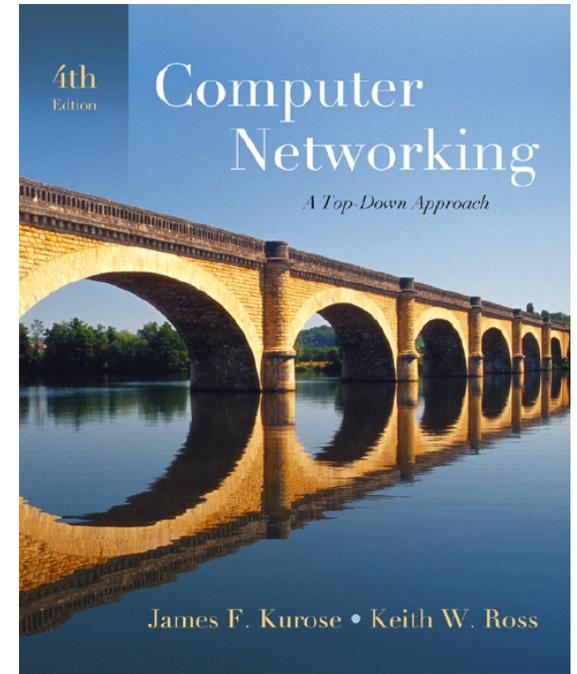


Introducción a las Redes de Computadoras

Capitulo 2

Capa de Aplicación



Nota acerca de las transparencias del curso:
Estas transparencias están basadas en el sitio web que
acompaña el libro, y
han sido modificadas por los docentes del curso.
All material copyright 1996-2007
J.F Kurose and K.W. Ross, All Rights Reserved

Capa de Aplicación

1. Principios de aplicaciones de red
2. Web y HTTP
3. FTP
4. e-mail
 - SMTP, POP3, IMAP
1. DNS
2. P2P
3. Programación de sockets con TCP
4. Programación de sockets con UDP

Capa de Aplicación

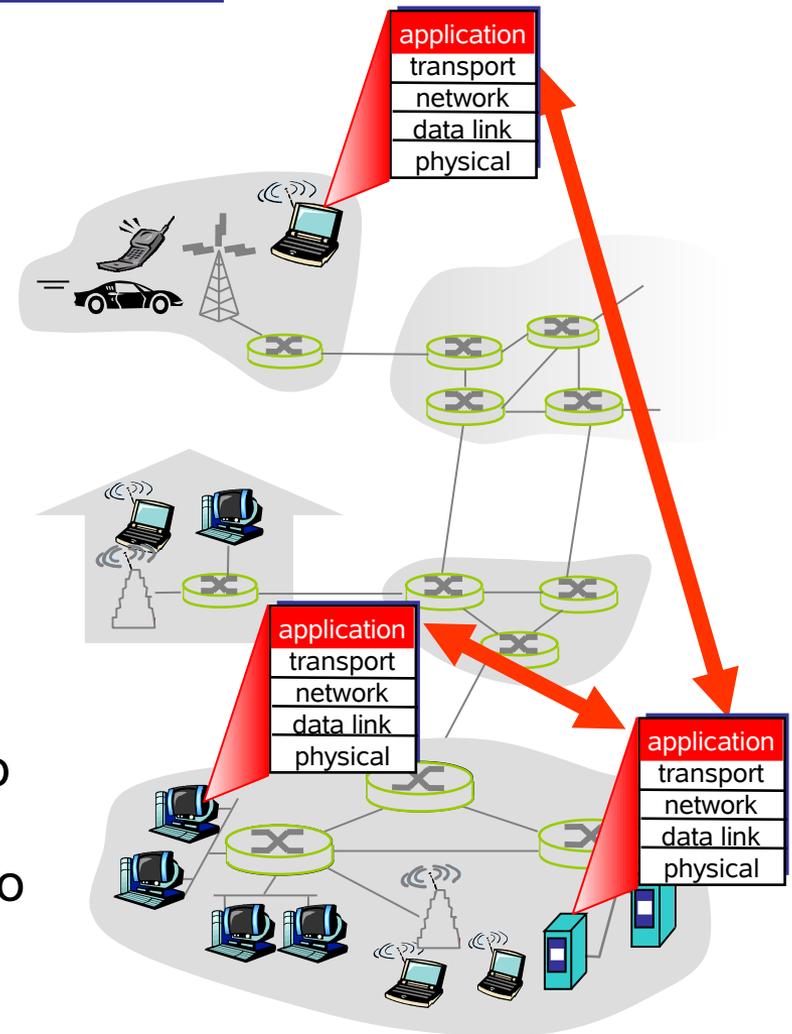
- Objetivos
 - Aspectos conceptuales y de implementación de protocolos de aplicación
 - Modelos de capa de transporte
 - Paradigma cliente servidor
 - Paradigma P2P (peer to peer)
 - Comprender protocolos de aplicación populares
 - HTTP
 - FTP
 - SMTP / POP3 / IMAP
 - DNS
 - Programar aplicaciones de red
 - API de sockets

Aplicaciones de red

- e-mail
- Web
- Mensajería instantánea
- login remoto
- Compartir archivos por P2P
- Juegos en red
- Streaming de video almacenado
- Streaming de video conferencia en tiempo real
- Voz sobre IP (VoIP)
- Procesamiento distribuído

Que es una aplicación de red?

- Programas que
 - Ejecutan en sistemas diferentes
 - Se comunican por la red
 - Ejemplos
 - Servidor Web
 - Explorador Web
- No se necesita escribir programas para dispositivos internos de la red (network-core devices)
 - Los dispositivos internos no ejecutan aplicaciones de usuario
 - Las aplicaciones en sistemas finales permiten rápido desarrollo de aplicaciones

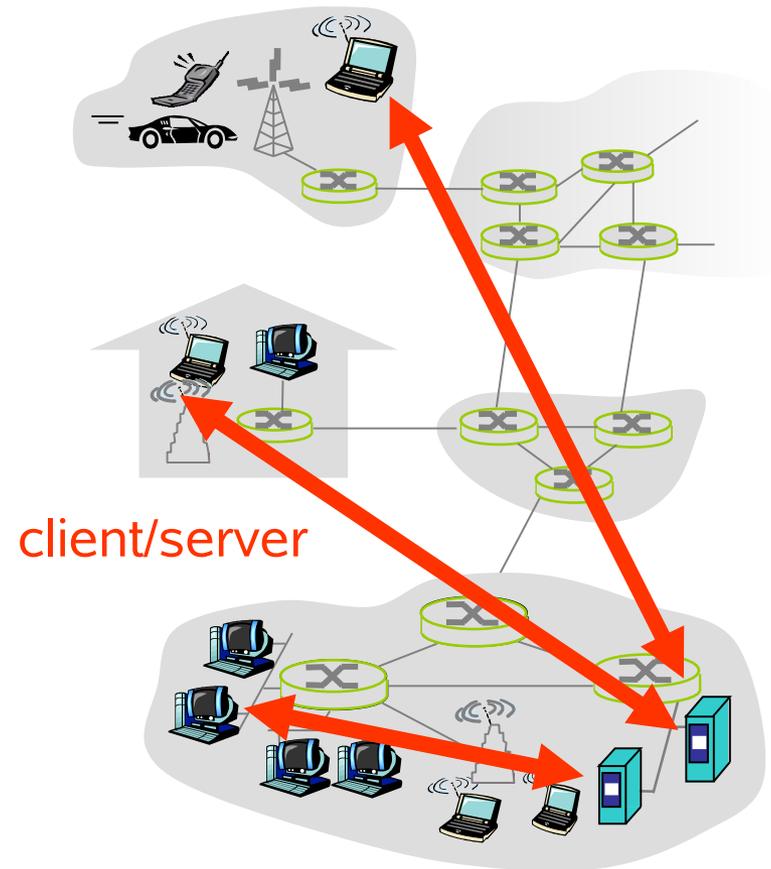


1. Principios de aplicaciones de red

- Arquitecturas de aplicaciones
 - Cliente servidor
 - P2P (peer to peer)
 - Híbridas cliente servidor / P2P

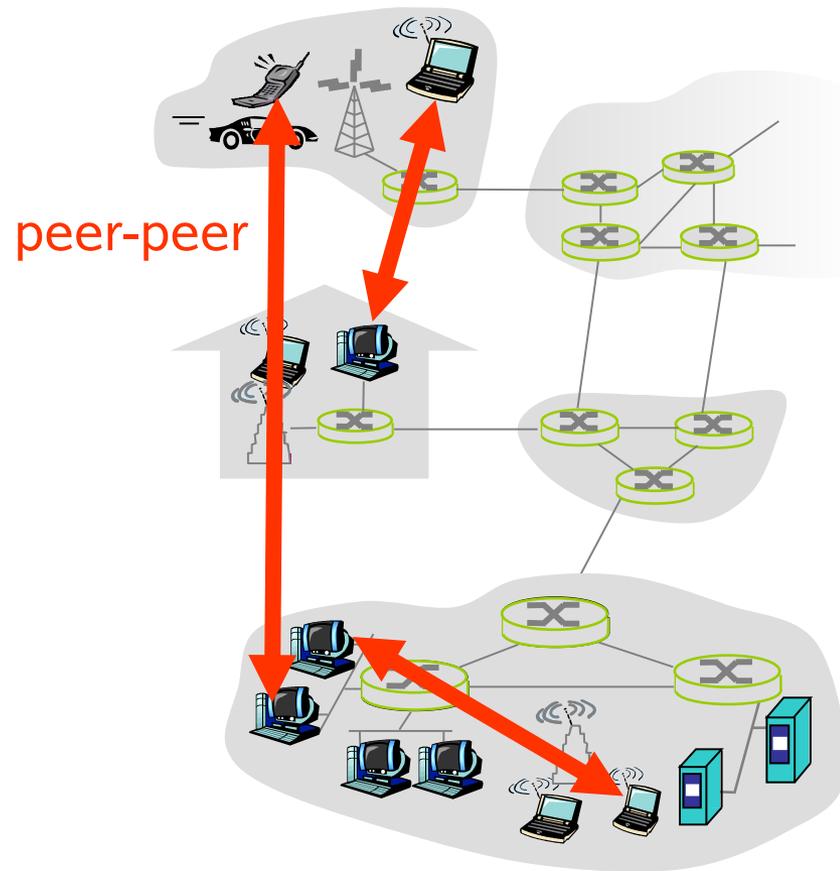
Arquitectura cliente servidor

- Servidor
 - Equipo de alta disponibilidad (siempre encendido)
 - Dirección IP fija
 - Granjas de servidores para escalar
- Cliente
 - Se comunica con el servidor
 - Se comunica a demanda (intermitentemente)
 - Dirección IP dinámica
 - No se comunica con otros clientes



Arquitectura P2P

- Servidor de disponibilidad variable (no siempre encendido)
- Se comunican directamente sistemas finales diversos
- Los “peer” se conectan intermitentemente y pueden tener IP dinámica
- De muy alta escalabilidad pero difícil de administrar



Arquitectura Híbrida

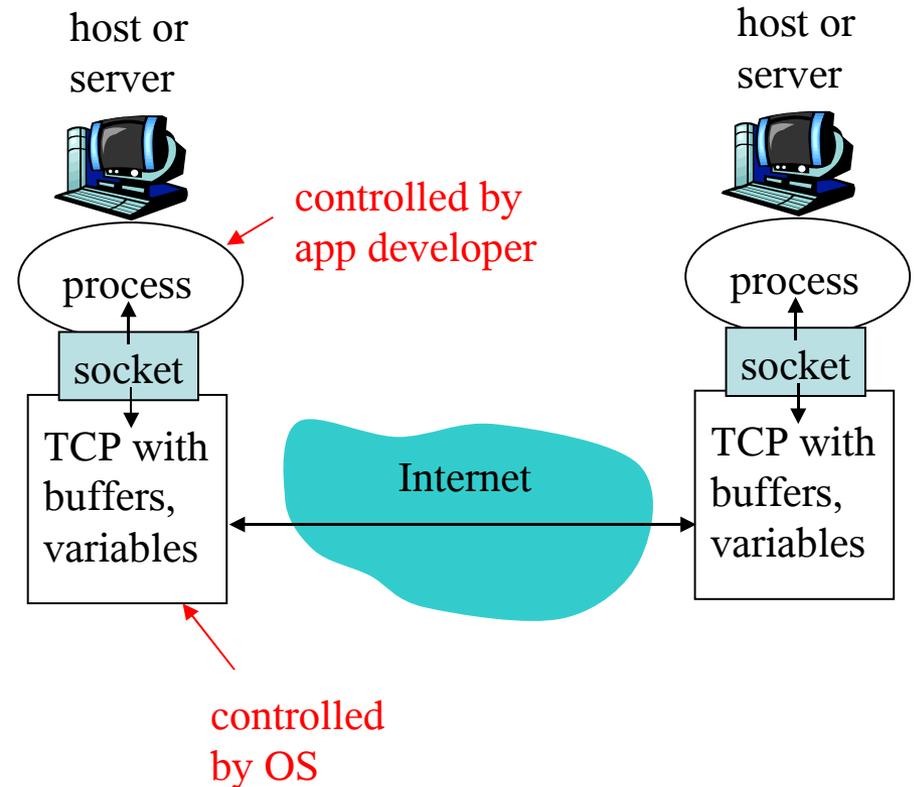
- Skype
 - Aplicación VoIP (Voz sobre IP) P2P
 - Servidor centralizado, encuentra las direcciones de los “peer” remotos
 - Conexión cliente-cliente directa (no interviene el servidor)
- Mensajería instantánea
 - Conversaciones entre usuarios es P2P
 - Servicio centralizado: presencia de clientes, detección, localización
 - Usuario se registra con servidor central
 - Usuario se conecta con servidor central para encontrar contactos

Comunicación de procesos

- Proceso: programa ejecutándose en un equipo (host)
 - Proceso cliente: proceso que inicia la comunicación
 - Proceso servidor: proceso que espera la comunicación de un proceso cliente
- En un mismo equipo, los procesos usan comunicación inter-procesos (definida por el sistema operativo)
- En diferentes equipos, los procesos usan intercambio de mensajes

Sockets

- Los procesos envían/reciben mensajes a través del socket
- El Socket se puede pensar como una puerta de comunicación
 - El proceso que envía deja mensajes en la puerta
 - Confía en una infraestructura del otro lado de la puerta que se encarga de manejar y dejar el mensaje en el socket del proceso receptor



Sockets

- Del lado del programador
 - Se puede elegir el método de transporte
 - Se pueden fijar parámetros para el método de transporte

Identificación de los procesos

- Para recibir mensajes, el proceso debe tener un identificador
- El equipo tiene una única dirección IP de 32 bits
- La dirección IP no es suficiente para identificar el proceso, varios procesos pueden ejecutarse en la misma máquina

Identificación de los procesos

- Además de la dirección IP que identifica el equipo, hay números de puertos asociados a cada proceso
 - Ejemplo:
 - Servidor HTTP: puerto 80
 - Servidor SMTP (E-mail): puerto 25
- Enviar mensaje HTTP para obtener página de www.fing.edu.uy
 - Dirección IP: 164.73.32.3
 - Número de puerto: 80

Protocolo de capa de aplicación

- Define
 - Tipo de mensajes intercambiados
 - Ejemplo: request, response
 - Sintaxis de los mensajes
 - Que campos, parámetros y como son enviados
 - Semántica de los mensajes
 - Que significa la información en los campos
 - Reglas para como y cuando un proceso debe enviar y otro responder a los mensajes
- Protocolos de dominio público
 - Definidos en RFC (Request For Comments)
 - Permiten interoperabilidad entre procesos de diferentes máquinas
 - Ejemplos: HTTP, SMTP
- Protocolos propietarios
 - Ejemplos: Skype

Servicios de transporte

- Pérdida de datos
 - Se pueden tolerar pérdidas (ej: audio)
 - No se pueden tolerar pérdidas (ej: transferencia de archivos)
- Tiempo
 - Algunas aplicaciones requieren que no haya retardos (delay) en las transferencias (ej: VoIP)
- Tasa de Transferencia Efectiva(Throughput)
 - Algunas aplicaciones requieren una gran tasa de transferencia efectiva de datos (ej: video)
- Seguridad
 - Encriptación de los datos
 - Integridad de los datos

Servicios de transporte

<i>Aplicación</i>	<i>Pérdida de datos</i>	<i>Transferencia (Throughput)</i>	<i>Sensible a retardos</i>
Transferencia de archivos	No	Adaptable	No
E-mail	No	Adaptable	No
Páginas web	No	Adaptable	No
Audio/Video en línea	Tolerante	Audio: 5kbps – 1mbps Video: 10kbps – 5mbps	Si 100 ms
Audio/Video almacenado	Tolerante	Audio: 5kbps – 1mbps Video: 10kbps – 5mbps	Si 1 – 5 s
Juegos interactivos	Tolerante	Variable	Si 100 ms
Mensajería instantánea	No	Adaptable	Variable

Servicios de transporte en Internet

- Servicios TCP
 - Orientado a conexión: hay un establecimiento previo entre los procesos cliente y servidor
 - Transporte confiable: los datos llegan en forma correcta
 - Control de flujo: el proceso no envía más de lo que puede aceptar el receptor
 - Control de congestión: maneja el envío cuando la red esta sobrecargada
 - No provee
 - Control de retardo
 - Asegura o garantiza una mínima tasa de transferencia
 - Seguridad

Servicios de transporte en Internet

- Servicios UDP
 - Transferencia de datos no confiable
 - No provee:
 - Establecimiento previo de conexión
 - Confiabilidad
 - Control de flujo
 - Control de congestión
 - Control de retardo
 - Garantía de tasa de transferencia
 - Seguridad
- Por qué proveer servicios UDP

Servicios de transporte en Internet y aplicaciones

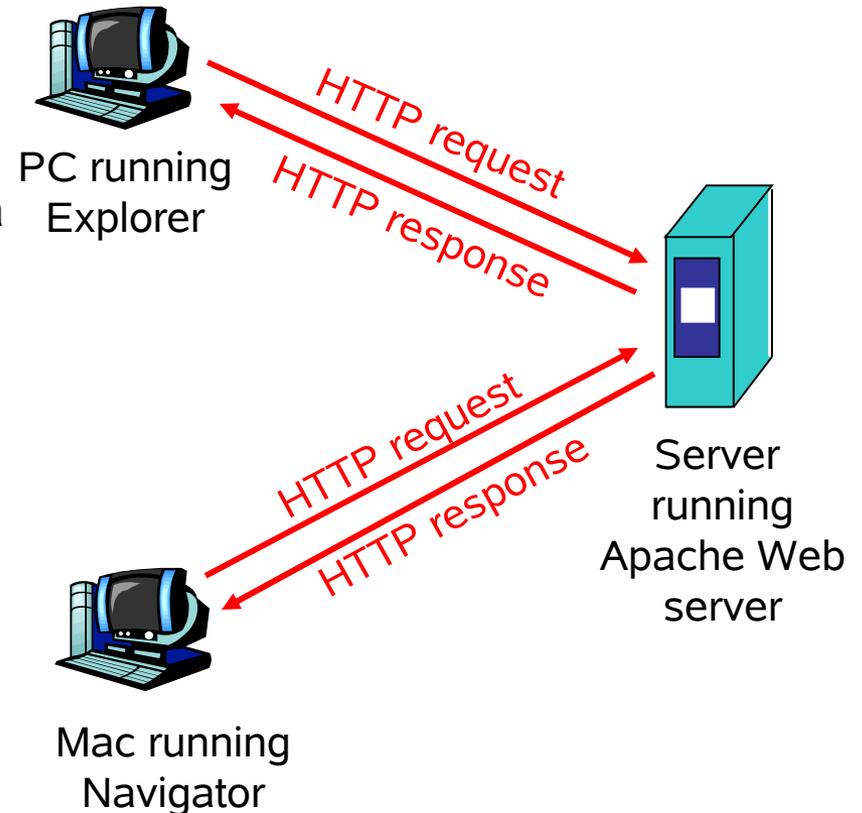
<i>Aplicación</i>	<i>Protocolo de aplicación</i>	<i>Protocolo de transporte</i>
E-mail	SMTP (RFC 2821)	TCP
Terminal remota	Telnet (RFC 854)	TCP
Web	HTTP (RFC 2616)	TCP
Transferencia de archivos	FTP (RFC 959)	TCP
Multimedia	HTTP (Youtube) RTP (RFC 1889)	TCP/UDP
Telefonia (VoIP)	SIP, RTP, Skype	UDP

2. Web y HTTP

- Conceptos
 - Página Web: contenedor de objetos
 - HTML (Hypertext Markup Language)
 - Aplicación
 - Multimedia
 - Documento HTML contiene referencias a objetos
 - Cada objeto es identificable en la red por una dirección URL (Uniform Resource Locator)
 - Ejemplo:
<http://www.fing.edu.uy/inco/cursos/redescomp/horarios.php>

HTTP

- HTTP (Hyper Text Transfer Protocol)
 - Protocolo de aplicación de la Web
 - Modelo cliente servidor
 - Cliente: navegador que realiza pedidos, recibe objetos (páginas HTML) y los muestra
 - Servidor: servidor Web envía objetos en respuesta a los pedidos
 - Protocolo interoperable, variedad de navegadores en diferentes equipos/sistemas operativos, con variedad de servidores Web en diferentes equipos/sistemas operativos



