Módulo 02
La Capa de Aplicaciones (Pt. 2)

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

Copyright

- Copyright © **2010-2024** A. G. Stankevicius
- Se asegura la libertad para copiar, distribuir y modificar este documento de acuerdo a los términos de la **GNU** Free Documentation License, versión 1.2 o cualquiera posterior publicada por la Free Software Foundation, sin secciones invariantes ni textos de cubierta delantera o trasera
- Una copia de esta licencia está siempre disponible en la página **http://www.gnu.org/copyleft/fdl.html**
- La versión transparente de este documento puede ser obtenida de la siguiente dirección:

http://cs.uns.edu.ar/~ags/teaching

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`

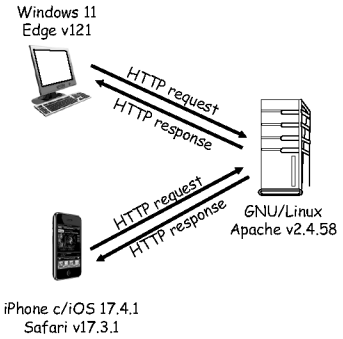
protocolo — computadora — puerto — documento

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 cuatro variantes:
  - **HTTP**/1.0 (**RFC 1945**)
  - **HTTP**/1.1 (**RFC 9112**)
  - **HTTP**/2 (**RFC 9113**)
  - **HTTP**/3 (**RFC 9114**)

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

Estas consultas sí que van

You
¿no podés hacer una comparativa de la modalidad persistente versus no persistente en el protocolo HTTP?

ChatGPT
Claro, aquí tienes una comparativa entre la modalidad persistente y no persistente en el protocolo HTTP:

Modalidad No Persistente:

- **Funcionamiento:** En la modalidad no persistente, cada solicitud de recurso (por ejemplo, una página web, una imagen, un archivo) genera una conexión TCP separada entre el cliente y el servidor. Una vez que se completa la transferencia del recurso solicitado, la conexión TCP se cierra.
- **Ventajas:**
  1. **Simplicidad:** Es más simple de implementar tanto en el cliente como en el servidor, ya que cada solicitud y respuesta se maneja de forma independiente.
  2. **Menor sobrecarga de recursos:** Al cerrar la conexión después de cada solicitud, se liberan los recursos de red y se reduce la carga en el servidor.
- **Desventajas:**
  1. **Overhead de conexión:** La apertura y cierre repetidos de conexiones TCP pueden resultar en un mayor overhead de conexión, especialmente para sitios web con múltiples recursos.
  2. **Retraso de inicio:** El tiempo necesario para establecer una conexión TCP puede agregar un retraso significativo, especialmente para sitios web con muchos recursos pequeños.
  3. **Limitación en la optimización del rendimiento:** Las técnicas de optimización de rendimiento, como la compresión de recursos y el uso de cachés, pueden ser menos efectivas debido a la naturaleza de abrir y cerrar conexiones frecuentes.

Modalidad Persistente:

- **Funcionamiento:** En la modalidad persistente, se mantiene abierta una conexión TCP entre el cliente y el servidor después de que se completa la primera solicitud. Esta conexión persistente se reutiliza para enviar múltiples solicitudes y recibir múltiples respuestas sin cerrarse.
- **Ventajas:**
  1. **Menor latencia:** Al mantener abierta la conexión TCP, se reduce el tiempo de establecimiento de la conexión en solicitudes posteriores, lo que puede mejorar la latencia percibida por el usuario.
  2. **Menor overhead de conexión:** Al evitar la apertura y cierre repetidos de conexiones TCP, se reduce el overhead de conexión y se mejora la eficiencia de la transferencia de datos.
- **Desventajas:**
  1. **Mayor complejidad:** La implementación de la persistencia requiere manejar la gestión del estado de las conexiones y la multiplexación de solicitudes y respuestas dentro de una conexión TCP.
  2. **Posibles problemas de escalabilidad:** Si no se gestiona correctamente, mantener conexiones persistentes puede aumentar la carga en el servidor y afectar la escalabilidad del sistema.

