Capa de presentación:

Nos va a permitir que dos extremos (dispositivos totalmente distintos) se entiendan. Los lleva a un modelo estándar para que todos se entiendan.

Tiene los datos de como se manda la información, si se manda encriptada, como se encriptó, desencriptó.

Capa de Aplicación:

Nos va a permitir comunicarnos con la red

Hay muchos protocolos. Cada uno de estos define su forma de trabajar, como se manda el mensaje, que formato, etc.

Hay 4 modelos:

* Mainframe (dumb client)
* Cliente/Servidor: El servidor en estado “listening”
* Peer to Peer (P2P):
* Hibrido: Puede haber servidores o P2P, es una mezcla

Atraves de TCP/IP podemos identificar un equipo. Solo identificar el equipo, no lo que pasa dentro.

El port o puerto nos indica a que servicio o proceso va dirigido tal mensaje.

En la parte de aplicación tenes el servidor y la aplicación web.

5 Valores:

* Ip local
* puerto local
* ip receptor
* puerto receptor
* protocolo

Protocolo HTTP:

Formato de URL: protocol://[user:pass@]host:[port]/[path]

Aplica el modelo Cliente/Servidor

Es un servidor sin estado, no se acuerda lo que hace.

Los puertos 0 a 1023, son puertos privilegiados

Los puertos 1024 en adelante son puertos no privilegiados

Clientes: browsers o navegadores

Servidores: NGINX, Apache, etc.

Client REQUEST:

* La primer línea es el header, request.
* Desde la segunda línea en adelante se cuentan los headers que tienen un formato: “VALOR: CONTENIDO”. La cantidad de headers que hay son los que siguen ese formato

Server REQUEST:

* La primer línea de estado.

Respuestas:

* HTTP/ 200 OK
* HTTP/ 301 Moved Permanently
* HTTP/ 400 Bad Request
* HTTP/ 403 Access Forbidden
* HTTP/ 404 Not Found
* HTTP/ 405 Method Not Allowed
* HTTP/ 500 Internal Server Error (CGI Error)
* HTTP/ 501 Method Not Implemented

Metodos:

* GET: Para obtener un valor. Puede enviar información, pero no demasiada. No espera recibir datos en body.
* HEAD: Idéntico al GET pero no me devuelve el objeto. Solo devuelve la cabecera
* POST: Esta pensado para enviar datos al servidor. Lo envia dentro del body.
* PUT: Sube un objeto desde el cliente al servidor.
* DELETE: Permite borrar un archivo dentro del servidor.
* LINK, UNLINK: Permite establecer relaciones entre distintos objetos del servidor.

La cabecera HOST en la versión 1.1 es obligatoria. Puede aparecer en la 1.0 pero no es obligatoria.

HTTPS:

En https nos conectamos por default al puerto 443, en cambio en http es el 80.

El servicio HTTPS es el mismo que el HTTP, pero la información se envia cifrada.

El cliente quiere hacer un get, se abre el TCP, y antes de que devuelva la respuesta, se encripta todo

Protocolo TLS:

Provee verificación de cifrado de autenticación

Diferencia entre HTTP 1.0 Y 1.1:

El http 1.1 es persistente y no necesita conectarse por cada petición.

EL http 1.1 puede recibir todos los requerimientos en diferente orden, que en cambio el http 1.0 debía recibir todo en orden y por ejemplo hasta no recibir el primero no empezaba.

El http 1.1 tiene el **PIPELINING** hace todas las solicitudes a la vez, en vez de esperar a que llegue la solicitud anterior y sea mas trabado.

Gráfico, Diagrama

El contenido generado por IA puede ser incorrecto.

El primero es http 1.0 que debe hacer la conexión TCP en cada petición. El segundo es el HTTP 1.1 sin pipelining que espera a que llegue para enviar otra petición. El tercero es el http 1.1 con pipelining enviando y recibiendo todo.

**HTTP 2**:

No se puede utilizar sin cerficado.

STREAMS:

Son subconexiones

PUSH\_PROMISE:

Se envía información sin que el cliente lo pida ya que el servidor es conciente de lo que se debe procesar.

Headers:

Se mantienen casi todos los headers del 1.1, pero se crean nuevos pseudo-headers:

Por ejemplo: HEAD /algo HTTP/1.1 se reemplaza con http2:

* :method: head
* :path: /algo
* :scheme: https o http
* :authority: www.site.com reemplaza al header “Host:”
* Para las respuestas: :status: códigos de retornos 200, 301, 404, etc.

Control de flujo:

Controla la taza de transferencia

Si lo hago con TCP se hace a todo, en cambio, si lo hago con stream puedo parar la información de un solo stream.

**HTTP 3**:

Desaparece TCP y aparece QUIC.

QUIC es un protocolo que esta dentro de la aplicacion.

Protocolo de transporte pero metido en la aplicación.

Un problema de UDP es que facilita algunos ataques de seguridad. Por ejemplo: Con UDP podes falsear tu dirección IP.

DNS: