Módulo 02
La Capa de Aplicaciones (Pt. 2)

Redes de Computadoras
Depto. de Cs. e Ing. de la Comp.
Universidad Nacional del Sur

Copyright

Contenidos

- Servicios que requiere la capa de aplicaciones
- Protocolos de la capa de aplicaciones
  - **HTTP**
  - **DNS**
  - **SMTP**, **POP** e **IMAP**
- Arquitectura de las aplicaciones **P2P**
- Programación basada en sockets

World Wide Web

- Desde el punto de vista de las aplicaciones, las páginas web son meras colecciones de objetos
- Esos objetos pueden ser documentos **HTML** (Hyper-Text Markup Language), imágenes en formato **JPEG** o **PNG**, etc.
- Cuenta con un archivo **HTML** base
- Todos los objetos se direccionan a través de un **URL** (Uniform Resource Locator):

http://cs.uns.edu.ar:80/~ags/RC/index.html

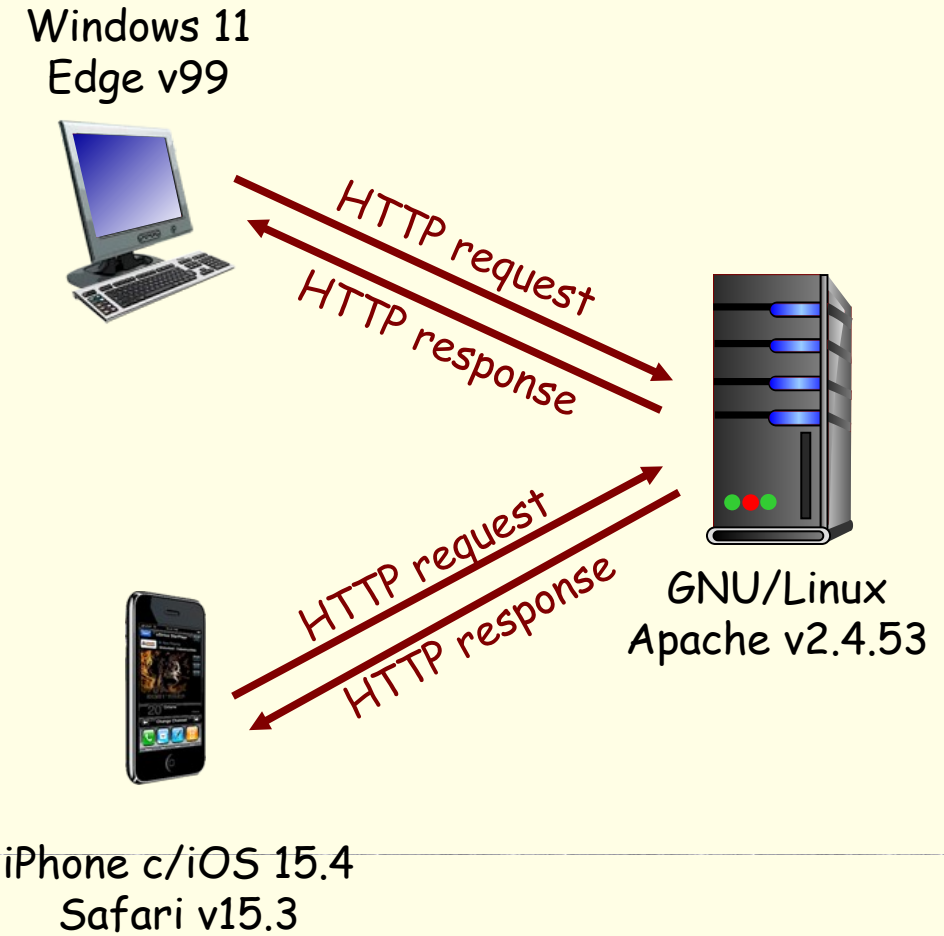
protocolo computadora puerto documento

El protocolo HTTP

- El protocolo **HTTP** (Hyper-Text Transfer Protocol) fue concebido por el padre de la web, Sir Tim Berners-Lee
  - Es el protocolo de la capa de aplicaciones de la web
- Existen tres versiones del protocolo:
  - **HTTP**/1.0 (**RFC 1945**)
  - **HTTP**/1.1 (**RFC 2068**)
  - **HTTP**/2 (**RFC 7540**)

El protocolo HTTP

- Adopta un claro modelo cliente-servidor:
  - Los clientes, llamados navegadores, piden, reciben y muestran por pantalla “objetos”
  - Los servidores, llamados servidores web, envían los “objetos” que le soliciten