En resumen, la modalidad persistente en HTTP ofrece ventajas en términos de rendimiento y eficiencia al reducir el overhead de conexión y la latencia percibida por el usuario. Sin embargo, requiere una mayor complejidad en la implementación y la gestión de conexiones, lo que puede tener implicaciones en la escalabilidad del sistema. La elección entre persistente y no persistente depende de los requisitos específicos de la aplicación y las consideraciones de rendimiento y eficiencia.

Traza HTTP/1.0

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

1. El cliente **HTTP** inicia una conexión **TCP** con el server **HTTP** a la espera de nuevos requerimientos en el puerto 80 de la computadora google.com
2. El server **HTTP** ubicado en google.com acepta la conexión y le avisa de ésto al cliente
3. El cliente **HTTP** envía un mensaje de requerimiento **HTTP** (el cual contiene un **URL**) usando el socket **TCP**. El mesaje indica que el cliente quiere acceder al documento index.html
4. El server **HTTP** recibe el mensaje de requerimiento, arma un mensaje de respuesta conteniendo el objeto requerido y envía este mensaje usando su socket **TCP**

tiempo

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

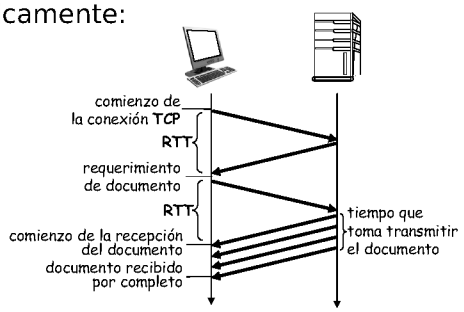
tiempo

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

¡Anotate otro poroto!

You
Y la operatoria con o sin pipeline, siempre en el contexto del protocolo HTTP persistente?

ChatGPT
La operatoria con o sin pipeline en el contexto del protocolo HTTP persistente se refiere a cómo se gestionan las solicitudes y respuestas dentro de una conexión persistente.

Con Pipeline:

- **Funcionamiento:** En la operatoria con pipeline, el cliente envía múltiples solicitudes HTTP al servidor a través de una conexión persistente sin esperar las respuestas correspondientes. El servidor procesa las solicitudes en orden y envía las respuestas de vuelta al cliente en el mismo orden en que se recibieron las solicitudes.
- **Ventajas:**
  1. **Reducción de la latencia:** Al enviar múltiples solicitudes de manera secuencial, se puede reducir la latencia total, ya que no es necesario esperar a recibir la respuesta de cada solicitud antes de enviar la siguiente.
  2. **Aprovechamiento de la conexión persistente:** Al enviar varias solicitudes a través de una única conexión TCP persistente, se reduce el overhead de establecimiento de conexión y cierre, lo que mejora la eficiencia y el rendimiento de la comunicación.
- **Desventajas:**
  1. **Bloqueo de las respuestas:** Si una de las solicitudes enviadas en el pipeline requiere mucho tiempo para procesarse en el servidor, puede bloquear el envío de las respuestas de las solicitudes posteriores en la cola del pipeline, lo que puede afectar la percepción del rendimiento por parte del cliente.

Modalidad Persistente:

- **Funcionamiento:** En la modalidad persistente, se mantiene abierta una conexión TCP entre el cliente y el servidor después de que se completa la primera solicitud. Esta conexión persistente se reutiliza para enviar múltiples solicitudes y recibir múltiples respuestas sin cerrarse.
- **Ventajas:**
  1. **Menor latencia:** Al mantener abierta la conexión TCP, se reduce el tiempo de establecimiento de la conexión en solicitudes posteriores, lo que puede mejorar la latencia percibida por el usuario.
  2. **Menor overhead de conexión:** Al evitar la apertura y cierre repetidos de conexiones TCP, se reduce el overhead de conexión y se mejora la eficiencia de la transferencia de datos.
- **Desventajas:**
  1. **Mayor complejidad:** La implementación de la persistencia requiere manejar la gestión del estado de las conexiones y la multiplexación de solicitudes y respuestas dentro de una conexión TCP.
  2. **Posibles problemas de escalabilidad:** Si no se gestiona correctamente, mantener conexiones persistentes puede aumentar la carga en el servidor y afectar la escalabilidad del sistema.

