

# Redes Bloque II

Tema 4

## Protocolos de transporte

### Funcionalidad del nivel de transporte

Objetivo → dar servicios que permitan la comunicación entre procesos que se ejecutan en la capa de aplicación.

Los protocolos se ejecutan en los nodos terminales que se están comunicando HOST FINALES

Tienen como objetivo cubrir todas las carencias que da el protocolo de nivel de red utilizado en el nivel de control de errores y ofrecer al nivel superior una comunicación fiable.

## **PROTOCOLOS → TCP/IP: TCP UDP SCTP QUIC**

Servicios:

Direccionamiento: identifica los procesos en sus puertos, un proceso de un extremo de la conexión que requiere servicios desde un proceso remoto, servidor. Al haber varios procesos es necesario tener una dirección del nivel de transporte, TSAP en TCP y UDP para elegir entre los múltiples procesos que se ejecutan en una máquina.

Números de puerto son enteros sin signo de 16 bits con 3 rangos:

Bien conocidos (0-1023), Registrados (1024-49151),  
dinámicos (49152-65535)

El proceso cliente elige (define) un número de puerto arbitrario

- Puerto efímero (normalmente elegido dentro los dinámicos)
- El proceso servidor también define su número de puerto, pero no de forma arbitraria
- Número de puerto bien conocido

- Puerto de un servicio estándar (ej: 80)
- Puerto de un servicio no estándar

3 identificadores:

Dirección IP, protocolo y número de puerto. La combinación se denomina dirección socket

- <protocolo,IP,puerto>
- Ej: <tcp,150.215.12.37,80>, <udp,180.220.21.33,9>
- Un protocolo de transporte necesita un par de direcciones de sockets (para cliente y servidor)



Seguimiento de la comunicación individual: Protocolos con un transporte fiable de datos, control de congestión y flujo.

Segmentación y reensamblado de datos: Segmentar los datos del usuario en varios mensajes

Multiplexación: Multiplexar varias conexiones de transporte en una sola.

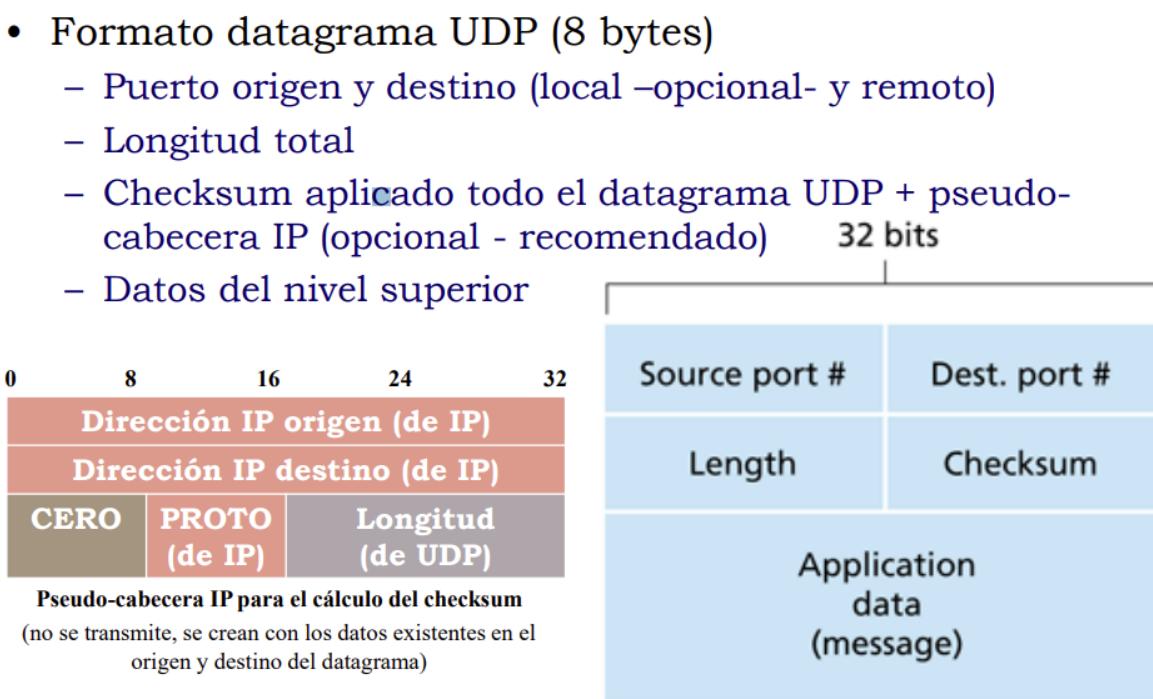
Relacion muchos-uno donde acepta mensajes de distintos procesos diferenciados por el puerto o uno-muchos donde recibe datagramas y entrega el mensaje al proceso adecuado basándose en el puerto.

Control de errores: Asegura que todo el mensaje llega, se corrigen los errores normalmente con retransmisiones.

Control de flujo: Permite controlar la velocidad de datos enviados por el emisor, se controla mediante técnicas de ventana deslizante.

## **PROTOCOLO UDP**

- Características
  - Orientado al mensaje: envía datagramas
  - Protocolo no orientado a la conexión
  - No fiable
  - No incorpora control de congestión
  - Servicio
    - sendto(puerto origen, puerto destino, datos)
    - recvfrom(puerto origen, puerto destino, datos)
  - Extremos
    - Extremo 1: <udp,IPlocal,Plocal>
    - Extremo 2: <udp,IPremoto,Premoto>



No hay control de flujo, ni de errores ni fase de conexión o desconexión. El mensaje es encapsulado en un datagrama IP

Colas:

Cada puerto lleva asociadas 2 colas, entrada y salida.

UDP saca de la cola de salida , añade la cabecera y entrega a IP.

Al llegar un mensaje se comprueba si hay cola de entrada para el puerto, en caso de que haya se deposita al final de la cola, si no existe se descarta el datagrama y se pide a ICMP que envíe un mensaje de puerto no alcanzable (unreachable port)

Todos los mensajes de un proceso son encolados en la misma cola de salida da igual el destino.

Todos los mensajes recibidos son encolados en la misma cola de entrada da igual el origen.

#### Ventajas frente a TCP

- Más rápido (no requiere conexión, sin control de errores)
- Menor penalización en la transmisión al tener una cabecera más corta
- Uso:
  - Envío de mensajes pequeños
  - Protocolos de nivel de aplicación tipo mensaje/respuesta (ej: daytime)
  - Protocolos de nivel de aplicación tolerantes a fallos
  - No preocupa la fiabilidad (ej: envío de datos multimedia)
  - Necesidad de envíos Multicast/broadcast

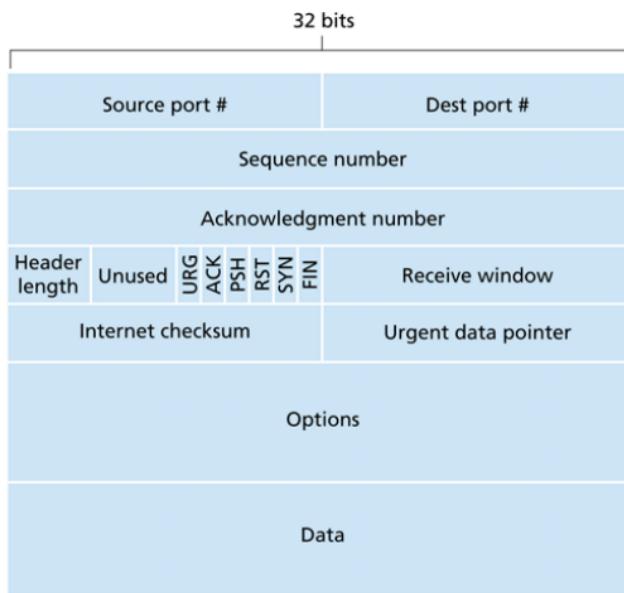
### **PROTOCOLO TCP**

Envía segmentos TCP

- Características del servicio
- Protocolo orientado a la conexión
- Las conexiones son full-duplex
- Canales punto a punto (no broadcasting ni multicasting)
- Ofrece un servicio fiable
- Conexión orientada a flujos de bytes
- No a flujo de mensajes
- No orientado a datos estructurados (stream de bytes sin tipo)
- Buffering
  - Los segmentos TCP se envían a un buffer de envío

- TCP enviará los datos en el buffer de envío cuando lo estime oportuno (distintos algoritmos – configurable)
- Existe también un buffer de recepción (los datos se leen a petición del protocolo de la capa superior)
  - MSS (Maximum Segment Size)
- Máxima cantidad de datos que puede llevar un segmento TCP
- Se negocia durante el establecimiento de conexión
- Depende de la implementación
- Ej.: 1460 bytes en Ethernet

- 
- Estructura del segmento TCP
    - Cabecera de 20 bytes + opciones



### Campos de la cabecera TCP

Puertos origen y destino (16): Son los TSAPs que identifican extremos de la conexión.

Número de secuencia y confirmación, ambos de 32 bits

Longitud de la cabecera (4) en palabras de 32 bits

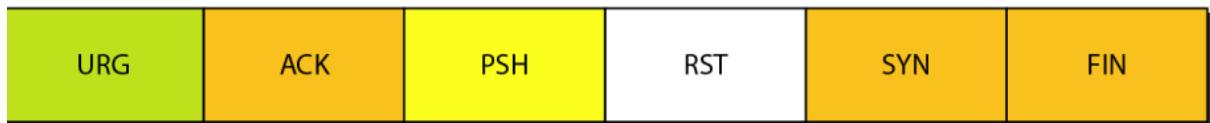
Tamaño de ventana (16) para control de flujo.

Bit ACK: flag de confirmación

Bit RST SYN FIN: Para establecer/finalizar conexiones.

URG: Urgent pointer is valid  
ACK: Acknowledgment is valid  
PSH: Request for push

RST: Reset the connection  
SYN: Synchronize sequence numbers  
FIN: Terminate the connection



### Números de secuencia:

TCP usa el número de byte para enumerar los datos, no comienza en 0 → nº aleatorio [0 -  $2^{32} - 1$ ]

Ejemplo:  
Nº aleatorio = 1057  
y el total de datos = 6000 bytes  
→ Los bytes se enumeran de 1057 a 7056

El número de secuencia es el del primer byte que se transporta

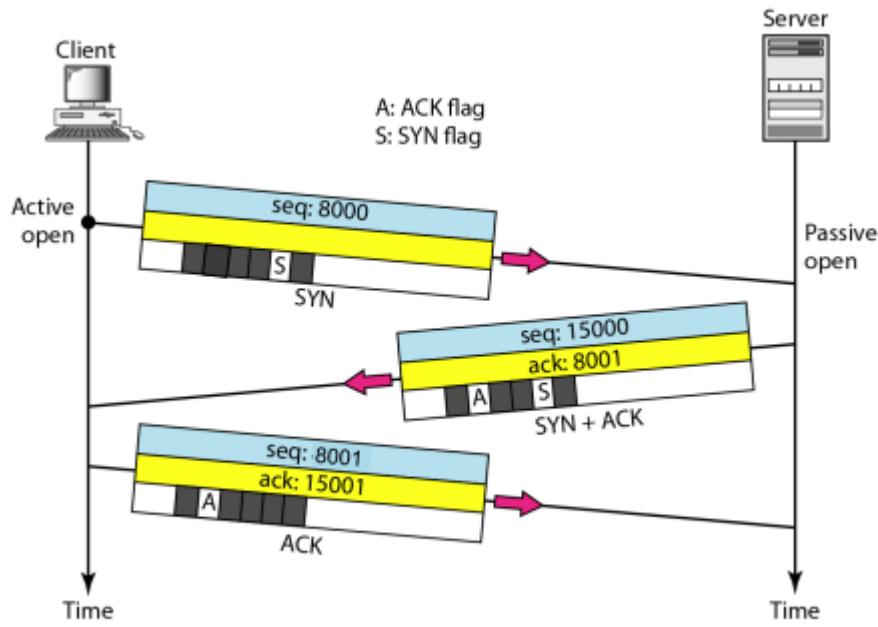
El ACK cuando el flag está activo indica el siguiente byte que se espera recibir.

