Redes y Comunicaciones

- **¿Qué es una red de computadoras/ordenadores?**

Es un grupo de computadoras/dispositivos interconectados. Su objetivo es compartir recursos (dispositivos, información, servicios, etc.).
El conjunto de computadoras, software de red, medios y dispositivos de interconexión forma un sistema de comunicación. Por ejemplo: red de la sala de PCs, red Universitaria, Internet.

- **Componentes de un Sistema de Comunicación**
    - Fuente (Software).
    - Emisor/Transmisor (Hardware).
    - Medio de transmisión y dispositivos intermedios (Hardware).
    - Procesos intermedios que tratan la información (Software y Hardware).
    - Receptor (Hardware).
    - Destino (Software).
    - Otros: Protocolos (Software), Información, mensaje transmitido (Software).
    - Señal de información, materialización del mensaje sobre el medio.

Fuera del punto de vista sistémico podemos ver un gran número de componentes:

- Computadoras, en el modelo de Internet: Hosts (PCs, laptops, servidores).
- Routers/Switch, Gateway, AP (Access Points).
- NIC (placas de red), Módems.
- Vínculos/enlaces (conformados por cables, fibras ópticas, antenas, etc.).
- Programas: Browsers, Servidores Web, Clientes de Mail, Streaming.

Las componentes de la red deben interactuar y combinarse a través de reglas.

- **Protocolos**

Es un conjunto de conductas y normas a conocer, respetar y cumplir no solo en el medio oficial ya establecido, sino también en el medio social, laboral, etc.
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 un conjunto de reglas que especifican el intercambio de datos y órdenes durante la comunicación entre las entidades que forman parte de una red. Permiten la comunicación y están implementados en las componentes.

La cantidad de componentes de red a interactuar genera complejidad, se requiere una organización de las mismas (Modelos de Organización).

- Modelo en Capas: Layering, divide la complejidad en componentes reusables. Esto reduce la complejidad en componentes más pequeñas. Las capas de abajo ocultan la complejidad a las de arriba (abstracción). Las capas de arriba utilizan servicios de las de abajo (interfaces, similar a APIs). Los cambios en una capa no deberían afectar a las demás si la interfaz se mantiene.
  Facilita el desarrollo, evolución de las componentes de red asegurando interoperabilidad. Facilitad el aprendizaje, diseño y administración de las redes.

- **Modelo OSI (Open System Interconnection)**

Lo crea la ISO (International Estándar Org.) en el año 1984. Está basado en modelos de red (en capas), como DECNET (Digital), SNA (IBM), y TCP/IP (DoD USA – Dept. of Defense USA). Es un modelo de referencia, abierto y estándar, dividido en siete capas.

- Funcionalidad por Capas:
    - Aplicación (7): servicios de red a los usuarios y procesos, aplicaciones.
    - Presentación/Representación (6): formato de los datos.
    - Sesión (5): mantener track de sesiones de la aplicación.
    - Transporte (4): establecer y mantener canal “seguro” end-to-end (applic-to-applic).

        - Red (3): direccionar y rutear los mensajes host-to-host. Comunicar varias redes.
        - Enlace de datos (2): comunicación entre entes directamente conectados. Comunicar una misma red. Acceso al medio.
        - Física (1): transportar la información como señal por el medio físico. Características físicas. Información binaria.

- **Modelo TCP/IP**

Se convirtió en estándar. Es un modelo de cinco capas, aunque por simplicidad algunos hablan de cuatro capaz agrupando a la capa de enlace y capa física en una sola capa que llaman capa de acceso a la red.

    - Capas:
        - De aplicación (Process/Application).
        - De transporte o Host-to-hots.
        - De internet o internetworking.
        - De enlace (link layer).
        - De física.

- **Comparación de los Modelos OSI y TCP/IP**
    - Similitudes:
        - Se dividen en capas.
        - Tienen capas de aplicación, aunque incluyen servicios distintos.
        - Tienen capaz de transportes similares.
        - Tienen capa de red similar pero con distinto nombre.
        - Se supone que la tecnología es de conmutación de paquetes (no de conmutación de circuitos).
        - 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 la capa de enlace de datos y la capa física del modelo OSI en una sola capa.
        - TCP/IP es más simple porque menos capas.
        - Los protocolos TCP/IP son los estándares en torno a los cuales se desarrolló internet, de modo que la credibilidad del modelo se debe en gran parte a sus protocolos.
        - El modelo OSI es un modelo “mas” de referencia, teórico, aunque hay implementaciones.

- **Clasificación de redes**
    - Clasificación por cobertura, distancia, alcance:
        - LAN (Local Area Network): red de cobertura local. Ethernet, Wi-Fi.
        - MAN (Metropolitan Area Network): red de cobertura metropolitana, dentro de una ciudad. MetroEthernet, MPLS, Wi-Max.
        - WAN (Wide Area Network): red de cobertura de área amplia. Geográficamente distribuida. PPP, Frame-Relay, MPLS, HDLC, SONET/SDH.
        - SAN (Storage Area Network): red de almacenamiento. iSCSI, Fibre Channel, ESCON.
        - PAN: red de cobertura personal. Red con alcance de escasos metros para conectar dispositivos cercanos a un individuo. Bluetooth, IrDA, USB.
    - Clasificación públicas y privadas:
        - Internet: red pública global, tecnología TCP/IP.
        - Intranet: red privada que utiliza la tecnología de Internet.
        - Extranet: red privada virtualizada sobre enlaces WAN: Internet. Intranet con acceso de usuarios remotos. VPN (Virtual Private Network) IPSec, PPTP, SSL, OpenVPN. Una intranet mapeada sobre una red pública como internet.
    - Clasificación física de redes:
        - Redes de conmutación de circuitos.
        - Redes de conmutación de tramas/paquetes.

- Servicios orientados a conexión. Circuitos virtuales.
- Servicios No orientados a conexión. Datagramas.

- **¿Qué es internet?**

Internet es una red de redes de computadoras, descentralizada, publica, que ejecutan al conjunto abierto de protocolos (suite) TCP/IP. Integra diferentes protocolos de un nivel más bajo: internetworking.

- Estructura en Jerárquica, en Tiers.
  - Capa de Acceso (EDGE): Acceso Residenciales, Acceso de Organizaciones.
  - Capa de Núcleo (Core): dividida en diferentes niveles.
    - Proveedores Regionales (Regional ISPs).
    - Proveedores Nacionales.
    - Proveedores Internacionales.
    - Proveedores Internacionales en el Tier 1.

Capa APLICACIÓN

Objetivos:

- Aspectos conceptuales y de implementación de los protocolos de aplicación.
  - Modelo de servicio de la capa transporte.
  - Paradigma cliente – servidor.
  - Paradigma peer-to-peer (par-a-par).
- Aprendizaje de protocolos examinando protocolos de aplicación populares.
  - HTTP
  - FTP
  - SMTP / POP3 / IMAP
  - DNS
- Programación de aplicaciones de red.
  - API de socket

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 a través de mensajes.
- Aplicación de red <> Protocolo de capa de aplicación: el protocolo de la capa de aplicación es solo una parte de la aplicación de red.