El protocolo HTTP

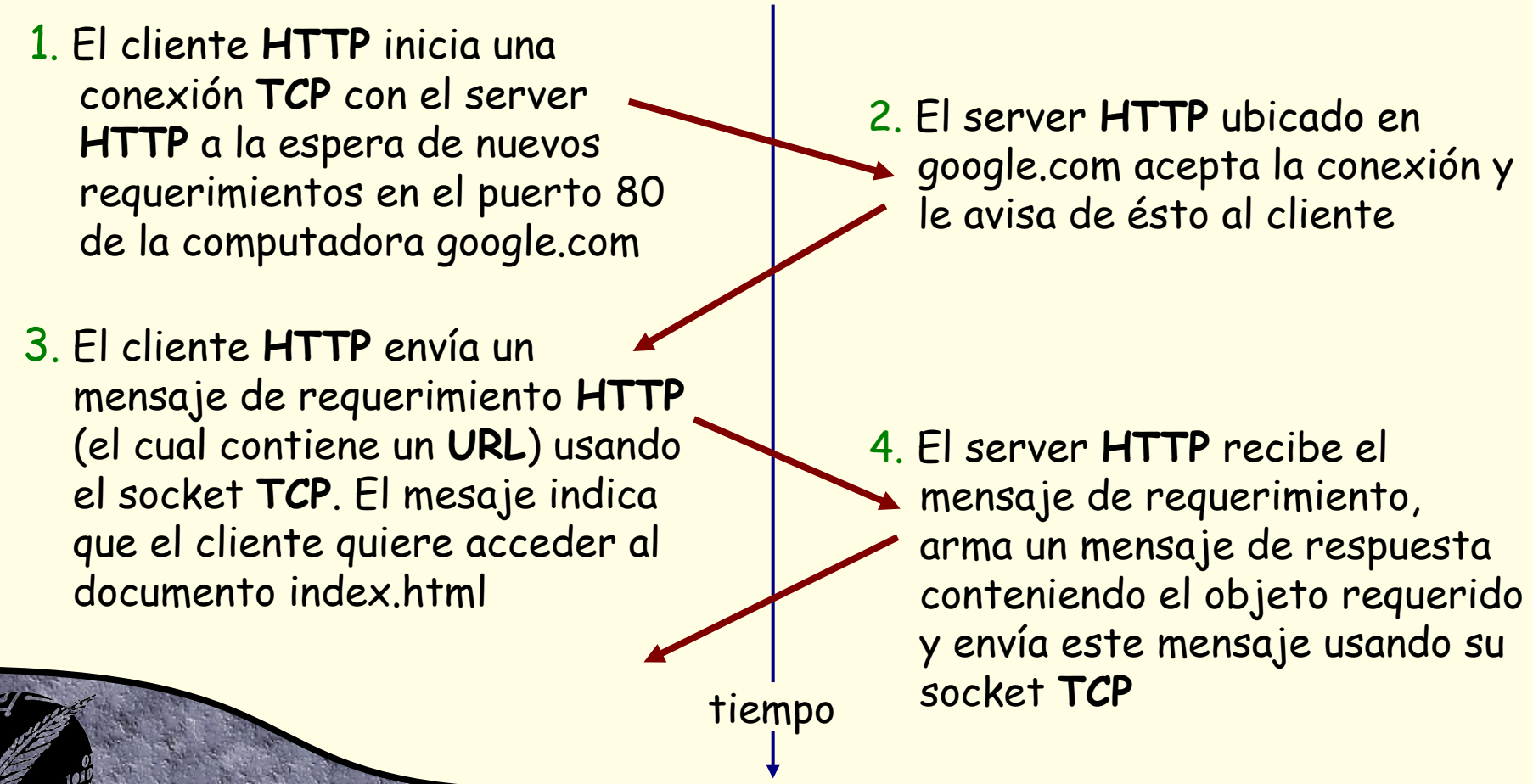
- A grandes rasgos, el protocolo **HTTP** se compone de las siguientes fases:
  - El cliente abre una conexión **TCP** al puerto 80 del servidor mediante un socket
  - El servidor acepta la conexión **TCP** del cliente
  - Se intercambian mensajes **HTTP** entre el cliente y el servidor, respetando el protocolo **HTTP**
  - Al terminar, se finaliza la conexión **TCP**
- **HTTP** es un protocolo sin estado

Estilos de conexión

- **HTTP** no persistente:
  - A lo sumo un objeto es enviado a través de cada conexión **TCP**
  - Es el estilo de conexión adoptado por **HTTP**/1.0
- **HTTP** persistente:
  - Múltiples objetos pueden ser enviados por la misma conexión **TCP**
  - Es el estilo de conexión adoptado en el modo por defecto de **HTTP**/1.1

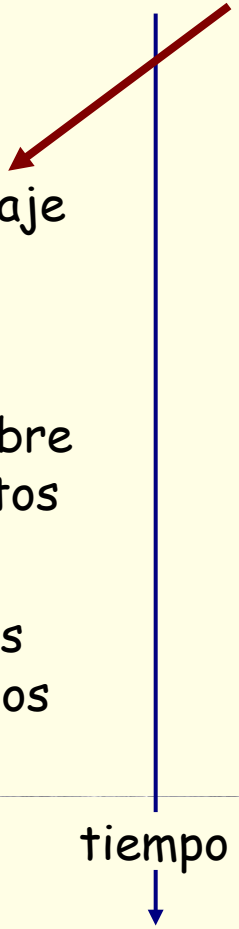
Traza HTTP/1.0

- Supongamos que el usuario ingresa la dirección **http://google.com** en el navegador



Traza HTTP/1.0

- Continúa:



5. El server **HTTP** cierra la conexión, pues el estilo adoptado es el no persistente

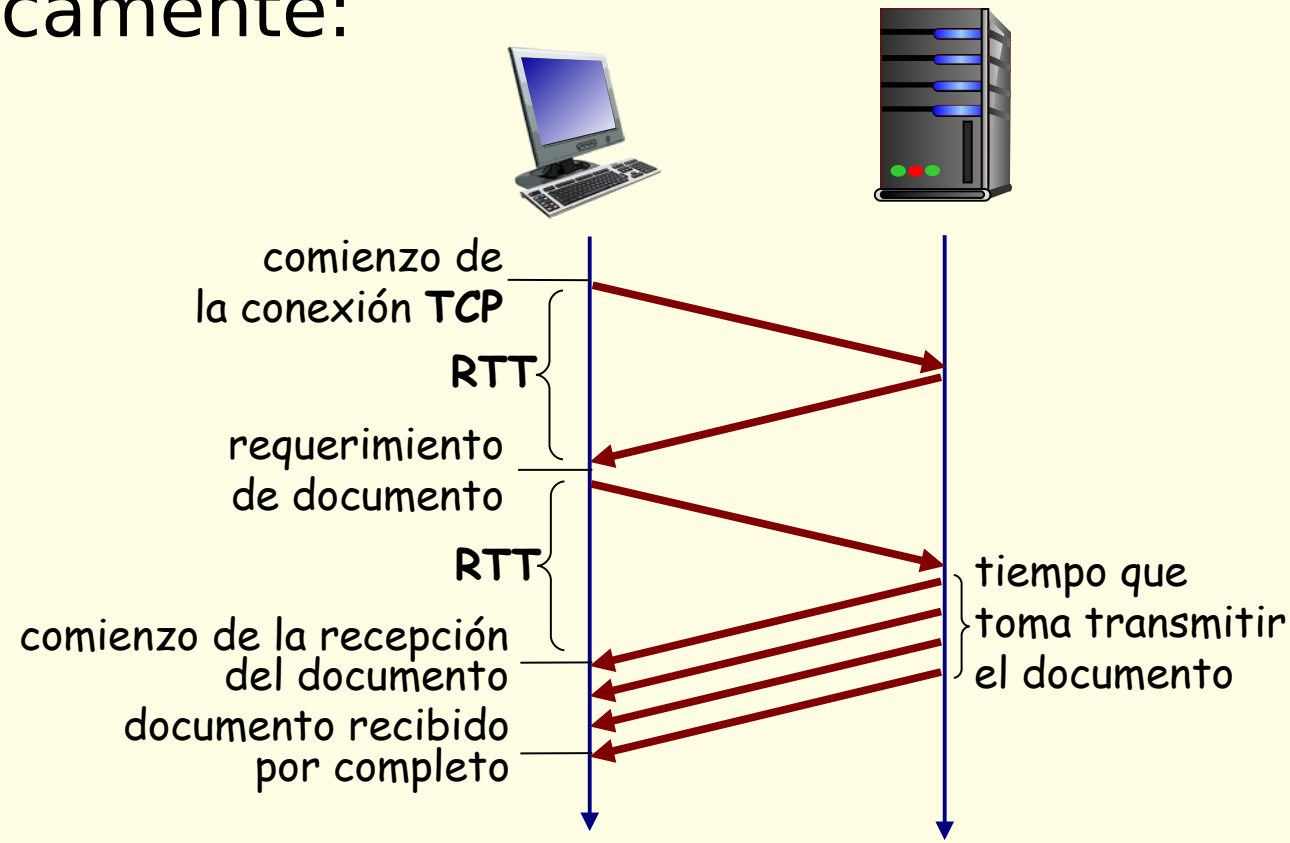
6. El cliente **HTTP** recibe el mensaje de respuesta conteniendo el archivo **HTML** solicitado y lo muestra por pantalla.
   Al recorrer este archivo descubre tres referencias a otros objectos (por caso, archivos **PNG**)
7. El cliente **HTTP** repite los pasos 1.-6. para cada uno de los objetos restantes

Tiempo de Respuesta

- ¿Podremos acotar de alguna manera el tiempo de respuesta de un servidor **HTTP**?
- Denominaremos **RTT** (Round-Trip Time) al tiempo que le toma a un mensaje arbitrario en ir del cliente al servidor y volver
  - Hace falta un **RTT** para establecer la conexión
  - Hace falta otro **RTT** para enviar el pedido y recibir los primeros bytes de la respuesta correspondiente
  - Finalmente, hace falta esperar que se termine de transferir el documento solicitado

Tiempo de Respuesta

- Gráficamente:



Tiempo Total ≈ 2 RTT + Tiempo de Transmisión

Persistente vs. no persistente

- El estilo de conexión no persistente presenta ciertos inconvenientes:
  - Requiere 2 **RTT** por cada objeto transferido
  - El sistema operativo tiene que establecer, mantener y cerrar una conexión **TCP** por cada objeto transferido
  - Los navegadores tienden a abrir múltiples conexiones **TCP** simultáneas para recuperar todos los objetos referenciados en el menor tiempo posible

Persistente vs. no persistente

- El estilo de conexión persistente parece corregir estos defectos:
  - El servidor no cierra la conexión **TCP** luego de atender el requerimiento **HTTP**
  - Los subsecuentes mensajes **HTTP** se envían y reciben reutilizando la conexión **TCP** preexistente
- ¿Cómo podemos hacer si necesitamos acceder a más de un objeto a la vez?
  - ¿Esperamos a terminar el requerimiento anterior antes de solicitar el siguiente?

Persistencia con y sin pipeline

- Estilo de conexión persistente sin pipeline:
  - El cliente sólo puede reutilizar la conexión **TCP** preexistente cuando el último requerimiento enviado ya haya sido atendido y contestado
  - Un **RTT** para cada objeto referenciado
- Estilo de conexión persistente con pipeline:
  - El cliente reutiliza la conexión **TCP** en todo momento
  - En el mejor de los casos insume tan solo un **RTT** para la totalidad de los objetos
  - Es el estilo adoptado por **HTTP**/1.1

