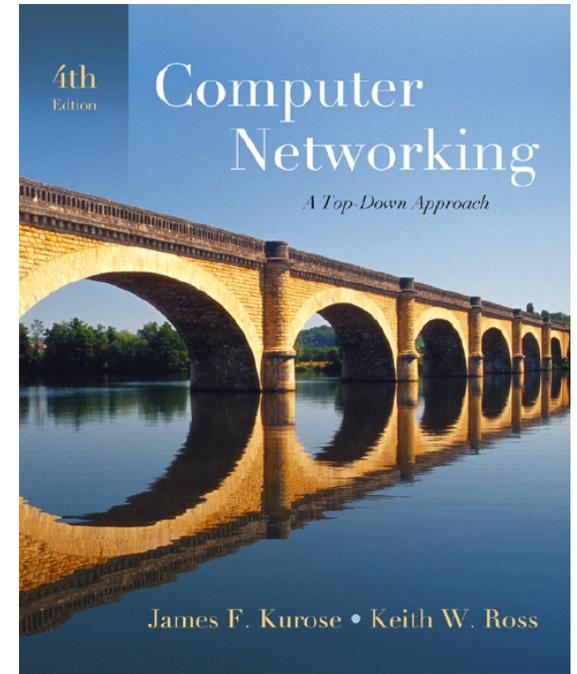


# Introducción a las Redes de Computadoras

## 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

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

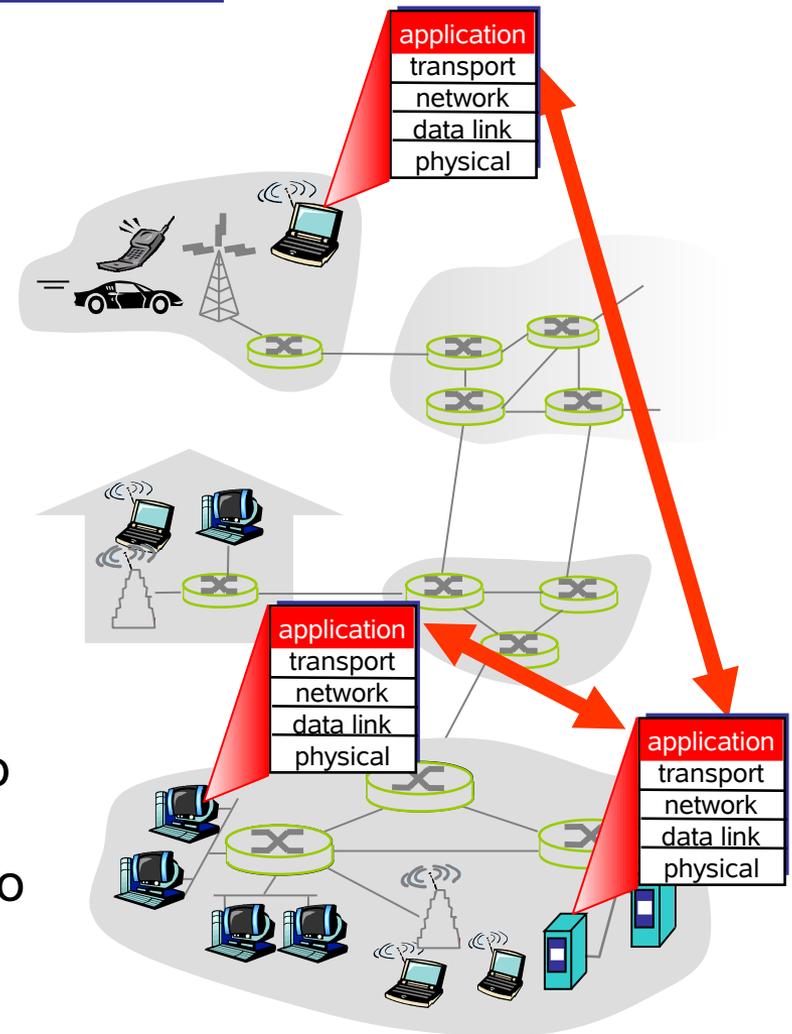
- Objetivos
  - Aspectos conceptuales y de implementación de protocolos de aplicación
    - Modelos de capa de transporte
    - Paradigma cliente servidor
    - Paradigma P2P (peer to peer)
  - Comprender protocolos de aplicación populares
    - HTTP
    - FTP
    - SMTP / POP3 / IMAP
    - DNS
  - Programar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

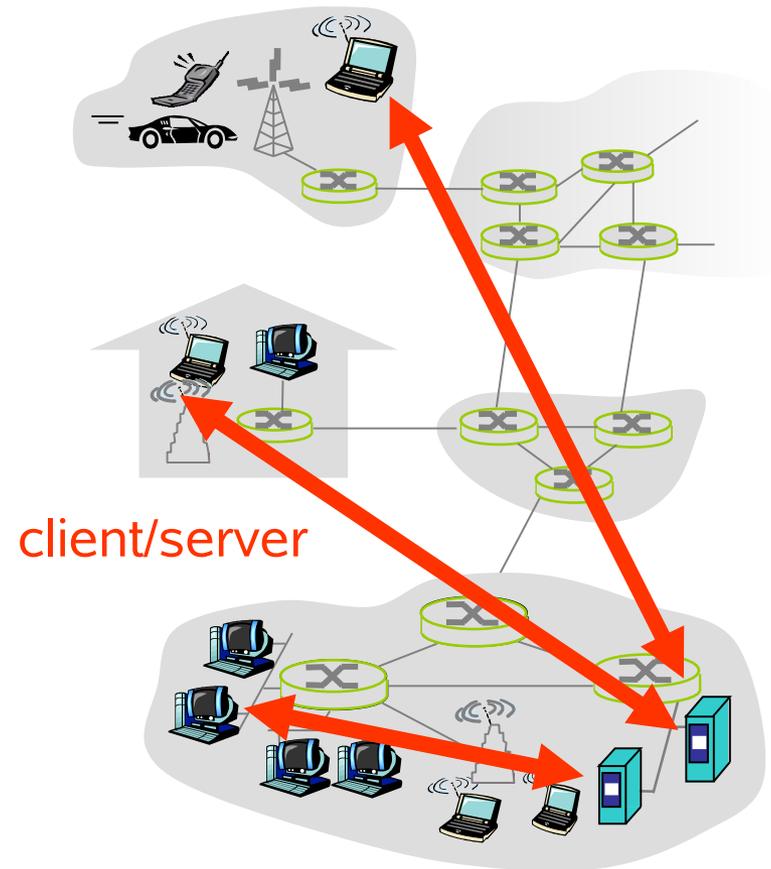


# 1. Principios de aplicaciones de red

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

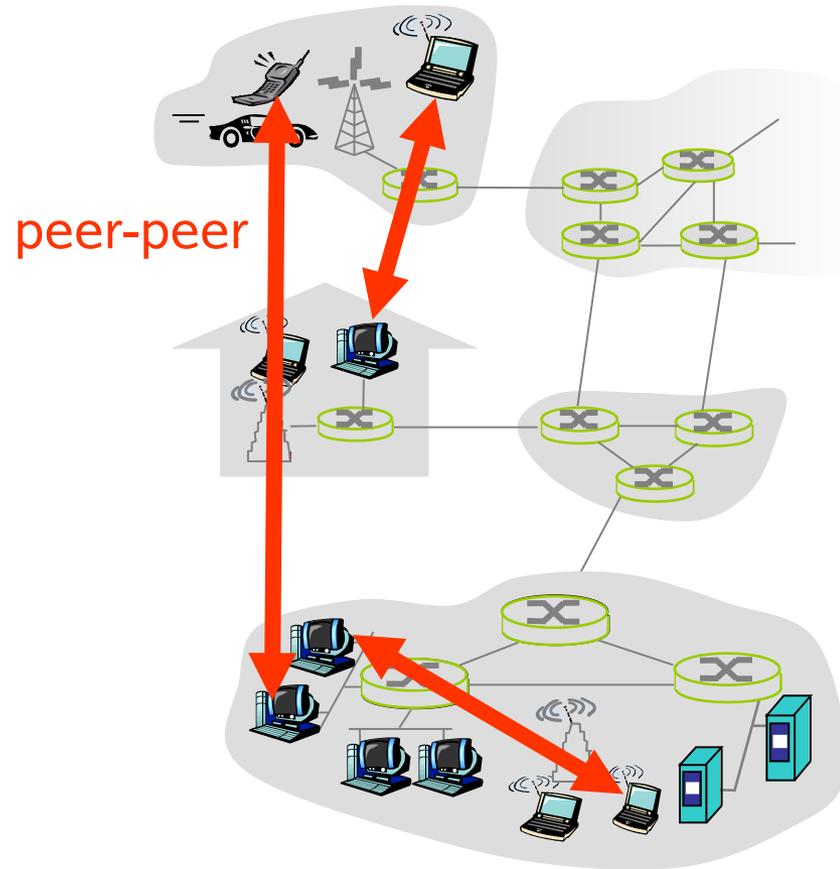
# Arquitectura cliente servidor

- 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

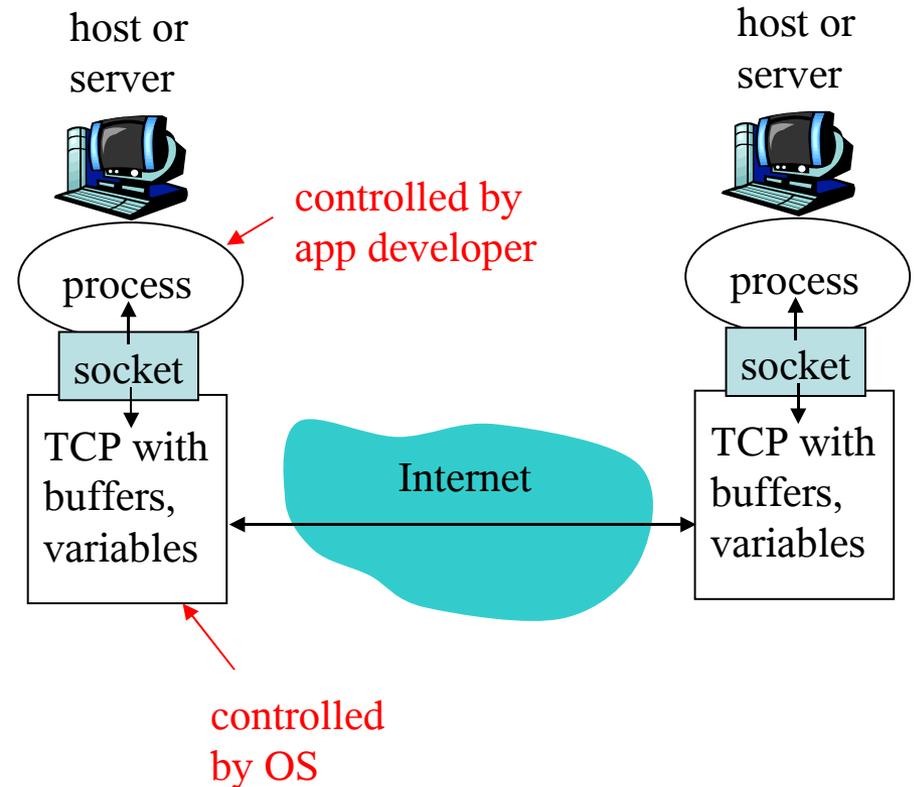
- Skype
  - Aplicación VoIP (Voz sobre IP) P2P
  - Servidor centralizado, encuentra las direcciones de los “peer” remotos
  - Conexión cliente-cliente directa (no interviene el servidor)
- Mensajería instantánea
  - Conversaciones entre usuarios es P2P
  - Servicio centralizado: presencia de clientes, 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](http://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

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

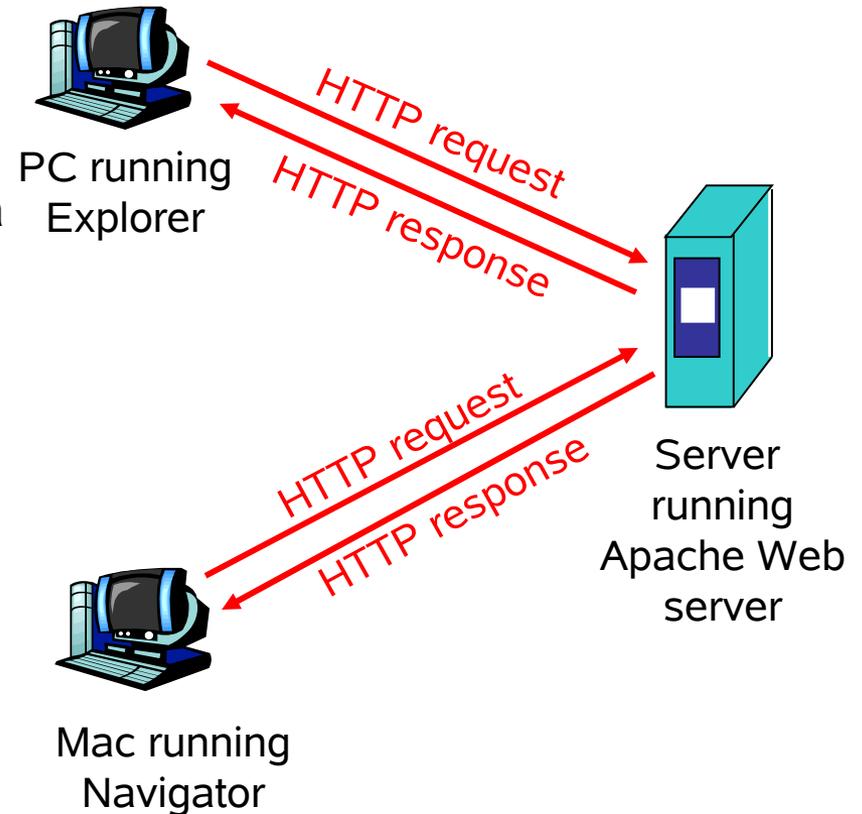
<b><i>Aplicación</i></b>	<b><i>Protocolo de aplicación</i></b>	<b><i>Protocolo de transporte</i></b>
<b>E-mail</b>	SMTP (RFC 2821)	TCP
<b>Terminal remota</b>	Telnet (RFC 854)	TCP
<b>Web</b>	HTTP (RFC 2616)	TCP
<b>Transferencia de archivos</b>	FTP (RFC 959)	TCP
<b>Multimedia</b>	HTTP (Youtube) RTP (RFC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