Si un segmento no lleva datos de usuario no consume número de secuencia.

Excepción de segmentos de control SYN y FIN.

### **PROTOCOLO DE NEGOCIACIÓN A 3 BANDAS O THREE-WAY HANDSHAKE**

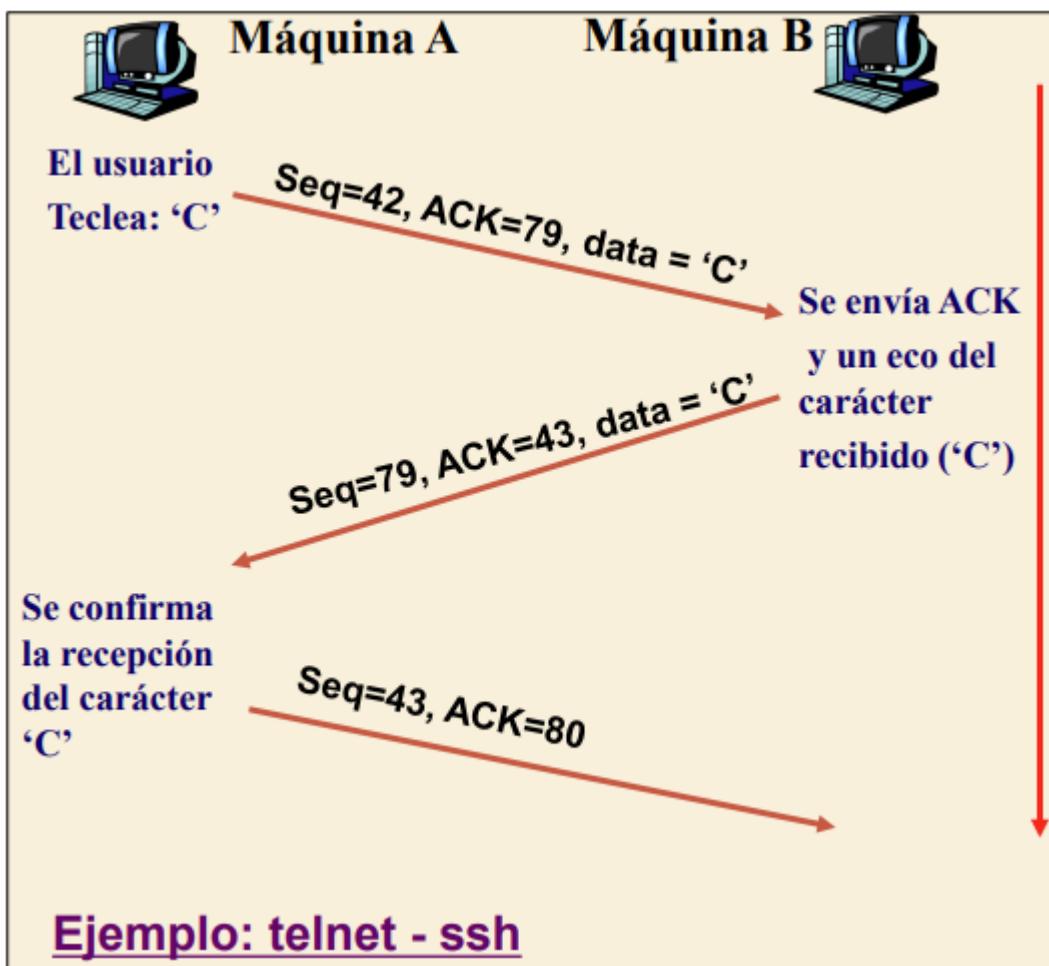
El número inicial se elige aleatoriamente, los segmentos SYN y SYN+ACK no llevan datos pero consumen número de secuencia



ACKs:

Número de secuencia del siguiente byte que se espera recibir

ACK= SEQ + long(datos)



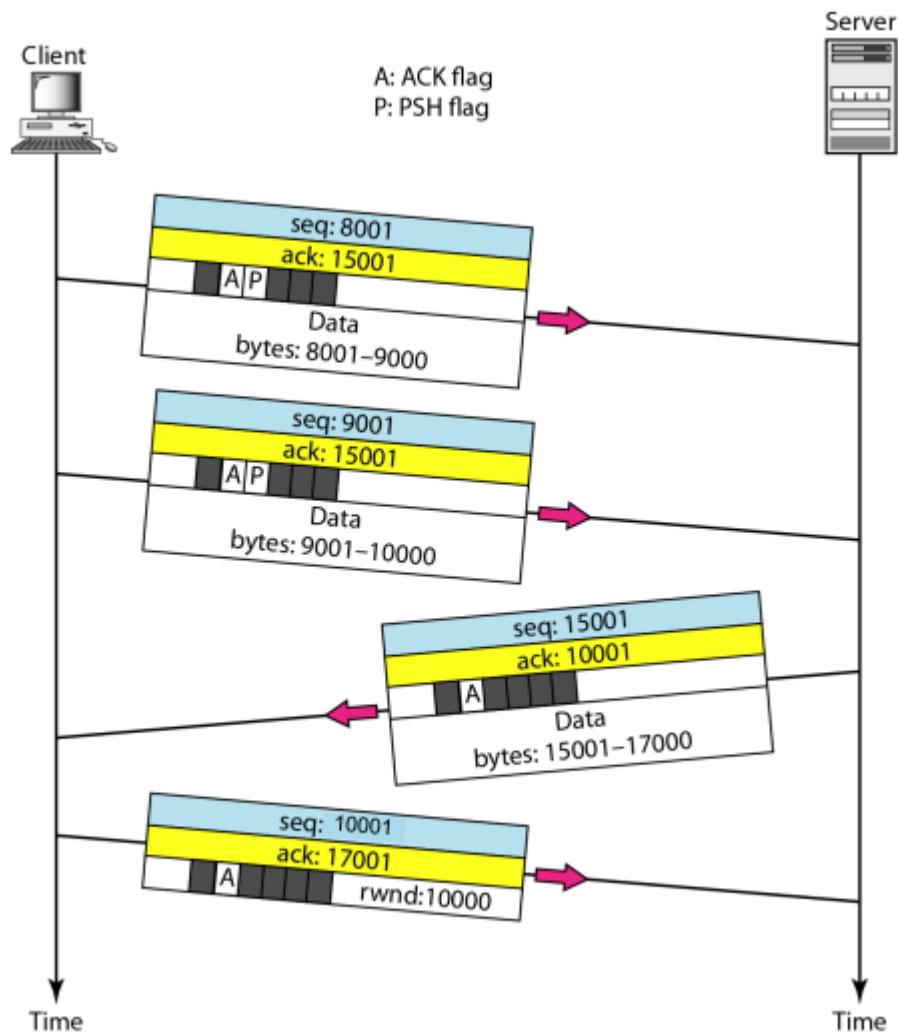
Transferencia de datos TCP:

Transmisión de datos bidireccional:

Ambos extremos pueden enviar datos y confirmaciones, la confirmación se incluye con los datos (Piggybacking)

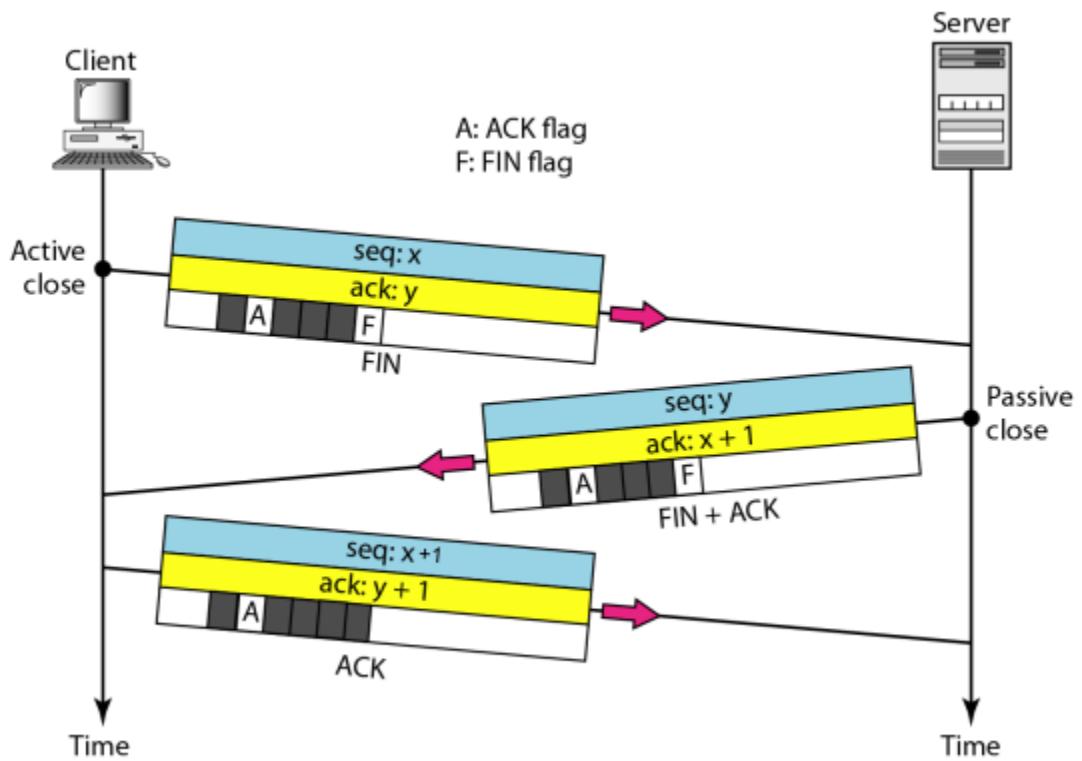
Envío inmediato de datos (Pushing, PSH):

El emisor no debe esperar a completar el buffer, el receptor entrega los datos lo antes posible.



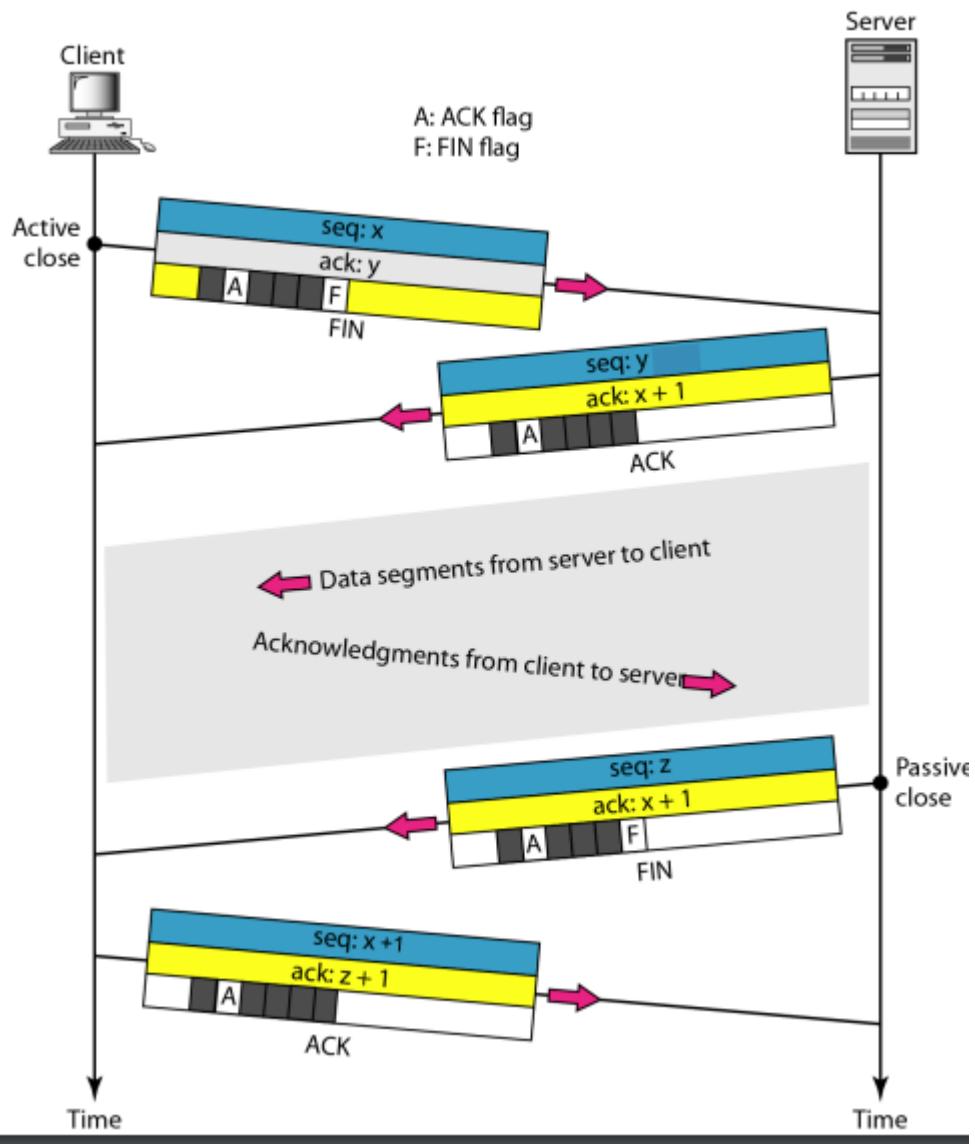
### CIERRE EN 3 PASOS O THREE-WAY HANDSHAKE