Los protocolos de la capa de aplicación definen:

- Tipos de mensajes intercambiados, mensajes de requerimiento y respuesta.
- Sintaxis de los tipos de mensajes: que campos del mensaje y como estos son delimitados.
- Semántica de los campos, es decir el significado de la información en los campos.
- Reglas para cuando y como los procesos envían y responden a mensajes.
- Protocolos de dominio público:
  - Definidos en RFCs.
  - Permite inter-operatividad.
  - Eg, HTTP, SMTP.
- Protocolos propietarios:
  - Eg, KaZaA.

- **Arquitecturas de Aplicación**
  - Cliente – Servidor
  - Peer – To – Peer (P2P)
  - Híbridos de cliente-servidor y P2P
- Cliente-Servidor
  - Servidor: Computador siempre on. Direccion IP permanente.

        - Cliente: se comunica con un servidor. Puede ser conectado intermitentemente. Puede tener direcciones IP dinámicas. No se comunican directamente entre sí (dos clientes puros).
    - Arquitectura P2P pura
        - Servidor no siempre on. Sistemas terminales arbitrarios se comunican directamente. Paras se conectan intermitentemente y cambian sus direcciones IP. Ejemplo: Gnutella.
        - Altamente escalable pero difícil de administrar.
    - Híbridos de cliente-servidor y P2P
        - Napster: Transferencia de archivos P2P.
          Búsqueda de archivos centralizada: pares registran contenidos en servidor central. Pares consultan algún servidor central para localizar el contenido.
        - Mensajería instantánea: dialogo entre los usuarios es P2P.
          Detección/localización de presencia es centralizada: usuario registra su direccion IP en un servidor central cuando ingresa al sistema. Usuarios contactan servidor central para encontrar las direccion IP de sus amigos.

- **Procesos que se comunican**
    - Proceso: programa que corre en una máquina.
    - Dentro de la maquina dos procesos se comunican usando comunicación entre proceso (definida por OS). Procesos en diferentes hosts se comunican vía intercambio de mensajes.
    - Proceso Cliente es el proceso que inicia la comunicación (crea y envía mensajes sobre la red).
    - Proceso Servidor es el proceso que espera por ser contactado (recibe los mensajes y, posiblemente responda enviando mensajes).
    - Nota: aplicaciones con arquitectura P2P tienen procesos clientes y procesos servidores.

- **Sockets**
  Los procesos envían/reciben mensajes a/desde sus socket. Los socket son análogos a puertas: proceso transmisor saca mensajes por la puerta. Proceso transmisor confía en la infraestructura de transporte al otro lado de la puerta la cual lleva los mensajes al socket en el proceso receptor.
  API: debemos elegir el protocolo de transporte. Podemos definir algunos parámetros.

- **Direccionamiento de procesos**
  Para que un proceso reciba un mensaje, este debe tener un identificador. Un host tiene una direccion IP única de 32 bits.
  Muchos procesos pueden estar corriendo en el mismo host por lo tanto no es suficiente la direccion IP para identificarlo.
  El identificador incluye la direccion IP y un número de puerta asociado con el proceso en el host.
  Ejemplo de números de puertas: Servidor HTTP: 80. Servidor de Mail: 25.

- **Capa de aplicación: Agentes de usuario**
  Es una interfaz entre el usuario y la aplicación de red.
  Por ejemplo: en la WEB, el agente de usuario es el navegador, el cual permite al usuario visualizar las páginas WEB e interactuar con los elementos de la misma. El navegador es un proceso que envía/recibe mensajes por medio de un socket y además brinda la interfaz al usuario.

- **¿Qué servicios de transporte necesita una aplicación?**
    - Perdida de datos: algunas aplicaciones (audio) pueden tolerar perdida, y otras (transferencia de archivos, telnet) requieren transferencia 100% confiable.
    - Retardo: algunas aplicaciones (telefonía internet, juegos interactivos) requieren bajo retardo para ser “efectivas”.
    - Bandwidth: algunas aplicaciones (multimedia) requieren cantidad mínima de ancho de banda para ser “efectivas”, y otras (“aplicaciones elásticas”) hacen uso del bandwidth que obtengan.

Requerimientos de servicio de transporte de aplicaciones comunes

Aplicación	Pérdidas	Bandwidth	Sensible a Time
file transfer	no	Flexible	no
e-mail	no	Flexible	no
Web documents	no	Flexible	no
real-time audio/video	tolerante	audio: 5kbps-1Mbps video:10kbps-5Mbps	si, 100's msec
stored audio/video	tolerante	Igual al de arriba	si, pocos secs
interactive games	tolerante	Pocos Kbps	si, 100's msec
instant messaging	no	flexible	Si y no

- **Servicios de los protocolos de transporte en Internet**
  - Servicio TCP: Orientado a la conexión acuerdo requerido entre procesos cliente y servidor antes de transferencia. Transporte confiable entre proceso Tx y Rx.
    Control de flujo: Tx no sobrecargara al Rx.
    Control de congestión: frena al Tx cuando la red esta sobrecargada.
    No provee garantías de retardo ni ancho de banda mínimos.
  - Servicio UDP: transferencia de datos no confiable entre proceso Tx y Rx.
    No provee acuerdo entre los procesos, confiabilidad, control de flujo, control de congestión, garantías de retardo o ancho de banda.

Aplicaciones Internet: aplicación, protocolo de transporte

Aplicación	Protocolo capa aplicación	Protocolo de transporte que lo sustenta
e-mail	SMTP [RFC 2821] Telnet [RFC 854]	TCP
remote terminal access	HTTP [RFC 2616]	TCP
Web	FTP [RFC 959]	TCP
file transfer	proprietary	TCP
streaming multimedia	(e.g. RealNetworks) proprietary	TCP or UDP
Internet telephony	(e.g., Dialpad)	typically UDP

Protocolo HTTP y WEB-Cache

- **Historia:** WWW (red de sistemas de hipertexto inter-linkeados accesibles via Internet). Desarrollada en 1990 por Tim Berners-Lee en el CERN (desarrolla el protocolo HTTP y el lenguaje HTML). En 1993 nace el primer cliente/browser GUI: Mosaic. En 1994 aparece Netscape Navigator 1.0.
  También hubo desarrollo de servidores, por ejemplo 1995: Apache Server. Nuevos modelos: Web 2.0, Tim O'Reilly en 2004, modelo de interacción entre usuarios, consumen y producen mediante servicios especiales: blogs, redes sociales, wikis, multimedia, etc.
- **Elementos WEB:** recurso u Objeto HTTP: ej. Web page. Referenciado por una URI (Uniform Resource Id): URL (Uniform Resource Location) o URN (Name). Formato de URL: *protocol://[user:pass@]host:[port]/[path].*
  Ejemplo: *http://www.NN.unlp.edu.ar:8080/dir/index.html*
  URN: no indica ubicación, solo identifican categorías poco imple. Ej. Urn:isbn:0132856204, urn:ietf:rfc:2616.
  Objetos pueden ser página web (web page), imágenes JPEG, PNG, GIF, Java Applet, archivos de multimedia: MP3, AVI, etc.
  Páginas Web: archivo HTML que incluye vínculos o directamente otros objetos.
- **Funcionamiento de HTTP:** Modelo cliente/servidor, Request/Response (sin estados – stateless).