En resumen, la modalidad persistente en HTTP ofrece ventajas en términos de rendimiento y eficiencia al reducir el overhead de conexión y la latencia percibida por el usuario. Sin embargo, requiere una mayor complejidad en la implementación y la gestión de conexiones, lo que puede tener implicaciones en la escalabilidad del sistema. La elección entre persistente y no persistente depende de los requisitos específicos de la aplicación y las consideraciones de rendimiento y eficiencia.

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 se consolidó en el último borrador del protocolo **HTTP/3** (esto es, el **RFC 9114**) que consiste en implementar el transporte confiable de información pero ¡sobre **UDP**!

¿Me lo resumís?

You
hacame una comparativa entre las distintas versiones del protocolo HTTP

ChatGPT
Aquí tienes una comparativa entre las distintas versiones del protocolo HTTP:

HTTP/1.0:

- **Fecha de lanzamiento:** 1996.
- **Características:**
  1. **Conexiones no persistentes:** Cada solicitud HTTP establecía una conexión TCP separada, lo que resultaba en una latencia significativa debido al tiempo de establecimiento de la conexión.
  2. **Mensajes de solicitud y respuesta simples:** Los mensajes HTTP/1.0 consistían en una línea de solicitud o respuesta seguida opcionalmente por cabeceras HTTP y el cuerpo del mensaje.
  3. **Soporte limitado para encabezados:** HTTP/1.0 tenía un conjunto limitado de encabezados HTTP estándar.
  4. **Sin soporte para la compresión de datos:** No se incluyeron mecanismos estándar para comprimir los datos transmitidos.
- **Limitaciones:**
  1. **Latencia alta:** Debido a las conexiones no persistentes y al tiempo de establecimiento de la conexión para cada solicitud, se experimentaba una alta latencia en la transferencia de recursos web.
  2. **Eficiencia baja:** La falta de soporte para la compresión de datos y la multiplexación de solicitudes resultaba en un uso ineficiente de los recursos de red.
  3. **Ineficiencia para la carga de páginas complejas:** Para las páginas web con muchos recursos (imágenes, scripts, etc.), el rendimiento era significativamente inferior debido al tiempo de establecimiento de la conexión para cada recurso.

HTTP/1.1:

- **Fecha de lanzamiento:** 1999.
- **Características:**
  1. **Conexiones persistentes por defecto:** HTTP/1.1 introdujo conexiones persistentes por defecto, lo que permitió reutilizar la misma conexión TCP para enviar múltiples solicitudes y recibir múltiples respuestas.
  2. **Soporte para encabezados de host y conexiones persistentes:** HTTP/1.1 introdujo encabezados de host, lo que permitió a los servidores alojar múltiples sitios web en una única dirección IP, y también permitió conexiones persistentes para reducir la latencia.
  3. **Pipelining:** HTTP/1.1 introdujo la posibilidad de enviar múltiples solicitudes a través de una única conexión TCP sin esperar las respuestas correspondientes, lo que mejoró la eficiencia de la transferencia de recursos.
  4. **Soporte para la compresión de datos:** HTTP/1.1 introdujo la compresión de datos, permitiendo comprimir los recursos antes de ser enviados al cliente para reducir el tiempo de transferencia.
- **Mejoras:**
  1. **Reducción de la latencia:** Las conexiones persistentes y el pipelining redujeron significativamente la latencia percibida por el usuario.
  2. **Eficiencia mejorada:** La compresión de datos y la reutilización de conexiones TCP mejoraron la eficiencia en el uso de los recursos de red.
  3. **Soporte para la carga de páginas complejas:** HTTP/1.1 mejoró el rendimiento para las páginas web con muchos recursos al reducir el tiempo de establecimiento de la conexión y la sobrecarga de transferencia de datos.

