Teóricos - Redes y Sistemas Distribuidos 2021

ÍNDICE:

Elaboración: El siguiente resumen se hizo en colaboración de los siguientes boluditos:

- Pepi
- Valen
- Sofi
- Vene
- Law
- Marquidios
- Migue
- Tomi, a veces

Se recomienda discreción al leerlo, ningún miembro del equipo se hace responsable de los efectos secundarios que puede traer el aprendizaje de esta materia. Y sobre todo, tengan en cuenta que pueden experimentar momentos muy intensos aprendiendo dichos temas. Para más información, contactar con su profesor de preferencia, Juan 1 o Juan 2.

Seguinos en el Zulip para más consejos. (PEGI 18)

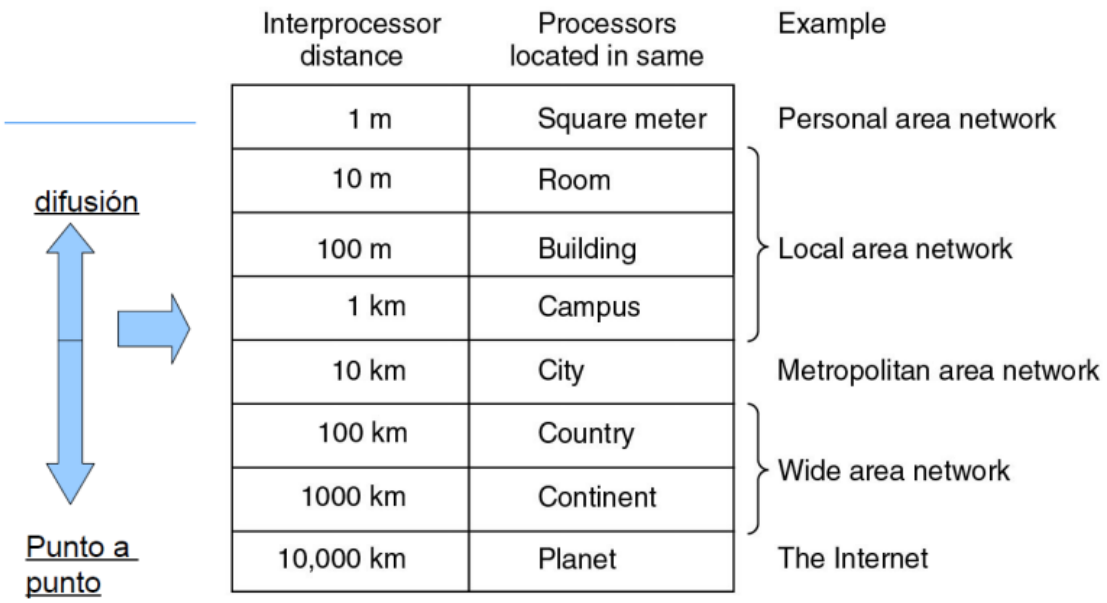
Introducción

Redes de computadoras
Conjunto de sistemas finales **interconectados**.
Dos hosts están **interconectados** si pueden intercambiar información entre ellos.
La **interconexión** se hace por medios de transmisión como cables, ondas, fibra óptica, etc.
Tipos de máquinas para **interconectar** por medio de redes:

- Hosts o sistemas finales (PCs, notebooks, smartphones, etc)
- Dispositivos IoT, que pueden:
    - Intercambiar datos con otros dispositivos interconectados.
    - Recolectar datos de otros dispositivos y procesarlos localmente.
    - Realizar tareas localmente y otras tareas dentro de la infraestructura de la red.

El intercambio de información entre hosts se hace por medio de señales que viajan en los medios de transmisión.

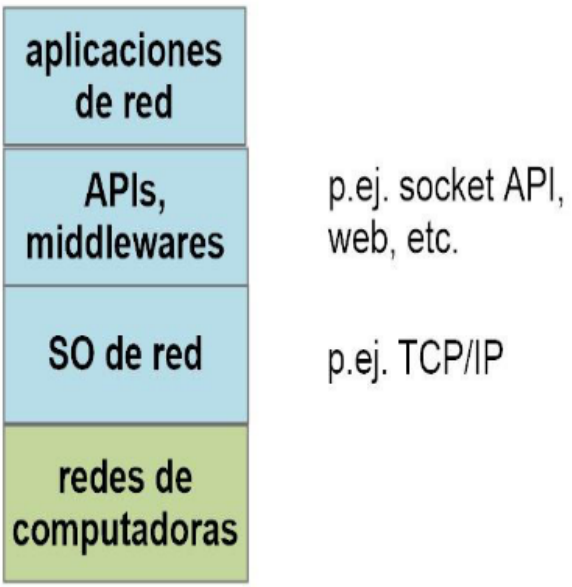
Tipos de redes



Interprocessor distance	Processors located in same	Example
1 m	Square meter	Personal area network
10 m	Room	Local area network
100 m	Building	
1 km	Campus	
10 km	City	Metropolitan area network
100 km	Country	Wide area network
1000 km	Continent	
10,000 km	Planet	The Internet

Para comunicar redes entre sí, se utilizan interredes (por ejemplo internet). Una **interred** es un conjunto de redes **interconectadas**. Las **puertas de enlace** conectan redes de distintas tecnologías.
Las redes de computadoras se usan para proveer **servicios**. Para ello, se crean **aplicaciones de red** (programadas mediante **APIs** o **Middlewares**), que se ejecutan en la internet. Para envío y recepción de mensajes se utilizan **protocolos**.

Los hosts acceden a la internet a través de proveedores de servicios de internet, para que dos hosts que están conectados a diferentes ISP se comuniquen entre sí, estas 2 ISP deben estar **interconectadas**.
Problema: Dados miles de ISP de acceso, ¿cómo conectarlos entre sí?
Solución: tener ISPs globales de tránsito que conectan los ISP de acceso.



Las ISP de acceso son interconectadas a través de redes ISP nacionales e internacionales de más alto nivel llamados ISPs de capa superior o globales de tránsito

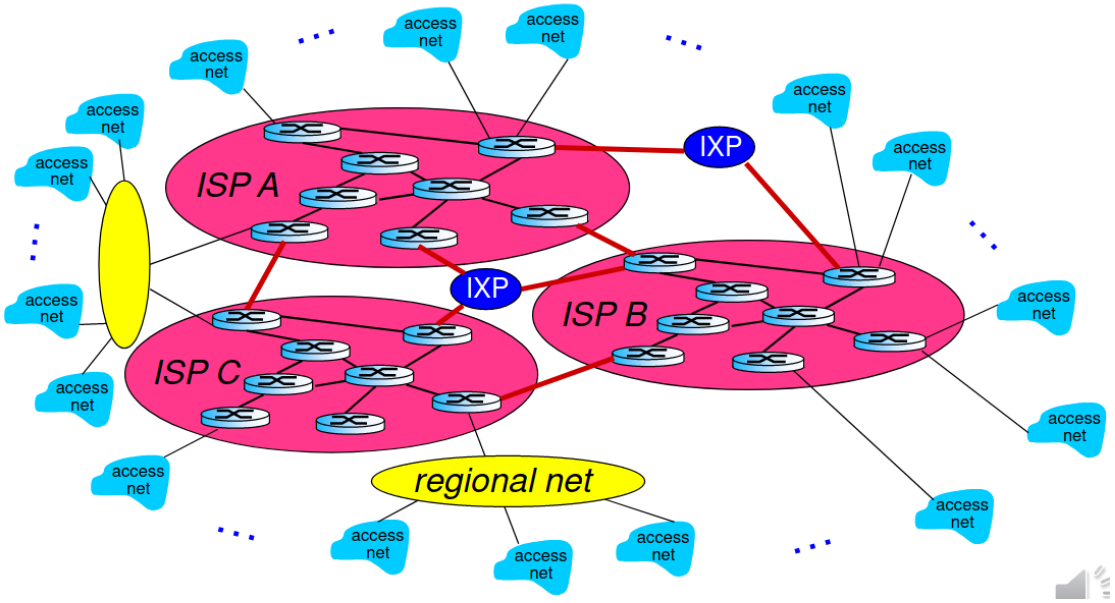
- Las ISP globales de tránsito deben estar interconectadas entre sí.
- Cada red ISP, ya sea de acceso o de capa superior, es manejada independientemente

Problema: Los ISP globales no tienen presencia en cada ciudad o región del mundo, esto implica que hay ISPs de acceso que no se pueden conectar a ISP globales.

Solución, en una región puede haber un ISP regional al cual se conectan los ISP de acceso en la región.

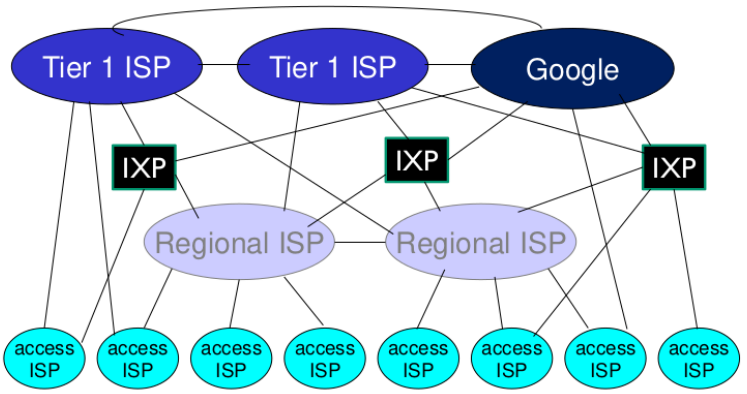
¿Cuáles son las consecuencias de la solución anterior?

- Cada ISP regional se conecta con ISPs globales de tránsito
- Los ISP de acceso pagan al ISP regional al cual se conectan, y cada ISP regional paga al ISP global de tránsito al cual se conecta.
- En algunos lugares un ISP regional; puede cubrir un país entero y a ese ISP regional se conectan otros ISP regionales.



▪"tier-1" ISPs comerciales(p.ej. redes globales de tránsito) cobertura nacional e internacional.

▪Redes proveedoras de contenido
▪En el medio ISP regionales.
▪Finalmente ISPs de acceso.
IXP = internet exchange point



Redes de área amplia (WAN)

Una WAN cubre un área geográfica grande, como país o continente.
Una WAN se organiza en: Subred: varios enrutadores conectados entre sí forman un grafo
•Un arco representa cable que une 2 enrutadores.
•A una subred pueden estar conectadas computadoras o LAN enteras.
•Para ir de una máquina a otra hay distintas rutas alternativas.

Para enviar mensajes en una WAN se utilizan algoritmos de **almacenamiento y reenvío**:

- El paquete sigue una ruta de enrutadores
- Se almacena en cada enrutador de la ruta
- Espera allí hasta que la línea de salida requerida esté libre y se reenvía al siguiente enrutador
- Los paquetes se pueden perder si se llena el buffer

Toma ***L/R*** segundos transmitir un paquete de L-bit en un enlace de R bps.
Demora de almacenamiento y reenvío:
$d_{nodal} = d_{proc} + d_{queue} + d_{trans} + d_{prop}$
d_{queue} es la demora por encolado y depende de la congestión.
d_{proc} es el procesamiento del nodo (chequeo de errores y determinar línea de salida)
d_{trans} es la demora por transmisión.
d_{prop} es la demora por propagación.

Redes de área metropolitana (MAN)

- Redes de cable: red TV por cable
  - Cable coaxial para unir varias casas.
  - Elementos de conmutación unidos por fibra óptica para comunicar viviendas con distintos coaxiales.
  - Asimétricas (más bajada que subida de datos).
- Redes móviles: Redes inalámbricas de alta velocidad.

