*-Clase del 28-09-2021. “Capa de Aplicación”*

Vamos a ver un protocolo de capa de aplicación, en el libro de Tanenbaum habla de un montón de protocolos, en la materia se decidió ver uno por falta de tiempo.

Esto ya es en capa de aplicación. Con un programa que yo haga, puedo hacer un cliente o un servidor que cumpla el protocolo o no. Ósea podemos desarrollar el cliente y el servidor y que cumpla su propio protocolo.

Lo bueno de capa de aplicación, es que nosotros podríamos desarrollar solo un cliente y utilizar un servidor de un 3ro que cumpla el mismo estándar. Y se comunicarían bien. Nos ahorra de hacer los 2 extremos. Y Probamos nuestro proceso contra distintos fabricantes o programas que usen o hayan implementado el mismo protocolo o estándar.

Puntualmente vamos a ver HTTP, que usa el entorno de WWW.

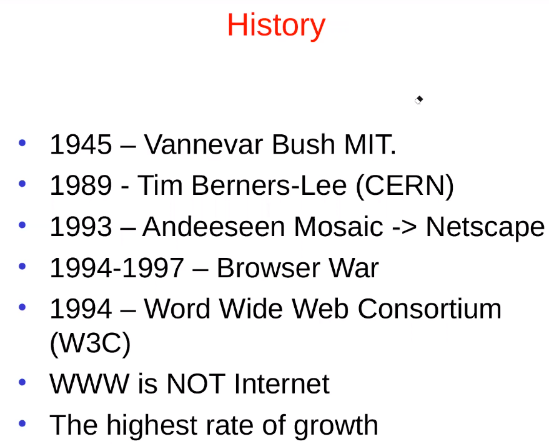
-Vamos a ver la arquitectura, como esta compuesto.

-Uno de los componentes son las páginas, vamos ver paginas estáticas y dinámicas. La diferencia y de que se trata.

-Después puntualmente vemos el protocolo.

-Los últimos 2 temas no los vemos.

Nos vamos a enfocar en ver el protocolo HTTP y como esta construido, cual es la arquitectura de www.



EL protocolo WWW o la web surgió en los 90s, pero su historia se remonta bastante antes, un científico del MIT en el 1945 escribió un libro que se refería a como vincular la información. Tenía un montón de bibliografía de distintos temas, donde era realmente era un caos buscar algo. Entonces ideo esta especie de, que hoy usamos, llamados hipervínculos. Que vinculan documentación de distintos sitios, de distintas fuentes, en distintos puntos.

Tim se topo con el mismo problema, y decidió crear un sistema para compartir la información entre los distintos integrantes del CERN que le llamo www.

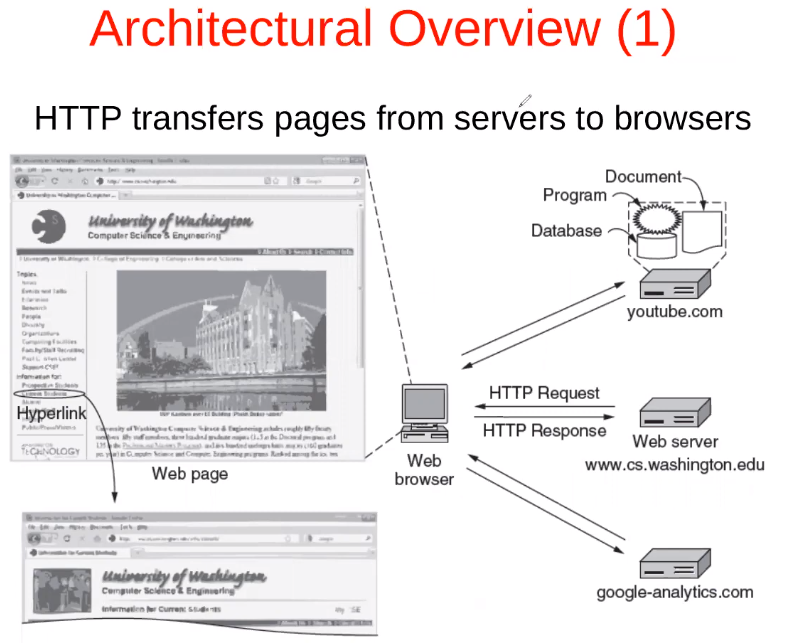
En el 1993 se creo el 1er navegador gráfico, antes eran todos navegador de caracteres.

1994 al 1997 Cada fabricante le agregaba funcionalidades nuevas a su servidor y a su cliente, que no estaban estandarizadas en ningún lado. Entonces muchas veces si se hacia un desarrollo web el problema que tenían era que no todos los navegadores podían ver la página bien, se veía mal o no se veía.

Como se estaba desmadrando el tema, se formo el W3C (World Wide Web Consortium) que fue el que estandarizo el protocolo HTTP y sus funcionalidades. Entonces de esta manera los fabricantes tanto de clientes como de servidores no podían agregar funcionalidades nuevas sino estaban estandarizadas en este consorcio.

Esta fue la Aplicación utiliza y seguirá utilizando el mayor porcentaje de trafico de internet, pero debería quedar claro que esta arquitectura www que incluye el protocolo, aplicaciones y algunas convenciones. NO ES INTERNET, es solo un protocolo o arquitectura que se utiliza sobre Internet.

Internet tiene www y otras cosas más. Hoy en día www es el protocolo de mayor uso, pero no significa que es el único.

Arquitectura:

Acá hay un ejemplo de cómo veríamos un navegador / browser o cliente, que es el proceso que corro en mi máquina. Hoy es gráfico, en algún momento no lo fueron.

El navegador tiene un montón de información, y cuando le hacemos click a un enlace o hyperlink nos muestra otra página de la universidad.

¿De dónde saca esta información el proceso cliente que está corriendo en nuestra maquina?

El cliente para obtener la página, en realidad no va solo al sitio o servidor de la universidad de Washington, sino en este caso, lo que hace es, que va a 3 servidores web, ósea hay 3 procesos que corren en distintos lugares.

Y lo que hace normalmente es un requerimiento de información (las flechas están al revés), y con estos 3 (que podrían ser muchos más), lo que hace es componer esta página.

Lo que nosotros vemos, que normalmente decimos pagina web, en realidad está compuesta por 1 o un montón de consultas. EJ: si utilizáramos el sniffer wireshark en el Diario “los Andes” veríamos que hace cientos de consultas.