Cualquiera de los extremos cierra la conexión



## SEMICIERRE

Uno de los extremos deja de enviar datos pero sigue recibiendo, sin tener datos pendientes que enviar se suele usar este esquema, difícil coordinar close.

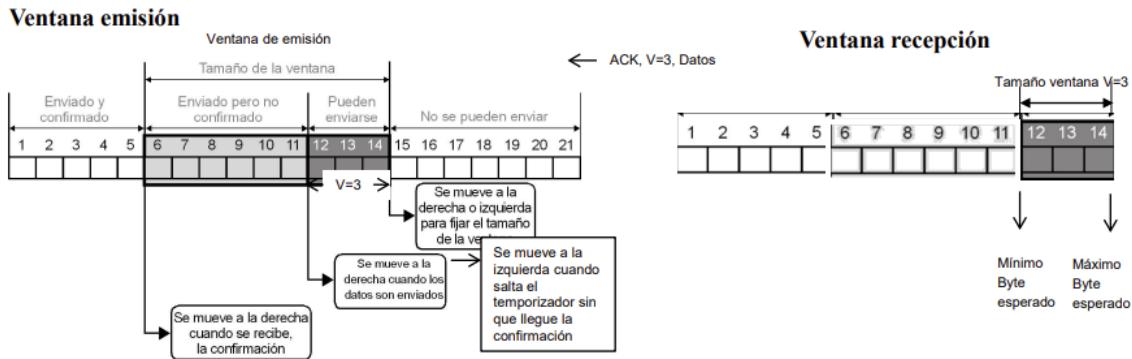


### Control de flujo TCP: sliding window o ventana deslizante

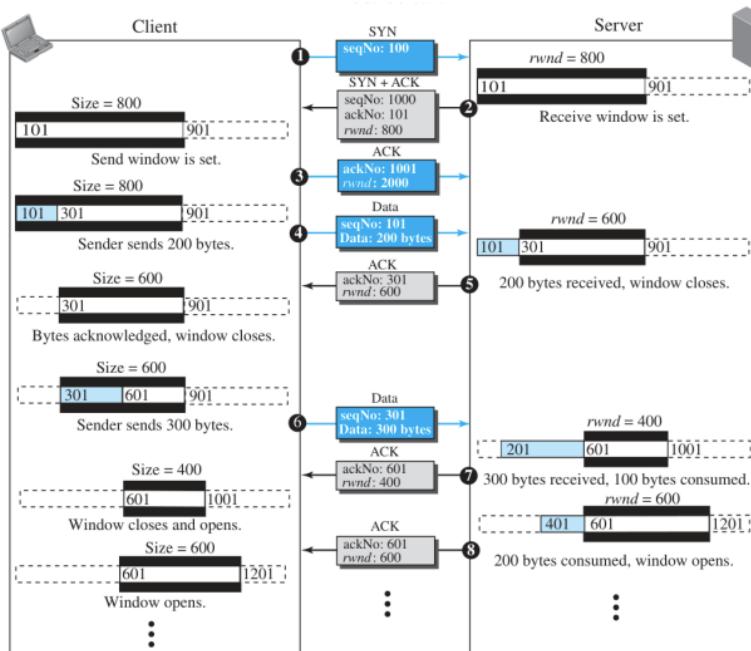
Se usa una ventana deslizante para hacer la transmisión más eficiente, las ventanas son orientadas a byte y son de tamaño variable. Los bytes dentro de la ventana se pueden enviar sin preocuparse de la confirmación.

La ventana de recepción es el valor notificado por el segmento opuesto en un segmento que contiene una confirmación → representa el número de bytes que puede aceptar antes de que su almacén se desborde y tenga que descartar datos.

Hay 4 ventanas para una conexión full-dúplex



## Control de flujo TCP: ventana deslizante (sliding window)



## CONTROL DE FLUJO VENTANA SILLY:

Silly Window Syndrome:

Se produce cuando el emisor o receptor envían o procesan datos de manera muy lenta lo que provoca ajustes excesivos en las ventanas de transmisión y recepción

Cuando la cantidad de datos del usuario enviado son menores que el tamaño de la cabecera de TCP (20 bytes) decimos que la transacción esta sufriendo de síndrome de la ventana silly provocado por implementaciones pobres de TCP

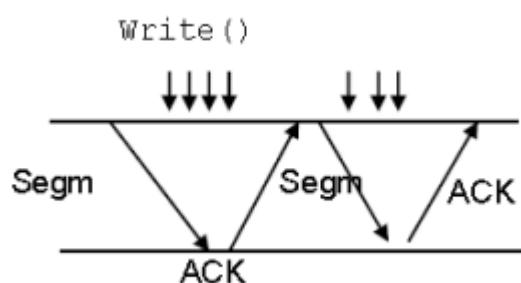
Como solución del lado del emisor podemos aplicar el algoritmo de Nagle

Si no existen datos esperando a ser confirmados → write(datos)  
independientemente del tamaño de estos

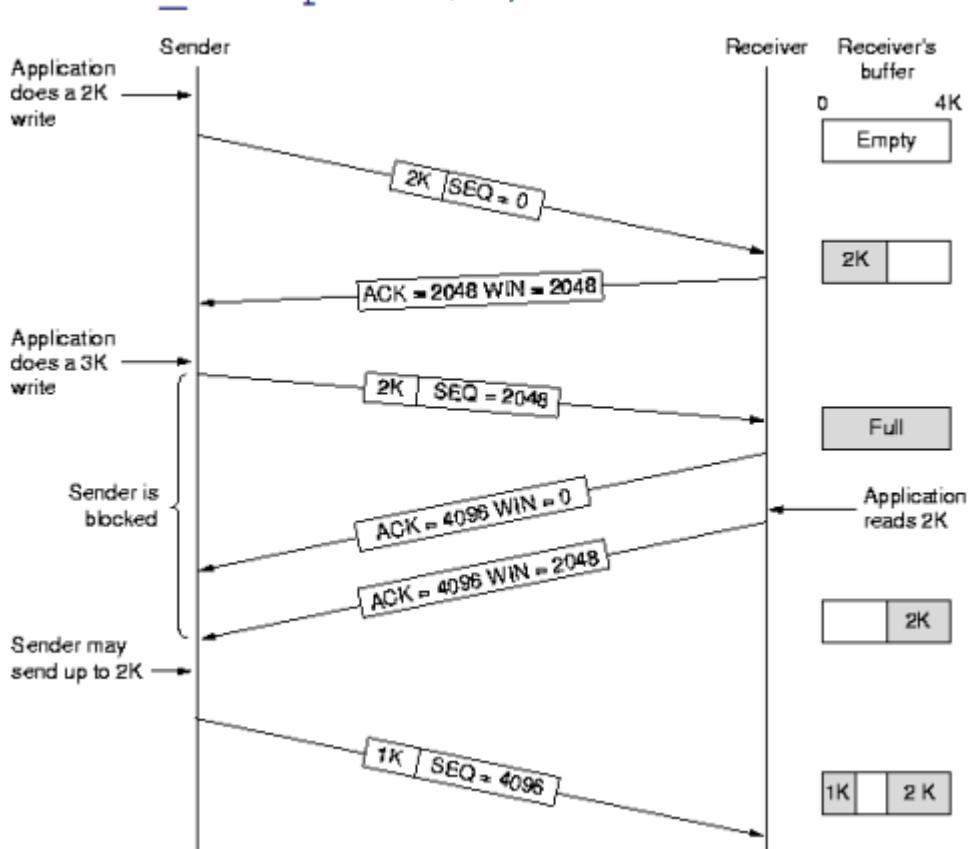
Si hay datos pendientes no se envían nuevos segmentos de datos hasta que:

-Se reciba confirmación de todos los datos pendientes o la longitud de los datos a transmitir sea mayor que MSS

Siempre hay que respetar que la longitud de los datos tiene que ser menor o igual a l tamaño de la ventana de recepción (a.k.a rwnd)



Como solución del lado del receptor tras cerrar una ventana el receptor no debe anunciar una ventana distinta de 0 hasta que no pueda anunciar una ventana de tamaño suficiente, habitualmente este tamaño "suficiente" se fija a  $T = \min(\text{MSS}, \text{buffer\_recepción}/2)$



## Control de Errores en TCP

- TCP incluye mecanismos para detectar y corregir errores por segmentos
  - Corruptos
  - Perdidos
  - Fuera de orden
  - Duplicados
- Estos mecanismos son
  - Suma de comprobación
  - Confirmación acumulativa (positiva)
  - Retransmisión (con temporizadores)
  - Números de secuencia

Suma de comprobación: Detecta segmentos corruptos que son descartados y considerados perdidos.

Confirmación: indican la recepción de segmentos, segmentos de control sin datos pero que estos llevan nº de secuencia se confirman (SYN,FIN). Los segmentos ACK no se confirman

Retransmisión: Se retransmite cuando expira un temporizador de retransmisión o cuando se reciben 3 ACKs duplicados para el mismo número de secuencia, los segmentos ACK no se retransmiten.