Redes de área Local (LAN)

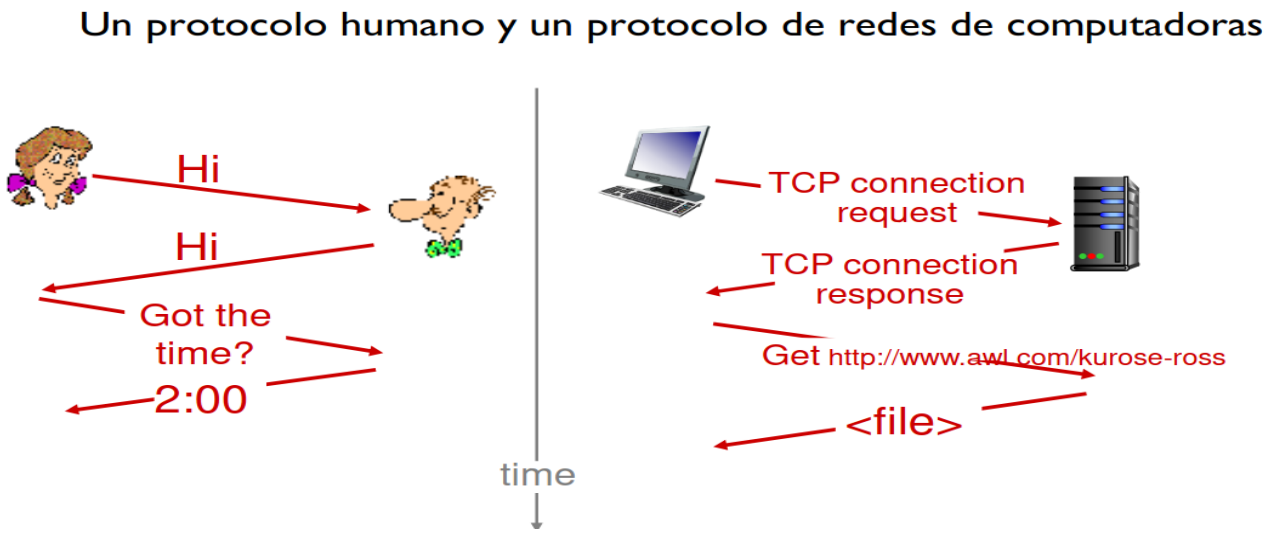
Una red de área local (LAN) es una red operada privadamente dentro de un edificio o casa.
Las LAN usadas por compañías se llaman redes empresariales.

- **Inalámbricas**: Máquinas comunicadas sin cable por medio de un access point
- **Ethernet**: Las máquinas se comunican entre sí por cables a un conmutador (switch).

Si una máquina envía un mensaje, todas las demás lo reciben.
Situación indeseable: Se envían mensajes en una red de difusión y se pierden.
Causa: **Colisión**: más de una máquina manda simultáneamente un mensaje.

– Los mensajes colisionan y se dañan.

Protocolos:



Los sistemas operativos de red consisten de varios protocolos de comunicación. Estos definen: formato, orden de mensajes enviados y recibidos entre máquinas de la red, y acciones tomadas en la transmisión y recepción de mensajes.

Jerarquías de protocolos:

Los **sistemas operativos** de redes (SOR) están organizadas como una pila de capas o niveles, cada una construida arriba de la que está debajo de ella.

- La cantidad de capas, los nombres de las capas, sus contenidos y su función, difieren de un tipo de red a otro.

Las pilas de capas se usan para reducir la complejidad del diseño de los SOR.

¿Cuál es el propósito de una capa en una arquitectura multicapa?

1. Ofrecer ciertos servicios a las capas superiores
2. Ocultar la implementación a las capas superiores

Interfaces entre capas = operaciones y servicios primitivos ofrecidos por una capa inferior a una capa superior.

Una capa n se piensa como una conversación entre la capa n de una máquina con la capa n de otra máquina, sin tener que preocuparnos de ciertos problemas que resuelven las capas inferiores a la capa n. Para especificar cómo es esta conversación se definen protocolos

arquitectura de red = conjunto de capas y protocolos = pila de protocolos.

Comprender problemas de diseño a resolver en distintas capas:

- Problema: Hace falta un mecanismo para identificar a las máquinas de una red.
- Solución: Se usan direcciones para las máquinas.

Control de flujo: *Sucede cuando un emisor muy rápido quiere saturar a un receptor muy lento*

Situación indeseable: mensajes que llegan al receptor se pierden

Causa:un emisor rápido satura de datos al receptor hasta que este ya no puede almacenar más datos que le llegan y comienza a perder datos.

Solución:Uso de retroalimentación al emisor. Es decir, indicarle cuándo y cuánto puede enviar.

Fragmentación de mensajes: *Cuando entre capas se quieren enviar mensajes estos deben dividirse en pequeños paquetes para no sobrepasar su* ***tamaño máximo*** *que normalmente imponen las capas, estos paquetes luego se vuelven a ensamblar cuando llegan a su destino.*
Situación indeseable: mensajes que llegan no pueden ser aceptados en una capa.
Causa: los procesos son incapaces de aceptar mensajes que superan una cierta longitud
Idea de solución: fragmentar mensajes, transmitir fragmentos y re-ensamblar mensajes.

Congestión: *Cuando un host quiere sobrecargar la red de paquetes y esto puede causar que se pierdan u ocasione demora en la llegada al host destino*
Situación indeseable: los mensajes enviados de host de origen a destino se pierden antes de llegar o demoran demasiado en llegar.
Idea de solución: que máquinas emisoras se enteren de la congestión y reduzcan el tráfico de salida.

Distintos tipos de capas:

Capa de aplicación

En la capa de aplicación tenemos las aplicaciones de red.
Hay dos opciones para desarrollar aplicaciones de red:

1. El programador para especificar la comunicación usa una interfaz para programas de aplicación(API).
2. El programador se apoya en middlewares para construir la aplicación red.

Capa de transporte

La CT se ejecuta por completo en los hosts.
¿Qué cosas se debería solucionar la CT?

- Uso de temporizadores y las retransmisiones de paquetes.
- Uso de búferes y control de flujo.
- Evitar congestionar la red poniendo demasiados paquetes en ella

La capa de transporte tiene dos protocolos:

- **TCP (Transfer Control Protocol)**
- **UDP (User Datagram Protocol)**

Capa de Red

- Algoritmos de almacenamiento y reenvío
- Control De Congestión.
- Resolver problemas que surgen cuando un mensaje tiene que viajar por redes de distinta tecnología para llegar a destino.

Situación indeseable: un mensaje demora demasiado en llegar
Causa: en determinadas redes (p.ej.WAN,internet,etc.) hay múltiples rutas entre el origen y el destino y justo se toma una ruta demasiado lenta o larga entre origen destino
De solucionar esto se encargan los algoritmos de enrutamiento.
Los mensajes viajan a su destino de forma independiente esto significa que pueden llegar en un orden diferente al que fueron enviados.

¿Cómo se distingue entre diferentes máquinas (que tienen una conexión a internet)?
Direcciones IP: 4 números entre 0 y 255 separados por '.' .

Para crear la ruta de los paquetes se usan protocolos de enrutamiento: se usan OSPF y BGP para enrutamiento de paquetes.

Comunicación de procesos:
Procesos: programas ejecutándose dentro de un host.
Proceso cliente: proceso que inicia la comunicación.
Proceso servidor: proceso que espera ser contactado
Los procesos en diferentes hosts se comunican intercambiando mensajes.
¿Cómo se identifican los procesos? Mediante direcciones IP y número de puerto.

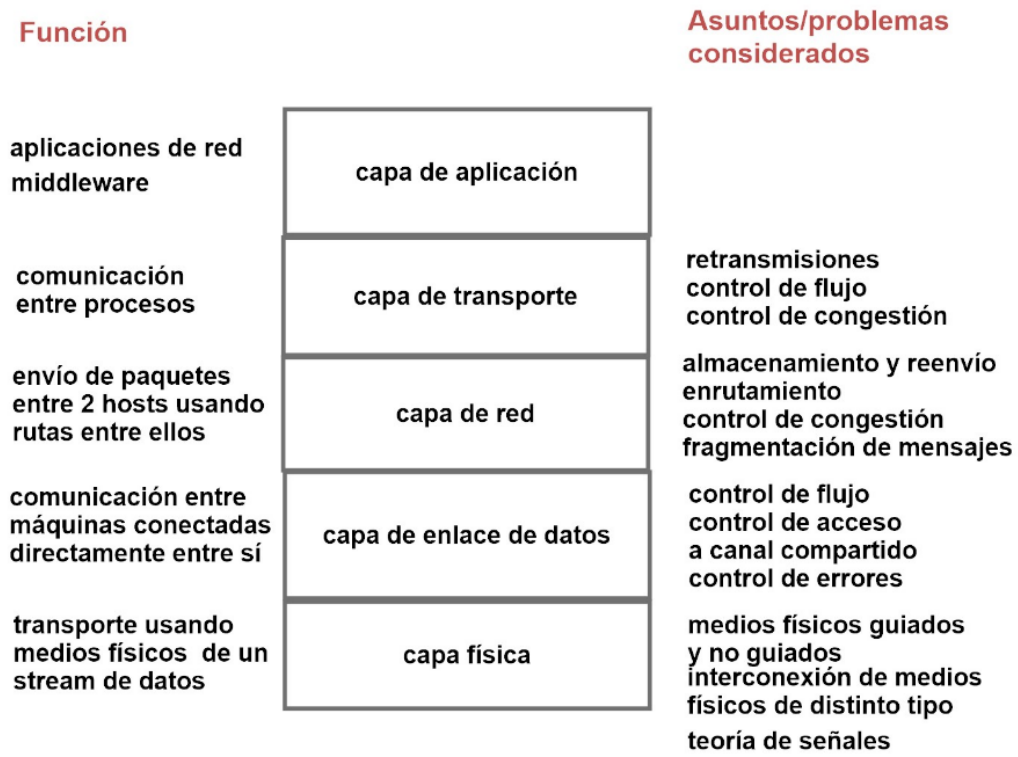
Capa de enlace de datos

Su propósito es transformar un medio de transmisión puro en una línea de comunicación que aparezca libre de errores de transmisión.
Situación indeseable: mensajes llegan con errores
Causa: medio físico de comunicaciones es imperfecto y ocasiona errores

Capa Física

Transportar un stream de datos de una máquina a otra usando medios físicos.
Enlace físico: lo que yace entre el transmisor y receptor.
Medios guiados: Las señales se propagan en medios sólidos: copper, fiber, coaxial.
Medios no guiados : Las señales se propagan libremente, e.j., radio.

Modelo Híbrido:



Capa de Aplicación

En esta capa se encuentran las aplicaciones de red, cada una ofrece un servicio específico, con su propia forma de interfaz con el usuario.
La capa de aplicación se encuentra sobre la capa de transporte por lo cual usa los servicios de dicha capa.

Opciones de desarrollo

- Interfaz para programas de aplicación (API): Conjunto básico de funciones. Por ejemplo socket API para software que se comunica sobre la internet.
- Middleware: Provee servicios al software de la aplicación. Es una función mucho más amplia, está autocontenida y es un software que corre de manera autónoma a parte de la aplicación que se esté usando.