HTTP/2

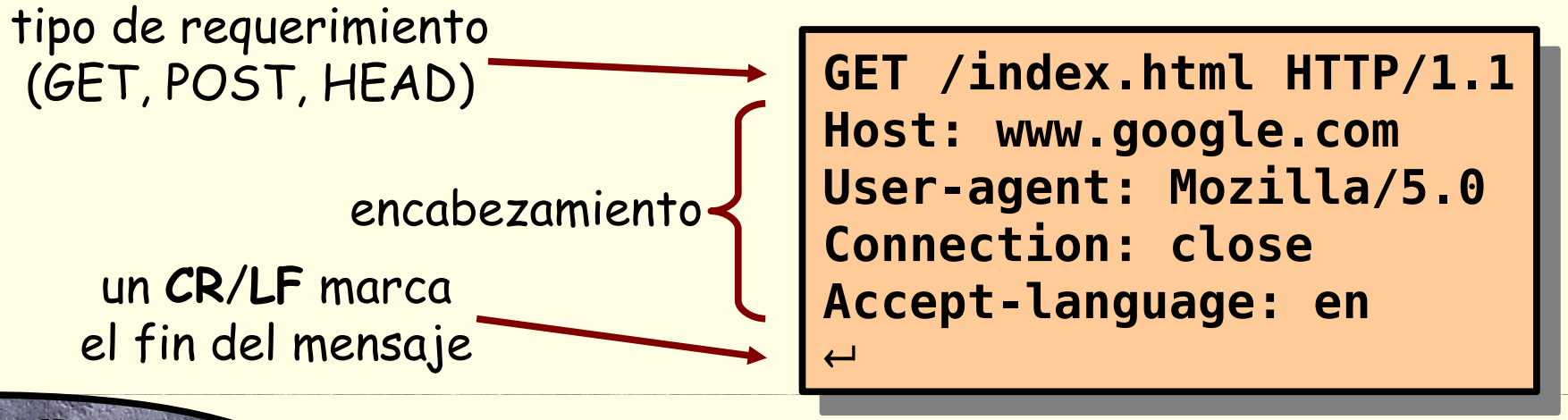
- El estándar **HTTP**/2 (antes conocido como **HTTP**/2.0), fue sancionado recientemente por la **IETF**, el organismo a cargo de la aprobación de los nuevos **RFC**s
- Se basa en un protocolo actualmente en uso por la compañía Google en su navegador **Chrome** al acceder a los servidores propios
  - Este protocolo se denomina **SPDY** e implementa ciertas optimizaciones que hacen que las páginas webs resulten mucho más reactivas

HTTP/3

- La rapidez en sancionar el estándar **HTTP/2** generó críticas por la oportunidad perdida para atender otros cuestionamientos
  - Por caso, las conexiones multiplexadas no se llevan del todo bien con el transporte confiable de información que implementa **TCP**
- La propuesta que está siendo explorada en el borrador **HTTP/3** consiste en implementar transporte confiable de información ¡sobre **UDP**!
  - Cabe acotar que todavía no se trata de un estándar, si bien algunos navegadores ya lo implementan

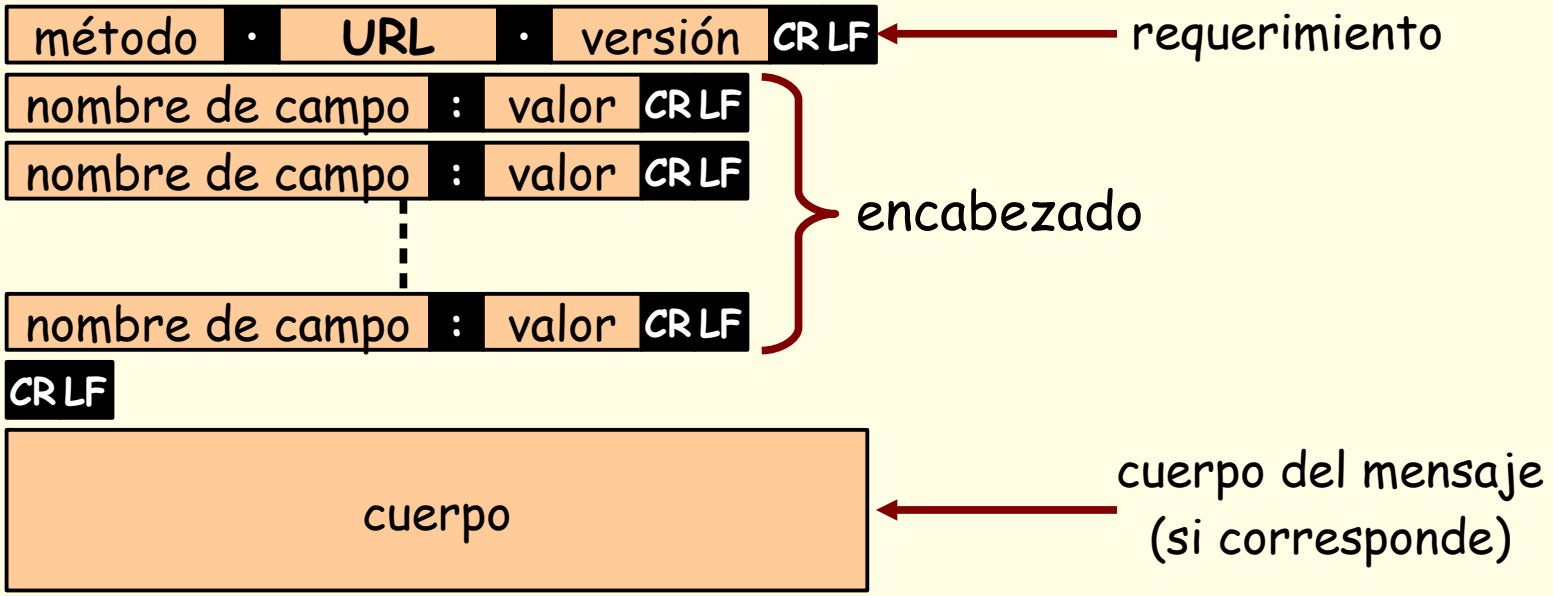
Mensajes HTTP

- **HTTP** contempla sólo dos tipo de mensajes:
  - Los requerimientos (**HTTP** request)
  - Las respuestas (**HTTP** response)
- Los mensajes son enviados en **ASCII**, es decir, una codificación entendible por los humanos



