Capa de Aplicación – Resumen

PDU ⇨ Mensaje

La capa de aplicación es la capa superior en la estructura de la red y es donde se encuentran las aplicaciones y sus protocolos. En Internet, esta capa abarca protocolos como HTTP, que se utiliza para solicitar documentos web, SMTP, que se utiliza para el correo electrónico, y FTP, que se utiliza para la transferencia de archivos. Además, en esta capa se lleva a cabo la conversión de nombres comprensibles por humanos a direcciones IP de 32 bits a través del Sistema de Nombres de Dominio (DNS), que es un protocolo de capa de aplicación.

Un protocolo de la capa de aplicación define cómo los procesos de una aplicación, que se ejecutan en distintos sistemas terminales, se pasan los mensajes entre sí. En particular, un protocolo de la capa de aplicación define:

- Los tipos de mensajes intercambiados; por ejemplo, mensajes de solicitud y mensajes de respuesta.
- La sintaxis de los diversos tipos de mensajes, es decir, los campos de los que consta el mensaje y cómo se delimitan esos campos.
- La semántica de los campos, es decir, el significado de la información contenida en los campos.
- Las reglas para determinar cuándo y cómo un proceso envía mensajes y responde a los mismos.

Funciones

- Define el formato de los mensajes: Existen protocolos que trabajan de forma binaria, por ejemplo usando ASN y otros en forma textual ASCII como HTTP.

- Define la semántica de cada uno de los mensajes.

- Define como debe ser el dialogo (intercambio de mensajes). Que mensajes se deben intercambiar.

- Ejemplo concreto: Protocolo HTTP y sus implementaciones mediante servidores WEB y browsers (navegadores).

Comunicación

Si dos procesos deben comunicarse:

- En la misma maquina: Cuando los procesos se ejecutan en el mismo sistema terminal, pueden comunicarse a través de sistemas de comunicación interprocesos que aplican reglas establecidas por el sistema operativo del sistema terminal.

- En distintas maquinas: Dos procesos de sistemas terminales diferentes se comunican intercambiando mensajes a través de una red de computadoras. Un proceso es el emisor, crea y envía mensajes, y el otro proceso es el receptor, recibe estos mensajes y puede responder con mensajes propios.

Modelos de Comunicación de Aplicaciones

- Modelo Mainframe (dumb client): Modelo de carga centralizada. Es un sistema donde todo el procesamiento ocurre en una computadora central (mainframe, el servidor) y los terminales (clientes) solo corren la comunicación y la interfaz física con el usuario. El mainframe es el que decide cuando le da el control al cliente. El mainframe maneja el dialogo de las comunicaciones

- Modelo Cliente/Servidor: Modelo de carga compartida. El cliente solicita servicios o recursos a un servidor central, que los proporciona. El servidor corre servicio esperando de forma pasiva la conexión, mientras que el cliente se conecta al servidor y se comunica a través de este. La idea inicial es que el cliente pone procesamiento de interfaz y el servidor el resto del procesamiento.

- Modelo Peer to Peer (P2P): Modelo de carga compartida y distribuida. Una arquitectura de red donde los participantes (peers) pueden actuar como clientes y servidores al mismo tiempo para compartir recursos entre ellos. Es un sistema escalable en cuanto a rendimiento pero no en cuanto a la administración.

- Modelo Híbrido: Modelo de carga compartida y distribuida. Al igual que P2P los participantes pueden actuar como clientes y servidores al mismo tiempo para compartir recursos entre ellos, la diferencia recae en que existen diferentes tipos de nodos con diferentes roles. Hay nodos centrales donde se registra la información y al resto de los nodos. Es un sistema escalable en cuanto a rendimiento y *supongo que* mejor en administración que P2P puro.

Requerimientos

Cada aplicación puede tener diferentes requerimientos: seguridad, tiempo de respuesta, confiabilidad, optimizar ancho de banda. De acuerdo a estos requerimientos estará asociada a determinado protocolo transporte.

User Agent

Es una interfaz entre el usuario y la aplicación de red.

