Capítulo 2: Capa Aplicación

Web y HTTP

ELO322: Redes de Computadores

Agustín J. González

Este material está basado en:

- Material de apoyo al texto *Computer Networking: A Top Down Approach Featuring the Internet.* Jim Kurose, Keit.

Capítulo 2: Capa Aplicación

- 2.1 Principios de las aplicaciones de red
- 2.2 Web y HTTP
- 2.3 Correo Electrónico
  - SMTP, POP3, IMAP
- 2.4 DNS
- 2.5 P2P para archivos compartidos
- 2.6 Video streaming y redes de distribución de contenidos
- 2.6 Programación de sockets con UDP y TCP

Web y HTTP

- Una página Web está compuesta de objetos
- En este contexto objetos pueden ser archivos HTML, imágenes (JPEG, GIF,...), archivos de audio, archivos de vídeo,...
- Páginas Web consisten generalmente de un archivo HTML base el cual incluye referencias a objetos.
- Cada objeto es direccionable por un Universal Resource Locator (URL)
- Ejemplo URL:

```
http://www.elo.utfsm.cl/imgmenu/header.jpg
http://www.elo.utfsm.cl:80/imgmenu/header.jpg
```

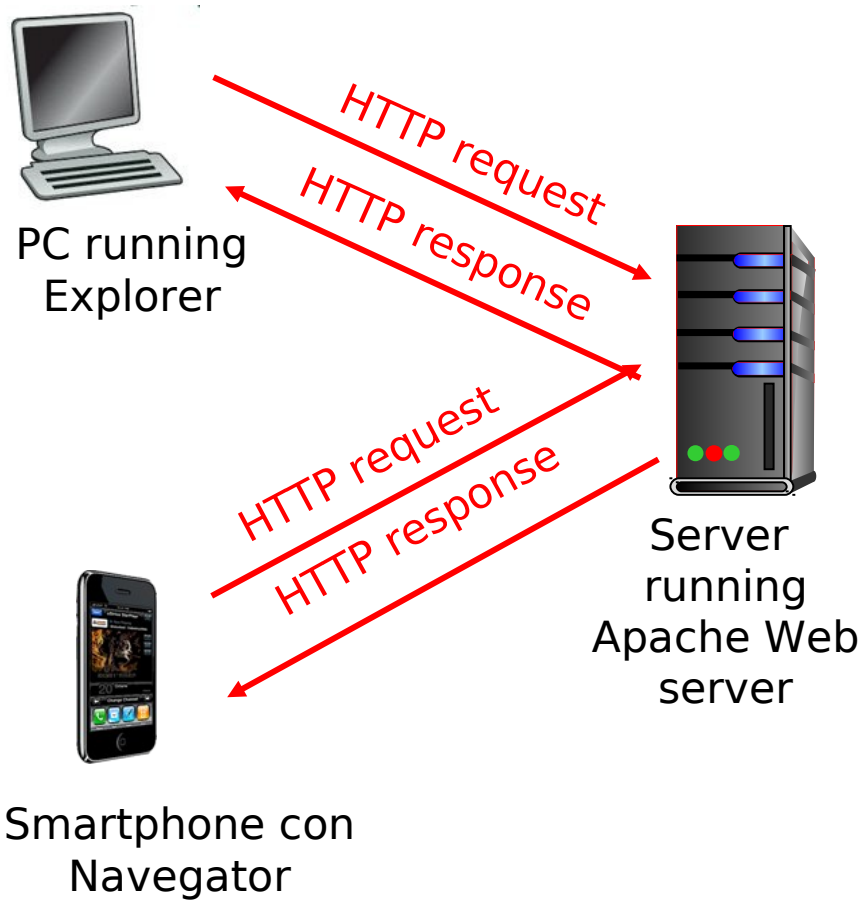
Nombre de la máquina y puerto

Nombre de camino (path name)

HTTP Generalidades

HTTP: hypertext transfer protocol

- Es un protocolo de la capa aplicación usado por la Web
- Usa modelo cliente/servidor
  - *cliente:* browser primero requiere, luego recibe (usando HTTP) y "despliega" objetos Web
  - *servidor:* Servidor Web envía (usando HTTP) objetos en respuesta a requerimientos



HTTP generalidades (cont.)

Usa TCP:

1) Cliente inicia conexión TCP (crea socket) al servidor, puerto 80 (puede ser otro!)
2) Servidor acepta conexión TCP del cliente
3) Mensajes HTTP (mensajes del protocolo de capa aplicación) son intercambiados entre browser (cliente HTTP) y servidor Web (servidor HTTP)
4) Se cierra la conexión TCP

HTTP no guarda "estado"

- El servidor no mantiene información sobre los requerimientos del clientes

Protocolos que mantienen "estado" son complejos!, ¿Por qué?

- Historia pasada (estado) debe ser mantenida
- Si servidor o cliente se cae, las vistas del estado pueden ser inconsistentes, y deben ser sincronizadas