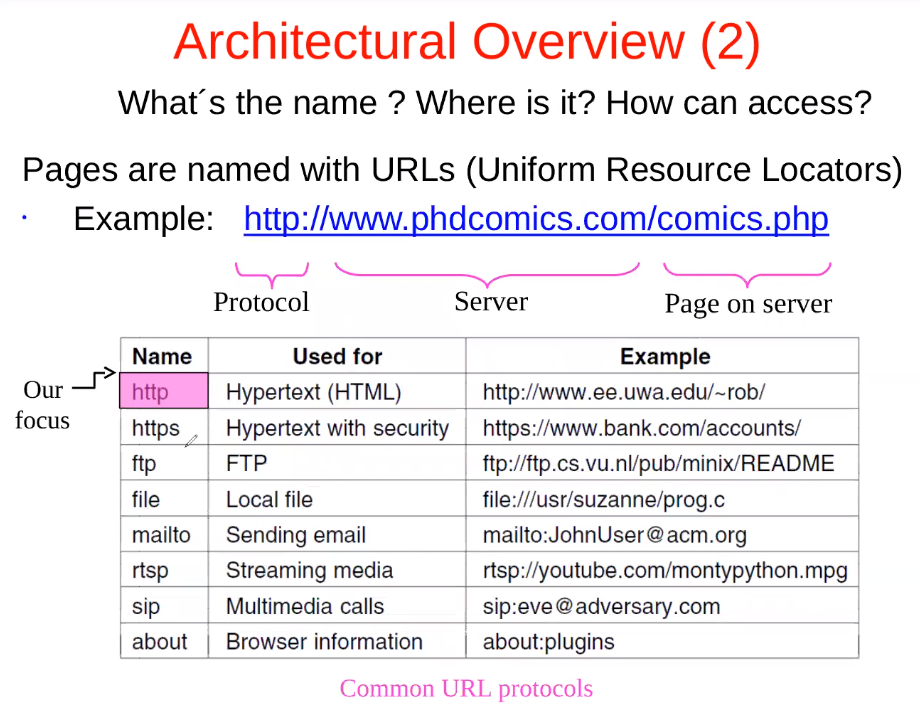
Puede traer texto, iconos, imágenes, videos, audios, etc. de diversos servidores o reportar o informar en Google-analytics.com, entre otras cosas.

Así es básicamente como trabaja eso.

¿Cómo sabe mi Navegador o browser donde tiene que ir a buscar las cosas que necesita para componer la página?

¿Sabe qué YouTube es un servidor, pero como sabe que video tiene que traer?

¿Cómo identifico unívocamente cada uno de los: recursos o de las cosas que me aparecen en la página?



Usa algo que se llama “Ubicadores de recursos uniforme” o “Localizador uniforme de recursos” más conocido como URL - Uniform Resource Locator.

Significa que, por cada recurso tengo una misma forma de describir los recursos que esta separada en varias partes.

Tengo un protocolo que lo utilizo para interactuar con el proceso servidor. Hay varios protocolos. No solo los de la figura.

Otro campo de servidor y otro campo de archivo dentro de ese servidor.

Entonces para acceder a un recurso primero tengo que saber con que protocolo acceder, a que servidor, obviamente en el ejemplo hay un nombre, pero hay que remplazarlo por una dirección IP, el protocolo DNS de capa de aplicación resuelve este tema. Y finalmente podría haber una, ninguna o varias barras “/” y un nombre. Esto da todo el Path o el camino hacia el archivo.

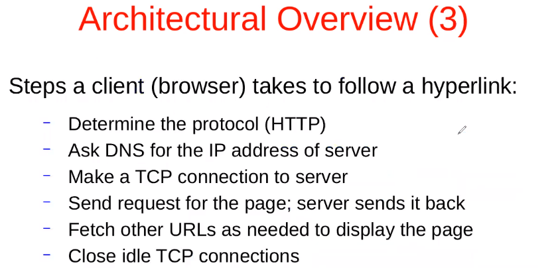
Siempre se pone: Protocolo://Nombre del servidor/ruta del archivo

O podría ser: “Protocolo:Usuario:Clave//Nombre del servidor/ruta del archivo” para enviar información de autenticación del usuario.

El palito de la Ñ, indica el home del usuario en Linux.

www no es obligatorio, muchos sitios lo colocan al principio del nombre para saber que el host dentro de por ejemplo phdcomics.com es un servidor web, pero no es una obligación es una convención nada más.

Ósea es una parte del nombre del servidor dentro de la URL, pero podría tener cualquier nombre.

¿Cuál es el procedimiento para que el cliente acceda a la información?

Lo primero que hace cuando colamos una URL y le damos Enter, es revisar el protocolo que pusimos a ver si lo entiende el cliente, porque si no lo conoce no va a saber como comunicarse con el servidor, una vez que determino el protocolo que básicamente es “parcear la URL desde que inicia hasta los primeros : ”, lo siguiente es parcear la parte del nombre del host del servidor web, lo que hace es una consulta a quien tenga configurado como servidor DNS para obtener su IP.

Una vez que sabe con qué protocolo conectarse, aca cuando hablamos de protocolo nos referimos también a los PUERTOS, ej si utilzamos HTTP se que el servidor espera conexiones en el puerto 80 si uso HTTPS en el puerto 443 y que el trafico de la información es cifrado, FTP puerto 21, etc.

Entonces ya tiene a que puerto conectarse y a que IP, lo que hace es el saludo de 3 vías de TCP porque www utiliza TCP para intercambiar información, ósea es orientado a conexión en capa de Transporte. Luego lo que hace es pedir un requerimiento y después que el servidor le responde, lo muestra en la página. Finalmente termina la conexión. Esto es para cada pagina que accedemos.

ACA HAY UN PROBLEMON

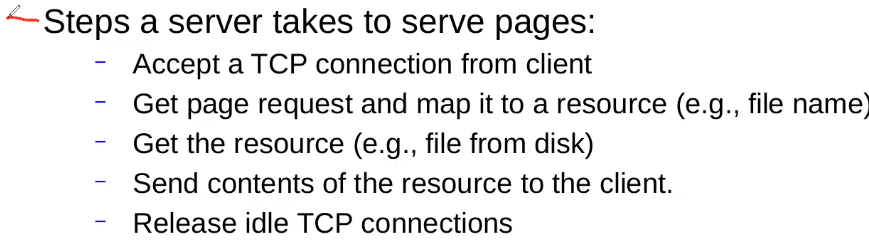
Si cada vez que me conecto per ejemplo, a Gmail, pongo mi usuario y mi clave y me muestra el correo, pero después corta. Pero sin embargo si hago clic en “enviados” o alguna opción …. No me vuelve a pedir la clave ni el usuario, pero es otra conexión … Ósea cuando hice click se conecto de nuevo, intercambio la información y se volvió a desconectar …. Con cada click

Ósea se la tuvieron que ingeniar de otra manera para que esto en capa de aplicación fuera orientado a conexión porque esto era sin conexión, ósea hacia el saludo de 3 vías, mandaba los datos, responde y cortaba, listo.

El problema: es que sino en cada pedido o Request debería estar mandando quien soy porque el servidor no sabe.

Se había diseñado para otra cosa este protocolo, y de repente lo empezaron a usar para sistemas interactivos o semejantes. El diseño original era para paginas estáticas.

¿Por otro lado que hace el Servidor, Cuales son los pasos del Proceso o Aplicación Servidor?

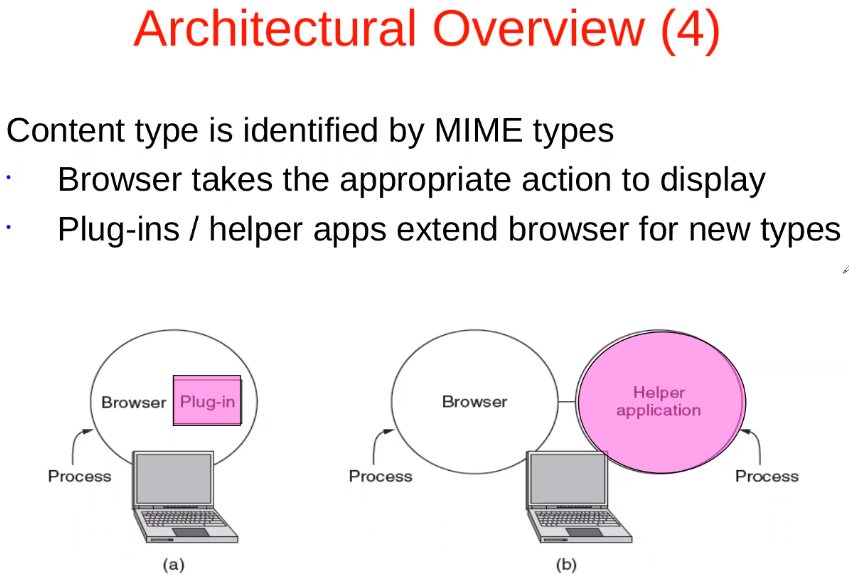


Está bloqueada esperando que alguien se conecte, bloqueada en la llamada al sistema Accept(), cada vez que viene un nuevo cliente lo saca del backlog.