HTTP

HTTP se implementa mediante dos programas, cliente y servidor (modelo cliente/servidor) que se comunican entre sí intercambiando mensajes HTTP. HTTP define la estructura de estos mensajes y cómo el cliente y el servidor intercambian los mensajes. HTTP define cómo los clientes web solicitan páginas web a los servidores y

cómo estos servidores web transfieren esas páginas a los clientes. Utiliza TCP como su protocolo de transporte.

El cliente HTTP primero inicia una conexión TCP (puerto 80 como default, el cliente escoge cualquier puerto no privilegiado) con el servidor. Esto asegura que los mensajes se transmitan sin pérdida de datos, sin que HTTP se preocupe por pérdidas. HTTP es un protocolo sin estado.

HTTP Trabaja sobre texto ASCII, permite enviar información binaria con encabezados MIME.

- Clientes (llamados browsers o navegadores): Firefox, IE, Opera, Safari, Chrome.
- Servidores: Apache Server, MS IIS, NGINX , Google GWS, Tomcat.

HTTP VS HTML

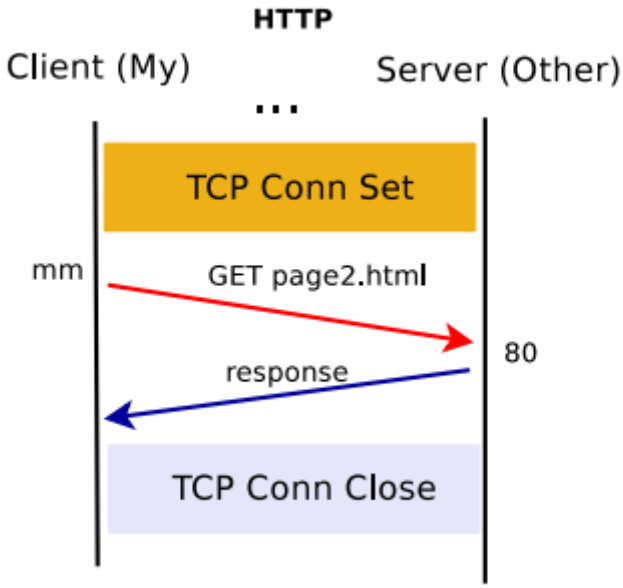
HTML es el lenguaje de marcado utilizado para crear la estructura y el contenido de las páginas web. La mayoría de las páginas web están constituidas por un archivo base HTML y varios objetos referenciados. Por ejemplo, si una página web contiene texto HTML y cinco imágenes JPEG, entonces la página web contiene seis objetos: el archivo base HTML y las cinco imágenes. El archivo base HTML hace referencia a los otros objetos contenidos en la página mediante los URL de los objetos. Cada URL tiene dos componentes: el nombre de host del servidor que alberga al objeto y el nombre de la ruta al objeto.

El navegador web (cliente) es el que interpreta y renderiza el código HTML para mostrar la página web al usuario y el servidor web es el encargado de enviar los archivos HTML. El contenido HTML se encuentra en la entidad de respuesta, y es lo que el servidor envía al cliente como parte de la respuesta a la solicitud HTTP.

El envió y recibimiento de este mismo se hace a través de mensajes respetando el protocolo HTTP.

HTTP 0.9

- Nunca se eestandarizó
- Pasos para obtener un documento:
  1. Establecer la conexión TCP
  2. HTTP Request vía comando GET.
  3. HTTP Response enviando la página requerida.
  4. Cerrar la conexión TCP por parte del servidor.
  5. Si no existe el documento o hay un error directamente se cierra la conexión.



- Solo una forma de Requerimiento.
- Solo una forma de Respuesta.
- Request/Response sin estado.

  Request ::== GET <document-path> <CR><LF>

  Response ::== ASCII chars HTML Document.

  GET /hello.html <CR><LF>

  GET / <CR><LF>

HTTP 1.0

- Versión de HTTP 1.0 estándar [RFC-1945]
- Define formato basado en HTTP 0.9
  - Se debe especificar la versión en el requerimiento del cliente.
  - Para los Request, define diferentes métodos HTTP.
  - Define códigos de respuesta.
  - Admite repertorio de caracteres, además del ASCII, como: ISO-8859-1,
  - UTF-8, etc.
  - Admite MIME (No solo sirve para descargar HTML e imágenes).
  - Por default NO utiliza conexiones persistentes.
  - Todo se basa en HEADER (básicamente permiten al cliente y al servidor enviar información adicional junto a una petición o respuesta, son un par clave:valor)

Formato Request

```
<Method> <URI> <Version>
[<Headers Opcionales>]
<Blank>
[<Entity Body Opcional>]
<Blank>

<Method HTTP 1.0> ::== GET, POST, HEAD, PUT,
                       DELETE, LINK, UNLINK
```

Formato Response

```
<HTTP Version> <Status Code> <Reason Phrase>
[<Headers Opcionales>]
<Blank>
[<Entity Body Opcional>]
```

Ejemplo

```
GET /index2.html HTTP/1.0
User-Agent: telnet/andres (GNU/Linux)
Host: estehost.com
Accept: */*

HTTP/1.1 200 OK
Date: Mon, 21 Apr 2008 00:28:51 GMT
Server: Apache/2.2.4 (Ubuntu)
Last-Modified: Mon, 21 Apr 2008 00:18:14 GMT
ETag: "a3b36-1f-91d5d80"
Accept-Ranges: bytes
Content-Length: 31
Connection: close
Content-Type: text/plain
```

```
<HTML>
<H1> HOLA </H1>
...
</HTML>
```

Métodos HTTP

- ✧ GET: obtener el documento requerido. Puede enviar información, pero no demasiada. Es enviada en la URL del requerimiento. Con el formato ?var1=val1&var2=val2.... La cantidad e información está restringida al tamaño de la URL. En general, 256 bytes.
- ✧ HEAD: idéntico a GET, pero solo requiere la metainformación del documento, por ejemplo, su tamaño. No obtiene el documento en sí.
- ✧ POST: hace un requerimiento de un documento, pero también envía información en el body. Generalmente, usado en el fill-in de un formulario HTML(FORM). Puede enviar mucha más información que un GET.
- ✧ PUT: usado para reemplazar un documento en el servidor. En general, deshabilitada. Utilizado, por ejemplo, por protocolos para compartir archivos y carpetas montados sobre HTTP, como WebDAV [WDV].
- ✧ DELETE: usado para borrar un documento en el servidor. En general, deshabilitada. También, puede ser utilizada por WebDAV.
- ✧ LINK, UNLINK: establecen/des-establecen relaciones entre documentos.

Los mensajes (operaciones) PUT, DELETE, LINK, UNLINK son pasados directamente a la aplicación si el servidor web los soporta. PUT/DELETE No se implementan directamente en el servidor. Se deben agregar como una extensión CGI para manejarlos. La diferencia de estos dos con POST es que el archivo que se indica puede no existir y el script de procesamiento es el mismo.

Ejemplos de respuestas HTTP/1.0

HTTP/<version> 200 OK

HTTP/<version> 301 Moved Permanently

HTTP/<version> 400 Bad Request

HTTP/<version> 403 Access Forbidden

HTTP/<version> 404 Not Found

HTTP/<version> 405 Method Not Allowed

HTTP/<version> 500 Internal Server Error (CGI Error)

HTTP/<version> 501 Method Not Implemented

Multiplexación

Mediante el parámetro Host se pueden multiplexar varios servicios sobre un mismo host. Esto quiere decir que permite que un único servidor web responda a solicitudes para múltiples dominios o servicios en el mismo host

Autenticación

HTTP/1.0 contempla autenticación con WWW-Authenticate Headers. El servidor, ante un requerimiento de un documento que requiere autenticación, enviara un mensaje 401 indicando la necesidad de autenticación y un Dominio/Realm. El navegador solicitara al usuario los datos de user/password (si es que no los tiene cachedos) y los enviara en texto claro al servidor. El servidor dará o no acceso en base a esos valores. Para los siguientes requerimientos, el navegador usara los valores que tiene almacenados para el Realm solicitado.

