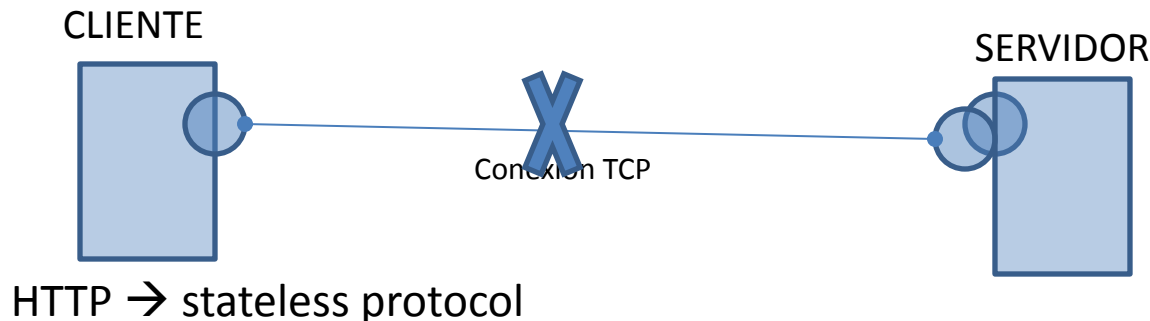
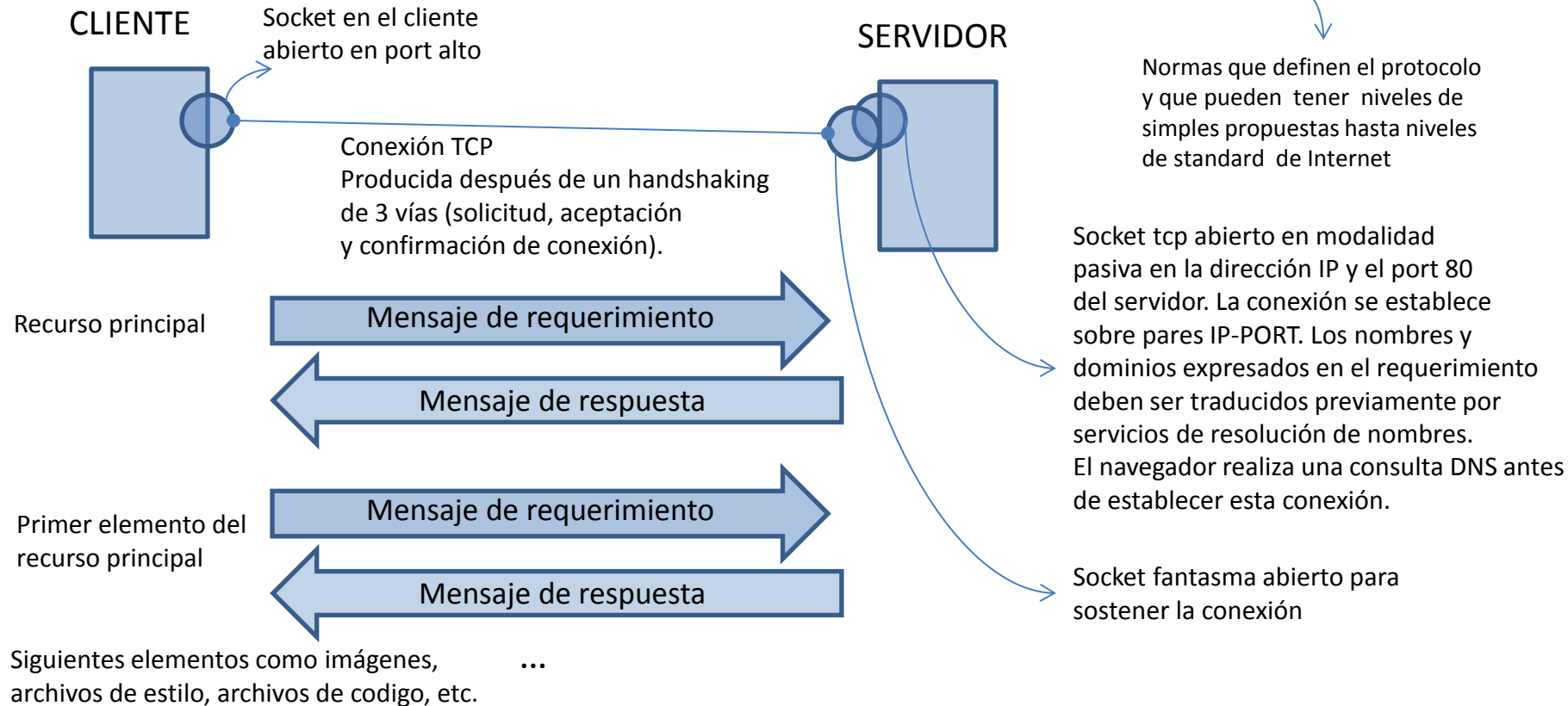