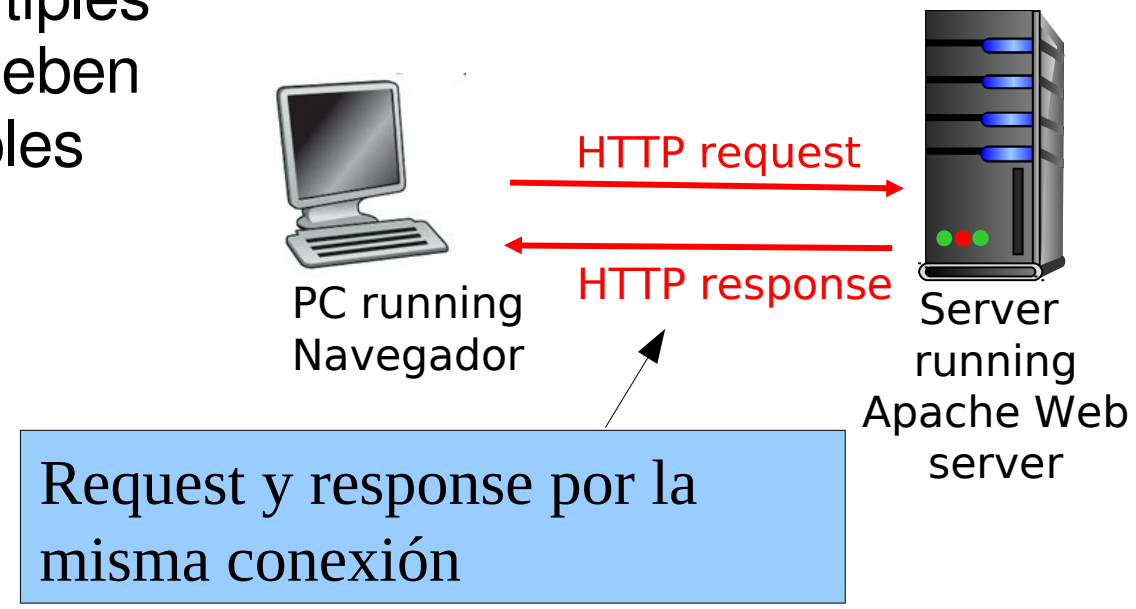
Conexiones HTTP

HTTP No-persistente

- A lo más un objeto es enviado por una conexión TCP. Luego conexión es cerrada.
- Para bajar múltiples objetos se deben hacer múltiples conexiones.

HTTP Persistente

- Múltiples objetos pueden ser enviados por una única conexión TCP entre el cliente y servidor.



HTTP no-persistente

Supongamos que el usuario ingresa URL
`www.someSchool.edu/someDepartment/home/index`

(contiene texto, y referencias a 10 imágenes jpeg)

1a. Cliente HTTP inicia una conexión TCP al servidor HTTP (proceso) en www.someSchool.edu en puerto 80

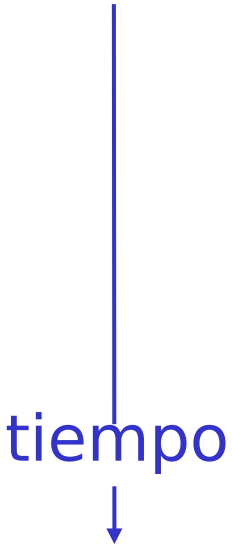
1b. Servidor HTTP en host www.someSchool.edu esperando por conexiones TCP en puerto 80 "acepta" conexión, notifica al cliente

2. Cliente HTTP envía *mensaje de requerimiento* (conteniendo el URL) por el socket de la conexión TCP. El mensaje indica que el cliente quiere el objeto someDepartment/home/index

3. El servidor HTTP recibe el mensaje de requerimiento, forma el *mensaje de respuesta* que contiene el objeto requerido y envía el mensaje por su socket.

tiempo

HTTP no-persistente (cont.)



4. Servidor HTTP cierra la conexión.

5. Cliente HTTP recibe el mensaje respuesta que contiene el archivo html y despliega el html. Analizando el archivo html file, encuentra 10 referencias a objetos jpeg

6. Pasos 1-5 son repetidos para cada uno de los 10 objetos jpeg.

No-persistente: tiempo de Respuesta

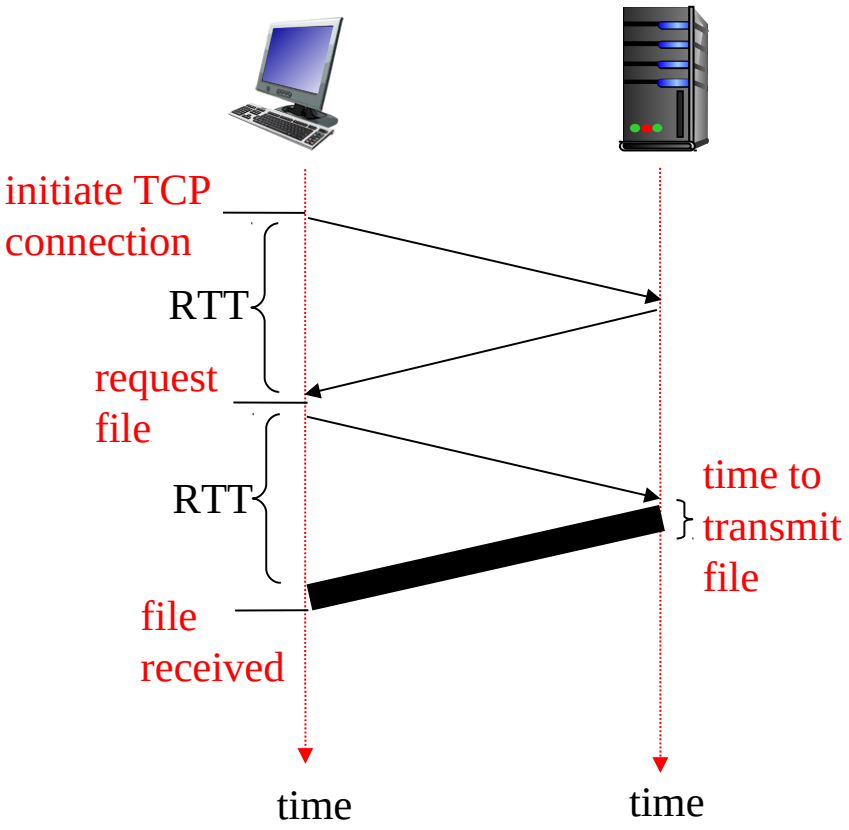
Definición de RTT(round-trip time):

tiempo de ida y vuelta de un **paquete pequeño** desde que sale de cliente, llega al servidor, y hasta que regresa.

Tiempo de respuesta:

- Un RTT para iniciar la conexión
- Un RTT por requerimiento HTTP y primeros bytes de la respuesta
- Tiempo de transmisión del archivo

Total HTTP no-persistente = 2RTT + tiempo de transmisión



HTTP Persistente

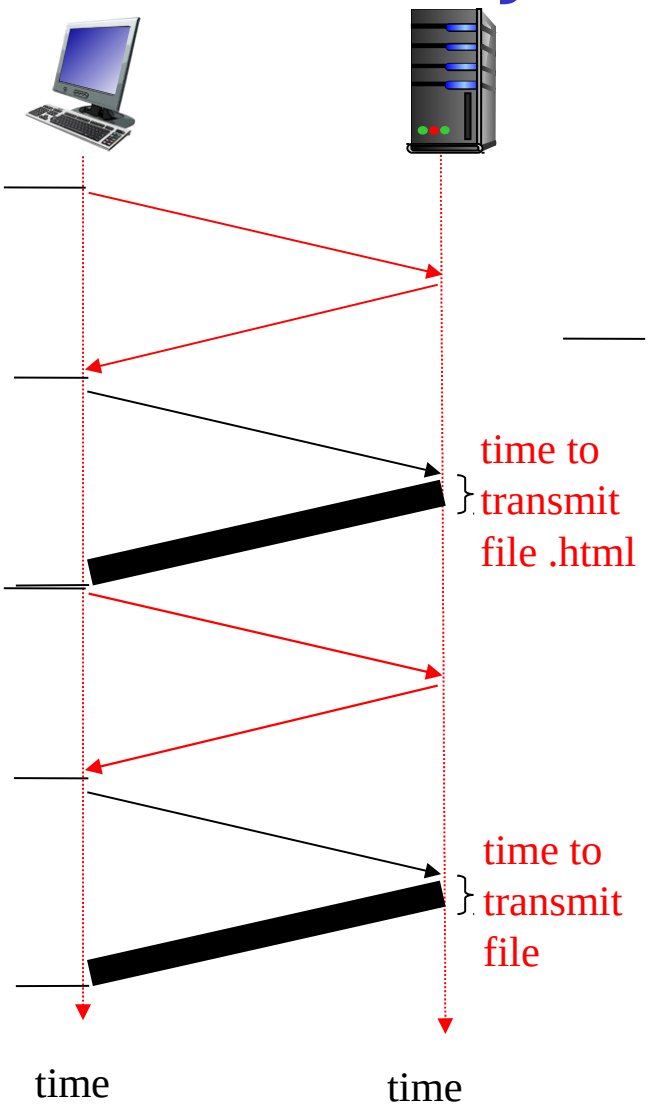
Análisis de HTTP no-persistente:

- requiere al menos 2 RTTs por objeto
- el navegador abre conexiones paralelas generalmente para traer objetos referenciados.
- OS debe trabajar y dedicar recursos para cada conexión TCP

HTTP Persistente

- servidor deja las conexiones abiertas después de enviar la respuesta
- mensajes HTTP siguientes entre los mismos cliente/servidor son enviados por la conexión
- Clientes envían requerimientos tan pronto encuentra una referencia a objeto
- Poco más de un RTT para solicitar todos los objetos

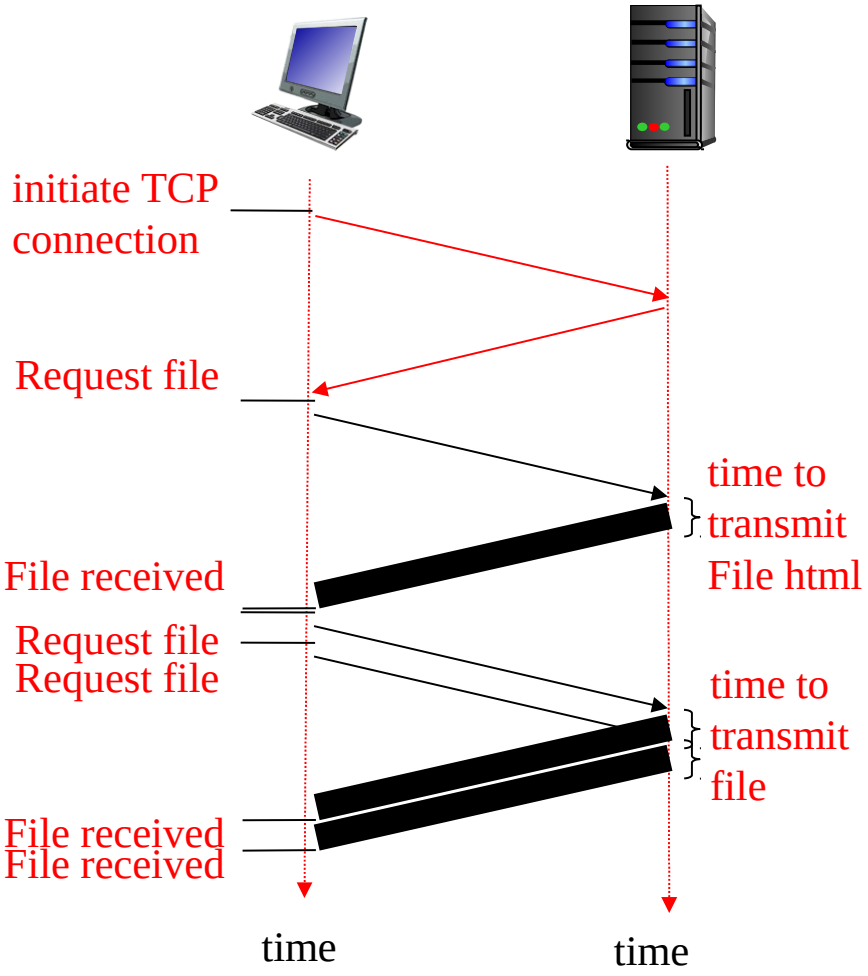
HTTP No persistente: un .html chico con varios objetos en él



Conexiones posteriores podrían ser paralelas.

En todos estos diagramas suponemos que los objetos caben en un segmento (=paquete) TCP.

HTTP persistente



Mensaje HTTP de requerimiento

- Dos tipos de mensajes HTTP: *requerimiento y respuesta*
- Mensaje de requerimiento HTTP:
  - ASCII (es decir, formato legible)

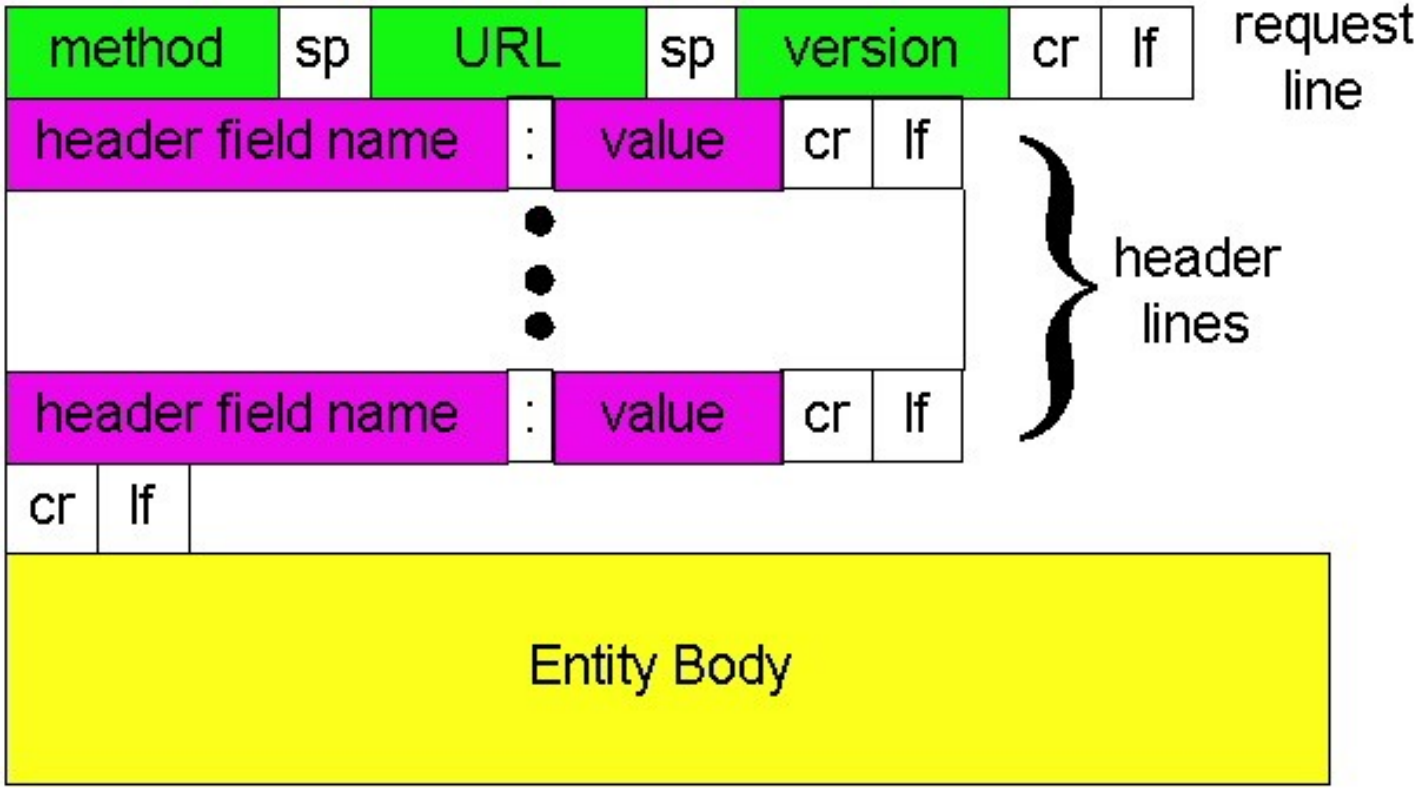
Línea de requerimiento (métodos GET, POST, HEAD, PUT, Delete)

Líneas de encabezado

Carriage return, line feed al inicio de la línea Indica fin de mensaje

```
GET /somedir/page.html HTTP/1.1\r\n
Host: www.someschool.edu\r\n
User-agent: Firefox/3.6.10.0\r\n
Connection: keep-alive\r\n
Keep-Alive: 115\r\n
Accept-language:fr\r\n
Connection: keep-alive\r\n
\r\n
```

Mensaje HTTP de requerimiento: formato general



Subiendo datos de formulario

Vía Método Post:

- Páginas Webs usualmente incluyen entradas de formularios
- Los datos son subidos al servidor en el cuerpo del mensaje

Vía Método URL:

- Usa método GET
- Entrada es subida en campos URL de la línea de requerimiento:

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

Tipos de Métodos

HTTP/1.0

- GET
- POST
- HEAD
  - Pide al servidor que deje el objeto requerido afuera de la respuesta. Respuesta incluye sólo el encabezado.

HTTP/1.1

- GET, POST, HEAD
- PUT
  - Sube archivos en cuerpo del requerimiento en localización indicada por el campo URL
- DELETE
  - Borra archivo especificado en el campo URL

Mensajes HTTP de respuesta

Línea de estatus (código de estatus del protocolo Frase de estatus)

Líneas de encabezado

data, e.g., archivo HTML solicitado

```
HTTP/1.1 200 OK\r\n
Date: Thu, 06 Aug 1998 12:00:15 GMT\r\n
Server: Apache/2.0.52 (CentosOS)\r\n
Last-Modified: Sun, 30 Jun 2016 17:00:02
    GMT\r\n
Content-Length: 6821\r\n
Content-Type: text/html
Connection: Keep-Alive\r\n
Keep-Alive: timeout=10, max=100\r\n
\r\n
data data data data data ...
```

Códigos HTTP de respuesta

En primera línea de respuesta del servidor al cliente.

Algunos códigos de muestra:

200 OK

- request exitoso, objeto requerido es incluido luego en mensaje

301 Moved Permanently

- Se movió el objeto requerido, nueva ubicación es especificada luego en el mensaje (Location:)

400 Bad Request

- Requerimiento no entendido por el servidor

404 Not Found

- Documento no encontrado en servidor

505 HTTP Version Not Supported

Probando HTTP (lado cliente)

1. Telnet a tu servidor favorito:

telnet profesores.elo.utfsm.cl 80

Telnet abre una conexión TCP al puerto 80 (puerto servidor HTTP por omisión) en profesores.elo.utfsm.cl. Cualquier cosa ingresada es enviada a puerto 80 de mateo

2. Escribir un requerimiento GET HTTP:

```
GET /~agv/elo322/HTTP/prueba.html HTTP/1.1
Host: profesores.elo.utfsm.cl

NOTA: Campo Host es obligatorio
en encabezado, requerido por proxy.
```

Tipeando esto (doble carriage return), enviamos un GET request mínimo (pero completo) al servidor HTTP

3. Observar el mensaje de respuesta enviado por el servidor HTTP!

Hacer algo similar con navegador y wireshark

Cómo conocer estado usuario-servidor: cookies

Muchos sitios Web importantes usan cookies

- Las cookies fueron implementadas para permitir personalizar la información Web.
- Cookies es información generada por un Web server y almacenada en el computador del usuario para **acceso** futuro.
- Las cookies son trasportadas entre el computador del usuario y el servidor.
- Por ejemplo, cookies son usadas para almacenar ítems en un carro de compra mientras recorres un mall virtual.

Estado usuario-servidor: cookies

Cuatro Componentes:

1) Línea encabezado cookie en el mensaje respuesta HTTP
2) Archivo cookie es almacenado en la máquina del usuario y administrada por su navegador.
3) Línea de encabezado cookie en requerimiento HTTP
4) Base de datos en sitio Web

Ejemplo:

- Susan accede Internet siempre desde el mismo PC
- Ella visita un sitio e-commerce específico por primera vez.
- Cuando el requerimiento HTTP inicial llega al sitio, éste crea un ID único y lo guarda en la base de datos para ese ID.
- En mensaje respuesta va información del sitio e ID (cookie)
- El navegador de Susan almacena la cookie en disco.
- En nuevo acceso al sitio, el navegador incluye ID. Así ese sitio sabrá que el mismo usuario reaparece.

Cookies: conservando el “estado” (cont.)

cliente

servidor

Cookie file
Ebay: 8734

Cookie file
Ebay: 8734
Amazon: 1678

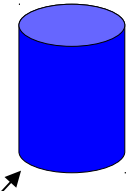
Requerimiento http

Respuesta http +
`Set-cookie: 1678`

Servidor crea ID 1678 para usuario

Entrada en base de datos

amazon



requerimiento http
`cookie: 1678`

respuesta http usual

Acción específica de la cookie

acceso

Una semana más:

Cookie file
Ebay: 8734
Amazon: 1678

requerimiento http
`cookie: 1678`

respuesta http usual

Acción específica de la cookie

acceso

Cookies (cont.)

Qué usos pueden tener las cookies:

- Autorización
- Shopping carts
- Sugerencias
- Estado de la sesión del usuario (Web e-mail)

Cómo conservar estado:

- Los de lados extremos mantienen el estado de Tx/Rx a través de transacciones múltiples usando cookies
- Usando cookies mensajes HTTP llevan estado.

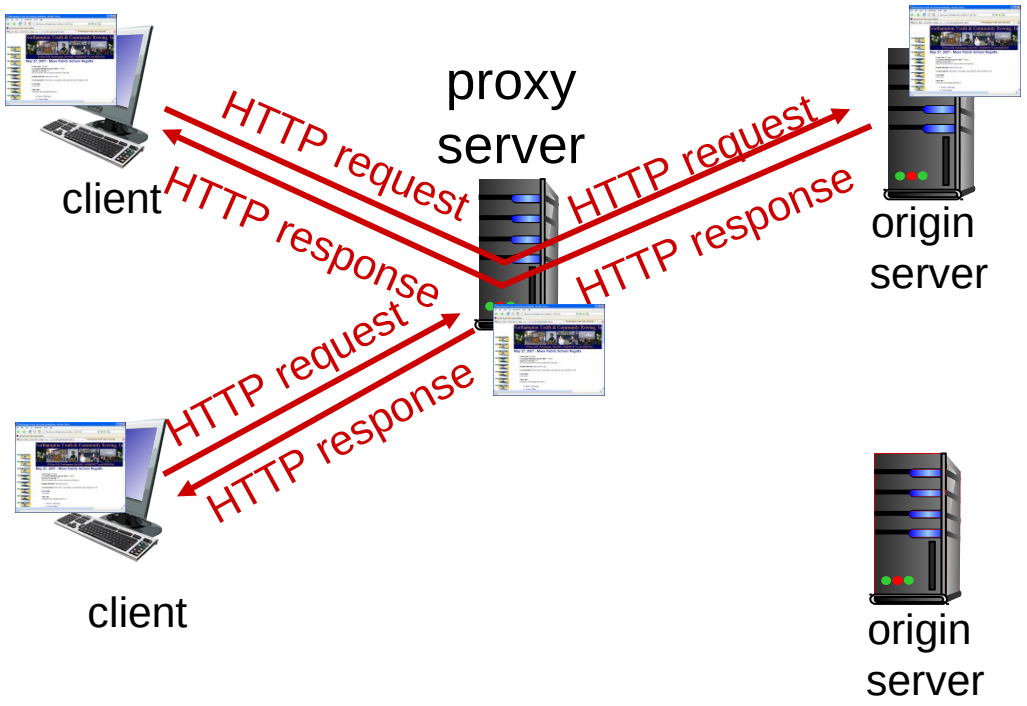
Cookies y privacidad:

- Cookies permiten que el sitio aprenda mucho sobre uno.
- Podríamos proveer nombre y correo al sitio, y éste lo recuerda.

Web caches (también servidores proxy)

Objetivo: satisfacer el requerimiento del cliente sin involucrar al servidor destino.

- Usuario configura el browser: Acceso Web vía proxy
- Browser envía todos los requerimientos HTTP al proxy
  - Si objeto está en cache: cache retorna objeto
  - Si no, cache requiere los objetos desde el servidor Web, los almacena y retorna el objeto al cliente.



Caches v/s proxy

- La idea del cache es almacenar “localmente” datos ya solicitados y así poder acceder a éstos más rápidamente en el futuro.
  - Un problema que debe atender el cache es la obsolescencia que puede tener los datos locales.
  - El cache puede usar tiempos de expiración, o consultar a la fuente por vigencia del dato local.

- Un proxy es un servicio que consiste en realizar una solicitud a pedido de otro.
- ¿Les ha pasado que para algunas cosas ustedes desean pedir a otro enviar un mensaje por ustedes?
- Por ejemplo podemos usar proxy para acceder a servicios externos de una intranet, para que desde fuera no se sepa qué computadores hay dentro. Los servidores verán un mismo origen para todas las consultas de la intranet.
- Es muy conveniente instalar un cache en un proxy.

Más sobre Web caching

- Un proxy-cache actúa como cliente y servidor
- Típicamente el proxy-cache es instalado por ISP (universidad, compañía, ISP residencial)

Por qué Web caching?

- Reduce tiempo de respuesta a las peticiones del cliente.
- Reduce tráfico en el enlace de acceso al ISP.
- Internet densa con caches permite a proveedores de contenido “chicos” (poco $ $) entregar contenido en forma efectiva.

Ejemplo de Cache

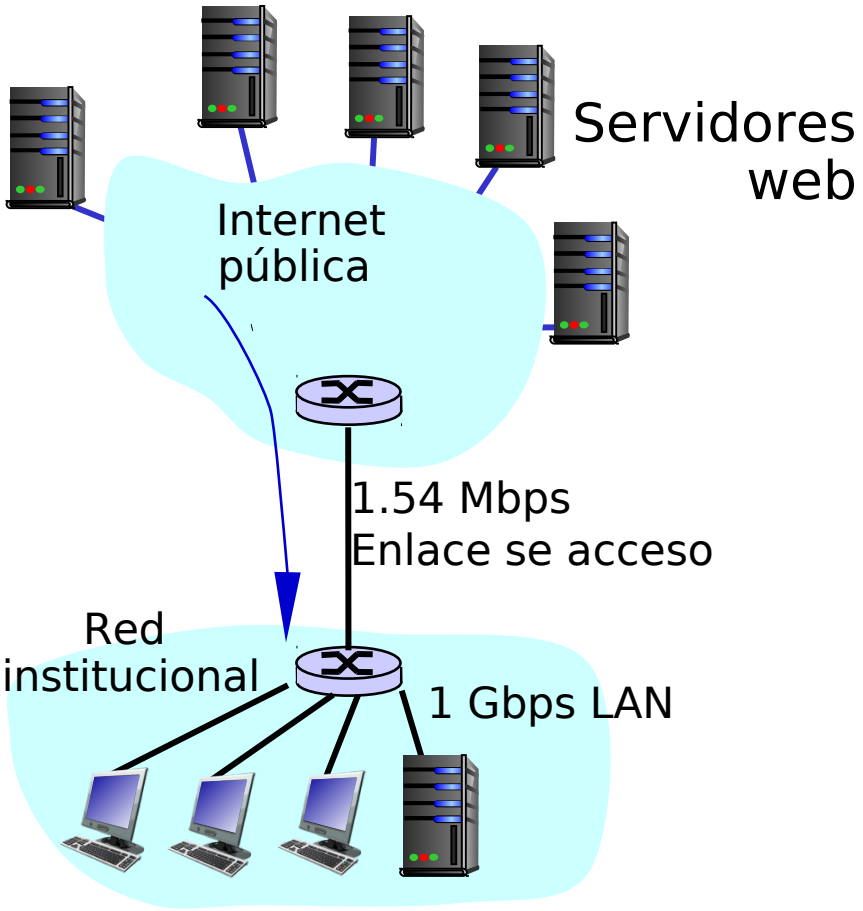
Suposiciones

- Tamaño promedio de objetos = 100 Kbits
- Tasa de requerimientos promedio desde browsers de la institución a servidores WEB = 15/sec
- => Datos promedios a browsers 1.50 Mbps (=15*100 kbps).
- Retardo desde el router institucional a cualquier servidor web y su retorno = 2 sec

Consecuencias

- utilización de la LAN =0.15%
- utilización del enlace de acceso = **99**%
- Retardo total = retardo Internet + retardo de acceso + retardo LAN

  = 2 sec + minutos + microsec



Sin Cache institucional

Hay cuello de botella en la entrada.

