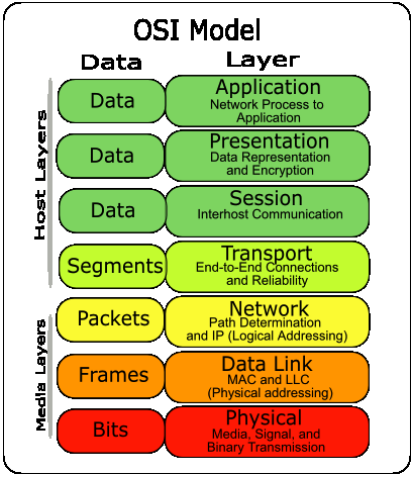
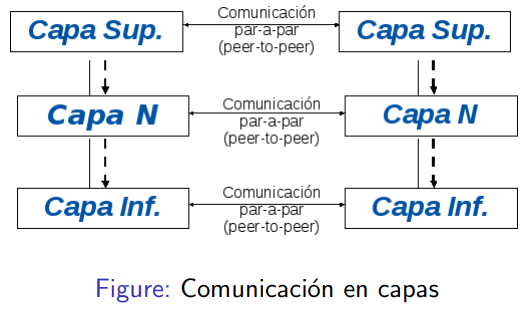
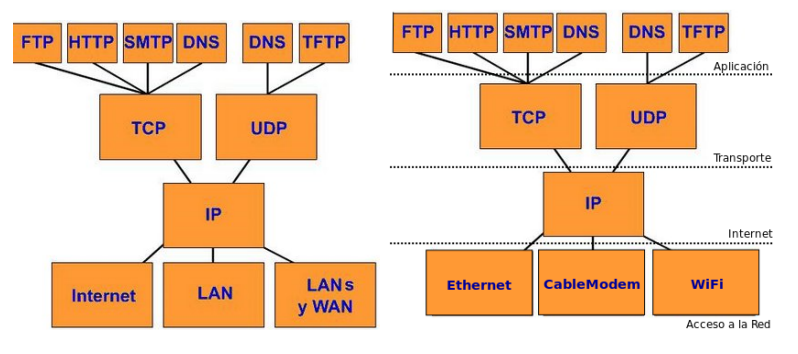
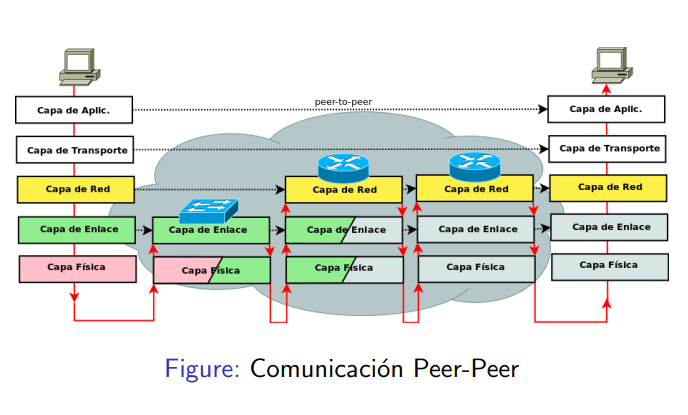
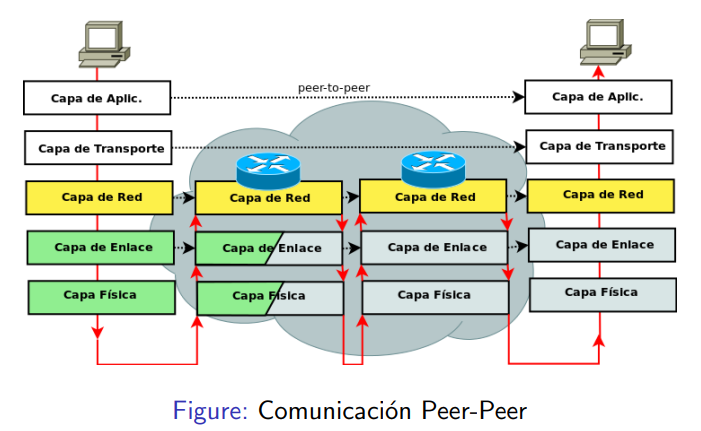
* **Red de Computadoras**: Es un grupo de computadoras / dispositivos interconectados con el objetivo de compartir recursos, información o servicios.
* **Sistema de comunicación**: Un sistema de comunicación es el conjunto de computadoras, software de red, medios y dispositivos de interconexión.
  + Fuente
  + Emisor/Transmisor
  + Medio de transmisión y dispositivos intermedios.
  + Procesos intermedios de tratamiento de información
  + Receptor
  + Destino
  + Protocolos, información, mensaje transmitido
  + Señal de información
* Componentes de un Red: Estos componentes deben interactuar y combinarse a través de reglas.
  + Computadoras
  + Routers/switches, Gateways, Access Points
  + NIC(Placa de red), Módems
  + Vínculos/ enlaces: conformados por:
    - Medios: cable, fibra óptica, antenas, etc.
  + Programas: Servidores web, clientes de mail, servers de streaming.
  + Etc…
* **Protocolo**:
  + **Definición trivial**: El protocolo es un conjunto de conductas y normas a conocer, respetar y seguir en todo medio.
  + **Definición informática**: Un protocolo define el formato, el orden de los mensajes intercambiados y las acciones que se llevan a cabo en la transmisión y/o recepción de un mensaje u otro evento.
  + **Protocolo de Red**: Es el conjunto de reglas que especifican el intercambio de información durante la comunicación entre las entidades que forman parte de una misma red.
* Protocolos de Redes Propietarios
  + En un principio cada compañía implementó su propia red privada, la consecuencia fue que cada red tenía sus protocolos propios, generando incompatibilidad.
* Combinación de Protocolos
  + Por la complejidad generada al ver la interacción de tantos componentes de red, se requiere un **Modelo de Organización**.  
    - Modelo en Capas: Divide la complejidad en componentes reusables
      * Reduce la complejidad en componente más pequeños.
      * Las capas de abajo ocultan la complejidad a las de arriba, **las abstrae**.
      * Las capas de arriba utilizan los servicios de las de abajo. **Funcionan como interfaces**.
      * Los cambios en una capa no deberían afectar a las demás si la interfaz se mantiene.
      * Facilita el desarrollo, evolución de los componentes de red asegurando interoperabilidad.
      * Facilita el aprendizaje, diseño y administración de las redes.
    - **Modelo OSI**(Open System Interconnection)
      * Creada por La ISO en el 84
      * Basado en los modelos de red:
        + DECNET
        + SNA|
        + TCP/IP
      * Modelo Abierto y estándar.
      * Modelo dividido en 7 capas.
      * Modelo de referencia.

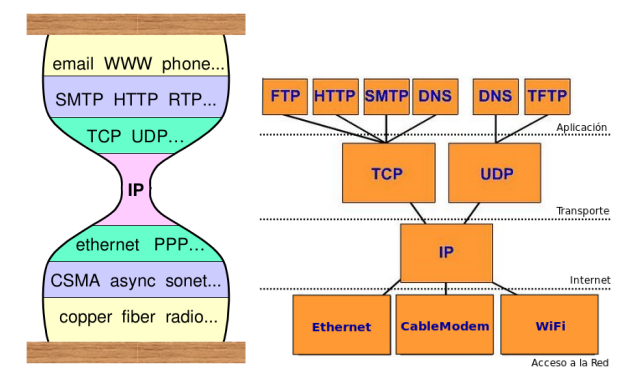


Modelo en Capas: capa ofrece servicios a la capa superior, usa modelos de la inferior, mediante interfaz.

* + Capas de Host: 7,6,5,4, proveen envío de datos de forma confiable.  
    - (7)Aplicación: Servicios de red a los usuarios y a procesos, aplicaciones.
    - (6)Presentación/Representación: formato de los datos.
    - (5)Sesión: Mantener track de sesiones de la aplicación.
    - (4)Transporte: Establecer y mantener canal “seguro” end-to-end.
  + Capas de Medio: 3,2,1, controlan el envío físico de los mensajes sobre la red.  
    - (3)Red: Direccionar y rutear los mensajes host-to-host. Comunicar varias redes
    - (2)Enlace de Datos: Comunicación entre entes directamente conectados.
    - (1)Física: Transportar la información como señal por el medio físico.