Retransmisión después de un RTO (Retransmission time out) RFC 6298: El emisor mantiene un temporizador RTO por conexión, cuando vence envía el primer segmento sin confirmar, no se activa un RTO para los segmentos que solo confirman, el valor de este temporizador es dinámico y se actualiza en base al tiempo de ida y vuelta (RTT), usa su media (SRTT) y varianza (RTTVar)

$$SRTT = (1 - \alpha).SRTT + \alpha.R'; \alpha = 1/8 \text{ y } R' \text{ es una nueva muestra}$$

$$RTTVar = \beta.|SRTT - R'| + (1 - \beta).RTTVar; \beta = 1/4$$

$$RTO = SRTT + MAX(G, 4 * RTTVar); \text{ Se sugiere } G \leq 100\text{ms}$$

Mínimo valor  $\geq 1\text{s}$  y Máximo valor  $\leq 60\text{s}$

Retransmisión después de tres ACKs

Se pierde un segmento y se reciben algunos fuera de orden, un problema si el almacén receptor es limitado

Se reciben 3 ACKs duplicados para el mismo número de secuencia → se reenvía el segmento inmediatamente (retransmisión rápida)

- Emisor TCP simplificado

```

1  sigNumSec=NumeroSecuenciaInicial;
2  baseEmision=NumeroSecuenciaInicial;
3
4  loop(true){
5      switch(suceso){
6          suceso: datos recibidos de aplicación superior
7              Crear segmento TCP con número de secuencia sigNumSec
8              if(el temporizador no está funcionando){
9                  iniciar temporizador
10             }
11             pasar segmento a IP
12             sigNumSec=sigNumSec+longitud(datos);
13             break;
14         suceso: fin de temporización del temporizador
15             retransmitir el segmento no reconocido con el número de secuencia más pequeño
16             iniciar temporizador
17             break;
18         suceso: ACK recibido, con valor de campo ACK igual a y
19             if(y>baseEmision){
20                 baseEmision=y;
21                 if(existen actualmente segmentos aún no reconocidos){
22                     iniciar temporizador
23                 }
24             }
25             break;
26     }
27 }
```

### Segmentos fuera de orden:

Antes se descartaban los segmentos fuera de orden, actualmente se almacenan temporalmente y se marcan como fuera de orden hasta que llega el segmento perdido, al ser marcados no se entregan al proceso perdido. NINGUNO FUERA DE ORDEN SE ENTREGA

### **Control de Congestión**

Conceptos diferentes

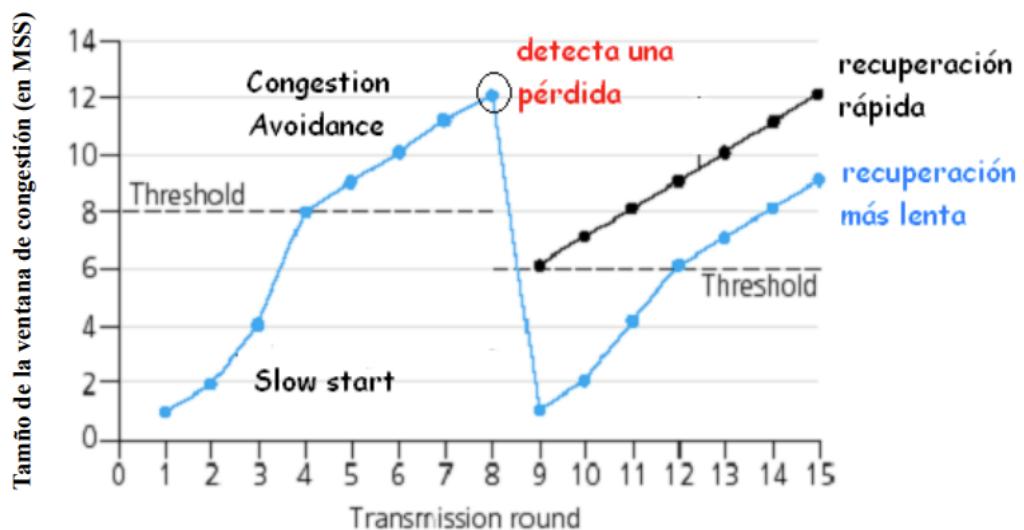
Control de flujo → no mas paquetes de los que puede recibir el receptor

Control de congestión → no mas paquetes en una parte de la red (subred)

- Control de congestión en TCP
  - La detección de congestión se basa en observar los siguientes hechos:
    - El aumento del RTT de los segmentos confirmados, respecto a anteriores segmentos.
    - La finalización de temporizadores de retransmisión
      - ... son causados por congestión en la red
  - Uso de la ventana de congestión (cwnd) para controlar la velocidad de envío
    - Tamaño de la ventana útil =  $\min(\text{rwnd}, \text{cwnd})$

Algoritmo de congestión de TCP RFC5681

Está compuesto de 3 algoritmos, slow start, congestion avoidance (obligatorios) y fast recovery (opcional)



### Slow start:

Incremento exponencial de la ventana, incrementa en 1MSS cada vez que llega un ACK

TCP comienza con cwnd = 1MSS

TCP envía un segmento

Se recibe una confirmación se libra 1 MSS del buffer de recepción

se incrementa cwnd en 1 MSS por cada ACK

cwnd = cwnd + 1 MSS = 2MSS

TCP envía dos segmentos:

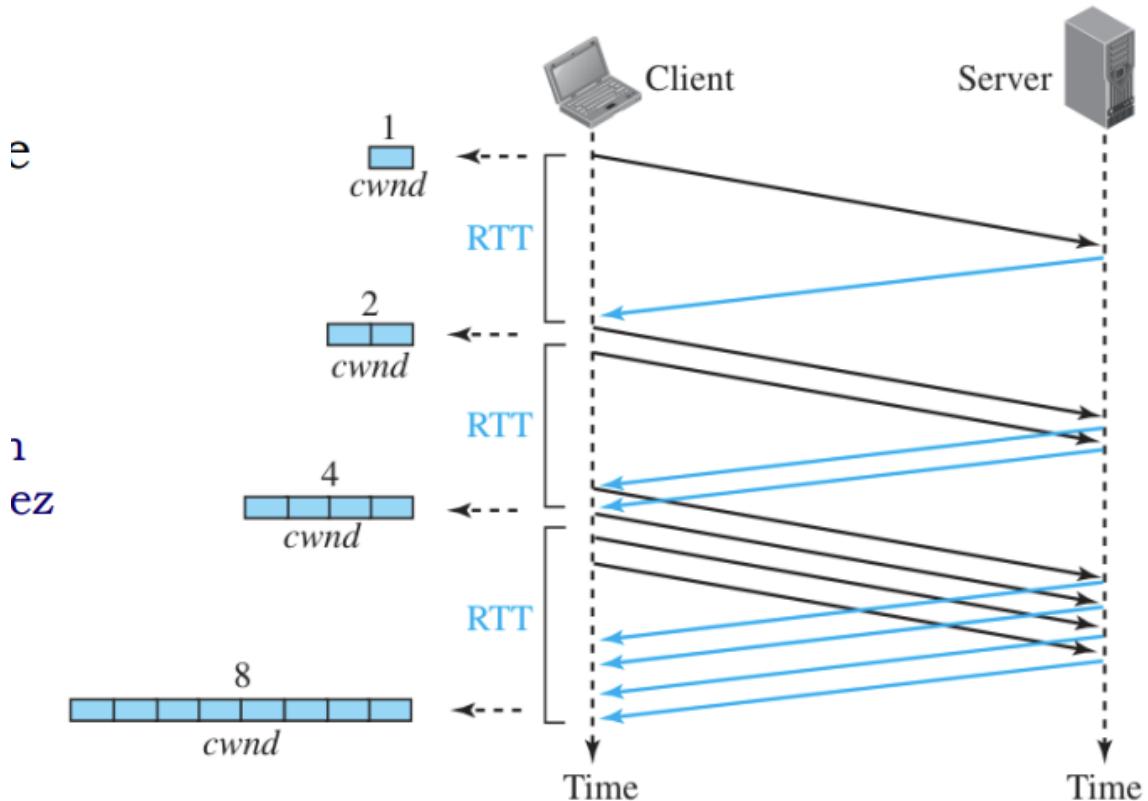
tantos como permite su ventana

se reciben dos confirmaciones y se liberan 2MSS del buffer de recepción

Se incrementa cwnd en 1 MSS por cada ACK

cwnd = cwnd + 2 MSS = 4 MSS

### **EXPONENCIAL**



Congestion avoidance:  
incremento aditivo

El tamaño de la ventana de congestión incrementa en 1 si se confirma toda la ventana enviada.

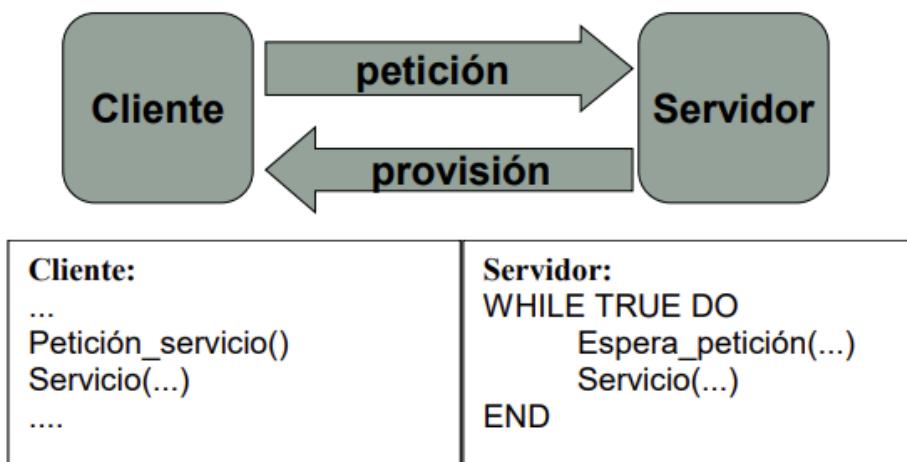
- Si llega un ACK,  $cwnd = cwnd + (1/cwnd)*MSS$

## CAMBIO ENTRE ALGORITMOS

TCP ejecuta un inicio lento, pero cuando  $cwnd$  llega a cierto umbral pasa a la fase de evitación de la congestión, en cualquiera de las fases, si hay un evento de pérdida se vuelve a la fase de inicio lento con  $cwnd = 1MSS$  y el umbral se actualiza al  $cwnd/2$  que provocó la pérdida

- El Stream Control Transmission Protocol (SCTP) fue definido por SIGTRAN de IETF (2000).
  - Características:
    - Orientado al mensaje (como UDP).
    - Orientado a la conexión, confiable, con control de flujo y de congestión y secuenciación (como TCP).
    - Permite opcionalmente envío de mensajes fuera de orden.
  - Ventajas:
    - Multihoming (posibilidad de varias IP en cada extremo de la comunicación).
    - Selección y monitorización de caminos (múltiples flujos).
    - Mecanismos de validación y protección contra ataques.
  - Implementado en diversos sistemas tipo UNIX (Linux, Solaris, FreeBSD, ...).

## PARTE PRÁCTICAS



- Dos tipos básicos de sockets
  - **Sockets TCP:** comunicación orientada a la conexión, fiable
  - **Sockets UDP:** comunicación no orientada a la conexión, no fiable

## Tema 5

Protocolos a nivel de aplicación definen como los procesos de aplicación que se ejecutan en sistemas diferentes, se envían mensajes entre si

Protocolos relacionados Request for Comments (RFCs)

SMTP y HTTP

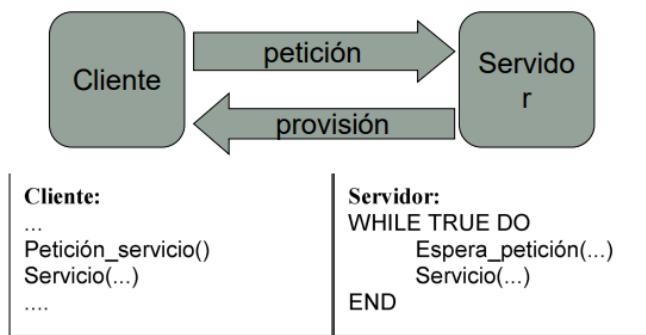
- Ejemplos de aplicaciones:

Aplicación	Pérdida de datos	Ancho de banda	Timing
Transferencia de ficheros	No	Flexible	No
Correo electrónico	No	Flexible	No
Páginas Web	No	Flexible (varios Kbps)	No
Audio/video en tiempo real	Sí	Audio: kbps – 1Mbps Video: 10kbps – 5 Mbps	100s de msegms
Mensajería instantánea	No	Flexible	Pocos segs

Modelos básicos: Cliente/Servidor

Se componen de dos entidades, clientes y servidores, estos últimos son entidades que ofrecen servicios mientras que los clientes los solicitan.

- Esquema:



- Ejemplos:
  - Telnet: terminales virtuales remotos
  - FTP: transferencia de ficheros
- Un sistema puede implementar a su vez un cliente y un servidor

Los procesos servidores son permanentemente activos que ofrecen un servicio concreto siempre disponible para los usuarios.