Arquitecturas

- **Cliente – Servidor:** un proceso cliente y un proceso servidor en distintas máquinas se comunican
  - Cliente manda solicitud al servidor
  - Cliente espera respuesta
  - Servidor recibe y procesa solicitud
  - Servidor envía respuesta al cliente
- **Peer-to-peer (P2P):** Por ejemplo bittorrent para distribución de archivos.
  - Hosts arbitrarios (compañeros/peers) se comunican directamente entre sí.
  - Mínimo o ningún apoyo en servidores.
  - Peers piden servicio de otros peers y proveen servicios en retorno
  - Nuevos peers traen nueva capacidad al servicio (+peers = +capacidad)
  - Los peers se conectan intermitentemente y cambian las direcciones IP

Características\Arquitecturas	**Cliente - Servidor**	**Peer - To – Peer (P2P)**
Servidor	Siempre presente (IP fija)	Mínimo o sin servidor
Comunicación con clientes	Nula	Constantemente
Conexión clientes	Intermitente	Intermitente
Bajada de datos	Pedido al servidor	Pedido a los demás clientes
Tiempo de Distribución	D = Max{NF/Us, F/Dmin}	D = Max{F/Us, F/Dmin, NF/(Us+ SumUi)}

N = Número de clientes; Us = Subida servidor; Ui = Subida compañero i; F = Tamaño archivo; Di = Descarga del compañero i; Dmin = descarga más rápida de un cliente (Min {D1,D2,…,Dn})

Cliente-Servidor

UDP	**TCP**
Cliente crea datagrama con IP y puerto del servidor y los envía al servidor	Se ejecuta el servidor
Si llega, servidor lee datagrama	Servidor espera pedidos
Servidor envía respuesta especificando dirección y puerto cliente	Cliente requiere pedido
Si, llega cliente lee datagrama	Servidor acepta conexión
FIN DE LA CONEXIÓN	Cliente envía pedido (*)
	Servidor lee pedido y envía respuesta
	Cliente lee respuesta
	Cliente o vuelve a (*) o cierra conexión.
	Servidor cierra conexión

BitTorrent

- Trozos de 256 Kb
- Compañeros envían y reciben trozos
- Tracker: lleva pista de los compañeros participando en torrent
- Torrent: grupo de compañeros intercambian trozos de archivos

Cuando un compañero se une a torrent:

- No tiene trozos pero va a acumularlos a lo largo del tiempo
- Se registra con tracker para obtener la lista de compañeros
- Se conecta con un subconjunto de compañeros llamado vecinos
- Un compañero avisa periódicamente a tracker que está en BitTorrent

Pedir trozos:

- Diferentes compañeros tienen diferentes subconjunto de trozos de archivos
- Periódicamente pedir a cada compañero la lista de trozos disponibles → conviene pedir primero los trozos menos comunes

Enviar trozos:

- Se envían trozos a los top 4 mejores subidores del compañero (los que tienen mayor velocidad de envío) → el top 4 se reevalúa cada 10 segundos
- Cada 30 segundos se elige al azar otro compañero y comienza a enviarle trozos

Protocolos

Cosas a definir en un protocolo de Aplicación

Tipo de mensaje	Pedido o respuesta
Sintaxis del mensaje	Qué campos y como están delineados
Semántica del mensaje	Significado de los campos
Reglas	Cuándo y cómo los procesos envían y responden
Estados de mensajes	En qué consiste y cómo se mantiene
Tipos de protocolos	Abiertos o Propietarios

Protocolo FTP

- Usado en transferencia de archivos hacia/desde host remoto.
- Servidor FTP: Puerto 21.
- Permite mensajes de control textuales e inspeccionar carpetas.
- Tipos de mensajes: Comando, respuesta y datos.

Reglas de FTP:

1. Cliente FTP contacta con Servidor FTP en puerto 21, usando TCP
2. Cliente es autorizado en la conexión de control
3. Cliente inspecciona el directorio remoto, envía comandos sobre la conexión de control → comienza con identificación de usuario y contraseña.
4. Servidor recibe comando y comienza con transferencia de archivo → abre una segunda conexión de datos TCP (puerto 20).
5. Cuando termina la transferencia, el servidor cierra la conexión de datos

El servidor FTP mantiene el “estado”: directorio corriente o autenticación previa.

Web

Páginas Web

Pueden contener vínculos a otras páginas. Suelen contener texto. Suelen referenciar a varios objetos (HTML, imágenes, etc).
Las páginas/objetos se nombran usando URLs (localizadores uniformes de recursos).
Partes de una URL:

- Nombre del protocolo
- Nombre DNS de host que contiene la página
- Camino al archivo → nombre del archivo que contiene a la página

Si cambia el nombre de una máquina, la IP del dominio no va a cambiar. Solo habría que cambiar la asociacion IP-dominio.

Browsers

Sirven para ver la página web.
Cómo funcionan:

- A través del protocolo HTTP, pide una página u objetos al servidor web.
- Servidor web retorna la página/objetos solicitados.
- Browser interpreta el texto y los comandos de formateo que contiene y despliega la página adecuadamente formateada en pantalla.

Las páginas web se escriben en HTML para que los navegadores las entiendan.

Sitio web: Conjunto de páginas web relacionadas, localizadas bajo un único nombre de dominio, publicadas por al menos un servidor web. Pueden ser producidos por personas u organizaciones.
Home Page: Página de entrada al sitio web que sirve de guía hacia las páginas que contienen la información necesaria. Es la página que se carga por default.

¿Cómo se sabe con qué máquina se va a comenzar la conexión TCP? Se traduce el URL → Servidores DNS convierten nombres de dominio a direcciones IP.

Orden de comunicación entre Browser y Servidor web:

1. Cliente inicia una conexión TCP con el servidor web en puerto 80
2. Servidor web acepta la conexión del cliente
3. Mensajes HTTP(del protocolo) intercambiados entre browser y servidor web
4. La conexión TCP se cierra.

Con HTTP el servidor web no mantiene información acerca de pedidos pasados del cliente.
Los protocolos que mantienen el estado son complejos. El estado de la historia pasada debe ser mantenido. Si el servidor/cliente se caen sus visiones del estado pueden ser inconsistentes y deben reconciliarse.
No todas las páginas contienen solamente HTML. Para ello el servidor regresa con la página el tipo MIME de ella. Las páginas de tipo text/HTML se despliegan de manera directa.
Tipo MIME (Multipurpose Internet Mail Extension): Si no es de los integrados, el navegador consulta una tabla de tipos MIME que asocia un tipo MIME con un visor.
Para desarrollar visores se pueden utilizar: Plug-ins o aplicaciones auxiliares.

Plug-ins:

Es un modulo de codigo. El navegador los obtiene de un directorio especial del disco. Navegador instala un plug-in como una extensión del sistema. Los plug-in se ejecutan dentro del proceso del navegador, por lo tanto pueden acceder a la página actual y modificar su apariencia.
Interfaz del plug-in: Conjunto de procedimientos que todos los plug-in tienen que implementar para que el navegador pueda llamarlos.
Interfaz del navegador: Conjunto de procedimientos del navegador que el plug-in puede llamar.
Cuando un plug-in termina su trabajo se elimina de la memoria del navegador.

Aplicaciones auxiliares:

Se ejecutan en procesos separados del browser. No ofrecen interfaz al navegador ni usan servicios de este. Suelen aceptar el nombre de un archivo y lo abren y lo despliegan.

Servidores web:

Tiene páginas web y objetos que permiten construir páginas web.
Se le proporciona el nombre de un archivo correspondiente a una página a buscar y regresar.
Problema: Cada solicitud requiere un acceso al disco para obtener al archivo → Ineficiente → la página puede pedirse innumerables veces.
Solución: caché en la memoria.

Arquitectura de un módulo front end y k módulos de procesamiento(hilos) → hacer al servidor web más rápido.

Todos los MP(módulos de procesamiento) tienen acceso al caché.
Pasos de un servidor web con múltiples hilos:

1. Cuando llega una solicitud el front-end lo acepta y construye un registro corto que lo describe.
2. Entrega el registro a uno de los MP
3. MP verifica la caché, si el archivo se encuentra ahí.
4. Si está, actualiza el registro para incluir un apuntador al archivo
5. No está, MP inicia una operación de disco, cuando el archivo llega del disco se incluye en el caché y se regresa al cliente.

6. Mientras los MP están bloqueados haciendo operaciones de disco, otros MP pueden estar trabajando en otras solicitudes
7. Conviene tener múltiples discos, más de un disco puede ser ocupado al mismo tiempo.

Funcionamiento del MP:

1. Resuelve el nombre de la página web solicitada. No siempre se solicita un URL completo. Manejo de solicitud entrante sin el nombre real del archivo.
2. Control de acceso en la página web → páginas no disponibles para el público general o si la solicitud se puede satisfacer a partir de la identidad y ubicación del cliente → servidores listan el tipo de acceso. Se puede prohibir que dominios particulares accedan a la web.
3. Verifica el cache
4. Obtiene del disco la página solicitada o ejecuta un programa para construirla
5. Determina el tipo MIME de la página a través de un algoritmo, se incluye en la respuesta
6. Regresa la respuesta al cliente.
7. Realiza una entrada en el registro del servidor.

Cookies

En los pedidos y respuestas HTTP se envía información del estado de sesión → se usan cookies.
Son pequeños archivos o cadenas de a lo sumo 4 KB. Y su contenido es de la forma nombre = valor.

Campos de una cookie:

1. Dominio: Se pueden almacenar hasta 20 cookies por cliente
2. Ruta en la estructura del directorio del servidor → Identifica qué partes del árbol de archivos del servidor podrían usar el cookie
3. Contenido: nombre = valor.
4. Expira → Si este campo está ausente → navegador descarta el cookie cuando sale. Sino proporciona una fecha y una hora → mantiene la cookie hasta que expira ese horario.
5. Seguro → Indica que el navegador solo puede retornar la cookie a un servidor usando transporte seguro → aplicaciones seguras. (bancarias por ej).
6. Eliminacion de una cookie del disco duro del cliente → el servidor la envía nuevamente con una fecha caducada.

Directorio de cookies:

El navegador puede almacenar cookies en el disco duro de la máquina del cliente. La información de las cookies (del lado del servidor) se almacenan en una base de datos.
Comunicación de las cookies al cliente → cuando un cliente solicita una página web → el servidor envía una cookie junto a la página.
Comunicación de las cookies al servidor web:

- Antes que un navegador solicite una página a un sitio web, verifica su directorio de cookies para ver si el dominio al que está solicitando la pagina ya colocó alguna cookie
- Si lo hizo → todas las cookies para ese dominio se incluyen en el mensaje de solicitud
- Cuando el servidor web las obtiene las interpreta de la forma que desee.

Necesidades para un protocolo para la web:

- pedido de páginas/objetos/ejecución de programas que generan páginas
- manejos del estado de sesión
- mantener el sistema de archivos del servidor web
- recepción de páginas por un browser
- seguridad (encriptación del mensaje)
- feedback adecuado cuando no se puede responder a los pedidos
- comunicación confiable.

HTTP

Hipertexto porque las páginas tienen enlaces a otras páginas.

- Transfiere páginas de servidores web a navegadores y manda pedidos de navegadores a servidores web
- Tipos de mensajes: HTTP-Request y HTTP-Response.

HTTP no persistente	HTTP persistente
Un solo objeto se manda por TCP luego se cierra la conexión	Múltiples objetos pueden ser enviados a través de una única conexión TCP entre el cliente y el servidor
Descargar múltiples objetos requiere múltiples conexiones	Soportado por HTTP 1.1
HTTP 1.0 → establece conexión por cada solicitud y se libera al recibir respuesta	

RTT: tiempo necesario para que un paquete pequeño viaje del cliente al servidor y de regreso.

Tiempo de respuesta de HTTP no persistente:
- RTT para iniciar la conexión TCP
- RTT para el pedido HTTP y regreso de los primeros bytes de la respuesta HTTP
- Tiempo de transmisión
- Tiempo de respuesta = **2RTT + tiempo de transmisión**

Pedidos HTTP :

Información:

- Si se quiere recibir una página: URL del documento y especificación del programa que genera una página web.
- Tipo de acción que se quiere hacer en el sistema de archivos del servidor web
- Enviar información sobre la máquina/software del cliente → Servidor retorna páginas adecuadas
- Mandar información del estado de sesión para que el servidor se entere
- Restricciones sobre el tipo de páginas que el cliente puede aceptar.
- Indicar el tipo de acción que se quiere hacer → usar un campo de acción con el tipo de acción requerido → primera palabra de la primera línea es el nombre del método.

HTTP Métodos:

Fetch a page →

Used to send input data to a server program →

Method	Description
GET	Read a Web page
HEAD	Read a Web page's header
POST	Append to a Web page
PUT	Store a Web page
DELETE	Remove the Web page
TRACE	Echo the incoming request
CONNECT	Connect through a proxy
OPTIONS	Query options for a page

Para subir el input de un formulario hay dos opciones:

- POST: página web a menudo contiene input de formulario → input se encuentra en un campo llamado cuerpo de la entidad
- GET: usa URL → input se sube en el campo URL de la línea de pedido

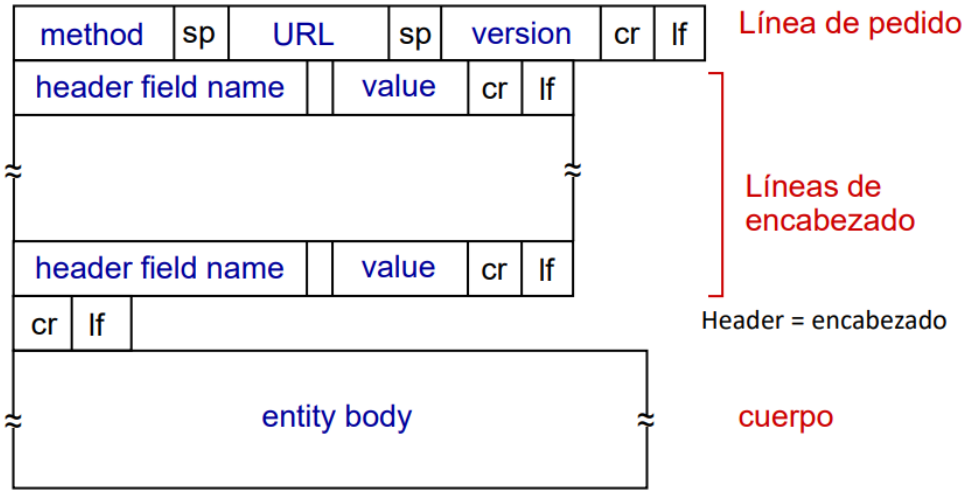
Método para actualizar páginas del servidor web:

- PUT: Sube un archivo en el cuerpo de la entidad en el campo especificado por el campo URL
- DELETE: Borra el archivo especificado en el campo URL

Otros métodos:

- HEAD: solicita el encabezado del mensaje sin la página real.
- OPTIONS: cliente consulta con el servidor y obtiene los métodos y encabezados que pueden ser usados con esa página.