* **Modelo TCP/IP**
  + Modelo que se convirtió en estándar
  + Protocolos que se encuentran en Internet
    - Modelo Abierto
    - Protocolos de nivel de enlace: Ethernet, 802.3, PPP, HDLC, etc.
    - Protocolos propios de Internet y Transporte: ARP, IP, ICMP, TCP, etc.
    - Protocolos de Aplicaciones: DNS, HTTP, FTP, SSH, SMTP, IMAP, etc.
    - API abierta para generar nuevos protocolos.
  + Modelo de 5 capas:
    - Capa de Aplicación
    - Capa de Transporte o Host-to-Host.
    - Capa de Internet o Internetworking.
    - Capa de Enlace.
    - Capa de Física.
  + Algunos autores agrupan la Capa de Enlace con la Capa de Física llamándola Capa de acceso a la Red.
* Comparación: **OSI** vs. **TCP/IP**
  + Similitudes:
    - Ambos están divididos en capas
    - Ambos tienen capas de aplicación, aunque sus servicios sean distintos.
    - Ambos tienen capas de transporte similares.
    - Ambos tienen capa de red similar pero con diferente nombre.
    - Se supone que la tecnología es de conmutación de paquetes.
    - Es importante conocer ambos modelos.
  + Diferencias:
    - TCP/IP combina las funciones de la capa de presentación y de sesión en la capa de aplicación
    - TCP/IP combina las capas de enlace de datos y la capa física del modelo de OSI en una sola capa.
    - TCP/IP es más simple porque tiene menos capas.
    - Los protocolos TCP/IP son los estándares en torno a los cuales se desarrolló Internet, de modo que la credibilidad del modelo TCP/IP se debe a sus protocolos.
    - El modelo OSI es un modelo “más” de referencia, aunque hay implementaciones.
* Comunicación entre Capas Peer-Peer
  + Cada capa usa el servicio de la de abajo.



Cada capa se comunica con la capa del otro extremo.

* Clasificación de Redes
  + Clasificación por cobertura, distancia, alcance.
    - LAN
    - MAN
    - WAN
    - SAN
    - PAN
    - CAN
  + Clasificación por acceso abierto o privado.
    - Internet
    - Intranet
    - Extranet
  + Clasificación por topología física.
    - Redes de Conmutación de Circuitos.
    - Redes de Conmutación de Tramas/Paquetes.
      * Servicios Orientados a Conexión.
      * Servicios NO Orientados a Conexión.
  + Clasificación por tipo de conexión/medio.
  + Etc.
* **Internet**: Es una red de redes de computadoras, descentralizada, pública, que ejecutan el conjunto abierto de protocolos TCP/IP. Integra diferentes protocolos de un nivel más bajo:
  + Modelo:  
    
  + Su estructura está organizada en una jerarquía.  
    - Capa de Acceso (Edge): Acceso residencial, acceso de organizaciones.
    - Capa de núcleo (Core): Dividida en niveles
      * Proveedores Regionales
      * Proveedores Nacionales
      * Proveedores Internacionales
      * Proveedores Internacionales en el TIER 1
    - Tecnologías de Internet:
      * Diferentes tecnologías de última milla y de redes locales.
      * Despliegue local y de acceso de las personas y las organizaciones.
    - Estructura (Core):
      * Fibra óptica, Cobre, satélites.
      * Se interconectan mediante Point Of Presence con proveedores.
      * Entre proveedores se interconectan mediante NAPs o conexiones directas.
      * Actualmente los NAPs se los llama IXPs
* **Capa de Aplicación**
  + Funciones
    - Provee servicios de comunicación a los usuarios y a las aplicaciones.
    - Posee un modelo de comunicación Machine to Machine, no hay usuarios.
    - Provee la Interfaz con el usuario u otras aplicaciones.
    - Las aplicaciones que usa la red y los protocolos que implementan pertenecen a esta capa
    - Existen aplicaciones que NO son de red que deben trabajar con aplicaciones/servicios para lograr acceso a la red.
  + Componentes
    - Elementos de capa de aplicación: **Programas** que corren en diferentes plataformas y se comunican entre sí. Junto con los **Protocolos** que implementan
    - Las aplicaciones en la mayoría de los casos corren en los nodos finales, no en el núcleo de la red. Siguen el principio end-2-end.
    - Integra la Capa de Aplicación, Capa de Presentación y Capa de Sesión
* **Capa de Sesión**
  + Administra las conversaciones entre las aplicaciones
  + Es posible que provea mecanismos de sincronización
  + Maneja actividad
  + Informa de excepciones
  + Integrada en las aplicaciones de red mismas.
  + Puede estar ausente.
    - Consideraciones
      * Ninguna de las redes sacando servicios de modelo OSI la usa
      * Es muy delgada y posee pocas características
      * No se utiliza generalmente
* **Capa de Presentación**
  + Se encarga de la conversión y codificación de datos a codificaciones comunes, por ejemplo: ASCII, UTF-8, Unicode16, etc.
  + Comprensión y descompresión de datos.
  + Cifrado y descifrado de datos.
  + Define formatos y algoritmos para esto: JPEG, MPEG, etc.
  + Está integrada en las aplicaciones de red mismas.
* **Capa de Aplicación**
  + Define el formato de los mensajes.
  + Define la semántica de cada uno de los mensajes.
  + Define cómo debe ser el intercambio y qué mensajes se deben intercambiar entre las aplicaciones.
* Arquitectura / Modelos de Comunicación de Aplicaciones
  + **Modelo Mainframe**
  + **Modelo Cliente/Servidor.**
  + **Modelo P2P**
  + **Modelo Híbrido.**
  + **Modelo Mainframe Centralizado**
    - Modelo de Carga concentrada.
    - El cliente solo corre la información y la interfaz física con el usuario
    - El servidor se encarga de todo el procesamiento
    - Modelo antiguo
    - Modelo puro: El mainframe decide cuando el cliente tiene el control y es el encargado de manejar el intercambio de mensajes entre aplicaciones
    - El cliente ejecuta el mainframe
  + **Modelo Cliente/Servidor**
    - Modelo de carga compartida
    - La idea inicial consta de que el cliente ponga procesamiento de interfaz. Luego el servidor pone el resto del procesamiento.
    - Existen modelos en varios tiers.
    - El servidor corre servicio esperando de forma pasiva la conexión
    - El cliente se conecta al servidor y se comunica a través de este
  + **Modelo P2P**
    - Modelo de carga completamente compartida y distribuida
    - Los participantes(peers) pueden cumplir el rol de cliente, servidor o ambos en un instante
    - Sistema escalable en cuanto a rendimiento
    - Sistema NO escalable en cuanto a administración
    - Ejemplo: Bittorrent.
      * BitTorrent
        + Protocolo para el intercambio de archivos grandes de forma masiva como P2P
        + Mediante el archivo .torrent obtenemos la dirección del TRACKER para unirse al grupo de PEERs
        + Desde el TRACKER se obtienen PEERs: SEEDs(Archivo completo) y LEECHERs(contiene el archivo parcial). El TRACKER se actualiza con el nuevo PEER.
        + La comunicación con el TRACKER se hace con HTTP o FTP
        + Los archivos se dividen en chunks, cada uno puede ser descargado de un PEER diferente.
        + Se conecta con otros PEERs y comienza la descarga. Cada chunk se comparte con otros PEERs.
        + Por cada archivo puede haber como máximo 4 PEERs remotos descargando. Los peers se rankia.
    - Modelo asimétrico N a N.
  + **Modelo P2P Híbrido**
    - Existen diferentes tipos de nodos con diferentes roles
    - Más escalable que P2P
    - Ejemplo: Skype.
      * Skype
        + Protocolo propietario que implementa VoIP
        + Nodos clientes pueden ser solo nodos y Supernodos.
        + Los nodos hacen de STUN, permitiendo conexión de computadoras detrás de NAT.
        + Los Super-Nodos pueden actuar como directorios, además de crear una red sobre otra red.
        + Los super nodos se conectan entre sí, utilizando servicios de autenticación.
        + Están los Mega SuperNodos que pertenecen a Skype
  + **Direccionamiento de Procesos**
    - Las aplicaciones son implementadas por los S.O como procesos
    - Posee conceptos de IPC(Comunicación interprocedural)
    - Para que un proceso reciba un mensaje, éste debe tener un identificador unívoco.
    - Se agrega un id por proceso junto con el del host.
    - Por nivel de transporte hay un servicio de multiplexación.