Una vez que aceptó, puede leer lo que el cliente mando, entonces busca si existe o como puede encontrar el recurso que pide el cliente ya sea en un archivo o lo genera “al vuelo” dependiendo cual sea el recurso, una vez que lo tiene se lo envía al cliente y cierra la conexión.

La única salvedad es que el servidor debería poder soportar muchos clientes a la vez. Debería ser multiproceso o multi hilo.

Del lado del cliente hay un par de detalles.

Esto tiene que ver con que tipo de recurso devuelve el servidor al cliente.

Podría ser una imagen en formato .jpg o png u otro formato, un audio en mp3 u otro, un video en cualquier formato, etc.

Entonces el tema es ¿Cómo sabe el cliente como representar esa información?

Hay que darle alguna pista

El servidor cada vez que envía un recurso solicitado por el cliente tendría que especificar con algo llamado “MIME types” (Multipurpose Internet Mail Extensions o MIME; en español "extensiones multipropósito de correo de internet).

El tipo Extensiones multipropósito de Correo de Internet (MIME) es una forma estandarizada de indicar la naturaleza y el formato de un documento, archivo o conjunto de datos.

De alguna manera codifican los distintos tipos de archivo. En la respuesta del servidor, va ante que nada que tipo de archivo es y después el archivo, para que el cliente lo pueda procesar. El servidor si o si tiene que mandar un MIME type porque el cliente no puede adivinar, no es mago.

De esta manera lo que hace el cliente es INTERPRETA, pero el numero de posibles tipos de recursos o extensiones que debería poder soportar un navegador es enorme. Significa que debería poder interpretar todos esos tipos de recursos. Ósea como mostrar los videos, las imágenes, etc.

La cuestión es que la aplicación o el proceso del browser debería ser gigante porque debería tener un montón código para poder representar o procesar todos los tipos de recursos que me puede mandar el servidor.

Si ya de por sí, son enormes en cuanto al tamaño del binario, imaginemos nos si tuviera que soportar todas estas extensiones.

Entonces lo que se ha hecho, básicamente son 2 aproximaciones. O 2 maneras de encarar el tema.

Utilizando Plug-in o utilizando una Helper application.

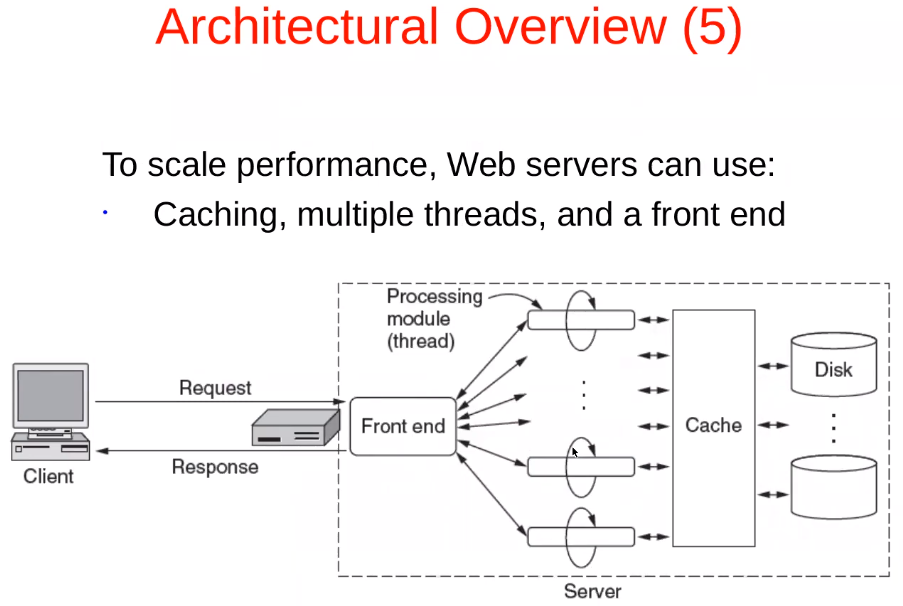
Y básicamente el comportamiento es así: el navegador mismo, si sale un nuevo formato de recurso ej: video o audio, podríamos agregarle a esa aplicación un Plug-in que entiende o interpreta como mostrar el video o recurso en cuestión.

La otra alternativa seria que el browser no tenga un Plug-in pero si cuando llega un recurso de un tipo determinado, lo que hace es lanzar una aplicación, otro proceso, que se encarga de presentar el recurso traído o que fue solicitado por el navegador.

Ej: el navegador me muestra un pdf en otra pestaña del navegador seria un Plug-in .. si abre otra aplicación para reproducirlo ejemplo libre office seria un helper application

Estas son las 2 maneras que tienen los navegadores para soportar continuamente cada ves mas tipos de recursos que el servidor pueda mandar.

Desde el lado del servidor.

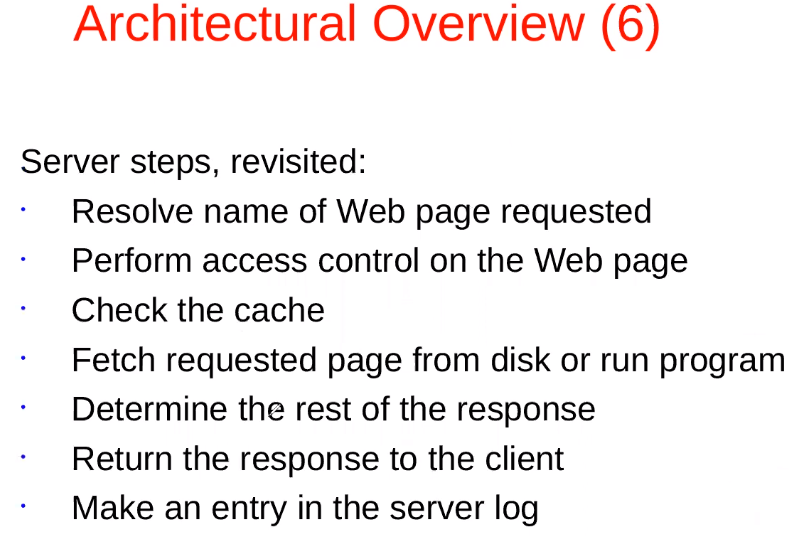
Lo que normalmente tenemos es la aplicación y para que escale o tenga una buena performance lo que normalmente lo que hacen es utilizar múltiples hilos o múltiples procesos.