Mensajes de requerimiento

- El formato general de los mensajes de requerimiento **HTTP** es el siguiente:



Envío de información

- Hasta ahora los mensajes **HTTP** vistos sólo permiten obtener información del servidor
  - En este caso, el cuerpo del mensaje es nulo
- Para poder enviar información ingresada por el usuario se puede usar el método POST
  - El cuerpo del mensaje contiene esa información
- También se puede usar el método GET, pasando lo ingresado por el usuario dentro del **URL**

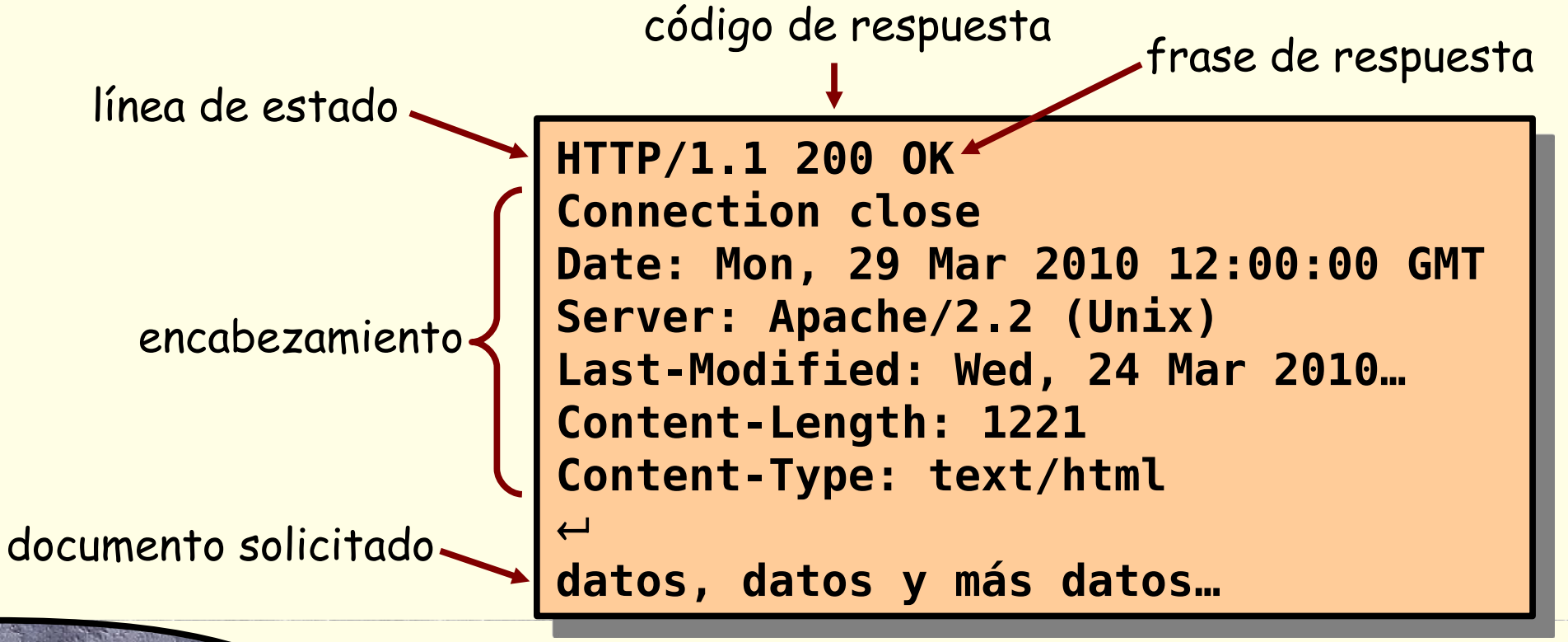
```
http://cs.uns.edu.ar/form.html?nombre=foo&apellido=bar
```

Métodos disponibles

- **HTTP**/1.0:
  - GET, para acceder a los distintos objetos **HTML**
  - POST, para enviar datos ingresados por el usuario
  - HEAD, para verificar la validez de los hipervínculos
- **HTTP**/1.1:
  - GET, POST y HEAD, igual que antes
  - PUT, para publicar un archivo en una determinada ubicación dentro del servidor
  - DELETE, para eliminar un archivo del servidor

Mensajes de respuesta

- Los mensajes de respuesta **HTTP** sólo se generan a consecuencia de una solicitud previa



Códigos de respuesta

- Una solicitud **HTTP** puede recibir cinco clases de respuestas, distinguidas por su código:
  - 1xx, mensajes de información (si bien el pedido no fue resuelto aún, tampoco ha sido rechazado)
  - 2xx, éxito, la solicitud fue recibida, entendida, atendida y respondida
  - 3xx, redirección, hace falta hacer ciertas correcciones a la solicitud original
  - 4xx, error insalvable por parte del cliente
  - 5xx, error insalvable por parte del servidor

Códigos de respuesta

- Ejemplos en concreto de códigos de respuesta:
  - 200 OK, requerimiento aceptado y cumplido
  - 301 Moved Permanently, el objeto solicitado fue movido (la nueva ubicación es informada)
  - 400 Bad Request, el mensaje de solicitud no fue entendido por el servidor
  - 404 Not Found, el documento solicitado no fue encontrado
  - 503 Service Unavailable, el servidor está temporalmente fuera de servicio

Chateando con un servidor

- ¿Por qué será que este protocolo es tan “conversado” por así decir?

```
$ telnet www.google.com.ar 80
…
GET /index.html HTTP/1.0
Host: www.google.com.ar
```