Conexión no persistente vs Conexión persistente

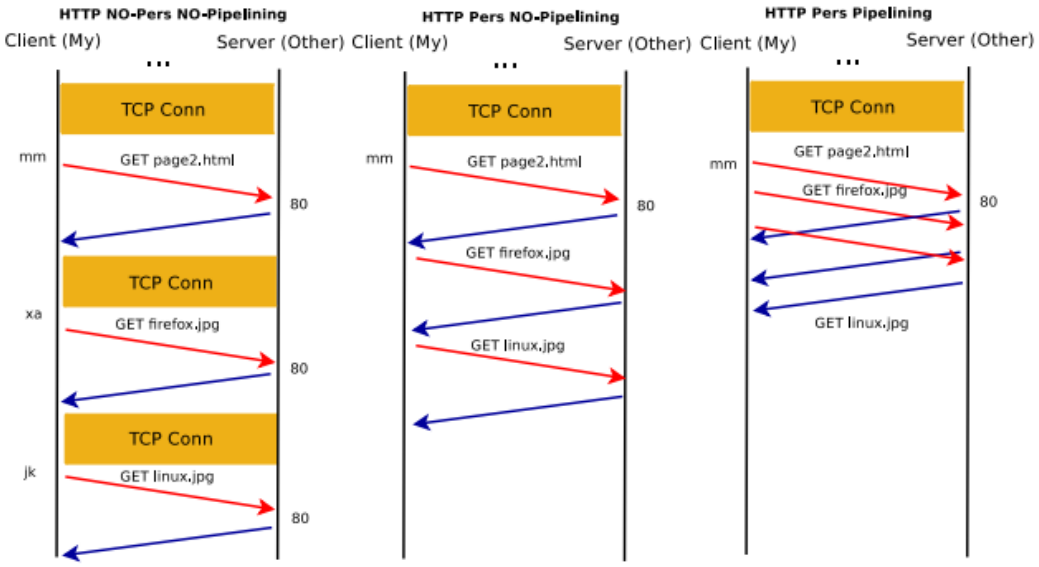
Cada recurso se solicita por separado al servidor. Si se tiene una conexión no persistente, por cada recurso se abre y cierra la conexión TCP. Esto implica una sobrecarga asociada a la apertura y cierre de conexiones, lo que conduce a una carga más lenta de las páginas web.

En cambio, si se tiene una conexión persistente, no se abre y cierra la conexión TCP por cada transacción.

Sin embargo, hay un problema asociado con la conexión persistente, especialmente cuando se trata de recursos grandes o de larga duración. Si un recurso se mantiene en la conexión durante un período prolongado, podría bloquear la conexión y retrasar la obtención de otros recursos. Esto se conoce como el problema de head-of-line blocking. Para abordar este problema, los navegadores modernos utilizan técnicas como la multiplexación y el pipelining para permitir la transferencia de múltiples recursos a la vez. Aun así Pipelining requiere que los responses sean enviado en el orden solicitado, HOL posible

HTTP 1.1

- HTTP/1.1, 1997 con la [RFC-2068] se actualiza con [RFC-2616].
- Nuevos mensajes HTTP 1.1: OPTIONS, TRACE (utilizada para debugging) , CONNECT (utilizada para generar conexiones a otros servicios montadas sobre HTTP).
- Conexiones persistentes por default.
- Pipelining, mejora tiempo de respuestas.
    - No necesita esperar la respuesta para pedir otro objeto HTTP.
    - Solo se utiliza con conexiones persistentes.
    - Mejora los tiempos de respuestas.
    - Sobre la misma conexión de debe mantener el orden de los objetos que se devuelven.
    - Se pueden utilizar varios threads para cada conexión



Redirects

Los redirect son mensajes que se utilizan para enviar al user-agent (browser) a otra ubicación debido a que el recurso ha sido movido. Se realiza un Redirect temporal 302, indicando la nueva URL/URI, donde se encuentra la respuesta. La respuesta debe darse en el campo Location y/o en un HTML corto

El user-agent no debería re-direccionarlo salvo que el usuario confirme, aunque varios browser lo hacen. Moved Permanently 301, se indica que cualquier acceso futuro debe realizarse sobre la nueva ubicación (mejora Indexadores). Se pueden generar problemas con Cookies.

CGI y Javascript

- CGI (Common Gateway Interface): aplicación que interactúa con un servidor web para obtener información de los requerimientos generados por el cliente y así poder responder. Todo el procesamiento de CGI es realizado del lado del servidor: server-side, el cliente solo genera los datos y los envía para que luego sean procesados.
- Javascript: client-side script (ejecuta del lado del cliente). Usa modelo de objetos DOM, permiten extensiones como AJAX (Asynchronous JavaScript And XML). Este hace requerimientos particulares y no necesita recargar toda la página.

Cookies

Las cookies, definidas en [RFC 6265] son un mecanismo que permite a las aplicaciones web del servidor “manejar estados”. Habitualmente son utilizadas para:

- Autenticación por aplicación.
- Carritos de compras (shopping carts).

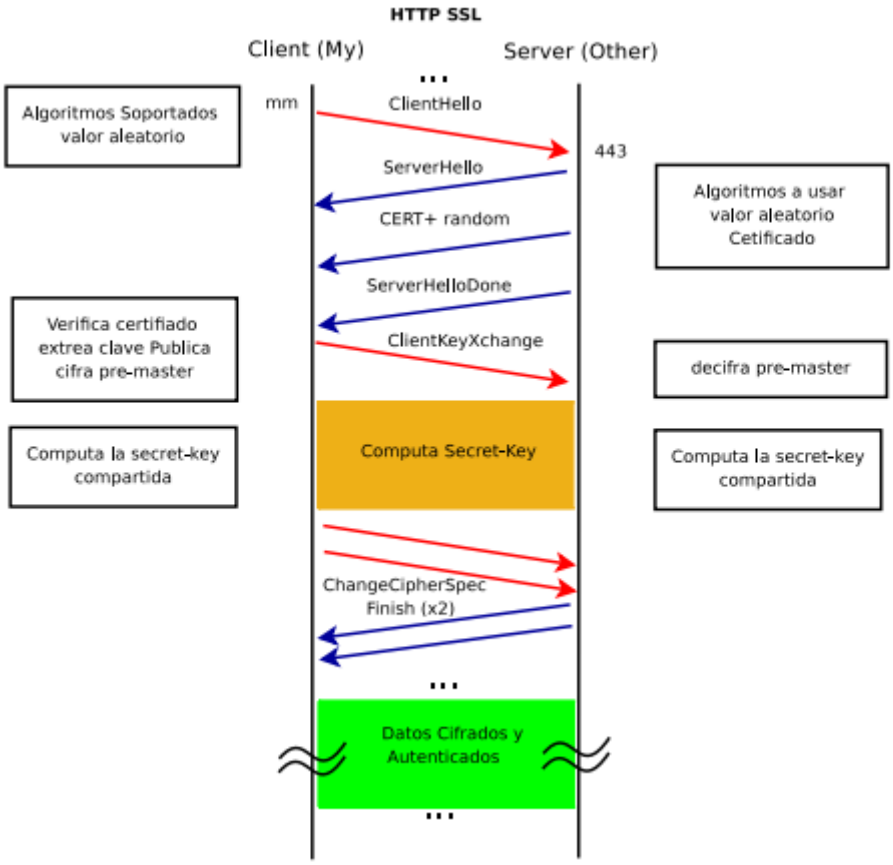
- Preferencias, recomendaciones.
- Estado de sesión de usuario (user session state, e.g. web-email).

¿Cómo funcionan?

El cliente hace un request, el servidor retorna un recurso (un objeto HTTP, como una página HTML) indicando al cliente que almacene determinados valores por un tiempo (esta cookie es introducida al cliente mediante el mensaje en el header Set-Cookie: mensaje que indica un par nombre,valor e incluye fecha de expiración). Despues el cliente en cada requerimiento luego de haber almacenado la Cookie se la enviara al servidor con el header Cookie:. El servidor puede utilizarlo o no y el servidor puede borrarlo. Las cookies se almacenan por site y puede haber varias por c/u.

HTTPS

Utiliza el port 443 por default. Hay una etapa de negociación previa, despues de esto se cifra y autentica todo el mensaje HTTP (incluso el header).



Web-Cache

Sirve para “Proxiar” y Chachear recursos HTTP.

Objetivos:

- Mejorar tiempo de respuesta (reducir retardo en descarga).

- Ahorro recursos de la red.
- Balance de carga, atender a todos los clientes.

Se solicita el objeto, si está en cache y esta “fresco” se retorna desde allí (HIT). Si el objeto no esta o es viejo se solicita al destino y se cachea (MISS). Se puede realizar control de acceso.