En este dibujo hay una aplicación front end, que recibe los requerimientos y redirecciona a los hilos los pedidos, y cada pedido se implementa en un hilo, que busca el recurso solicitado por el cliente, el recurso puede ser que este en el disco, entonces lo que hace la mayoría de los servidores web además de implementar multi hilos o multiprocesos es cachearlo o tenerlo en la memoria RAM. Normalmente por un tiempo, porque es mucho más rápido que el hilo saque el dato de la RAM que lo vaya a buscar al disco.

También se puede colocar un balanceador de carga, que lo que hace es enviar los requerimientos en vez de a múltiples hilos a múltiples servidores. Obviamente la cuestión seria que todos los servidores deberían tener la misma información replicada.

Todos estos enfoques se siguen para el escalamiento de servidores WEB.

Revisando los pasos. El autor detalla un poco más lo que hace el servidor.

-Primero, resuelve el nombre.

-Realiza el control de acceso al sitio web , no lo mencionamos, pero los servidores pueden tener una autenticación.

-Verifica si lo tiene en cache, en RAM.

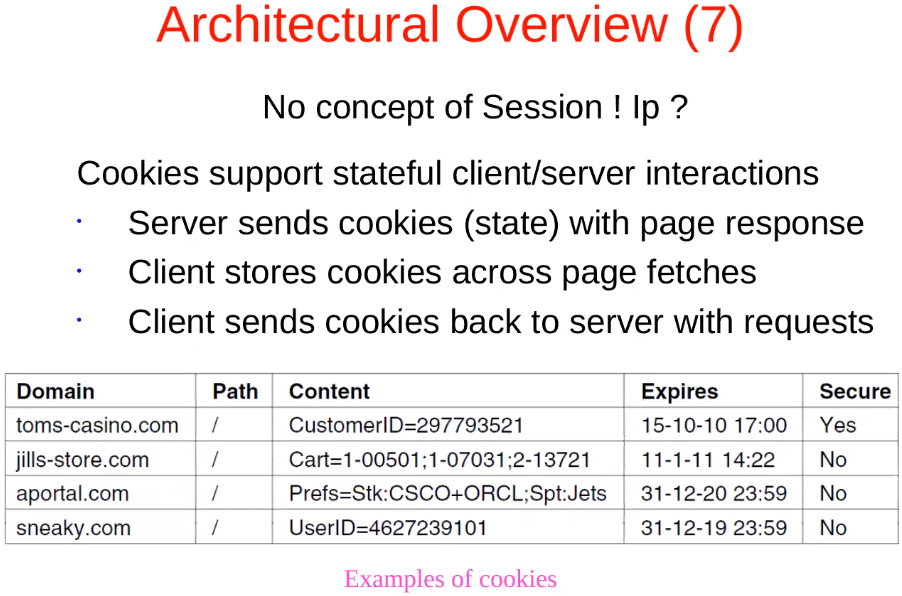
-Sino lo busca en el disco.

-Determina cual es la respuesta, porque podría pasar que no exista y la respuesta depende si lo encontró o no al recurso solicitado. (Famoso “ 404 Not Found ” no existe el recurso). Entonces dependiendo si encuentra el dato o no, determina como es el resto de la respuesta del servidor.

-Envía la respuesta al cliente.

-Finalmente, no lo habíamos mencionado, pero guarda un Log de eventos para realizar estadísticas varias. Ej: paginas mas visitadas, problemas, etc.

No son Stateful, no son con conexiones las comunicaciones del sistema web. WWW no tiene sesión.

Ósea cada vez que voy a pedir un requerimiento, me los responden y se corta la conexión. Y después quiero pedir un nuevo requerimiento por más que sea el mismo servidor, no sabe que soy el mismo usuario que pide más información.

Podría ser cualquier otro usuario que este en mi máquina, no tiene forma un servidor web de establecer una sesión o conexión, no es que se conecta y hasta que yo no me salga del servidor se mantiene la sesión.

Entonces HTTP, como vimos, tiene un saludo de 3 vías, se envía el requerimiento y la respuesta, se corta y a otra cosa.

Y si yo quiero otro sitio de la misma página, se vuelve hacer el saludo de 3 vías y repite todo lo mismo.

El tema no existe el concepto de sesión como cuando nos conectamos remotamente a algún lugar, me conecto y hasta que no termino de hacer todo o que tenga que hacer no me desconecto. Ej: protocolo SSH o algún escritorio remoto.

Me autentico, después interactuó todo el tiempo necesario, y cuando termino me desconecto. Pero durante todo el tiempo el servidor sabe quien soy yo, porque estoy utilizando el mismo socket. En este caso de HTTP, son todos sockets distintos, entonces no hay sesión. Por lo tanto, no tengo forma de saber que una nueva conexión pertenece al mismo proceso o cliente que ya me había enviado sus datos de usuario y clave para autenticación.

¿Entonces como hago?

No tengo forma de saberlo, a no ser que

Cookies: La única manera que tiene el proceso servidor de saber que una nueva conexión es del mismo cliente que se había conectado antes y ya había autenticado su usuario es, cuando el cliente se autentico, el servidor le responde ok, pero en el encabezado de respuesta envía una información adicional. (metadatos creo).

Entonces cada vez que el cliente realiza algún nuevo requerimiento al servidor y realiza el saludo de 3 vías a ese sitio, en el encabezado del pedido le manda esa información adicional, esa cookie, ese dato.

Entonces el proceso servidor lo que hace cuando lo recibe, es identificar que era un cliente que ya le había pedido recursos. Ej: compras del carrito en una pagina web, apago y prendo el pc y siguen los elementos de carro igual. (Porque ni siquiera podría tener la misma IP, los proveedores las cambian)

La única forma que tiene el servidor web, es que el cliente le mande información adicional, que no tiene nada que ver con el protocolo, sino con la aplicación.

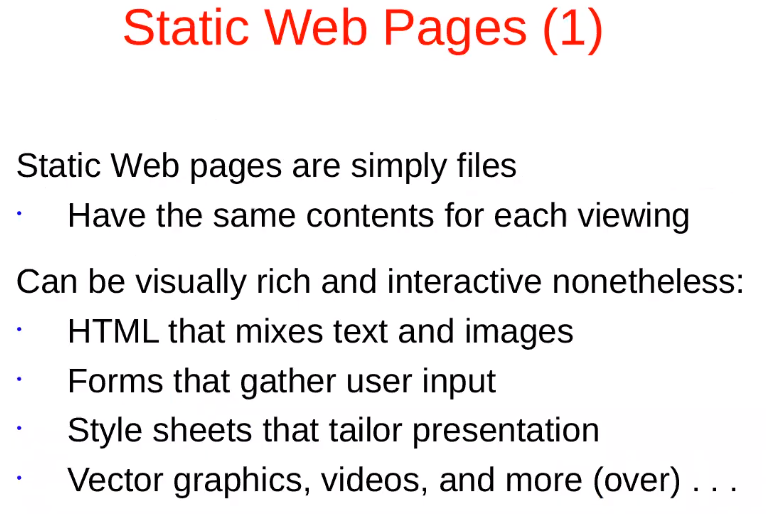
Aclarar si en el saludo de 3 vías el servidor también envía meta datos

Ver ejemplos de la filmina, normalmente las cookies tienen fecha de vencimiento, puede durar distintos periodos de tiempo, la sesión que este abierto el navegador, etc.

Algunos sitios o servidores no funcionan si no se aceptan las cookies.

Lo que hace el navegador, es decir, che este servidor me esta mandando una cookie y nos pide permiso al usuario para almacenarla localmente en el disco. Normalmente no son ejecutables, pero si es un servidor malicioso podría ser una vulnerabilidad.