Protocolo corre sobre TCP (requiere 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 (llamados browsers o navegadores): Firefox, IE, Opera, Safari, Chrome.
Servidores: Apache Server, MS IIS, NGINX, Google GWS, Tomcat.

- **Versiones:**
  - **Versión HTTP 0.9:**
    - Fue la primera versión y nunca se estandarizo.
    - Los pasos para obtener un documento eran:
      - Establecer la conexión TCP.
      - HTTP Request vía comando GET.
      - HTTP Response enviando la página requerida.
      - Cerrar la 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.

      Request ::== GET <document-path> <CR><LF>
      Response ::== ASCII chars HTML Document.
      GET /hello.html <CR><LF>
      GET / <CR><LF>
  - **Versión HTTP 1.0:**
    - Versión estándar [RFC-1945].
    - Define formato, proceso 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.
      - Request:
        <Method> <URI> <Version>
        [<Headers Opcionales>]
        <Blank>
        [<Entity Body Opcional>]
        <Blank>
        <Method HTTP 1.0> ::== GET, POST, HEAD, PUT,
        DELETE, LINK, UNLINK
      - Response:
        <HTTP Version> <Status Code> <Reason Phrase>
        [<Headers Opcionales>]
        <Blank>
        [<Entity Body Opcional>]
    - Métodos:
      - GET: obtener el documento requerido. Puede enviar información, pero no demasiada. Es enviada en la URL. Formato ?var1=val1&var2=val2… Limitación de tamaño de URL por parte de las implementaciones.
      - HEAD: idéntico a GET, pero solo requiere la meta información del documento, por ejemplo, su tamaño. Usado por clientes con cache.

- 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 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.

▪ Mediante el parámetro Host se pueden multiplexar varios servicios sobre un mismo host.
▪ PUT, DELETE:
- Ya existían en HTTP 1.0.
- No se implementan directamente en el servidor.
- Se deben agregar como una extensión CGI para manejarlos.
- Debe leer el campo en 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 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 auth. 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 cacheados) y los enviara en texto claro al servidor. El servidor dara o no acceso en base a esos valores. Para los siguientes requerimientos, el navegador usara los valores que tiene almacenados para el Realm solicitado.
▪ En HTTP 1.0 no se contemplaron las conexiones persistentes por default.
▪ A partir de HTTP 1.1 [RFC-2068] sí.
▪ En HTTP 1.0 se pueden solicitar de forma explícita.

o **<u>Versión HTTP 1.1:</u>**
▪ HTTP 1.1, 1997 con la [RFC-2068] se actualiza con [RFC-2616].
▪ Nuevos mensajes HTTP 1.1: OPTIONS, TRACE, CONNECT.
▪ Conexiones persistentes por omisión.
▪ 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 respuesta.
- Sobre la misma conexión se debe mantener el orden de los objetos que se devuelven.
- Se pueden utilizar varios threads para cada conexión.
  - Sin pipelining: 1RTT + FT por cada objeto, n objetos nRTT + nFT.
  - Con pipelining óptico: n objetos: 1RTT + nFT.

▪ TRACE y CONNECT:
- TRACE utilizada para debugging.
- El servidor debe copiar el requerimiento tal cual.
- El cliente puede comparar lo que envía con lo que recibe.
- CONNECT utilizada para generar conexiones a otros servicios montadas con HTTP.
- Proxy-Agent genérico.

- 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 (mejora indexadores).
  - Se puede generar problemas con Cookies.

- **<u>Otras características HTTP</u>:**
  - CGIs Scripts y JavaScript:
    - Necesidad de dinamismo para generar aplicaciones.
    - Server – Side Script:
      - Ejecuta del lado del servidor.
      - CGI (Common Gateway Interface): aplicación que interactúa con un servidor web.
      - CGIs leen de STDIN (POST) o de variables de entorno (GET): QUERY_STRING datos de usuario.
      - Escriben en el STDOUT response.
      - Deben anteponer el content-type en el header.
      - POST permite enviar más datos.
      - Lenguajes de Scripting mas flexibles y seguros: PHP, ASP, JSP.
      - Implementados como CGIs, dentro de propios del web-server.
    - Client – Side Script:
      - Ejecuta del lado del cliente, en el browser.
      - JavaScript estándar W3C.
      - Usan modelo de objetos DOM (Document Objet Model).
      - Otros lenguajes: JScript, VBScript.
      - Permiten extensiones como AJAX (Asynchronous JavaScript And XML).
      - AJAX hace requerimientos particulares y no necesita recargar toda la página.
      - Parseo XML para comunicarse.
      - Existen numerosos frameworks que encapsulan esta funcionalidad brindando una interfaz de programación API fácil de utilizar.
    - Server – Side to Server – Side Script:
      - Permiten comunicación entre servidores.
      - Modelo de “objetos” y servicios distribuidos.
      - Conjunto de convenciones para implementar RMI (Remoto Method Invocation) sobre HTTP (u otro protocolo de texto).
      - Previo XML – RPC.
      - SOAP (Simple Object Access Protocol).
      - Web-Services.
      - REST.
  - Cookies:
    - Mecanismo que permite a las aplicaciones web del servidos “manejar estados”.
    - El cliente hace un request.
    - El servidor retorna un recurso (un objeto HTTP, como una pagina HTML) indicando al cliente que almacene determinados valores por un tiempo.
    - La Cookie es introducida al cliente mediante el mensaje en el header Set-Cookie: mensaje que indica un par (nombre,valor).
    - 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.
    - El servidor puede borrarlo.
    - Esta información puede ser utilizada por Client-Side scripts.
  - HTTPS (HTTP sobre TLS/SSL):