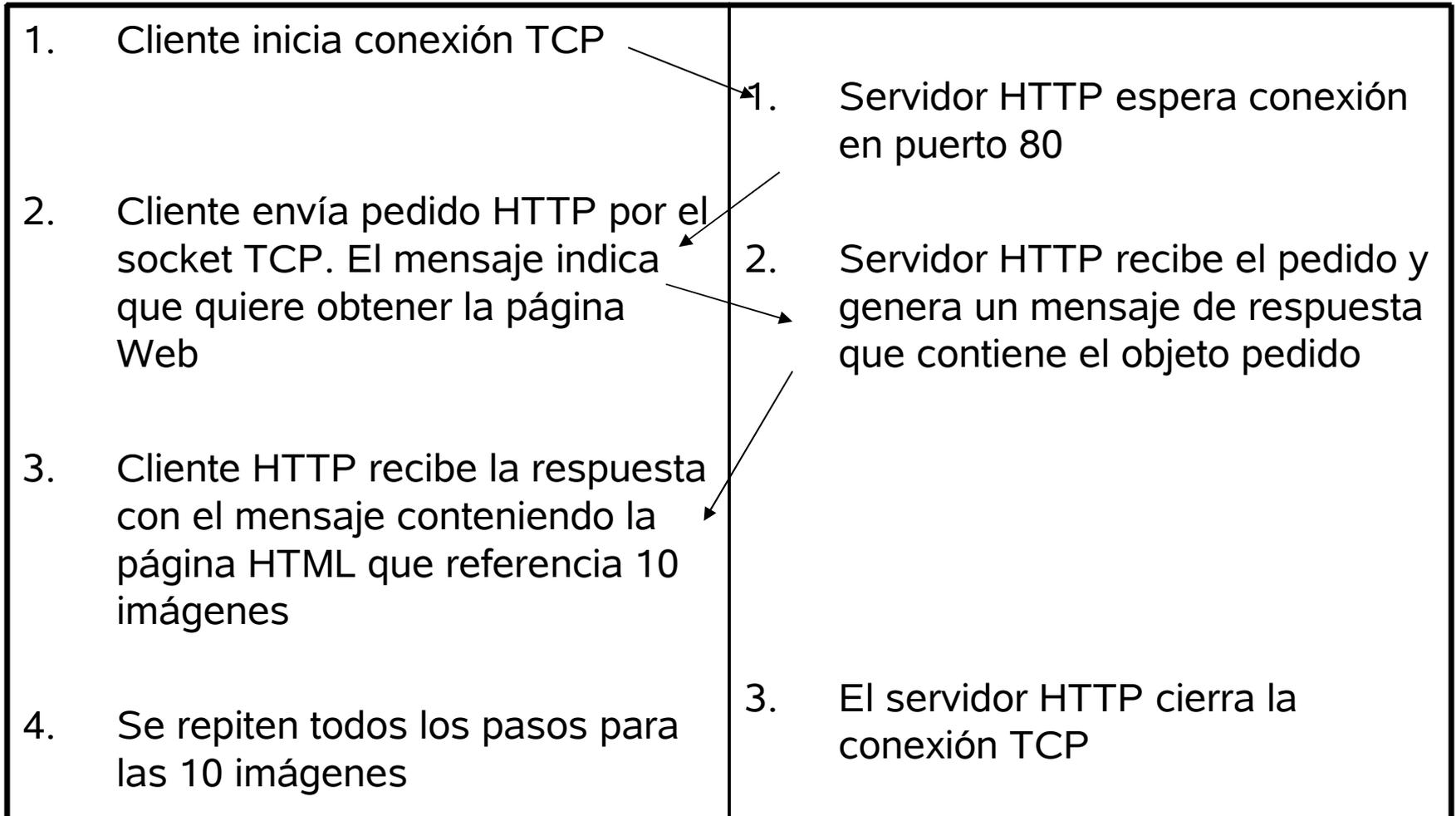
HTTP

- Utiliza TCP
 - Cliente inicia una conexión TCP (crea socket) al servidor, en el puerto 80
 - Servidor acepta una conexión TCP del cliente
 - Mensajes HTTP son intercambiados entre cliente y servidor
 - Se cierra la conexión TCP
- Protocolo sin estado
 - El servidor no mantiene información sobre los pedidos hechos, simplemente responde a cada pedido independientemente
 - Observación:
 - Protocolos con estado, aumenta complejidad
 - Se debe mantener un estado, información sobre los pedidos
 - Si se interrumpe el procesamiento en cliente o servidor, el estado puede ser inconsistente y debe ser solucionado

HTTP

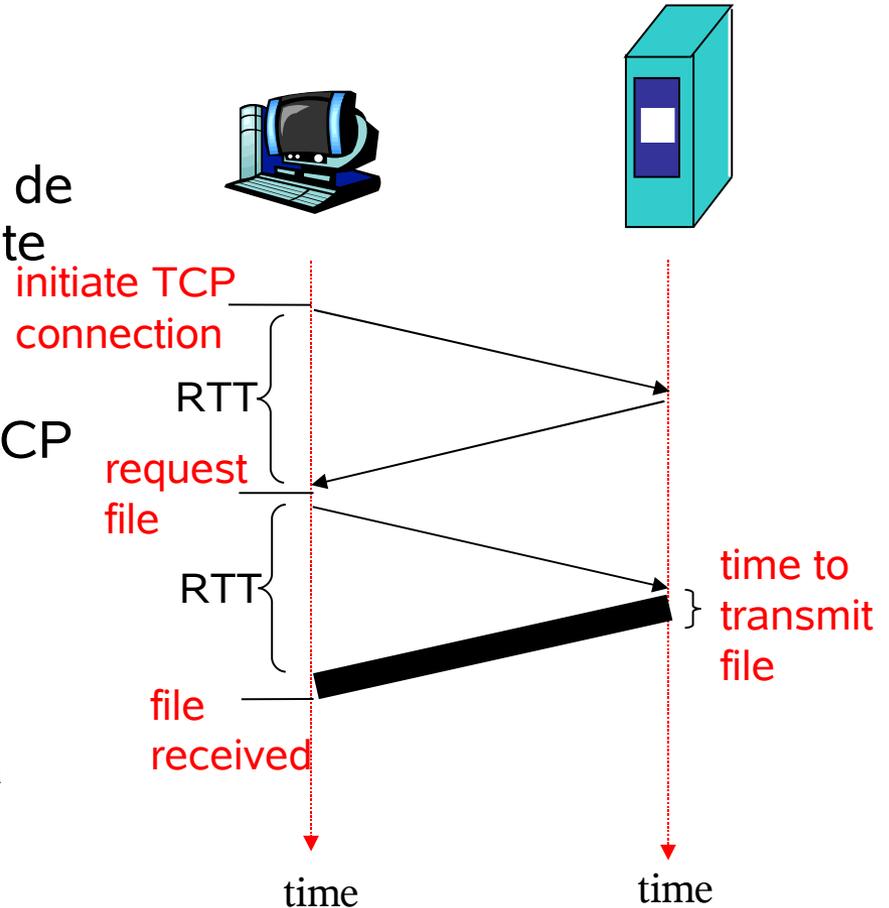
- No persistente
 - Cada objeto es enviado en una conexión TCP diferente
- Persistente
 - Se envían múltiples objetos en cada conexión TCP

HTTP No persistente



HTTP No persistente

- RTT (Round Trip Time)
 - Tiempo que tarda un paquete de información en viajar del cliente al servidor y volver
- Tiempo de respuesta
 - 1 RTT para iniciar conexión TCP
 - 1 RTT para realizar el pedido HTTP y recibir la respuesta
 - Tiempo de transferencia del archivo
 - TOTAL: 2RTT + transferencia



HTTP No persistente

- Requiere 2 RTT por cada objeto de la página
- Sobrecarga de sistema y red por conexiones TCP extras
- Navegadores suelen abrir conexiones paralelas para obtener objetos referenciados

HTTP Persistente

- El servidor deja la conexión abierta luego de enviar la respuesta
- Los mensajes subsecuentes entre el mismo cliente/servidor son enviados por la misma conexión abierta
- El cliente envía pedidos cuando encuentra objetos referenciados
- Se utiliza 1 RTT para todos los objetos

Mensajes HTTP

- Dos tipos de mensajes
 - Request (pedido)
 - Response (respuesta)
- HTTP Request
 - ASCII puede ser interpretado
 - Ejemplo:

```
GET /inco/cursos/redescomp/horarios.php HTTP 1.1
Host: www.fing.edu.uy
User-agent: Mozilla/4.0
Connection: close
Accept-language: fr
—
Return indica fin de mensaje
```

HTTP Request

METODO	-	URL	-	VERSION	CR	LF
CAMPO	:	VALOR			CR	LF
CAMPO	:	VALOR			CR	LF
					CR	LF
CUERPO DEL MENSAJE					CR	LF

HTTP Entrada de Formularios

- Método Post
 - Las páginas Web pueden tener formularios de ingreso de datos
 - Los datos son enviados al servidor en el cuerpo de datos del mensaje HTTP
- Método URL
 - Usa el método GET
 - La información es enviada en campos URL de la línea del pedido en forma de parámetros
 - Ejemplo:
 - www.sitioweb.com/busquedanombre?virginia&juan&martin

HTTP Metodos

- **HTTP/1.0**
 - Get
 - Post
 - Head
- **HTTP/1.1**
 - Get
 - Post
 - Head
 - Put
 - Sube un archivo en el cuerpo a la ruta especificada en el campo URL
 - Delete
 - Borra el archivo especificado en el campo URL

HTTP Response

status line
(codigo de estado del protocolo)

HTTP/1.1 200 OK

header lines

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

datos, ej.,
archivo
HTML

datos ...

HTTP Response códigos de estado

- **200 OK**
 - El pedido fue exitoso, el objeto pedido se encuentra mas adelante en el mensaje
- **301 Moved Permanently**
 - El objeto pedido fue movido a una nueva URL, especificada luego en el mensaje en el campo Location
- **400 Bad Request**
 - El mensaje de pedido no fue entendido por el servidor
- **505 HTTP Version Not Supported**
 - El servidor no soporta la version HTTP

Probando un servidor HTTP

1. Ejecutar Telnet a un servidor HTTP

telnet www.poly.edu 80

Abre conexión TCP a puerto 80
En cis.poly.edu.
Cualquier cosa que se escriba es
Anviada al puerto 80 en cis.poly.edu

2. Escribir pedido GET HTTP request:

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

GET request al servidor HTTP

3. Analizar la respuesta del servidor HTTP

Estado del lado del servidor: cookies

Utilizadas por muchos sitios Web grandes

Componentes:

- 1) cookie header line of HTTP *response* message
- 2) cookie header line in HTTP *request* message
- 3) cookie file kept on user's host, managed by user's browser
- 4) back-end database at Web site

Example:

- Susana siempre accede a Internet de su PC
- Visita un sitio de comercio electrónico (Amazon.com) por primera vez
- Cuando un HTTP request llega al sitio, el sitio crea:
 - ID única
 - Entrada en la base de datos para la IDentry in backend database for ID

Cookies: guardan "estado" (cont.)

cliente

servidor



cookie file



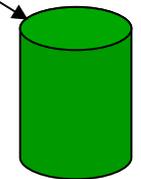
Amazon crea ID

1678 para usuario

Crea entrada

cookie

acceso



Base de datos

acceso

cookie-

acceso

Una semana despues:



Cookies (continúa)

Observación

Información en las cookies:

- autorización
- Carritos de compras
- recomendaciones
- Estado de sesión del cliente (Web e-mail)

Cookies y privacidad:

- Las cookies permiten a los sitios tener información del cliente
- Puede entrarse información personal en los sitios

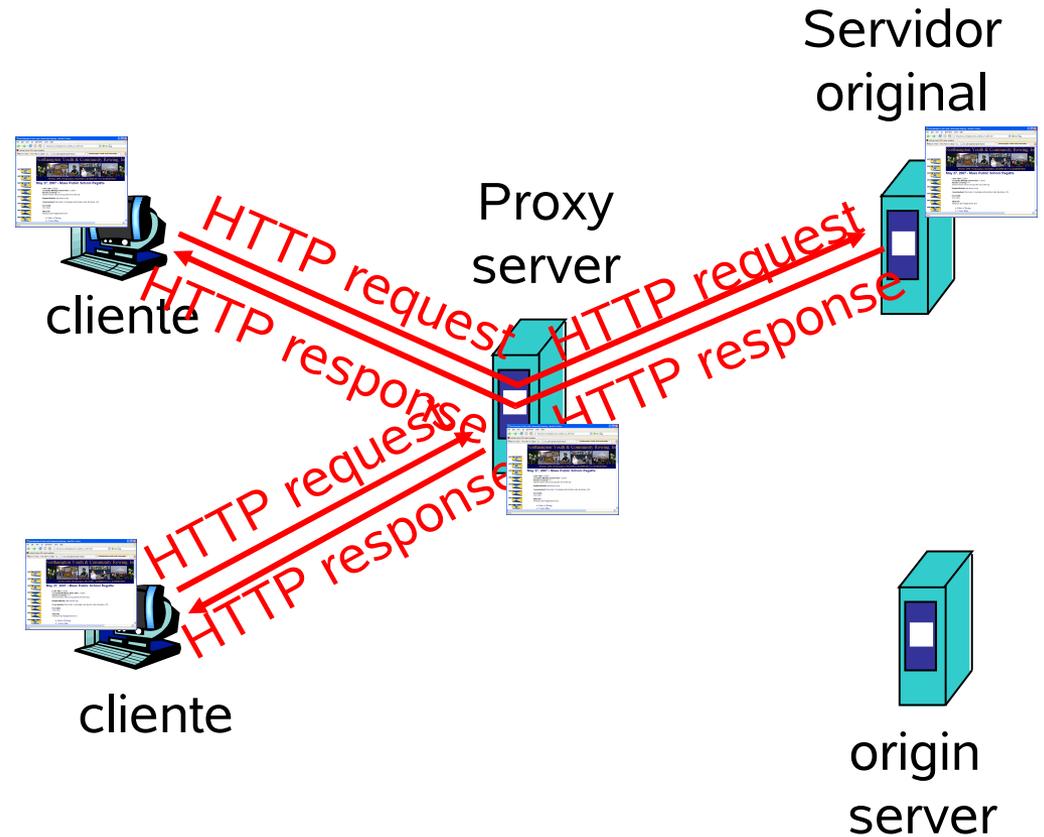
Como se mantiene el estado

- Se mantiene el estado entre el que envía y recibe durante varias transacciones
- cookies: los mensajes http transportan el estado

Web caches (proxy server)

Objetivo: satisfacer el pedido del cliente sin involucrar el servidor original

- Configuración en el navegador: Acceso mediante cache
- Navegador envía todos los pedidos al cache
 - Si el objeto se encuentra se devuelve del cache
 - Si no se encuentra se obtiene del cliente original y se devuelve al usuario



Más sobre Web caching

- Actúa como cliente y servidor
- Típicamente instalado por ISP (universidad, empresa, proveedor residencial ISP)

Por que Web caching?

- Reduce tiempo de respuesta al cliente
- Reduce trafico en la institución.
- Habilita a proveedores con poco contenido a brindar más contenido (también lo hace P2P)

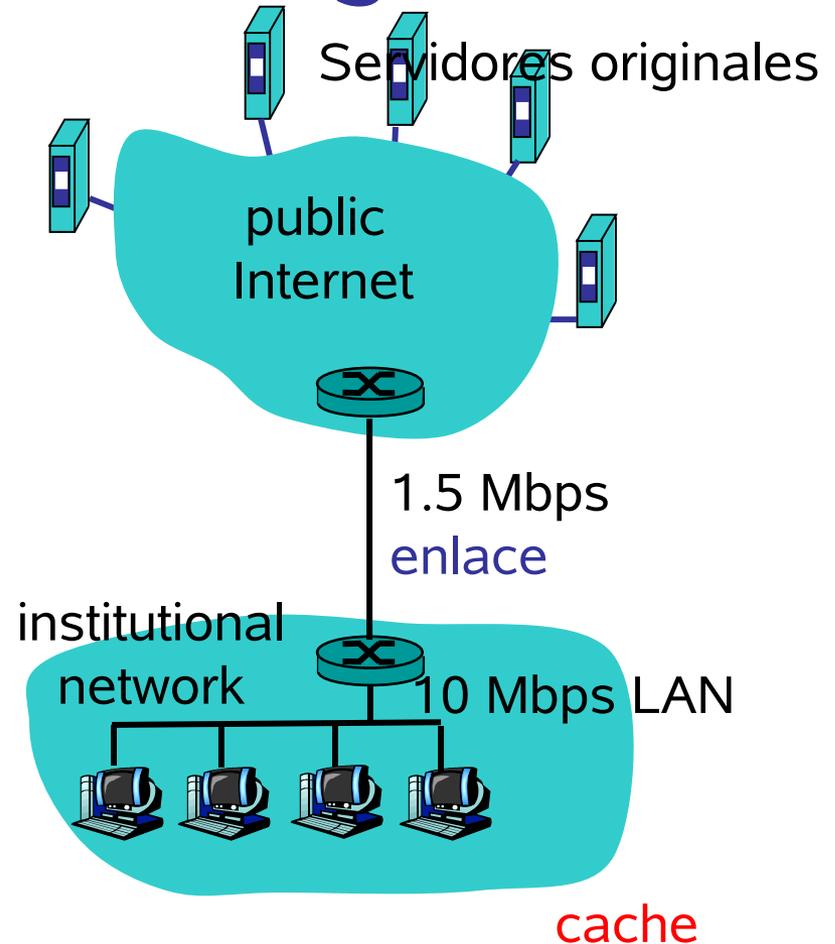
Ejemplo Caching

Asumimos

- Tamaño promedio = 100,000 bits
- Pedidos promedio a los servidores originales = 15/seg
- Retardo del router a un servidor original y de vuelta = 2 seg

Consecuencias

- Utilización de LAN = 15%
- Utilización de enlace = 100%
- Retardo total = retardo Internet + retardo acceso + retardo LAN = 2 seg + minutos + ms



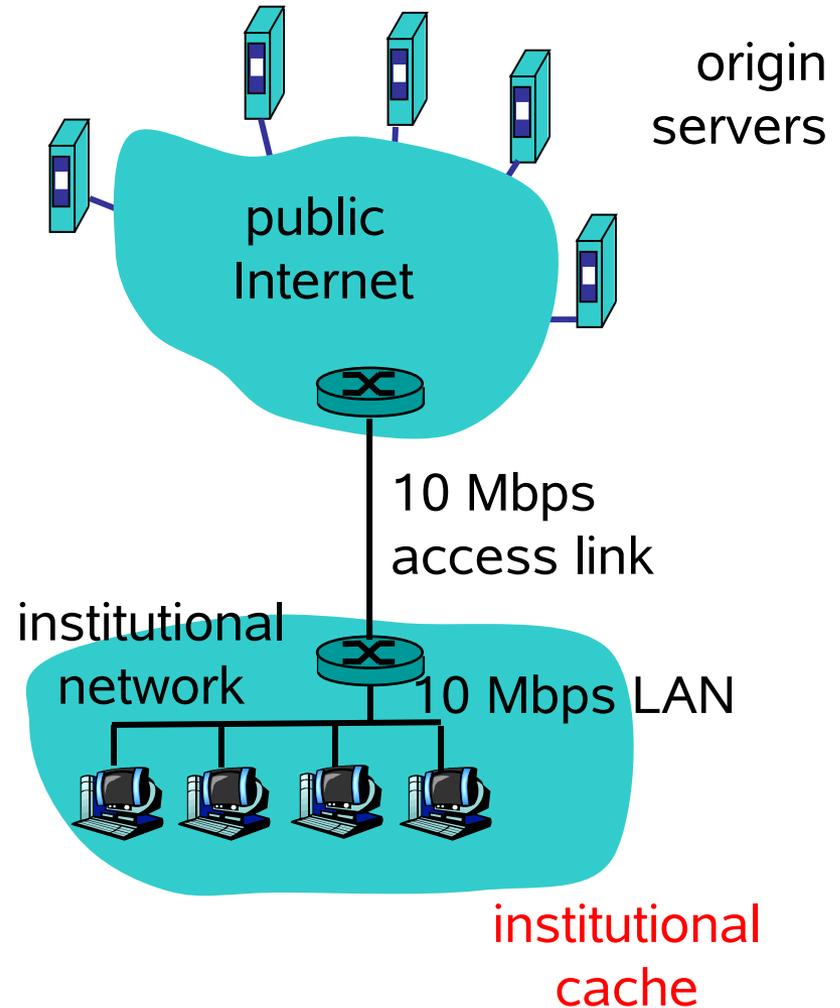
Caching example (cont)

Solucion posible

- Incrementar ancho de banda del enlace 10 Mbps

consequence

- utilización LAN = 15%
- Utilización enlace = 15%
- Retardo total = retardo Internet + retardo acceso + retardo LAN = 2 seg + ms + ms
- Mejora costosa