**PAGINAS:**

Hasta acá, contando un poco lo que es arquitectura… ahora vamos hablar del otro componente que conforma www, que son los recursos o llamadas a simple vista páginas.

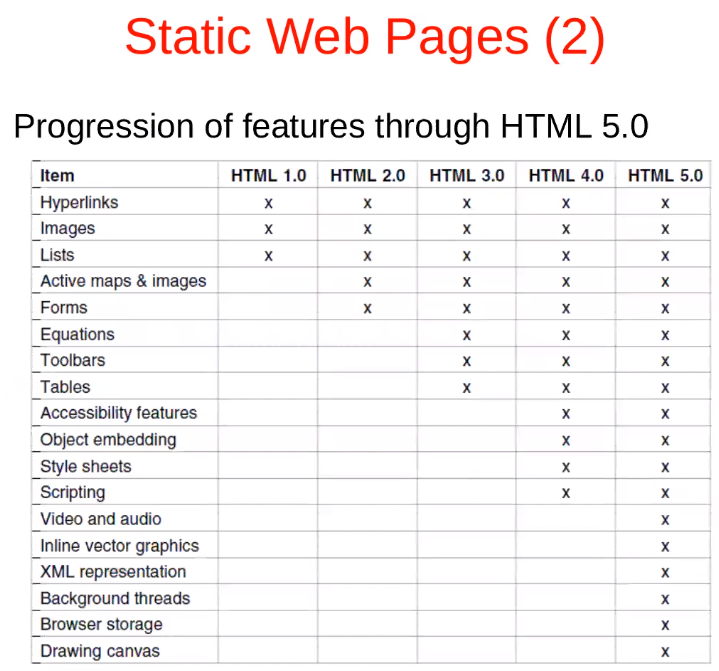
Podríamos tener 2 tipos de recursos, paginas estáticas y dinámicas.

Las páginas estáticas son páginas que están escritas en un archivo local y directamente cada vez que un cliente pide esa pagina el server la busca o si la tiene cacheada la muestra directamente, pero siempre es la misma, no cambia.

Este fue el origen del sistema www, para paginas estáticas, poner información que este disponible para todos y una forma rápida de vincular distintos documentos en distintos sitios, pero siempre estáticos. Si necesito que cambie hay que ir a la maquina donde esta corriendo el proceso servidor, ir al recurso o archivo en el directorio del servidor y editarlo. No hay otra forma.

Estos archivos se escriben normalmente en HTML, que es un lenguaje de maquetado, con algunas etiquetas que le permite dar formato, insertar cosas y lo mas IMPORTANTE le permite poner HIPERVINCULOS. Que seria poner dentro de una pagina HTML una URL que cuando el usuario le haga clic permite hacer un requerimiento a ese nuevo recurso.

Las primeras paginas web solo tenían este TAG o hipervínculo para redireccionar a otro lugar, con esto ya podían vincular la información entre distintos archivos de distintos servidores web

Hay un estándar para escribirlas, pero siempre son estáticas.

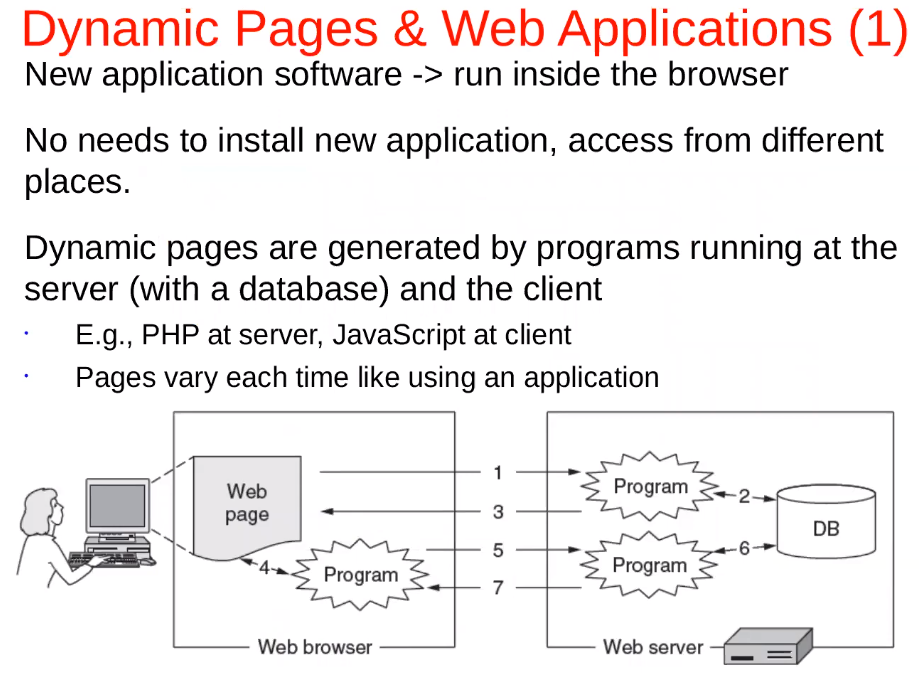
Obviamente cada vez se están usando menos las páginas estáticas.

Hoy lo que ha avanzado mucho, es poner aplicaciones que utilizan www como plataforma.

¿A qué se refiere?

Porque esto les permite a las aplicaciones servidoras independizarse del lado del cliente, ósea podría modificar la aplicación que estamos poniendo sobre www de manera que los clientes no tienen que actualizar una aplicación cada vez que se realiza un cambio, no tiene que descarar nada, no hay que verificar nada, todos tienen la misma versión, etc.

Directamente cambio la aplicación, sus funcionalidades o lo que sea, y ya con los navegadores acceden a la nueva versión. Entonces es mucho más cómodo el deploy de las aplicaciones. No hay que hacer nada del lado del cliente. (salvo actualizar el navegador si es muy viejo, pero no es por la app)

Ahora como funcionan estas aplicaciones que no son estáticas.

Yo tengo nuevas versiones de aplicaciones y todas las hago correr dentro del navegador. Entonces no necesito que del lado del cliente actualicen nada, se actualiza del lado del servidor y automáticamente todos los clientes a la vez disponen de la nueva versión.

Esto esta buenísimo, para eso es para lo que mas e utilizan las paginas dinámicas.

FUNCIONAMIENTO:

Básicamente en el funcionamiento, se pueden correr aplicaciones tanto del lado del cliente como del servidor, va a depender de cómo se desarrollen.

Pero lo que se hace es, tenemos un cliente y un servidor, y hacemos un requerimiento. Y en vez de ir a buscar un archivo y transferirlo, lo que hace es correr una aplicación y el resultado de esa aplicación es lo que manda de respuesta el servidor al cliente.

Y en ese resultado o respuesta el servidor también podría enviar una aplicación que se lanza o ejecuta en el cliente. La cual a su vez podría enviar nuevos datos a la aplicación que se ejecuta del lado del servidor. Y esta última podría responder información.

Observar los números en la imagen.

Ejemplo de caso de uso:

Estamos jugando un juego del Facebook en la computadora, en el navegador, ponele el candy crush …. Si se corta internet, podeos seguir jugando … recién nos damos cuenta cuando pasas de nivel y debería haber un intercambio de información que se almacena en el servidor. Esto es así porque en realidad yo tengo una aplicación que esta corriendo embebida dentro del navegador.