* **Principios de los protocolos de Capa Aplicación**
  + El software de una aplicación está distribuido entre dos o más sistemas.
  + Ese “pedacito” de software es un proceso.
  + Los procesos se comunican mediante mensajes.
  + La aplicación de red es diferente del protocolo de capa de aplicación. El protocolo de la capa de red es sólo una parte de la aplicación de red.
    - Una aplicación de red puede ser:
      * E-mail
      * La misma web
      * Juegos online
      * Telefonía Internet
      * Compartición de archivos P2P
  + Los protocolos de Capa de aplicación definen:
    - Tipos de mensajes intercambiados, mensajes de requerimiento y respuesta.
    - Sintaxis de los tipos de mensajes.
    - Semántica de los campos.
    - Reglas para cuándo y cómo los procesos envían y responden mensajes.
  + Protocolos de dominio público
    - Definidos en RFCs
    - HTTP, SMTP
  + Protocolos propietarios
    - KazaA (Bonzi Buddy)
* **Creación de una aplicación de red**
  + Escribe un programa que
    - Corra en diferentes sistemas y se comunique por la red
  + Los dispositivos internos no funcionan en la capa de aplicación
  + Este diseño permite desarrollos rápidos
* **Comunicación entre procesos**
  + Un proceso es un programa que corre en una máquina
  + Dentro de la máquina dos procesos se comunican usando comunicación entre proceso(Definida por el S.O).
  + Procesos en diferentes host se comunican vía intercambio de mensajes.  
    - Proceso Cliente: Proceso que inicia la comunicación(crea y envía mensajes sobre la red)
    - Proceso Servidor: Proceso que espera por ser contactado(Recibe los mensajes y, posiblemente responde con otros mensajes)
* **Sockets**
  + Los sockets son canales de comunicación que permiten que procesos no relacionados intercambien datos localmente y entre redes.
  + Los procesos envían / reciben mensajes a / desde sus socket
  + Se debe elegir un protocolo de transporte y es posible definir algún parámetro.
* **Direccionamiento de procesos**
  + Para que un proceso reciba un mensaje debe tener un identificador
  + Un host tiene una IP única de 32 bits. Esta IP incluye su dirección junto con un número de puerta asociado con el proceso en el host.
* **Agentes de Usuario**
  + Es una interfaz entre el usuario y la aplicación de red. Funciona como intermediario permitiendo ver e interactuar al usuario con la WEB.
* **Servicios de Aplicación**
  + Pérdida de Datos
    - Algunas aplicaciones pueden tolerar pérdida
    - Otras requieren transferencias 100% confiables
  + Retardo
    - Algunas aplicaciones requieren bajo retardo para ser “efectivas”
  + Bandwidth
    - Algunas aplicaciones requieren cantidad mínima de ancho de banda para ser “efectivas”
* **Servicios de los protocolos de Transporte**
  + Servicio TCP
    - Orientado a la conexión. Acuerdo requerido entre procesos cliente y servidor antes de la transferencia.
    - Transporte confiable entre proceso Tx y Rx
    - Control de flujo: Tx no sobrecarga a Rx
    - Control de congestión: frena al Tx cuando la red está sobrecargada
    - No provee garantías de retardo, ni ancho de banda mínimo.
  + Servicio UDP
    - Transferencia de datos no confiable entre proceso Tx y Rx.
    - No provee: nada de lo que posee TCP
      * **Esta al reverendo pedo.**
* **Elementos WEB**
  + Recurso u Objeto HTTP
  + Referenciado por una URI(Uniform Resource ID), URL o URN(Nombre)
  + Formato de URL: ***protocol://[user:pass@]host:[port]/[path].***
  + URN: No indica una ubicación, solo identifica categorías, poco implícitas.  
    **urn:isbn:0132856204**
  + Objetos pueden ser página web, imágenes JPEG, PNG, GIF, Java Applet, MP3, AVI, etc.
  + Páginas web: Archivo HTML que incluye vínculos o directamente otros objetos.
* **HTTP**
  + Modelo cliente/servidor, Request/Response
  + El protocolo corre sobre TCP (Requiere un protocolo de transporte confiable), puerto 80 el default.