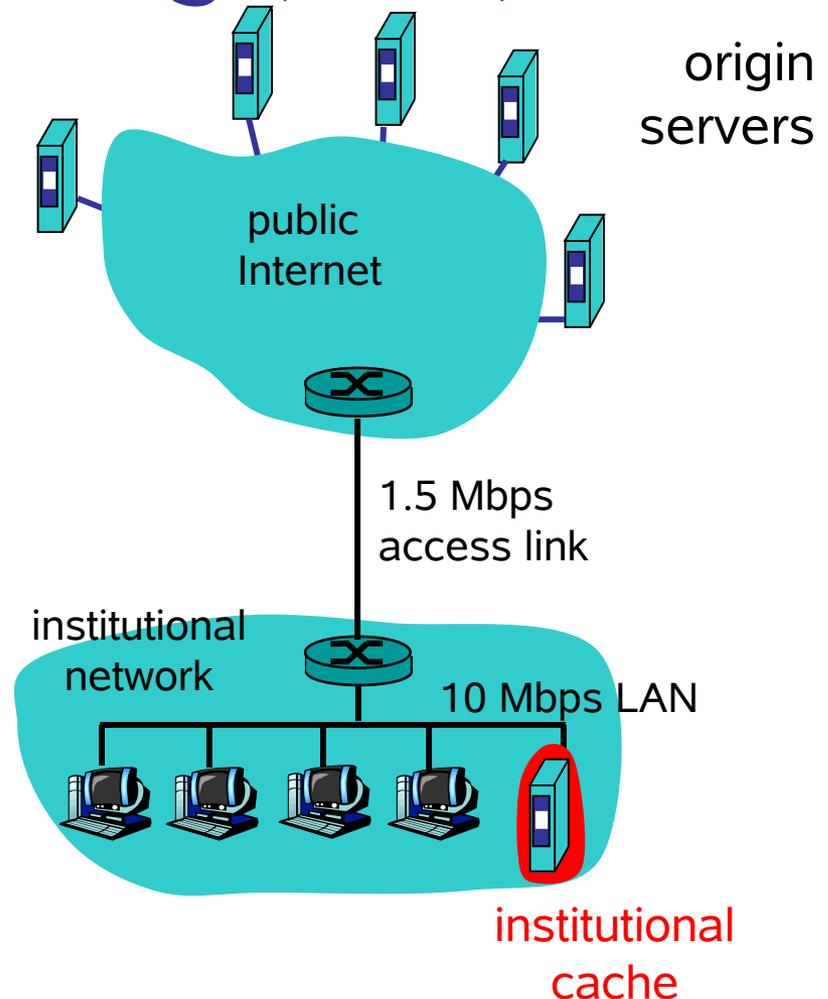
Ejemplo Caching (cont)

Posible solución cache

- Tasa de acceso 0.4

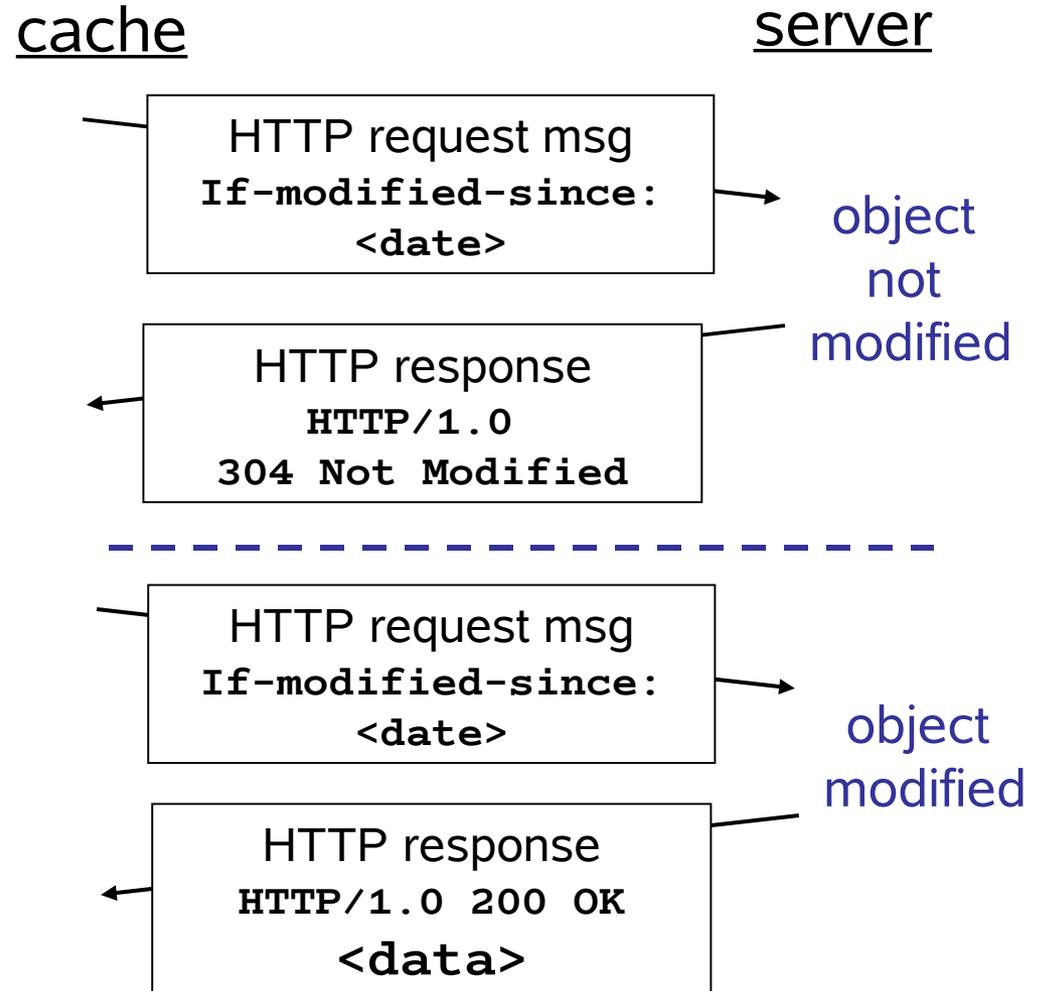
Consecuencias

- 40% de los pedidos se atienden casi de inmediato
- 60% de los pedidos son satisfechos por el servidor original
- Se reduce un 60% el uso del enlace, resultando en pocos retardos (10 msec)
- Retardo total = retardo Internet + retardo acceso + retardo LAN
= $0.6 \cdot (2.01) \text{ seg} + 0.4 \cdot \text{ms} < 1.4 \text{ s}$



GET Condicional

- **Objetivo:** no enviar objeto si el cache tiene una versión actualizada
- cache: especificar la fecha de la copia en HTTP request
If-modified-since: <date>
- server: response no contiene objeto si no fue modificado:
HTTP/1.0 304 Not Modified



Introducción a las redes de Computadoras

Capítulo 2

Clase 2

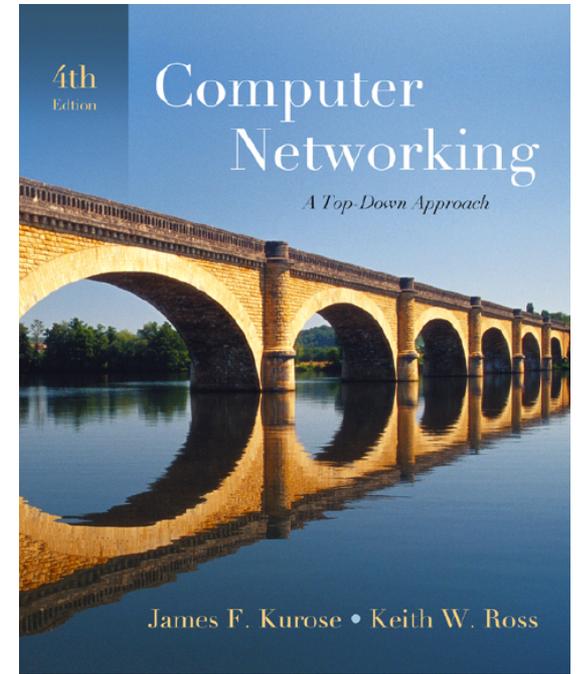
Nota acerca de las transparencias del curso:

Estas transparencias están basadas en el sitio web que acompaña el libro, y

han sido modificadas por los docentes del curso.

All material copyright 1996-2007

J.F Kurose and K.W. Ross, All Rights Reserved



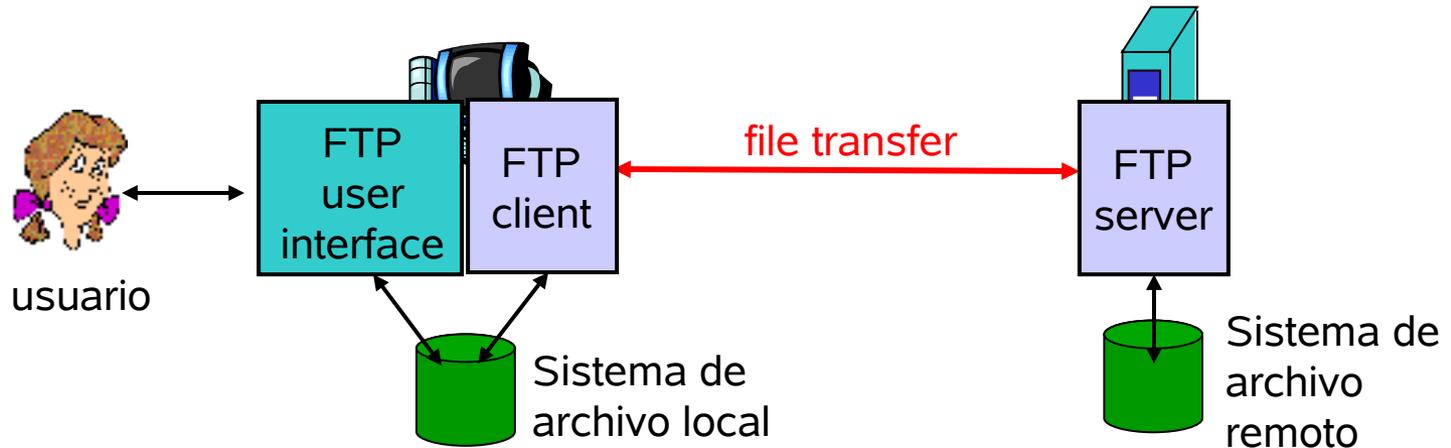
*Computer Networking: A
Top Down Approach,*
4th edition.

Jim Kurose, Keith Ross
Addison-Wesley, July
2007.

Capítulo 2: Capa de Aplicación

- ❑ 2.1 Principles of network applications
- ❑ 2.2 Web y HTTP
- ❑ 2.3 FTP
- ❑ 2.4 Correo Electronico
 - ❖ SMTP, POP3, IMAP
- ❑ 2.5 DNS
- ❑ 2.6 Aplicaciones P2P
- ❑ 2.7 Programación de Socket con TCP
- ❑ 2.8 Programación de Socket con UDP

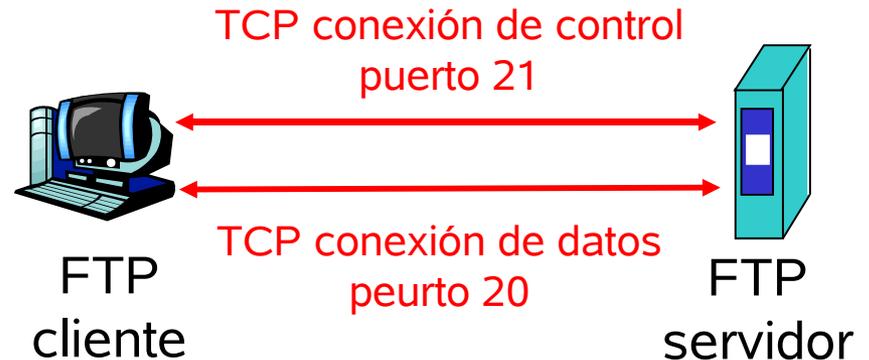
FTP: File Transfer Protocol



- ❑ Se transfiere al y desde el equipo remoto
- ❑ Arquitectura cliente/servidor
 - ❖ *cliente*: inicia la conexión
 - ❖ *servidor*: remote host
- ❑ ftp: RFC 959
- ❑ ftp servidor: puerto 21

FTP: separación control, datos

- ❑ Cliente FTP conecta al servidor FTP en el puerto 21, utilizando TCP como protocolo de transporte
- ❑ El cliente es autorizado en la conexión de control
- ❑ El cliente navega en el sistema de directorio enviando comandos en la conexión de control.
- ❑ Cuando el servidor recibe un comando de transferencia de archivo inicia una conexión TCP en el puerto 20
- ❑ Luego de transferir el archivo el servidor cierra la conexión



- ❑ El servidor abre otra conexión TCP para transferir otro archivo.
- ❑ La conexión de control se encuentra fuera de la transferencia de datos.
- ❑ Servidor FTP mantiene estado: directorio actual, autenticación

Comandos y respuestas FTP

Comandos:

- ❑ Enviados como texto ASCII
- ❑ **USER *username***
- ❑ **PASS *password***
- ❑ **LIST** devuelve la lista de archivos en el directorio actual
- ❑ **RETR filename** obtiene un archivo
- ❑ **STOR filename** guarda un archivo

Códigos de retorno

- ❑ Código de estado y descripción (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**

Capítulo 2: Capa de Aplicación

- ❑ 2.1 Principles of network applications
- ❑ 2.2 Web y HTTP
- ❑ 2.3 FTP
- ❑ 2.4 Correo Electronico
 - ❖ SMTP, POP3, IMAP
- ❑ 2.5 DNS
- ❑ 2.6 Aplicaciones P2P
- ❑ 2.7 Programación de Socket con TCP
- ❑ 2.8 Programación de Socket con UDP

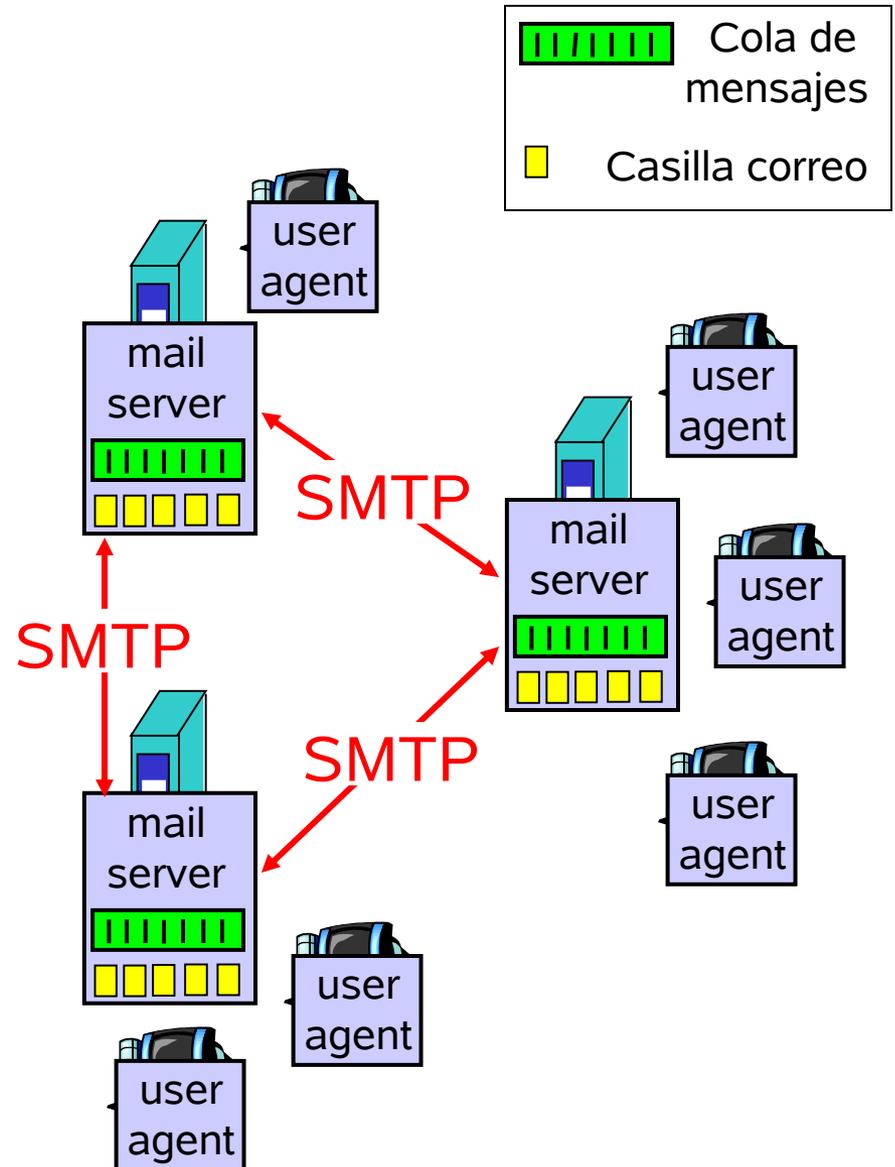
Correo Electrónico

Tres componentes:

- ❑ Usuarios
- ❑ Servidores
- ❑ SMTP: Simple Mail Transfer Protocol

Usuarios

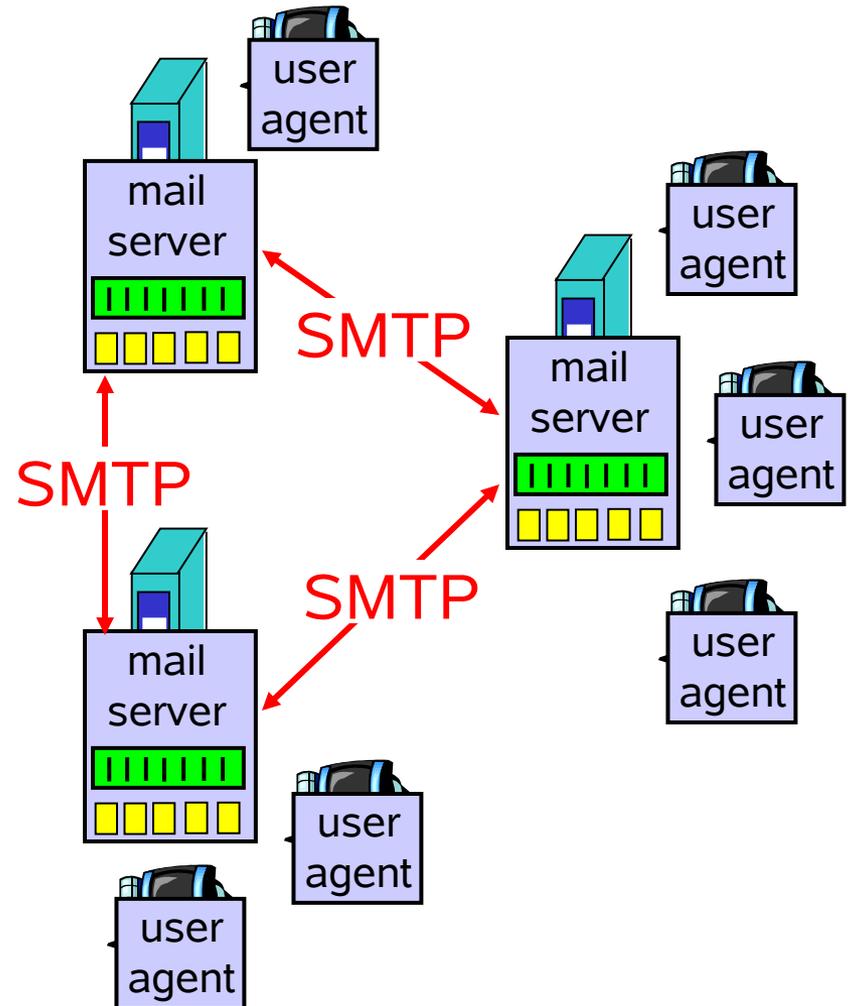
- ❑ Aplicación para leer correo
- ❑ Crear editar y leer mensajes
- ❑ Eudora, Outlook, elm, Mozilla Thunderbird
- ❑ Mensajes son guardados en servidor



Correo Electrónico: Servidores

Servidores

- ❑ Casilla de correo (mailbox) contiene los mensajes entrantes
- ❑ Cola de mensajes (message queue) mensajes salientes
- ❑ Protocolo SMTP entre usuarios y servidores
 - ❖ cliente: envia mail
 - ❖ servidor: recibe mail

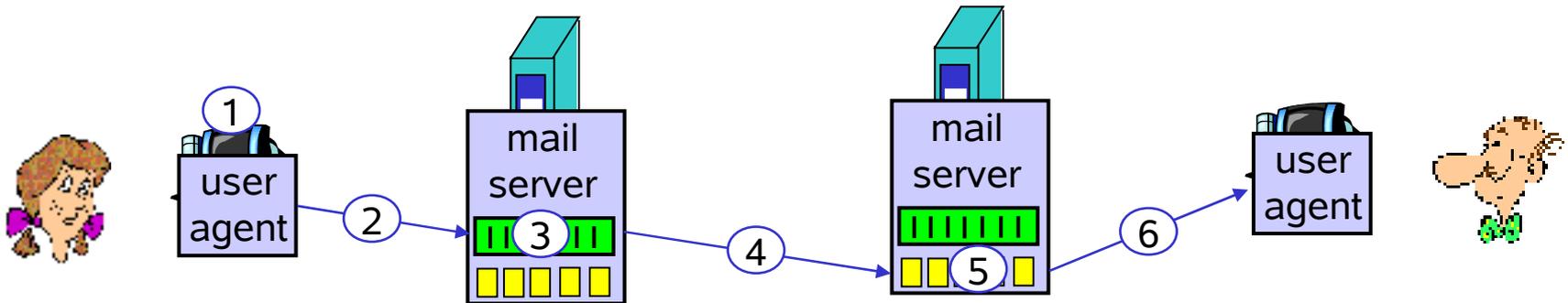


Correo Electrónico: SMTP [RFC 2821]

- ❑ Utiliza TCP para enviar mensajes en forma confiable del cliente al servidor en el puerto 25
- ❑ Tres fases de transferencia
 - ❖ Saludo (handshaking)
 - ❖ Transferencia de mensajes
 - ❖ Finalización
- ❑ Conexión directa servidor - servidor
- ❑ Interacción comandos y respuestas
 - ❖ **comandos:** texto ASCII
 - ❖ **respuesta:** código de estado y descripción
- ❑ Mensajes en ASCII 7-bit