HTTP/2:

- **Fecha de lanzamiento:** 2015.
- **Características:**
  1. **Multiplexación de solicitudes:** HTTP/2 introdujo la multiplexación de solicitudes, lo que permite enviar múltiples solicitudes y recibir múltiples respuestas de forma simultánea a través de una única conexión TCP.
  2. **Compresión de cabeceras:** HTTP/2 incluye la compresión de cabeceras HTTP, lo que reduce la sobrecarga de datos en la transferencia de solicitudes y respuestas.
  3. **Priorización de flujo:** HTTP/2 permite priorizar el envío de recursos críticos antes que los recursos menos importantes, lo que mejora el rendimiento percibido por el usuario.
  4. **Server Push:** HTTP/2 introduce la capacidad para que el servidor envíe recursos adicionales al cliente sin una solicitud previa, lo que acelera la carga de páginas web complejas.
- **Mejoras:**
  1. **Rendimiento mejorado:** La multiplexación de solicitudes, la compresión de cabeceras y la priorización de flujo en HTTP/2 mejoran significativamente el rendimiento de las aplicaciones web.
  2. **Reducción del overhead de transferencia:** HTTP/2 reduce el overhead de transferencia de datos y mejora la eficiencia de la red al enviar y recibir múltiples recursos de forma simultánea a través de una única conexión TCP.
  3. **Soporte para páginas web complejas:** HTTP/2 facilita la carga de páginas web complejas al permitir el envío de recursos adicionales y priorizar la entrega de recursos críticos.

En resumen, cada versión del protocolo HTTP ha introducido mejoras significativas en términos de rendimiento, eficiencia y funcionalidad. HTTP/2 mejora la latencia y la eficiencia mediante conexiones persistentes y pipelining, mientras que HTTP/3 introduce características avanzadas como la multiplexación de solicitudes, la compresión de cabeceras y el server push para ofrecer un rendimiento aún mejor y soporte para páginas web más complejas.

¡Extraordinario!

You
explicame como si fuera un nene de cinco años qué motivó la utilización de UDP en vez de TCP en el marco del protocolo HTTP/3

ChatGPT
¡Claro! Imagina que estás jugando con tus amigos a pasar una pelota de un lado a otro. Cuando pasas la pelota, siempre quieres asegurarte de que llegue a tu amigo y viceversa, ¿verdad? Eso es lo que hace TCP en Internet. Se asegura de que cada pedazo de información (como una pelota) que envías a alguien llegue a su destino sin errores.

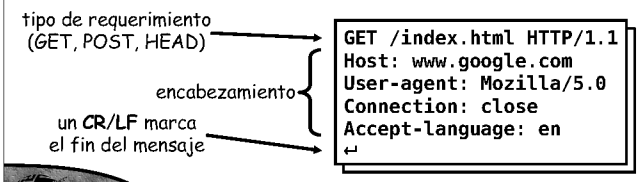
Pero a veces, jugar así puede ser un poco lento. Imagina que estás jugando a un juego de pelota con muchos amigos, y tienes que esperar a que cada uno pase la pelota antes de poder jugar. ¡Eso sería aburrido! Especialmente si quieres jugar súper rápido.

Entonces, a veces, cuando quieres enviar información realmente rápida, como en algunos juegos o cuando estás viendo videos en línea, prefieres usar algo llamado UDP. Es como si lanzaras la pelota a tus amigos sin preocuparte demasiado de si la reciben o no. Si se pierde una vez, ¡no importa! Seguirás lanzando más pelotas, ¡y algunas de ellas llegarán a tu amigo!

HTTP/3 usa UDP en lugar de TCP porque en algunos casos, como cuando estás viendo videos o jugando juegos en línea, quieres que la información se mueva rápidamente, ¡aunque algunas partes se pierdan en el camino! Es como si quisieras jugar a la pelota súper rápido sin preocuparte demasiado por cada lanzamiento perfecto.