Para tratar información diversa, se indica el tipo de información que se trata y luego la información en si → encabezados de pedido → pares encabezado, valor.



- A la línea de solicitud le pueden seguir líneas adicionales llamadas **encabezados de solicitud**.

Respuestas HTTP

Información:

- Feedback adecuado sobre el pedido realizado
- Página o documento solicitado
- Información sobre el tipo de documento enviado → Tipo MIME
- Información de estado de sesión para mantener actualizado al cliente.
- Feedback → Línea de estado: Contiene un código de estado de 3 dígitos que indica si la solicitud fue atendida y si no, porqué.

Code	Meaning	Examples
1xx	Information	100 = server agrees to handle client's request
2xx	Success	200 = request succeeded; 204 = no content present
3xx	Redirection	301 = page moved; 304 = cached page still valid
4xx	Client error	403 = forbidden page; 404 = page not found
5xx	Server error	500 = internal server error; 503 = try again later

Los encabezados de respuesta son pares (nombre de encabezado, valor).
Partes de una respuesta HTTP:

1. Línea de estado
2. (Opcional) encabezado de respuesta
3. Datos.

Encabezados HTTP:

- Mensajes HTTP suelen tener encabezados
- Son de pedido, respuesta o ambos
- Proveen información a ser procesada por el receptor del mensaje
- Fijan restricciones que deben cumplir los mensajes futuros
- Proveen información sobre eventos importantes
- Datos de los estados de sesión

Header	Type	Contents
User-Agent	Request	Information about the browser and its platform
Accept	Request	The type of pages the client can handle
Accept-Charset	Request	The character sets that are acceptable to the client
Accept-Encoding	Request	The page encodings the client can handle
Accept-Language	Request	The natural languages the client can handle
Host	Request	The server's DNS name
Authorization	Request	A list of the client's credentials
Cookie	Request	Sends a previously set cookie back to the server
Date	Both	Date and time the message was sent
Upgrade	Both	The protocol the sender wants to switch to
Server	Response	Information about the server
Content-Encoding	Response	How the content is encoded (e.g., gzip)
Content-Language	Response	The natural language used in the page
Content-Length	Response	The page's length in bytes
Content-Type	Response	The page's MIME type
Last-Modified	Response	Time and date the page was last changed
Location	Response	A command to the client to send its request elsewhere
Accept-Ranges	Response	The server will accept byte range requests
Set-Cookie	Response	The server wants the client to save a cookie

Páginas estáticas:

Documentos en algún formato → usualmente HTML → actualmente se usa HTML5

HTML

- Lenguaje estándar para crear páginas web
- estructura de una página web
- indica al navegador cómo mostrar el contenido
- sintaxis similar a XML
- permite crear páginas que incluyan texto, gráfico, hipervínculos, etc
- tablas y formularios.
- Un documento HTML es una serie de elementos. Un elemento es contenido encerrado entre etiquetas. Las etiquetas tienen la forma <nombre>. Las etiquetas a su vez pueden tener atributos, un atributo es de la forma nombre = "valor".

Tag	Description
<html> ... </html>	Declares the Web page to be written in HTML
<head> ... </head>	Delimits the page's head
<title> ... </title>	Defines the title (not displayed on the page)
<body> ... </body>	Delimits the page's body
<h *n*> ... </h*n*>	Delimits a level *n* heading
...	Set ... in boldface
...	Set ... in italics
...	Center ... on the page horizontally
...	Brackets an unordered (bulleted) list
...	Brackets a numbered list
	Starts a list item (there is no)
 	Forces a line break here
<p>	Starts a paragraph
<hr>	Inserts a Horizontal rule
<img src="...">	Displays an image here
 ... 	Defines a hyperlink

Páginas dinámicas:

Las páginas estáticas son muy ineficientes cuando la información cambia frecuentemente o la información de la página varía de acuerdo a parámetros.
La solución es usar páginas dinámicas:

- páginas HTML generadas por un programa del lado del servidor
- tienen parámetros de entrada
- estos parámetros suelen ser ingresados por formularios HTML

Pasos para crear una página dinámica:

1. El usuario llena un formulario y lo envía
2. se envia mensaje al servidor web con el contenido del formulario → se lo envía a un programa o secuencia de comandos → el programa lo procesa
3. El programa solicita información a un servidor de base de datos
4. El servidor de base de datos responde con la información requerida
5. El programa genera una página HTML personalizada y la envía al cliente
6. El browser muestra la página

Se pueden omitir los pasos 3 y 4 si el servidor no está conectado a una base de datos.

Con un URL no basta para especificar la página dinámica deseada.

- necesita parámetros para crear la página dinámica
- hace falta poder ingresar los parámetros en el pedido HTTP

Solución 1: el URL contiene nombre de programa y parámetros. Los parámetros son ingresados por medio de formulario HTML.

- parámetros: nombre = valor
- se separan del nombre del programa con '?'
- se separan parámetros con '&'

Solución 2: Los parámetros se ingresan separados por '&' en el cuerpo de la entidad.
La solución 2 se utiliza cuando los parámetros ocupan mucho espacio → límite de una URL 2048 caracteres.

Formulario HTML:

Tag	Description
<form action="", method =""> … <\form>	Declara un formulario. Action es URL de la página ejecutable que procesa formulario. Method especifica cómo los datos se mandan al servidor (p.ej. GET, POST)
<select> … <\select>	Para especificar una lista de la que usuario elige un elemento.
<option value = ""> … <\option>	Para indicar opción de <select>
<textarea rows = "" cols=""> … </textarea>	Control de ingreso de texto de varias líneas
<input name="" type="" value="">	Permite definir campo de input donde type puede ser: button, radio, password, text, submit, checkbox, hidden, etc.

Dentro de select se usa option.

PHP

Define páginas dinámicas mediante la inserción de comandos especiales (scripts) dentro de páginas HTML. (permite definir clases y objetos).
Para utilizar PHP, el servidor web debe entender PHP:

- Código PHP interpretado por el servidor web
- Diseñado para trabajar en el servidor web Apache
- puede usarse en la mayoría de los servidores web

Utilidad de PHP:

- puede generar contenido de página dinámica
- operar con archivos del servidor
- recolectar datos de formulario
- enviar y recibir cookies
- acceder a encabezados de pedido HTTP
- definir encabezados de respuesta HTTP
- acceder a base de datos

Construcción	Description
<?php … ?>	PHP script Puede ir en cualquier lugar del documento
'$' NAME	Declaración de variable Case sensitive
echo EXPR	Para mostrar datos en pantalla
//	comentarios
Define(name, value)	Definición de constantes
VARIABLE = EXPR	Igual que en C (+=, -=, *=, /=)
Include 'filename'	Toma el texto/código/markup en un archivo y lo copia en el archivo que usa la sentencia *include*

Acceder a campos de formulario

- $_POST → recolectar datos de un formulario con método POST
- $_GET → recolectar datos de un formulario con método GET

Ejemplo: $_POST[`fname`] → datos del campo fname

Operadores

- De asignación y comparación para los distintos tipos de datos.

Acceder a información de encabezados HTTP:

$_SERVER contiene la información de los encabezados, caminos y localización de scripts.
Para acceder a los encabezados, necesitamos agregar como argumentos:

- HTTP_USER_AGENT
- SERVER_ADDR
- SERVER_NAME
- SERVER_SOFTWARE
- SERVER_PROTOCOL
- REQUEST_METHOD
- REQUEST_TIME
- QUERY_STRING
- HTTP_ACCEPT
- HTTP_ACCEPT_CHARSET
- HTTP_HOST

entre otros.

Definir encabezados de respuesta HTTP:

Se utiliza la función header() → se debe fijar encabezados antes de la etiqueta <html>.

Definición de cookies:

Se utiliza la función setcookie() → define una cookie que va a ser enviada junto con los encabezados HTTP. → se debe fijar antes de la etiqueta <html> → antes de generar cualquier salida.
se utiliza de la siguiente forma: setcookie(name, value, expire, path, domain, httponly)

Acceder al valor de una cookie:

- $_COOKIE → retorna el valor de una cookie
- htmlspecialchars() → convierte caracteres especiales en entidades html

Forma de uso: $_COOKIE[‘fname’] → valor de la cookie fname

Críticas al modelo actual:

Que el servidor web tenga que construir páginas dinámicas puede ser ineficiente por los siguientes motivos:

- la página nueva a generar dinámicamente puede tener muchas partes en común con la que tiene el browser → estas partes se generarían de vuelta y serán nuevamente enviadas → van a tener que ser reinterpretadas por el browser.
- El cliente se queda bloqueado esperando luego de hacer un pedido HTTP y recién puede continuar con la ejecución cuando recibe una página → pedidos sincrónicos.

Liberando al servidor web y aprovechando el poder del cliente:

El servidor web en respuesta a un pedido HTTP solo enviará datos para actualizar parte de una página web en el navegador y el browser se ocupa de actualizar la interfaz del usuario.

- Pedidos asincrónicos → el cliente puede seguir ejecutando tras realizar un pedido
- Cuando llegan datos del servidor web se genera e inserta la parte nueva de la IU y se deja igual el resto a preservar.

Implementación:

- Se pide el código de la IU de la aplicación al servidor web → escrito en HTML + JavaScript + Pedidos HTTP asincronicos → IU enriquecida similar a la IU de las aplicaciones de escritorio

- Los datos recibidos de un pedido asincrónico son procesados por el cliente para actualizar la IU

JavaScript:

Construcción	Description
<script> … </script>	Para insertar un script en JavaScript
function NAME (PARAMETERS) {BODY}	Declaración de funciones
Var NAME = EXPR;	Declaración e inicialización de variables
Objeto document	Representa la página siendo visualizada
Document.write(t)	Escribe texto *t* en el stream de salida de la página siendo visualizada.
Método document.getElementById(*X*)	Retorna elemento de identificador *X*. *Es el elemento con atributo id y valor X.*
Propiedad *e*.innnerHTML	Contenido del elemento *e*

Elementos HTML pueden tener atributo id → este debe ser único en todo el documento, sirve para referirse a ese único elemento.