El servidor tiene una dirección IP fija y un puerto conocido por el usuario

Cientes y servidores de una misma aplicación se comunicarán mediante el intercambio de mensajes utilizando los servicios del nivel de transporte, este intercambio de mensajes se hace mediante sockets

## Direccionamiento

- Puertos
  - Vienen identificados por números enteros
  - Muchos números están asignados a aplicaciones de red concretas
- Fichero /etc/services

Servicio	Puerto	Puerto (TLS/SSL)	Protocolo
ftp-data	20/tcp	989/tcp	File Transfer [Default Data]
ftp	21/tcp	990/tcp	File Transfer [Control]
ssh	22/tcp		Acceso remoto seguro
telnet	23/tcp	992/tcp	Telnet
smtp	25/tcp	465/tcp	Correo electrónico
dns	53/udp		Resolución de nombres
dhcp	67/udp	-	Configuración dinámica
http	80/tcp	443/tcp	World Wide Web HTTP
pop3	110/tcp	995/tcp	Recuperación de e-correo
imap	143/tcp	993/tcp	Recuperación de correo

## Protocolos de transporte

TCP → orientado a la conexión, servicio transporte fiable

UDP → Servicio sin conexión, no fiable

Aplicación	Protocolo de aplicación	Protocolo de transporte
Correo electrónico	SMTP [RFC 2821]	TCP
Terminal remoto	Telnet [RFC 854]	TCP
Acceso a la Web	HTTP [RFC 2616]	TCP
Transferencia de ficheros	FTP [RFC 959]	TCP
Multimedia ( <i>streaming</i> )	Propietario	TCP o UDP
Telefonía por Internet (VoIP)	Propietario	Tipicamente UDP
Resolución de Nombres	DNS	UDP

TCP y UDP no encriptan los datos, por eso se utiliza Secure Sockets Layer o SSL, es una mejora sobre TCP implementada en la capa de aplicación  
SSL encripta, proporciona integridad y autenticación.

Servicio de Dominios, se identifican por la dirección IP, los usuarios utilizan nombres, ej: www.uma.es establecido mediante el sistema de nombres de dominio que es una base de datos distribuida implementada por una jerarquía de servidores de nombres o mediante un protocolo de la capa de aplicación. Función de UDP puerto 53

FQDN: fully qualifies domain name, nombre completo con equipo y dominios con un max de 255 caractertes es decir 63 por dominio

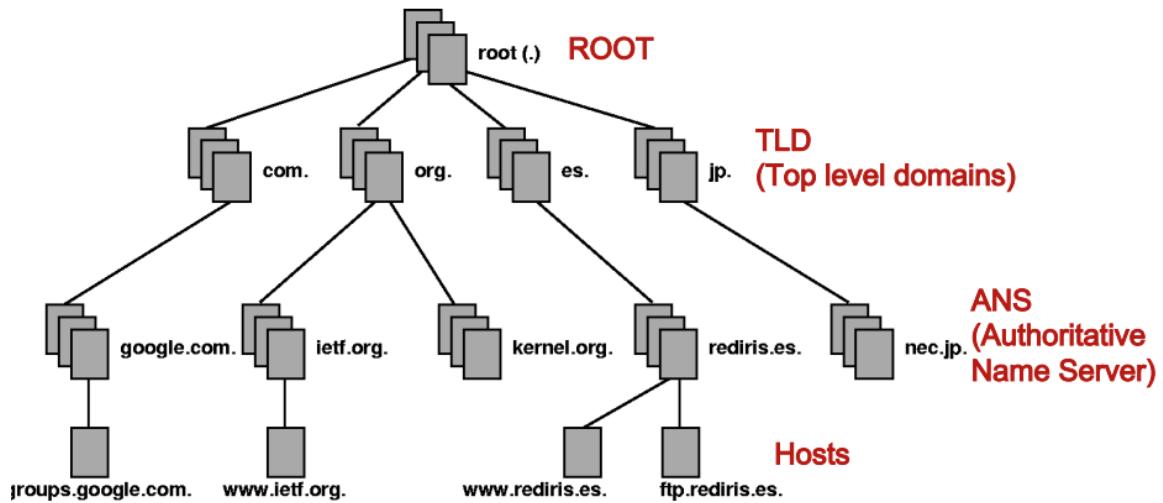


Servicios y esctructura del DNS:

Servicios son una traducción del nombre de host a la dirección IP , mantenimiento de alias y balanceo de carga.

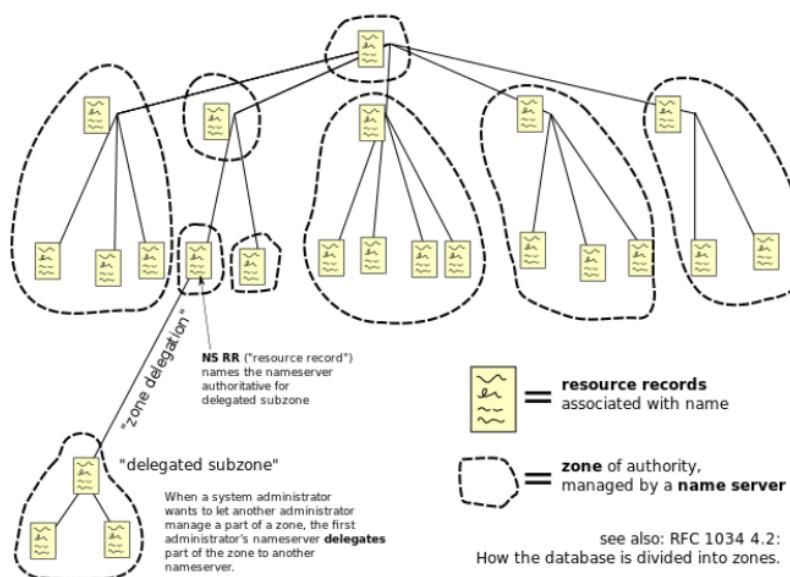
La estructura es jerárquica, no es una sola base centralizada, cada nodo tiene ciertos registros

(equivalencia nombre e ip) y sabe a quien consultar sus subdominios (hijos)



## DNS: Servicios y estructura

- Un mismo organismo (servidor) puede tener almacenado varios nodos del árbol (**zona**)



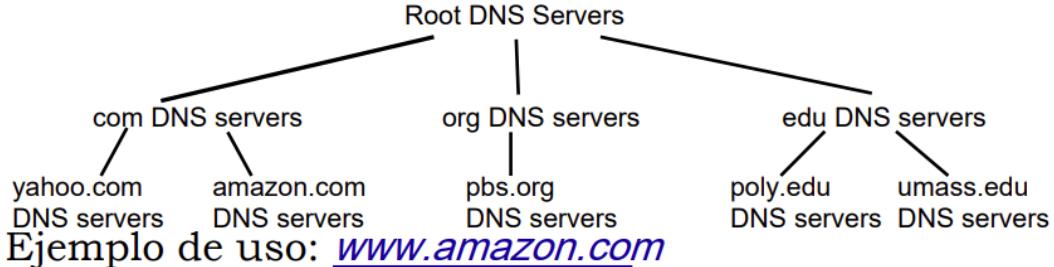
Encontramos 3 tipos de servidores, root (raíz)

Servidores top-level domain (TLD)

Servidores DNS autorizados (ANS)

Servidores locales (LNS)

### — SERVIDORES LOCALES (LNS)



Ejemplo de uso: [www.amazon.com](http://www.amazon.com)

- Consultamos nuestro servidor local (**LNS**)
- LNS solicita **root** server encontrar el servidor DNS (**TLD**) de .com
- LNS solicita al TLD .com obtener el servidor DNS (**ANS**) amazon.com
- LNS pregunta al ANS amazon.com la dirección IP de www.amazon.com

- Servidor de nombre raíz (root name server):
  - Contacta con un servidor de nombres con autoridad si no se conoce una asignación de nombres



### Servidores top-level domain (TLD):

- Responsables de los dominios com, org, net, edu, aero, jobs, museums, y todos los dominios de países: : uk, fr, ca, jp, es, ...

### Servidores DNS autorizados:

- Servidores DNS propios de una organización que proporcionan traducciones autorizadas de los hosts de su organización
- Pueden ser mantenidos por la propia organización o por un proveedor de servicios

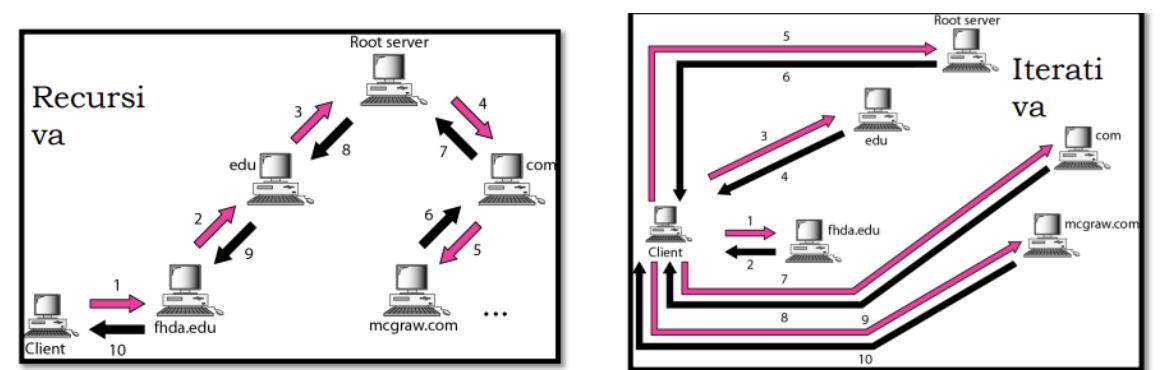
## Servidores locales:

- No pertenece estrictamente a la jerarquía y cada ISP tiene uno:
  - Denominado "servidor de nombres por defecto"
- Cuando un host realiza una consulta (DNS query), ésta es enviada a su servidor DNS local
  - Tiene una cache local con las traducciones más recientes
  - Actúa de proxy, reenviado la consulta hacia la jerarquía

## DNS resolución:

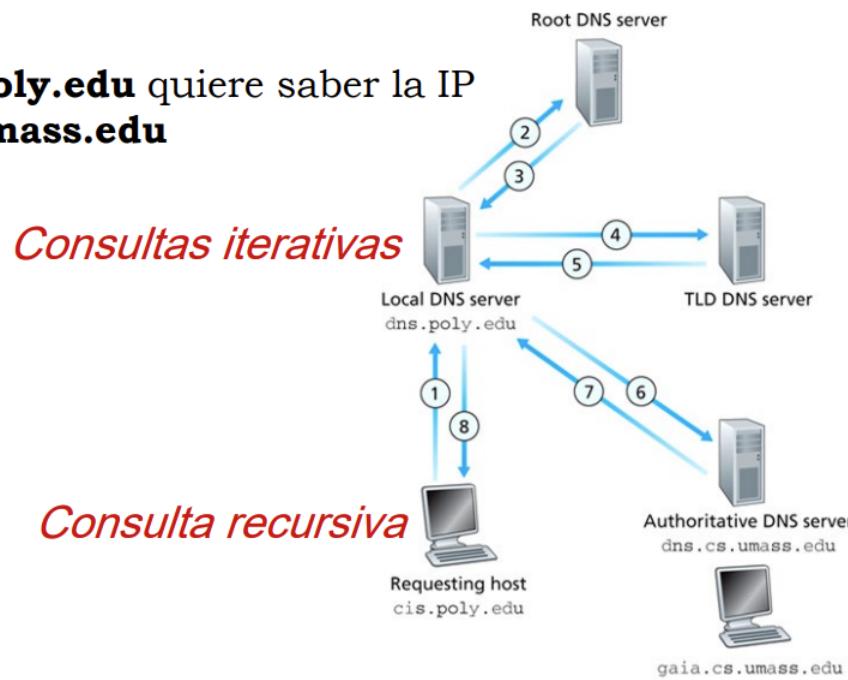
Recursiva (mucha carga en niveles superiores) , el servidor local debe entregar al cliente una respuesta final

Iterativa, si es un servidor de autoridad envía la respuesta de lo contrario devuelve al cliente la dirección IP del servidor que cree que puede resolver el nombre.