Escenario: Alicia envía un mensaje a Roberto

- 1) Alicia usa aplicación para crear mensaje a roberto@fing.edu.uy
- 2) Alicia usa aplicación para enviar mensaje a su servidor de correo. El mensaje es puesto en una cola de mensajes
- 3) El servidor de Alicia abre una conexión TCP con el servidor de Roberto
- 4) El servidor SMTP de Alicia envía el mensaje por la conexión TCP
- 5) El servidor de Roberto coloca el mensaje en la casilla de Roberto
- 6) Roberto usa su aplicación para leer el mensaje



Ejemplo SMTP

```
220 smtp-s03.adinet.com.uy ESMTP Service ready
HELO notebook
250 smtp-s03.adinet.com.uy
MAIL FROM: <gabgg@adinet.com.uy>
250 MAIL FROM:<gabgg@adinet.com.uy> OK
RCPT TO: <gabgg@adinet.com.uy>
250 RCPT TO:<gabgg@adinet.com.uy> OK
DATA
354 Start mail input; end with <CRLF>.<CRLF>
Esto es una prueba linea 1
Esto es una prueba linea 2
.
250 <47D975C5006879DA> Mail accepted
QUIT
221 smtp-s03.adinet.com.uy QUIT
```

Try SMTP interaction for yourself:

- ❑ `telnet adinet.com.uy 25`
- ❑ Esperar respuesta 220
- ❑ Ingresar comandos HELO, MAIL FROM, RCPT TO, DATA, QUIT

SMTP

- ❑ SMTP usa conexiones persistentes
- ❑ SMTP requiere que el mensaje (cabezal y cuerpo) este en ASCII 7-bit
- ❑ SMTP usa CRLF .CRLF para determinar el fin de mensaje

Comparación con HTTP:

- ❑ HTTP: se extraen datos del servidor
- ❑ SMTP: se envían datos
- ❑ Ambos tienen comandos ASCII con estados
- ❑ HTTP: cada objeto está encapsulado en su propio mensaje
- ❑ SMTP: muchos objetos en un solo mensaje

Formato del mensaje

SMTP: protocolo para intercambiar mensajes de correo

RFC 822: estándar para formato del mensaje

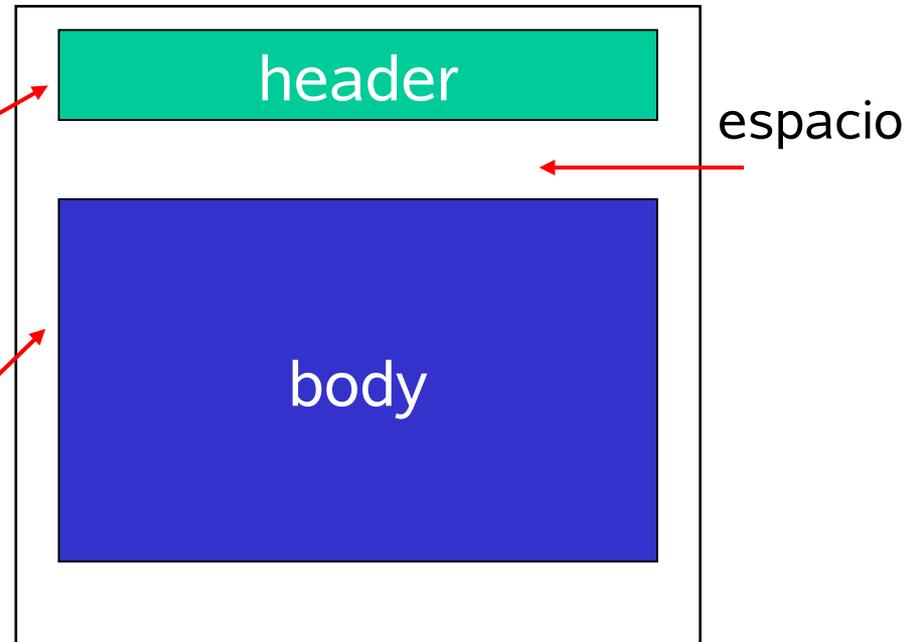
☐ Cabezal (header)

- ❖ To:
- ❖ From:
- ❖ Subject:

diferente de comandos SMTP

☐ Cuerpo (body)

- ❖ el mensaje, soloASCII



Formato del mensaje: extensiones multimedia

- ❑ MIME: multimedia mail extension, RFC 2045, 2056
- ❑ líneas adicionales declaran el tipo de contenido MIME

Versión MIME

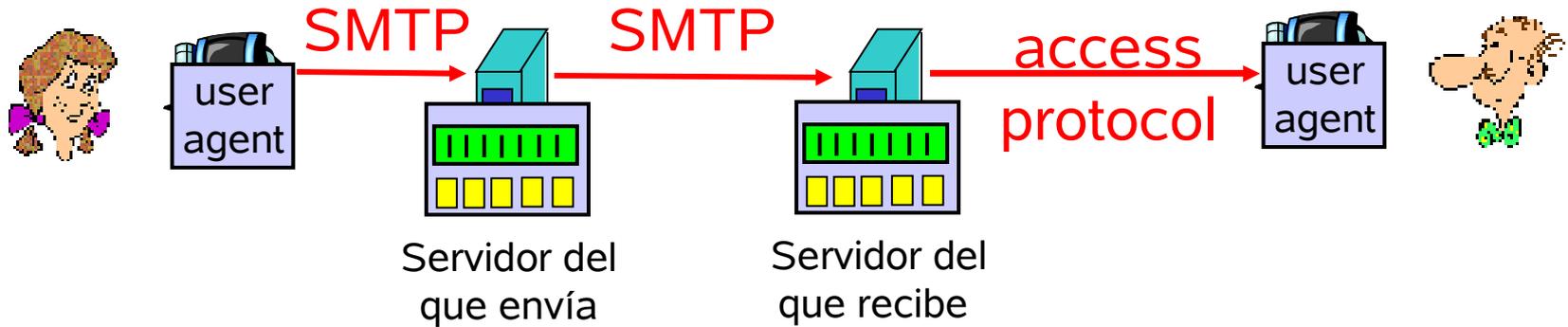
Método de codificación

multimedia tipo,
subtipo,
declaración

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
```

Protocolo de acceso a correo



- SMTP: envío/almacenamiento
- Protocolo de acceso a correo: obtener del servidor
 - ❖ POP: Post Office Protocol [RFC 1939]
 - autorización (usuario <--> servidor) y bajada
 - ❖ IMAP: Internet Mail Access Protocol [RFC 1730]
 - Mas funcionalidad (mas complejo)
 - Manipulación de mensajes almacenados en el servidor
 - ❖ HTTP: gmail, Hotmail, Yahoo! Mail, etc.

POP3

Autorización

- Comandos del cliente:
 - ❖ **user**: declare username
 - ❖ **pass**: password
- Respuestas del servidor
 - ❖ **+OK**
 - ❖ **-ERR**

Interacción

- **list**: número de mensajes
- **retr**: obtiene mensaje por número
- **dele**: borra mensaje
- **quit**

```
S: +OK POP3 server ready
C: user roberto
S: +OK
C: pass roberto123
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

More about POP3

- ❑ En el ejemplo se obtiene y borra el mensaje
- ❑ Roberto no puede volver a leer el mensaje en otro cliente de correo
- ❑ Se puede obtener el mensaje sin borrar
- ❑ POP3 no tiene estado entre sesiones

IMAP

- ❑ Se guardan todos los mensajes en el servidor
- ❑ Se pueden organizar los mensajes en directorios
- ❑ IMAP mantiene estado entre sesiones:
 - ❖ Nombres de directorios, mensajes y directorios.

Introducción a las Redes de Computadoras

Capítulo 2

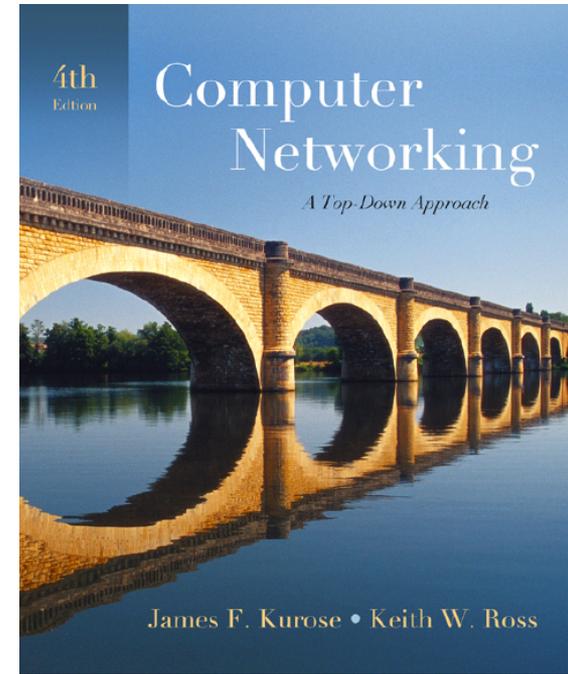
Clase 3

Nota acerca de las transparencias:

Estas transparencias están basadas en el sitio web que acompaña el libro, y han sido modificadas por los docentes del curso.

All material copyright 1996-2007

J.F Kurose and K.W. Ross, All Rights Reserved



*Computer Networking: A
Top Down Approach,
4th edition.*

Jim Kurose, Keith Ross
Addison-Wesley, July
2007.

Capítulo 2: Capa de aplicación

- 2.1 Principio de aplicaciones de red
- 2.2 Web y HTTP
- 2.3 FTP
- 2.4 Correo electrónico
 - ❖ SMTP, POP3, IMAP
- 2.5 DNS
- 2.6 Aplicaciones P2P

DNS: Domain Name System

Personas: como identificarlas

- ❖ CI, nombre, pasaporte

Equipos en internet:

- ❖ Direcciones IP(32 bit)
- ❖ nombre, ej:,
ww.yahoo.com usado por personas

Pregunta: como se relaciona la dirección IP y el nombre?

Domain Name System:

- ❑ *Base de datos distribuída* implementada en una jerarquía de muchos *servidores de nombres*
- ❑ *Protocolo de capa de aplicación* usado para resolver nombres

DNS

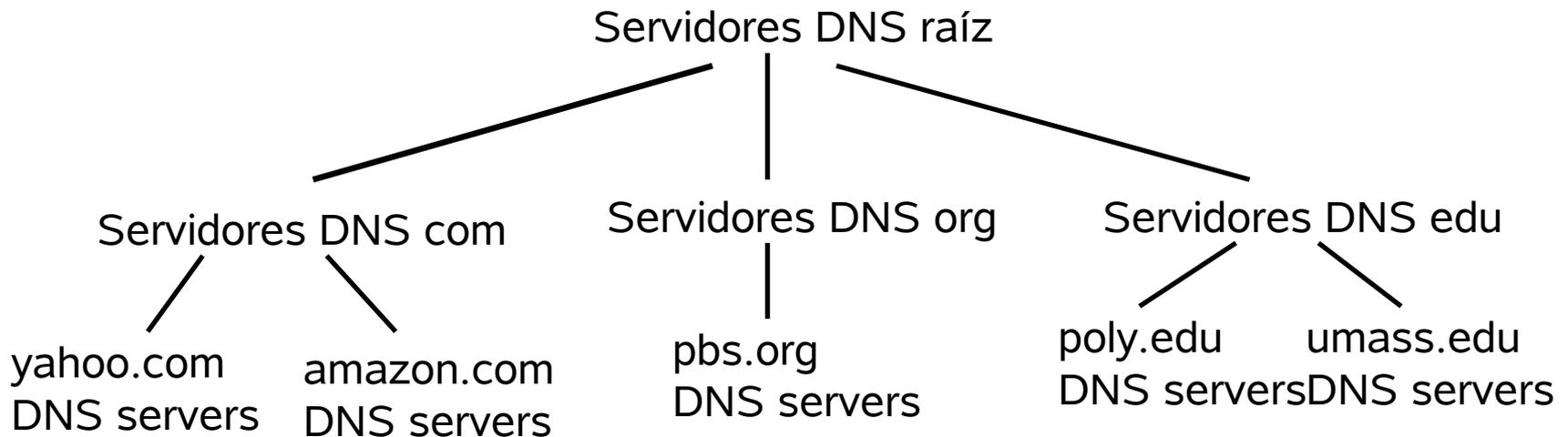
Servicios DNS

- ❑ Traducción de nombre a dirección IP
- ❑ host alias
 - ❖ Canónicos, alias
- ❑ Alias servidor de correo
- ❑ Distribución de carga
 - ❖ Servidores replicados

Por que no DNS centralizado?

- ❑ Punto de falla único
- ❑ Volumen de tráfico
- ❑ Base de datos centralizada distante
- ❑ Mantenimiento
- ❑ NO ESCALA

Base de datos distribuída jerárquica

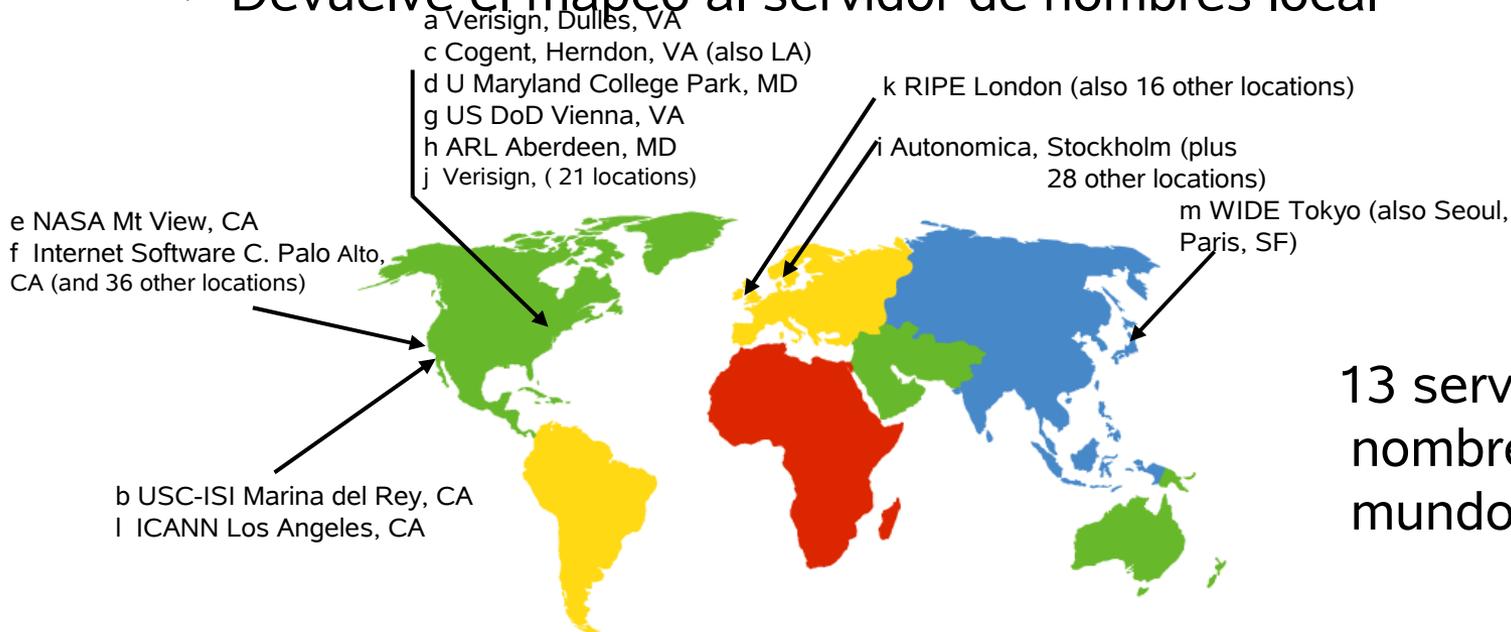


Cliente quiere la dirección IP de www.amazon.com

- ❑ Cliente consulta servidor DNS raíz para obtener servidor DNS com
- ❑ Cliente consulta servidor DNS com para obtener servidor DNS Amazon
- ❑ Cliente consulta servidor DNS amazon para obtener la dirección IP de www.amazon.com

DNS: Servidores raíz

- ❑ Contactados por servidores de nombre locales que no pueden resolver un nombre
- ❑ Servidor de nombres raíz:
 - ❖ Consulta servidor autoritativo si no conoce el mapeo de nombres
 - ❖ Obtiene el mapeo
 - ❖ Devuelve el mapeo al servidor de nombres local



13 servidores de nombres raíz en el mundo

TLD y Servidores autoritativos

□ Servidores Top-level domain (TLD):

- ❖ responsables de com, org, net, edu, etc, y todos los dominios de países uk, fr, ca, jp, uy
- ❖ Network Solutions mantiene los servidores com
- ❖ Educause mantiene los servidores edu

□ Servidores autoritativos:

- ❖ Servidores DNS de organizaciones, proveen mapeos autoritativos (el servidor tiene la autoridad por el registro por el que se consulta, implica que es uno de los servidores del dominio que se consulta)
- ❖ Puede ser mantenido por la organización o el proveedor de servicio

Servidor de nombres local

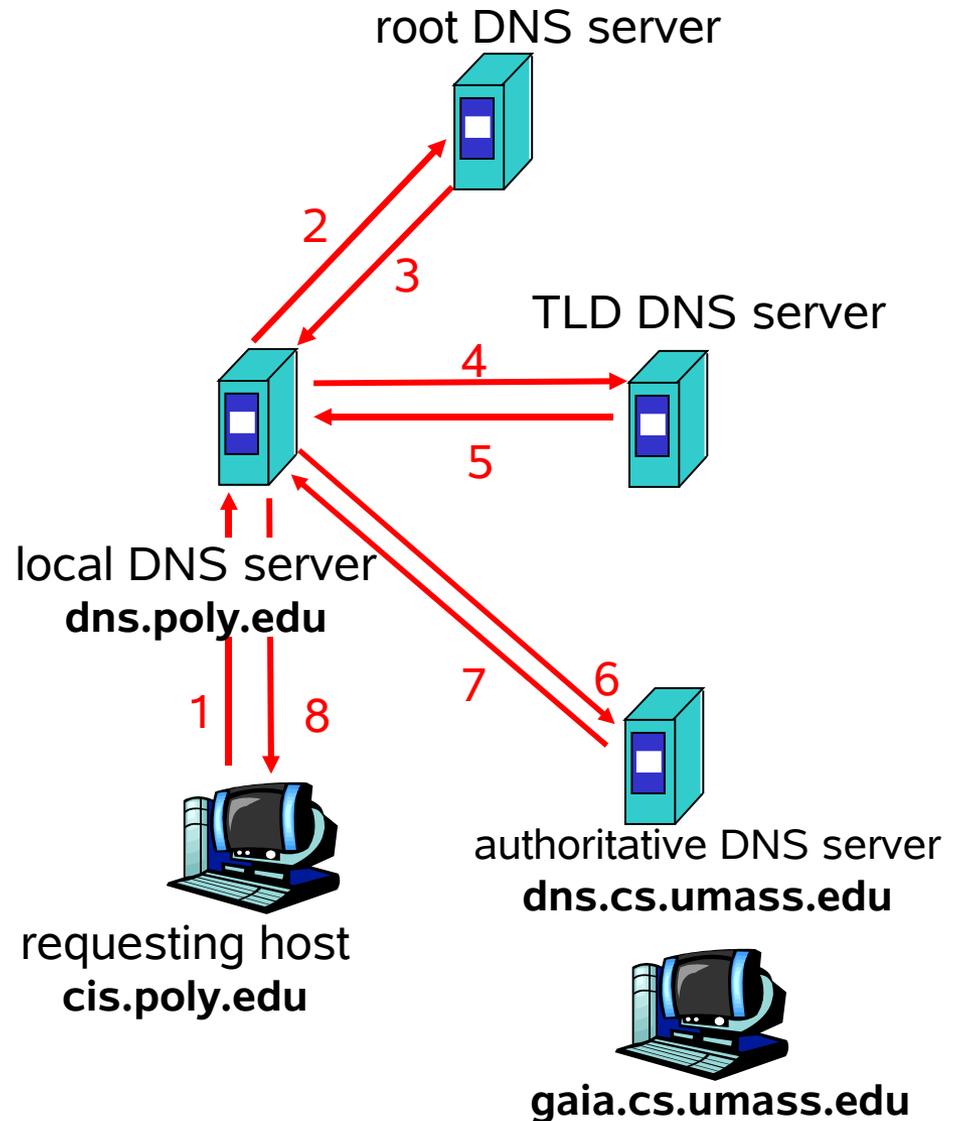
- ❑ No pertenece a la jerarquía
- ❑ Cada ISP (residencial, compañía, universidad) tiene uno.
 - ❖ Se llama “default name server”
- ❑ Cuando un equipo hace una consulta DNS, se envía al servidor local
 - ❖ Actúa como proxy, reenvía la consulta a la jerarquía

DNS ejemplo de resolución

- ❑ Equipo en cis.poly.edu quiere la dirección IP de gaia.cs.umass.edu

Consulta iterativa:

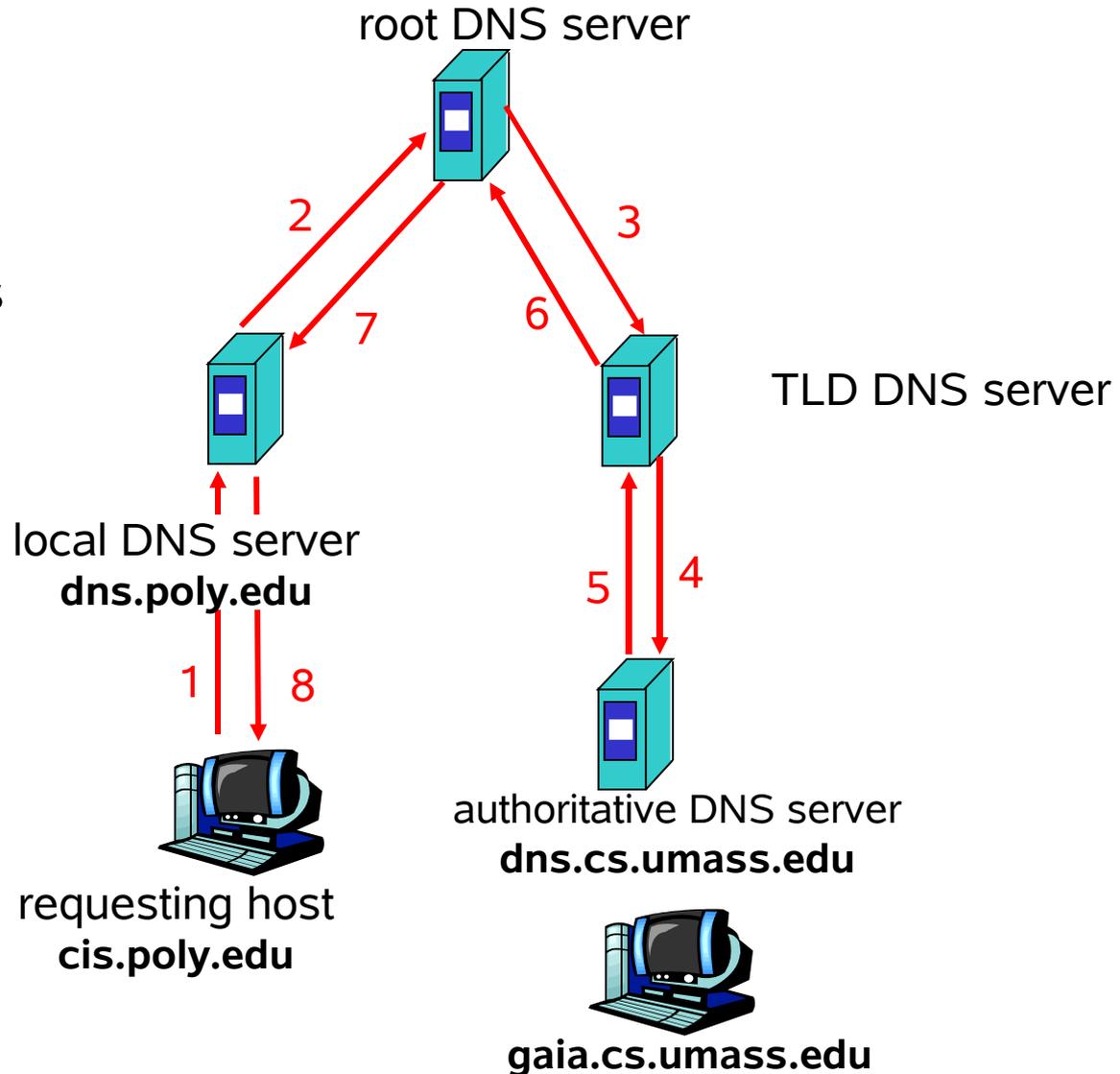
- ❑ El servidor responde con el servidor al cual contactar
- ❑ “No conozco ese nombre pero pregunta a este servidor”



DNS ejemplo de resolución

Consulta recursiva:

- El servidor de nombres se encarga de la resolución



DNS: caching y actualizar registros

- Una vez que el servidor aprende un mapeo, lo cachea
 - ❖ Las entradas vencen despues de cierto tiempo
 - ❖ TLD típicamente cacheados en servidores locales
 - Servidores raíz no son visitados frecuentemente
- Mecanismos de actualización y notificación diseñados por IETF (Internet Engineering Task Force)
 - ❖ RFC 2136
 - ❖ www.ietf.org/proceedings/96dec/charters/dnsind-charter.html

Registros DNS

DNS: base de datos distribuída que almacena los registros de recursos (RR)

Formato RR: (**nombre**, **valor**, **tipo**,
t11)

- Tipo=A
 - ❖ **nombre** es el nombre del equipo
 - ❖ **valor** es la dirección IP
- Tipo=NS
 - ❖ **nombre** es el dominio
 - ❖ **valor** es el nombre del servidor autoritativo para el dominio
- Otros: PTR / SOA / HINFO / TXT / LOC / WKS /SRV / SPF
- Tipo=CNAME
 - ❖ **nombre** es el alias para un nombre “canónico” (real)
`www.ibm.com` es realmente
`servereast.backup2.ibm.com`
Ej:
`dominio=ejemplo.com`
`servicio ftp=ftp.ejemplo.com`
 - ❖ **valor** es el nombre canónico
- Tipo=MX
 - ❖ **valor** es el nombre del servidor de correo asociado con **nombre**

Protocolo DNS, mensajes

Protocolo DNS: consulta y respuesta, mismo formato

Cabezal

- ❑ **identification**: 16 bit
- ❑ **flags**:
 - ❖ Consulta o respuesta
 - ❖ Si se usa recursiva
 - ❖ Si la respuesta es autoritativa

identification	flags
number of questions	number of answer RRs
number of authority RRs	number of additional RRs
questions (variable number of questions)	
answers (variable number of resource records)	
authority (variable number of resource records)	
additional information (variable number of resource records)	



Protocolo DNS protocol, mensajes

Nombre, tipo y campos de una consulta

RRs respuesta a una consulta

Registros para servidores autoritativos usados para seguir con la consulta

Información adicional

identification	flags
number of questions	number of answer RRs
number of authority RRs	number of additional RRs
questions (variable number of questions)	
answers (variable number of resource records)	
authority (variable number of resource records)	
additional information (variable number of resource records)	

↑
12 bytes
↓

Insertar registros en DNS

- Ejemplo: nuevo “Network Utopia”
- Registrar nombre networkutopia.com en *DNS* (ej: Network Solutions)
 - ❖ Proveer nombres, direcciones IP y servidores autoritativos (primario y secundario)
 - ❖ Se insertan dos registros RR en servidor TLD:

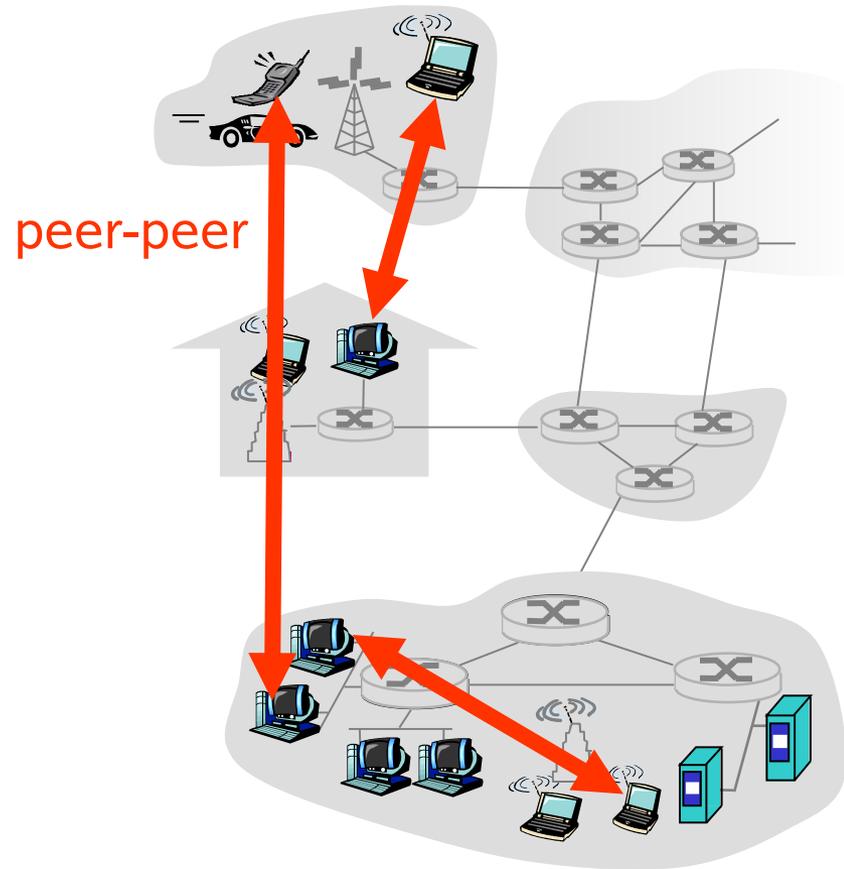
`(networkutopia.com, dns1.networkutopia.com, NS)`
`(dns1.networkutopia.com, 212.212.212.1, A)`
- Crear servidor registro de servidor autoritativo Tipo A para `www.networkutopia.com`;
Tipo MX `networkutopia.com`
- **Como las personas obtienen la dirección IP del sitio?**

Chapter 2: Application layer

- ❑ 2.1 Principio de aplicaciones de red
- ❑ 2.2 Web y HTTP
- ❑ 2.3 FTP
- ❑ 2.4 Correo electrónico
 - ❖ SMTP, POP3, IMAP
- ❑ 2.5 DNS
- ❑ 2.6 Aplicaciones P2P
- ❑ 2.7 Programación de sockets con TCP
- ❑ 2.8 Programación de sockets con UDP

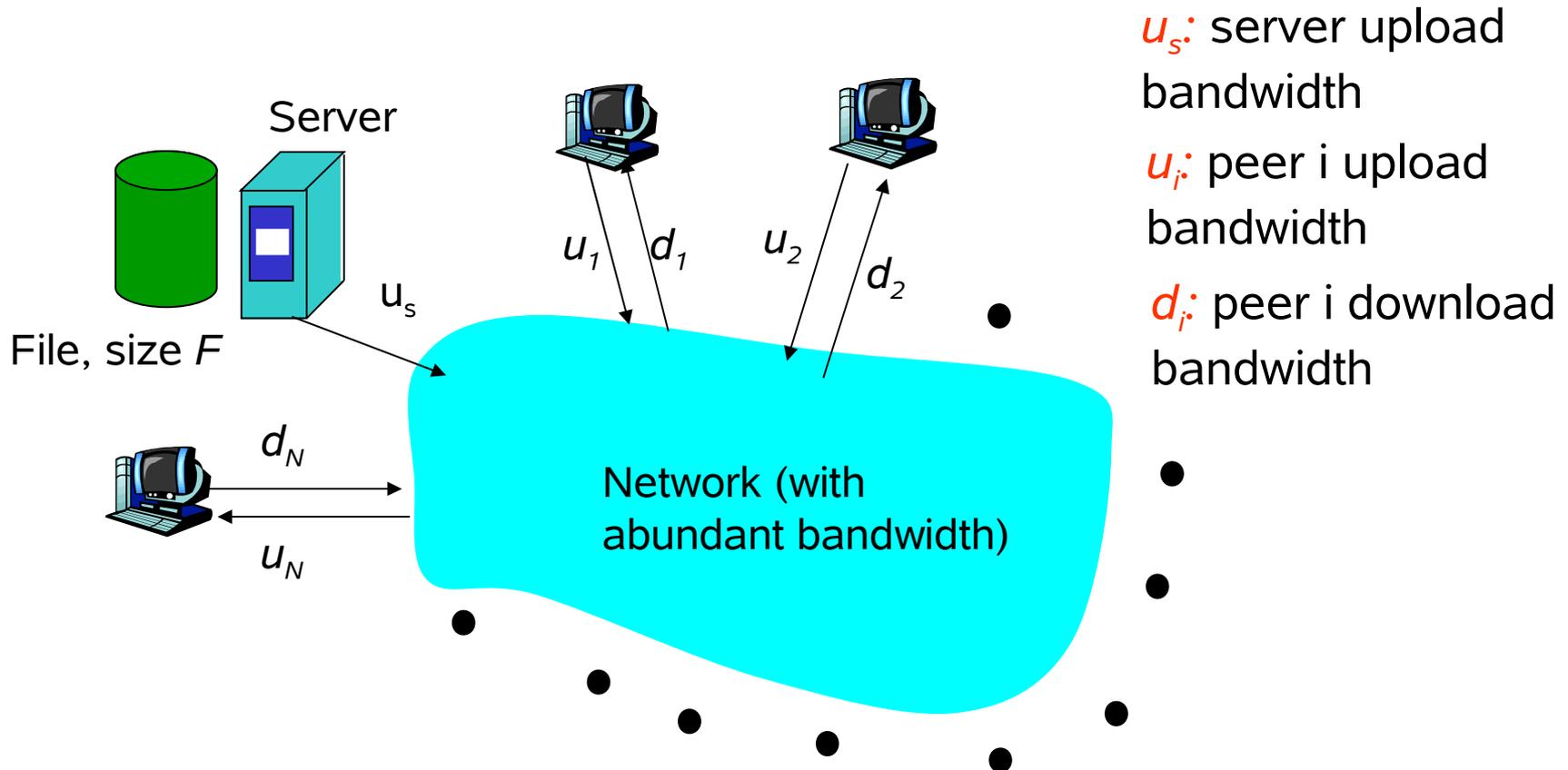
Arquitectura P2P pura

- ❑ *Servidor intermitente*
- ❑ Se comunican sistemas finales variados
- ❑ Cambian direcciones IP
- ❑ Temas de estudio:
 - ❖ Distribución de archivos
 - ❖ Búsqueda de información
 - ❖ Caso: Skype



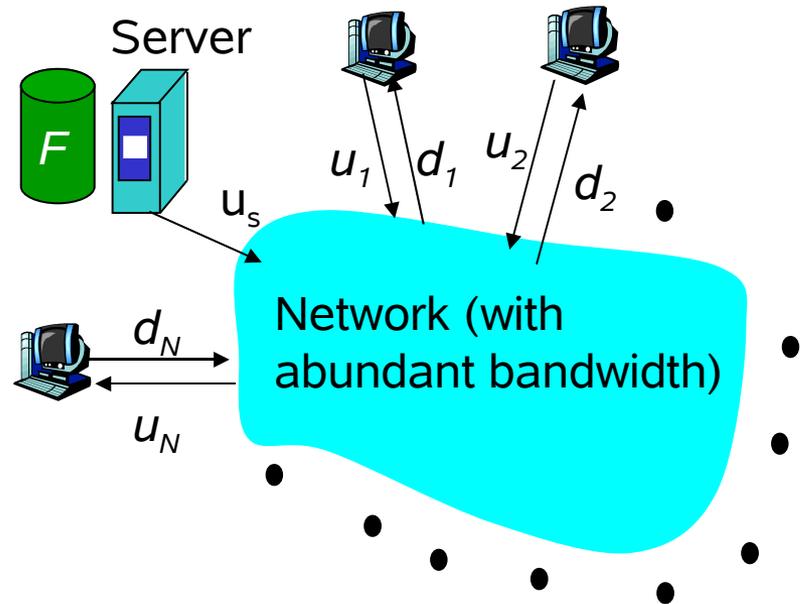
Distribución de archivos: cliente/servidor vs P2P

Cuanto tiempo lleva distribuir un archivo de un servidor a N pares?



Tiempo de distribución de archivo: cliente/servidor

- Servidor envia N copias:
 - ❖ NF/u_s
- cliente i F/d_i para bajar



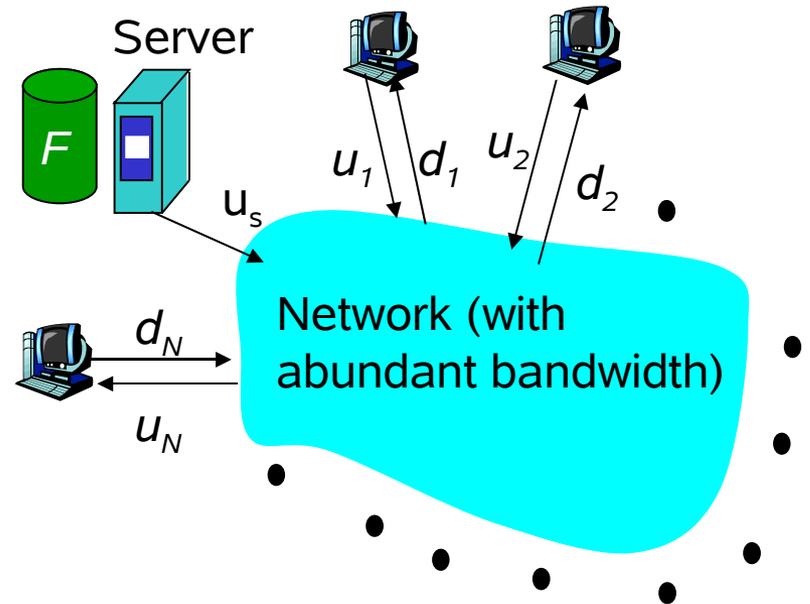
Tiempo para distribuir F

a N clientes $t_{cs} = \max \{ NF/u_s, F/\min_i(d_i) \}$

Incrmenta linealmente en N

Tiempo de distribución de archivo: P2P

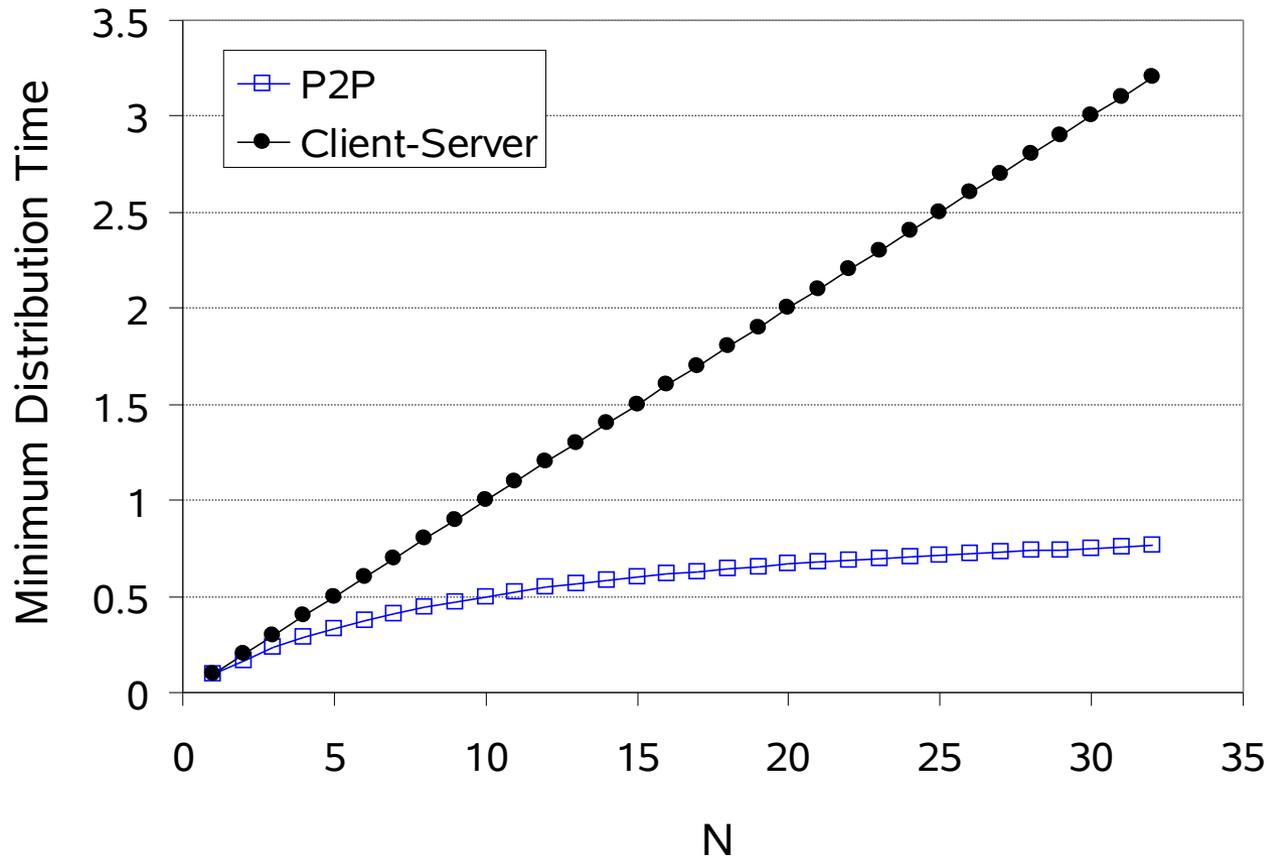
- ❑ Servidor debe enviar 1 copia F/u_s
- ❑ cliente i F/d_i para bajar
- ❑ Se deben bajar NF bits
- ❑ Subida posible: $u_s + \sum u_i$



$$d_{\text{P2P}} = \max \left\{ F/u_s, F/\min(d_i), NF/(u_s + \sum u_i) \right\}$$