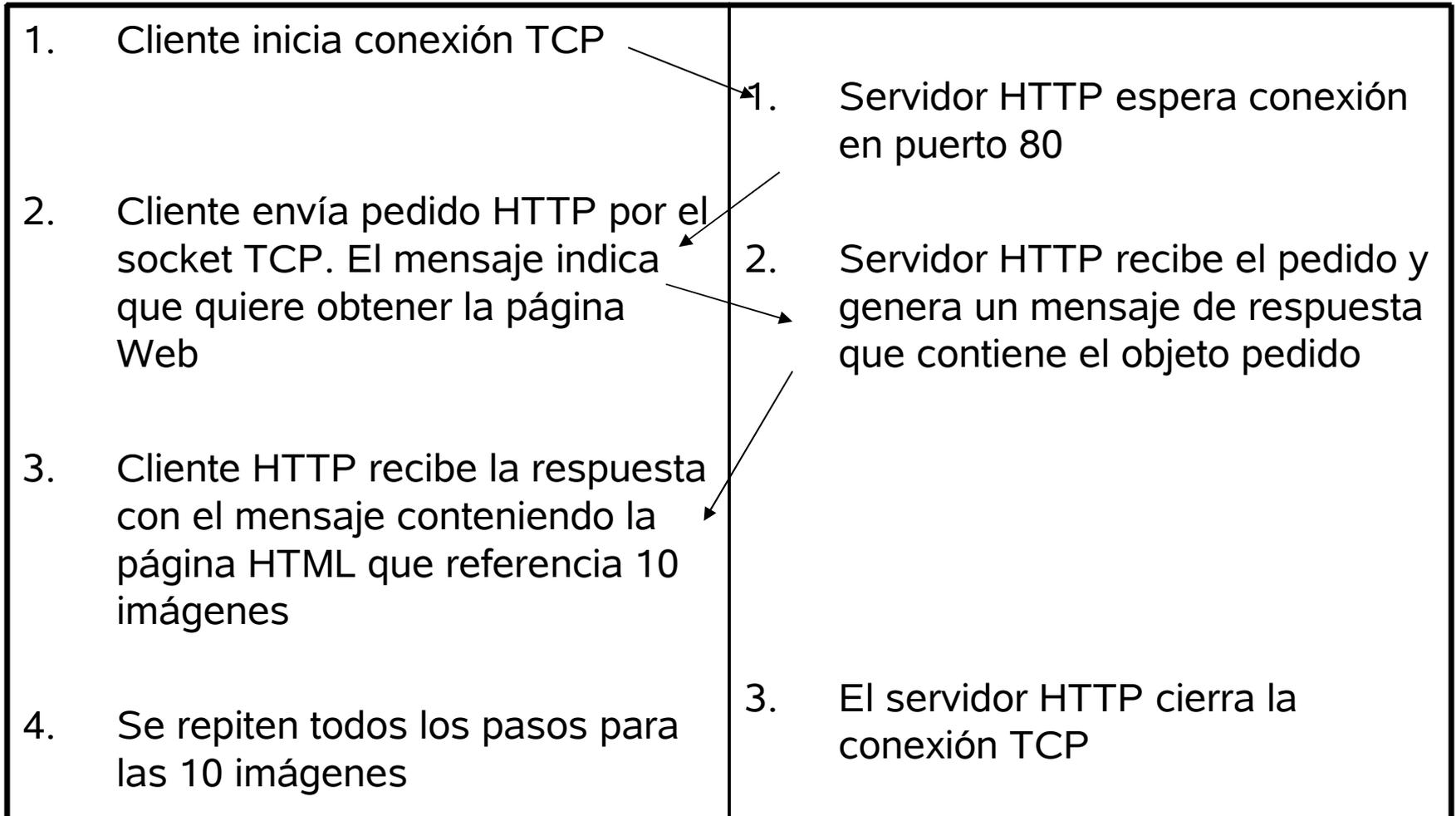
# HTTP

- 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

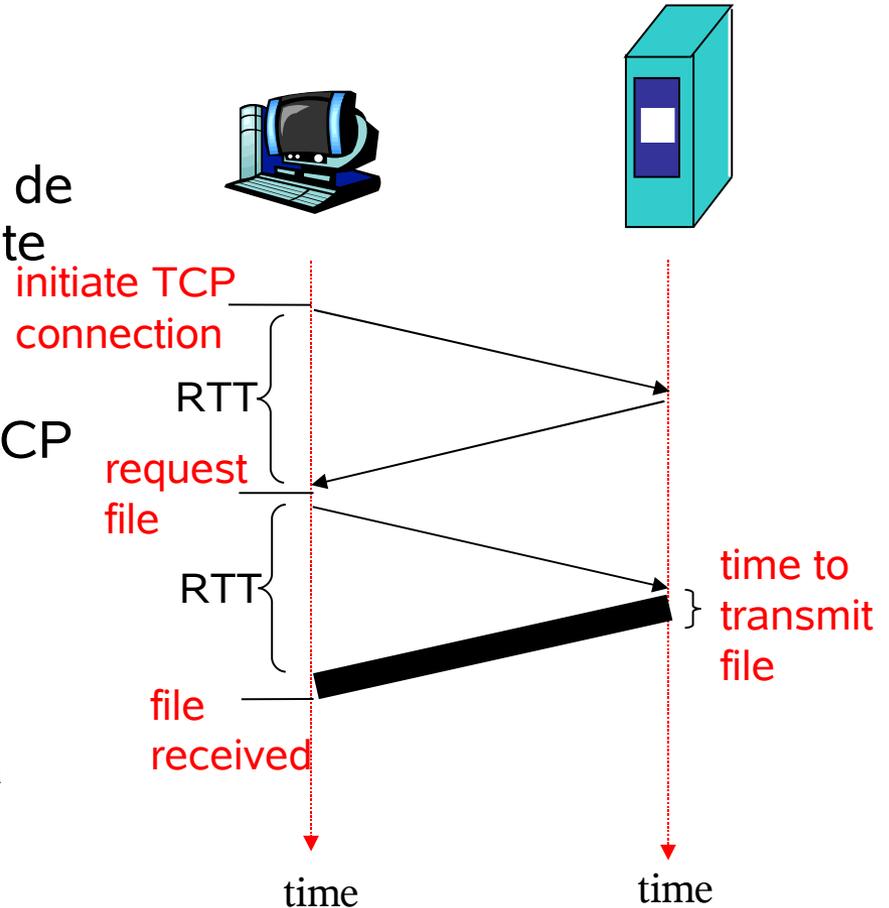
- No persistente
  - Cada objeto es enviado en una conexión TCP diferente
- Persistente
  - Se envían múltiples objetos en cada conexión TCP