- Ejemplo:

El host **cis.poly.edu** quiere saber la IP de **gaia.cs.umass.edu**



DNS cacheado:

Cuando un servidor de nombres aprende un nombre, lo almacena en la caché.

Las entradas del cache son limpiadas periódicamente

Los servidores TLD servers están normalmente guardados en los servidores de nombre haciendo que los servidores raíz no necesiten ser visitados.

Las entradas memorizadas pueden estar desfasadas, cada entrada tiene un tiempo de vida TTL tras el cual expira. Los mecanismos para la actualización y notificación están recogidos en el RFC 2136

### TELNET TERMINAL VIRTUAL

Es una aplicación cliente/servidor de propósito general que permite el establecimiento de una conexión con un sistema remoto de forma que en el terminal local aparece como un terminal de sistema remoto.

Protocolo/puerto: TCP/23

Problemas debido a la heterogeneidad de los sistemas. Los caracteres viajan a través de la conexión en un conjunto de caracteres llamado

## Caracteres de terminal virtual de red NVT

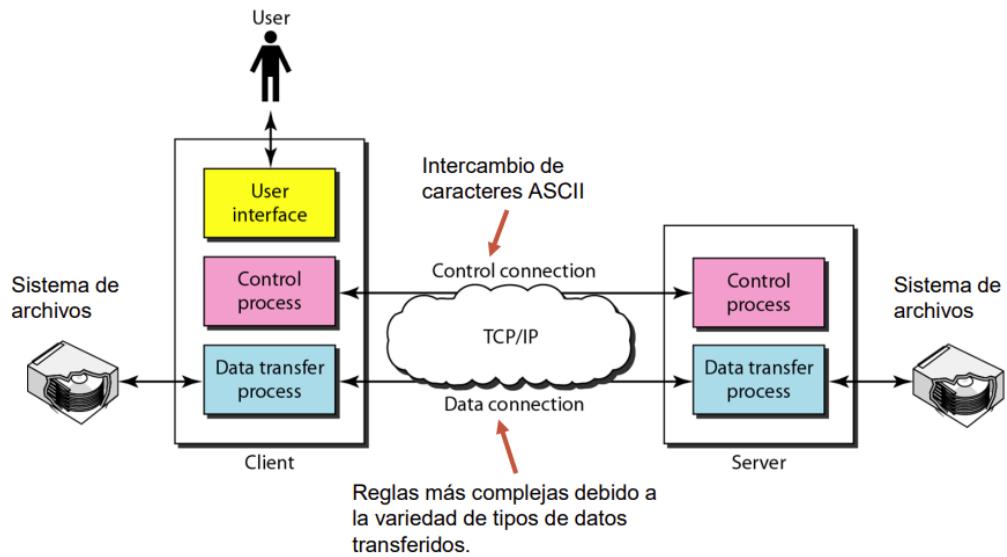
Telnet: en el 79, solo se puede acceder en modo terminal, el mayor problema que tiene es la seguridad ya que los nombres de usuario y contraseñas viajan por la red como texto plano lo que facilita vulnerabilidades.

SSH: desarrollado 1995, con un protocolo de comunicación muy similar pero en el que las comunicaciones viajan de forma encriptada , SSH puerto 22 tcp

## INTERCAMBIO DE FICHEROS

- Características
  - Es un protocolo que permite transferir ficheros entre ordenadores
  - Se basa en el modelo cliente/servidor, usa el protocolo TCP y los usuarios necesitan autorización.
  - Dos modos de transferencia: ASCII de 8bits para su envío de Tipo A
  - Binario los datos se envían byte a byte sin conversión

Operaciones (usuario): open,bye,put,get,mput,mget



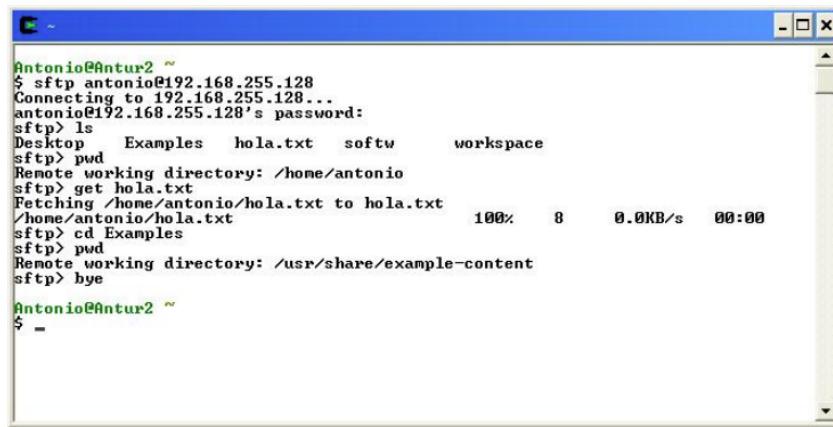
**Conexiones:** Establece dos tipos de conexiones simultáneas entre los computadores involucrados en la transferencia de ficheros:

Otra conexión se dedica al envío de info de control (TCP, puerto 21)

Una conexión se utiliza para la transmisión de ficheros en sí (TCP puerto 20)

Dos estrategias diferentes en cada conexión, la de control permanece abierta mientras dura la sesión, las de datos se crean y destruyen dinámicamente en cada transferencia de fichero.

- Ejemplo de sesión



```

Antonio@Antur2 ~
$ sftp antonio@192.168.255.128
Connecting to 192.168.255.128...
antonio@192.168.255.128's password:
sftp> ls
Desktop Examples hola.txt softw workspace
sftp> pwd
Remote working directory: /home/antonio
sftp> get hola.txt
Fetching /home/antonio/hola.txt to hola.txt
/home/antonio/hola.txt                               100%   8     0.0KB/s  00:00
sftp> cd Examples
sftp> pwd
Remote working directory: /usr/share/example-content
sftp> bye
Antonio@Antur2 ~
$ -

```

## Activo

El cliente crea el socket para la transferencia de datos.

El cliente envía el puerto al servidor mediante un comando PORT

El servidor se conecta (puerto 20) y establece la conexión de datos.

## Pasivo

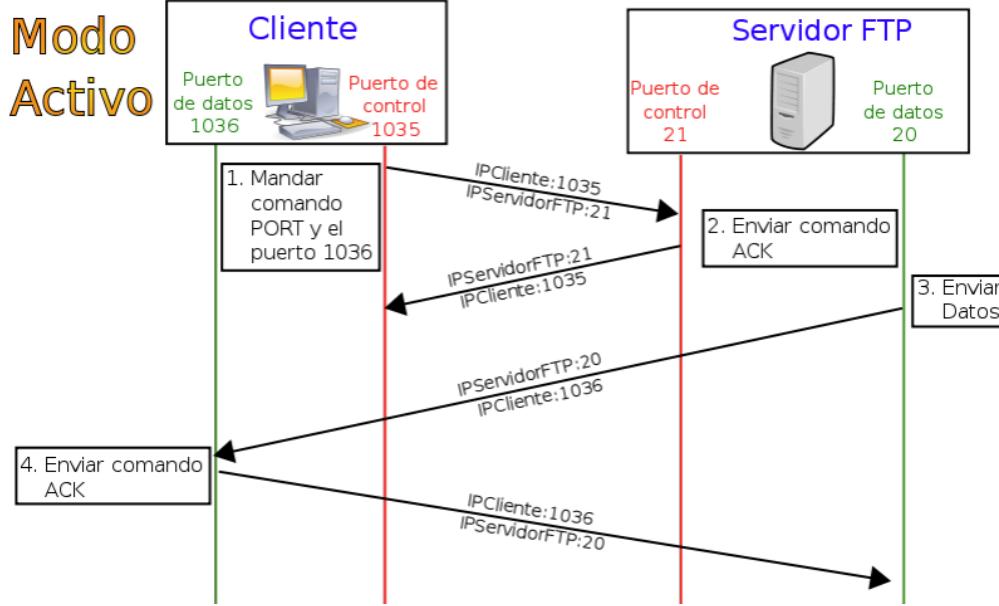
El cliente envía el comando PASV

El servidor crea el socket para la transferencia de datos.

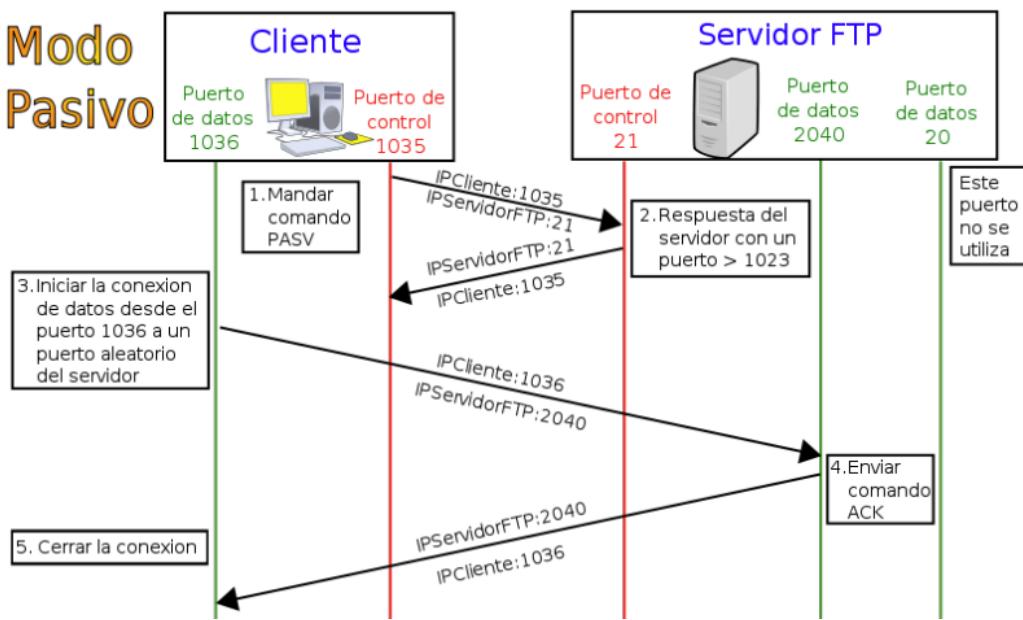
El servidor comunica el puerto al que debe conectarse.

El cliente se conecta a dicho puerto y se establece la conexión de datos.