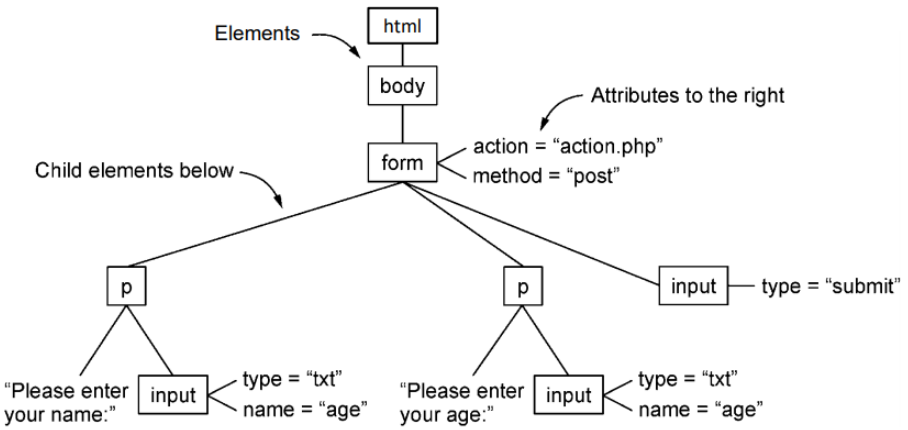
Procesamiento de eventos en JavaScript:

La reacción a un evento puede significar hacer cambios en la IU o hacer cierta tarea de procesamiento en el cliente o hacer pedidos al servidor web y mostrar respuesta en pantalla.
Definir reglas ECA (evento-condición-acciones):

- Si ocurre un evento y se da una condición entonces ejecutar una secuencia de acciones
- nombre del evento: abort, click, focus, load, select, submit, onload.
- Los eventos se asocian a etiquetas de páginas HTML → insertar atributo on[nombre-evento]
- La parte de acciones es el procesamiento es el procesamiento de un evento en sí → cuando ocurre un evento, ejecutar una serie de acciones o función JavaScript → encerradas en <script> dentro de <head>

DOM (Document Object Model):

Representación de una página HTML que es accesible a programas. La página está estructurada como un árbol DOM → estructura jerárquica de los elementos HTML.
Además al lado de un elemento puede uno dibujar sus atributos. Permite a los programas cambiar partes de una página sin necesidad de sobreescribir toda la página. JavaScript puede acceder al modelo DOM. Se pueden tener scripts que modifican el documento HTML siendo visualizado.



JavaScript usa '.' para navegar a través de las propiedades y referencias asociadas con el documento.

- Jerarquía del árbol DOM es navegada comenzando con el objeto window → representa el navegador
- Window tiene una propiedad llamada document → representa a la página HTML
- Objeto document → tiene una colección de formularios llamada forms → indexada por el número de aparición en el documento, comienza en 0.
- Un formulario tiene una colección de elementos llamada elements → indexada por el número de aparición en el formulario → comienza en 0.

Contenedores:

 → Define una división o sección en un documento HTML. Contenedor para otros elementos HTML. Pueden anidarse unos dentro de otros y permite definir una jerarquía de contenedores y organizar una página.

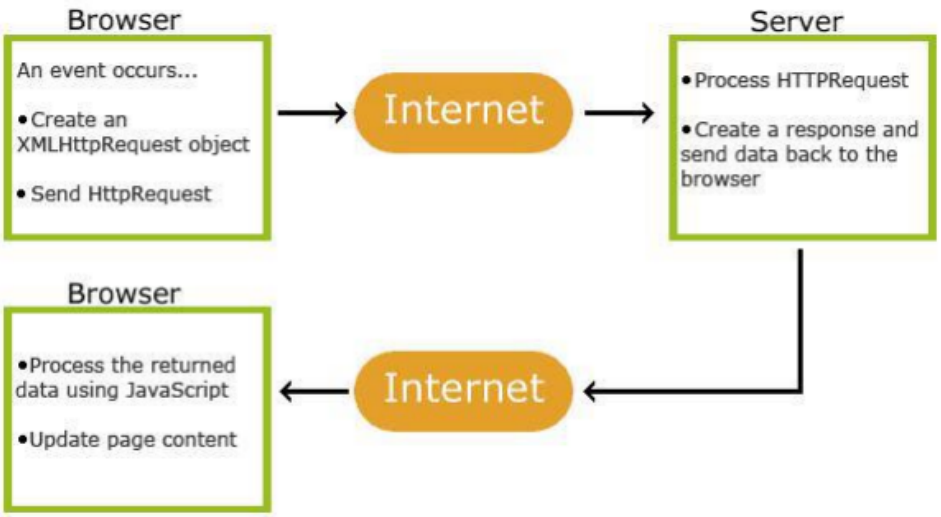
- Algunas sentencias y métodos que permiten modificar una página HTML siendo visualizada.

Construcción	Description
e.innerHTML = c;	Cambia contenido HTML de *e* por *c*
e.a = v;	Cambia valor de atributo *a* de elemento *e* por *v*.
e.setAttribute(a,v)	Crea en elemento *e* el atributo *a* con valor *v*. Si *a* ya estaba , solo cambia su valor.
document.createElement(*N*)	Crea elemento HTML de etiqueta *N*
document.createTextNode(*T*)	Crea nodo de texto *T*
e.appendChild(n)	Se agrega elemento *n* como último hijo del elemento *e*
e.removeChild(n)	De elemento *e* se remueve subelemento *n*.
e.replaceChild(n1,n2)	De elemento *e* se reemplaza subelemento *n1* por elemento *n2*.

Primitivas para hacer y procesar pedidos/respuestas HTTP

Construcción	Description
Objeto XMLHttpRequest	Objeto usado para intercambiar datos con servidor web.
Método Open(method, url, async, user, psw)	Para especificar el pedido HTTP
Método send()	Para enviar el pedido HTTP
Método setRequestHeader()	Para fijar encabezados del pedido HTTP
Propiedad responseText de objeto XMLHttpRequest	Se refiere a la respuesta del servidor como un string
Propiedad readyState	Estatus del XMLHttpRequest
getAllResponseHeaders()	Retorna toda la información de encabezados de respuesta

Pasos para hacer pedido HTTP y contestarlo:



1. Especificar pedido HTTP con open()
2. Un pedido HTTP se envía con send() → pedido de datos o texto
3. Servidor obtiene los datos pedidos → crea respuesta HTTP y la envía al navegador
4. El cliente recibe la respuesta → procesa los datos → modifica la página actual → primitivas de IU

Manejo de pedidos HTTP al servidor:

El objeto XMLHttpRequest puede usarse para intercambiar datos con un servidor web.
` var = new XMLHttpRequest();`
Se especifica el pedido HTTP con método open().
` open(method, url, async, usr, psw)`

- method → GET o POST
- url → localización del archivo
- async → true si es asincrónico
- user y password

Tipos de pedidos:

- Pedidos asincrónicos → cliente hace pedido y sigue trabajando
- Pedidos sincrónicos → cliente hace pedido y se bloquea esperando una respuesta.

Para enviar pedido GET → send()
Para enviar pedido POST → send(string)
Para fijar encabezados → setRequestHeader()

Manejo de respuestas HTTP del servidor:

La propiedad responseText retorna la respuesta del servidor como un string.
La propiedad readyState mantiene el estatus del XMLHttpRequest
La propiedad onreadystatechange → define una función a ser ejecutada cuando el readyState cambia

- conexión con el servidor establecida
- pedido recibido
- procesando pedido
- pedido terminó y respuesta lista

getAllRequestHeaders() → retorna toda la información de encabezados de respuesta del servidor
getResponseHeader() → retorna un encabezado específico de la respuesta.

XML:

Extensive Markup Language. Se utiliza para definir datos:

- Nombre de los elementos de datos son expresados mediante etiquetas (<nombre>)

- Un elemento de datos de nombre n consiste en información cerrada entre etiquetas. <n>y</n>
- Pueden ser construidos por otros elementos anidados o solo texto

La propiedad responseXML retorna la respuesta como un documento XML.
d.getElementsByTagName(N) retorna los elementos XML dentro del documento d que tiene nombre de etiqueta N
Como recorrer un elemento XML:

- pensar el documento XML como un árbol → recorrerlo usando expresiones de camino → cada sección separada por '.'.
- e.childNodes[i] → retorna el nodo hijo i de e
- e.nodeValue → si es un elemento de solo texto retorna el texto.
- HTML respeta XML → se puede generalizar el árbol DOM para XML.

AJAX:

Conjunto de tecnologías que trabajan juntas para permitir que las aplicaciones web sean tan eficientes y poderosas como las aplicaciones de escritorio tradicionales

- HTML y CSS → presentar información como páginas
- DOM → cambiar partes de las páginas mientras son vistas
- XML → permitir que los programas intercambien datos de aplicación con el servidor
- comunicación asincrónica → del cliente con el servidor web para enviar y retornar datos
- JavaScript → lenguaje para ligar todas estas facilidades.

Capa de Transporte

Propósito de la capa de transporte
La **capa de transporte (CT)** provee comunicación lógica entre procesos de aplicación que se ejecutan en diferentes sistemas finales.

- esto no lo puede hacer la capa de red CR.
- La CT se implementa (salvo alguna excepción) solo en los sistemas finales.

Comunicación lógica como si los hosts ejecutando los procesos estuvieran directamente conectados.
Para **mejorar la calidad** los servicios de la CR:

- P.ej retransmisiones de paquetes perdidos en redes no orientadas a la conexión.
- P.ej cuando hay congestión en la red, regulando de manera fina la variación de la tasa de transmisión de paquetes de los hosts.

La CT se ejecuta por completo en los hosts/sistemas finales.
La CT confía en los servicios de la CR.
Entidad de transporte (ET) = software/hardware de la CT.

Problemas que soluciona la capa de transporte:

- Uso de temporizadores y las retransmisiones de paquetes.
- El direccionamiento explícito de los destinos.
- Uso de búferes y control de flujo.
- Evitar congestionar la red poniendo demasiados paquetes en ella.
- Cuando la CR pierde paquetes, la CT puede solucionarlo.
- No se preocupa de si un paquete se pierde por congestión.
- O si una máquina se quiere conectar con un servidor que no se está ejecutando.

Segmento unidad de datos del protocolo de transporte.
Confirmaciones de recepción de paquetes enviados.
Tipos de paquetes que deben ser confirmados

- ➢ paquete de datos
- ➢ paquetes con información de control

Si pasa demasiado tiempo sin una confirmación se asume que se perdió un paquete(suele suceder por congestión) → se usan temporizadores.
Si altero el flujo de datos estoy entregando un mensaje que el emisor quiere enviar que NO es y la capa de aplicación va a hacer algo que no tiene que hacer.

Para la entrega ordenada de segmentos el host puede:

- usar números de secuencia → números respetando el flujo de segmentos de la capa de aplicación.
- usar temporizadores de retransmisión por cada número de segmento enviado.
- confirmación de recepción (**ACK**).
- si caduca el temporizador sin recibir un **ACK** → retransmitir el segmento.
- reensamblar y enviar a la capa de aplicación los segmentos recibidos (en orden).

TCP proporciona un flujo de bytes confiables de extremo a extremo a través de una interred no confiable. Se adapta dinámicamente a las propiedades de la interred y se sobrepone a muchos tipos de fallas.

Problemas que resuelve TCP:

- retransmisión de paquetes → uso de números de secuencia, ACKs y temporizadores
- fijar la duración de temporizadores retransmisión
- manejo de conexiones entre pares de procesos
- direccionamiento
- control de congestión
- control de flujo

Una ETCP acepta flujo de datos a transmitir de procesos locales.
El servicio TCP se obtiene al hacer que tanto el servidor como el cliente creen sockets. ***Dirección de un socket = IP + puertos***. Servicio TCP a través de una conexión entre el socket emisor y el socket receptor.
Un socket puede usarse para múltiples conexiones:

- dos o más conexiones pueden terminar en el mismo socket
- conexiones se identifican mediante los identificadores de sockets de los dos extremos.

Cada byte de un flujo de datos a enviar en una conexión TCP tiene su propio número de secuencia (32 bits). → límite en el tamaño de los datos.
Segmentos → encabezado TCP ++ Datos
Cada red tiene una unidad máxima de referencia (MTU) → cada segmento debe caber en el MTU (1500 B).
Los datagramas que llegan podrían llegar en orden incorrecto. Esto sucede en redes de datagramas.

1. Cuando un transmisor envía un segmento → inicia un temporizador.
2. Cuando llega a destino → ETCP receptora → devuelve ACK, es el siguiente número de secuencia.
3. Si el temporizador expira antes del ACK → emisor envía nuevamente el segmento

Campos de un segmento TCP:

1. encabezado fijo → 20 bytes
2. Opciones de encabezado en palabras de 32 bits
3. Datos opcionales

Los segmentos sin datos se usan como ACKs o paquetes de control.

Direccionamiento:

¿Cómo hacer para que un procesador adecuado atienda las necesidades de una máquina cliente?

- El cliente conoce cual es el procedimiento adecuado para un servicio en particular(Servidor activo).
- El cliente podría no saber cuál es el procedimiento adecuado para un servicio particular.(Servidor activo).
- El cliente si lo sabe, pero el servidor no está activo .