        - Utiliza el port 443 por default.
        - Etapa de negociación previa.
        - Luego se cifra y autentica todo el mensaje HTTP (incluso el header).
- **Web-Cache:**
    - "Proxiar" y Cachear recursos HTTP.
    - Objetivos: mejorar el tiempo de respuesta (reducir retardo en descarga). Ahorro de BW (recursos de la red). Balance de carga, atender a todos los clientes.
    - Se solicita el objeto, si esta 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.
    - Cache del lado del cliente.
    - El web browser tiene sus propias caches locales.
    - Los servidores agregar headers:
        - Last-Modified: date
        - ETag: (entity tag) hash
    - Requerimientos condicionales desde los clientes:
        - If-Modified-Since: date
        - If-None-Match: hash
    - Respuesta de los servidores:
        - 304 Not Modified
        - 200 OK
    - 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.
    - Existen:
        - Proxy no-transparente.
        - Proxy transparente.
        - Proxy en jerarquía o mesh (ICP y HTCP).
        - CDN (Content Delivery Network), funcionan por DNS.
    - Protocolos de comunicación entre web-cache servers.
        - ICP (Internet Cache Protocol).
        - (HTCP) Hyper Text Caching Protocol.
    - Diferentes relaciones: parent, siblings.
    - Protocolo de comunicaciones entre router y web-cache servers.
        - WCCP (Web Cache Control Protocol).
    - En general corren sobre UDP.
- **Version HTTP 2:**
    - Es el reemplazo de como HTTP se transporta. No es un reemplazo de protocolo completo. Se conserva método y semántica. Base del trabajo protocolo desarrollado por Google SPDY/2.
    - Definido en:
        - RFC7540: Hypertext Transfer Protocol versión 2.
        - RFC7540: HPACK-Header Compression for HTTP/2 RFC7541.
    - Problemas con HTTP 1.0 y 1.1
        - Un request por conexión, por vez, muy lento.
        - Alternativas (evitar HOL):
            - Conexiones persistentes y pipelining.
            - Generar conexiones paralelas.
        - Problemas:
            - Pipelining requiere que los responses sean enviados 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).
    - Diferencias principales con HTTP 1.1
        - Protocolo binario en lugar de textual (ASCII), binary framing: (más eficiente).
        - Multiplexa varios request en una petición en lugar de ser una secuencia ordenan y bloqueante.
        - Utilizar una conexión para pedir/traer datos paralelos, agrega: datos fuera de orden, priorización, Flow control por frame.
            - Usa comprensión de encabezado.
            - Permite a los servidores "pushear" datos a los clientes.
            - La mayoría de las implementaciones requieren TLS/SSL, no el estándar.
    - HTTP 2 mux stream, framing:
        - Todos los stream en una misma conexión.
        - Los streams son identificados y divididos en frames.
        - Stream codificados en binario y cada frame header común fijo (9B).
    - HTTP 2 priorización y flow-control:
        - Los streams dentro de una misma conexión tienen Flow-control individual.
        - Los streams pueden tener un weight (prioridad).
        - Los streams pueden estar asociados en 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, por ejemplo, CSS o JavaScript.
        - No siempre es lo que necesita el cliente, depende de que funcionalidad ofrece.
    - Compresión y Soporte:
        - Compresión de encabezados.
        - SDPY/2 propone usar GZIP.
        - GZIP + cifrado, tiene "bugs" utilizados por atacantes.
        - Se crea un nuevo compresor Headers: HPACK.
        - H2 y SPDY, soportados en la mayoría de los navegadores.
    - Otras características:
        - HTTP 1.1, posibilidad de hacer un upgrade durante la conexión: Upgrade Header.
        - Negociar el protocolo de aplicación: ALPN: Application-Layer Protocol Negotiation. Se negocia como extensión SSL en Hello (anteriormente NPN).
        - Posibilidad de negociar protocolo alternativo: Alterantive Service: alt-svc.

Capa de APLICACION

- **Protocolo DNS**
    - Historia:
        - Internet: necesidad de usar nombres en lugar de direcciones IP.
        - Mecanismos para mapear nombre de internet (nombre de dominio) a dirección IP.
        - 1973, archivo globar, HOSTS.TXT, mantenido por el SRI (Standford Research Institute, hoy SRI international).
        - Sistema centralizado: solicitudes de cambio por email. Bajada por FTP.
        - 1980 el servicio era muy difícil de mantener y no escalable.
        - 1983 Paul Mockapetris, de USC, desarrolla DNS (Domain Name System): [RFC-882], [RFC-883].
        - 1984 primeras implementaciones Unix BSD.
        - El servicio ha ido teniendo modificaciones: [RFC-1034], [RFC-1035], etc.
        - Servicio NO utilizado directamente por los usuarios.
    - Aspecto de DNS:

        - DNS cubre los siguientes aspectos:
            - Especifica la sintaxis de los nombres y las reglas para delegar autoridad sobre los nombres.
            - Especifica sistema distribuido para “mapear” nombres con direcciones y otras operaciones.
            - Define la implementación de un protocolo para comunicación de las componentes del sistema.
        - Descentralizar el mecanismo de asignación de nombres, aunque sigue sistema jerárquico.
        - Delegar autoridad y responsabilidad de la asignación y mapeo en organismos intermedios.
        - Podemos verlo como una base de datos distribuida.
    - Elementos de DNS:
        - Nombre de dominio FQDN: lista de etiquetas (labels) separadas por puntos.
        - Se leen desde el nodo/etiqueta izquierda hasta la raíz del árbol (el punto), estructura jerárquica con sub-nombres (niveles).
        - La sintaxis jerárquica refleja la delegación de autoridad.
        - No son case-sensitive, cada etiqueta máxima 63 chars.
        - Máximo etiquitas 127, nombre no mas de 255 chars, acepta valores internacionales, UTF-8, Unicode.
    - TLDs (Top Level Domains)
        - Los TLDs se podrían clasificar en 3 grupos:
            - gTLDs, Generic TLDs: continen dominios con propósitos particulares, de acuerdo a diferentes actividades políticas definidas por el ICANN: Unsponsored TLD, o definidas por otra organización: Sponsored TLD.
            - ccTLD Country-Code TLDs: contienen dominios delegados a los diferentes países del mundo. ISO 3166-1 alfa-2.
            - .ARPA TLD: es un dominio especial, usado internamente para resoluciones de reversos.
    - Organización del DNS:
        - Sistema distribuido y jerarquico.
        - Organización mediante dominios, subdominios y hosts o servicios.
        - IANA a través de ICANN (Internet Corporation for Assigned Names and Numbers) controla el funcionamiento.
        - Existen organizaciones paralelas: Open Root Server Network (ORSN), OpenNIC.
        - Delegación mediante RIRs (Regional Internet Registers):
            - American Registry for Internet Numberts (ARIN).
            - RIPE NCC – Europa y parte de Asia – (RIPE).
            - Asia-Pacific Network Information Centro (APNIC).
            - Latin American and Caribbean NIC (LANIC).
            - African Network Information Centre (AfriNIC).
        - Nombres se delegan a países, direcciones IP no.
    - Funcionamiento de DNS:
        - Tipos de servidores:
            - Servidor Raiz: servidor que delega a todos TLD (Top Level Domains). No debería permitir recursivas.
            - Servidor autoritativo: servidor con una zona o sub-dominio de nombres a cargo. Podría sub delegar.
            - Servidor local: es un servidor que es consultado dentro de una red. Mantiene cache. Puede ser Servidor Autoritativo. Permite recursivas internas.
            - Open Name Servers: servidores de DNS que funcionan como locales para cualquier cliente. Por ejemplo 8.8.8.8, 8.8.4.4, 4.2.2.2, 4.2.2.3
            - Forwarder Name Server: interactúan directamente con el sistema DNS exterior. Son DNS proxies de otros DNS internos.