Ejemplo de Cache (cont)

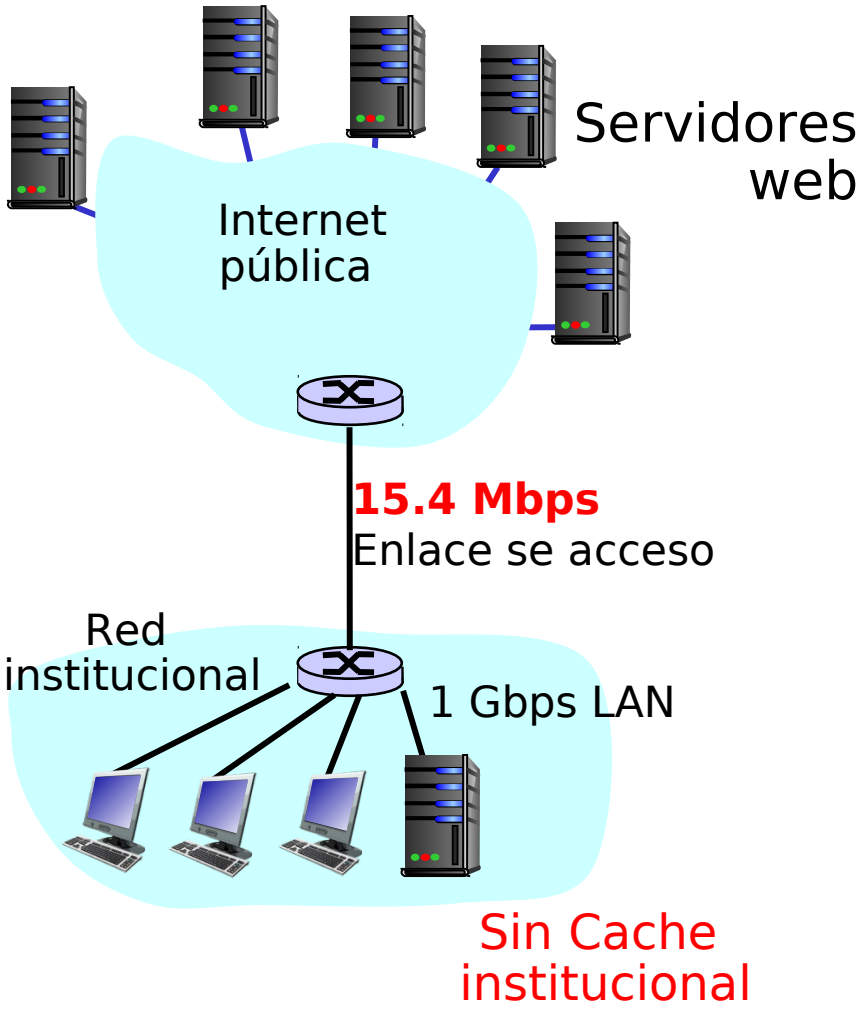
Posible solución

- Aumentar ancho de banda del enlace, por ejemplo, a 15.4Mbps

Consecuencias

- Utilización de la LAN = 0.15%
- Utilización del enlace de acceso = 9.9%
- Retardo Total = Retardo Internet + retardo de acceso + retardo LAN

  = 2 sec + msec + microsecs
- A menudo un upgrade caro.



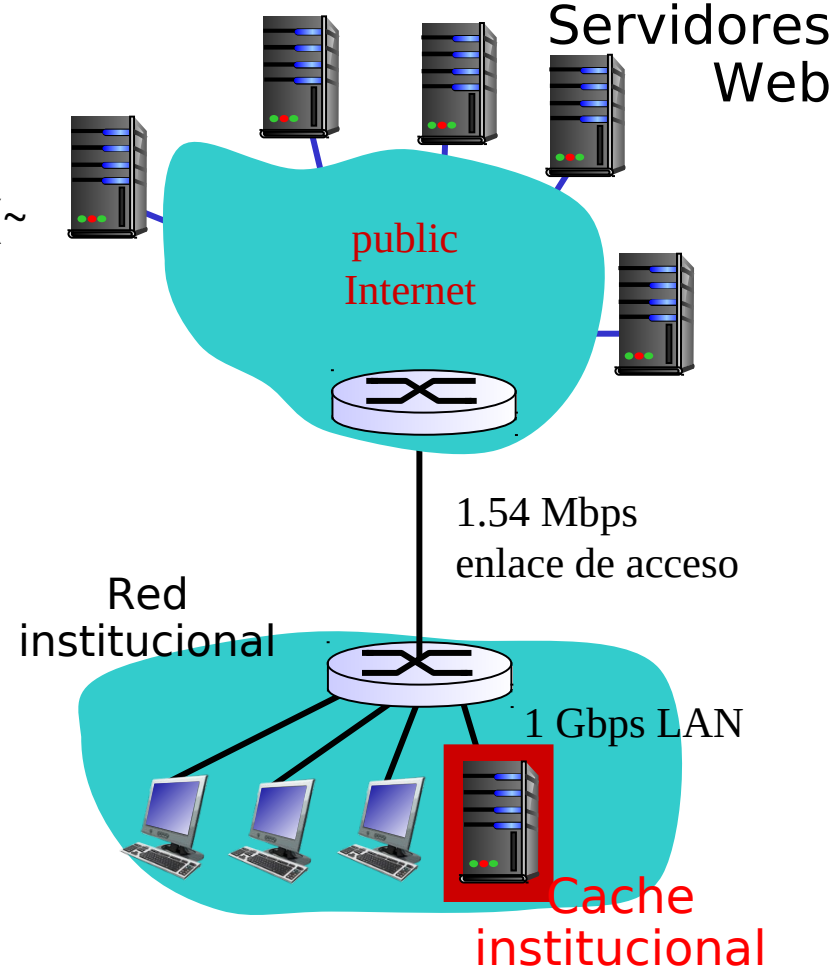
Ejemplo de cache (cont)

