 2821]	TCP
acceso a terminal remota	Telnet [RFC 854]	TCP
Web	HTTP [RFC 2616]	TCP
transferencia de archivos	FTP [RFC 959]	TCP
streaming multimedia	propietario (p.ej. RealNetworks)	TCP o UDP
telefonía Internet	propietario (p.ej. Dialpad)	típicamente UDP

Capa de Aplicación

2.1 Principios de aplicaciones de red

2.2 Web y HTTP

2.3 FTP

2.4 Correo electrónico

SMTP, POP3, IMAP

2.5 DNS

2.6 Archivos compartidos con P2P (*peer-to-peer*)

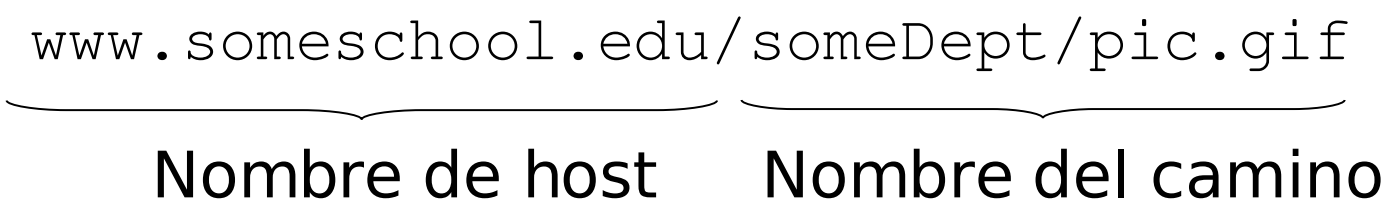
2.7 Programación con Sockets TCP

2.8 Programación con Sockets UDP

2.9 Creación de un Web server

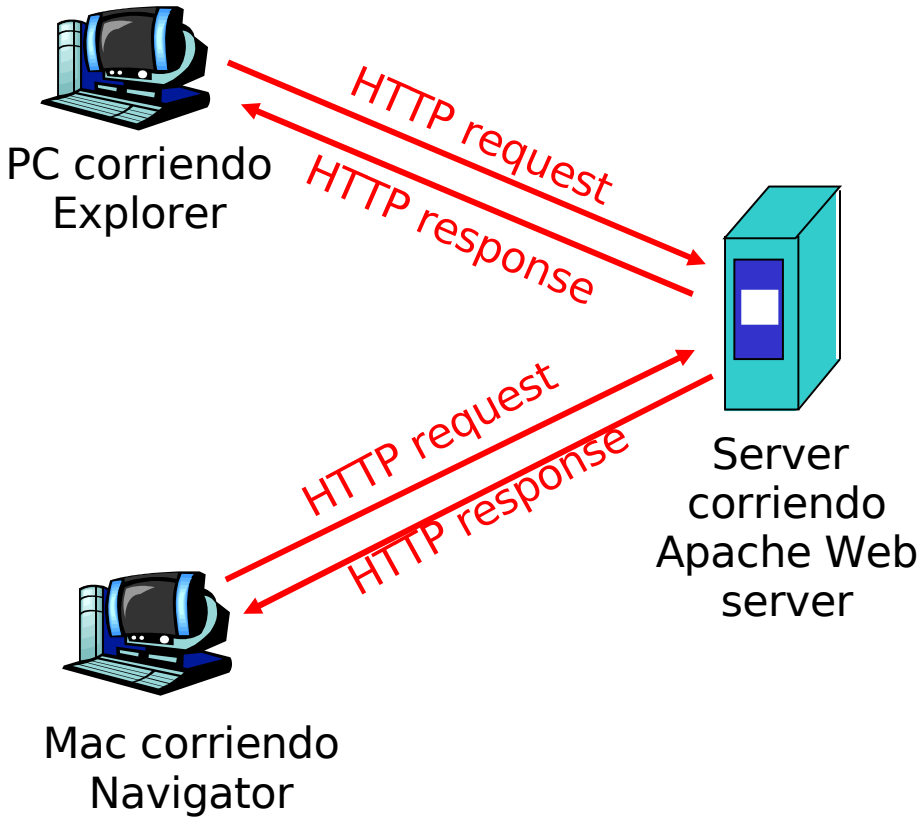
Web y HTTP

- Un poco de vocabulario
  - Una página Web consiste de objetos
  - Un objeto puede ser un archivo HTML, una imagen JPEG, un applet en Java, un archivo de audio...
  - Una página Web consiste de un archivo base HTML que incluye varios objetos referenciados
  - Cada objeto es direccionable por un URL
  - URL ejemplo:



HTTP

- HTTP: HyperText Transfer Protocol
  - Protocolo de aplicación de la Web
  - Modelo client/server
  - client: navegador (*browser*) que requiere, recibe y “visualiza” objetos Web
  - server: envía objetos en respuesta a requerimientos
- HTTP 1.0: RFC 1945
- HTTP 1.1: RFC 2068



HTTP

- Usa a TCP
  - El cliente inicia una conexión TCP (creando un socket) hacia el server, port 80
  - El server acepta la conexión TCP del cliente
  - Se intercambian mensajes HTTP (de nivel de aplicación) emtre el browser y el servidor
  - Se cierra la conexión TCP

- HTTP es *sin estado* (“*stateless*”)
  - El servidor no mantiene información sobre requests anteriores de los clientes
  - Los protocolos que mantienen estado son complejos
    - Se debe mantener la historia pasada
    - Si el server o el cliente caen, sus vistas del “estado” puede ser inconsistentes y deben ser reconciliadas

Conexiones HTTP

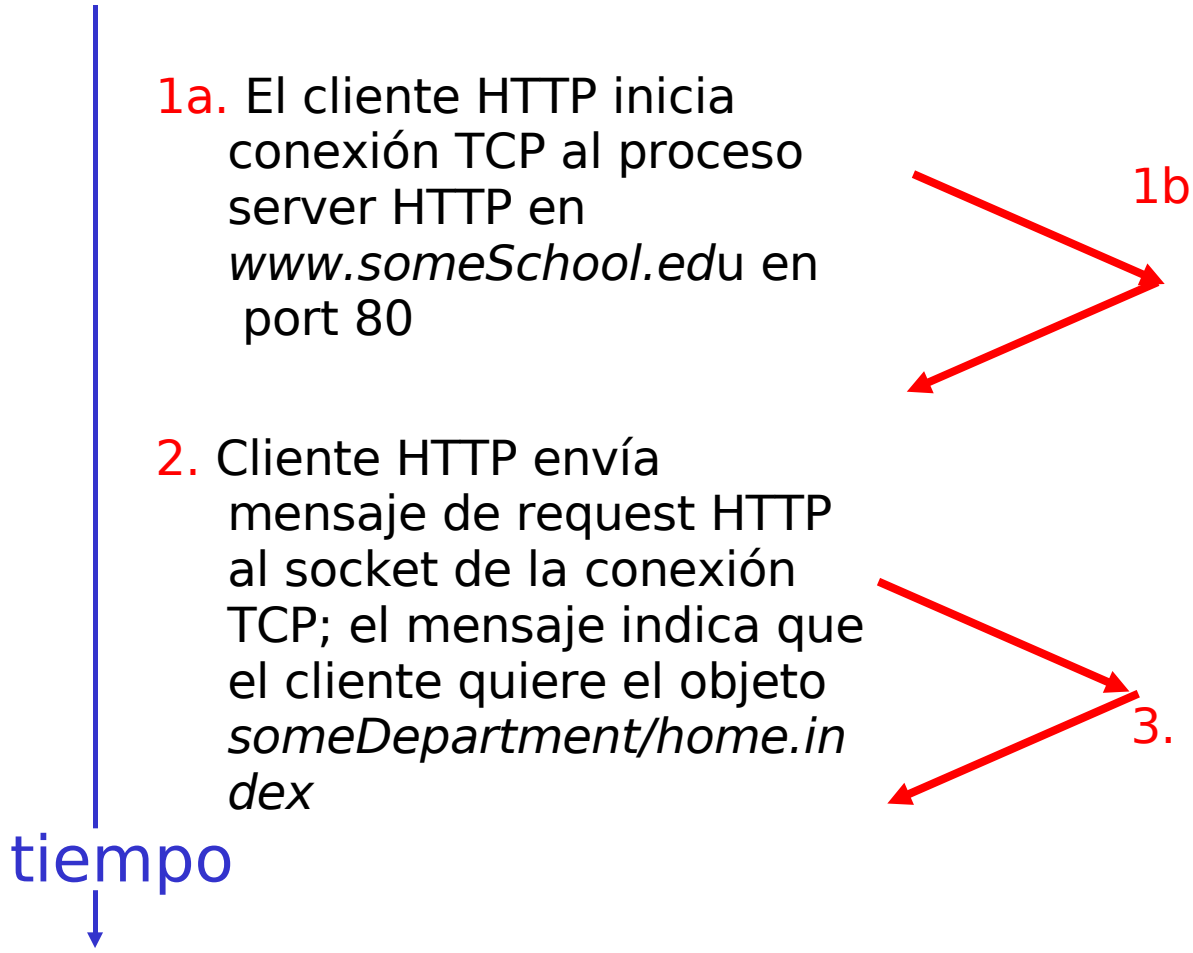
- HTTP no persistente
  - A lo sumo un objeto es enviado sobre una conexión TCP
  - HTTP/1.0 usa HTTP no persistente
- HTTP persistente
  - Se pueden enviar múltiples objetos sobre una misma conexión TCP entre cliente y servidor
  - HTTP/1.1 usa conexiones persistentes en modo default

HTTP no persistente

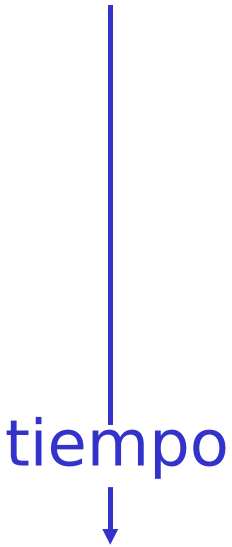
El usuario ingresa el URL

`www.someSchool.edu/someDepartment/home.index`

(contiene texto, referencias a 10 imágenes JPEG)



HTTP no persistente



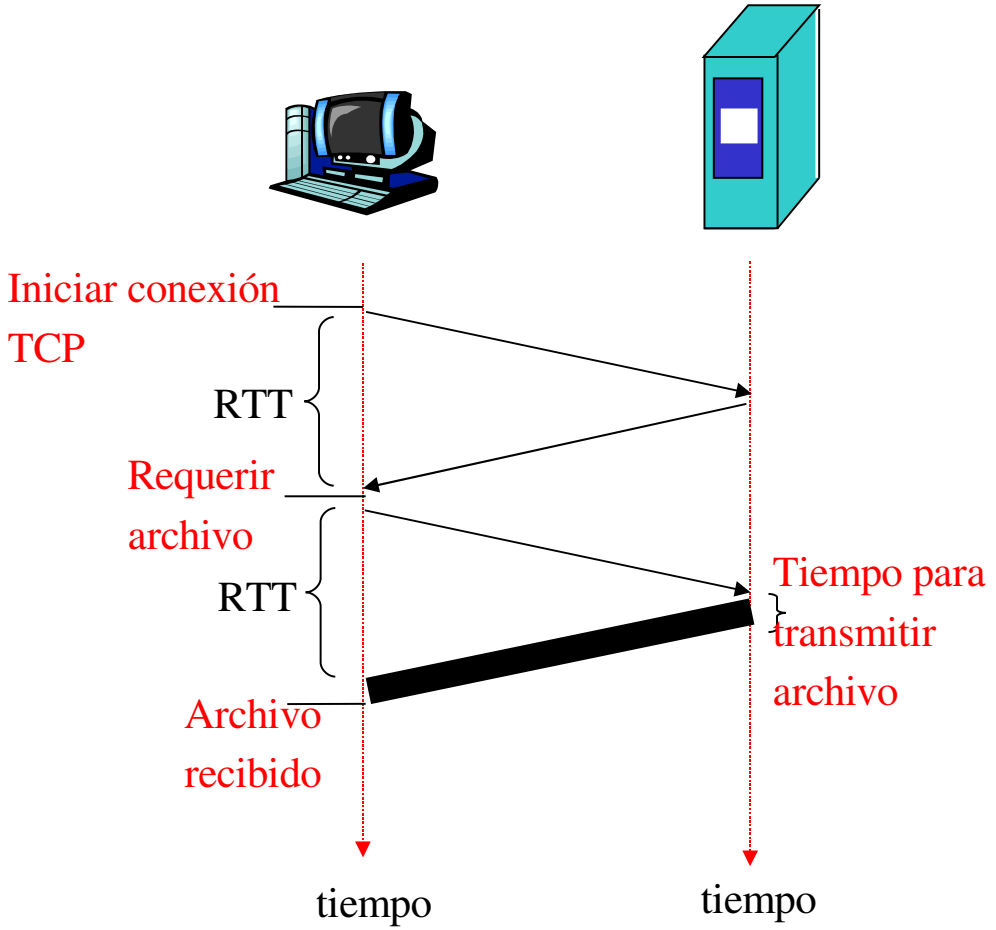
4. El server HTTP cierra la conexión TCP

5. El cliente HTTP client recibe mensaje de respuesta conteniendo archivo HTML, lo visualiza; al hacer el *parsing* del HTML encuentra 10 referencias a objetos JPEG

6. Se repiten los pasos 1 a 5 para cada uno de los 10 objetos JPEG

Modelo del tiempo de respuesta

- RTT (Round Trip Time)
  - tiempo para enviar un paquete pequeño de cliente a servidor ida y vuelta
- Tiempo de respuesta
  - Un RTT para iniciar la conexión
  - Un RTT para el request HTTP y para que vuelvan los primeros bytes de la respuesta HTTP
- Transmisión de un archivo
  - 2RTT+tiempo de transmisión