Cuando en algún momento yo solicite el jueguito desde el navegador, el servidor no me transfirió una página estática, me transfirió un programa que se va a ejecutar en mi máquina. Y una ves que esta corriendo, dependiendo de lo que haga el programa, de vez en cuando va pedir información o interactuar con el servidor, por ejemplo, cuando paso de nivel. Mientras no pase de nivel, toda la información la tengo local y puedo seguir jugando.

Hay muchas tecnologías para llevar a cabo estos programas: algunas conocidas,

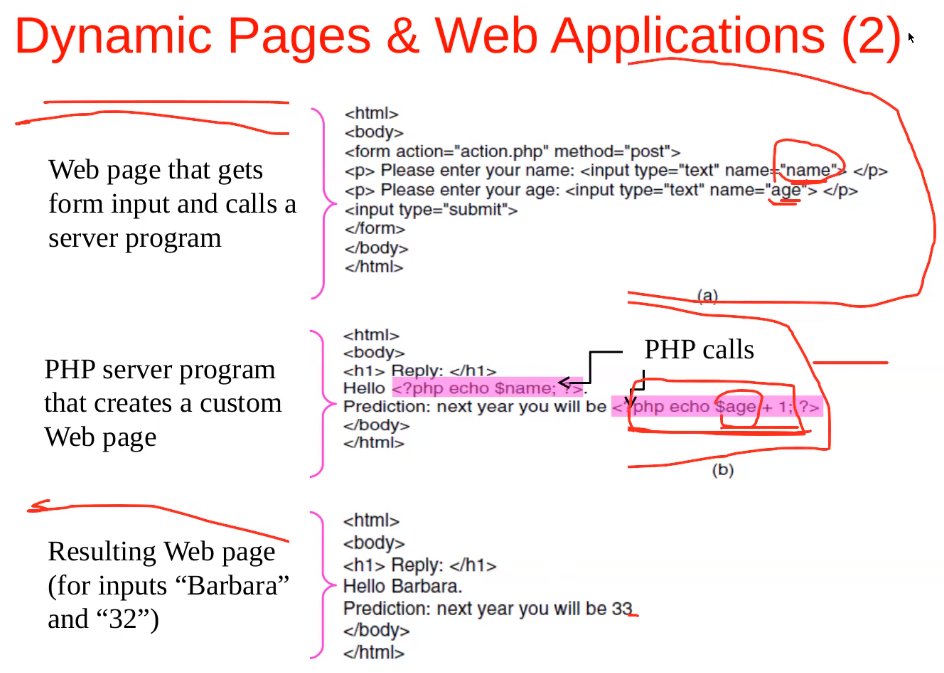
Lenguaje php, se corre del lado del servidor, es interpretado. Y cuando el Cliente manda un formulario con algunos datos, la aplicación del lado del servidor podría acceder a una base de datos con información del usuario y generar al vuelo, un archivo html de respuesta (que se trasfiere en 3), el cliente no se da cuenta que el archivo u aplicación que le están enviando se genero por un proceso o se leyó desde un archivo en el disco o la RAM.

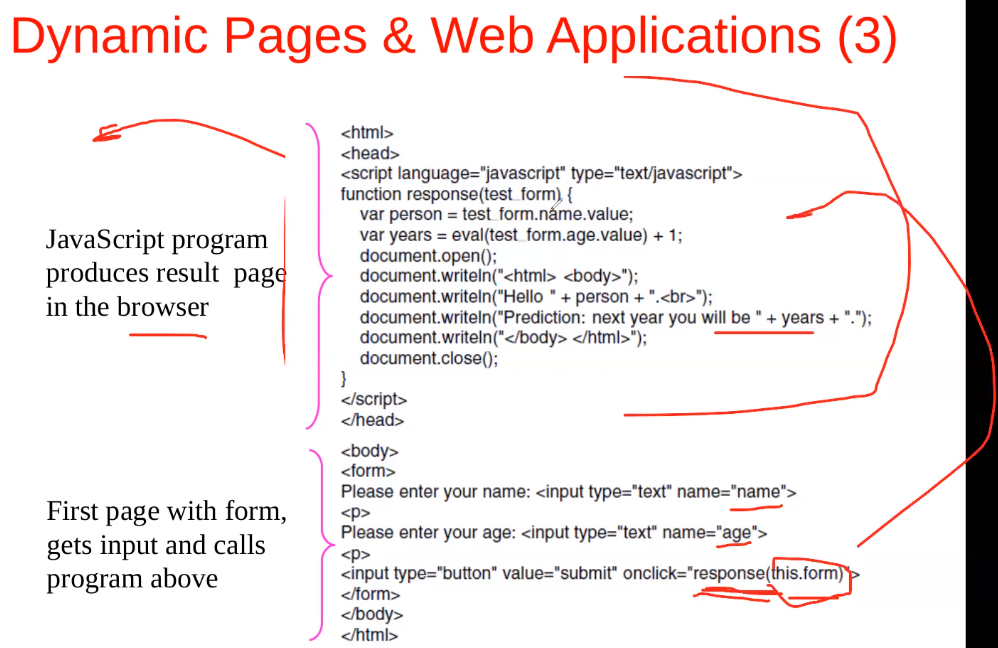
Porque la aplicación servidora en php lo que hace es generar código en html, en función de los datos que recibe y accediendo a una base de datos y le envía las respuestas al cliente.

Otra alternativa es JavaScrip. Es similar. Cuando se manda el formulario, el sever responde con código html mas unos tags que son de JS, y lo que hace el navegador en el cliente es ejecutar el código embebido de JS. Lo corre y es lo que interactúa con el usuario.

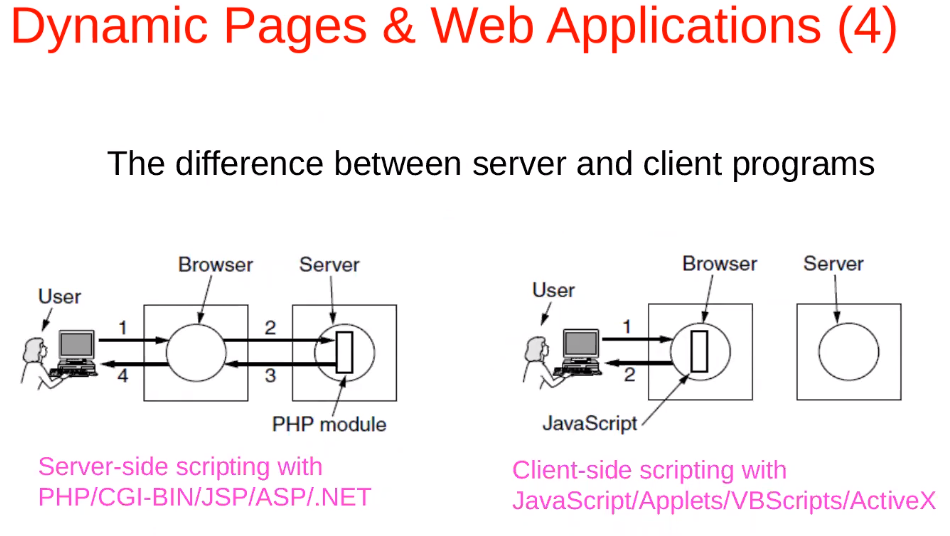
Se logran paginas dinámicas del lado del servidor con PHP y del lado del cliente con JavaScript