        - Servidor Primario y Secundario: solo una cuestión de implementación. Donde se modifican los datos realmente.
    - Modelo cliente/servidor, Request/Response.
    - También hay dialogo entre los servidores.
    - Protocolo corre sobre UDP y TCP, puerto 53.
    - El cliente escoge cualquier puerto no privilegiado.
    - No trabaja sobre texto ASCII.
    - Si el mensaje supera los 512 bytes se utiliza TCP, e.g zone transfer.
    - Clientes: resolver + cualquier aplicación que requiera la resolución de nombres.
        - Unix el resolver conjunto de funciones C library (libc).
        - Otras implementaciones Smart Resolver servidor local en cada equipo, caching.
    - Servidores: BIND (Berkeley Internet Name Domain/Daemon) de ISC; UNBOUND.
- Servicios y Registros de DNS:
    - Servidor de DNS almacena la información formando base de datos de RR (Resource Records).
    - No necesariamente es base de datos relacional.
    - Cada registro tiene diferente tipo de información:
        - A, AAAA (Address): nombre → IP, IPv6
        - PTR (Pointer): IP → nombre
        - CNAME (Canonical Name): nombre → nombre
        - HINFO (Hardware Info): nombre → info
        - TXT (Textual): nombre → info
        - MX (Mail Exchanger): nombre-dom → mail exchanger
        - NS (Name Server): nombre-dom → dns server
        - SOA (Start Of Authority): parámetros de dominio

- **Email (Mensajería de internet)**
    - Uno de los primeros servicios de internet.
    - Creado por Ray Tomlinson en 1971.
    - Primeros protocolos de transporte UUCP (Unix-To-Unix Copy).
    - Conexiones via modems.
    - Formato de email: mach2!mach1!user.
    - En 1972: se escoge el símbolo @ para denotar “en”, “at”.
    - Casilla de email: user@fdqn-domain o user@fqdn-server.
    - En 1982 surge SMTP.
    - En 1988: Gusano Morris, primer virus, usaba el sendmail.
    - Arquitectura:
        - Componentes principales:
            - MUA (Mail User Agent).
            - MTA (Mail Transport Agent).
            - MDA (Mail Delivery Agent).
        - Componentes secundarios:
            - Servidor de autenticación.
            - Web Mail (Front-End WWW).
            - Servidor de Anti-Virus, Anti-Spam.
            - Servidores de listas.
        - Arquitectura MUA:
            - Cliente
            - Lector/Emisor Local de correo.
            - Escritura, edición, lectura de mensajes de correo (mails).
            - Ejemplos: Eudora, MS Outlook, Mozilla Thunderbird, elm, pine, mutt.
            - Web-mailers: Horde/IMP, Squirrel, Gmail, Yahoo, Hotmail.

        - Interfaz con el usuario.
        - Poseen un LMTA para comunicarse con el servidor de mail saliente.
        - Utilizan protocolos SMTP, POP, IMAP, habla con MTA y con MDA.
    - Arquitectura MTA:
        - Cliente y Servidor.
        - Se encarga de enviar el mensaje del mail al servidor donde esta almacenada la casilla de mensaje destino.
        - Almacena temporalmente el correo saliente.
        - Se encarga de recibir y almacenar temporalmente los mensajes para las casillas que sirve.
        - Ejemplos: Postfix, Sendmail, MS Exchange, Exim, Qmail.
        - Utiliza protocolo SMTP, entre servidores MTA.
    - Arquitectura MDA:
        - Cliente y Servidor.
        - Se encarga de tomar los mensajes recibidos por el MTA y llevarlos al usuario.
        - Existen MDA locales (LDA) y remotos (rMDA).
        - Ejemplos de LDA: procmail, maildrop, Deliver, Sendmail.
        - Ejemplos de rMDA: Cyrus IMAPd, Courier IMAPd, WU (University of Washington) IMAPd, Dovecot.
        - Utilizan protocolos POP y/o IMAP.
- Protocolo SMTP:
    - Simple Mail Transport Protocol.
    - Protocolo Cliente/Servidor.
    - Utiliza formato ASCII 7 bits en 8 NVT.
    - Usa TCP puerto servidor: 25.
    - Los LMTA de los MUA hablan SMTP con su servidor SMTP saliente.
    - Los servidores SMTP hablan entre si este protocolo.
    - RFC-821 (SMTP), extendida por RFC-1869 (ESMTP) y redefinia por RFC-2821.
    - Requiere finalizar con CRLF.CRLF.
    - Usa conexiones persistentes.
    - Trabaja de forma interactiva (Requerimiento/Respuesta) o Pipeline.
    - Puede o no requerir autenticación.
    - Puede o no trabajar de forma seguera: SSL/TLS.
- Formato del mensaje:
    - El concepto de envelope fue definido en RFC-821.
    - Definido el cuerpo y el encabezado en RFC-822, redefinido en RFC-2822.
        - Envelope (envoltorio), el usuario no lo ve, usado por MTAs, MAIL FROM:, RCPT TO:
        - Header (encabezado), meta información del mail: Subjects:, From:, To:, Return-Path:, X-Mailer:, X-…:
        - Body (cuerpo), separado por línea en blanco del header: contenido del email.
- Formato del mensaje extendido:
    - Extensiones para enviar datos binarios.
    - MIME (Multipurpose Internet Mail Extensions): definido en RFC-1521 y RFC-1522, redefinidas en RFC-2045 y RFC-2046.
    - Con tags especiales indica el tipo al sistema destino.
    - Luego codifica el mensaje binario en formato que no viole US-ASCII (NVT).
    - Algunos ejemplos: uunecode, base64, quotes-strings, etc.
- Formato del mensaje (ejemplo):
    - From: bob@other.test
      To: alice@cities.org
      Subject: Test Mail MIME
      Date: Thu, 11 Aug 2005 20:20:56 -0300 (ART)

MIME-Version: 1.0
Content-Transfer-Encoding: base64
Content-Type: image/jpeg
base64 encoded image.....
.....
..... base64 encoded image

- Protocolos Acceso a Correo:
  - POP: Post Office Protocol, RFC-1939: POPv3.
  - IMAP: Internet Mail Access Protocol, RFC-1730: IMAPv4.
  - Requieren autentificación.
  - Utiliza formato ASCII 7 bits en 8 NVT.
  - Usan TCP puertos servidor: 110 y 143.
  - Permiten correr de forma segura sobre SSL/TLS.
  - IMAP más flexible permite uso de carpetas y manipulación de mensajes en el servidor.
- SPAM:
  - Open Relay.
  - SPAMbots.
  - Métodos para contrarrestarlo:
    - RBL (Realtime Blackhole List) Blacklist.
    - Greylist, Whitelists.
    - Filtros de contenidos.
    - Reglas dirigidas/entrenadas por el usuario.