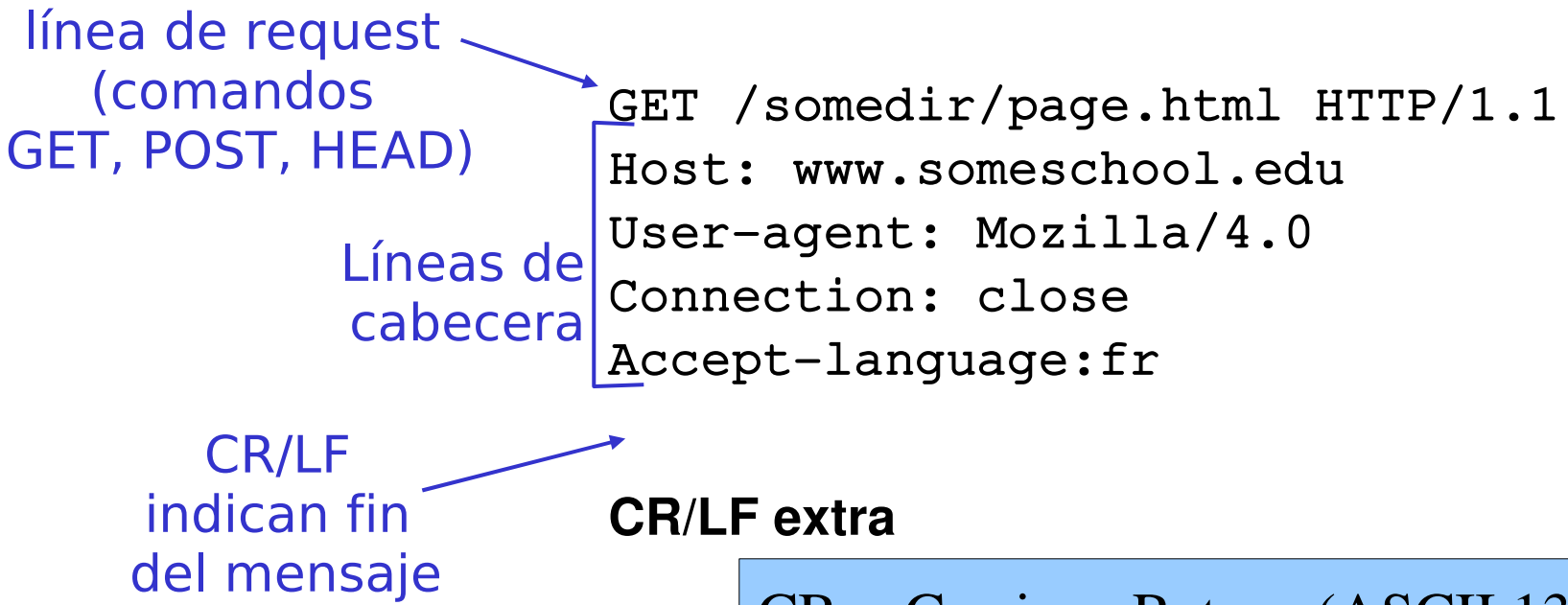


HTTP persistente

- HTTP No persistente:
  - Requiere 2 RTTs por objeto
  - El SO debe trabajar y asignar recursos para cada conexión TCP
  - Generalmente los browsers abren conexiones TCP paralelas
- HTTP persistente:
  - El server deja la conexión abierta luego de enviar cada respuesta
  - Los siguientes mensajes HTTP entre ese cliente y servidor van sobre la misma conexión
- Persistente sin pipelining
  - El cliente envía nuevo request sólo cuando se ha recibido la respuesta anterior
  - Un RTT para cada objeto referenciado
- Persistente con pipelining
  - Default en HTTP/1.1
  - El cliente envía requests apenas encuentra un objeto referenciado
  - Al menos un RTT por todos los objetos referenciados

Mensaje de request HTTP

- Dos tipos de mensajes HTTP: *request, response*
- Mensaje de request HTTP:
  - ASCII (formato legible por humanos)

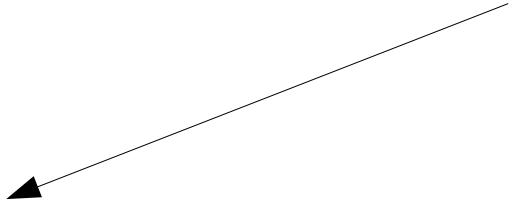


CR = Carriage Return (ASCII 13)
LF = Line Feed (ASCII 10)

Request HTTP, formato general

Entregando input al server

- Método POST
  - La página Web suele incluir ingreso de formularios
  - El input es ingresado al server en el cuerpo de una entidad
- Método URL
  - Utiliza el método GET de HTTP
  - El input es ingresado en el campo URL de la línea de request

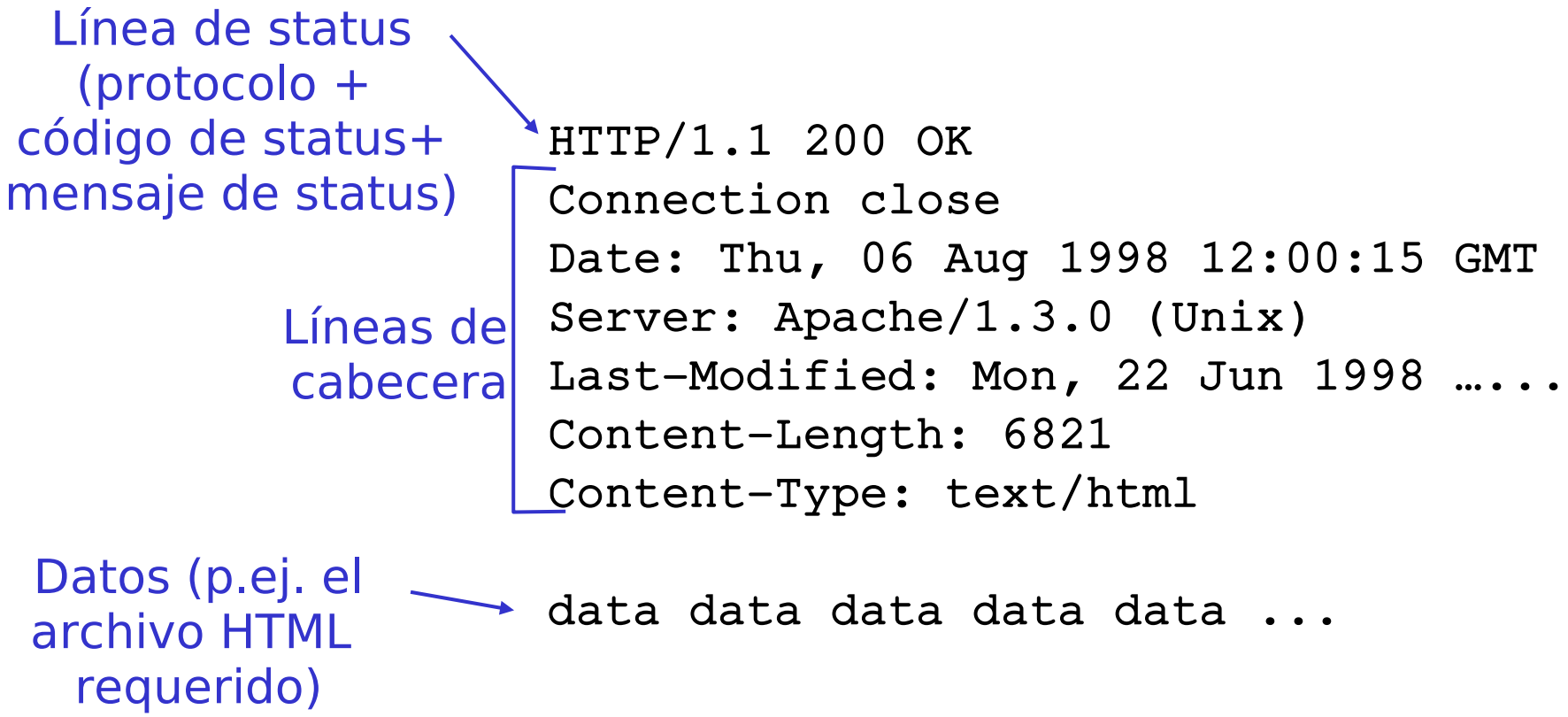


```
www.somesite.com/animalsearch?monkeys&banana
```

Tipos de métodos

- HTTP/1.0
  - GET
  - POST
  - HEAD
    - Pide al server que deje el objeto requerido fuera de la respuesta
- HTTP/1.1
  - GET, POST, HEAD
  - PUT
    - Sube un archivo en el cuerpo de entidad al camino especificado en el campo URL
  - DELETE
    - Borra el archivo especificado en el campo URL

Mensaje de respuesta HTTP



Códigos de status en la respuesta HTTP

- Aparecen en la primera línea de respuesta servidor -> cliente
- 200 OK
  - Solicitud exitosa, el objeto requerido viene a continuación en este mensaje
- 301 Moved Permanently
  - El objeto requerido ha sido desplazado, la nueva ubicación viene a continuación en este mensaje (Location:)
- 400 Bad Request
  - El servidor no comprendió el mensaje
- 404 Not Found
  - El servidor no posee el documento requerido
- 505 HTTP Version Not Supported
  - El servidor no soporta la versión especificada por el cliente

Probando HTTP desde el cliente

1. Telnet a un servidor HTTP

```
telnet cis.poly.edu 80
```

Abre conexión TCP al port 80 (port default para el server HTTP) en cis.poly.edu.
Lo que se tipee será enviado al port 80 en cis.poly.edu

2. Escribir un requerimiento GET de HTTP

```
GET /~ross/ HTTP/1.1
Host: cis.poly.edu
```

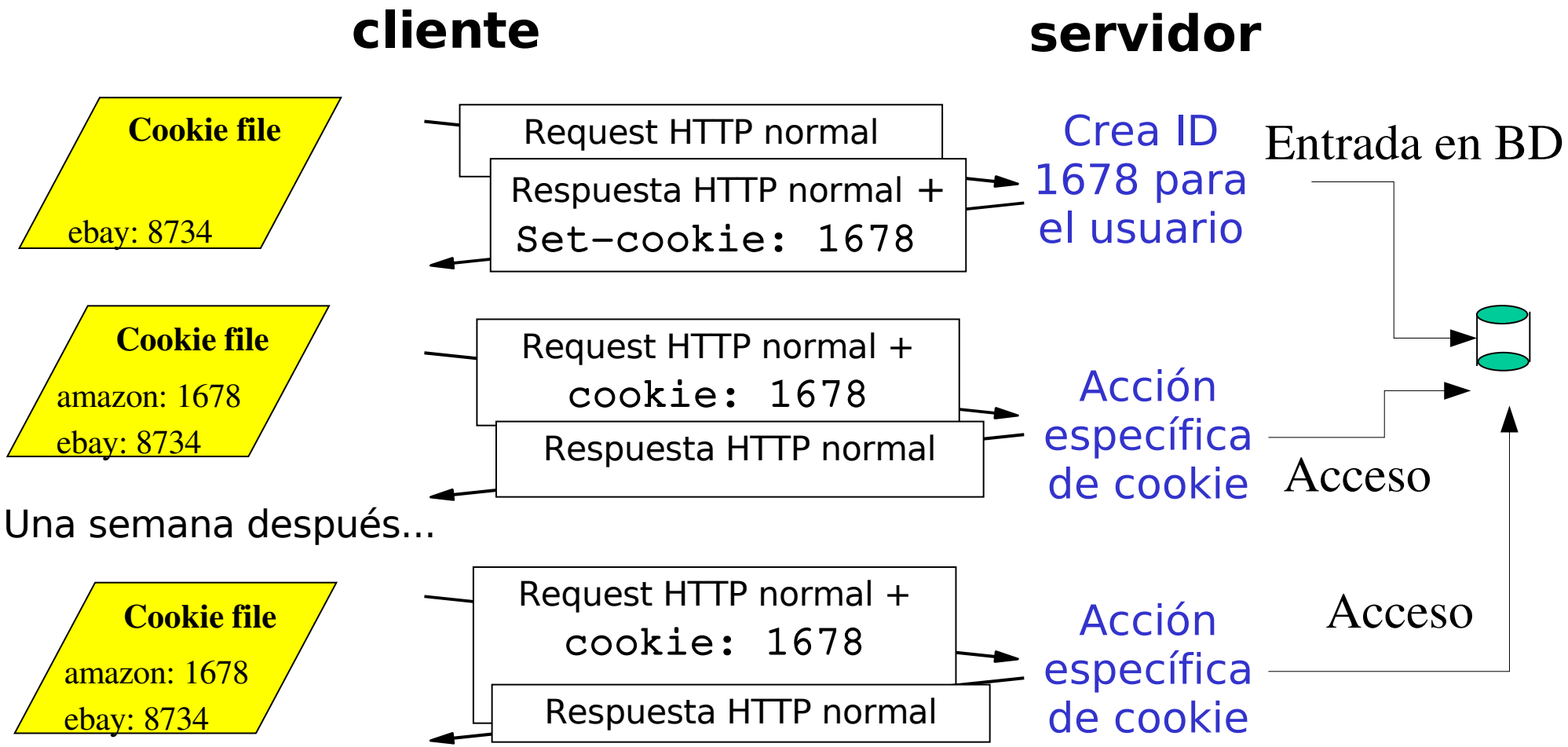
Tipeando esto (con ENTER dos veces) enviamos este requerimiento minimal, pero completo, al servidor HTTP

3. Visualizamos el mensaje de respuesta del servidor

Conservando estado con cookies

- Cuatro componentes
  - 1) Línea de cabecera de cookie en la respuesta HTTP
  - 2) Línea de cabecera de cookie en la respuesta HTTP
  - 3) Archivo cookie mantenido en el host del usuario y administrado por el browser
  - 4) Base de datos como *back-end* en el servidor Web
- Ejemplo
  - Usuario accede Internet siempre desde el mismo equipo
  - Visita un sitio de e-commerce por primera vez
  - Cuando los primeros requests HTTP llegan al sitio, el server crea un ID único y una entrada en la base de datos para este ID

Conservando estado con cookies



Cookies

- Las cookies pueden ofrecer
  - Autorización
  - Carrito de compras (*shopping carts*)
  - Recomendaciones
  - Estado de sesión de usuario (Web e-mail)

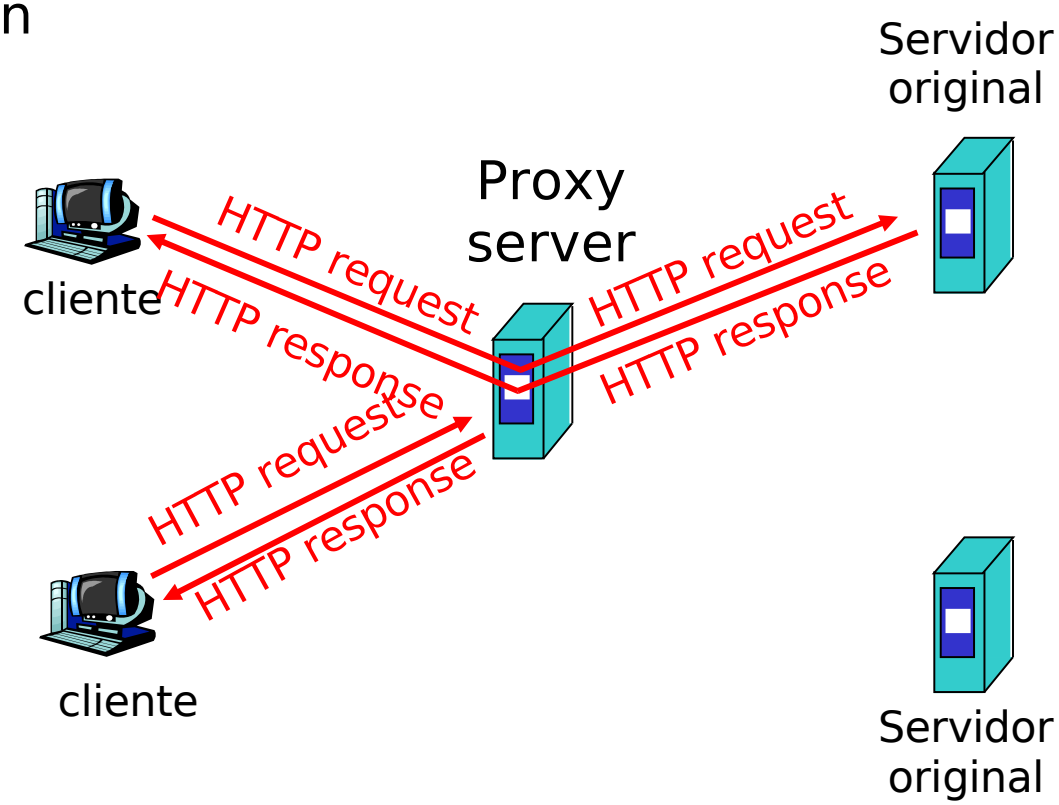
NOTA

- Cookies y privacidad
  - Las cookies permiten a los sitios conocer cosas sobre nosotros
  - Podemos proveer nombre y email a los sitios
  - Los motores de búsqueda usan redirección y cookies para obtener aun más conocimiento
  - Las compañías de publicidad recolectan info sobre varios sitios