HTTP

HTTP Hipertext transport protocol

Version 1.1 RFC 2616



Estructura del mensaje HTTP

Válido tanto para mensajes de requerimiento como para mensajes de respuesta.

Aclaración (servidores RTSP): Los servidores de streaming de alta calidad tienen las siguientes ventajas sobre los basados en http:
El cliente puede avanzar o retroceder a cualquier punto fino del video ya que este es transmitido en forma continua y no por trozos.
Permite al server conocer exactamente lo que la gente consume.
Usa eficientemente el ancho de banda ya que solo transmite lo que el cliente pide.
El video nunca es almacenado del lado del cliente. La única desventaja es que por lo general estos protocolos son bloqueados por los firewalls corporativos. Por este motivo es poco común ver servidores RTSP en redes privadas corporativas. Por lo general se usan servicios de conexión para teleconferencia o de contenido en la nube (ej: netflix) y a los cuales se accede desde requerimientos salientes de las redes privadas.

<Linea inicial>

Líneas de encabezados:

Header1:valor1

Header2:valor2

Header3:valor3

Definidos en RFC 822
Todos terminan con CRLF
Ascii 13 y 10

<Linea en blanco (CRLF)>

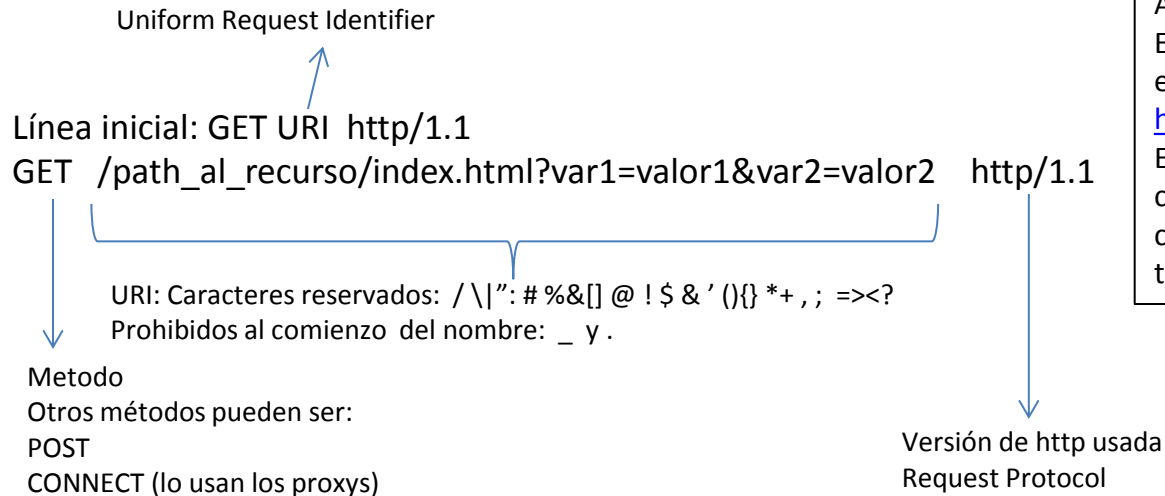
<BODY>

Puede llevar files o flujo de datos
Los files deben tener un header asociado que indique el tipo y otro que indique el tamaño.