  + El cliente escoge cualquier puerto no privilegiado
  + Trabaja sobre texto ASCII, permite enviar información binaria con encabezados MIME
  + Clientes(navegadores): Firefox, IE, Opera, Safari, Chrome
  + Servides:: Apache Server, MS ISS, NGINX, Google GWS, Tomcat.
* **Versión HTTP 0.9**
  + Primera versión de HTTP. Nunca se estandarizó
  + Para obtener un documento:
    - Establecer la conexión TCP
    - HTTP Request vía comando GET.
    - HTTP Response enviando la página requerida.
    - Cerrar conexión TCP por parte del servidor.
    - 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
* **Versión HTTP 1.0**
  + Versión de HTTP 1.0 estándar [RFC-1945]
  + Define formato, proceso basado en la ver 0.9
    - Se debe especificar la versión en el requerimiento del cliente.
    - Para los request, define diferentes métodos HTTP.
    - Define códigos de respuesta para diferentes situaciones(OK, Not Found, Bad Request, Etc).
    - Admite repertorio de caracteres, además de ASCII, como: UTF-8, etc.
    - Admite MIME(No solo sirve para descargar HTML e imágenes).
    - Por default NO utiliza conexiones persistentes.
    - Métodos:
      * GET: obtener el documento requerido. Puede enviar información pero no demasiada. Es enviada en la URL.
      * HEAD: idéntico al GET, pero sólo requiere la meta información del documento, por ejemplo, su tamaño, Usado por cliente con caché
      * POST: hace un requerimiento de un documento, pero también envía información en el Body. Generalmente, usado en el fil-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.
      * DELETE: usado para borrar un documento en el servidor. En general, deshabilitada.
      * LINK/UNLINK: vinculan/desvinculan documentos.
    - PUT, DELETE
      * No se implementan directamente en el servidor.
      * Se deben agregar como una extensión CGI para manejarlos
      * Debe leer un campo de longitud y leer esa cantidad de información de la entrada, luego generando el archivo.
      * Los utilizan protocolos como WebDAV, Ms File sharing/ OpenCloud.
      * Otra forma de subir o borrar archivos es usando el método POST y programando una CGI adecuada.
      * La diferencia con el POST es que el archivo que se indica puede no existir y el script de procesamiento es el mismo.
* **Autenticación HTTP**
  + HTTP 1.0 contempla autenticación con **WWW-Authenticate Headers**.
  + Encabezados, el cliente y el servidor intercambian información en el auth.
  + El servidor, ante un requerimiento de un documento que requiere autenticación, enviará un mensaje 401 indicando la necesidad de autenticación y un Dominio / Realm.
  + El navegador solicitará al usuario los datos de user/password y los enviará en el texto claro al servidor.
  + El servidor dará o no acceso en base a esos valores.
  + Para los siguientes requerimientos, el navegador usará los valores que tiene almacenados para el Realm solicitado.
* **HTTP 1.0 con conexiones persistentes**
  + No se contemplaron las conexiones persistentes por default.
  + A partir de HTTP 1.1, si.
  + En HTTP 1.0 se pueden solicitar de forma explícita.
* **HTTP 1.1**
  + Nuevos mensajes: OPTIONS, TRACE, CONNECT.
    - TRACE: utilizada para debugging.
    - CONNECT: utilizada para generar conexiones a otros servicios montadas sobre HTTP
  + Conexiones persistentes por omisión.
  + Pipelining, mejora el tiempo de respuestas.
  + No se necesita esperar para pedir otro objeto HTTP.
  + Solo se utiliza con conexiones persistentes.
  + Se debe mantener el orden de los objetos que se devuelven.
* **Redirects HTTP**
  + Redirect temporal 302, indicando la nueva URL/URI
  + El user-agent no debería redireccionarlo salvo que el usuario confirme.
  + Moved Permanently 301, se indica que cualquier acceso futuro debe realizarse sobre la nueva ubicación.
  + Se pueden generar problemas con Cookies.
* **CGI (Common Gateway Interface)**: Aplicación que interactúa con un servidor web.
* **CGIs Scripts y JavaScript**
  + Necesidad de dinamismo para generar aplicaciones
  + Server-Side Script:
    - Ejecuta del lado del servidor.
    - CGIs leen de **STDIN(POST)** o de variables de entorno (**GET**): **QUERY\_STRING** datos de usuario.
    - Escriben en la STDOUT response.
    - Deben anteponer el content-type en el header.
    - POST permite enviar más datos.
  + Client-Side Script
    - JavaScript estándar W3C
    - Usan modelo de objetos DOM(Document Object Model)
    - Permite extensiones como AJAX
    - Parseo XML para comunicarse.
  + Server-Side to Server-Side Script
    - Permite la comunicación entre servidores
    - Modelo de “objetos” y servicios distribuidos
    - Conjunto de convenciones para implementar RMI sobre HTTP
    - SOAP(Simple Object Access Protocol)
    - REST
* **Cookies**
  + Mecanismo que permite a las aplicaciones web del servidor “manejar estados”
  + El cliente hace un request
  + El servidor retorna un recurso indicando al cliente que almacene determinados valores por un tiempo
  + La Cookie es introducida al cliente mediante el mensaje en el header **Set-Cookie**
  + El cliente en cada requerimiento luego de haber almacenado la Cookie se le enviará al servidor con el header Cookie:.
  + El servidor puede utilizarlo o no, o hasta borrarlo.
  + Esta información puede ser usada por Client-Side scripts
* **HTTPS**
  + Utiliza el port 443 por default
  + Etapa de negociación previa
  + Luego se cifra y autentica todo el mensaje HTTP
* **WEB-Cache**
  + “Proxiar” y Cachear recursos HTTP
  + Objetivos
    - Mejorar el tiempo de respuesta
    - Ahorrar recursos de la red
    - Balance de carga, atender a todos los clientes.
  + Se solicita el objeto, si está en cache y está “fresco” se retorna desde allí(**HIT**).
  + Si no está o es viejo se solicita al destino y se cachea(**MISS**).
  + Se puede realizar control de acceso.
  + Caché del lado del cliente
  + Los web browsers tienen sus propias cache locales.
  + Los servidores agregan headers, requerimientos y respuestas.
  + Los cache como servers funcionan como Proxy
  + Son servidores a los clientes y clientes a los servidores web.
  + Protocolos de comunicación entre WEB-Cache servers
    - ICP(Internet Cache protocol)
    - HTCP(Hyper text caching protocol)
  + Diferentes relaciones: Parent, siblings
  + Protocolo de comunicación entre router y web-cache servers.
  + Corren sobre UDP.
* **HTTP 2**
  + Reemplazo de cómo HTTP se transporta, manteniendo métodos y semántica
  + BAse del trabajo protocolo desarrollado por Google
* **Problemas con HTTP 1.0 y 1.1**
  + un request por conexión, por vez, muy lento.
  + Alternativas
    - Pipelining y conexiones persistentes.
    - Generar conexiones paralelas.
  + Problemas
    - Pipelining requiere que los responses sean enviados en el orden solicitado, HOL posible.
    - POST no siempre puede ser enviado en pipelining
    - Demasiadas conexiones generan problemas, control de congestión, mal uso de red.
* **Diferencias principales con HTTP 1.1**
  + Protocolo binario en lugar de textual
  + Multiplexa varios request en una petición
  + Utiliza una conexión para pedir y traer datos en paralelo
  + Utiliza compresión de encabezado.
  + Permite a los servidores “pushear” datos a los clientes.
* **HTTP 2 mux stream, framing**
  + Puede generar una o más conexiones TCP
  + Un stream es como una sub-conexión
  + Un stream tiene un ID y una prioridad
  + Sobre una conexión TCP multiplexa uno o más streams
  + Los streams transportan mensajes. Están divididos en frames(header fijo + payload variable)
  + Frame Types:
    - HEADERS
    - DATA
    - PUSH\_PROMISe
    - WINDOW\_UPDATE
    - SETTINGS
    - etc.
* **HTTP 2 HEADERS**
  + Se mantienen de HTTP 1.1
  + No se codifica más en ASCII
* HTTP 2 Priorización y flow-control
  + Streams dentro de una misma conexión tienen flow-control
  + Los streams pueden tener un weight(prioridad)
  + Pueden estar asociados de forma jerárquica, dependencias.
* HTTP 2 inline vs. push
  + Cuando el cliente solicita una página, “parsea” el primer response HTML luego solicita el resto.
  + El server puede enviar el HTML más otros datos
  + No siempre es lo que necesita el cliente, depende de qué funcionalidad ofrece