Web caches (proxy server)

- Meta

- Satisfacer request del cliente sin involucrar al servidor original
- Usuario configura navegador
  - Accesos Web via cache
- Navegador envía todos los requests HTTP a la cache
  - Si objeto en cache, cache lo devuelve
  - Si no, cache requiere objeto al servidor original, y lo devuelve al cliente



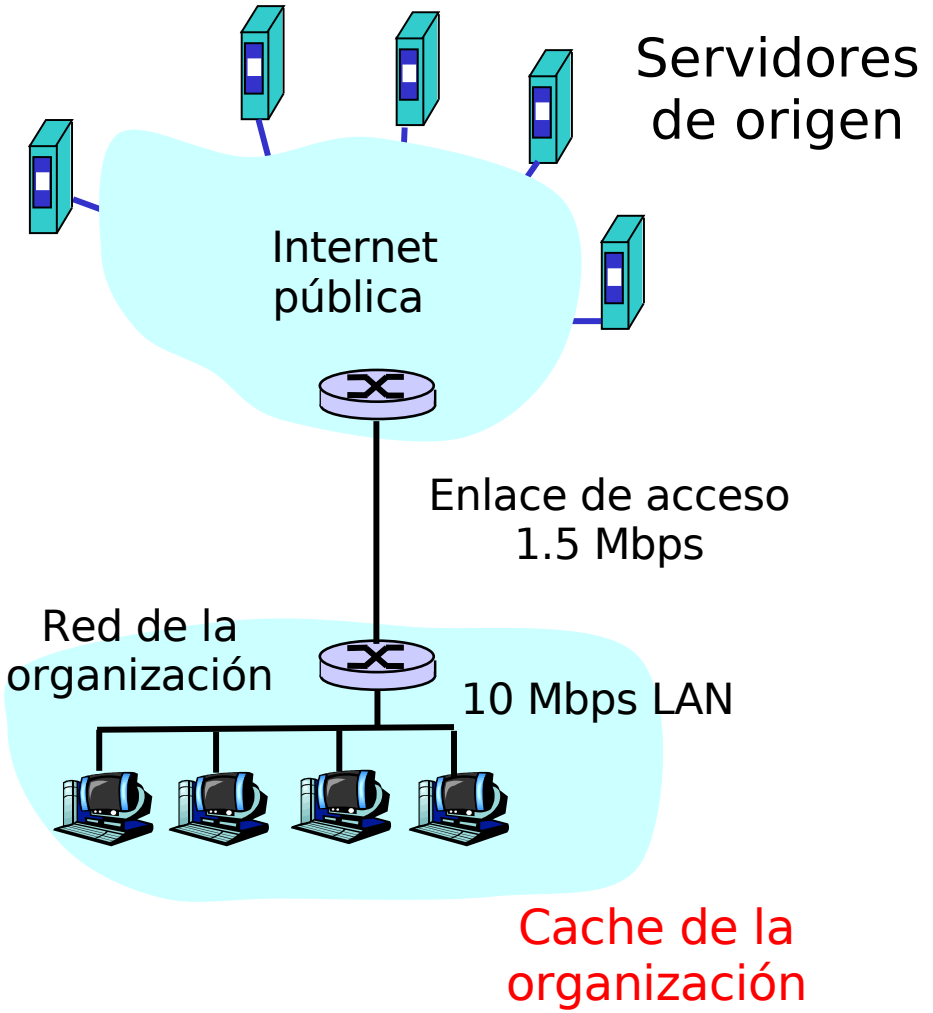
Más sobre Web cache

- La cache actúa tanto como cliente como servidor
- La cache es típicamente instalada y administrada por un ISP (universidad, empresa, ISP domiciliario)
- ¿Por qué usar cache de Web?

- Reducir tiempo de respuesta de requests del cliente
- Reducir tráfico sobre un enlace de acceso de la organización
- Poblar Internet de caches permite a los proveedores de contenido “pobres” hacer llegar efectivamente esos contenidos

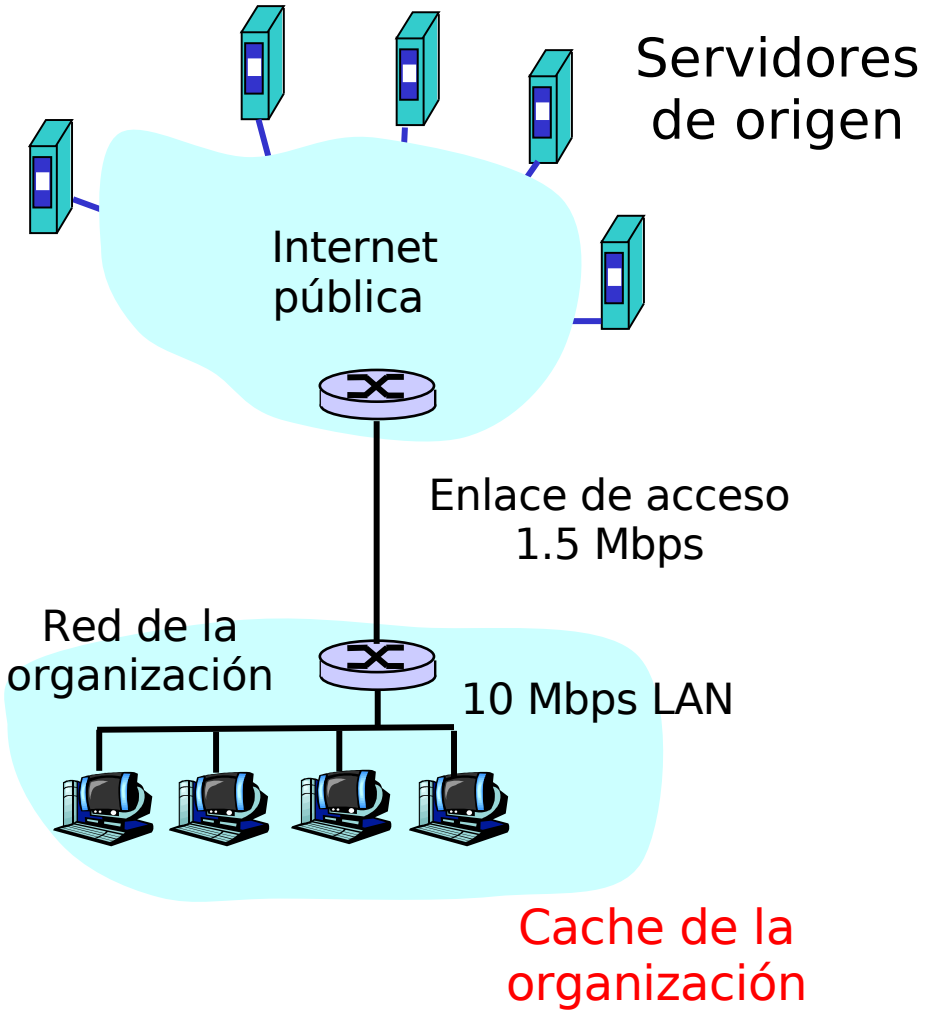
Ejemplo de Caching

- Supuestos
  - Tamaño de objeto promedio 100Kbits, tasa promedio de requests = 15/s, retardo ida y vuelta desde el router de la organización a cualquier servidor de origen= 2 s
- Consecuencias
  - Utilización de la LAN = 15%; del acceso = 100%
  - Retardo total = Delay de Internet + delay de acceso + delay LAN = 2 s + minutos + milisegundos



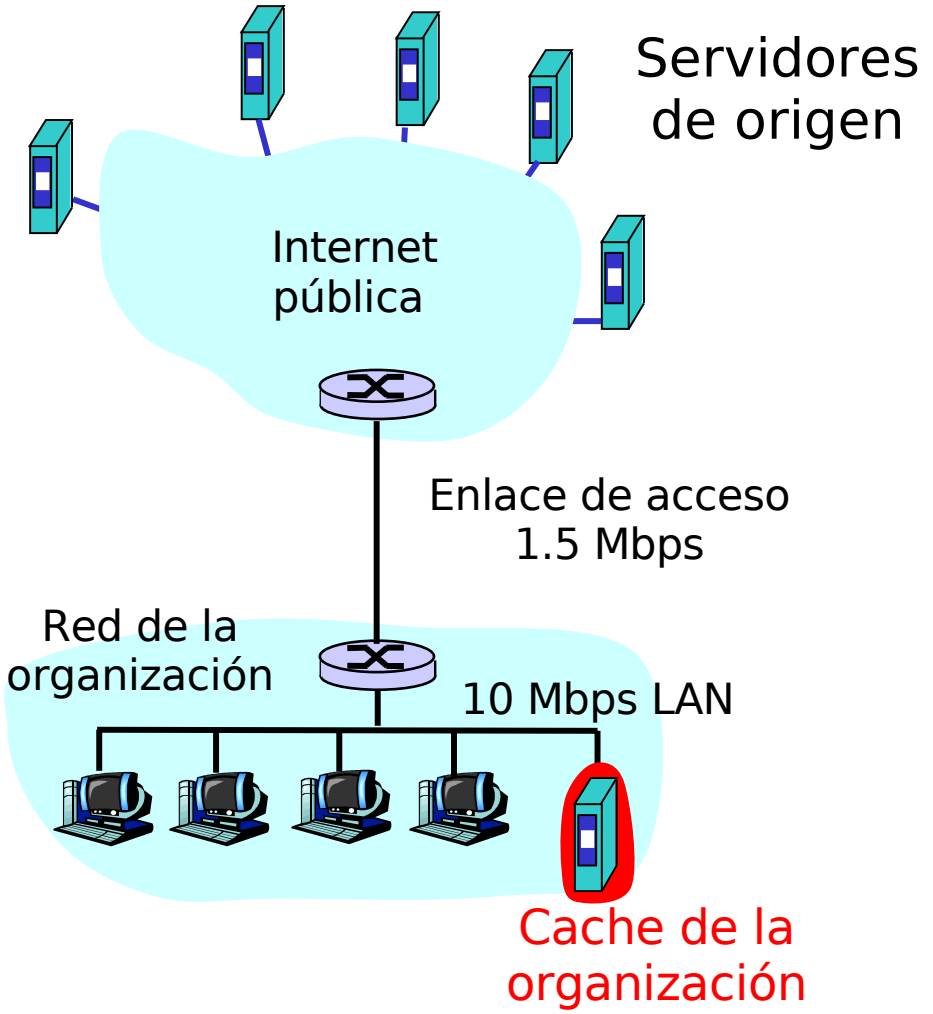
Ejemplo de Caching

- Solución posible
  - Incrementar BW del enlace de acceso a 10Mbps
- Consecuencias
  - Utilización de LAN = 15%
  - Utilización del acceso = 15%
  - Retardo total = Delay de Internet + delay de acceso + delay LAN = 2 s + milisegundos + milisegundos
  - Con frecuencia caro



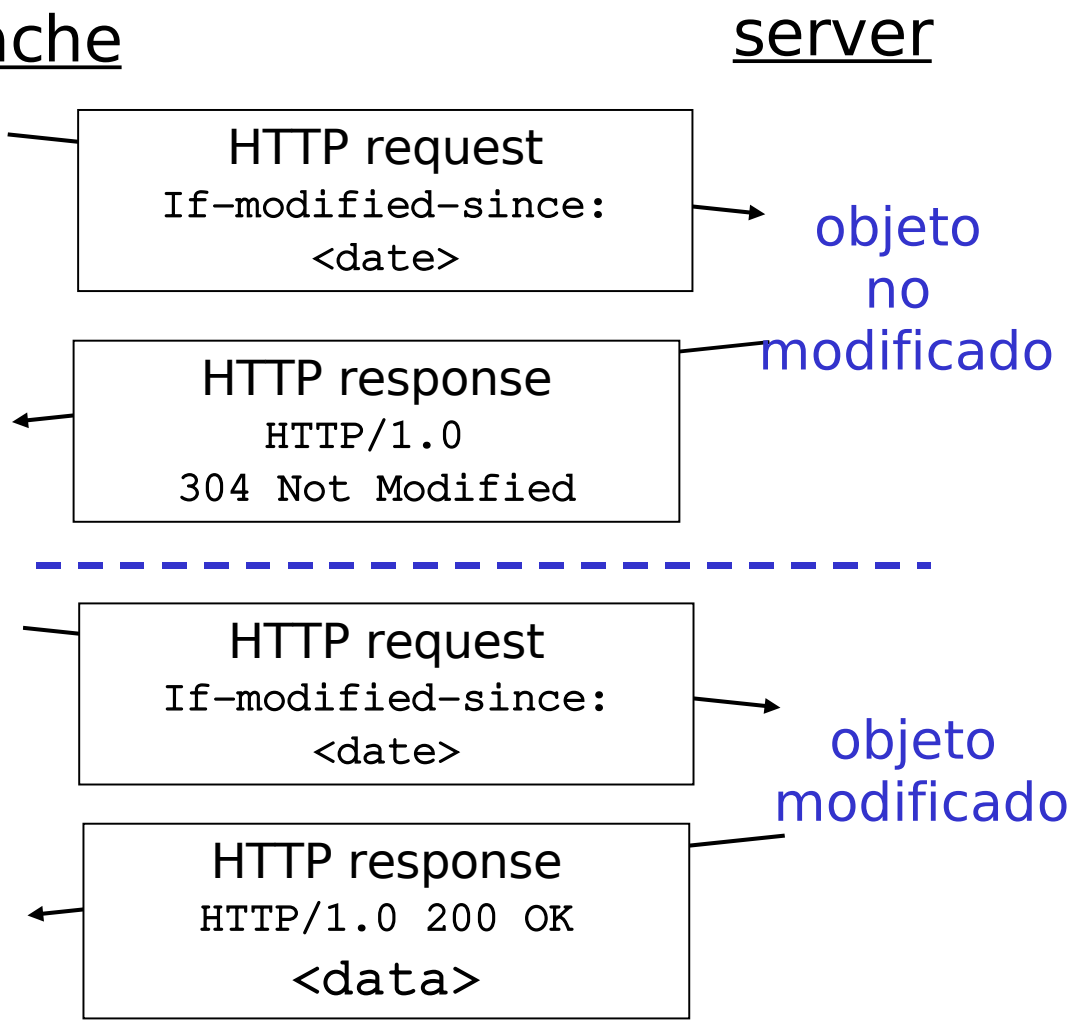
Ejemplo de Caching

- Instalando una cache
  - Supongamos hit ratio = 0.4
- Consecuencia
  - 40% de los Requests casi inmediatamente, 60% de los Requests por el server de origen
  - Utilización del enlace de acceso reducido al 60%, con retardos despreciables (en el orden de 10ms)
  - Retardo total = Delay de Internet + delay de acceso + delay LAN = .6*(2.01) s + milisegundos < 1.4 s



GET Condicional

- Objetivo
  - No enviar el objeto si la cache tiene versión actualizada almacenada
- Cache: especifica fecha de copia en cache en su request HTTP
  - `If-modified-since: <date>`
- Server: la respuesta no contiene objeto si la copia en cache está actualizada
  - `HTTP/1.0 304 Not Modified`



Capa de Aplicación

2.1 Principios de aplicaciones de red

2.2 Web y HTTP

2.3 FTP

2.4 Correo electrónico

SMTP, POP3, IMAP

2.5 DNS

2.6 Archivos compartidos con P2P (*peer-to-peer*)

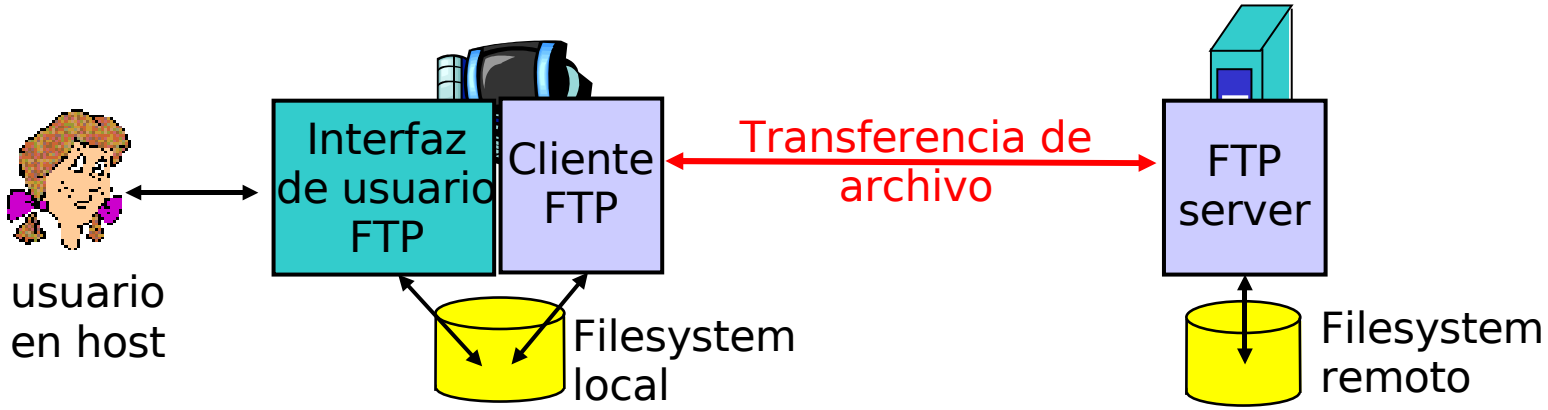
2.7 Programación con Sockets TCP

2.8 Programación con Sockets UDP

2.9 Creación de un Web server

FTP, File Transfer Protocol

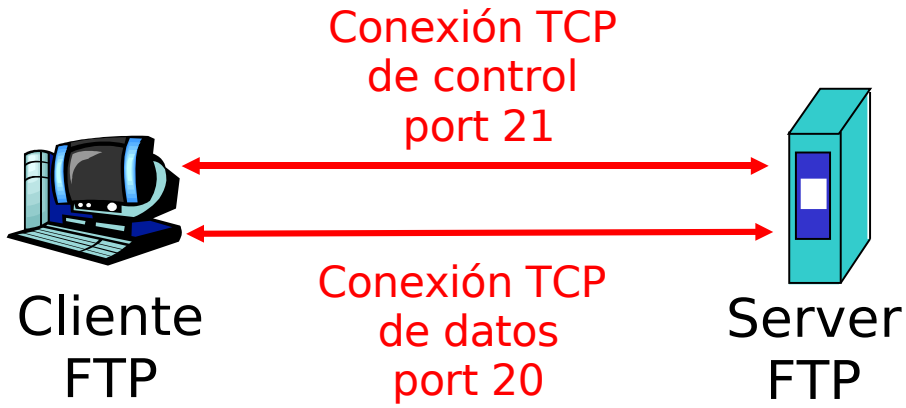
- Transferir archivos desde/hacia host remoto
- Modelo Cliente/Servidor
  - *Cliente:* inicia la transferencia
  - *Servidor:* host remoto
- FTP: RFC 959
- Servidor FTP: port 21



FTP, conexiones de datos y de control

- El cliente FTP contacta al server sobre el port 21, especificando TCP como protocolo de transporte
- Cliente obtiene autorización sobre la conexión de control
- Cliente navega el directorio remoto enviando comandos sobre la conexión de control
- Al recibir un comando de file transfer, el server abre conexión TCP de datos hacia el cliente
- Luego de transferir un archivo, el server cierra la conexión de datos

- Server abre una segunda conexión de datos para transferir otro archivo
- Conexión de control “fuera de banda”
- El server FTP mantiene “estado”: directorio activo, autenticación



Comandos y respuestas FTP

Comandos

- Enviados como texto ASCII sobre el canal de control
- USER username
- PASS password
- LIST devuelve listado de archivos en directorio activo
- RETR filename obtiene archivo
- STOR filename almacena archivo en host remoto

Códigos de retorno

- Código y frase de status (como en HTTP)
- `331 Username OK, password required`
- `125 data connection already open; transfer starting`
- `425 Can't open data connection`
- `452 Error writing file`

Capa de Aplicación

2.1 Principios de aplicaciones de red

2.2 Web y HTTP

2.3 FTP

2.4 Correo electrónico

SMTP, POP3, IMAP

2.5 DNS

2.6 Archivos compartidos con P2P (*peer-to-peer*)

2.7 Programación con Sockets TCP

2.8 Programación con Sockets UDP

2.9 Creación de un Web server

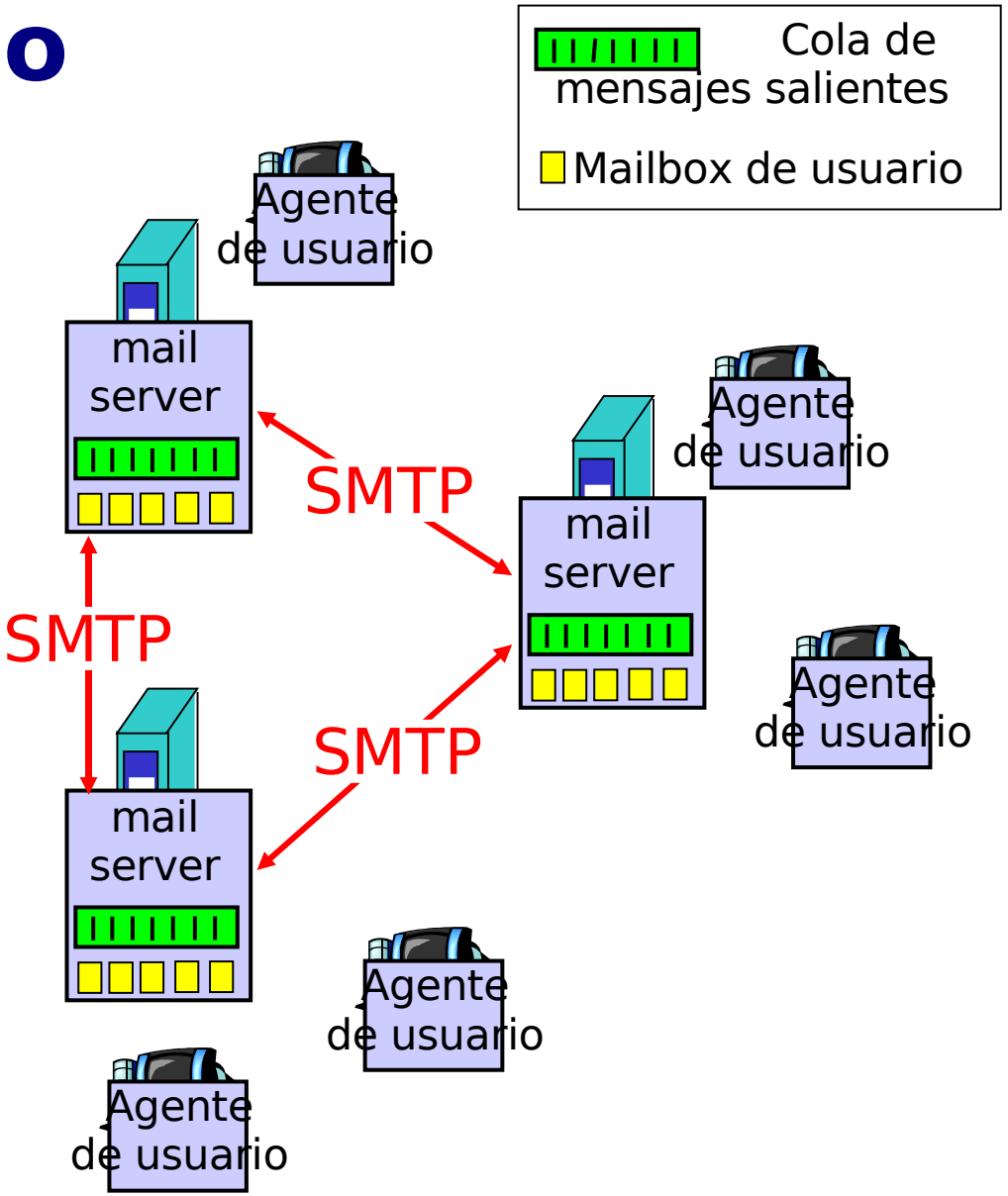
Correo electrónico

Tres componentes principales

- Agentes de usuario
- Servidores de mail
- Simple Mail Transfer Protocol: SMTP

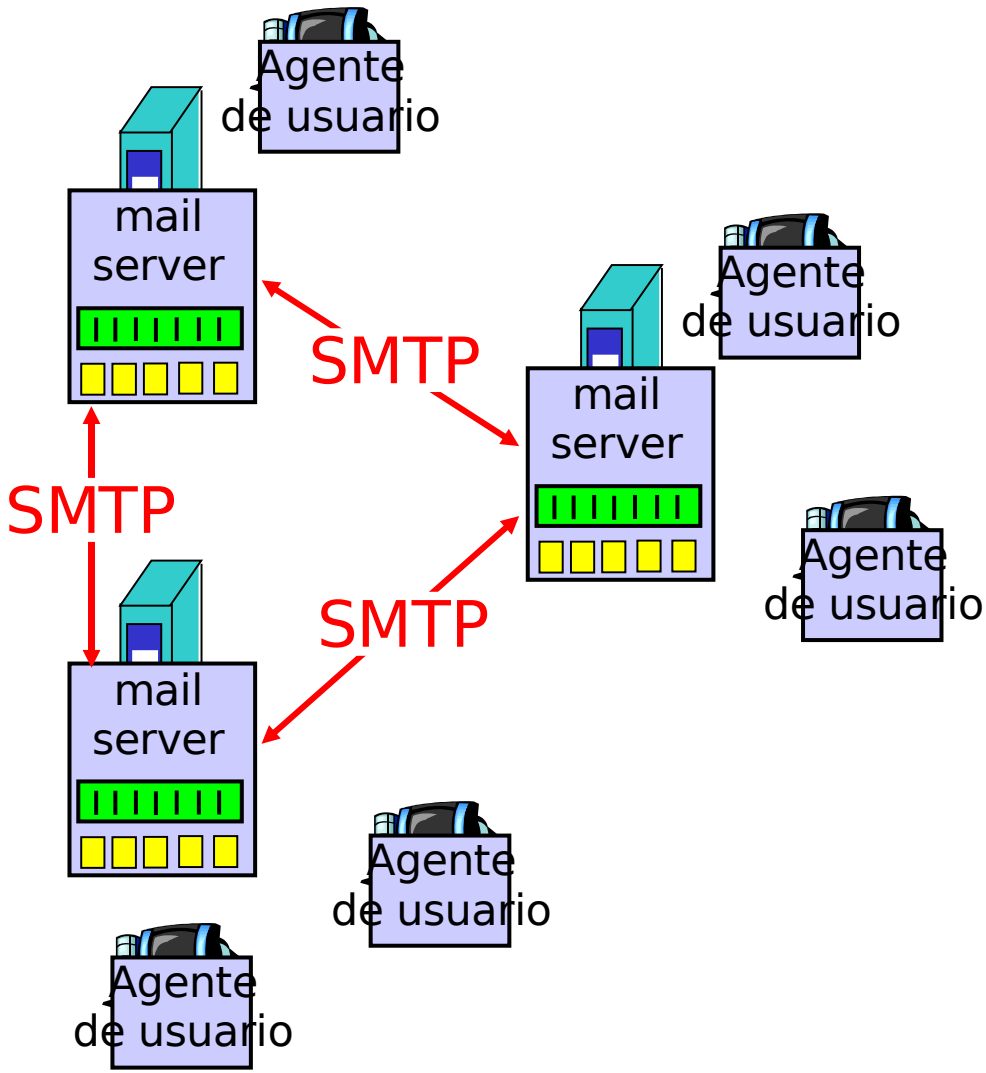
Agente de usuario

- Lector de mail
- Componer, editar leer mensajes de correo
- P.ej. Eudora, Outlook, elm, Netscape Messenger
- Los mensajes salientes y entrantes se almacenan en el server



Servidores de mail

- Mailbox contiene los mensajes entrantes para un usuario
- Cola de mensajes contiene mensajes de mail salientes (a ser enviados)
- Protocolo SMTP entre servidores de mail para enviar mensajes
  - Cliente: el mail server que envía
  - Server: el mail server receptor

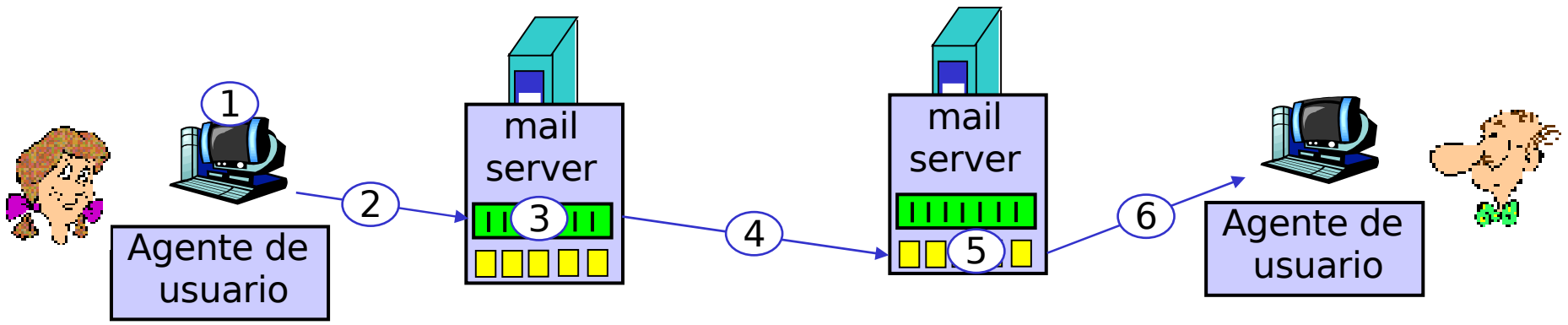


SMTP [RFC 2821]

- Usa TCP para transferir mensajes de mail en forma confiable al servidor, port 25
- Transferencia directa: el server que envía al que recibe
- Tres fases de transferencia
  - Handshaking (saludo)
  - Transferencia de mensajes
  - Cierre
- Interacción comando/respuesta
  - Comandos: texto ASCII
  - Respuestas: código y frase de status
- Los mensajes deberán estar en ASCII 7 bits

Usuario A envía mensaje a B

1) A usa un agente de usuario para componer un mensaje a `bob@someschool.edu`
2) El agente de usuario de A envía un mensaje a su mail server, el mensaje queda en cola
3) El cliente SMTP abre conexión TCP con el mail server de B
4) El cliente SMTP envía el mensaje de A sobre la conexión TCP
5) El mail server de B pone el mensaje en la mailbox de B
6) B invoca a su agente de usuario para leer el mensaje



Interacción SMTP ejemplo

```
S: 220 hamburger.edu
C: HELO crepes.fr
S: 250  Hello crepes.fr, pleased to meet you
C: MAIL FROM: <alice@crepes.fr>
S: 250 alice@crepes.fr... Sender ok
C: RCPT TO: <bob@hamburger.edu>
S: 250 bob@hamburger.edu ... Recipient ok
C: DATA
S: 354 Enter mail, end with "." on a line by itself
C: Do you like ketchup?
C: How about pickles?
C: .
S: 250 Message accepted for delivery
C: QUIT
S: 221 hamburger.edu closing connection
```

Probando la interacción SMTP

- Sin necesidad de usar un lector o cliente de email
  - Desde una consola ponemos `telnet servername 25`
  - El server responde `220 Ok`
  - Ingresamos comandos
    - `HELO,`
    - `MAIL FROM,`
    - `RCPT TO,`
    - `DATA,`
    - `QUIT`

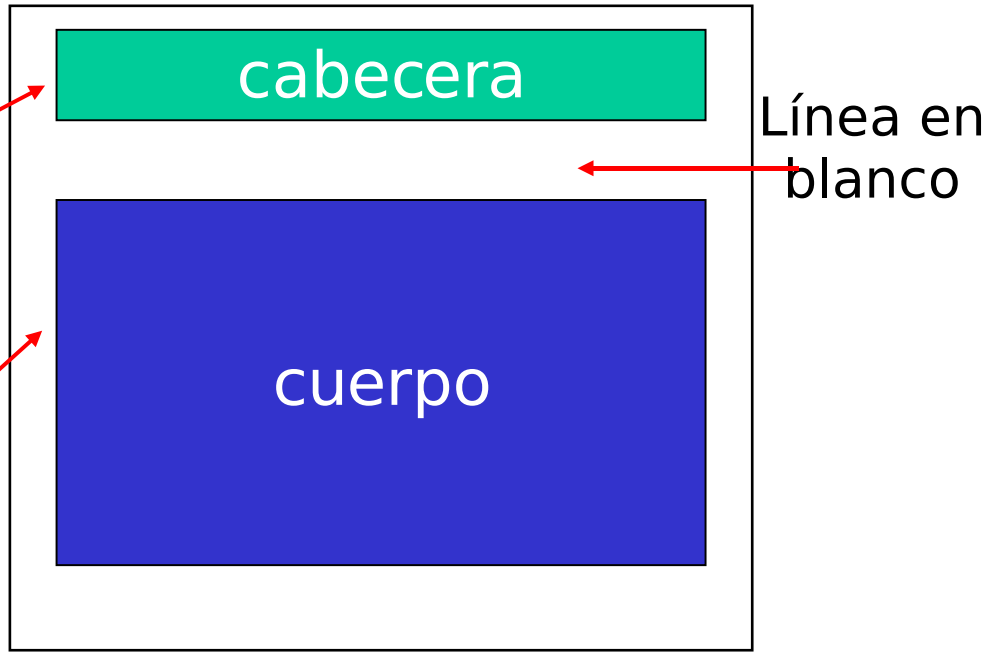
Resumen SMTP

- SMTP usa conexiones persistentes
- SMTP requiere que el mensaje (cabecera y cuerpo) esté en ASCII 7 bits
- SMTP server usa CRLF.CRLF para determinar el fin del mensaje

- Comparando con HTTP
  - HTTP: pull
  - SMTP: push
  - Ambos tienen interacción comando/respuesta y códigos de status ASCII
  - HTTP: cada objeto va encapsulado en su propio mensaje de respuesta
  - SMTP: objetos múltiples se envían en mensaje multipartes

Formato de mensajes de mail

- SMTP: protocolo de intercambio de mensajes de mail
- RFC 822: estándar para el formato de mensajes de texto
- Líneas de cabecera:
  - To:
  - From:
  - Subject:
- Diferentes de los comandos SMTP
- Cuerpo
  - El mensaje
  - ASCII únicamente



MIME, extensiones multimedia

- MIME: Multimedia Mail Extensions, RFC 2045, 2056
- Líneas adicionales en la cabecera declaran contenido MIME

Versión MIME

Método de codificación de datos

Tipo, subtipo, parámetros de los datos multimedia

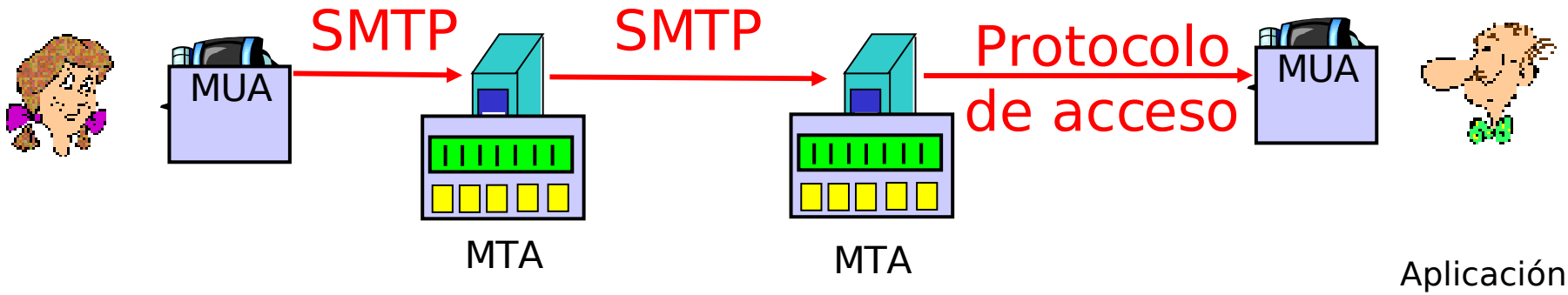
Datos codificados

```
From: alice@crepes.fr
To: bob@hamburger.edu
Subject: Picture of yummy crepe.
MIME-Version: 1.0
Content-Transfer-Encoding: base64
Content-Type: image/jpeg

base64 encoded data .....
.........................
......base64 encoded data
```

Protocolos de acceso de Mail

- SMTP: envío y almacenamiento en el server del receptor
- Protocolo de acceso de mail: recuperación del server
  - POP: Post Office Protocol [RFC 1939]
    - Autorización agente<->servidor y descarga
  - IMAP: Internet Mail Access Protocol [RFC 1730]
    - Más complejo, más características
    - Manipula los mensajes almacenados en el server
  - HTTP: Hotmail , Yahoo! Mail, etc.



Protocolo POP3

- Fase de autorización
  - Comandos del cliente
    - user:
    - pass:
  - Respuestas del servidor
    - +OK
    - -ERR
- Fase de transacción
  - list: Lista los números de mensaje
  - retr: Recupera mensaje por número
  - dele: Borra
  - quit

```
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
```

POP3 e IMAP

POP3

- El modo del ejemplo anterior usa el modelo “descargar y borrar”
- B no puede releer su email si cambia de cliente
- “descargar y conservar”: copias de mensajes en diferentes clientes
- POP3 no conserva estado entre sesiones

IMAP

- Guardar todos los mensajes en un lugar: el server
- Permite al usuario organizar los mensajes en carpetas
- IMAP mantiene estado entre sesiones
  - Nombres de carpetas y mapeo entre ID del mensaje y nombre de carpeta

Capa de Aplicación

2.1 Principios de aplicaciones de red

2.2 Web y HTTP

2.3 FTP

2.4 Correo electrónico

SMTP, POP3, IMAP

2.5 DNS

2.6 Archivos compartidos con P2P (*peer-to-peer*)

2.7 Programación con Sockets TCP

2.8 Programación con Sockets UDP

2.9 Creación de un Web server

DNS: Domain Name System

- Personas
  - Usan identificadores
  - DNI, nombre, CUIT
- Hosts y routers de Internet
  - Dirección IP (32 bits) Usada para direccionar datagramas
  - Nombres (www.yahoo.com) usados por los humanos
- ¿Cómo mapear entre direcciones IP y nombres?

- Domain Name System:
  - Base de datos distribuida, implementada en una jerarquía de muchos servidores de nombres
  - Hosts, routers, servidores de nombres, se comunican para resolver nombres (traducción dirección/nombre)
  - Función central de Internet, implementada como protocolo de aplicación
  - Complejidad en el borde de la red

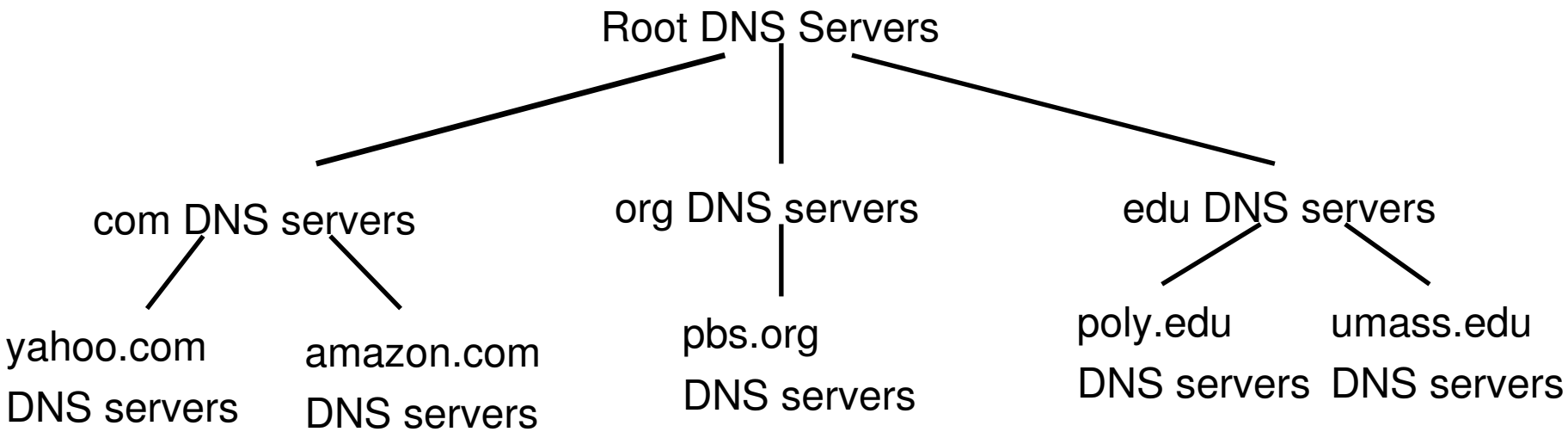
DNS

- ¿Por qué no centralizar el DNS?
  - Punto único de fallo
  - Volumen de tráfico
  - Base de datos centralizada distante
  - Mantenimiento
  - ¡No escala!

- Servicios DNS
  - Traducción de nombre de host a dirección IP
  - Aliases de hosts
    - Nombre canónico y aliases
    - Aliases de mail servers
    - Distribución de carga
  - Web servers replicados
    - Conjunto de direcciones IP para un mismo nombre canónico

Base de datos jerárquica, distribuida

- Cliente quiere IP de www.amazon.com
  - Consulta un root server para encontrar el DNS server de .com
  - Consulta el server de .com para obtener el DNS server de amazon.com
  - Consulta el DNS server de amazon.com para obtener IP de www.amazon.com



DNS: Root name servers

- Contactados por NS locales que no logran resolver un nombre
- Root name server:
  - Contacta a un NS *autoridad*, o *autorizado* (*authoritative*) si no conoce el mapeo del nombre
  - Obtiene el mapeo
  - Devuelve el mapeo al NS local

13 root name servers mundiales

TLD y servers autoridad

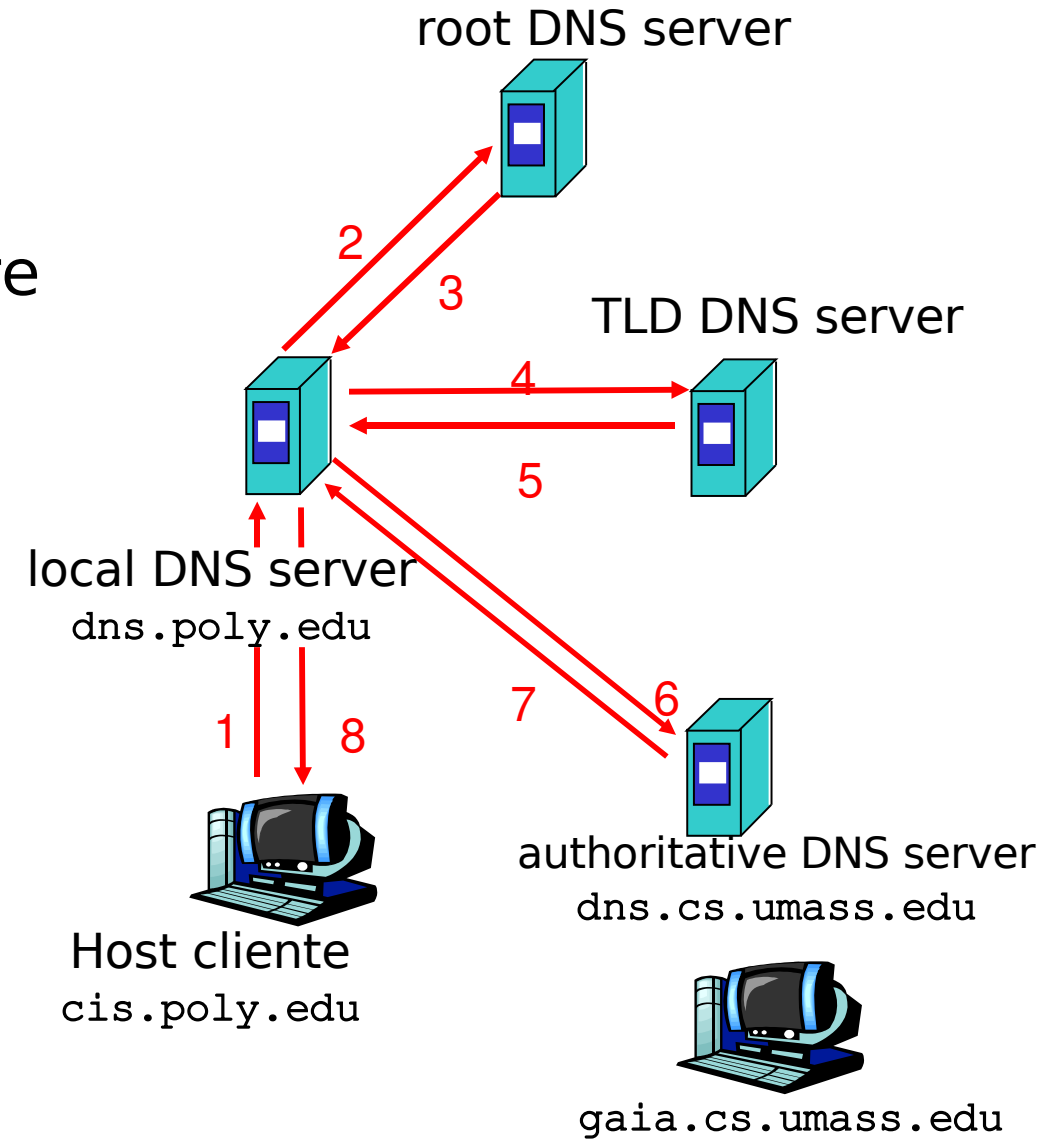
- Top-level domain (TLD) servers: responsables por los dominios com, org, net, edu, etc, y todos los dominios de países como uk, fr, ca, jp.
  - Network solutions mantiene los servidores para el TLD com
  - Educause para el TLD edu
- DNS Servers autoridad: los servidores DNS de la organización, ofreciendo mapeos *autoritativos* de nombre a IP para los servidores de la organización (p.ej. Web y mail).
  - Puede ser mantenido por la organización o por un proveedor de servicios

Name Server Local

- No pertenece estrictamente a la jerarquía
- Cada ISP (residencial, empresa, universidad) tiene uno
  - También llamado “default name server”
- Cuando un host hace una consulta DNS, es enviada a su DNS local
  - Actúa como un proxy que inyecta la consulta en la jerarquía

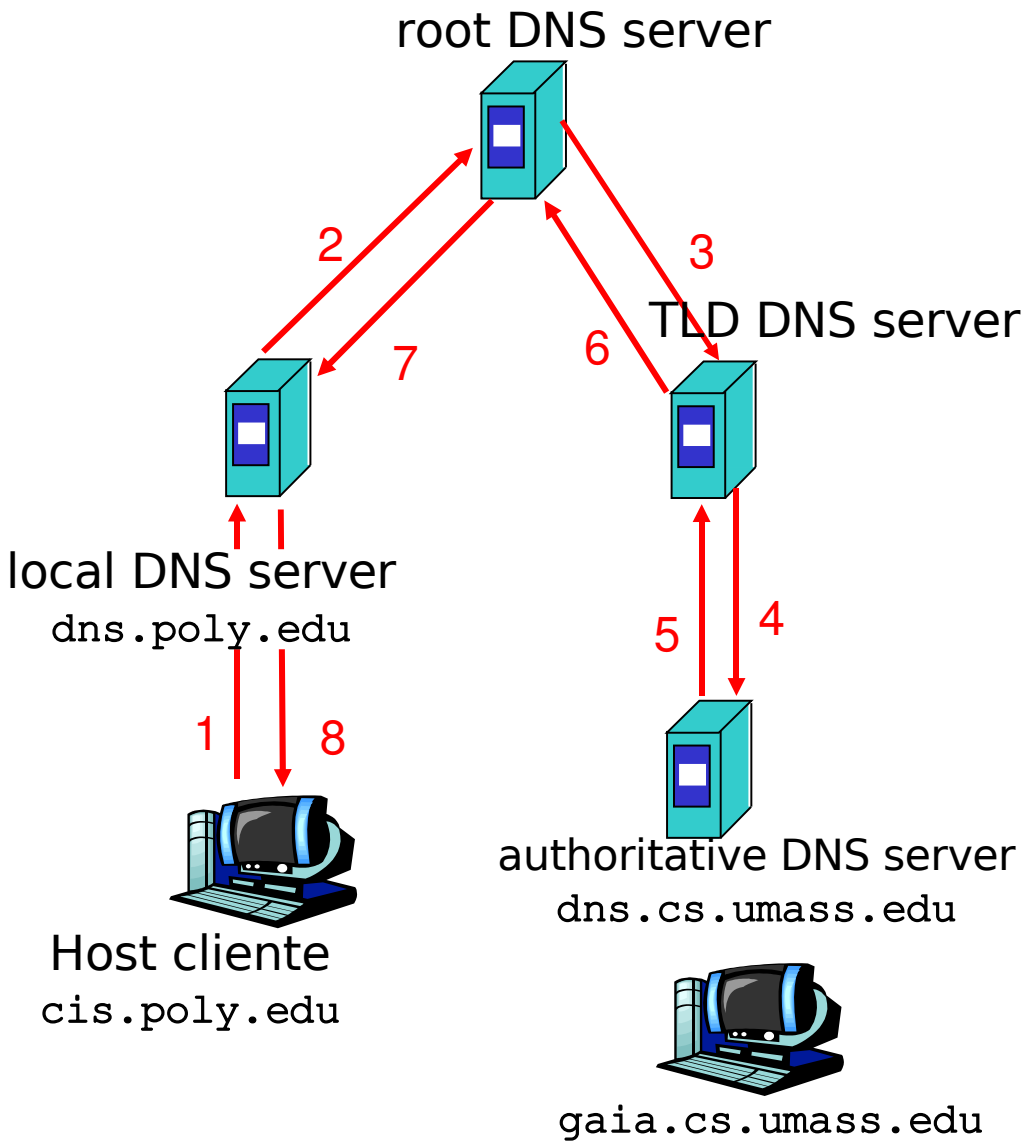
Ejemplo

- Host en cis.poly.edu quiere dirección IP de gaia.cs.umass.edu



Consultas

- Consulta recursiva
  - La carga de la resolución de nombres queda sobre el server contactado
  - ¿Carga pesada?
- Consulta iterativa
  - El server contactado responde con el nombre del server a contactar
  - “No conozco este dato, pero pregunte a este server”



Cache y actualización de registros

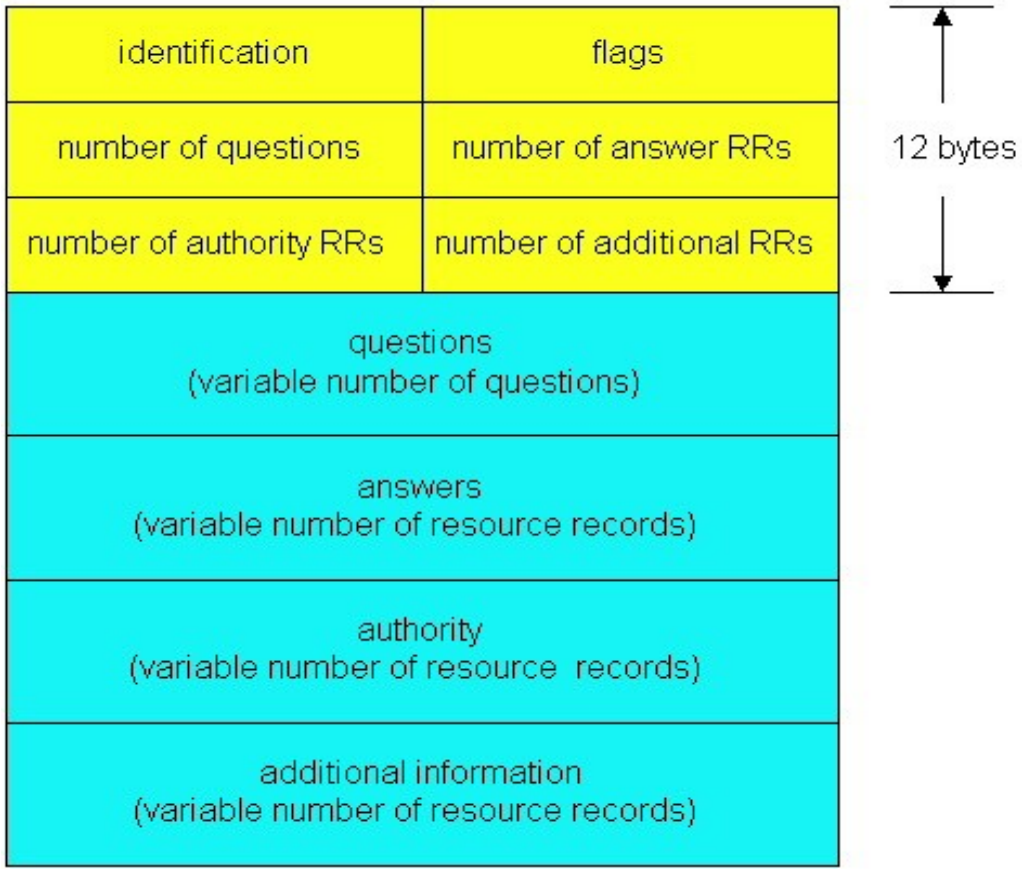
- Una vez que un NS conoce un mapeo, lo guarda en cache
  - Las entradas en cache desaparecen luego de cierto tiempo
  - Los servers TLD típicamente están en cache de los servidores locales
    - De forma que los servers root no son visitados muy a menudo
- Mecanismos de actualización y notificación bajo diseño por IETF
  - RFC 2136
  - http://www.ietf.org/html.charters/dnsind-charter.html

Registros DNS

- La BD almacena registros de recursos (RR)
  - Formato (name, value, type, ttl)
- Type=A
  - name = hostname
  - value = IP address
- Type=NS
  - name = domain (e.g. foo.com)
  - value = dirección IP del NS autoridad para este dominio
- Type=CNAME
  - name = alias de un nombre canónico
    - www.ibm.com es en realidad servereast.backup2.ibm.com
  - value = nombre canónico
- Type=MX
  - value = nombre del mail server asociado con el name

Mensajes de protocolo DNS

- Mensajes de consulta y respuesta con el mismo formato
- Cabecera
  - Identificación
    - 16 bits para la consulta
    - La respuesta usa el mismo número
- Flags
  - Consulta o respuesta
  - Solicito recursión
  - Recursión posible
  - Respuesta autoritativa



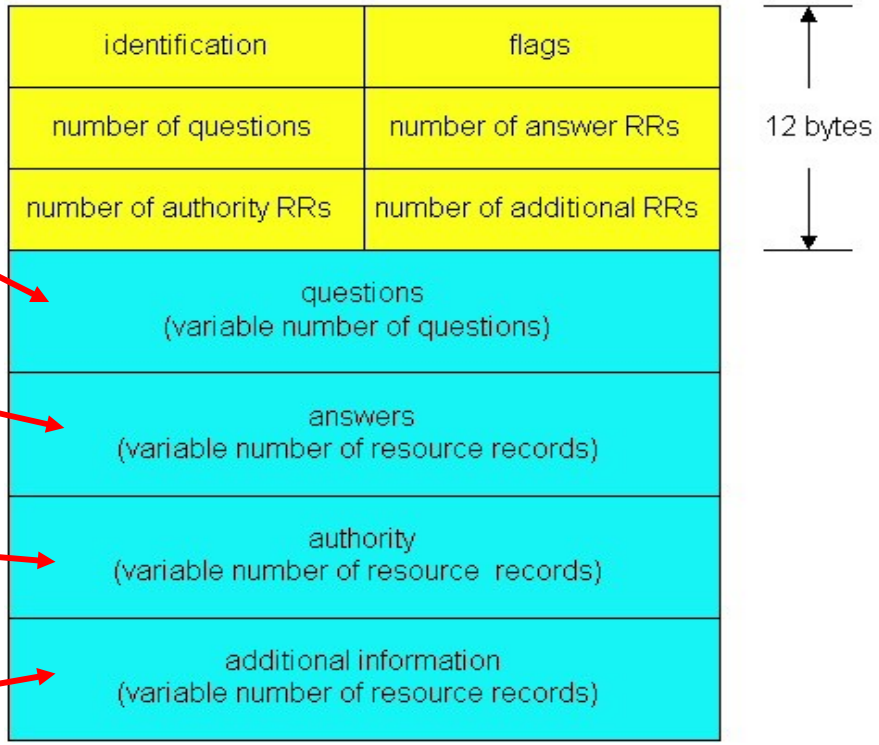
Mensajes de protocolo DNS

Campos name y type para una consulta

RRs en respuesta a la consulta

Registros para servers autoritativos

Otra información útil adicional



Inserción de registros DNS

- Ej. empresa nueva “Network Utopia”

- Registramos el nombre networkutopia.com en un *registrar* (como Network Solutions)
- El *registrar* debe recibir nombres y direcciones IP de su NS autoritativo (primario y secundario)
- El *registrar* inserta dos Rrs en el server TLD de com
    - (networkutopia.com, dns1.networkutopia.com, NS)
    - (dns1.networkutopia.com, 212.212.212.1, A)
- Cargar registro tipo A de server autoritativo para www.networkutopia.com y registro tipo MX para networkutopia.com
- ¿Cómo se obtiene la dirección de nuestro sitio?