El flujo (streaming http) no lleva header asociado (ej: video online). Se llama http live streaming (HLS).
El servidor divide el flujo de video que debe enviar en pequeños fragmentos que viajarán como archivos independientes y que irán siendo almacenados temporariamente en el navegador cliente.

Los datos transferidos pueden ser de tipo texto para el control en el navegador o de tipo binario para las muestras de video.
Los servidores de streaming de alta performance no usan protocolo http. (Usan por ej. RTSP real time streaming protocol y funcionan sobre UDP)

Mensaje de Requerimiento



Aclaración:

El campo URL del navegador es el que contiene el conjunto:

<http://nombreHost.dominioHost:puerto> + URI

Esta data la necesita el navegador primero para consultar la ip de destino al servidor de nombres configurado y luego para establecer la conexión tcp con dicho destino.

Headers:

Request method: GET

Request Protocol: http /1.1

User-Agent: Mozilla/5.0

Content-type: text/html (para datos enviados en el req.)

Content-length:xxxxx → logitud de los datos transmitidos hacia el servidor (variables o files)

Date:xxxxx

Accept image: /gif, application ms-word, application ms-powerpoint

Accept encoding: gzip

Accept language: es ar

Connection: Keep alive → Pide al servidor que no corte la conexión mientras sea posible

Host: nombrehost.mombredominio

Accept-encoding: gzip

Accept-language: es-ES, en-US

Un site para consultar que parametros son enviados por nuestro navegador puede ser: <http://request.urih.com/>

Body:

- Datos ingresados en campos de un formulario.
- Files enviados en un proceso de upload

Mensaje de Respuesta http

Línea inicial:

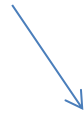
- Es una línea que indica el estado (status line)
- Informa acerca del estado del requerimiento

HTTP/1.1



versión

404



Código de estado
Entero de 3 dígitos

Not Found



Descripción de estado

Códigos:

- | | |
|-----|--|
| 1xx | Mensaje de información provisorio para procedimientos experimentales |
| 2xx | Recurso existente en el servidor (Ej: 200 OK) |
| 3xx | Redirección a otra URL o manejo de cache (Por ejemplo cuando indica al cliente que recupere de su propio cache.
Ej: 301 Recurso movido permanentemente a otra URL
Ej: 303 y 304 Recurso no modificado desde que fue cacheado por el navegador. |
| 4xx | Error en el requerimiento del cliente
Ej: 400 Bad request
Ej: 404 Recurso no encontrado |
| 5xx | Error en el procesamiento del requerimiento del lado del server (500 Server Error) |

Headers:

Los campos del Header indican entre otras cosas el tipo y el tamaño de los archivos y datos devueltos.

Server: Apache/1.2

Last-Modified: Fri, 3 Dec 2015

Content-type: text/html

Content-Lenght: xxx (en http 1.1 no es obligatorio. Si no está se trata de streaming)

Cache-control: no-cache → El servidor no adminte cache

Expires: 22 Jul 2016

Todos los navegadores tienen la opción de visualizar los headers que vienen
Con la respuesta http (ej: chrome → Descargar la extensión HTTP headers (Configuración – Mas herramientas → extensiones). Un nuevo icono se agregará al lado del menu de configuración de Chrome. Al entrar en cualquier página y clicar este ícono se podrán visualizar los headers http de la respuesta)

Body:

- En el Body de la respuesta va el recurso pedido en el requerimiento (Página html o archivo).
- Puede ir texto explicativo de algún estado de error.