# 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://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



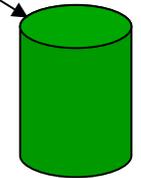
Amazon crea ID

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

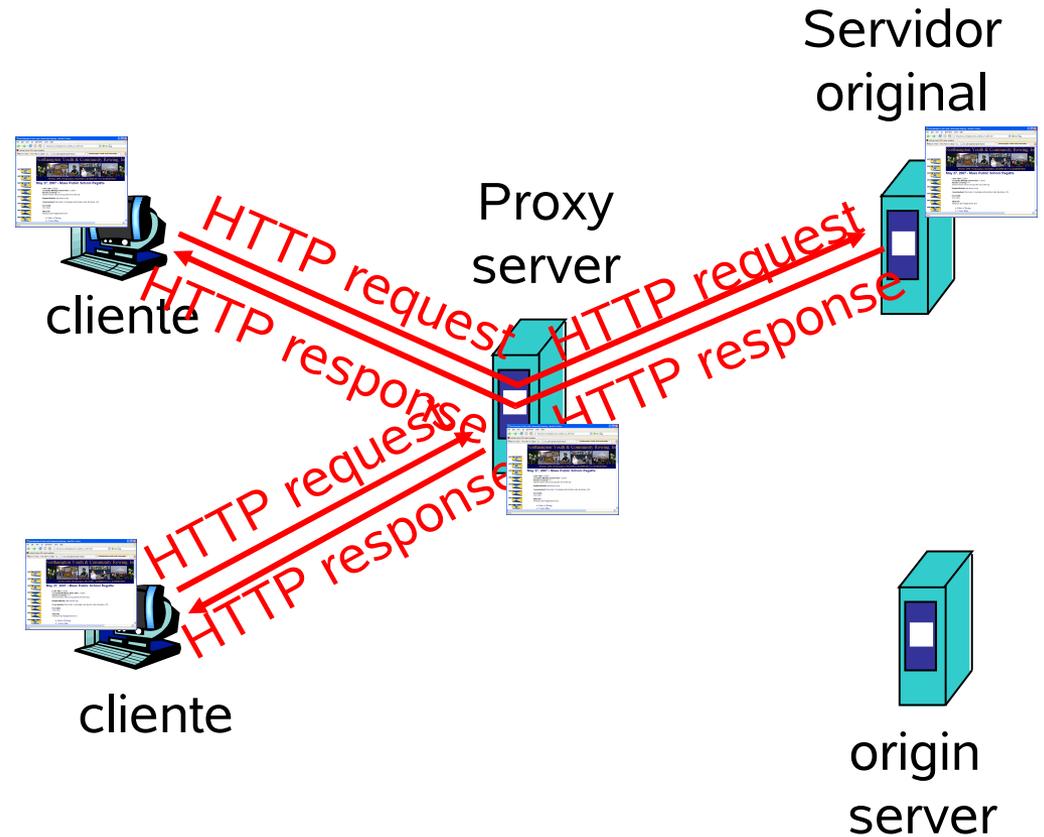
## Como se mantiene el estado

- 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)