Cliente/Servidor vs. P2P: ejemplo

Subida cliente = u , $F/u = 1$ hour, $u_s = 10u$, $d_{\min} \geq u_s$

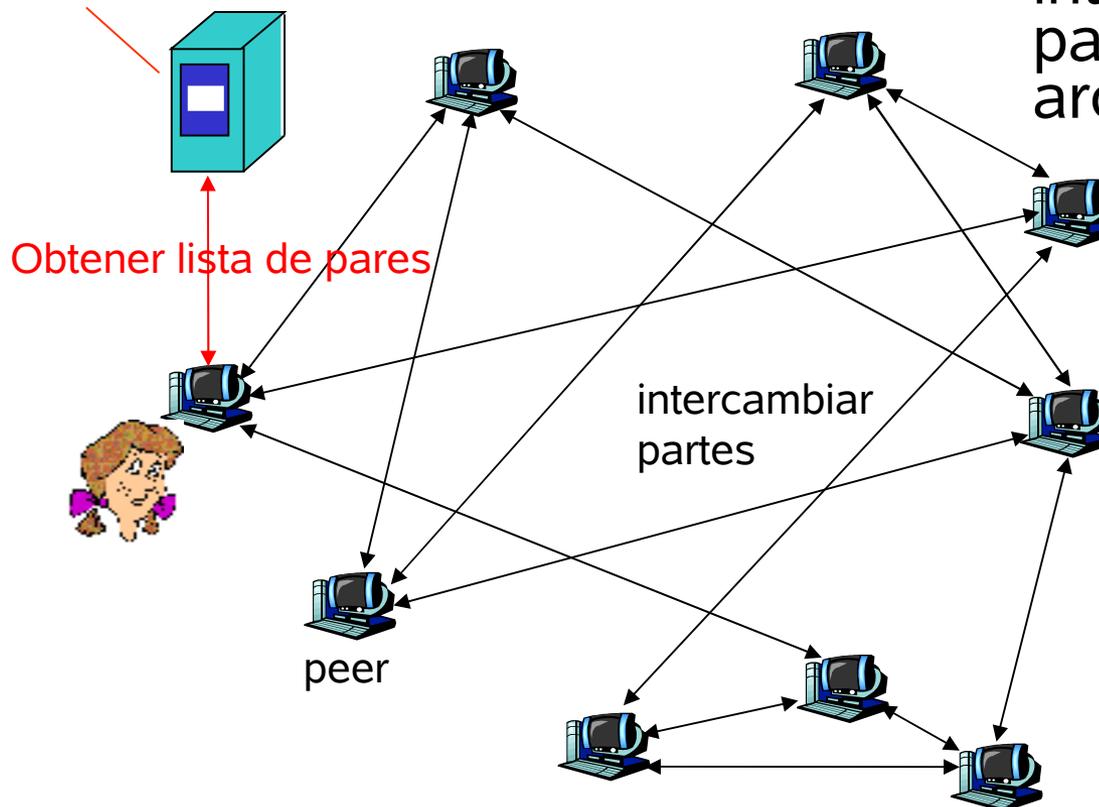


Distribución de archivos: BitTorrent

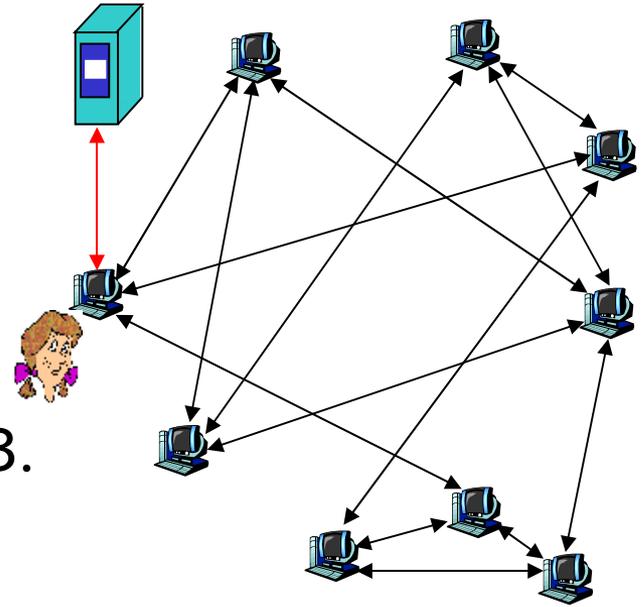
□ Distribución de archivos P2P

tracker: encuentra pares participando en torrent

torrent: grupo de pares intercambiando partes de un archivo



BitTorrent (1)



- ❑ Archivo dividido en partes de 256KB.
- ❑ Par uniéndose al torrent:
 - ❖ No tiene partes, las acumula luego
 - ❖ Se registra con tracker para obtener lista de pares, se conecta a un subconjunto de pares (vecinos)
- ❑ Mientras baja, sube partes a otros pares
- ❑ Los pares se conectan y desconectan
- ❑ Cuando un par tiene un archivo puede desconectarse o seguir conectado para seguir compartiéndolo

BitTorrent (2)

Obtener partes

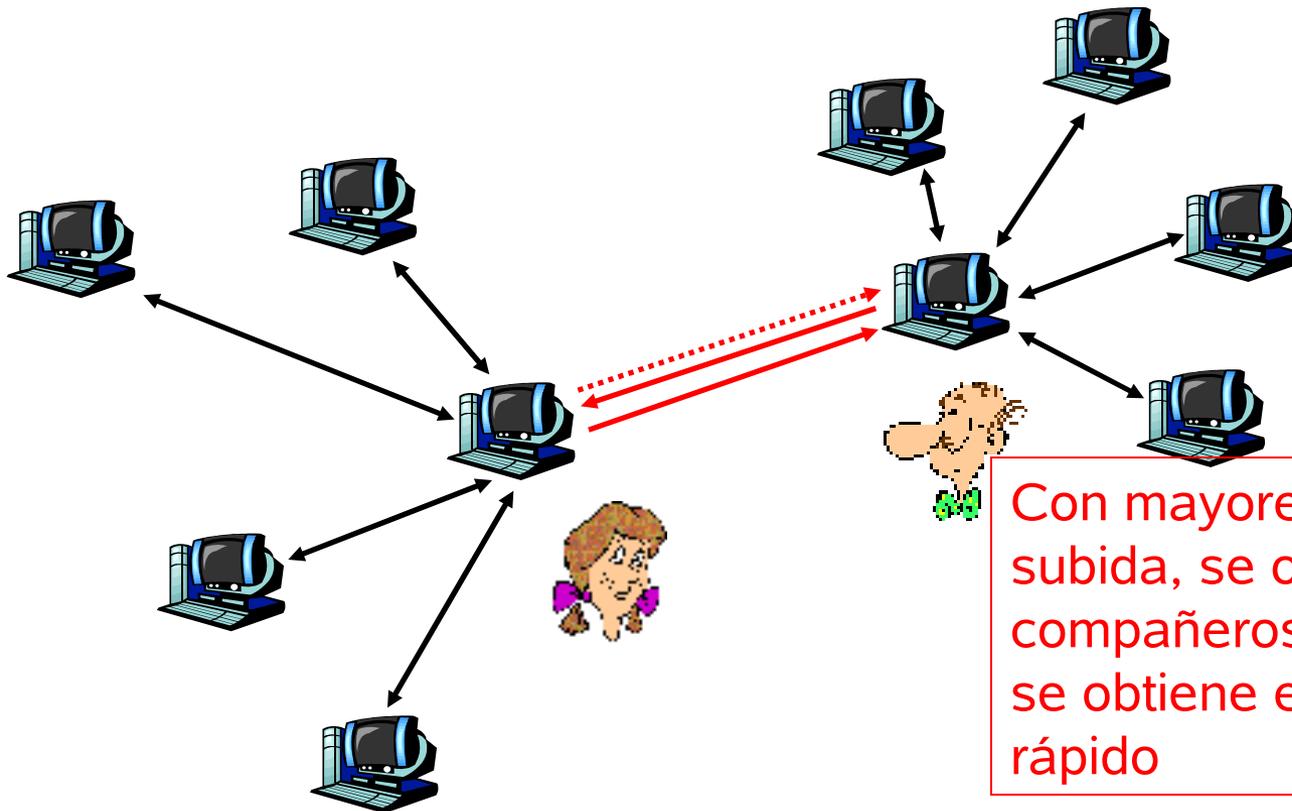
- ❑ En cierto momento, varios pares tienen diferentes conjuntos de partes de un archivo
- ❑ Periodicamente cada par pregunta a sus vecinos por la lista de partes que tienen.
- ❑ Cada par envía pedidos por las partes que le faltan
 - ❖ Primero las partes “raras”

Enviar partes: tit-for-tat

- ❑ Se envían partes a los 4 vecinos enviándole partes a la mayor velocidad
 - ❖ Se reevalúan los 4 cada 10 segundos
- ❑ Cada 30 segundos: se selecciona otro par al azar, se comienza a enviar partes
 - ❖ El nuevo puede unirse a los 4
 - ❖ “desbloqueo optimista”

BitTorrent: Tit-for-tat

- (1) Alicia “desbloquea optimísticamente” a Roberto
- (2) Alicia es uno de los 4 de Roberto; Roberto devuelve el favor
- (3) Roberto se convierte en uno de los 4 de Alicia



Con mayores velocidades de subida, se obtiene mejores compañeros de intercambio y se obtiene el archivo mas rápido

P2P: Búsqueda de información

Índice en un sistema P2P: mapea la información de localización de un par (localización = IP y puerto)

- Compartir archivos
(ej e-mule)

- ❑ Índice dinámicamente ubica los archivos que comparten los pares.
- ❑ Los pares informan que archivos tienen
- ❑ Los pares buscan el índice para ver que archivos pueden obtener

- Mensajería instantánea

- ❑ Índice mapea usuarios y localizaciones
- ❑ Cuando el usuario inicia aplicación, informa al índice su localización
- ❑ Los pares buscan el índice para determinar la dirección del usuario

P2P: índice centralizado

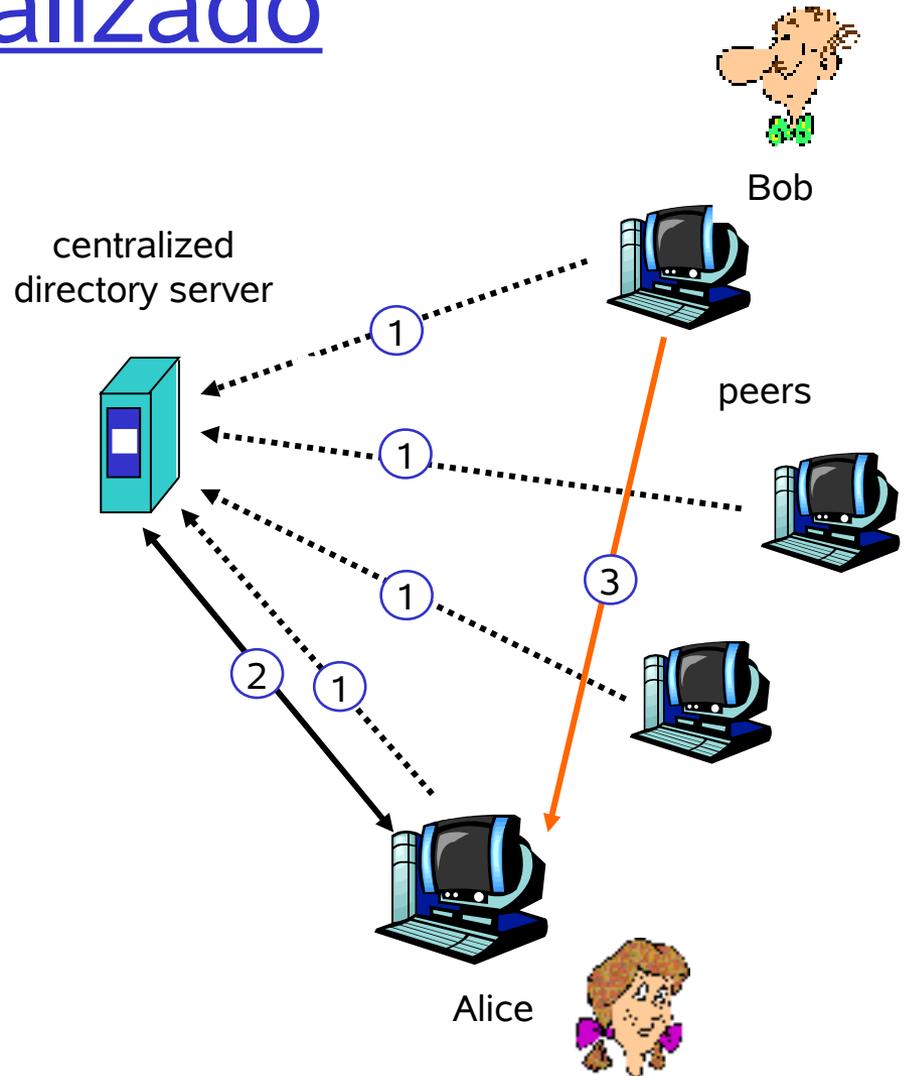
Diseño “Napster”

1) Cuando se conecta cada par, informa al servidor:

- ❖ IP
- ❖ contenido

2) Alicia busca la canción “Yendo a la casa de Damián”

3) Alice pide el archivo a Roberto



P2P: problemas con directorios centralizados

- ❑ Punto de falla único
- ❑ Cuello de botella
- ❑ Violación de copyright

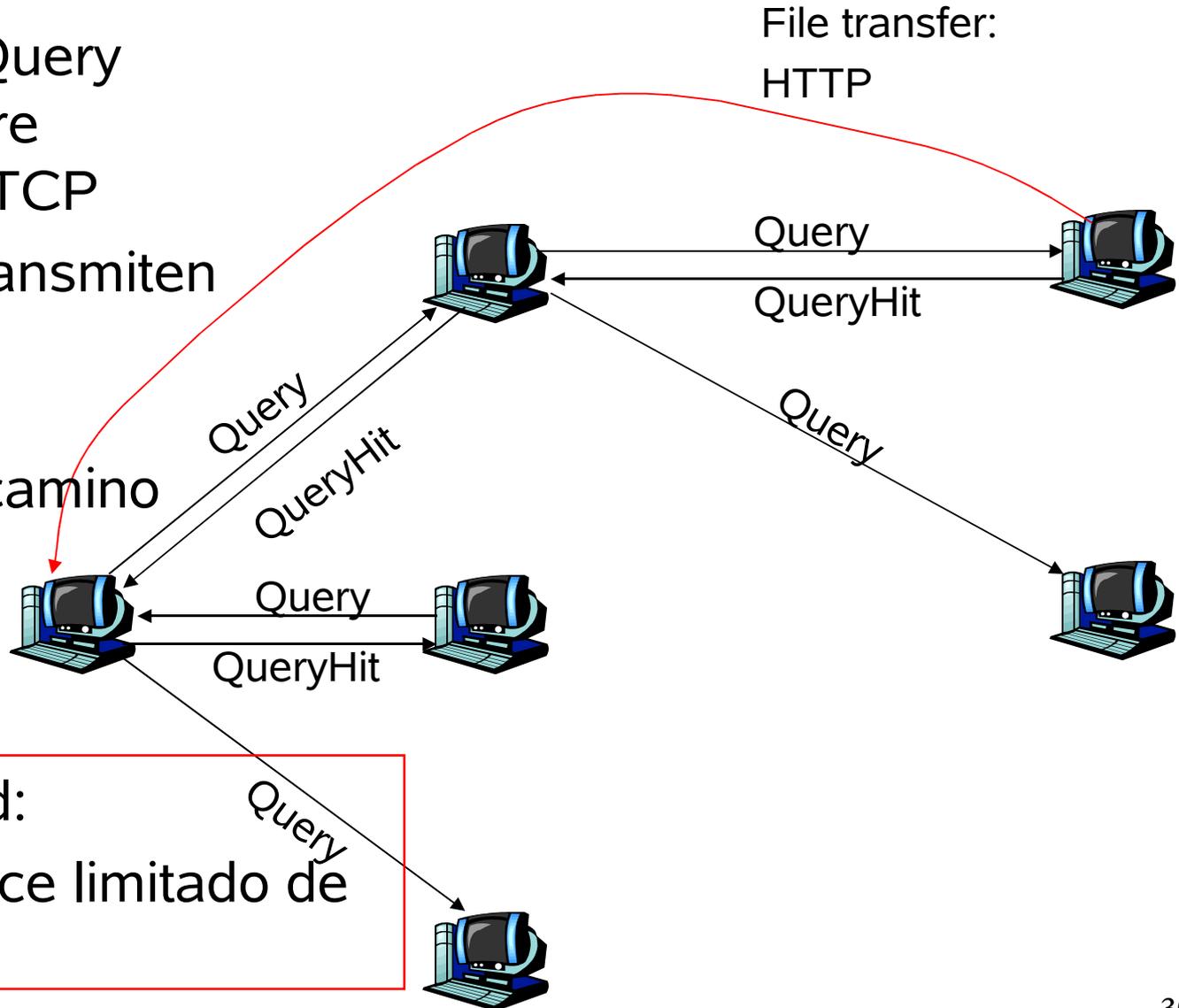
la transferencia de archivos es descentralizada, pero localizar el contenido es centralizado

Inundar consultas

- ❑ Totalmente distribuido
 - ❖ No hay servidor central
- ❑ Usado por Gnutella
- ❑ Cada par indiza los archivos que tiene para compartir

Inundar consultas

- ❑ mensaje Query enviado sobre conexiones TCP
- ❑ Pares retransmiten consulta
- ❑ QueryHit enviado en camino inverso



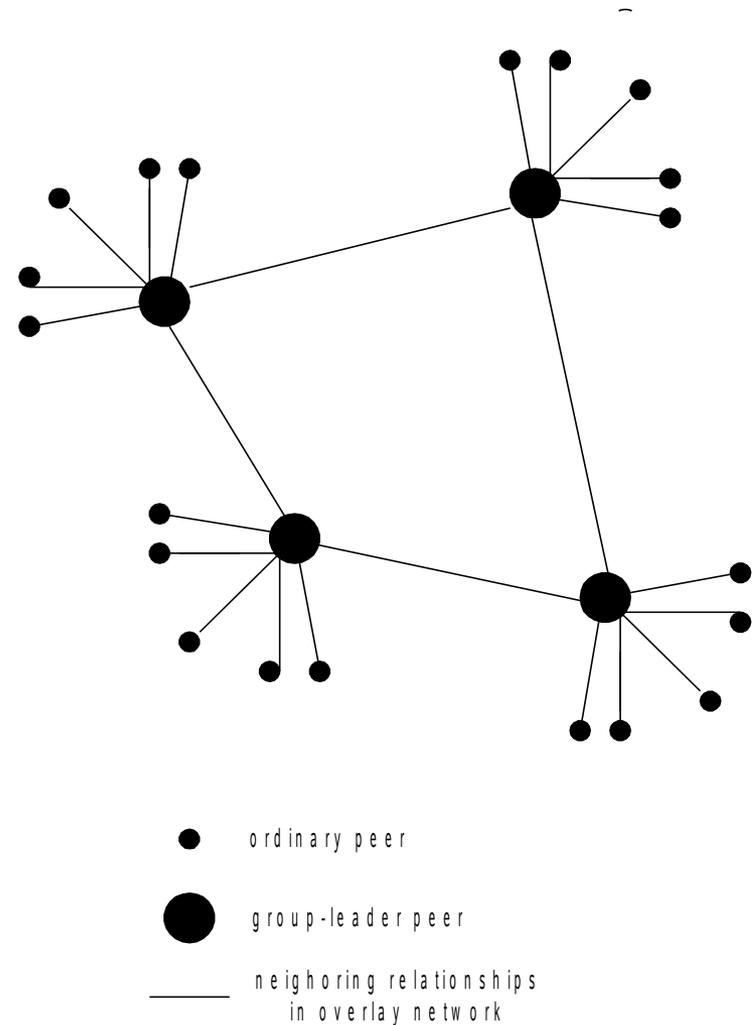
Escalabilidad:
Pobre, alcance limitado de inundación

Gnutella: Unir pares

1. Cuando Alicia quiere unirse a la red debe encontrar otro par Gnutella usando una lista de pares candidatos
2. Alicia intenta conexiones TCP con pares candidatos hasta que logra conectarse con Roberto
3. *Inundación (Flooding)*: Alicia envía Ping a Roberto; Roberto reenvía Ping a sus vecinos y así sucesivamente
 - Los pares que reciben el Ping responden con Pong
4. Alicia recibe muchos Pong, y puede establecer más conexiones TCP

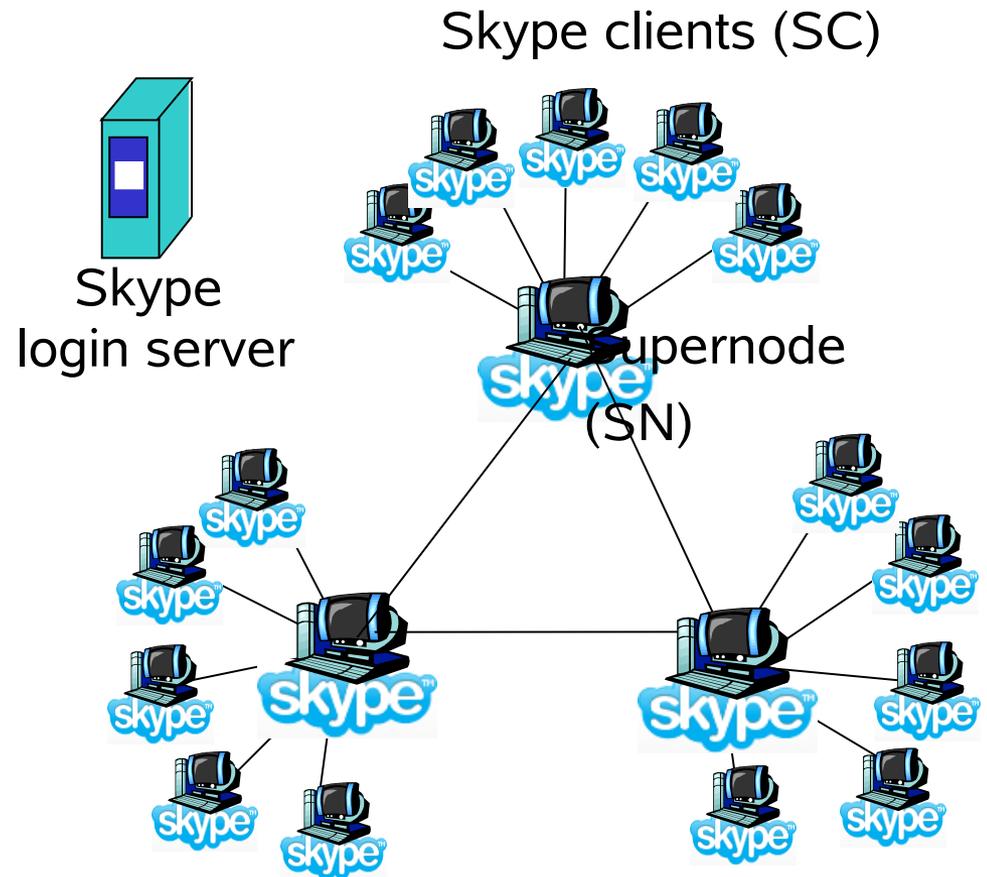
Red de Jerarquía

- Cada par es un supernodo o está asignado a un supernodo
 - ❖ Conexión TCP entre par y su supernodo
 - ❖ Conexiones TCP entre algunos supernodos
- Los supernodos tienen los contenidos de los nodos ordinarios



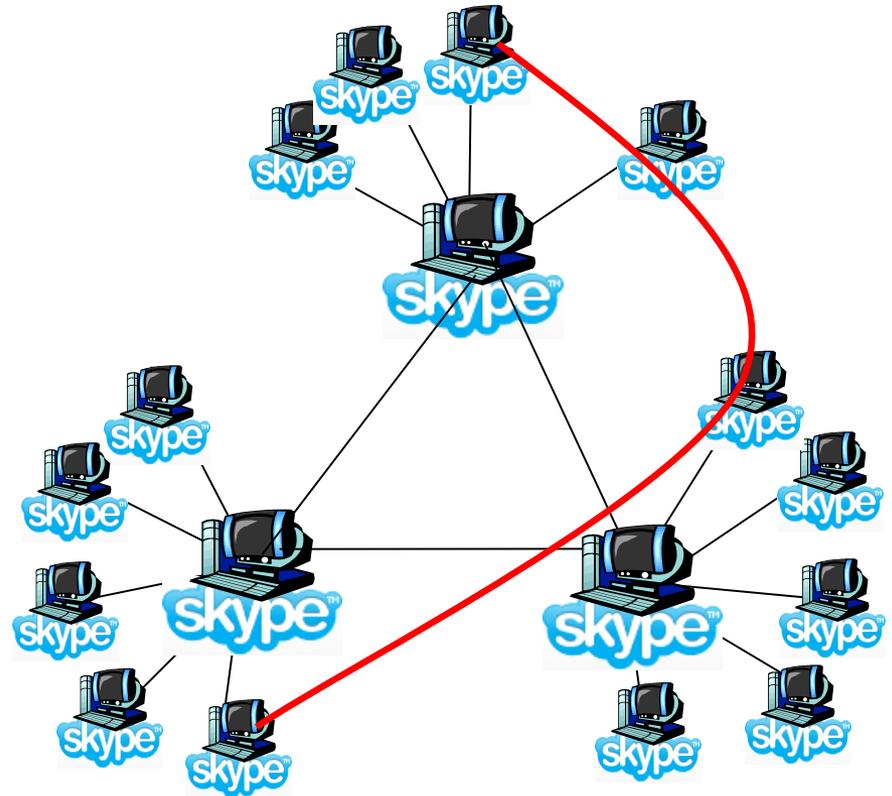
Caso de estudio P2P: Skype

- ❑ inherentemente P2P: se comunican pares de usuarios
- ❑ Protocolo propietario (inferido por ingeniería reversa)
- ❑ Jerarquía supernodos
- ❑ Índice mapea nombres de usuarios a direcciones IP, distribuido en los supernodos



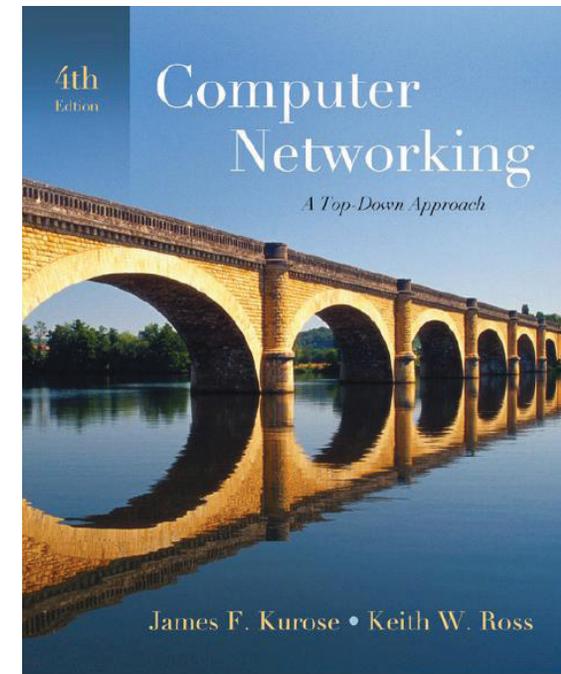
Pares como relay

- Problema cuando ambos están contra NAT
 - ❖ NAT evita que un par externo inicie una llamada a un par interno
- Solución:
 - ❖ Se utilizan los supernodos de ambos, utilizando relay
 - ❖ Cada par inicia una sesión con relay
 - ❖ Los pares se comunican a través de NAT utilizando relay



Introducción a las Redes de Computadoras

Capítulo 2 Capa de Aplicación



Nota acerca de las transparencias del curso:
Estas transparencias están basadas en el sitio web que
acompaña el libro, y
han sido modificadas por los docentes del curso.
All material copyright 1996-2007
J.F Kurose and K.W. Ross, All Rights Reserved

Capa de aplicación

2.1 Principio de aplicaciones de red

2.2 Web y HTTP

2.3 FTP

2.4 Correo electrónico
SMTP, POP3, IMAP

2.5 DNS

2.6 Aplicaciones P2P

2.7 Programación con sockets usando TCP

2.8 Programación con sockets usando UDP

2.9 Un servidor Web simple

Programación con Sockets

Objetivo: aprender a construir una aplicación cliente/servidor que se comunican a través de sockets

Socket API

- Presentada en BSD4.1 UNIX, 1981
- Explícitamente creados, utilizados y liberados por las aplicaciones
- Paradigma cliente/servidor
- Dos tipos de transporte vía sockets:
 - Datagramas (no confiable)
 - Flujo o stream de bytes (confiable)

socket

interfaz *local al host*,
creada por la aplicación
y *controlada por el S.O.*
(una "puerta") a través de
la cual los procesos
pueden *enviar y recibir*
mensajes desde y hacia
otros procesos

Programación con Sockets

- El socket es la interfaz que la aplicación tiene con las capas inferiores
- Encapsula todo el manejo de la comunicación entre un punto y otro y da la sensación a la aplicación de estar dialogando directamente con la aplicación del otro lado
- Los sockets están asociados a una **dirección** y un **puerto** en el host y conocen la dirección y puerto donde está el socket destino

Programación con Sockets

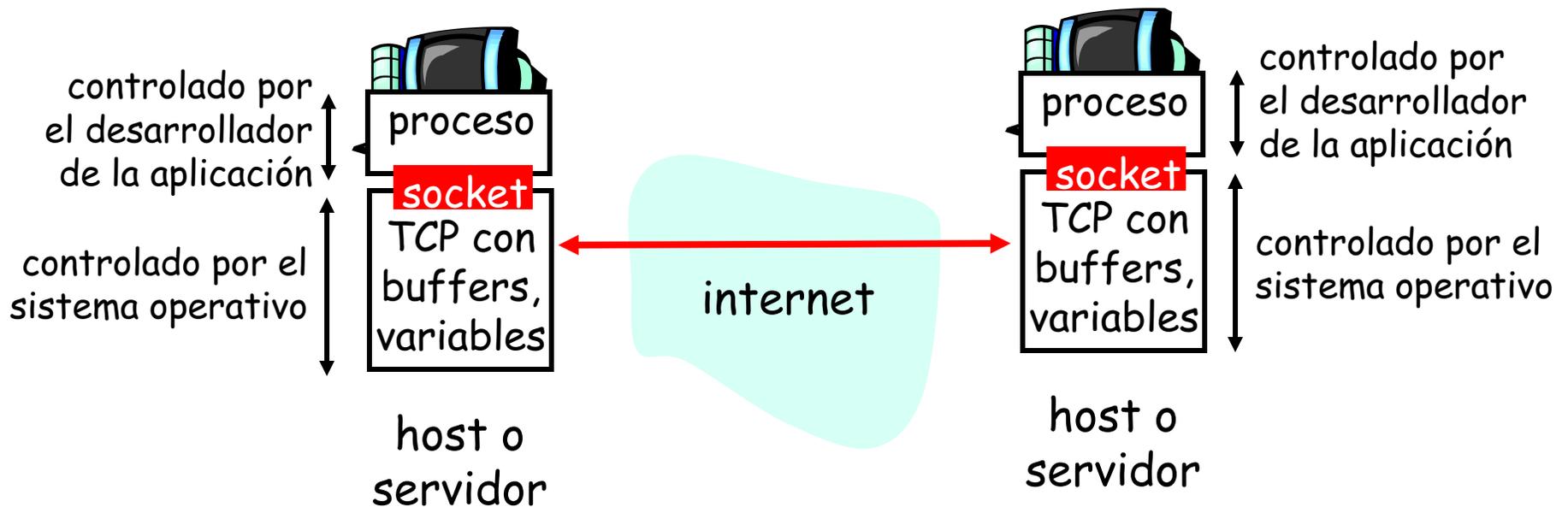
Las primitivas presentadas para la API de sockets son

SOCKET	Crea un nuevo punto de comunicación
BIND	Engancha el socket con una dirección y puerto locales
LISTEN	Anuncia que se aceptarán conexiones, e indica el largo de la cola
ACCEPT	Bloquea al llamador hasta que llega un intento de conexión
CONNECT	Activamente intenta establecer una conexión
SEND	Envía datos sobre la conexión
RECEIVE	Recibe datos de la conexión
CLOSE	Cierra la conexión

Programación con Sockets TCP

Socket: puerta entre los procesos de aplicación y la capa de transporte (UDP or TCP)

Servicio TCP: transferencia confiable de **bytes** de un proceso a otro



Programación con Sockets TCP

Lo que hace el servidor

- el proceso servidor debe estar ejecutándose
- el servidor debe haber creado un socket que recibe el contacto del cliente

Lo que hace el cliente

- crea un socket TCP local al cliente
- especifica la dirección IP y el puerto del servidor
- cuando el cliente crea el socket, **establece la conexión TCP** con el servidor

- Cuando es contactado por el cliente, **el servidor TCP crea un nuevo socket** a través del cual el proceso servidor se comunicará con el cliente
 - permite que un servidor dialogue con muchos clientes
 - cada cliente es atendido en un número de puerto distinto
(capítulo 3)

Programación con Sockets TCP

Desde el punto de
vista de la aplicación...

*TCP provee una transferencia
confiable y en orden de bytes
entre el cliente y el servidor*

Sockets TCP en Java

En Java se manejan primitivas de sockets ligeramente diferentes a las vistas anteriormente

- En una conexión TCP, java maneja dos clases de sockets diferentes: `Socket` y `ServerSocket`.
- Se puede pensar como que un `Socket` es una conexión ya establecida, donde se pueden enviar y recibir bytes.
- El `ServerSocket`, sin embargo, es un `Socket` que está "a la espera" de que un cliente venga a conectarse.

Sockets TCP en Java

En Java se manejan primitivas de sockets ligeramente diferentes a las vistas anteriormente

SOCKET	<code>Socket, ServerSocket</code>
BIND	<code>ServerSocket.bind()</code>
LISTEN	<i>No hay una primitiva correspondiente</i>
ACCEPT	<code>ServerSocket.accept()</code>
CONNECT	<code>new Socket(adress, port) o Socket.bind()</code>
SEND	<code>Socket.getOutputStream().write()</code>
RECEIVE	<code>Socket.getInputStream().read()</code>
CLOSE	<code>Socket.close()</code>

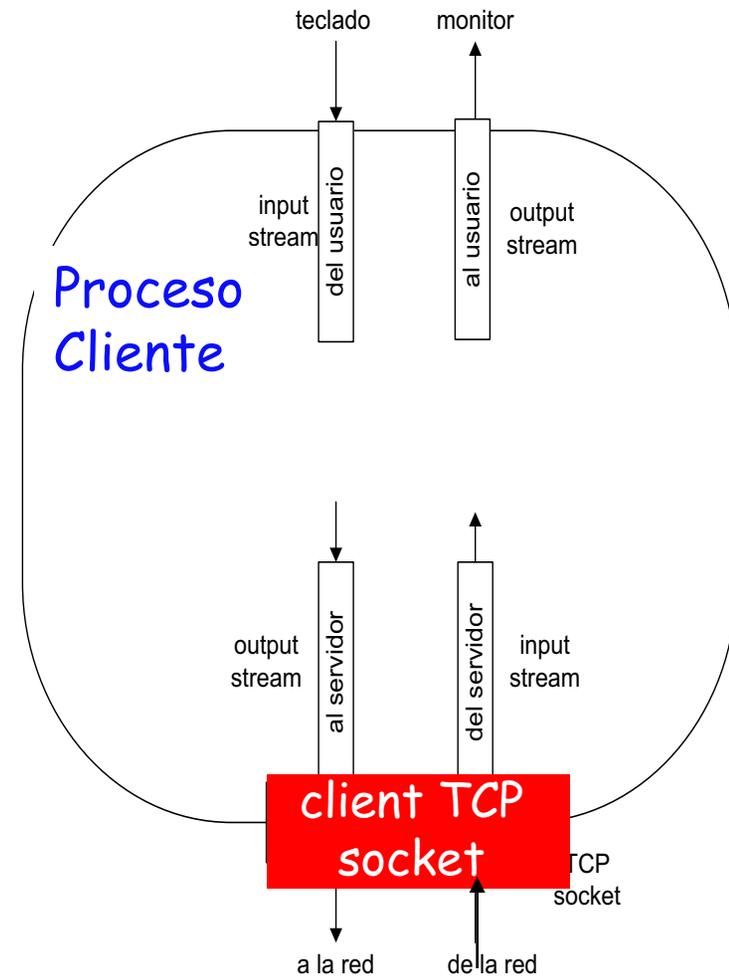
Sockets TCP en Java

- Un **stream** es una secuencia de caracteres que fluye hacia o desde un proceso
- Un **stream de entrada** se engancha a alguna fuente de entrada de datos del proceso. Por ejemplo el teclado o un socket
- Un **stream de salida** se engancha a una fuente de salida. Por ejemplo el monitor o un socket
- En java un socket provee dos streams: uno de salida y uno de entrada
- Estos streams son los que se utilizan para **enviar** y **recibir** datos respectivamente
- También tenemos un stream para tomar datos de la entrada estándar y otro para imprimir datos en pantalla

Programación con Sockets TCP

Aplicación cliente-servidor:

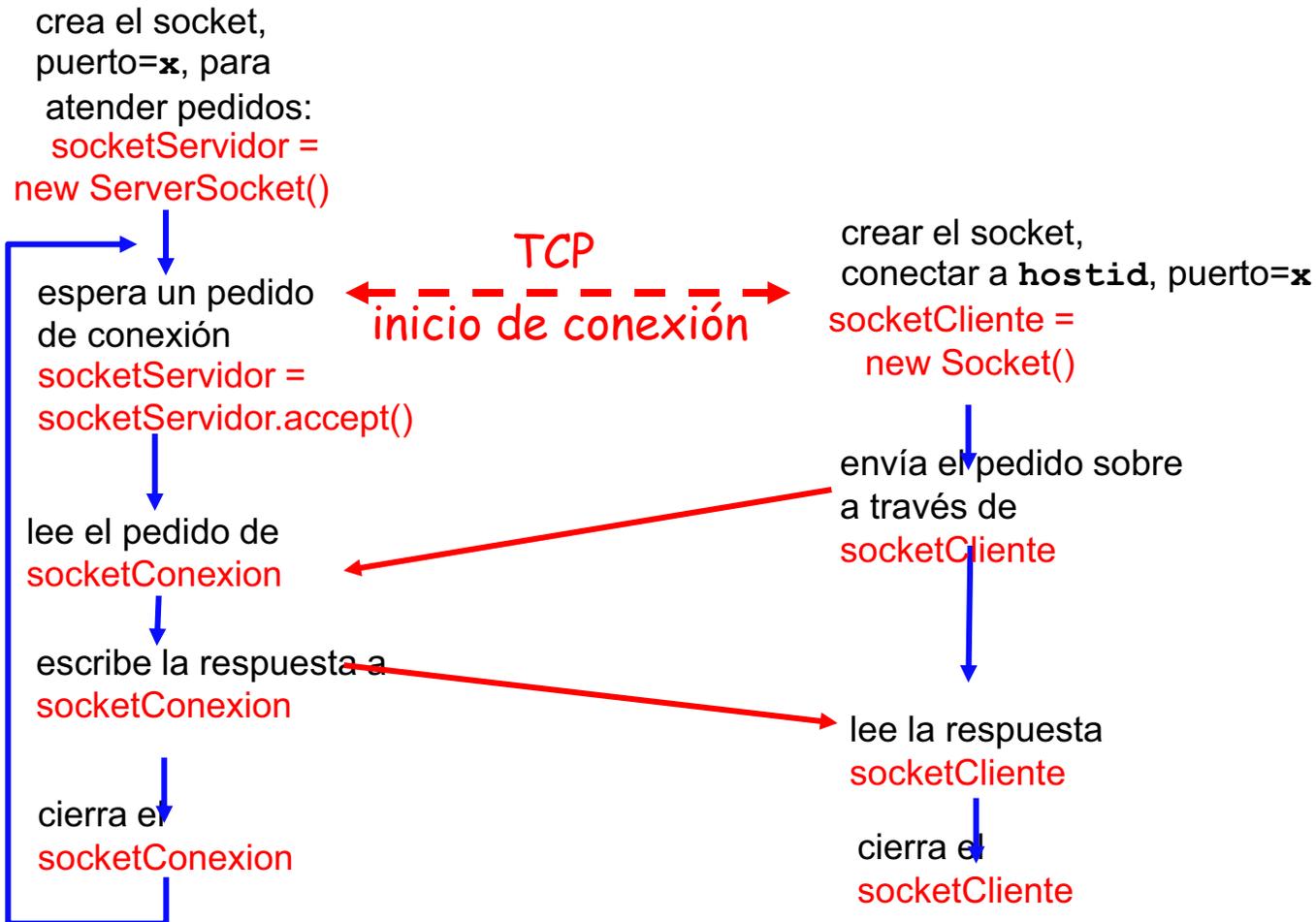
- 1) El cliente lee de la entrada estándar (stream delUsuario), y envía al servidor por un socket (stream alServidor)
- 2) El servidor lee del socket
- 3) El servidor pasa el texto a mayúsculas y lo envía al cliente
- 4) El cliente lee e imprime la línea modificada obtenida a través del socket (stream delServidor)



Interacción cliente/servidor: TCP

Servidor (se ejecuta en `hostid`)

Cliente



Ejemplo: Cliente TCP en Java

```
public class ClienteTCP {  
  
    public static void main(String[] args) throws Exception {  
        String host = "localhost";  
        int puerto = 6789;  
        if (args.length == 2) {  
            host = args[0];  
            puerto = new Integer(args[1]);  
        } else if (args.length == 1) {  
            puerto = new Integer(args[0]);  
        }  
  
        BufferedReader entradaDelUsuario =  
            new BufferedReader(  
                new InputStreamReader(System.in));  
  
        Socket socketCliente = new Socket(host, puerto);  
  
        DataOutputStream salidaAlServidor =  
            new DataOutputStream(  
                socketCliente.getOutputStream());  
    }  
}
```

configurar host
y puerto del
servidor

crear stream
de entrada

crear socketCliente
y conectarse al
servidor

crear stream de
salida enganchado
al socket

Ejemplo: Cliente TCP en Java

crear stream de
entrada enganchado
al socket

```
BufferedReader entradaDelServidor =  
    new BufferedReader(  
        new InputStreamReader(  
            socketCliente.getInputStream());
```

enviar datos
al servidor

```
String frase = entradaDelUsuario.readLine();  
salidaAlServidor.writeBytes(frase + "\n");
```

leer datos
desde el servidor

```
String fraseModificada =  
    entradaDelServidor.readLine();  
System.out.println(fraseModificada);
```

```
socketCliente.close();
```

```
}
```

```
}
```

Ejemplo: Servidor TCP en Java

```
public class ServidorTCP {  
  
    public static void main(String[] args) throws Exception {  
        int puerto = 6789;  
        if (args.length > 0) puerto = new Integer(args[0]);  
  
        ServerSocket socketServidor = new ServerSocket(puerto);  
  
        while (true) {  
            Socket socketConexion = socketServidor.accept();  
  
            BufferedReader entradaDelCliente =  
                new BufferedReader(  
                    new InputStreamReader(  
                        socketConexion.getInputStream()));  
        }  
    }  
}
```

configurar puerto

crear socket de bienvenida en

esperar alguna conexión de parte del cliente

crear stream de entrada, enganchado al socket

Ejemplo: Servidor TCP en Java

crear stream de salida, enganchado al socket

```
DataOutputStream salidaAlCliente =  
    new OutputStreamWriter(socketConexion.getOutputStream());
```

leer datos enviados por el cliente

```
String linea = entradaDelCliente.readLine();  
String mayusculas = linea.toUpperCase();
```

escribir la respuesta en el socket

```
salidaAlCliente.writeBytes(mayusculas + "\n");  
socketConexion.close();
```

```
}  
  
}
```

termina el loop de procesamiento, así que volverá a esperar la siguiente conexión

Capa de aplicación

2.1 Principio de aplicaciones de red

2.2 Web y HTTP

2.3 FTP

2.4 Correo electrónico
SMTP, POP3, IMAP

2.5 DNS

2.6 Aplicaciones P2P

2.7 Programación con sockets usando TCP

2.8 Programación con sockets usando UDP

2.9 Un servidor Web simple

Programación con Sockets UDP

UDP: no hay "conexión" entre el cliente y el servidor

- no hay *handshaking*
- el emisor indicará la IP y el puerto en cada paquete que envíe
- El receptor sabrá a qué IP y puerto contestar pues lo obtendrá del paquete

UDP: los datos pueden llegar a destino en desorden, o no llegar

Programación con Sockets UDP

Desde el punto de
vista de la aplicación...

*UDP provee una transferencia
no confiable de grupos de bytes
(datagramas) entre el cliente y
el servidor*

Sockets UDP en Java

- Mientras que las primitivas originales se usaban tanto para TCP como UDP, Java maneja una clase específica para los sockets UDP denominada `DatagramSocket`
- Cliente y servidor utilizan la misma clase. No existe un `ServerDatagramSocket` pues no existe la noción de "bloquearse esperando la conexión", que sí existe en TCP
- No hay streams de entrada y salida, porque la comunicación se basa en paquetes (datagramas)

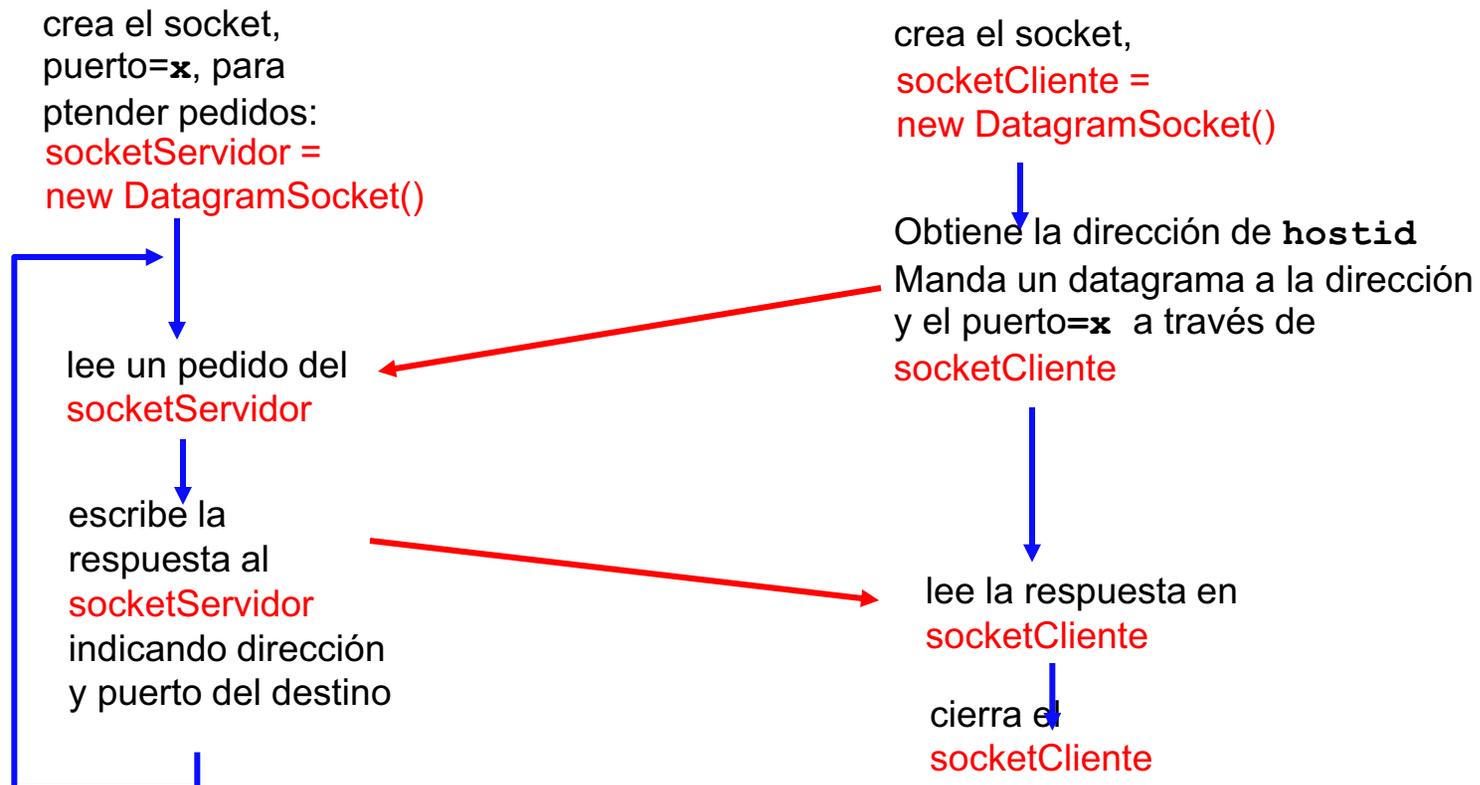
Sockets UDP en Java

- Los paquetes que viajan se representan mediante la clase `DatagramPacket`
- A cada paquete que se envíe se le debe indicar la dirección y puerto a donde está destinado
- Cuando el servidor recibe uno de estos paquetes, la dirección y puerto del cliente estarán en el paquete, por lo que sabrá a dónde responder

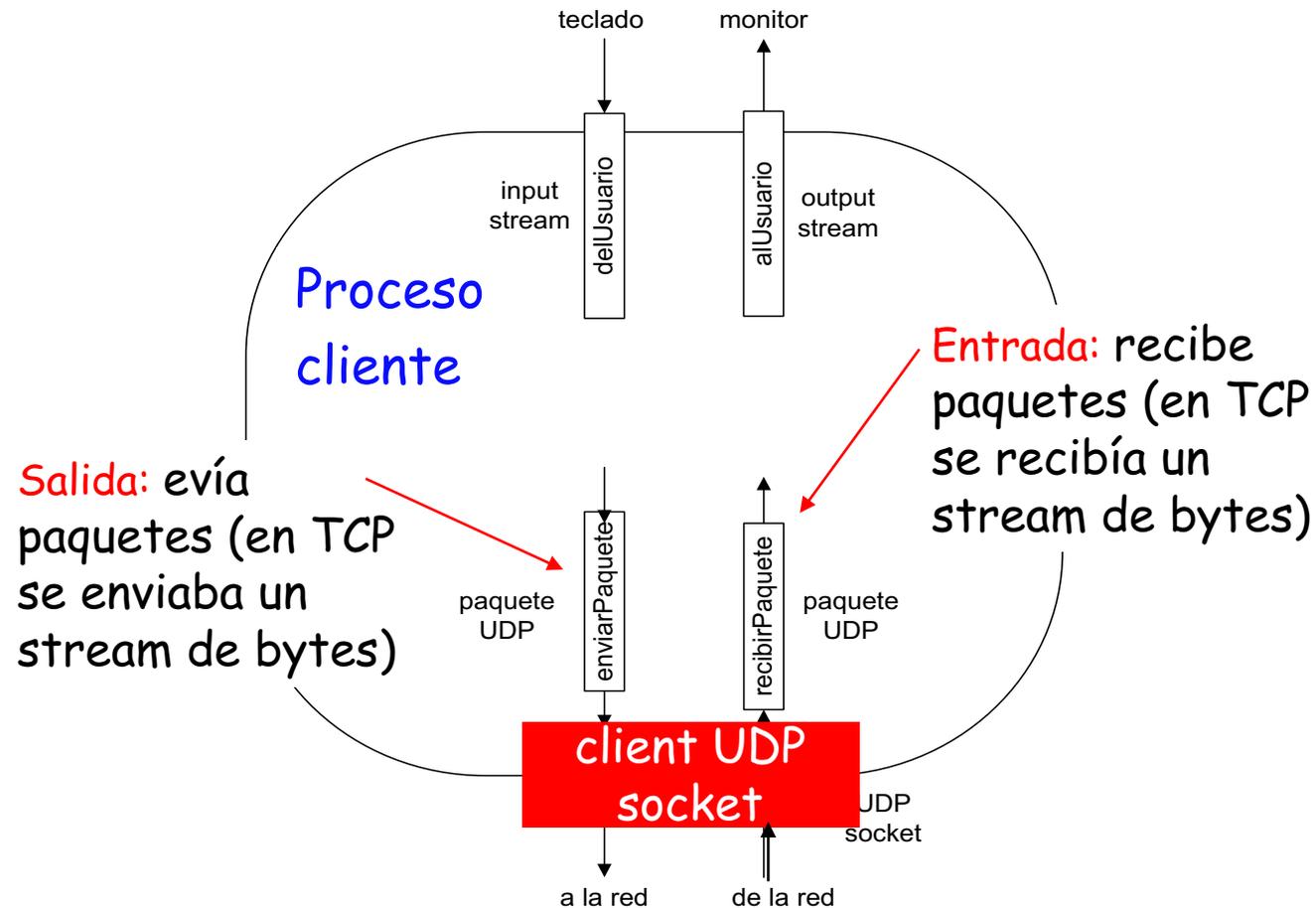
Interacción cliente/servidor: UDP

Servidor (ejecutándose en `hostid`)

Cliente



Ejemplo: Cliente UDP en Java



Ejemplo: Cliente UDP en Java

```
public class ClienteUDP {  
  
    public static void main(String[] args) throws Exception {  
        String host = "localhost";  
        int puerto = 9876;  
        if (args.length == 2) {  
            host = args[0];  
            puerto = new Integer(args[1]);  
        } else if (args.length == 1) {  
            puerto = new Integer(args[0]);  
        }  
  
        BufferedReader entradaDelUsuario =  
            new BufferedReader(  
                new InputStreamReader(System.in));  
  
        DatagramSocket socketCliente = new DatagramSocket();  
  
        InetAddress direccionIP = InetAddress.getByName(host);  
  
        String frase = entradaDelUsuario.readLine();  
    }  
}
```

configurar
host y puerto

crear stream
de entrada
del teclado

crear socket
del cliente

traducir el nombre
de host a su IP
(puede usar el DNS)

Ejemplo: Cliente UDP en Java

crear datagrama
con datos, largo,
IP y puerto

```
DatagramPacket paqueteEnvio =  
    new DatagramPacket(  
        frase.getBytes(), frase.length(),  
        direccionIP, puerto);
```

mandar datagrama
al servidor

```
socketCliente.send(paqueteEnvio);
```

leer datagrama
enviado por
el servidor

```
DatagramPacket paqueteRecepcion =  
    new DatagramPacket(new byte[1024], 1024);  
socketCliente.receive(paqueteRecepcion);  
int largo = paqueteRecepcion.getLength();
```

armar frase
a desplegar

```
String fraseModificada =  
    new String(paqueteRecepcion.getData(), 0, largo);  
  
System.out.println(fraseModificada);  
  
socketCliente.close();
```

```
}
```

```
}
```

Ejemplo: Servidor UDP en Java

```
public class ServidorUDP {  
  
    public static void main(String[] args) throws Exception {  
        int puerto = 9876;  
        if (args.length > 0) {  
            puerto = new Integer(args[0]);  
        }  
  
        DatagramSocket socketServidor =  
            new DatagramSocket(puerto);  
  
        while (true) {  
            DatagramPacket paqueteRecepcion =  
                new DatagramPacket(new byte[1024], 1024);  
  
            socketServidor.receive(paqueteRecepcion);  
        }  
    }  
}
```

configurar
el puerto

crear socket
del servidor

crear espacio para
el datagrama a recibir

recibir
datagrama

Ejemplo: Servidor UDP en Java

obtener la IP y
puerto del emisor

```
String frase =  
    new String(paqueteRecepcion.getData(), 0,  
              paqueteRecepcion.getLength());  
  
InetAddress direccionDestino = paqueteRecepcion.getAddress();  
int puertoDestino = paqueteRecepcion.getPort();
```

crear datagrama
con la respuesta

```
byte[] data = frase.toUpperCase().getBytes();  
  
DatagramPacket paqueteEnvio =  
    new DatagramPacket(  
        data, frase.length(),  
        direccionDestino, puertoDestino);
```

escribir
datagrama
en el socket

```
socketServidor.send(paqueteEnvio);
```

termina el loop de procesamiento,
así que volverá a esperar la llegada
de un datagrama nuevo

Capa de aplicación

2.1 Principio de aplicaciones de red

2.2 Web y HTTP

2.3 FTP

2.4 Correo electrónico
SMTP, POP3, IMAP

2.5 DNS

2.6 Aplicaciones P2P

2.7 Programación con sockets usando TCP

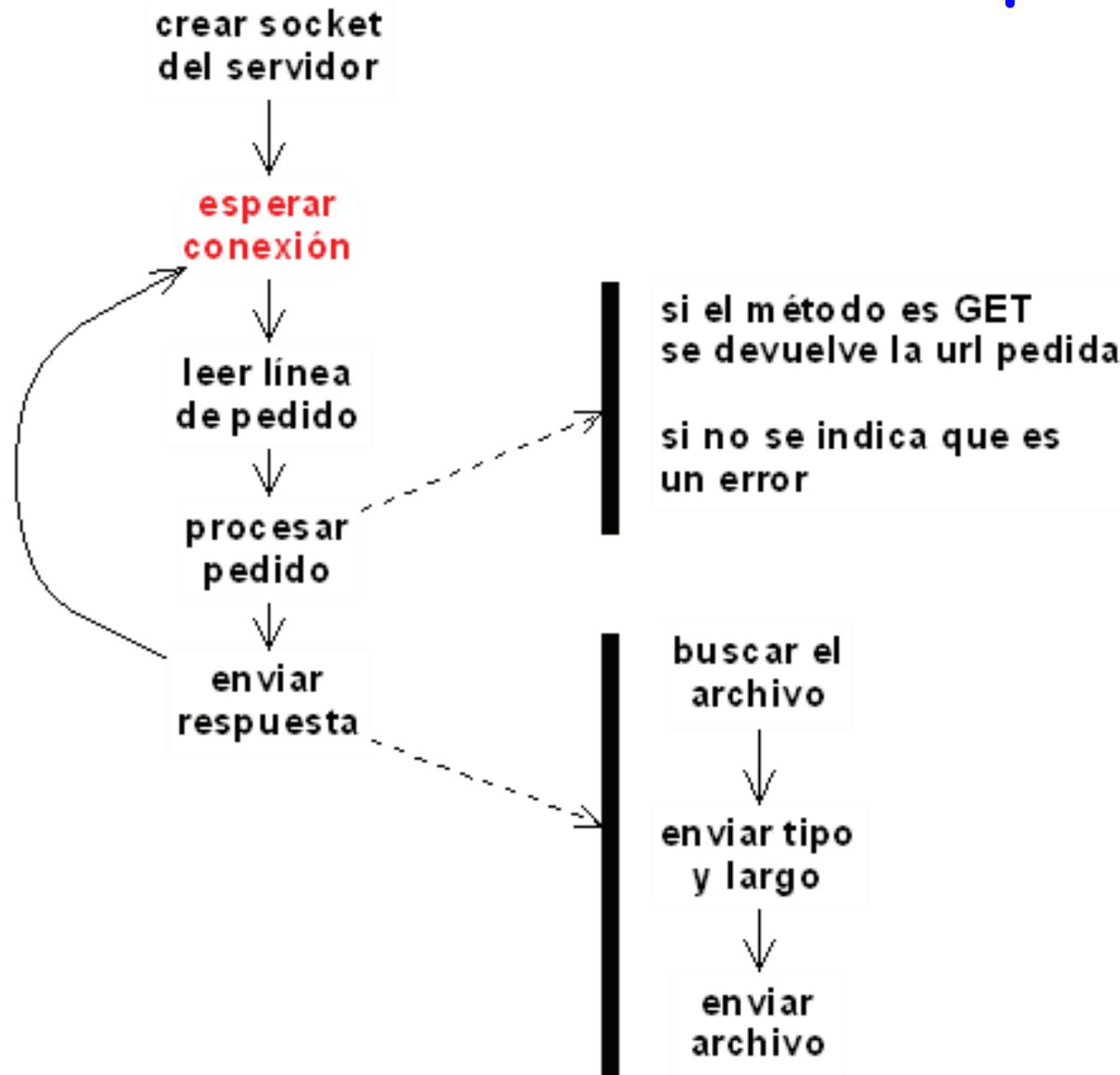
2.8 Programación con sockets usando UDP

2.9 Un servidor Web simple

Un servidor Web simple

- maneja conexiones HTTP de a una (un solo thread)
- acepta la conexión
- procesa el cabezal
- obtiene el archivo pedido del disco
- crea el mensaje de respuesta
 - cabezal y archivo
- envía respuesta al cliente
- si el protocolo HTTP se respeta, este servidor podrá ser invocado desde un browser (IE, Firefox)

Un servidor Web simple



Un servidor Web simple

¿Qué pasa si queremos que se puedan utilizar forms?

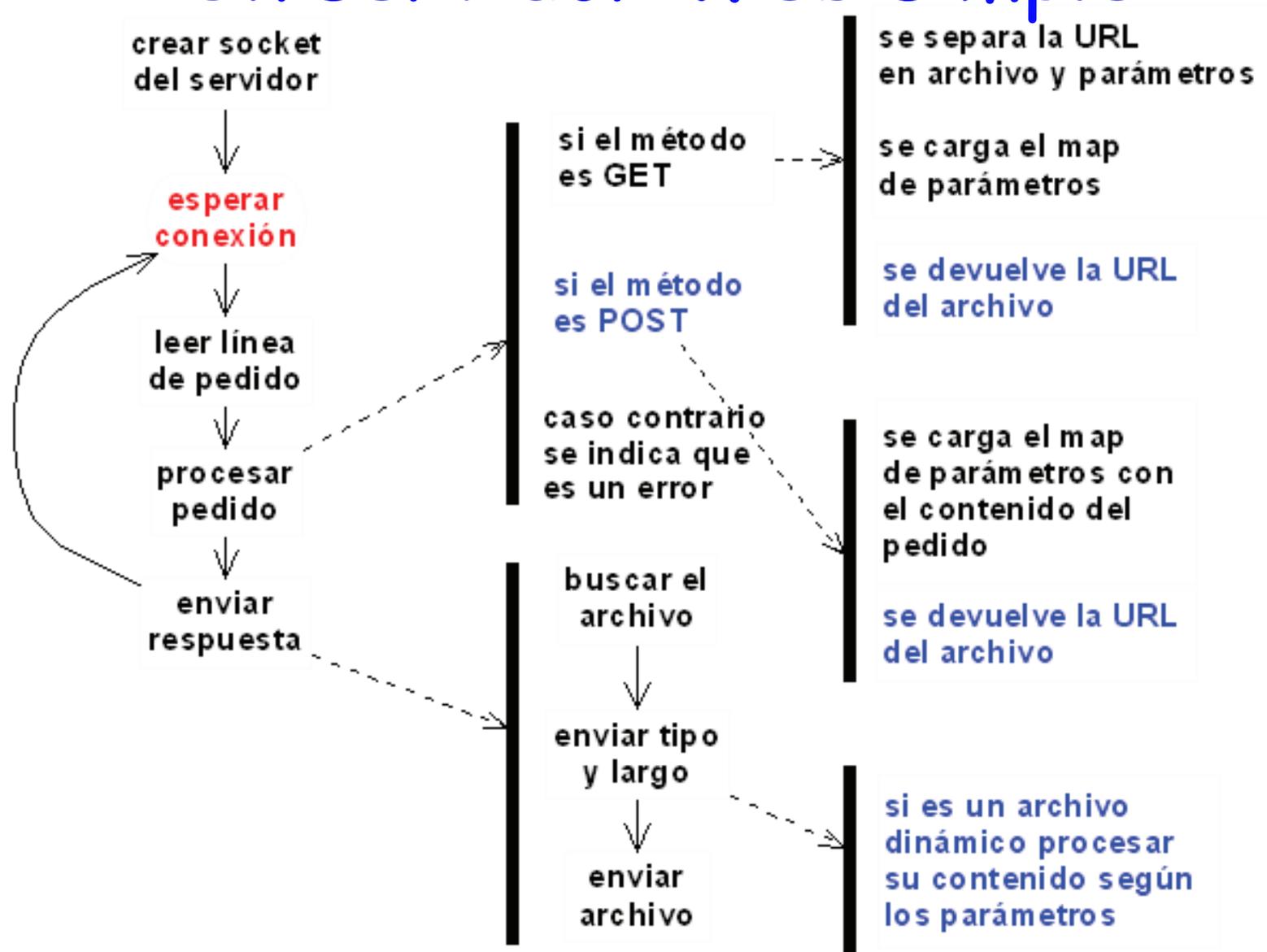
- Los datos de un form se envían al servidor al hacer submit
- Método GET y método POST de HTTP
- Lo ideal es poder utilizar estos datos del lado del servidor, generando algún tipo de respuesta dinámica

Un servidor Web simple

¿Qué pasa si queremos que se puedan utilizar forms?

- Si los parámetros del form van en el método `GET`, se codifican junto con la URL (son visibles)
 - Ejemplo: `GET /formulario.html?tarjeta=4040213`
- En cambio si se utiliza el método `POST`, los parámetros viajarán en el payload del mensaje HTTP (no son visibles, al menos desde la barra el browser)

Un servidor Web simple



Programación con sockets: referencias

Sockets en C

- "Unix Network Programming" (J. Kurose),
<http://manic.cs.umass.edu/~amldemo/courseware/intro>
- Tanenbaum - Computer Networks Fourth Edition

Sockets en Java

- "All About Sockets" (Sun tutorial),
<http://www.javaworld.com/javaworld/jw-12-1996/jw-12-sockets.html>
- "Socket Programming in Java: a tutorial,"
<http://www.javaworld.com/javaworld/jw-12-1996/jw-12-sockets.html>
- API de Sockets en Java 1.4.2
<http://java.sun.com/j2se/1.4.2/docs/api/index.html>