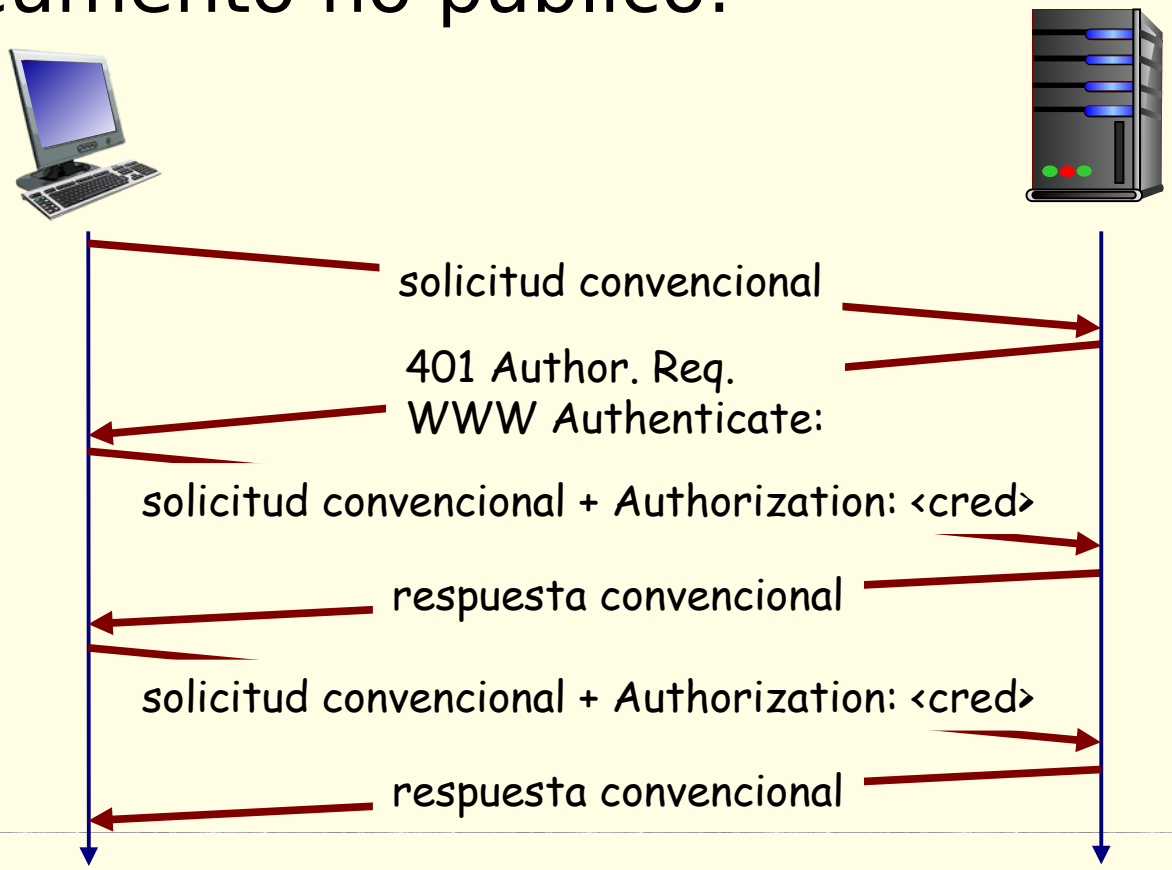
- Otra opción es directamente capturar los mensajes intercambiados usando el Wireshark o alguna extensión a tal efecto del navegador

Restricción de acceso

- **HTTP** contempla un modelo de restricción de acceso simple, si bien un tanto básico
- Es posible indicar en el servidor qué partes son públicas y qué partes requieren autorización para poder ser accedidas
  - Las credenciales típicas de acceso son la combinación nombre de usuario y contraseña
  - Se trata de un modelo sin estado, es decir, el cliente tiene que suministrar el nombre y la contraseña en cada interacción

Intercambio de mensajes

- Supongamos que intentamos acceder a un documento no público:



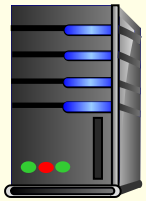
Cookies

- Las cookies constituyen un mecanismo bastante eficaz para sobrellevar la naturaleza sin estado del protocolo **HTTP**
  - Sin saber qué pasó antes no es posible implementar carritos de compras, evitar tener que estar enviando siempre las credenciales junto a cada solicitud, etc.
- Las cookies son en esencia un mapeo entre claves y valores
  - Este mapeo se almacena localmente en una carpeta mantenida por el navegador

Cookies

- Las páginas web de la mayoría de los principales sitios hacen uso de ellas
  - Por caso, Google, Facebook, Mercadolibre, etc.
- La utilización de cookies involucran cuatro componentes:
  - El campo de las cookies en las respuestas **HTTP**
  - El campo de las cookies en las solicitudes **HTTP**
  - El repositorio local de cookies
  - La base de datos de usuarios en el servidor web

Esquema de interacción



Repositorio

ebay: 4321

solicitud convencional

el server crea un nuevo número de usuario (# 1234)