Instalar un web Cache

- Supongamos tasa de éxito[1] (acierto) de 0.4

Consecuencias

- 40% de los requerimientos serán satisfechos en forma casi inmediata (~ msec) + consulta por obsolescencia
- 60% de los requerimientos satisfechos por servidores WEB
- Utilización del enlace de acceso por browsers = 0.6*1.5 Mbps=0.9 Mbps
- Utilización enlace =0.9/1.54=0.58
- Retardo total = Retardo Internet + retardo acceso + retardo LAN = 0.6*(retardo de servidores web) + 0.4*(retardo respuesta cache) = 0.6*(2.xx) +0.4*(~msec) =~1.2 sec



40% Se debe agregar tiempo para confirmar que dato en cache es vigente.

[1]Tasa de éxito: Fracción de los requerimientos satisfechos por el cache.

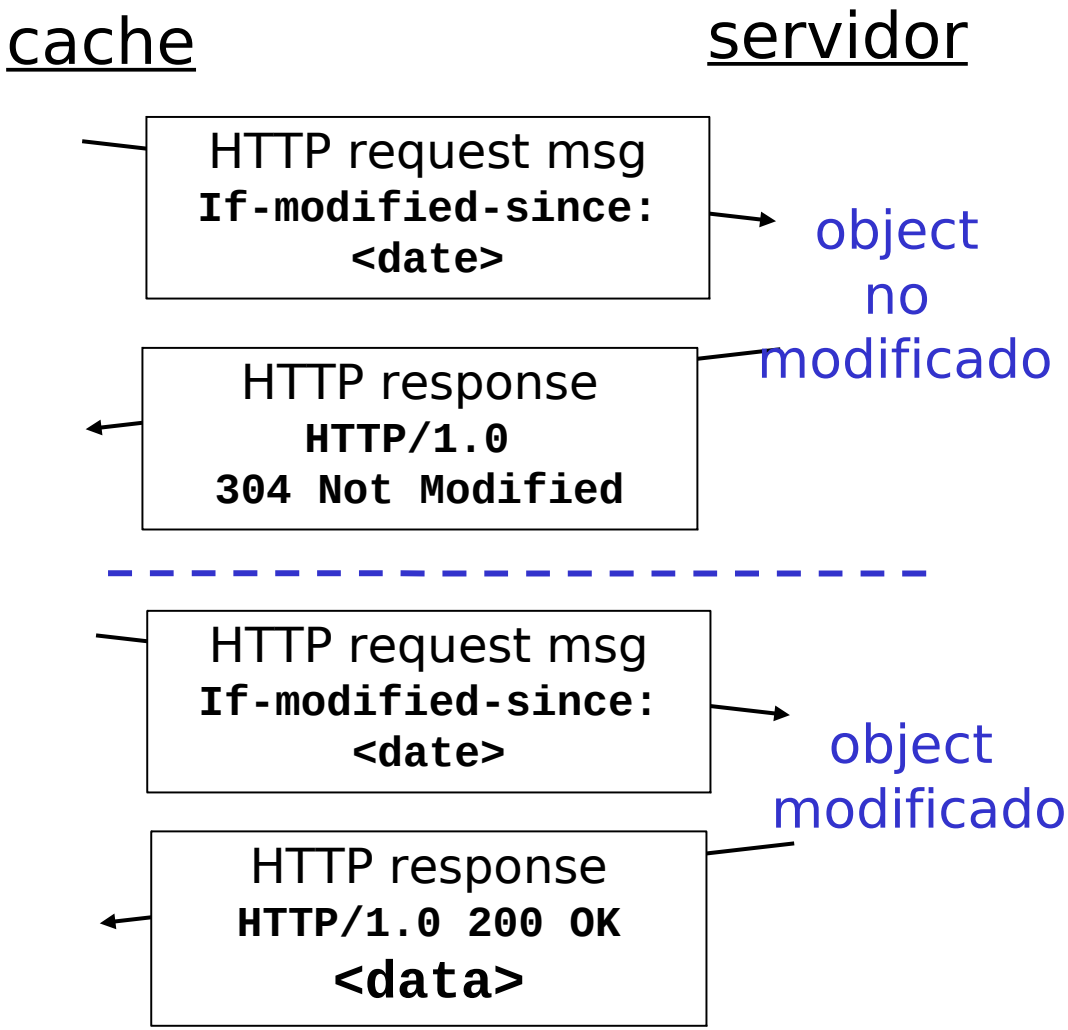
Get Condicional

- Objetivo: verificar que el cache tiene la versión actualizada de un objeto
- Cache: especifica la fecha de la copia en el requerimiento HTTP

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

- Servidor: responde sin el objeto si la copia de la cache es la última. :

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



Capítulo 2: Capa Aplicación

- 2.1 Principios de las aplicaciones de red
- 2.2 Web y HTTP
- 2.3 Correo Electrónico
  - SMTP, POP3, IMAP
- 2.4 DNS
- 2.5 P2P para archivos compartidos
- 2.6 Video streaming y redes de distribución de contenidos
- 2.6 Programación de sockets con UDP y TCP