Capa de Aplicación

2.1 Principios de aplicaciones de red

2.2 Web y HTTP

2.3 FTP

2.4 Correo electrónico

SMTP, POP3, IMAP

2.5 DNS

2.6 Archivos compartidos con P2P (*peer-to-peer*)

2.7 Programación con Sockets TCP

2.8 Programación con Sockets UDP

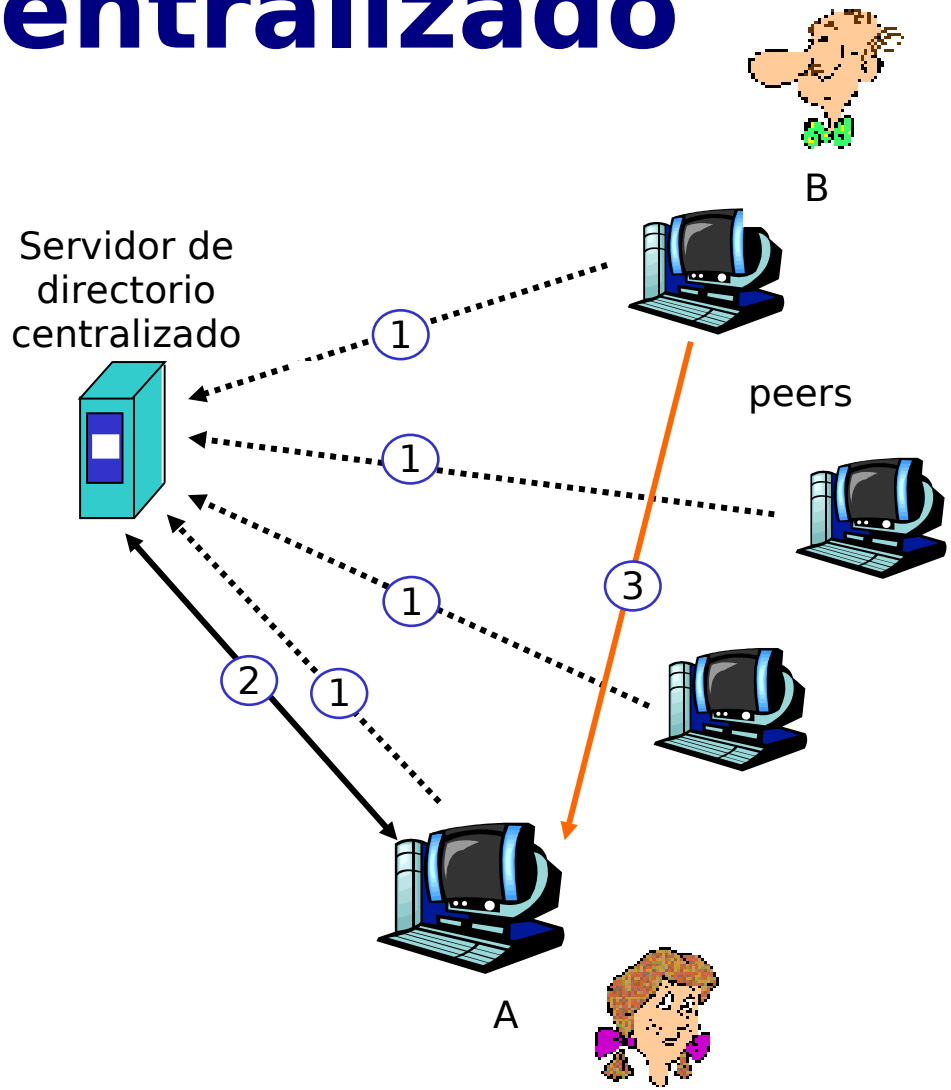
2.9 Creación de un Web server

P2P para compartir archivos

- Ejemplo
  - A corre aplicación cliente P2P en su notebook
  - Se conecta intermitentemente a Internet y obtiene nuevo IP a cada conexión
  - Pide “Hey Jude”
  - La aplicación muestra otros peers que tienen una copia de “Hey Jude”
  - A elige a uno de los peers, B
  - El archivo se copia desde la PC de B a la notebook de A mediante HTTP
  - Mientras A descarga, otros usuarios descargan desde A
  - A es a la vez un cliente Web y un server Web
  - Todos los peers son servers: altamente escalable

P2P: Directorio centralizado

- Diseño original de NAPSTER
- Cuando un peer se conecta, informa al servidor central:
  - Dirección IP
  - Contenido
- A pide “Hey Jude”
- A requiere el archivo a B



P2P: problemas del directorio centralizado

- Punto único de fallo (SPOF, Single point of failure)
- Cuello de botella para la performance
- Se infringen derechos de Copyright

- La transferencia de archivos es descentralizada, pero la localización de archivos es altamente descentralizada

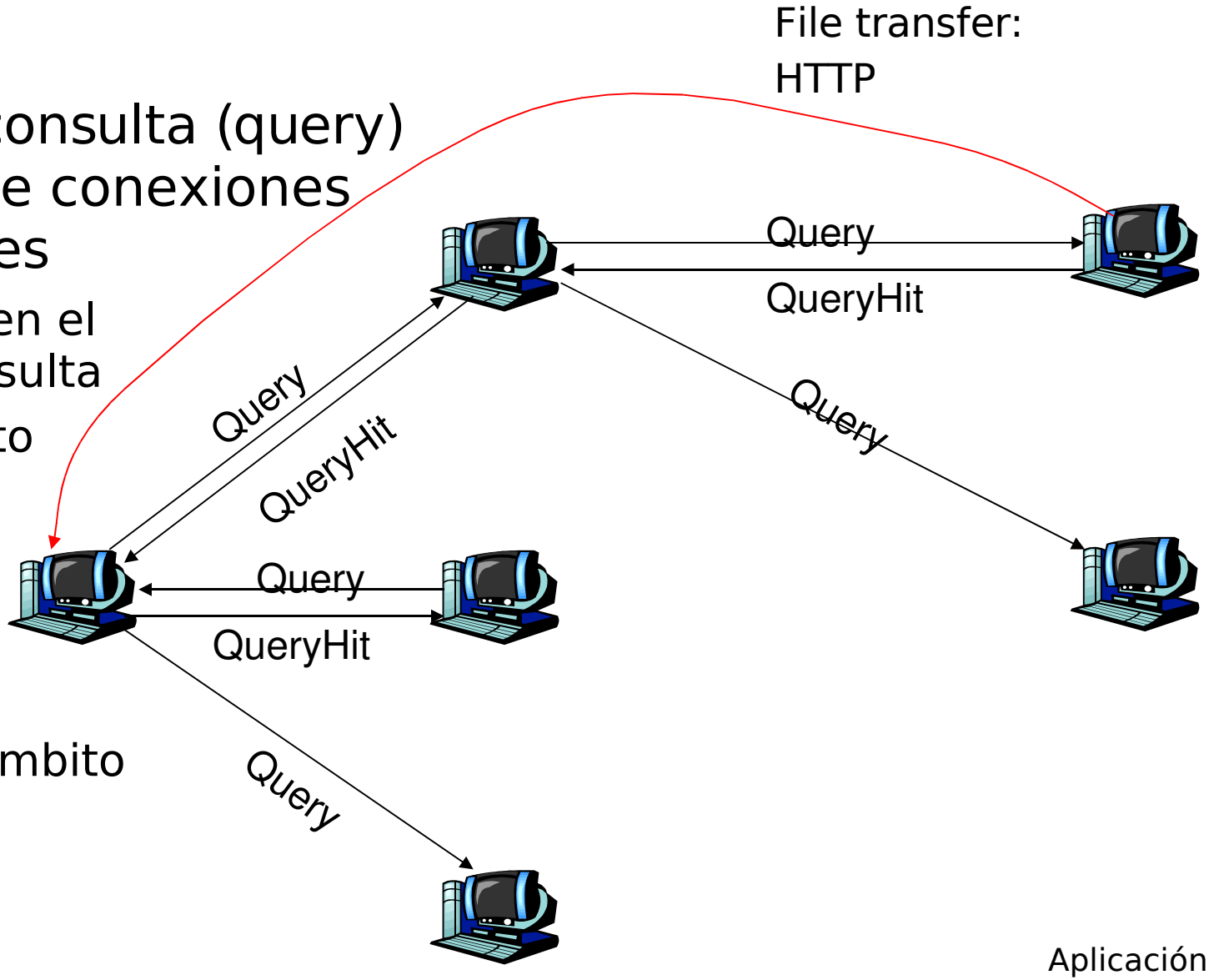
Inundación de consultas

- Gnutella
- Completamente distribuido
- No existe servidor central
- Protocolo en el dominio público
- Muchos clientes Gnutella implementan el protocolo

- Overlay network
  - Un grafo
- Vértice entre X e Y si hay una conexión TCP
- El conjunto de todos los peers y vértices es una overlay net
- Un arco no es un enlace físico
- Un peer estará típicamente conectado con menos de 10 vecinos en la red overlay

Protocolo Gnutella

- Mensaje de consulta (query) enviado sobre conexiones TCP existentes
  - Los peers repiten el mensaje de consulta
  - Mensaje de éxito (QueryHit) es enviado por el camino inverso
  - Escalabilidad: inundación de ámbito limitado

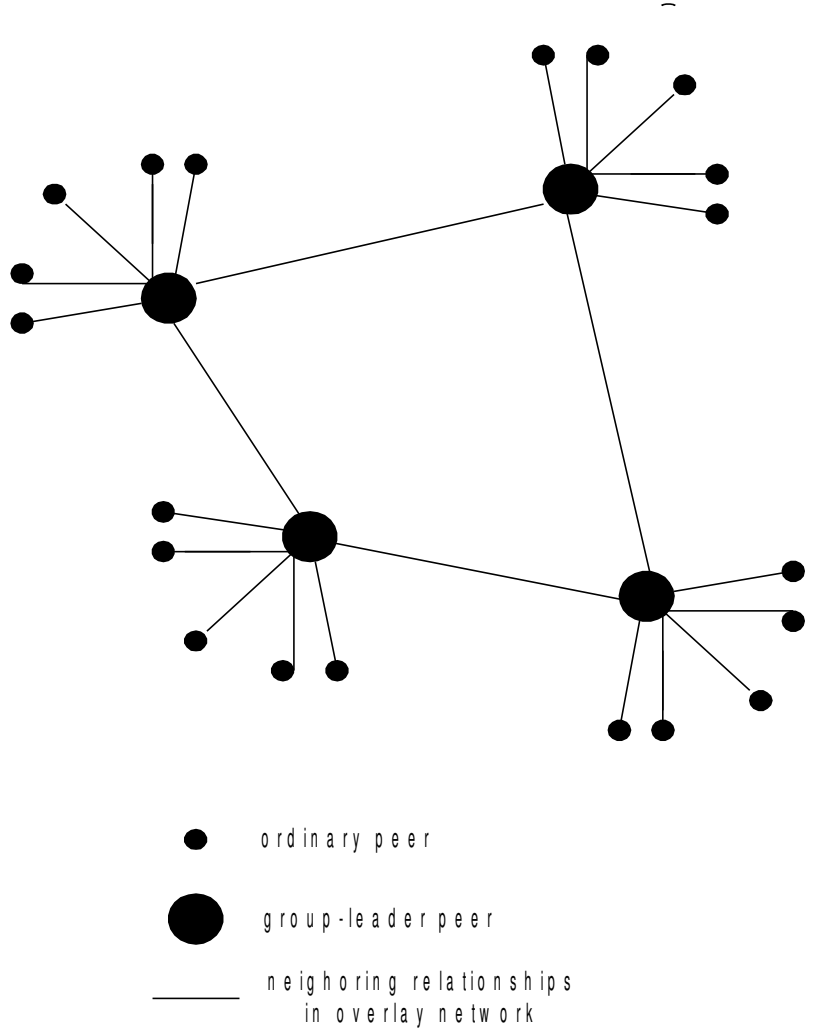


Gnutella: *Peer* que se une

- El peer X que llega debe encontrar algún otro peer en la red Gnutella. Usa una lista de peers candidatos
- X secuencialmente intenta conectar TCP con peers en la lista hasta lograr establecer conexión con Y
- X envía mensaje PING a Y; Y repite el mensaje
- Todos los peers que reciben el mensaje PING responden con mensaje PONG
- X recibe muchos mensajes PONG. Puede establecer conexiones TCP adicionales
- Peer que abandona: ver problema de homework

Explotando la heterogeneidad: KaZaA

- Cada peer o bien es un líder de grupo, o bien es asignado a uno
  - Conexión TCP entre el peer y su líder de grupo
  - Conexiones TCP entre algunos pares de líderes de grpuo
- El líder de grupo registra el contenido de todos sus hijos



Consultas en KaZaA

- Cada archivo tine un valor hash y un descriptor
- El cliente consulta por palabras clave a su líder de grupo
- Éste responde con coincidencias (matches)
  - Por cada match: metadatos, hash, dirección IP
- Si el líder de grupo reenvía la consulta a otros, éstos responden con matches
- Luego el cliente selecciona archivos para descargar
  - Enviando requests HTTP usando el hash como identificador a los peers que mantienen el archivo deseado

Cuestiones de KazaA

- Limitaciones para las subidas simultáneas
- Encolado de requests
- Prioridades como incentivo
- Descarga en paralelo

Capa de Aplicación

Programación Sockets

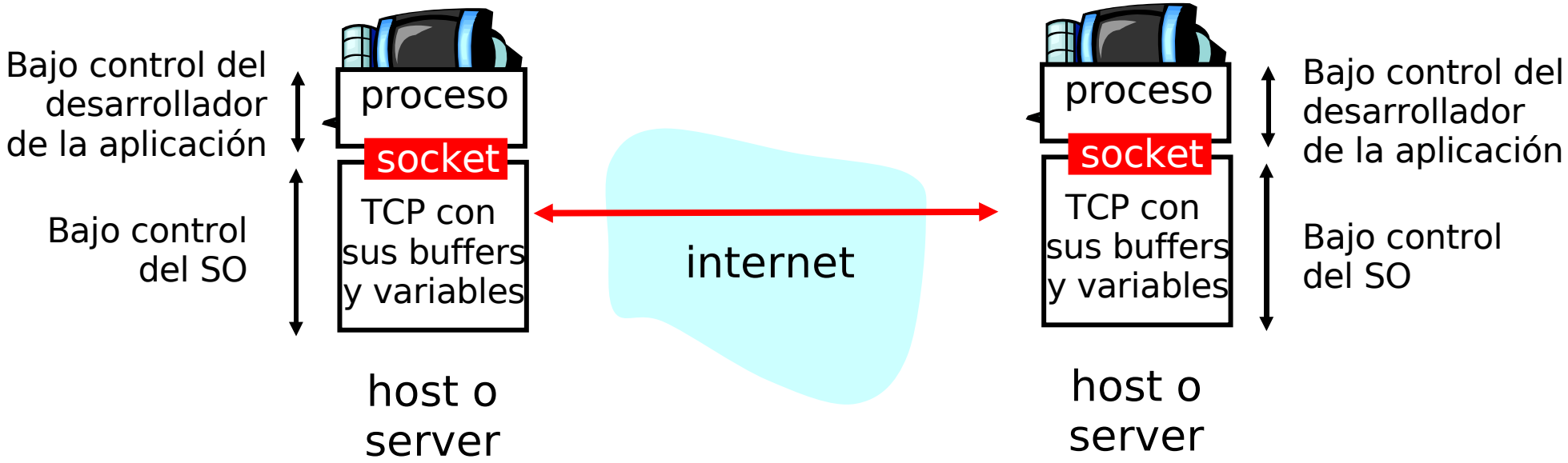
- Objetivo
  - Aprender cómo construir aplicaciones cliente/servidor que se comuniquen usando sockets
- API de Socket
  - Introducida en BSD4.1 UNIX, 1981
  - Explícitamente creados, usados, liberados por las aplicaciones
  - Paradigma cliente/servidor
  - Dos tipos de servicio de transporte via API de sockets
    - No confiable, de datagramas
    - Confiable, orientada a stream de bytes

socket

Una interfaz *local al host, creada por la aplicación, controlada por el SO* (una puerta) por la cual los procesos pueden enviar y recibir mensajes a y de otros procesos de aplicación