respuesta convencional + Set-Cookie: 1234

Repositorio

amazon: 1234

ebay: 4321

solicitud convencional + Cookie: 1234

respuesta convencional

DB

-una semana después-

Repositorio

amazon: 1234

ebay: 4321

solicitud convencional + Cookie: 1234

respuesta convencional

respuestas específicas en función del número de usuario suministrado

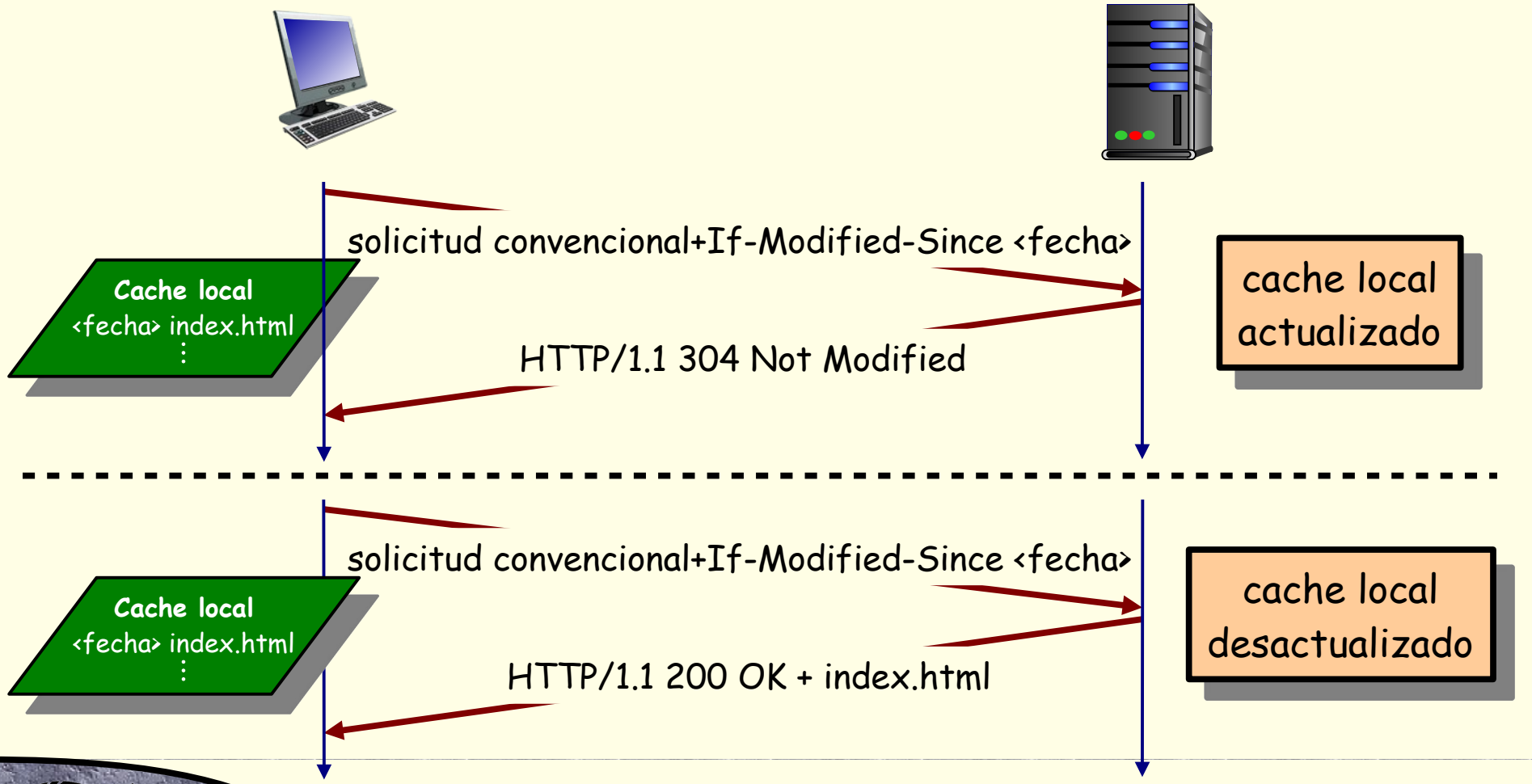
Ventajas y desventajas

- Ventajas:
  - Posibilita nuevos modelos de autorización
  - Carritos de compras y recomendaciones personalizadas
  - Mantener el estado de una sesión (webmail, etc.)
- Desventajas:
  - Resulta extremadamente difícil evitar que los servidores sepan más de la cuenta acerca de nosotros
  - ¿Es realmente gratis Google? ¿Y Facebook?

Cache local

- La implementación del estándar **HTTP**/1.1 mejoró el desempeño de los clientes y especialmente de los servidores
- ¿Será posible optimizar de alguna forma el tiempo de 2 **RTT** + transferencia del objeto?
  - Si bien el navegador no tiene control sobre el valor circunstancial del **RTT**, en ocasiones si puede evitar pagar el costo de transferencia
  - El cliente puede conservar una copia local, recibida anteriormente, de un dado objeto y sólo consultarle al servidor si la copia está aún vigente

Interacción con el cache local



Servidor proxy

- Un servidor proxy (también web cache) es en esencia una generalización de la idea del cache propio del navegador a toda una red local
  - La idea es poder satisfacer la mayor cantidad de requerimientos directamente desde el web cache, sin tener que acceder a los servidores en internet
  - El navegador se configura para que envíe todos sus requerimientos al servidor proxy
  - Los objetos presentes en el cache del proxy son enviados al cliente directamente; los restantes tienen que ser primero recuperados de servidor en internet

Servidor proxy

- El servidor proxy debe actuar como cliente y servidor en simultáneo
- El proxy inspecciona el valor del campo **`If-modified-since`**
  - El server proxy debería consultar al servidor en internet, pero... ¡hacer esto no tiene sentido!
  - La solución implementada consiste en hacer uso de heurísticas para decidir si la copia en el cache está o no actualizada sin consultar al servidor en internet