Ejemplo completo: Caso de envío de formulario

POST /path/file.html HTTP/1.1

<headers>

User-Agent: Mozilla/3.0, Content-type: text/html

Content-length:xxxxx,Date:xxxxx, host: www.miempresa.com:8080

[CRLF]

<body>

Variables de formulario

Headers

HTTP/1.1 200 OK

Date: xxxxxxxx

Content-type: text/html

<CRLF>

Body

<html>

<body>

<h1>Texto</h1>

</body>

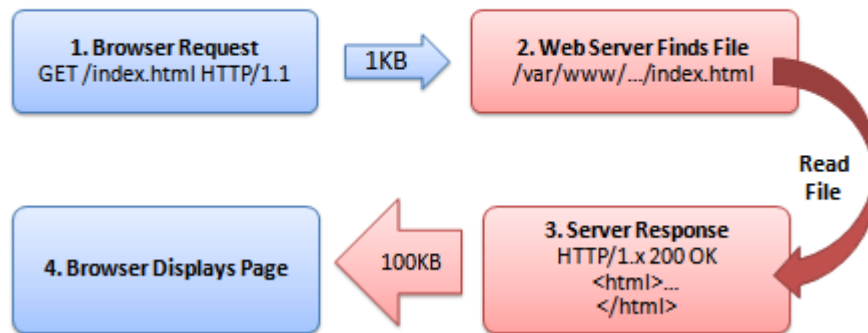
</html>

Caches:

Caso1: Sin cache

Requerimiento normal:
el navegador no almacena ningún
cache. La pagina es cargada totalmente
en cada requerimiento.

HTTP Request and Response



Caso2: Cache utilizado fechas

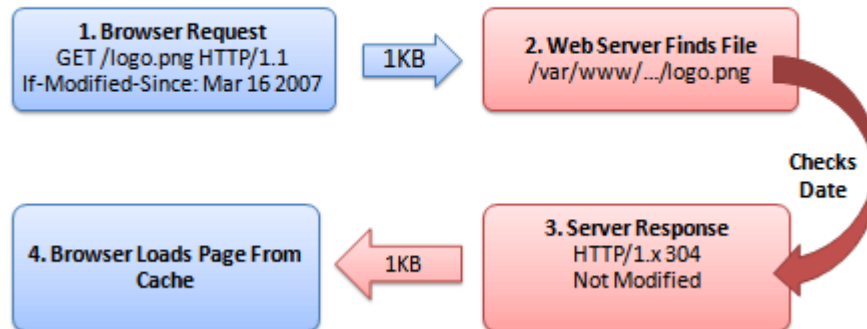
Requerimiento que envía un http header como el siguiente:

if-Modified-Since : xxxx (corresponde a ultima fecha de acceso al recurso)

El servidor busca el recurso y compara la fecha enviada en el header con la del mismo. Si es posterior entonces solo envia una respuesta con header «Modified = Not Modified» y el navegador carga el recurso de su cache.

Para que un navegador presente un header (if-Modified-Since) en el req. Este parámetro tiene que estar grabado en el cache. Es decir la asociación entre el recurso (url) y dicho dato de fecha. Tal como fuera recibido desde el servidor en el último acceso.

HTTP Cache: Last-Modified



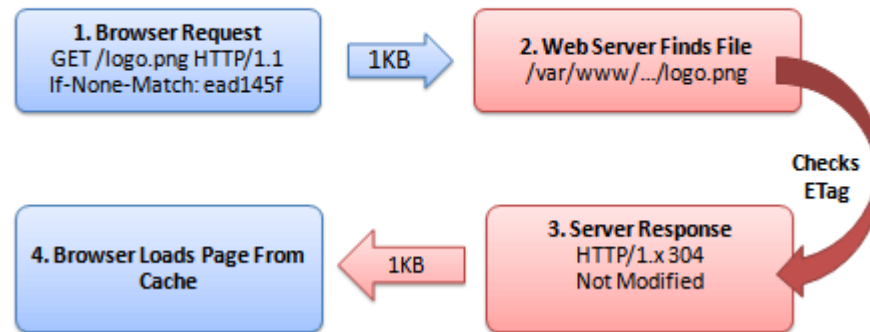
Caso3: Cache utilizando Etag's o hash's

Igual que el caso anterior pero utilizando un identificador (Etag) asociado a cada file del servidor. Este es una etiqueta o «finger print» que se modifica en el servidor cada vez que se modifica el recurso.