- **Protocolo FTP**
  - Historia:
    - FTP (File Transfer Protocol) conforma el grupo de los protocolos más viejos de internet aún más utilizados.
    - Propuesta original RFC-114, año 1971, MIT.
    - Existió antes de TCP/IP, ejecutaba sobre NCP.
    - El protocolo original ha sufrido varias modificaciones, adaptado a IP, la esencia es la misma.
    - Tuvo un gran auge con internet comercial, en 1992 era el protocolo que más volumen transportaba (en la actualidad lo supero HTTP).
    - FTP es un protocolo para copiar archivos completos, a diferencia de otros que brindan acceso a archivos: NFS, CIFS.
  - Características:
    - Estandarizado por RFC-765, luego convertido en obsoleto por RFC-959. Actualizado por RFC-2228 y RFC-3659.
    - Mensajes se codifican en ASCII estándar (de 7 bits codificados en 8). (Terminal ASCII NVT-Network Virual Terminal- CRLF).
    - Modelo Cliente/Servidor, command/response.
    - Protocolo corre sobre TCP (requiere protocolo de transporte confiable).
    - Los clientes FTP no requieren interfaz gráfica.
    - Soportado por los browsers/clientes Web mediante la URL: ftp://.... Ejemplos de clientes: WS_FTP de ipswitch, ftp, gFTP, FileZilla.
    - Ejemplo de servidores: BSD ftdp(8) de Unix BSD, WU-FTP, Pure-FTPd, vsFTPd, FileZilla-Server.
  - Funcionamiento:
    - Usa dos conexiones TCP:
      - Conexión de control (Out-Of-Band Control) port 21.
      - Conexión para la transferencia de datos.
    - Cada conexión requiere servicios diferentes:
      - Conexión de control: min delay.
      - Conexión de datos: max throughput.

        - El cliente escoge cualquier puerto no privilegiado, (n > 1023) y genera conexión de control contra el puerto 21 del servidor.
        - El servidor recibe los comandos por dicha conexión y responde/recibe por la conexión de datos aquellos que lo requieran.
        - La conexión de datos se crea y se cierra bajo demanda.
        - El estado de cada operación se transmite por el canal de control.
    - Comandos FTP:
        - RETR: obtener un archivo desde el servidor. A nivel de interfaz de usuario el comando que lo inicia es el get.
        - STOR: envía un archivo al servidor. A nivel de interfaz de usuario el comando que lo inicia es el put.
        - LIST: envía una petición de listar los archivos del directorio actual en el servidor. A nivel de interfaz de usuario el comando que lo inicia es el ls o dir.
        - DELE: comando para borrar un archivo en el servidor.
        - SIZE, STAT: obtiene información de un archivo en el servidor.
        - CD, PWD: cambia de directorio, obtiene el dir. actual.
        - RMD, MKD: borra y crea directorio.
        - HELP: obtener ayuda.
        - MODE: cambiar el modo.
    - Modalidades:
        - FTP Activo (modalidad vieja):
            - Conexión de control: port 21.
            - Conexión de datos: port 20.
            - Se diferencia como maneja la conexión de datos.
            - El servidor de forma activa se conecta al cliente para generar la conexión de datos.
                - PORT h1, h2, h3, h4, p1, p2.
                - PORT 127,0,0,1,4,3 == 127.0.0.1:1027 (4*256)+3
        - FTP Pasivo:
            - Conexión de control: port 21.
            - Conexión de datos: port no privilegiado.
            - El servidor de forma pasiva indica al cliente a que nuevo puerto debe conectarse.
                - PASV
                - 227 h1,h2,h3,h4,p1,p2 e.g. 227 127,0,0,1,4,3
    - Formato de datos (Bytes):
        - FTP tiene funcionalidad de la capa ISO L6 (representación).
        - Debido a los diferentes tipos de plataformas, los archivos puede ser convertidos a diferentes representaciones.
        - Es responsabilidad del cliente indicarle al servidor el tipo/formato, sino el default es ASCII, aunque hoy es más común encontrar image.
        - Los tipos son:
            - ASCII A NVT-ASCII.
            - EBCDIC E EBCDIC Text.
            - IMAGE I Raw binary, serie de bytes.
            - LOCAL L Raw binary, serie de bytes, usando var. byte size.
    - Formato de archivos:
        - Las plataformas (OS) pueden almacenar los archivos en diferentes estructuras.
        - FTP define estructuras para transportar datos.
        - Formato default file (F).
        - Se especifica el formato para transferencia con el comando STRU.
            - File F Unstructured, sequence of bytes.
            - Record R Series of records.
            - Page P Series of data blocks (pages).

- Modo de transferencia:
  - MODE se usa para especificar una codificación adicional aplicada sobre los datos transmitidos, de forma independiente del formato de archivo.
    - Stream S stream of bytes: si es R el formato EOF se pone como registro, si es F el formato, indica el cierre de stream.
    - Block B archivo se envía como header mas secuencia de bloques. Permite interrumpir y reiniciar.
    - Compressed C Datos comprimidos usando RLE: Run Length Encoding. BBBBBBNN = 6B2N.
  - Habitualmente no soportado.
- Alternativas a FTP:
  - Versiones de FTP seguras:
    - FTPS, FTP over SSL/TLS.
  - Versiones integradas con la suite de Open-SSH:
    - SCP (Secure remote Copy).
    - SFTP (Secure FTP).
  - Aplicaciones para compartir recursos:
    - NFS, SMB/CIFS, iSCI, etc.
  - TFTP.
- FTP vs HTTP:

FTP	HTTP
Diseñado para Upload	Se puede con PUT, POST
Soporte de Download	Soporte de Download
Formato ASCII/binary cliente selecciona	meta-data with files, Content-Type, más flexible
No maneja Headers	Manjea Headers, mas información
FTP Command/Response, archivos pequeños más lento	Pipelininig
Más complejo con Firewalls y NAT	Más amigable con Firewalls y NATs
Dos conexiones, y modo Activo o Pasivo	Una conexión, más sencillo
Soportan Ranges/resume	Range/resume HTTP opciones más avanzadas
Soporte de Autenticación	Soporte de Autenticación
Soporte de Cifrado (problemas fwall)	Soporte de Cifrado
Soporte de Compresión RLE	Soporte de Compresión Deflate: LZ77+Huffman

<u>Capa TRANSPORTE</u>

- Protocolos de Transporte: UDP, TCP.
- Introducción:
  - IP provee un servicio “débil”, pero eficiente (best-effort).
  - Para IP los paquetes puede ser descartados, des-ordenados, retardado, duplicados o corrompidos.
  - Paquetes IP solo dirección DST y SRC.
- Características de Transporte:
  - MUX/DEMUX App. to App. (Ports).
  - Soporte de datos de tamaños arbitrarios.
  - Control de errores.
  - ¿Cuándo y cómo una App debe enviar datos?
    - Control de flujo.
    - Control de congestión.
  - Dos modelos:
    - Modelo confiable: TCP.
    - Modelo NO confiable: UDP.
- **UDP (User Datagram Protocol) RFC-768**:
  - Protocolo minimalista. Menor overhead.
  - Características de IP: best-effort.
  - Orientados a Packets/Datagramas.
  - PDU: Datagrama (por coherencia con nivel transporte se suele llamar segmento).
  - Solo provee MUX/DEMUX.
  - No incrementa Overhead ent-to-end.