En el modo activo el cliente envía al servidor (port) el puerto de datos usado y el servidor inicia la conexión:



El cliente envía PASV. Entonces el servidor crea un nuevo puerto e informa al cliente y éste último inicia la conexión



Mensajes (SP: espacio y CRLF: Salto de línea y retorno de carro)

<b>COMANDO</b>	<b>[&lt;SP&gt; parametro]</b>	<b>&lt;CRLF&gt;</b>
USER	<SP> <username>	<CRLF>
PASS	<SP> <password>	<CRLF>
CWD	<SP> <pathname>	<CRLF>
QUIT		<CRLF>
PORT	<SP> <host-port>	<CRLF>
PASV		<CRLF>
TYPE	<SP> <type-code>	<CRLF>
MODE	<SP> <mode-code>	<CRLF>
RETR	<SP> <pathname>	<CRLF>
STOR	<SP> <pathname>	<CRLF>
LIST	[<SP> <pathname>]	<CRLF>
DELE	[<SP> <pathname>]	<CRLF>
HELP	[<SP> <string>]	<CRLF>
NOOP		<CRLF>

Si se requiere transmitir datos, se creará la conexión oportuna. Se enviará un mensaje de respuesta por la conexión de control

<b>XYZ</b>	<b>&lt;SP&gt; Explicación</b>	<b>&lt;CRLF&gt;</b>
<b>X = tipo</b>	<b>Y = función</b>	<b>Z = detalle</b>
<b>1</b>	yz<SP>Positive Preliminary reply	<CRLF>
<b>2</b>	yz<SP>Positive Completion reply	<CRLF>
<b>3</b>	yz<SP>Positive Intermediate reply	<CRLF>
<b>4</b>	yz<SP>Transient Negative Completion reply	<CRLF>
<b>5</b>	yz<SP>Permanent Negative Completion reply	<CRLF>

Ej:

**250<SP>Command successful.<CRLF>**  
**(La respuesta ocupa 24 bytes)**

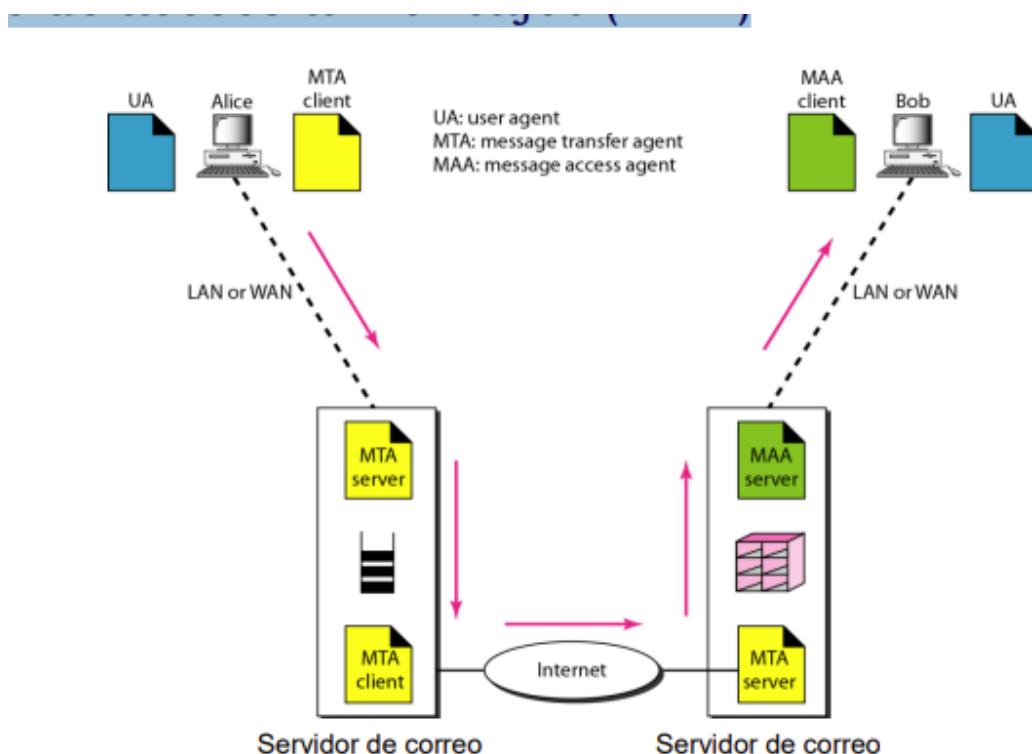
El correo electrónico es un medio de comunicación asíncrono. Las personas envían y leen mensajes cuando les conviene, sin tener que coordinarse con la otra parte.

Supuso una revolución en el momento de su implantación por sus ventajas frente al correo ordinario, es más rápido, barato y fácil de reproducir.

Uno de los servicios más populares de internet y de los más antiguos (anterior a HTTP)

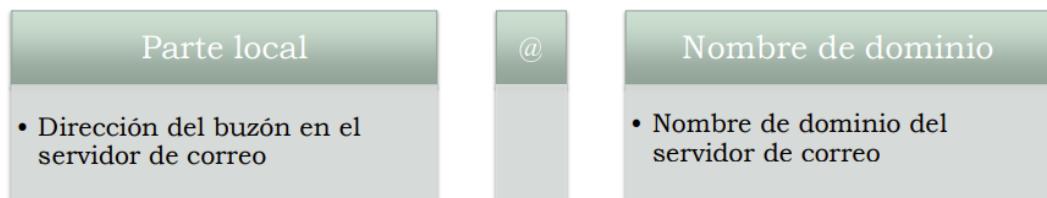
Hay tres componentes involucrados en el envío y recepción de correo electrónico

- Agentes de usuario (UA)
- Agentes de transferencia de mensajes (MTA)
- Agentes de acceso a mensajes (MAA)



- Los servidores de correo son el núcleo de la infraestructura del correo electrónico.

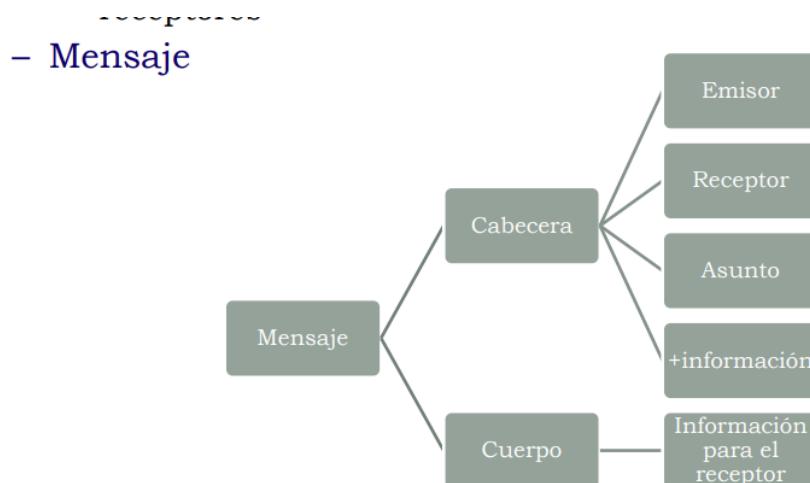
- Cada usuario de correo electrónico tiene un buzón en un servidor de correo
  - Gestionan y mantienen los mensajes enviados a un usuario
  - La dirección de correo electrónico se consigue concatenando el nombre del buzón con el servidor de correo en el que está situado.



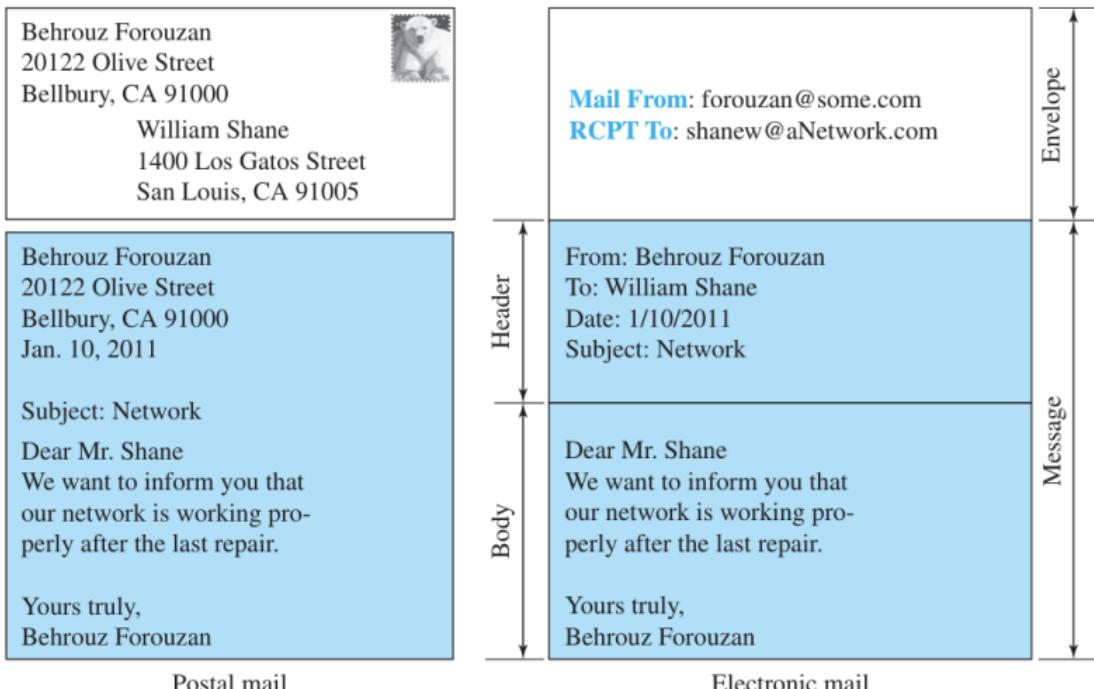
## Partes de un mensaje electrónico

Sobre: contiene las direcciones del emisor y los receptores

Mensaje:



## Correo electrónico



Agente de usuario: ofrece servicios para el envío o recepción de un mensaje

Es un programa que compone, lee , responde y envía mensajes:  
También incluye y gestiona buzones (uno de entrada y otro de salida)

Ejemplos : Eudora, Outlook,Thunderbird

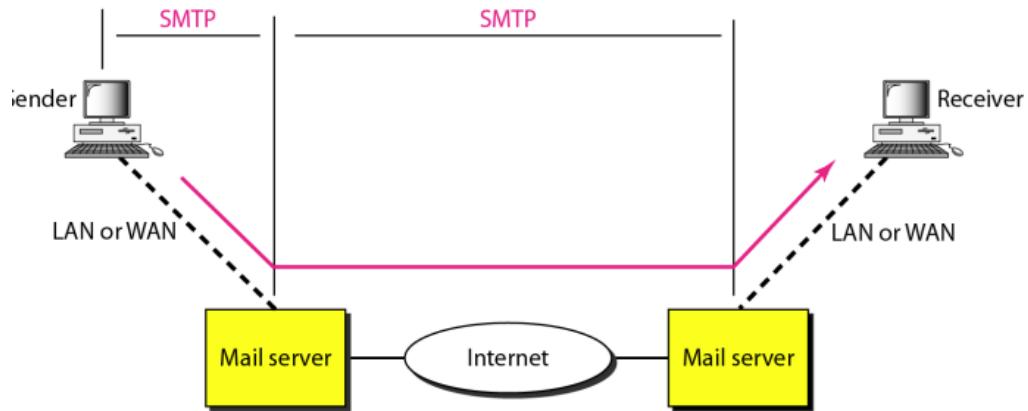


Agente de transferencia de mensajes (SMTP)

Para enviar un correo un sistema debe tener un cliente MTA y para recibir un correo electrónico debe tener un servidor MTA

El protocolo entre el cliente y el servidor es SMTP , se usan dos pares de cliente servidor en la mayoría de las situaciones

Puerto TCP/25



### Mensajes SMTP (implementación mínima)

```
HELO <SP> <domain> <CRLF>
MAIL <SP> FROM:<sender> <CRLF>
RCPT <SP> TO:<receiver> <CRLF>
DATA <CRLF>
RSET <CRLF>
NOOP <CRLF>
QUIT <CRLF>
```

### Respuestas SMTP (Ídem códigos respuestas FTP)

Ejemplo:

```
250<SP>Requested mail action okay completed<CRLF>
```

EJEMPLO

---

```

S: 220 sol10.lcc.uma.es Simple Mail Transfer Service Ready
C: HELO sol10.lcc.uma.es
S: 250 sol10.lcc.uma.es
C: MAIL FROM: gabriel@lcc.uma.es
S: 250 OK
C: RCPT TO: rusman@lcc.uma.es
S: 250 OK
C: RCPT TO: mercedes@lcc.uma.es
S: 550 No such user here
C: DATA
C: Subject: Reunión de RySD
C: To: rusman@lcc.uma.es
C: Content-Type: text/plain; charset=ISO-8859-1;
C: Content-Transfer-Encoding: 8bit
C: Hola,
C: ¿Quedamos a las 19:00 en mi despacho?
C: Gabriel
C: .
S: 250 OK
C: QUIT
S: 221 sol10.lcc.uma.es Service closing transmission channel

```

The diagram illustrates the structure of an SMTP message. It shows a series of lines of text representing the message exchange. On the right side, red brackets group specific parts of the message into four categories: 'Sobre' (envelope), 'Cabecera' (headers), 'Cuerpo' (body), and 'Mensaje' (message). The 'Sobre' bracket groups the envelope-related commands like HELO, MAIL FROM, RCPT TO, and QUIT. The 'Cabecera' bracket groups the header fields (Subject, To, Content-Type, Content-Transfer-Encoding). The 'Cuerpo' bracket groups the message body text (Hola, ¿Quedamos a las 19:00 en mi despacho? Gabriel.). The 'Mensaje' bracket groups the entire message content between the envelope and the body.