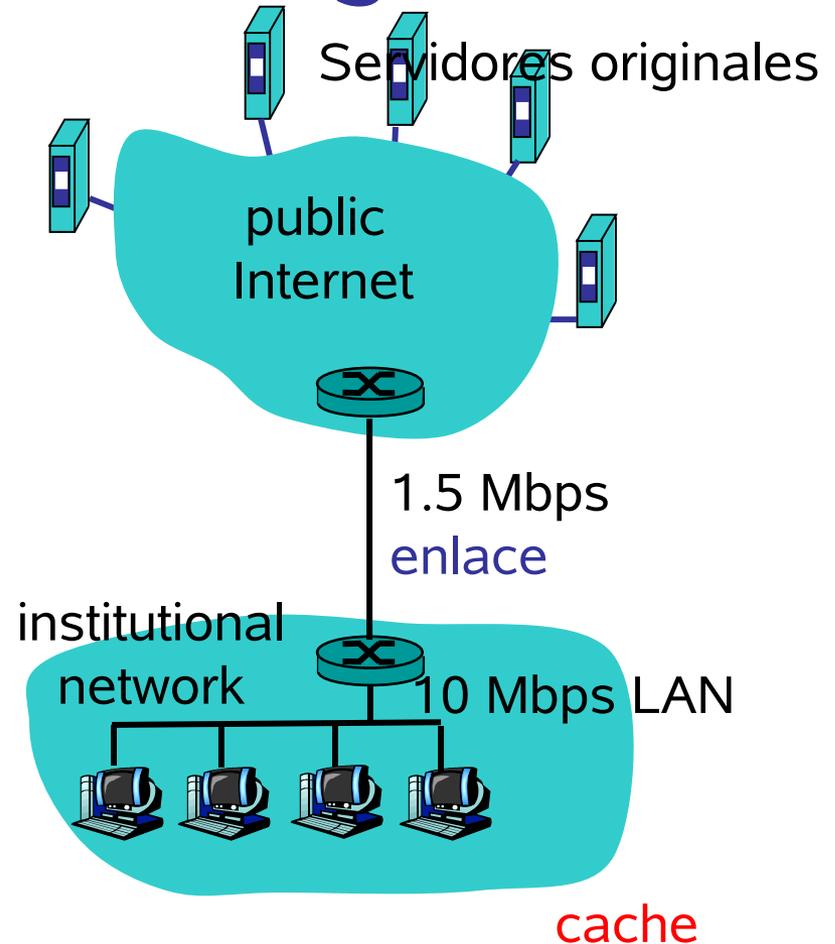
# Ejemplo Caching

## 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



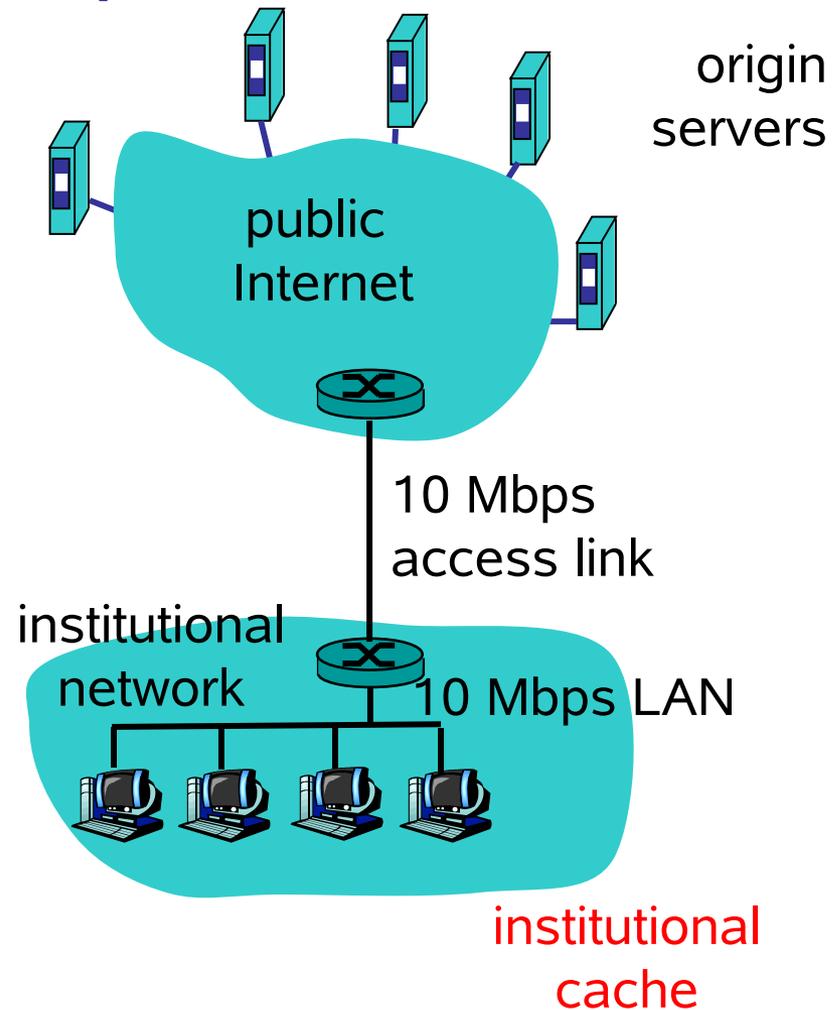
# Caching example (cont)