    - No requiere establecimiento de conexión.
    - Servicio FDX.
    - Aplicaciones video/voz streaming/TFTP/DNS/Bcast/Mcast.
- **TCP (Transport Control Protocol) RFC-793**:
    - Protocolo confiable, ordenado, buffering, control de flujo y de congestión.
    - Orientado a Streams.
    - PDU: segmento.
    - Provee MUX/DEMUX.
    - Incrementa Overhead end-to-end a costa de confiabilidad.
    - Requiere establecimiento de conexión (y cierre).
    - Servicio FDX.
    - Aplicaciones FTP/HTTP/SMTP/acceso remoto/Unicast.
- Headers/Encabezados:
    - El encabezado IP provee: ruteo, fragmentación, detección de algunos errores.
    - El encabezado UDP provee: MUX/DEMUX, detección de errores (no obligatorio).
    - El encabezado TCP provee: MUX/DEMUX, detección de errores, sesiones, control de flujo y control de congestión.
- Datagrama (UDP):
    - Puertos: MUX/DEMUX.
    - Longitud: UDP HDR + Payload.
    - Checksum:
        - Calculo Ca1, Opcional. 0 = sin checksum.
        - Calculo HDR + PseudoHDR + Payload.
        - PseudoHDR: IP.SRC + IP.DST + Zero + IP.Proto + UDP.LENGTH.
        - PseudoHDR: protección contra paquetes mal enrutados.
        - Aplicaciones de LAN por eficiencia lo podrían deshabilitar.
        - Si tiene error se descarta silenciosamente.
- Segmento (TCP):
    - Puertos: MUX/DEMUX.
    - No tiene longitud total, si de HDR LEN (variable, max 60B Unit=4B).
    - Total LEN se computa para PseudoHDR, no viaja en el segmento.
    - Checksum:
        - Calculo Ca1, obligatorio.
        - Calculado de forma igual que UDP.
        - Si tiene error podría pedir retransmisión, implementación de TCP descarta y espera TMOUT.
    - Necesidad de manejar timers, TMOUT por cada segmento. (implementaciones lo manejan más eficiente).
    - Campos de sesiones: FLAGs: Syn, Fin.
    - Máquina de estado finita por cada conexión.
    - Campos de control de flujo: ACK, Seq, Ack Seq, Win.
    - Permite opciones y negociación.
    - TCP entrega y envía los datos agrupados o separados de forma disasociada de la aplicación:
        - La aplicación puede enviar 300 bytes en un write y TCP lo podría enviar en 3 segmentos separados por 100 bytes cada uno.
        - La aplicación puede enviar 100 bytes y luego otros 200 y TCP esperar para enviarlos todos juntos.
        - La aplicación puede intentar leer 200 bytes del buffer y TCP solo entregar 150 bytes y luego el resto.
- TCP establecimiento de conexión:
    - 3Way-Handsake (3WH).
    - En el 3 segmento se puede enviar info.
    - El ISN debe ser un contado que se incrementa cada 4 mseg.
    - RST si no hay proceso en estado LISTEN.

    - Open pasivo y activo.
    - Open simultaneo.
- TCP cierre de conexión:
    - 4Way-Close (4WC).
    - Posibilidad de Half-Close.
    - Podría cerrarse en 3WC.
    - Espera en TIME_WAIT, 2MSL (aprox 2*2min).
    - Evitar con SO_REUSEADDR.
    - Cierre incorrecto con RST.
    - Close simultaneo.
- Otros campos TCP:
    - TCP entrega y envía los datos agrupados o separados de forma disasociada de la aplicación.
    - Datos urgentes: URG
        - Urgent Pointer valido su URG=1.
        - Indica: offset positivo + Seq Num = last Data Urgent byte.
        - Indicar a la App. datos urgentes, debe leer.
        - Debería combinarse con PSH. Habitualmente llamado OOB data (TCP no soporta OOB).
    - Pushear datos: PSH
        - Fuerza a TCP a pasar datos de la App.
        - No lo deja "bufferear" los datos recibidos (input).
- Opciones TCP:
    - Maximum Segment Size (MSS), recomendado 536B, RFc-879.
    - Window Scaling.
    - Selective Acknowledgements (SACK).
    - Timestamps.
    - NOP.

- **<u>Protocolo de transporte TCP:</u>**
    - Servicios de TCP:
        - Control de errores:
            - Mecanismo protocolar que permite ordenar los segmentos que llegan fuera de orden y recuperarse mediante solicitudes y/o retransmisiones de aquellos segmentos perdidos o con errores.
            - Se realiza por cada conexión: End-to-End, App-to-App.
        - Control de flujo (Flow-Control):
            - Mecanismo protocolar que permite al receptor controlar la tasa a la que le envía datos el transmisor.
            - Control cuando puede enviar una aplicación sabiendo que la receptora tiene capacidad de recibirlo.
            - Se realiza por cada conexión: End-to-End, App-to-App.
            - Permite que aplicaciones con diferentes capacidades dialoguen regulando la velocidad, tasa de transmisión.
            - Tiene en cuenta el estado del receptor y no el de la red.
        - Control de gestión:
            - Se realiza por cada conexión: End-to-End, App-to-App.
            - Permite que aplicaciones no saturen la capacidad de la red.
            - Tiene en cuenta el estado de la red.
        - Para realizar control de errores y control de flujo se utilizan técnicas ARQ.
        - ARQ solo no hace control de flujo, requiere de otros mecanismos como RNR o Dynamic Window.
        - La capacidad de envío será MIN (congestión, flujo, errores).
    - Técnicas de ARQ (Automatic Repeat reQuest):