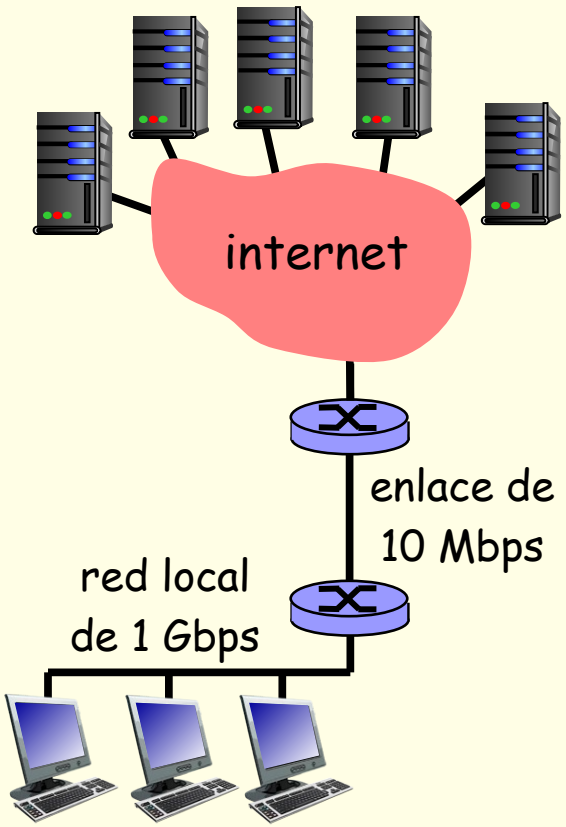
Servidor proxy

- Implementar un servidor proxy tiene sus costos
  - ¿Vale la pena asignar recursos a un servidor proxy?
- Beneficios:
  - Se logra reducir el tiempo de respuesta
  - También se reduce la ocupación del enlace a internet
  - La presencia de servidores proxy en la mayoría de las redes locales hace que incluso los servidores web más precarios puedan atender a un gran número de requerimientos a la vez

Métricas de desempeño

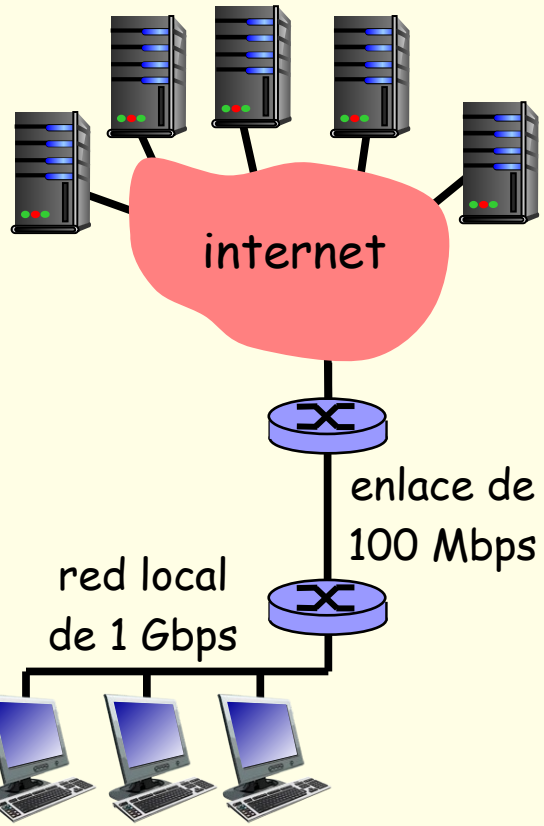
- Supongamos que nos enfrentamos con el siguiente escenario:
  - Tamaño por objeto: 100 Kb
  - Pedidos por segundo: 150
  - RTT a cualquier servidor: 2 s
- En este contexto:
  - Utilización **LAN**: 1.5%
  - Utilización enlace: 100%
  - Retardo: 2 s + minutos + ms

WAN enlace LAN



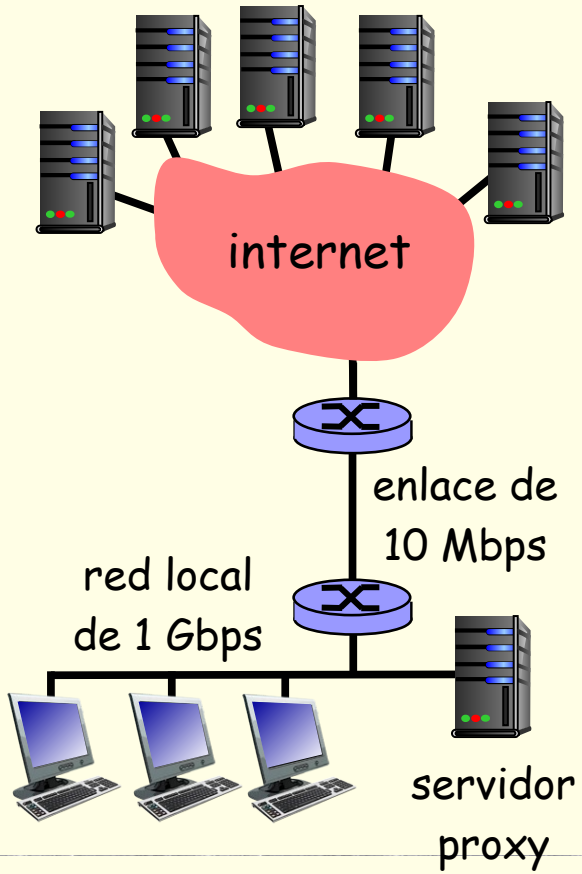
Métricas de desempeño

- Mejorando el enlace a 100 Mbps (a un mayor costo) se observa lo siguiente:
  - Utilización **LAN**: 1.5%
  - Utilización enlace: 15%
  - Retardo: 2 s + ms + ms



Un ejemplo numérico

- Implementando en cambio un servidor proxy con un hit-rate de 40%:
  - Utilización **LAN**: 1.5%
  - Utilización enlace: 90%
  - Retardo: <<2 s
  - El 40% de los requerimientos se resuelven a 1 Gbits de velocidad (ms).
  - El 60% restante insume el retardo usual (2 s + ms + ms)



¿Preguntas?