Programación de Sockets con TCP

- Socket: una puerta entre un proceso de aplicación y un protocolo de transporte extremo-a-extremo (UDP O TCP)
- Servicio TCP: transferencia confiable de bytes desde un proceso a otro



Programación de Sockets con TCP

- El cliente debe contactar al servidor
  - El proceso servidor debe estar coriendo primero
  - El servidor debe haber creado un socket (puerta) que acepte el contacto del cliente
- El cliente contacta al servidor:
  - Creando un socket TCP local
  - Especificando dirección IP y port del proceso servidor
  - Cuando el cliente crea el socket:
  - El cliente TCP establece la conexión al server TCP
  - Cuando es contactado por el cliente, el server TCP crea un nuevo socket para que el proceso servidor se comunique con el cliente
  - Permite al server hablar con múltiples clientes
  - Los números de port se usan para diferenciar los clientes

Para la aplicación

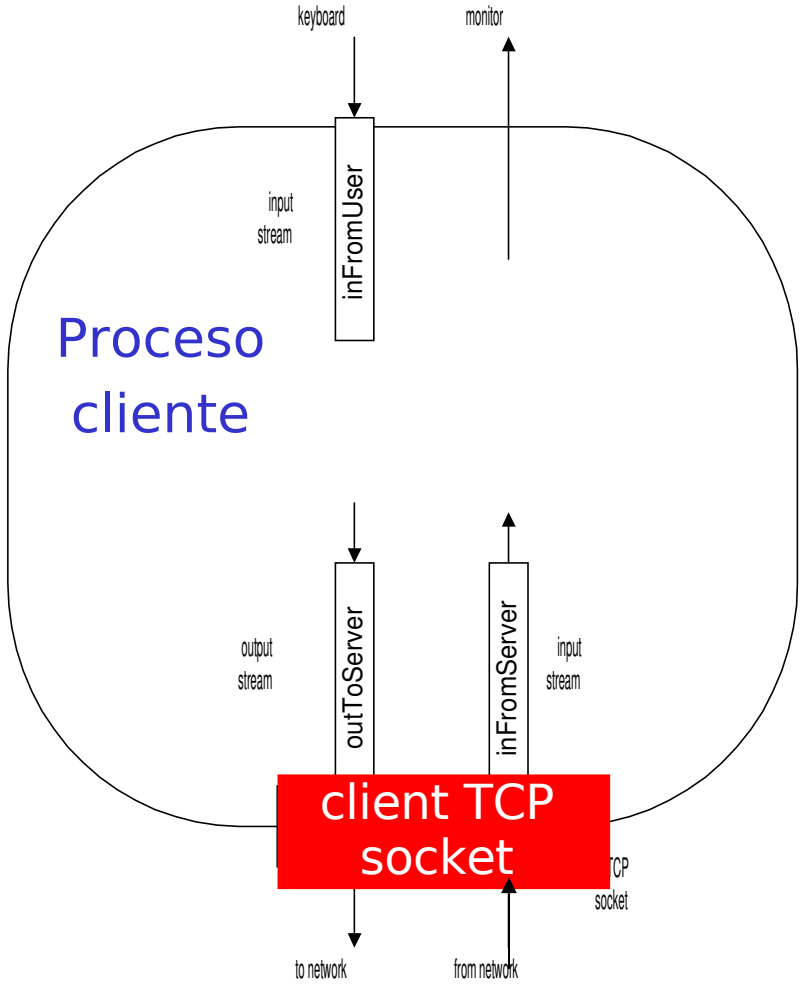
TCP provee transferencia confiable, en orden, entre cliente y servidor (un "caño")

Terminología

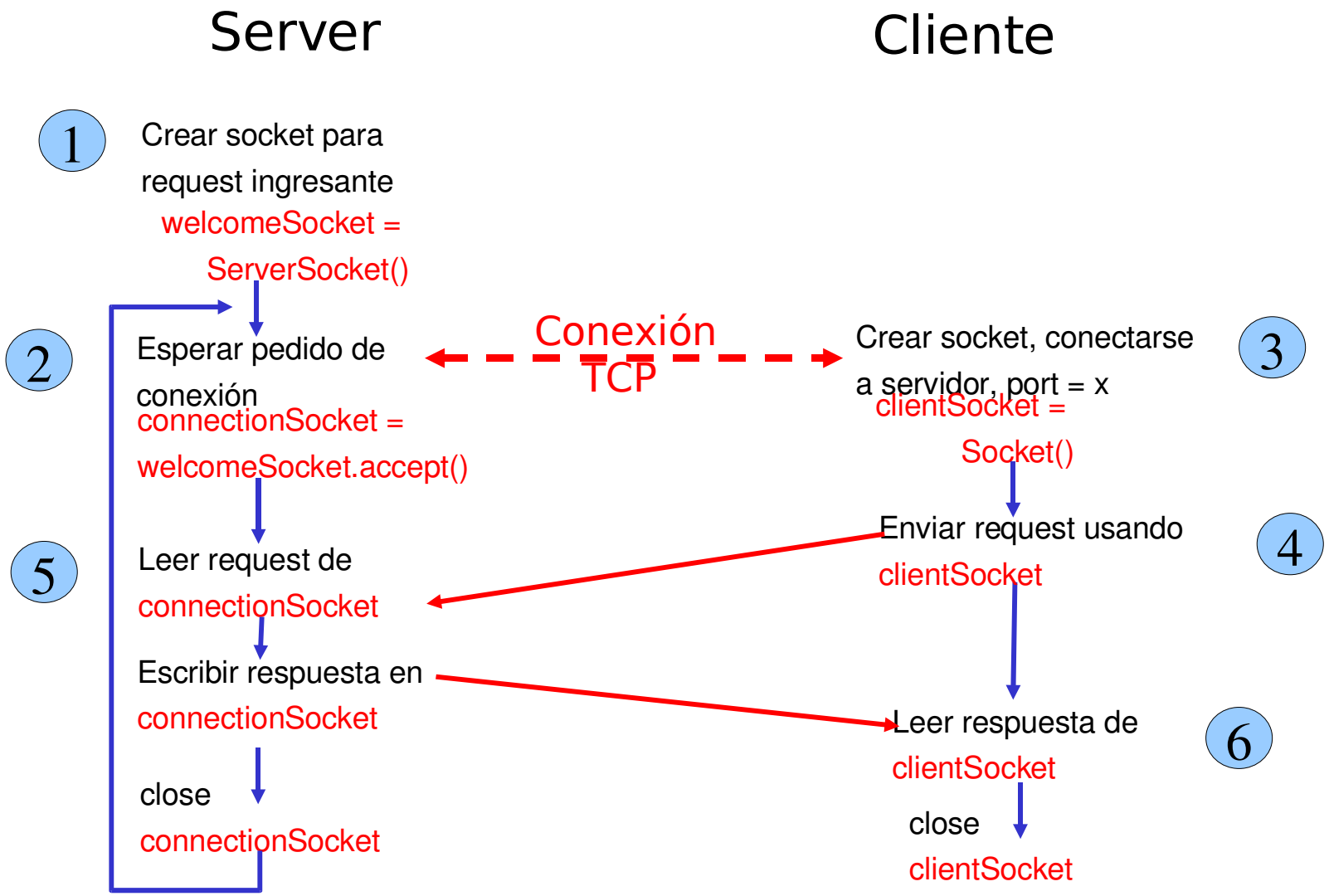
- Un flujo o stream es una secuencia de caracteres que fluye hacia dentro o fuera de un proceso
- Un flujo de ingreso (input stream) se vincula con alguna fuente de entrada para el proceso (como el teclado, o un socket)
- Un flujo de salida (output stream) se vincula con un destino de salida (monitor, o socket)

Programación de Sockets con TCP

- Aplicación cliente-servidor ejemplo
  - 1. Cliente lee línea de entrada standard, la envía al server mediante el socket
  - 2. El server lee línea del socket
  - 3. El server convierte la línea a mayúsculas y la envía de vuelta al cliente
  - 4. El cliente lee e imprime la línea modificada



Interacción cliente/servidor en TCP



Capa de Aplicación

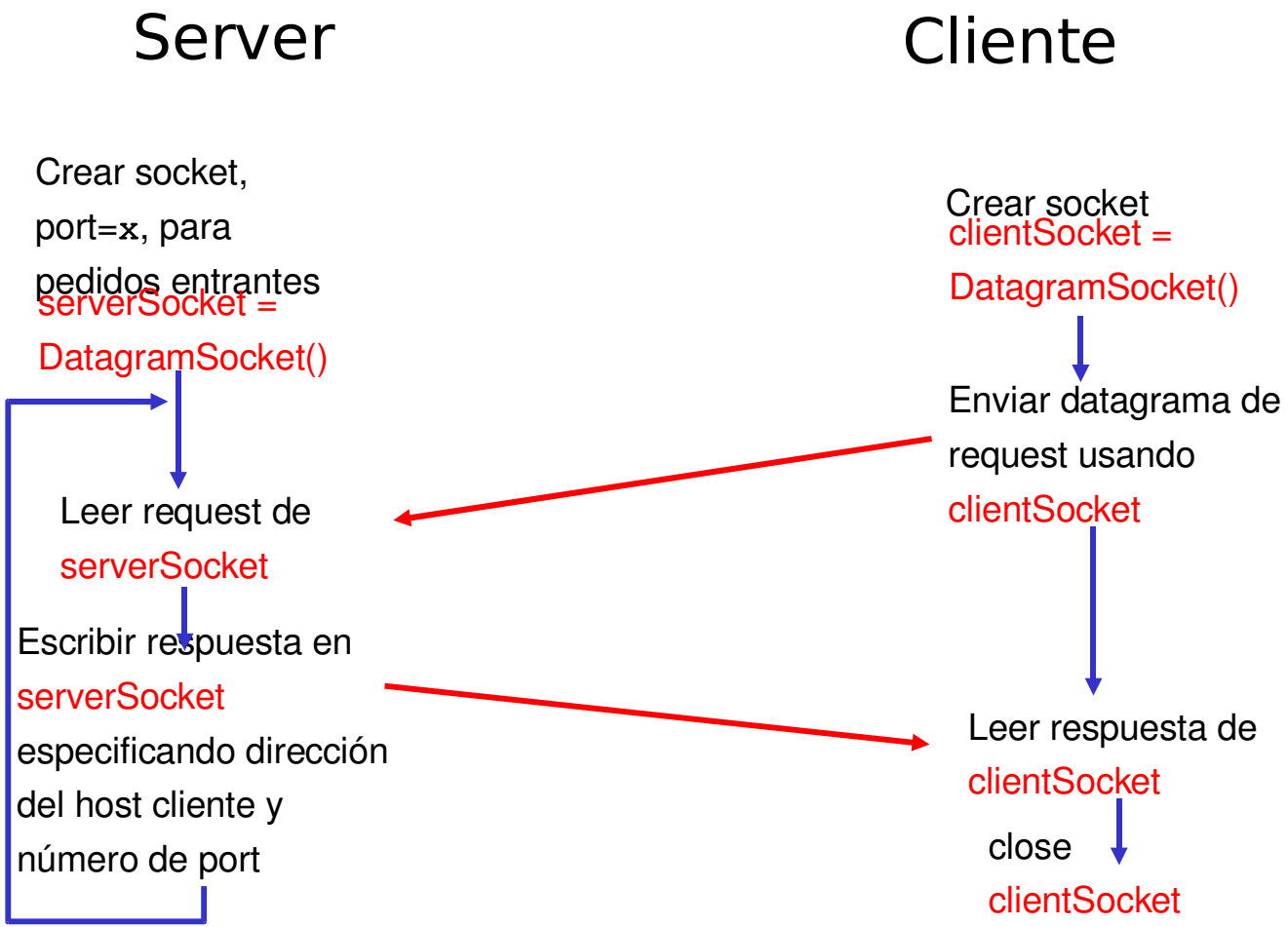
Programación de Sockets con UDP

- UDP
  - No hay conexión entre cliente y servidor
  - No hay acuerdo o handshaking
  - El emisor explícitamente agrega dirección IP y port destino a cada paquete
  - El servidor debe extraer dirección IP y port del emisor de cada paquete recibido
  - Los datos transmitidos con UDP pueden recibirse fuera de orden o perderse

Para la aplicación

UDP provee transferencia no confiable de grupos de bytes ("datagramas") entre cliente y servidor

Interacción cliente/servidor en UDP



Capa de Aplicación

Construcción de un web server sencillo

- Maneja un request HTTP
- Acepta el request
- Analiza la cabecera
- Obtiene el archivo requerido del filesystem del server
- Crea mensaje de respuesta HTTP:
  - Líneas de cabecera + archivo
- Envía respuesta al cliente

- Luego de crear el server, podemos requerirle un archivo con un navegador cualquiera

Resumen de Aplicación

- Arquitecturas de aplicaciones
  - Cliente-servidor
  - P2P
  - Híbridas
- Requerimientos de servicios de las aplicaciones:
  - Confiabilidad, ancho de banda, retardo
- Modelo de servicio de transporte de Internet
  - Orientado a conexión, confiable: TCP
  - No confiable, datagramas: UDP
- Protocolos específicos
  - HTTP
  - FTP
  - SMTP, POP, IMAP
  - DNS
- Programación con sockets
  - TCP y UDP

Resumen de Aplicación

- Típico intercambio de mensajes de request/reply
  - El cliente requiere información o servicio
  - EL servidor responde con datos, código de estado
  - Formatos de mensajes
  - Cabeceras: campos de metadatos
  - Datos: lo que se comunica
- Lo más importante: aprendimos sobre protocolos
- Mensajes de control y de datos
  - in-band, out-of-band
  - Centralizados y descentralizados
  - Con/sin estado (stateful/stateless)
- Mensajes confiables vs. no confiables
- “Complejidad en los bordes de la red”