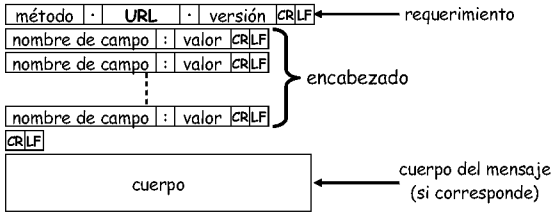
Mensajes HTTP

- **HTTP** contempla sólo dos tipo de mensajes:
  - Los requerimientos (**HTTP** request)
  - Las respuestas (**HTTP** response)
- Los mensajes eran 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**

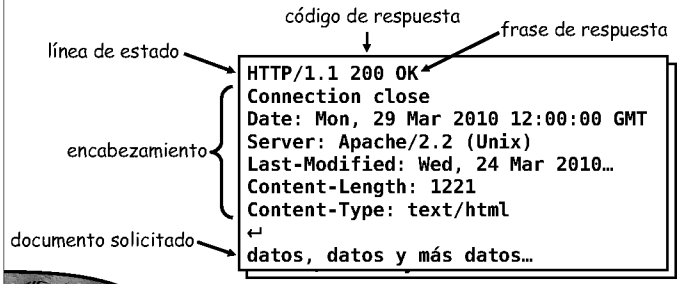
```
http://cs.uns.edu.ar/form.html?nombre=foo&apellido=bar
```

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

Estamos en sintonía...

You
dame ejemplos de las distintas categorías de códigos de respuesta HTTP

ChatGPT
¡Claro! Aquí tienes ejemplos de algunas categorías comunes de códigos de respuesta HTTP junto con sus significados:

1. **Códigos de Respuesta 1xx - Respuestas informativas:**
   - **100 Continue:** Indica que el servidor ha recibido la parte inicial de la solicitud y está esperando la solicitud completa antes de continuar.
   - **101 Switching Protocols:** Indica que el servidor está cambiando los protocolos de comunicación.
2. **Códigos de Respuesta 2xx - Respuestas exitosas:**
   - **200 OK:** Indica que la solicitud fue exitosa y se recibió, entendió y aceptó.
   - **201 Created:** Indica que la solicitud ha sido completada y se ha creado un nuevo recurso.
   - **204 No Content:** Indica que la solicitud fue exitosa, pero no hay contenido para devolver.
3. **Códigos de Respuesta 3xx - Redirecciones:**
   - **301 Moved Permanently:** Indica que el recurso solicitado ha sido movido permanentemente a una nueva URL.
   - **302 Found:** Indica que el recurso solicitado ha sido temporalmente movido a una nueva URL.
   - **304 Not Modified:** Indica que el recurso no ha sido modificado desde la última vez que fue solicitado.
4. **Códigos de Respuesta 4xx - Errores del cliente:**
   - **400 Bad Request:** Indica que la solicitud del cliente no pudo ser entendida por el servidor debido a una sintaxis incorrecta.
   - **401 Unauthorized:** Indica que se requiere autenticación para acceder al recurso solicitado.
   - **404 Not Found:** Indica que el servidor no pudo encontrar el recurso solicitado.
5. **Códigos de Respuesta 5xx - Errores del servidor:**
   - **500 Internal Server Error:** Indica que se produjo un error interno en el servidor y no pudo completar la solicitud.
   - **503 Service Unavailable:** Indica que el servidor no está disponible temporalmente para manejar la solicitud debido a una sobrecarga o mantenimiento.

Estos son solo algunos ejemplos de las distintas categorías de códigos de respuesta HTTP y sus significados. Cada código de respuesta proporciona información importante sobre el resultado de una solicitud HTTP y ayuda a diagnosticar problemas durante la comunicación cliente-servidor.

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
```

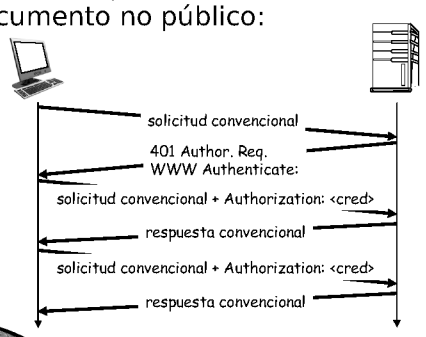
- Otra opción es directamente capturar los mensajes intercambiados usando Wireshark o alguna extensión a tal efecto del navegador

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:



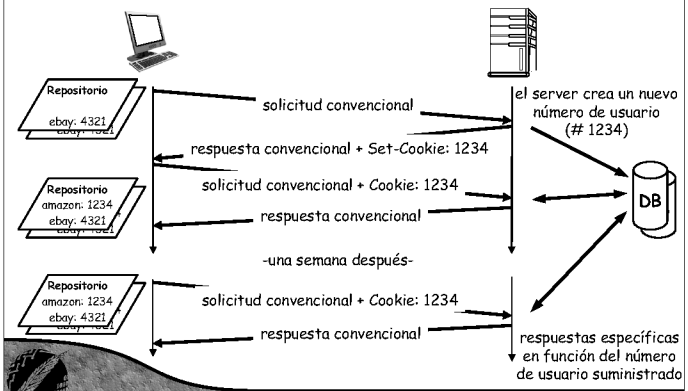
Cookies

- 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



Ventajas y desventajas

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

¡Salí de ahí maravilla!

You
cómo es que ganan plata los servicios gratuitos tipo Google, Instagram o TikTok?

ChatGPT
Los servicios gratuitos como Google, Instagram y TikTok ganan dinero principalmente a través de modelos de negocio basados en la publicidad y otras estrategias relacionadas con los datos de los usuarios. Aquí hay algunas formas en que estos servicios generan ingresos:

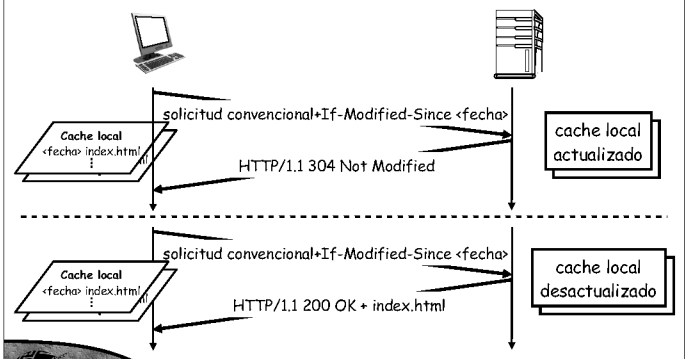
1. **Publicidad:** Estos servicios muestran anuncios publicitarios a los usuarios mientras navegan por la plataforma. Los anunciantes pagan a la plataforma por mostrar sus anuncios, y la plataforma utiliza datos de usuario para dirigir los anuncios de manera más efectiva a grupos específicos de personas que puedan estar interesadas en ellos. Google, por ejemplo, muestra anuncios tanto en su motor de búsqueda como en otros productos como YouTube y Gmail.
2. **Datos de usuario:** Estos servicios recopilan una gran cantidad de datos sobre sus usuarios, como sus intereses, comportamientos de navegación, interacciones sociales y más. Luego utilizan estos datos para personalizar la experiencia del usuario, mejorar sus productos y, lo más importante, vender acceso a estos datos a terceros, como anunciantes o empresas de investigación de mercado. Esto puede incluir información demográfica, preferencias de compra y patrones de comportamiento.
3. **Servicios premium o características adicionales:** Algunos de estos servicios ofrecen versiones premium o características adicionales por las que los usuarios deben pagar. Estas características pueden incluir almacenamiento adicional, acceso a contenido exclusivo, funciones de privacidad mejoradas u otras ventajas.
4. **Comisiones por transacciones:** En el caso de servicios como Instagram y TikTok, también pueden ganar dinero a través de acuerdos de colaboración con marcas o empresas. Por ejemplo, pueden recibir una comisión por cada venta que se genere a través de un enlace de afiliado o por la promoción de productos o servicios patrocinados en la plataforma.

En resumen, los servicios gratuitos como Google, Instagram y TikTok generan ingresos principalmente a través de la publicidad dirigida, la venta de datos de usuario, servicios premium y comisiones por transacciones. Estos modelos de negocio les permiten ofrecer sus servicios de forma gratuita para los usuarios mientras generan ingresos significativos.

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?
- <u>Beneficios:</u>
  - 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