Lo que se compara entonces en el server es este header con el identificador del recurso. Si no fue modificado entonces solo devuelve una respuesta con un header «not Modified» Esto es mejor ya que no depende de errores en la configuración de fechas del servidor.

Recordemos que en la mayoría de los casos los desarrolladores no tienen control sobre el sistema operativo del servidor donde corre el web server. Nuevamente aclaro que para que el navegador envíe un header «If-None-Match» habría sido necesario que la asociación entre recurso (url) y esta variable, esté grabada. Esto se habría realizado en el último acceso al recurso mencionado. El Etag queda grabado en el navegador ya que es tambien recibido por este en la cabecera de la respuesta y es enviado por con el requerimiento http en el siguiente acceso al URL.

HTTP Cache: If-None-Match



Mejoras del HTTP 1.1 respecto al HTTP1.0

1. Múltiples transacciones sobre conexiones persistentes.
2. Soporte de cache en el envío de respuestas. Se evalúan encabezados de requerimiento para comparar con los atributos de cada recurso y de esa manera decidir si el mismo se envía completo por haber sido modificado.
3. Permite respuestas de flujo de datos sin encabezados que indiquen su longitud previamente.
4. Virtual hosting (múltiples dominios sobre una misma ip).
5. Línea inicial en el requerimiento:
GET /path/file.html HTTP/1.1
6. Nuevo campo de header obligatorio:
Host: hostDeDestino.DominioDeDestino (el host virtual de destino)
7. El cliente pide requests sobre conexión persistente hasta el último recurso:
GET /path/foto.gif HTTP/1.1
8. Connection: keep alive. (El requerimiento pide mantener la conexión mientras sea posible)
9. Connection:close. (El requerimiento lleva este header para pedir desconexión). Esto ocurre por ejemplo si el usuario cierra el navegador sin salir de la aplicación. El mismo navegador envía un nuevo requerimiento a todos los servidores con conexiones abiertas pidiendo a través de este header el cierre de la conexión al servidor remoto.
10. El servidor tiene capacidad para enviar un header de desconexión luego de un tiempo de inactividad.
Aclaración → Un time Out (tiempo es especificado en el server para desconectar si no recibe nuevo requerimiento por parte del cliente.
Existen requerimientos asincrónicos frente a un evento en el cliente (ajax) que también se producen sobre una conexión persistente.
11. El server puede terminar con la conexión (por ejemplo luego de un time out) con una respuesta conteniendo el siguiente campo header:
Connection: close
Significa que no aceptará más requerimientos.

Directorios Virtuales y Host virtuales

Directorios virtuales

URL → <http://www.servHostingXX.com.ar/~miempresa.midominio/mirecurso.html>

URI

- Un directorio virtual presenta los datos de un directorio real del disco del server por ejemplo → /home/miempresa
- No es necesario definir entradas de DNS en ningún servidor de nombres porque la IP es única y el nombre del servidor también lo es (por ej: telefonica.com.ar) en este caso particular.

Hosts virtuales

URL → <http://miempresa.midominio/mirecurso.html>

URI

- Los servidores WEB que corren http 1.1 pueden atender múltiples dominios en una única IP.
- En el caso de usar hosts virtuales habrá que declarar el nombre de host en un servidor de DNS público. Por lo general quien hostea un recurso es un ISP que tiene su propia zona de nombres de dominio y solo tiene que agregarlo en su base de datos.
- El servidor WEB debe configurarse especialmente para atender host virtuales.
- Para simular servidores de DNS en una PC hay que editar el archivo de texto: `\winnt\system32\drivers\etc\hosts`