El cliente conoce el servidor (Servidor activo)

Para conectarse al servidor existe un proceso especial llamado **servidor de directorio** que para cada tipo de servicio sabe cuáles son los puertos de los servidores que prestan ese tipo de servicio

servidor de directorio siempre está escuchando esperando una señal de CONNECT.

Procedimiento:

1. El usuario establece una conexión con el servidor de directorio (que escucha en un puerto bien conocido).
2. El usuario envía un mensaje especificando el nombre del servicio.
3. El servidor de directorio le devuelve la dirección puerto.
4. El usuario libera la conexión con el servidor de directorio y establece una nueva con el servicio deseado.

Creación de un servicio nuevo:

- El servicio nuevo debe registrarse en el servidor de directorio, dando su nombre de servicio como la dirección de su puerto.
- El servidor de directorio registra esta información en su base de datos.

El cliente conoce el servidor (Servidor inactivo):

Usar un servidor que ejecuta los servidores inactivos: **protocolo inicial de conexión**

Procedimiento:

- Un usuario emite un CONNECT más el puerto de servicio activo
- Como el Servidor de Procesos tiene los servidores inactivos para esa solicitud, se conecta al protocolo inicial de conexión
- el protocolo inicial de conexión genera(activa) algun servidor con capacidad de atender la solicitud y le hereda la conexión

Control de flujo:

La ET emisora debe manejar búferes para los mensajes de salida porque puede hacer falta retransmitirlos. El emisor almacena en búfer todas los segmentos hasta que se confirma su recepción.

Hay que evitar que un host emisor rápido desborde a un host receptor lento.

¿Por qué control de flujo en la CT si la CR también lo tiene?

El receptor puede demorarse en procesar mensajes debido a problemas en la red

- pérdida de segmentos
- no se puede procesar segmentos porque faltan anteriores

Protocolo parada y espera:

- ➢ Una vez que un emisor manda un segmento y para de enviar (parada).
- ➢ El emisor espera por un ACK del segmento (espera).
- ➢ Si llega el ACK a tiempo se envía el siguiente paquete.

➢ Si se pierde un pkt o un ACK, expira un temporizador de retransmisiones y se va a tener que mandar de nuevo.

No existe el problema de desbordamiento ya que siempre tiene la confirmación de recepción.

Procedimiento:

- **Suposición:** el canal de comunicaciones subyacente puede perder paquetes (de datos, de ACKs)
  - Los paquetes tienen N° de secuencias,
  - Se trabaja con Acks
    - El receptor debe especificar N° de secuencia del paquete siendo confirmado.
  - Se usan retransmisiones de paquetes.
    - Para esto se requiere de uso de temporizadores.

Comportamiento del emisor:

1. El emisor envía paquete P y para de enviar.
2. **Espera**: El emisor espera una cantidad "razonable" de tiempo para el ACK
3. Si llega el ACK a tiempo, se envía siguiente paquete. Goto 2.
4. Sino se retransmite paquete P. Goto 2.

- Si hay paquete o ACK demorado pero no perdido:
  - La retransmisión va a ser un duplicado con igual N° de secuencia ; luego se descarta.

- Desempeño pobre.
- el protocolo limita el uso de recursos físicos
- RTT es tiempo de ida y vuelta de un bit: RTT.

Si el tiempo de transmisión de un segmento es mucho menor que el RTT, la utilización del canal en parada-espera va a ser muy baja.
Se nota cuando las máquinas están muy alejadas entre sí, ya que se agranda el problema → termina siendo ineficiente.

Protocolos de Tubería:

Tubería: el emisor puede enviar múltiples paquetes al vuelo a ser confirmados.

- El rango de números de secuencia debe ser incrementado
- Uso de búferes en el emisor y/o el receptor.

Hay dos formas genéricas de protocolos de tubería:

- Repetición Selectiva
- Retroceso N

Situación: Se perdió una confirmación de recepción y se envió el paquete de nuevo. El mismo paquete llega dos o más veces al receptor y la capa de transporte la pasa a la capa de aplicación más de una vez. Para evitar esto se asignan números de secuencia a los paquetes que salen.

Retroceso-N:

- Receptor envía *ack acumulativo*
  - No confirma paquetes si hay un agujero.
- El emisor tiene un timer para el paquete más viejo no confirmado
  - Cuando expira el timer retransmite todos los paquetes no confirmados.

Repetición selectiva:

- El receptor envía *confirmaciones individuales* para cada paquete
- El emisor mantiene un timer para cada paquete no confirmado
  - Cuando el timer expira, retransmite solo ese paquete no confirmado.

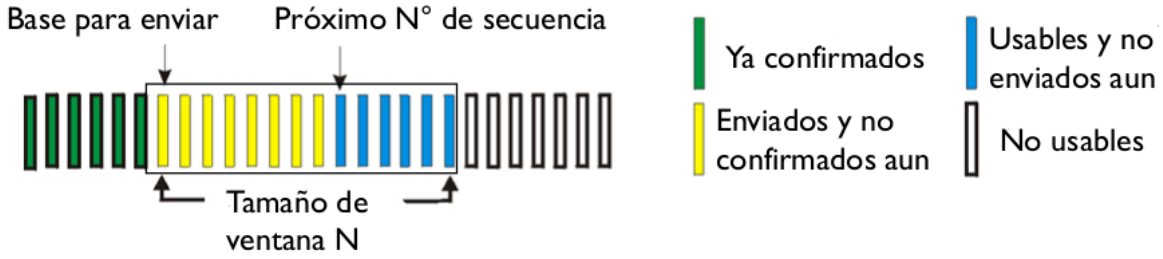
Retroceso N:
Se envía un ack acumulativo →
Ejemplo: Enviamos una rafaga de 10 paquetes con número de secuencia entre 0 y 9, el receptor solo devuelve un ack de respuesta con número de secuencia 9 para confirmar la llegada de los 10 paquetes.
Si se pierde alguno de esos paquetes el receptor descarta todos los paquetes de la rafaga y no devuelve ningún ack para que el emisor envíe toda la rafaga de nuevo.

- ➢ Receptor envía ack acumulativo (todos los segmentos anteriores se recibieron bien).
- ➢ Al expirar el timer del segmento más viejo no confirmado, retransmite todos los segmentos no confirmados → si pierdo un segmento debe enviar todos los que le siguen otra vez.

¿Cómo representamos cuántos paquetes seguidos SIN confirmación puede enviar el emisor?
A través de **intervalos de números de secuencia** en el **espacio de números de secuencia**.
ejemplo: con el ejemplo de rafagas de arriba, supongamos que enviamos 15 rafagas entonces el **espacio de números de secuencia** sería = {0,1,.....,149} y los **intervalos** serían de [0,..,9][10,...,19], etc. Estos **intervalos** reciben el nombre de **ventana corrediza.**

Ventana emisora: tramas(paquetes) enviadas sin ack positivo o tramas listas para ser enviadas

- El tamaño de la ventana emisora no puede superar MAX_SEQ cuando hay MAX_SEQ + 1 números de secuencia.
  - La "ventana" permite hasta *N* paquetes consecutivos sin confirmar
  - **ventana emisora** = tramas enviadas sin ack positivo o tramas listas para ser enviadas.



  - *timeout(n):* retransmite paquete *n* y todos los paquetes de mayor N° de secuencia en la ventana.

Problema Retroceso N:

- Uso ineficiente del canal cuando se pierden paquetes

Protocolo de Repetición Selectiva: soluciona el problema de retroceso N y algunas mejoras más

- Cuando se pierde un paquete de una rafaga de paquetes en lugar de descartar los que llegaron correctamente, los guardamos en un buffer y esperamos a que llegue el paquete perdido(reenviado) para entregarlos en orden a la capa de aplicación, existe un límite para la cantidad de paquetes que podemos almacenar en un buffer del receptor esperando el perdido.
- Confirmamos individualmente todos los paquetes, no hay ack acumulativo.
- tiene un ack Negativo(NAK): es decir, que cuando detecta que llegaron varios paquetes "mayores" a digamos N, genera un NAK para que el emisor reenvie el paquete N.

Para ser más rápido, ya no tiene que esperar a que expire el temporizador del paquete u otros métodos.

Ventana del emisor:

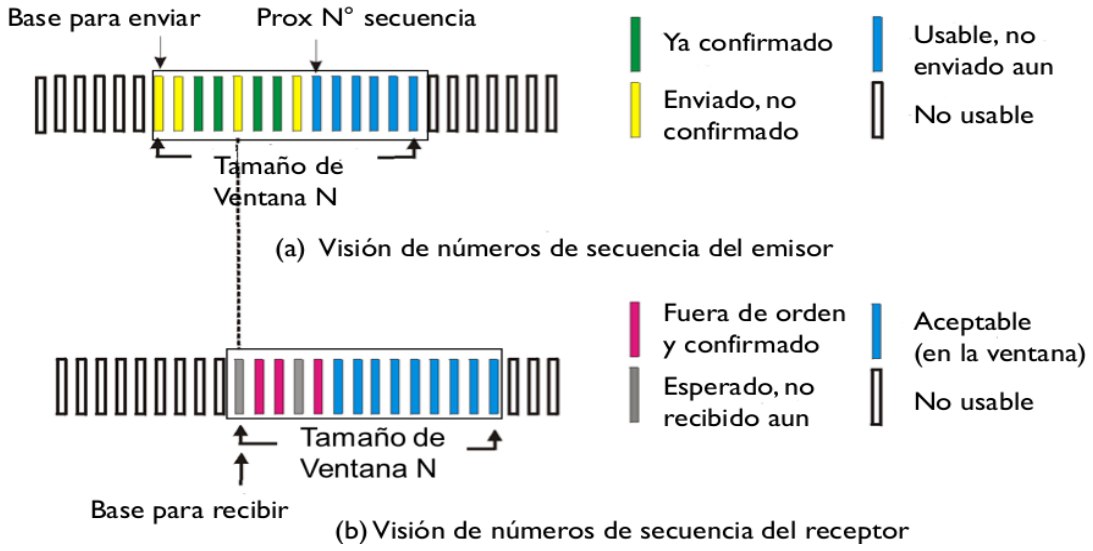
- Paquetes que contiene:
  - enviados y no confirmados
  - enviados y confirmados pero alguno enviado antes todavía no se confirmó
  - listos para enviar

Ventana del receptor:

- Tipos de paquetes:
  - paquetes esperados y no recibidos
  - paquetes recibidos fuera de orden
  - paquetes aceptables en la ventana que aún no han llegado

Entonces, se mantiene en buffer un paquete aceptado por la ventana receptora hasta que todos los que le preceden hayan sido entregados a la capa de aplicación.

Repetición Selectiva: ventanas del emisor y del receptor



Procedimiento:

Emisor

Datos vienen de arriba:

- Si el próximo N° secuencia a enviar de la ventana está disponible, almacenar y enviar paquete

timeout(*n*):

- Reenviar paquete *n*, reiniciar timer

ACK(*n*) en [sendbase,sendbase+N]:

- marcar paquete *n* como recibido
- Si *n* es paquete más pequeño no confirmado, ***avanzar base de ventana*** al siguiente N° secuencia no confirmado.

Receptor

pkt *n* en [base rcv, base rcv +N-1]

- Enviar ACK(n)
- Fuera de orden: almacenarlo
- En orden: entregar (también entregar paquetes en bufer en orden), avanzar ventana al siguiente paquete que no ha sido recibido aun.

pkt *n* en [base rcv-N, base rcv-1]

- Enviar ACK(n)

Sino:

- ignorar

Súposiciones: *N* es tamaño de ventana

Regla para el tamaño de la ventana receptora:

- Tamaño de ventana receptora = (MAX_SEQ + 1)/2.
- Con tamaños mayores de ventana receptora no funciona.

¿Cómo transmitir datos entre dos máquinas y en ambas direcciones eficientemente?
Solución: llevar a caballito (piggybacking).

- Cuando llega un segmento S con datos, el receptor se aguanta y espera hasta que la capa de aplicación le pasa el siguiente paquete P.
- La confirmación de recepción de S se anexa a P en un segmento de salida (usando el campo ack en el encabezado del segmento de salida).