Ejemplos:

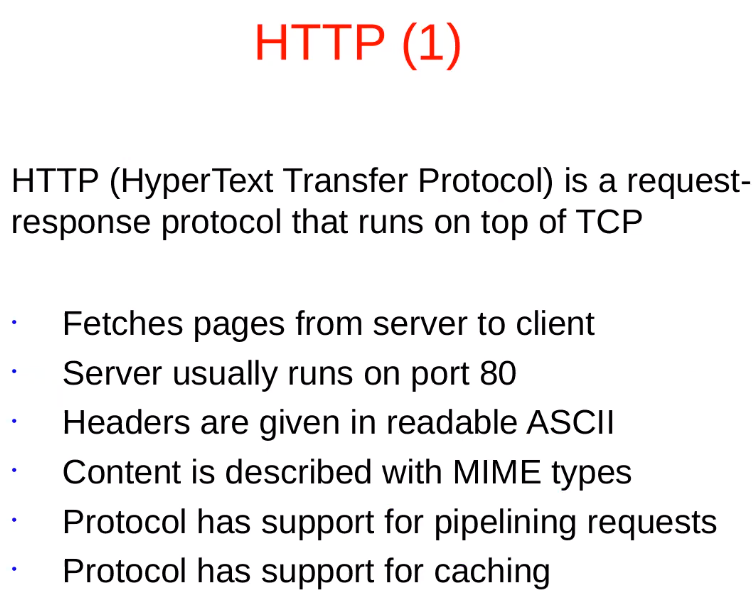




Los jueguitos usan JS para ejecutarse del lado del cliente. Así no les afecta la velocidad de la conexión de internet.



Los server también deberían tener una especie de plug in para que entiendan si tengo un tag que dice php u otro lenguaje.

Ahora vemos en si el protocolo HTTP, de que se trata, como es.

Hyper Text Transfer Protocol: es sencillo, justamente transfiere archivos de hipertexto, de html. Esto er originalmente, después se empezó complicar con otro tipo de archivos.

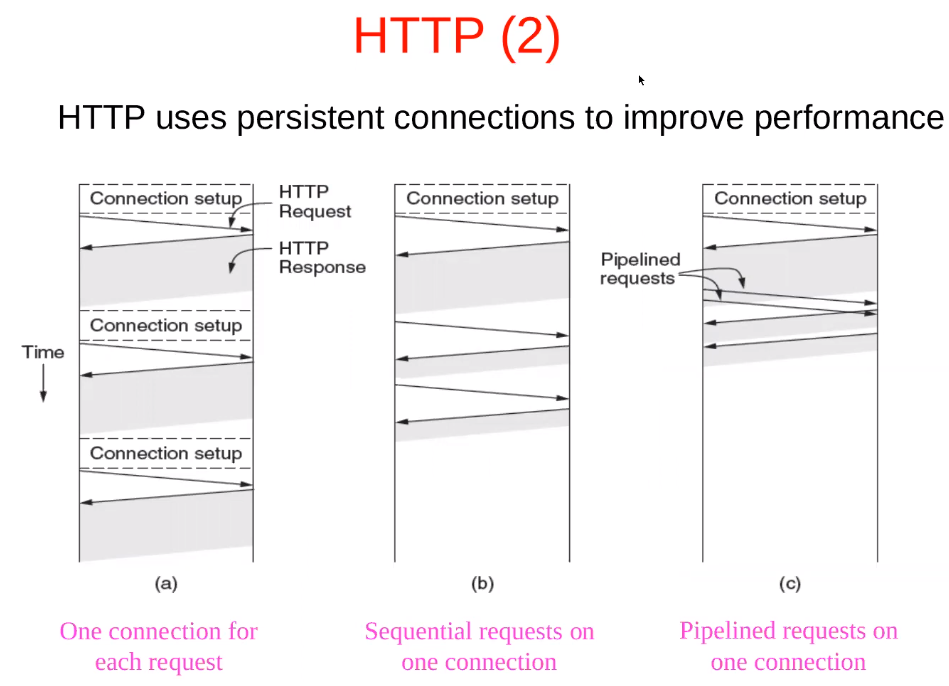
Básicamente es un protocolo de requerimiento/repuesta, pido un archivo y me los responde y corta la conexión.

Normalmente utiliza el puerto 80 del lado del server.

El encabezado va en ASCII

Dependiendo del tipo de archivo, tenemos que agregar un tipo de dato precediendo al archivo en si, para que el navegador sepa que hacer cuando recibe ese archivo. Si lanza un plug-in u otra aplicación. Etc etc

Tiene soporte de pipeline y de cache este protocolo. Se refiere a que se puede almacenar la respuesta para un futuro uso. Como vimos en el servidor, si una pagina estática la piden mucho, y si la tiene en RAM es mucho más rápido que buscarla en el disco.



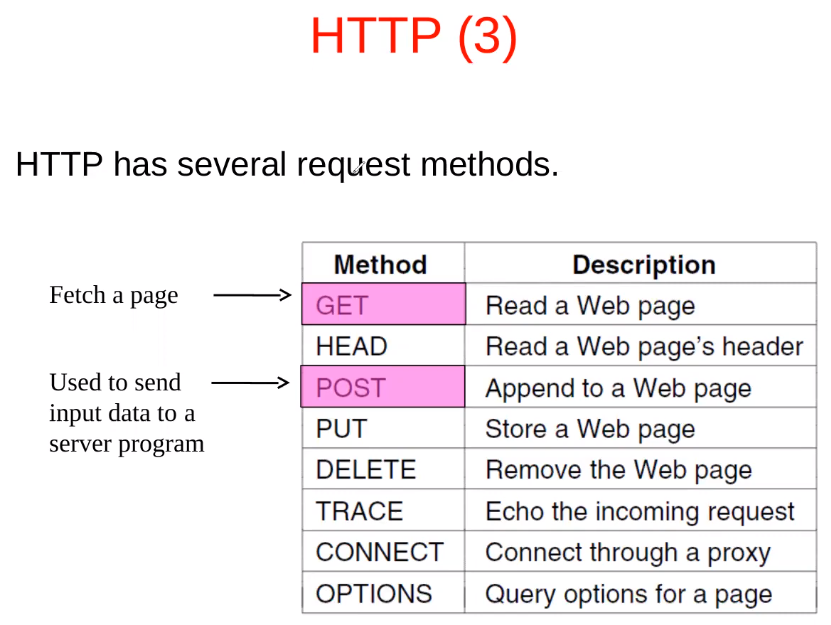
Normalmente HTTP no es persistente, esto significa que se hace el saludo de 3 vías, se envían requerimiento y respuesta con todo el archivo que se transfiere y luego se corta la conexión.

Si quiero algo nuevo, se repite el proceso.

Hay una alternativa, de dejar la conexión abierta, pero es para HTTP 1.1 o 2, donde no corta la conexión, pero periódicamente tengo que estar mandando avisos al servidor que no corte la conexión, porque sino la corta. Ósea queda un tiempo abierta y si no hay tráfico, la corta.

Pipelined request se refiere a que se puede pedir mas de una cosa a la vez. Ej: puedo hacer nuevos pedidos antes que lleguen las repuesta de request anteriores. Esto normalmente se puede observar en paginas que tengan muchos recursos donde se observan que se cargan en simultaneo, no de a uno por vez. … En paralelo se piden varios requerimientos.