Instalación del servidor apache sobre Linux Debian

1. Instalación de sistema operativo debian con Apache, smtp, exim
Esta instalación se debe hacer con conexión a Internet para la descarga de aplicaciones de algún servidor de paquetes de debian.
2. `apt-get update`
3. `apt-get install mc`
4. `/etc/network/interfaces` → Dejar ip asignada por server dhcp
5. `apt-get install ssh` (Instala componentes cliente y servidor)
6. `Apt-get install apache2`
7. `apt-get install mysql-server`
8. Crear cuentas de usuario para cada host virtual (`#useradd nombredeusuario -m`)
La opción `-m` crea la carpeta de trabajo del usuario en `/home`
9. Los usuarios que publicarán en el webserver tendrán sus carpetas reales en `/home/nombredeusuario`
10. Crear archivo de host virtual en carpeta `/etc/apache2/sites-available`. No borrar el site `00-default`.
11. Crear link a los archivos de host virtual en `/etc/apache2/sites-enable`
Pararse en `sites-enable` y escribir comando `ln`:
(`ln -s ../sites-available/nombrehost ./nombrehost`). NO borrar el site `00-default`
12. Restart del apache → `/etc/init.d/apache2 restart`

Instalación de server ssh en Windows (Freessh)

- Freessh es una aplicación para servicio de ssh en una maquina Windows.
- La configuración permite manejar la misma base de usuarios definidos para el sistema operativo.

Es necesario entonces definir tantas cuentas de equipo como host virtuales se hayan definido en la instalación de appserv.

- Cada cuenta de usuario generará un espacio de usuario ubicado en c:\Users. Si el sistema operativo esta instalado en español el verdadero nombre de la carpeta es Users pero se visualiza “Usuarios”.
- Será necesario iniciar sesión con cada usuario nuevo para que se cree el espacio de usuario y el escritorio correspondiente. Tambien debe crearse en cada espacio de usuario la carpeta www

Instalación del servidor appserv sobre Windows

AppServ contiene Apache, Php y Mysql

Como instalar host virtuales dentro de Appserv.

Instalar appserv. El directorio default de instalación es c:\appServ.

Editar archivo c:\appServ\apache2.2\conf\httpd.conf

Luego descomentar la línea:

Include conf/extra/httpd-vhosts.conf

Luego editar el archivo httpd-vhost y agregar lo siguiente:

```
<VirtualHost *:80>
```

```
ServerAdmin gwittbecker@google.com
```

```
DocumentRoot "C:/Users/grupo1/www"
```

```
ServerName grupo1
```

```
</VirtualHost>
```

Editar el c:\windows\system32\drivers\etc\hosts y agregar la línea

```
127.0.0.1 grupo1
```

Probar pingear contra grupo1

Crear carpeta C:/Users/grupo1/www

Crear archivo index.html con <h1>Prueba</h1>

restartear el apache

abrir desde el navegador el url: grupo1

Probar el script phpinfo.php agregandolo también a la carpeta www. Esto es para probar el php.

Instalación del cliente con Windows

1. Descargar e instalar winscp
2. Editar archivo /windows/system32/drivers/etc/hosts
3. Agregar las líneas correspondientes a cada host virtual, todas apuntando a la IP única del webserver

Pruebas del servidor

1. Desde el cliente ping contra la ip del web server
2. Desde el cliente ping contra el nombre de host virtual
3. Desde el server probar la apertura pasiva del servicio en el port 80:
netstat -a
4. Abrir la URL de algún host virtual:
5. [Http://mihostvirtual/paginaDePrueba.html](http://mihostvirtual/paginaDePrueba.html)

Crear usuario con su propia base de datos en phpMyAdmin:

(Si los scripts de phpMyAdmin estan en un host virtual del Apache, la url será de este tipo:
<http://nombreDelHostVirtual/phpMyAdmin>)

En el caso de nuestra práctica es: <http://usrBasesDeDatos> que ya tiene definido un index.html con un link a la aplicación.

- 1) Ingresar al phpMyAdmin con clave de root
- 2) Click en la opción de privilegios.
- 3) Agregar usuario nuevo
Nombre de usuario grupox
Host: localhost (Si se quiere que unicamente usuarios entren a través de phpMyAdmin.
Contraseña 2 veces: invitadox
Pulsar generar contraseña
Marcar boton de opción "Crear base de datos con el mismo nombre". Esto si se quiere que exista una base de datos con nombre grupox