4. Control de flujo cuando la cantidad de datos que quiere recibir y procesar el receptor varía.

Cuando cambia el patrón de tráfico de la red; se abren y cierran varias conexiones en el receptor. El receptor y el emisor deben ajustar dinámicamente sus alojamientos de búferes, esto significa ventanas de tamaños variables. Ahora el emisor no sabe cuántos datos puede mandar en un momento dado, pero sí sabe cuántos datos le gustaría mandar.

¿Qué reglas cumpliría un protocolo entonces?
El host emisor solicita espacio en búfer en el otro extremo. Para estar seguro de no enviar de más y sobrecargar al receptor, y porque sabe cuánto necesita.

¿Qué pasa con el receptor al recibir ese pedido?
Sabe cuál es su situación y cuánto espacio puede otorgar, y aquí el receptor reserva una cierta cantidad de búferes al emisor.

Comunicación entre host emisor y host receptor cuando el host emisor solicita espacio en búfer en el otro extremo:

1. Inicialmente el emisor solicita una cierta cantidad de búferes, con base en sus necesidades percibidas.
2. El receptor otorga entonces tantos búferes como puede.
3. El receptor, sabiendo su capacidad de manejo de búferes podría indicar al emisor “te he reservado X búferes”.
4. ¿Cómo hace el receptor con las confirmaciones de recepción? El receptor puede incorporar tanto las ack como las reservas de búfer en el mismo segmento.
5. Aclaración: TCP no lo hace, tiene un buffer único, solo otorga espacio.
6. El emisor lleva la cuenta de su asignación de búferes con el receptor.

Información de reserva de búferes viaja en segmento que no contiene datos y ese segmento se pierde. Esto termina ocasionando deadlock.

¿Cómo evitar esta situación de deadlock? Cada host puede enviar periódicamente un segmento de control con el ack y estado de búferes de cada conexión. Así el estancamiento se romperá tarde o temprano.

5. Control de flujo en TCP:

No se requiere que los emisores envíen datos tan pronto como llegan de la aplicación, que los receptores envíen confirmaciones de recepción tan pronto como sea posible, y que los receptores entreguen datos a la aplicación apenas los reciben, esta libertad puede explotarse para mejorar el desempeño.

Campo Tamaño de ventana en el encabezado TCP:

- N° de bytes que pueden enviarse comenzando por el byte cuya recepción se ha confirmado.
- 0: indica que se han recibido los bytes hasta n° de confirmación de recepción – 1, inclusive, pero el receptor quisiera no recibir más datos por el momento.
- El permiso para enviar puede otorgarse enviando un segmento con el mismo n° de confirmación de recepción y un campo tamaño de ventana distinto de 0

¿Qué se hace si la ventana anunciada por el receptor es de 0?
El emisor debe detenerse hasta que el proceso de aplicación del host receptor retire algunos datos del búfer y en cuyo momento el TCP puede anunciar una ventana más grande.

En TCP los hosts en cada lado de una conexión tienen un buffer de recepción circular para la conexión. Cuando la conexión TCP recibe bytes en el orden correcto y en secuencia, coloca los datos en el buffer de recepción.

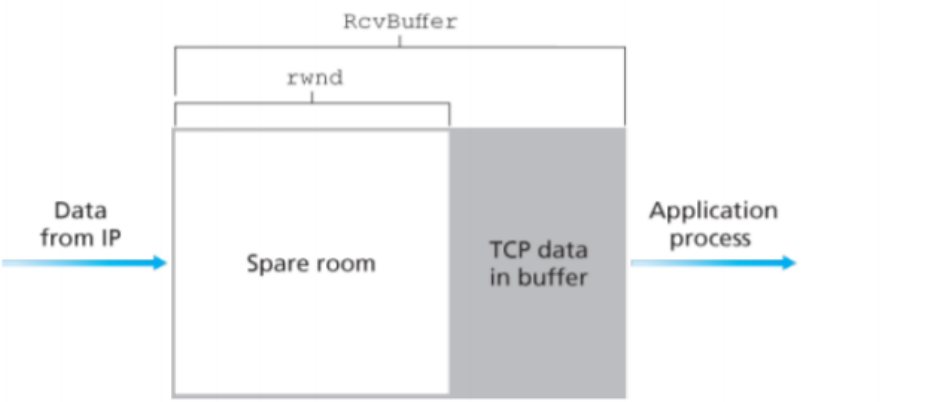


Figure 3.38 The receive window *(rwnd)* and the receive buffer *(RcvBuffer)*

El valor rwnd va en el campo tamaño de ventana.

Cálculo de tamaño de ventana:

- Tamaño de ventana = RcvBuffer − [LastByteRcvd − LastByteRead]
- Inicialmente se puede mandar: Tamaño de ventana = tamaño RcvBuffer
- Propiedad a respetar en el emisor: LastByteSent − LastByteAcked ≤ tamaño de ventana (Cantidad de bytes enviados pero no confirmados)

¿Cómo manejar pérdidas de segmentos en TCP?
La primera solución es que el receptor solicite segmento/s específico/s mediante segmento especial llamado NAK. Tras recibir segmento/s faltante/s, el receptor puede enviar una confirmación de recepción de todos los datos que tiene en búfer. Cuando el receptor nota una brecha entre el número de secuencia esperado y el número de secuencia del paquete recibido, el receptor envía un NAK en un campo de opciones.
La segunda solución es que (acks selectivos) el receptor le diga al emisor que piezas recibió. El emisor puede así reenviar los datos no confirmados que ya envió. Se usan dos campos de opciones:

- Sack permitted option: se envía en segmento SYN para indicar que se usarán acks selectivos.
- Sack option: Con lista de rangos de números de secuencia recibidos.

Control de Congestión:

Si un emisor manda a un receptor más información que la capacidad de carga de la subred:

- ➢ la subred se congestionará pues será incapaz de entregar los segmentos a la velocidad con que llegan.

- Aca vemos directamente los algoritmos de TCP para el control de congestión
- TCP controla la congestión haciendo que los hosts reduzcan la tasa de datos
- Cualquier expiración de temporizador para TCP implica congestión en la red

Para llevar la cuenta de cuántos datos un host puede enviar por la red:

- TCP maneja una ventana para la congestión (VC) - cuyo tamaño es el número de bytes que el emisor puede tener en la red en todo momento.

ACKs duplicados:

El emisor recibe ack por cada paquete entregado con éxito, entonces para un paquete de número de secuencia N si recibe 3 ack positivos con números de secuencia > N va a reenviar el paquete

Para calcular un tamaño para la ventana de congestión (VC):

- arranque lento
- TCP Tahoe
- TCP Reno

Arranque lento:

- probar con un mínimo de datos e ir duplicando gradualmente hasta que no se pueda más
- la ventana crece hasta el tamaño del espacio de números de secuencia o hasta que se detecta que se perdió un paquete, en cuyo caso se reduce a la mitad de su tamaño

Deficiencias:

- Recortar la ventana de congestión a la mitad porque hubo una expiración de temporizador y quedarse ahí, puede ser demasiado
- con la retransmisión disparada por expiración de temporizador el tiempo de espera puede ser relativamente grande

TCP Tahoe:

- Usa un umbral además de las ventanas de recepción y congestión
- Al ocurrir una expiración del temporizador o detectarse 3 acks duplicados, se fija el umbral en la mitad de la ventana de congestión actual, y la ventana de congestión se restablece a un segmento máximo.
- Luego se usa el arranque lento para determinar lo que puede manejar la red, excepto que el crecimiento exponencial termina al alcanzar el umbral.
- A partir del punto en el que se alcanza el umbral las transmisiones exitosas aumentan linealmente la ventana de congestión (en un segmento máximo por ráfaga).
- Recomenzar con una ventana de congestión de un paquete toma un RTT (para todos los datos previamente transmitidos que dejen la red y para ser confirmados, incluyendo el paquete retransmitido).
- Si no ocurren más expiraciones de temporizador/3 acks duplicados, la ventana de congestión continuará creciendo hasta el tamaño de la ventana del receptor.
    - • En ese punto dejará de crecer y permanecerá constante mientras no ocurran más expiraciones de temporizador y la ventana del receptor no cambie de tamaño.

Crítica a Tahoe:

- Comenzar con arranque lento cada vez que se pierde un paquete puede ser demasiado.

TCP Reno:

Evitar arranque lento (excepto cuando la conexión es comenzada) cuando expira el temporizador de reenvíos.

Funcionamiento:

1. Luego de iniciada la conexión se comienza con arranque lento.
2. A continuación la ventana de congestión crece linealmente hasta que se detecta una pérdida de paquete.
   - Se cuentan acks duplicados
   - Se considera pérdida de paquete 3 acks duplicados
3. El paquete perdido es re-transmitido usando **retransmisión rápida**
4. Retransmisión rápida:
   - Se manda un paquete por cada ack duplicado recibido.
   - Un RTT luego de la retransmisión rápida, el paquete perdido es confirmado.
   - La recuperación rápida termina con esa confirmación de recepción.
5. Luego de recibir el nuevo ack:
   - la ventana de congestión de una conexión se achica a la mitad de lo que era cuando se encontraron 3 duplicados (decremento multiplicativo).
   - El conteo de ack duplicados se pone en 0.
6. Luego la ventana de congestión va incrementando de a un segmento por cada RTT (crecimiento aditivo).
7. Este comportamiento continúa indefinidamente.

Problemas de tener segmentos duplicados retrasados y su resolución:

¿Pueden viajar segmentos duplicados de un host emisor a uno receptor?(OBVIO XD)

- Sí, por ejemplo, cuando se pierde un ack y un segmento se retransmite.
- También cuando por congestión un segmento se demora, expira su temporizador y el segmento se retransmite.

Exigencia: No se pueden entregar segmentos duplicados a la capa de aplicación.
Consecuencia: Por lo tanto es necesario saber si un segmento que llega a un host es duplicado o no.
Soluciones ineficientes:

- Comparar los segmentos bit a bit → sumado a los costos de almacenar los segmentos que llegaron previamente
- Enumerar los segmentos con números de secuencia → funcionaria bien si tuviéramos números de secuencia arbitrarios

No pueden ser arbitrarios porque queremos que los segmentos tengan longitud máxima → el espacio de números de secuencia es finito.
Numeros de secuencia alcanzan el maximo y vuelven a reiniciarse y aumentan hasta alcanzar el maximo de vuelta → espacio de secuencia no es suficientemente grande → duplicado retrasado permanece demasiado tiempo en la red.
Sucede la siguiente situación: un segmento S con n° de secuencia x queda demorado debido a que la red está congestionada.

- El temporizador de retransmisiones asociado a S expira y se retransmite S.
- El protocolo de enrutamiento cambia las rutas y la retransmisión de S llega rápido a destino.
- Pero aun quedó en la red un duplicado retrasado de S (de n° de secuencia x).
- Ese duplicado retrasado de S más adelante llega a destino generando problemas.

Este tipo de problemas es tan serio que debe ser evitado.

Para evitarlo:

- Asegurar que ningún paquete viva más allá de T sec
- Esto se refiere a paquetes de datos, retransmisiones de ellos y a confirmaciones de recepción.
- Eliminar los paquetes viejos que dan vueltas por la red.

Duplicados retrasados en una sola conexión

- El origen etiqueta los segmentos con número de secuencia que no va a ser reutilizados en T sec.
- El espacio de secuencia debe ser lo suficientemente grande para garantizar esto → logra que los segmentos viejos con el mismo número de secuencia hayan desaparecido después de que se regrese al principio de los números de secuencia.

Evitar que duplicado retrasado que pasa de una conexion a otra genere problemas:

- Una conexión debe tener un número de secuencia inicial de secuencia con el que comienza a operar.
- Elegir como un numero de secuencia inicial un numero de secuencia que haga imposible o improbable que el duplicado retrasado numero de secuencia x genere problemas. Además se mantiene dentro de una conexión que el origen etiqueta los segmentos número de secuencia que no van a utilizarse dentro de los T sec.

Implementaciones:

1. al crear una nueva conexión cada extremo genera un número de secuencia de 32 bits aleatorio que pasa a ser el número inicial para los datos enviados. → Alguna implementación de TCP usa esta solución. → La probabilidad de que un paquete duplicado retrasado genere problemas en una conexion es baja debido a la eleccion aleatoria del numero inicial de secuencia de la siguiente conexion.
2. vincular número de secuencia de algún modo al tiempo y para medir el tiempo usar un reloj. Cada host tiene un reloj de hora del dia. Los relojes de los hosts no necesitan ser sincronizados. Cada reloj es un contador binario que se incrementa a sí mismo en intervalos uniformes. El reloj continúa funcionando ante la caída de un host. → Cuando se establece una conexión los k bits de orden de mayor del reloj = número inicial de secuencia.