        - Stop-and-Wait (poco eficiente).
        - Ventana deslizante (Sliding Window).
            - Ventana estática: capacidad fijada en el emisor. Usado en L2 (Static Go Back-N). (las técnicas estáticas realizan control de perdidas, NO control de flujo por si solas, requiere mensajes de control extra para indicar sobre carga).
            - Ventana dinámica: capacidad anunciada por el receptor (Dynamic Go Back-N). Sirve para hacer control de flujo.
        - Ventana selectiva (Selective Sliding Window) / SR (Selective Repeat).
    - Stop-and-Wait:
        - No se envía el próximo mensaje hasta que no se confirma el que se envió.
        - Sistema simple y poco eficiente: no optimiza producto: Delay, Bandwidth: D x B, D = RTT.
        - Ventana de tamaño K = 1.
        - Cada vez que envía un segmento requiere arrancar un timer: RTO o T1.
        - Si no se recibe confirmación se vence el timer y retransmite.
        - Para ser libre de errores requiere un bit de secuencia 0..1: de formar de no confundir datos re-enviados de nuevo cuando se pierden ACK.
    - Go-Back-N estático:
        - Se tiene una ventana estática de tamaño K = n, n > 1.
        - Se puede enviar n mensajes, segmentos sin recibir confirmación.
        - Por cada mensaje enviado se inicia un timer de retransmisión: T1 o RTO.
        - Más eficiente, si llena el pipe.
        - Por cada confirmación se descarta/reinicia el timer RTO. Si no se recibe confirmación vence RTO (Timeout) y se retransmite.
        - Podría generarse confirmaciones negativas: NAK (No Acknowledge).
        - Se requiere numerar los mensajes, segmentos, más de un bit. Se realiza en modulo M, K <= (M – 1).
        - El receptor puede usar: timer ACK, T2, para confirmar. T1 > T2, T1 > T2 + RTT.
        - Se puede aprovechar tramas de datos para confirmar: Piggy-backing.
        - Se puede confirmar desde N hacia atrás (ACK acumulativos). No necesariamente se confirma individualmente.
        - Si se reenvía se hace desde N hacia adelante, los que ya se enviaron.
        - T2 debe aprovechar Piggy-backing, y confirmaciones acumulativas pero sin demorar demasiado tiempo el flujo de datos.
        - Permite llenar el pipe y mantener el throughput, si se pierde uno se debe vaciar el pipe y volver atrás: Go-Back.
        - No admite segmentos fuera de orden, ni confirmaciones fuera de orden.
    - Selective Repeat (SR):
        - Go-Back-N ante perdidas retransmite segmentos innecesarios si se perdió uno del medio del stream de datos.
        - Selective Sliding Window/Selective Repeat solo retransmite los que no se confirmaron.
        - El receptor puede confirmar de a uno o usar bit vectors/intervalos de confirmaciones.
        - No se debe confundir los segmentos de diferentes ráfagas. No se debe reusar #SEQ hasta asegurarse que tiene todos los mensajes previos o estos no están en la red.
        - Se realiza en modulo M, K <= (M – ½), para evitar confundir las ACK de segmentos.
        - La ventana se desliza sin dejar huecos, desde los confirmados más viejos.
    - <u>Control de errores</u> en TCP:
        - TCP hace el control de flujo por bytes (byte oriented), y no por segmentos.
        - Los segmentos se enumeran de acuerdo a bytes enviados (número del primer byte).
        - Las confirmaciones son “anticipativas”, indican el número del byte que esperan.
        - Utiliza Go-Back-N con ventana dinámica (flow-control), utiliza piggy-backing y permite negociar Ventana Selectiva con opciones.
        - TCP utiliza los campos: #SEQ, #ACK, flag ACK.

- Por cada segmento TCP (con datos) que envía TCP inicia un timer local, RTO (TMOUT) y pone copia del segmento en cola local (RFC-793) TxBuf.
- Por cada segmento ACKed descarta el timer asociado y descarta la copia del segmento (RFC-793) del TxBuf. Permite hacer lugar para nuevos segmentos Tx.
- Si RTO expira antes que se confirme el segmento TCP lo copa del TxBuf y retransmite (RFC-793).
- Segmentos ACKed no indica leído por aplicación, si recibido por TCP (RFC-793) (ubicado en el RxBuf del receptor).
- Si el receptor detecta error en el segmento simplemente descarta y espera que expire RTO en el emisor (podría enviar un NAK, reenviar ACK para el ultimo recibido en orden, forma de solicitar lo que falta).
- Receptor con segmentos fuera de orden descarta directamente y podrá reenviar ACK (podría dejar en RxBuf pero no entregar a la aplicación, tiene huecos).
- Se puede confirmar con ACK acumulativos.
- TCP no arranca un RTO por cada segmento, solo mantiene uno por el más viejo enviado y no ACKed y arranca uno nuevo solo si no hay RTO activo.
- Si se confirman (ACKed) datos, se inicia un nuevo RTO (RFC-6298) recomendado.
- El nuevo RTO le está dando más tiempo al segmento más viejo aun no confirmado.
- Si se vence un RTO se debe retransmitir el segmento más viejo no ACKed y se debe doblar: Back-Off timer RTO = RTO * 2 RTOmax = 60s (RFC-6298) recomendado.

- Calculo de RTO:
  - Debe ser dinámico, debe contemplar estado de la red.
  - Estático solo sirve para L2 (directamente conectados).
  - Para calcular RTO se estima RTT (Round Trip Time). RTT inicial RFC-2988(2000), 3seg – RFC-6298(2011), 1seg. Cambio en las redes.
  - RTO = SRTT + (4 * DevRTT). (RFC-6298).
  - SRTTsubi = (1 – α) * SRTTsubi-1 + α * RTT, α = 1/8.
  - Influencia de las muestras pasadas decrece exponencialmente.
  - DevRTTsubi = (1 – β) * DevRTTsubi – 1 + β * |RTT – SRTTsubi|, β = ¼.
  - Si hay gran variación en SRTTsubi se usa un mayor margen.
  - RTO < 1seg: redondeando a 1 seg (RFC-6298).
  - Se mide por cada RTT. Se puede utilizar la opción TimeStamp.
- TimeStamp (RFC-1323):
  - Se envía en el primer SYN el timeStamp local.
  - En cada mensaje TCP con esta opción, se copia el timeStamp local y se hace el hecho de último timeStamp recibido desde otro extremo.
  - Con el valor recibido como hecho y el valor del reloj local se calcula el RTT.
  - Si el mensaje no es un ACK valido no se actualiza la estimación del RTT SRTT.
  - Relaja la necesidad de usar timer por cada segmento para estimar RTT.
- Control de flujo TCP:
  - Por cada segmento que envía indica el tamaño del buffer de recepción RxBuf (mbufs). Cada conexión mantiene su propio (buffer) en espacio del kernel (TCP): WIN (Ventana).
  - WIN (Ventana) indica la cantidad de datos que el emisor le puede enviar sin esperar confirmación (mejora notablemente contra Stop & Wait).
  - La ventana de recepción de cada extremo es independiente.
  - Cada vez que llega un segmento es puesto por TCP en el RxBuf, TCP lo debe confirmar.
  - Cada vez que la aplicación lee se hace espacio en el RxBuf. Se va modificando el tamaño de la ventana.