## 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



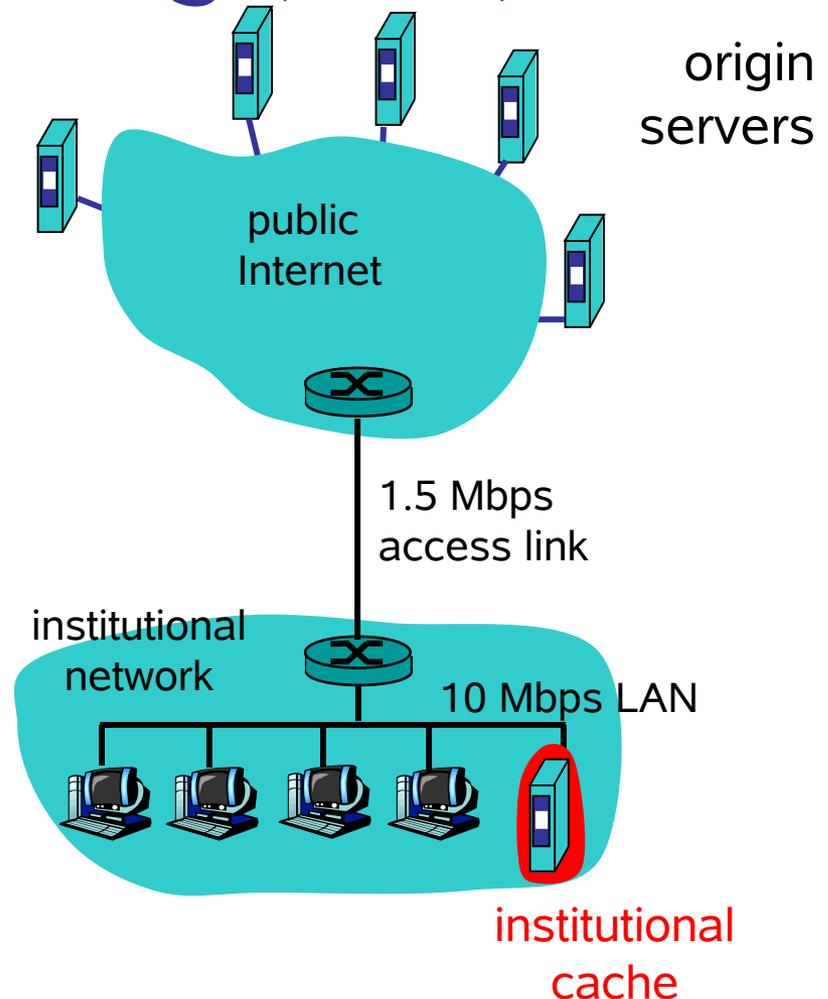
# Ejemplo Caching (cont)

## 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**