Si el reloj se mueve mas rapido que la asignacion de numeros de secuencia a los paquetes que se envian → el numero inicial de secuencia de una nueva conexion va a ser mayor al numero de secuencia de cualquier duplicado retrasado de una conexion previa.

Establecimiento y liberación de conexiones:

Como al establecer una conexión se usan segmentos, una conexión debería tener un N° inicial de secuencia con el que comienza a operar.

Idea: vincular N° inicial de secuencia de algún modo al tiempo, y para medir el tiempo, usar un reloj.

Implementación de la idea (de Tomlinson):

- Cada host tiene un reloj de hora del día.
  • Los relojes de los hosts no necesitan ser sincronizados, se supone que cada reloj es un contador binario que se incrementa a sí mismo en intervalos uniformes.
  • El reloj continúa operando aun ante la caída del host
  • Cuando se establece una conexión los k bits de orden mayor del reloj = número inicial de secuencia.

Para lograr que al regresar al principio de los n° de secuencia, los segmentos viejos con el mismo n° de secuencia hayan desaparecido hace mucho tiempo el espacio de secuencia debe ser lo suficientemente grande.

Problema: Cuando un host se cae, al reactivarse sus ET no saben dónde estaban en el espacio de secuencia.

- Este es un problema porque para el siguiente segmento a enviar no se sabe qué números de secuencia generar.
  - si se genera mal, entonces el nuevo segmento podría tener el mismo número de secuencia que otro segmento distinto circulando por la red.
- Solución: requerir que las ET estén inactivas durante T segundos tras una recuperación para permitir que todos los segmentos viejos expiren.

Establecimiento de Conexión: Acuerdo a tres bandas

Para establecer conexión el host de origen envía un segmento CONNECTION REQUEST(N,P) al destino y espera una respuesta CONNECTION ACCEPTED.
Donde N es el número de secuencia y P el puerto.

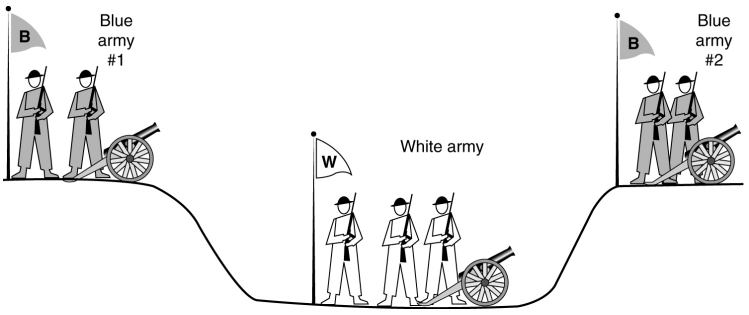
¿Qué pasa si hay un duplicado de **CONNECTION REQUEST**?
No tenemos forma de saber si un segmento **CR** conteniendo un n° de secuencia inicial es un duplicado de una conexión reciente o una conexión nueva.
¿Cómo lo solucionamos ?
Host 1 (emisor) sabe las conexiones que creó, entonces si recibe un CONNECTION ACCEPTED duplicado(mismo número de secuencia N) va a rechazar la conexión. Luego host 2 (receptor) se da cuenta que fue engañado por un duplicado y abandona la conexión.

Establecimiento de una conexión TCP:

- El n° de secuencia inicial de una conexión no es 0
- Se usa un esquema basado en reloj con un pulso de reloj cada 4 μ sec.
- Al caerse un host, no podrá reiniciarse durante el tiempo máximo de paquete 120 seg, para asegurar que no haya paquetes de conexiones previas vagando por Internet.

Campos del encabezado TCP para el establecimiento de conexiones:
SYN se usa para establecer conexiones:
Solicitud de conexión: SYN = 1 y ACK = 0.
Respuesta de conexión: SYN = 1 y ACK = 1.

Liberación de Conexiones: problemas de los dos ejércitos



Si los dos ejércitos azules atacan simultáneamente van a ganar. Por eso quieren sincronizar su ataque. Es muy difícil que ocurra esto porque se necesita saber si el ejército recibió el mensaje…

No existe un protocolo que resuelva el problema de los dos ejércitos.
Para el caso de liberación de conexión “atacar” equivale a “desconectar”. Si ninguna de las partes está preparada para desconectarse hasta estar convencida que la otra está preparada para desconectarse también, nunca ocurrirá la desconexión.

Idea 2: Vamos a permitir que cada parte decida cuando la conexión está terminada.
La liberación de conexión en un host significa que la ET remueve la información sobre la conexión de su tabla de conexiones abiertas y avisa de alguna manera al dueño de la conexión.
Se estudian en 4 casos:

1. Caso Normal → Host 1 envía un disconnection request e inicia temporizador para el caso que no llegue DR a host 2 → llega DR a host 2 → host 2 emite un DR e inicia un temporizador → host 1 recibe DR → envía ACK a host 2 y libera conexión → host 2 recibe el ack y libera la conexión
2. Si se pierde el último ACK → expira el temporizador y la conexión se libera
3. Si se pierde el segundo DR → host 1 no recibe la respuesta esperada → expiran los temporizadores → comienza todo de vuelta
4. Respuesta perdida y DRs subsiguientes perdidos → Tras N intentos el emisor se da por vencido y libera la conexión → Expira el temporizador del receptor y se desconecta.

Este protocolo falla si se pierde el DR inicial y N retransmisiones → El emisor se da por vencido y libera la conexión → el otro lado no se entera de los intentos de desconexión y quedará siempre activo. Conexión abierta a medias.
Soluciones de conexiones abiertas a medias:

1. Obligar al emisor seguir insistiendo hasta recibir una respuesta → si se permite que expire el temporizador del otro lado → el emisor insistirá eternamente, nunca va a aparecer una respuesta.
2. Si no ha llegado ningún segmento a host 2 durante una cierta cantidad de segundos → host 2 libera la conexión. → host 1 detecta falta de actividad y se desconecta. Resuelve el caso que la red se rompió y los hosts ya no pueden conectarse.

Implementación de la solución 2:

- Es necesario que cada ET tenga un temporizador que se detenga y se reinicie con cada envío de un segmento.
- No se puede garantizar absolutamente que cuando se libera la conexión, no ocurra pérdida de datos → se puede limitar esta situación.

Liberación simétrica:

- Cada parte se cierra por separado, independientemente de las otras.
- Una de las partes emite un DISCONNECT porque ya no tiene más datos para enviar y aun está dispuesta a recibir datos de la otra parte.
- Una conexión se libera cuando ambas partes emitieron un DISCONNECT

Es ideal cuando cada proceso tiene una cantidad fija de datos por enviar y sabe con certidumbre cuando los ha enviado. En otras situaciones la determinación de si se ha efectuado o no todo el trabajo o si debe determinarse o la conexión no es tan obvia.

Liberación de una conexión TCP:

Campo usado por TCP para liberación de conexiones. FIN especifica que el emisor no tiene más datos que transmitir (FIN prendido en 1). Tras cerrar una conexión, un proceso puede continuar recibiendo datos indefinidamente. Cuando ambos sentidos de la conexión se han apagado, se libera la conexión.

Para liberar una conexión cualquiera de las partes puede enviar un segmento TCP con el bit FIN establecido lo que significa que no tiene más datos por transmitir, pero todavía puede recibir datos del otro lado. Al confirmarse la recepción del FIN ese sentido se apaga (el receptor del ack no va a enviar más).

##FIJENSE SI HAY ALGO MÁS DE ESTO QUE QUIERAN AGREGAR

Administración del temporizador del TCP:

¿Qué tan grande debe ser el intervalo de expiración del temporizador de retransmisión?

- Si se hace demasiado corto entonces ocurrirán retransmisiones innecesarias
- Si se hace demasiado largo sufrirá el desempeño por el gran retardo de retransmisión de cada paquete perdido

Idea: Ajustar constantemente el intervalo de expiración del temporizador, con base en mediciones continuas del desempeño de la red. Se soluciona con →

Algoritmo de Jacobson(1988) usado por TCP:

Por cada conexión el TCP mantiene una variable, **RTT** (round trip time) → significa estimación actual del tiempo de ida y vuelta al destino.

Al enviarse un segmento se inicia un temporizador, para saber el tiempo que tarda el ack y para habilitar una retransmisión si se tarda demasiado.

Si llega el ack antes de expirar el temporizador: TCP mide el tiempo que tardó el ack digamos M, entonces actualiza el RTT así:

➢ RTT = α RTT + (1-α) M, α es el peso que se le da al valor anterior Por lo común α= ⅞.

Dado un RTT, hay que elegir una expiración adecuada del temporizador de retransmisión.

Solución 1: En las implementaciones iniciales.

Expiración del temporizador = 2 x RTT.

Solución 2: hacer que el valor de timeout sea sensible tanto a la variación de RTT como a la varianza de la función de densidad de probabilidad del tiempo de llegada de los ack.

- Se mantiene una variable amortiguada D la desviación media
- Al llegar un ack se calcula |RTT - M|
- Se mantiene en D mediante D = β D + (1 β)|RTT - M|, donde β típicamente es ¾.
- D es una aproximación bastante cercana a la desviación estándar.

La mayoría de las implementaciones TCP usan ahora este algoritmo y establecen expiración del temporizador = RTT + 4 x D.

Para estimar el temporizador de retransmisiones en ese caso → **algoritmo de Karn (lo usan la mayoría de las implementaciones TCP)**

- No actualizar el RT de ninguno de los segmentos retransmitidos
- Cuando ocurre un timeout se duplica la expiración del temporizador.
- Tan pronto se recibe un ack de segmento no retransmitido, el RTT estimado es actualizado y la expiración del temporizador se computa nuevamente usando la fórmula anterior.

UDP (protocolo de datagramas de usuario):

Es no orientado a la conexión.

- segmentos = encabezado de 8 B + carga útil.

- 2 puertos de 16 b.
- El campo **longitud UDP** incluye el encabezado de 8 bytes y los datos.

Especialmente útil en las situaciones cliente servidor.
El cliente envía una solicitud corta al servidor y espera una respuesta corta.
Si se pierde la solicitud o la respuesta, el cliente puede probar nuevamente.

Capa de Red

La Capa de Red – Generalidades:

El propósito de la capa de red es llevar paquetes de un host de origen a uno de destino siguiendo una ruta conveniente.

Asuntos de los que se encarga la capa de red:

- Almacenamiento y reenvío
- Enrutamiento
- Control de congestión
- Conectar redes de distintas tecnologías (Interredes)
- Fragmentación

Aclaración de Duran (y lo escribió Sofi🦄. Sofi no me pegues :'c):
El control de congestión usando solo la capa de transporte no es del todo eficiente, lo más que puede hacer el emisor es asumir que perdió un paquete. Pero si hay una capa de red, el host emisor se podría enterar antes de la congestión, para que se reduzca la tasa de transferencia.

Cómo es el hardware subyacente a la CR:

Hardware subyacente de la capa de red es:

- Subred: formada por enrutadores interconectados
- Hosts o LANs conectadas a subred
- Varias subredes de distinta tecnología unidas entre sí usando puertas de enlace

No puede pasar un paquete tal cual de una red a otra. ¿Por qué?

- Formatos de paquetes y tamaños máximos difieren de una red a otra.

Enfoques usados para envío de paquetes entre dos hosts:

- Para entender dos maneras existentes de hacer el enrutamiento en las subredes.

Enfoques para mandar un conjunto de paquetes desde un host de origen a un host de destino:

- Usar una ruta fija para mandar todos los paquetes (servicio orientado a la conexión).
- La ruta puede cambiar, por lo que distintos paquetes pueden seguir distintos caminos (servicio no orientado a la conexión).

Servicio no orientado a la conexión:

- Alentado por la comunidad de internet.
- Los paquetes se enrutan de manera independiente.
    - La ruta a usar entre los hosts va a cambiar cada cierto tiempo.
    - Cada paquete debe llevar una dirección de destino completa.
- Nomenclatura usada:
    - Paquetes = datagramas