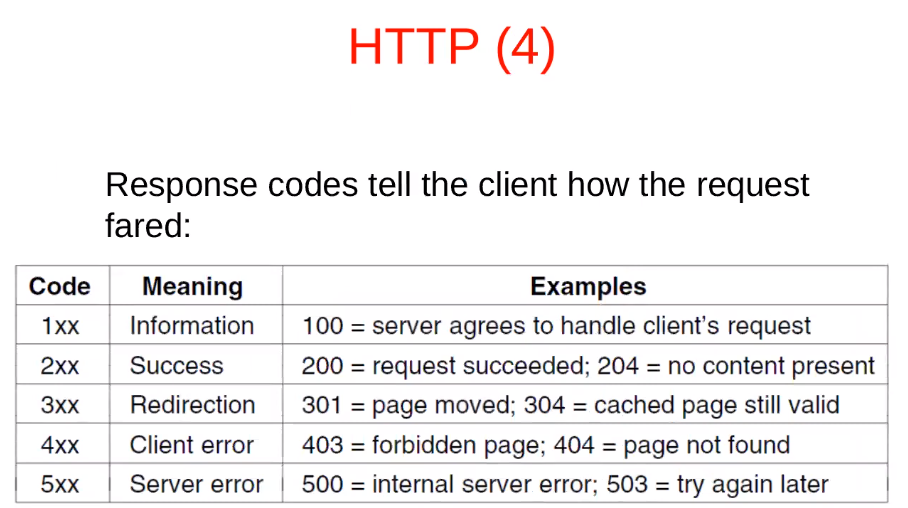
HTTP permite realizar mas de un pedido en simultaneo de forma paralela.

¿Cómo pido las cosas? ¿Cómo son los pedidos?

Básicamente hay métodos de requerimiento y respuesta. Los mas comunes son GET o Post.

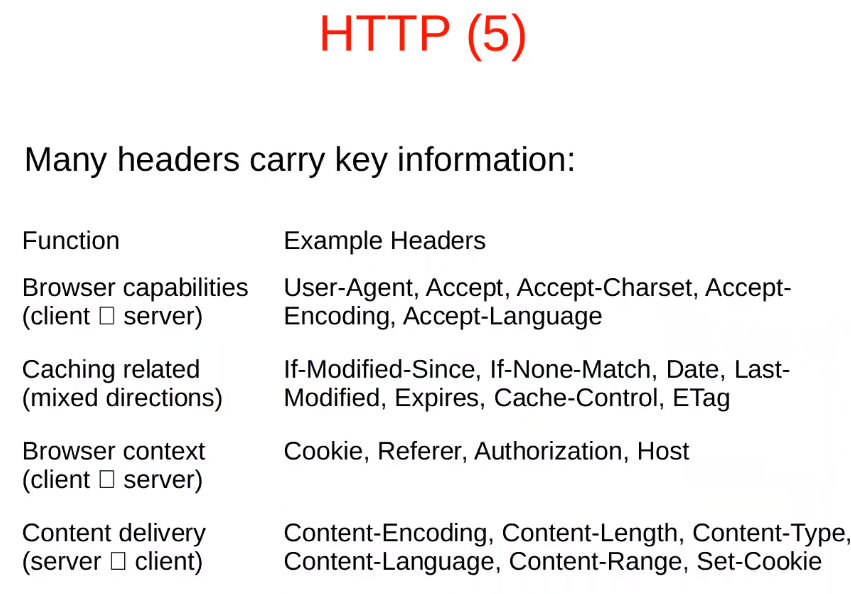
Donde se coloca Get + el archivo solicitado al servidor, y habrá un método de respuesta en el server que me dirá si lo encontró o no. Y el archivo anexo si fue encontrado.

El encabezado de los requerimientos comienza con alguno de estos métodos, luego el archivo o recurso que estoy solicitando, después la versión de HTML y algunos encabezados mas que el cliente puede mandar.



En cuanto a la respuesta enviada por el server, lo primero que viene es el “código de respuesta”, similares a los del correo.

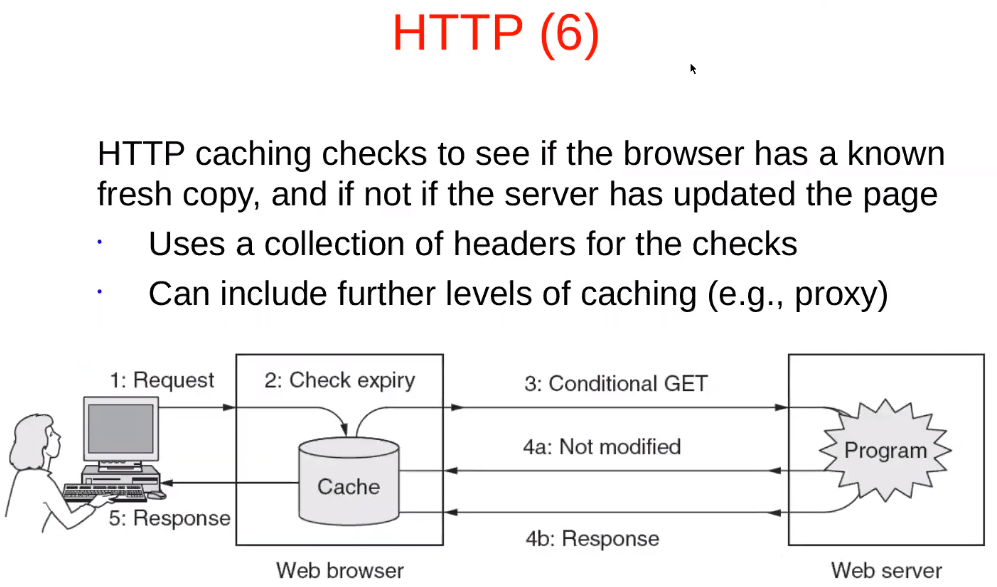
La respuesta vuelve con un código de respuesta, una descripción de respuesta y algunos encabezados más.

Ejemplos de algunos encabezados

Cada ves que el cliente realiza un requerimiento al servidor, en el encabezado puede indicar que capacidades tiene el navegador (, que navegadores es, set de caracteres, codificación, lenguaje que soporta, etc.)

Respecto al cache, es información que el cliente le pide al servidor para saber si tiene que actualizar, porque este expiro o fue modificado, un recurso que ya había solicitado antes. Porque si lo tiene cacheado en la memoria el cliente, y está vigente, lo muestra localmente. Es como un Get condicional, el cliente le dice al server que si se modificó o expiro lo mande de nuevo, sino confirmarme que está bien y muestro la capia que ya tengo.

Distribución de contenido: es lo que responde el servidor para indicar las características del recurso u archivo que esta enviando al cliente.



Respecto a como agilizar el sistema web: es haciendo cacheo.

Hay un par de aproximaciones o maneras de hacerlo.

Cada vez que el cliente va a solicitar una página, normalmente el navegador tiene una cache.

La aplicación antes de enviar el requerimiento al servidor web, verifica si esta almacenado en la cache y se fija en el encabezado del archivo si expiro. Porque cada ves que el servidor responde, generalmente en la respuesta viene incluida una fecha expiración de la página.

U otra alternativa es el GET condicional. Donde indica y consulta si fue modificado desde la fecha que lo tiene almacenado en la cache local.

4a vs 4b: Me ahorro la transferencia de todo el archivo, si este esta igual no es necesario.