Medio que se lavó las manos

You
vale la pena asignar recursos en una cierta institución para implementar un servidor proxy HTTP?

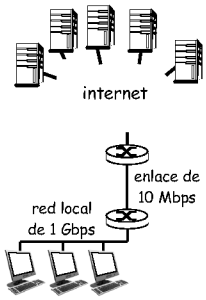
ChatGPT
La decisión de implementar un servidor proxy HTTP en una institución depende de varios factores y consideraciones específicas de la organización. Aquí hay algunos puntos a tener en cuenta al tomar esta decisión:

1. **Necesidades de la institución:** ¿La institución tiene necesidades específicas que podrían beneficiarse de un servidor proxy HTTP? Por ejemplo, si la institución tiene requisitos de seguridad estrictos o necesita administrar y monitorear el tráfico de red de manera más eficiente, un servidor proxy podría ser útil.
2. **Control y seguridad:** Un servidor proxy HTTP puede brindar a la institución un mayor control sobre el tráfico de red al permitir políticas de acceso y filtrado. Esto puede ser especialmente importante para proteger contra amenazas cibernéticas y garantizar la conformidad con las regulaciones de seguridad de datos.
3. **Optimización del rendimiento:** Dependiendo de cómo se configure, un servidor proxy HTTP puede ayudar a mejorar el rendimiento de la red al almacenar en caché recursos frecuentemente solicitados y reducir la carga en los servidores de origen. Esto puede ser beneficioso en entornos donde se requiere un acceso rápido a recursos web comunes.
4. **Monitoreo y registro:** Un servidor proxy HTTP puede proporcionar capacidades de monitoreo y registro detalladas, lo que permite a la institución rastrear y analizar el tráfico de red para detectar patrones de uso, identificar problemas de rendimiento o seguridad, y generar informes para fines de auditoría y cumplimiento.
5. **Costo y recursos:** Implementar y mantener un servidor proxy HTTP puede requerir recursos en términos de tiempo, personal y presupuesto. Es importante evaluar si la institución tiene la capacidad y los recursos necesarios para administrar adecuadamente el servidor proxy, incluida la configuración, el mantenimiento y la resolución de problemas.

En resumen, la decisión de implementar un servidor proxy HTTP en una institución depende de las necesidades específicas de la organización, así como de consideraciones relacionadas con la seguridad, el rendimiento, el control y los recursos disponibles. Si se implementa correctamente y se administra de manera eficiente, un servidor proxy HTTP puede proporcionar una serie de beneficios significativos para la institución. Sin embargo, es importante realizar un análisis exhaustivo y considerar todos los factores antes de tomar una decisión.

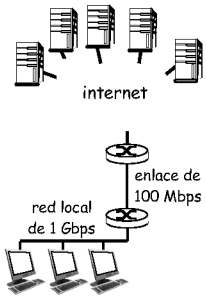
Métricas de desempeño

- 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: $\underbrace{2\text{ s}}_{\text{WAN}} + \underbrace{\text{minutos}}_{\text{enlace}} + \underbrace{\text{ms}}_{\text{LAN}}$



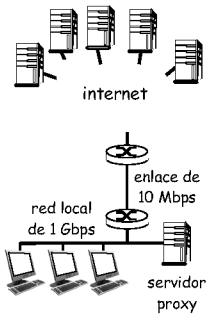
Métricas de desempeño

- 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?