Del lado de cliente, los web browser tienen sus propias cache locales. Los servidores agregan los headers Last-Modified: date ETag: (entity tag) hash y hay requerimientos condicionales desde los clientes: If-Modified-Since: date If-None-Match: hash. El servidor puede responder 304 Not Modified (si por ejemplo se solicita If-Modified-Since: date y no esta modificado) o 200 OK (si sí lo esta).

Del lado del servidor, los cache como servers funcionan como Proxy. Son servidores a los clientes y clientes a los servidores web. Los instalan ISP o redes grandes que desean optimizar el uso de los recursos. En general corren sobre UDP.

Protocolos de comunicación entre web-cache servers:

- ICP (Internet Cache Protocol).
- (HTCP) Hyper Text Caching Protocol.

HTTP 2

Es un reemplazo de como HTTP se transporta, no es un reemplazo completo, se conservan métodos y semánticas. Es un protocolo binario en lugar de ASCII y la mayoría está montado en TLS

Problemas con HTTP 1.1

- Pipelining requiere que los responses sean enviado en el orden solicitado, HOL posible.
- POST no siempre pueden ser enviados en pipelining.
- Demasiadas conexiones generan problemas, control de congestión, mal uso de la red.
- Muchos requests, muchos datos duplicados (headers)

Solución HTTP 2

- Multiplexa varios request en una petición en lugar de ser una secuencia ordenada y bloqueante.

HTTP 2 permite a los servidores “pushear” datos a los clientes (servidor enviar recursos adicionales al cliente antes de que este los solicite de manera explícita)

HTTP hace uso del **stream**, que es un canal virtual bidireccional que opera dentro de una única conexión TCP. Cada solicitud y respuesta se intercambia a través de un stream separado. Cada stream posee un ID y una prioridad asignada. Los mensajes HTTP 2 (solicitudes y respuestas) se transmiten utilizando estos streams. Los mensajes HTTP 2 se dividen en **frames** que se envían a través del mismo stream.

Básicamente, los streams son identificados y desglosados en frames, que constituyen la unidad de comunicación más pequeña. Estos frames se transmiten y se vuelven a ensamblar en el destino. Existen varios tipos de frames, como los de encabezado, de datos y de prioridad. Estos frames se combinan para formar las solicitudes y respuestas que se intercambian en los streams. Todos los streams coexisten en una misma conexión.

Frame types: HEADERS, DATA, PUSH_PROMISE, WINDOW_UPDATE, SETTINGS,

El mismo stream puede ser usado para llevar diferentes mensajes.

- Los streams dentro de una misma conexión tienen flow-control individual.
- Los streams pueden tener un weight (prioridad).
- Los streams pueden estar asociados de forma jerárquica, dependencias.

Headers en HTTP/2

No se codifican más en ASCII. Surgen nuevos pseudo-headers que contiene información que estaba en el método y otros headers.

Por ejemplo: `HEAD /algo HTTP/1.1` se reemplaza con HTTP 2:

`:method: head`

`:path: /algo`

`:scheme: https o http`

`:authority: www.site.com` reemplaza al headerHost:.

Para las respuestas: `:status:` códigos de retornos 200, 301, 404, etc.

Puede haber compresión de encabezados SPDY/2 propone usar GZIP. GZIP + cifrado, tiene “bugs” utilizados por atacantes. Se crea un nuevo compresor de Headers: HPACK. H2 y SPDY, soportados en la mayoría de los navegadores

Negociación de protocolo

Con HTTP 1, si tanto el cliente como el servidor soportan el mismo protocolo, se puede negociar hacer un “upgrade”. Hay un Upgrade Header. `Connection: Upgrade, HTTP2-Settings Upgrade: h2c|h2.`

En HTTP 2 (y HTTP 3) se puede negociar con ALPN (extensión de TLS) entre el cliente y el servidor el protocolo de aplicación que utilizarán durante la comunicación segura. En la negociación ALPN, el servidor envía una lista de protocolos de aplicación que admite. Los clientes pueden seleccionar uno de los protocolos ofrecidos. Los ejemplos que mencionaste, como "h2" para HTTP/2, "h3" para HTTP/3 y "http/1.1" para HTTP/1.1, son ejemplos comunes de lo que podría ofrecer un servidor.

HTTP 3

HTTP 3 utiliza QUICK(UDP) como protocolo de transporte.