SMTP se extendió con algunos comandos

EHLO: El servidor responde informando de las extensiones soportados

STARTTLS: Cambia a formato encriptado usando TLS

SIZE: Indica el tamaño máximo aceptado (servidor) o el tamaño estimado del correo (cliente)

AUTH: Autenticación del usuario

- AUTH PLAIN: Tras enviar este comando el usuario debe enviar su nombre de usuario y clave juntos codificado en Base64

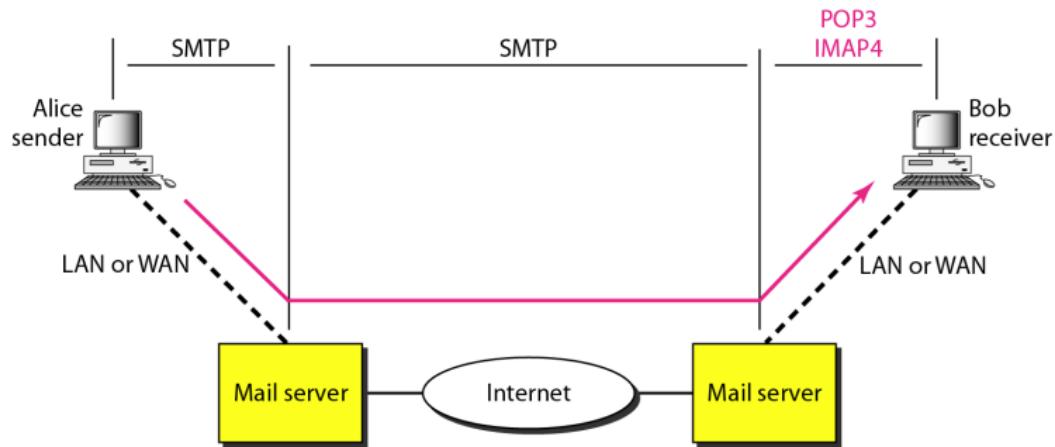
- AUTH LOGIN: Tras enviar este comando el servidor pedirá el nombre de usuario y posteriormente la contraseña (todo lo enviado, cliente y servidor, en Base64)

- AUTH CRAM-MD5: El servidor envía un reto para que el cliente envíe su contraseña cifrada.

Agente de Acceso a mensajes: POP e IMAP

Se encarga de extraer los mensajes del buzón servidor

Los mensajes van del servidor al cliente



POP3 Protocolo de la oficina de correos:

Sencillo pero con funcionalidad limitada, el cliente se instala en la máquina usuario, permite la descarga de correos en el buzón , protocolo TCP/110 y tiene 2 modos de funcionamiento, borrado y mantenimiento.

IMAP protocolo de acceso a correo de internet

Protocolo TCP/143 siendo mas complejo y potente que POP

Permite gestionar el buzón del servidor de correo con acciones como consultar mensajes antes de su descarga, buscar antes de descargar,descarga parcial y crear carpetas en el buzon y crear,borrar o renombrar buzones en el servidor de correo.

Correo Basado en Web (WEBMAIL)

Ejemplos como gmail o hotmail, permite el acceso desde el navegador, se combina el protocolo SMTP con HTTP

---

## Mensajes POP-3 (implementación reducida)

USER<SP>name<CRLF>  
PASS<SP>string<CRLF>  
STAT<CRLF>  
LIST [<SP>msg] <CRLF>  
RETR<SP>msg<CRLF>  
DELE<SP>msg<CRLF>  
QUIT<CRLF>

## Respuestas (ejemplos)

+OK<CRLF>  
-ERR<CRLF>

---

```
S: <wait for connection on TCP port 110>
C: <open connection>
S: +OK POP3 server ready <1896.697170952@dbc.mtvview.ca.us>
C: USER rdo
S: +OK rdo
C: PASS rdo123
S: +OK
C: STAT
S: +OK 2 320
C: LIST
S: +OK 2 messages (320 octets)
S: 1 120
S: 2 200
S:
C: RETR 1
S: +OK 120 octets
S: <the POP3 server sends message 1>
S:
C: DELE 1
S: +OK message 1 deleted
C: RETR 2
S: +OK 200 octets
S: <the POP3 server sends message 2>
S:
C: DELE 2
S: +OK message 2 deleted
C: QUIT
S: +OK dewey POP3 server signing off (maildrop empty)
C: <close connection>
```

La World Wide Web o Web es un repositorio de información diseminada por todo el mundo y enlazada entre si, es un servicio distribuido cliente/servidor donde el cliente es un navegador que accede a un recurso accediendo a un servidor web

El servicio de acceso está distribuido en numerosos sitios y cada sitio almacena uno o más documentos (**páginas web**). Los navegadores permiten recuperar y visualizar dichas páginas.

Una página web puede contener un enlace a otras páginas o recursos (del mismo u otros sitios) esto se conoce como **hipertexto**.

#### Transacciones HTTP:

HTTP se basa en operaciones sencillas de solicitud/respuesta

-Un cliente establece conexión con el servidor y envía un mensaje con datos de la solicitud.

-El servidor responde con un mensaje similar pero que contiene el estado de la operación y su posible resultado.

La World Wide Web fue inventada en los 90 por Tim Berners-Lee y es el concepto de hipertexto aplicado a internet.

Cada documento llamado “Página web” se encuentra en un servidor y es visto usando un programa llamado buscador o browser

#### Hipertexto:

Son documentos de texto entrelazados, el enlace normalmente es una palabra o frase que apunta a otro documento y al hacer clic sobre este el actual es reemplazado por el documento indicado en el enlace.

#### Página web:

Una página web está escrita en HTML y tiene un nombre o identificador único en todo internet, por una URL. Todas las operaciones pueden adjuntar un objeto o recurso Web, cada objeto Web (documento HTML, videos o aplicaciones) es conocido por su URL

Una página web se compone de objetos, un objeto puede ser un fichero HTML una imagen JPEG una applet de java, un fichero de audio etc...

Sitio Web:

Un sitio web es un conjunto de páginas webs que normalmente se encuentran en el mismo Host

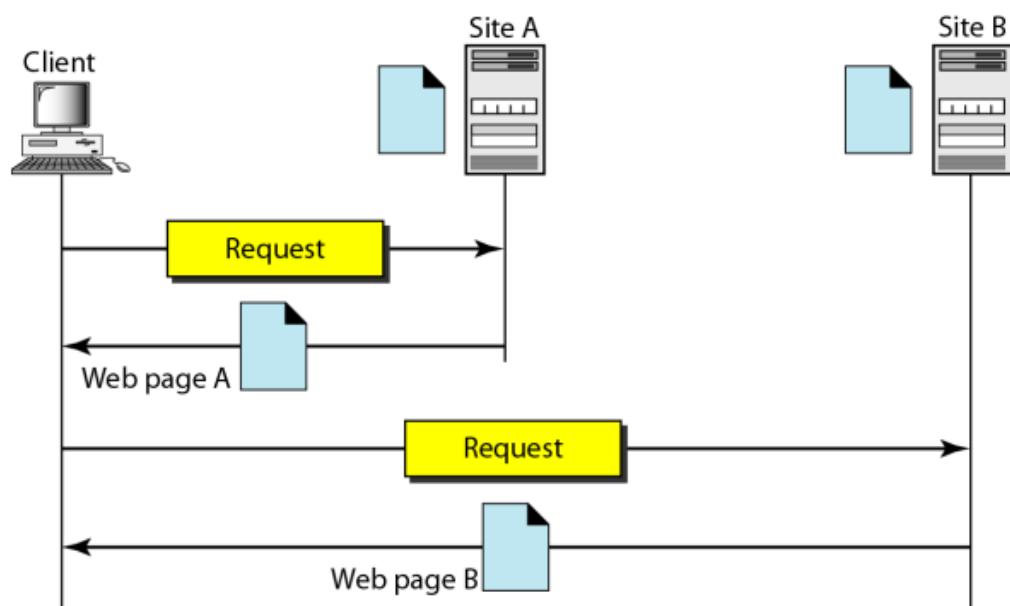
### Arquitectura WWW

El cliente quiere visualizar una información almacenada en un sitio A.

Este envía una petición a través de su navegador, la petición incluye la dirección del sitio y el recurso → URL

El servidor encuentra el documento y lo envía al cliente.

La página web A contiene una referencia a la página web B , la referencia contiene la URL de B y el cliente envía otra petición al nuevo sitio y recupera la página

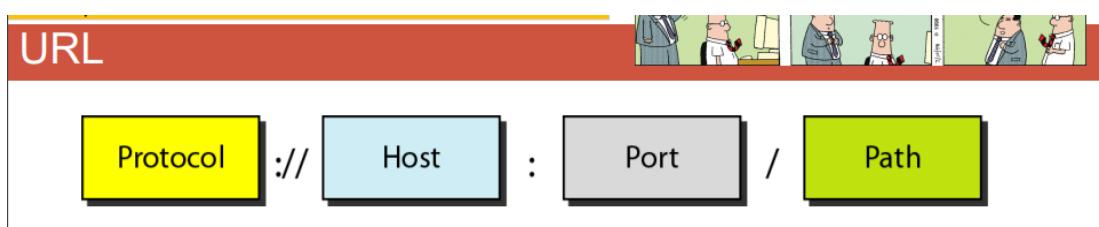


El localizador de recursos uniforme o URL es un estándar que identifica y especifica cualquier tipo de información en internet.

HTTP lo utiliza para el acceso de los documentos distribuidos por la web.

EL URL define 4 cosas:

- El protocolo: Protocolo o aplicación cliente/servidor utilizado para recuperar el recurso
- La estación Host: Dirección IP o nombre de dominio de la máquina donde se está el recurso.
- El puerto: información opcional, normalmente se toma el puerto por defecto 80/443 del protocolo.
- El camino (path): camino completo para localizar el recurso en la dirección indicada (directorios...)

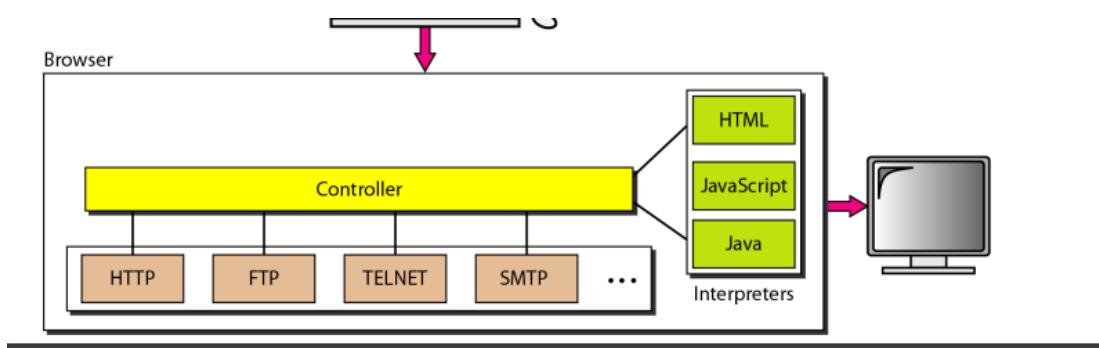


### Cliente Web - Navegador

Hay diversos fabricantes que proporcionan navegadores para interpretar y visualizar documentos web.

Cada navegador consta de un controlador, un protocolo cliente y un intérprete.

El controlador recibe la entrada de teclado y utiliza los protocolos (o programas) cliente para acceder al documento. Cuando el documento ha sido accedido utiliza uno de los intérpretes para visualizar el contenido.



## Documentos Web

Tres categorías:

Estáticos: Donde el contenido se determina en el momento de su creación , son HTML o otros recursos como imágenes o videos.

Dinámicos: El documento se crea en el servidor cuando llega una petición, son CGI, Servlets , lenguajes scripts como PHP o JS

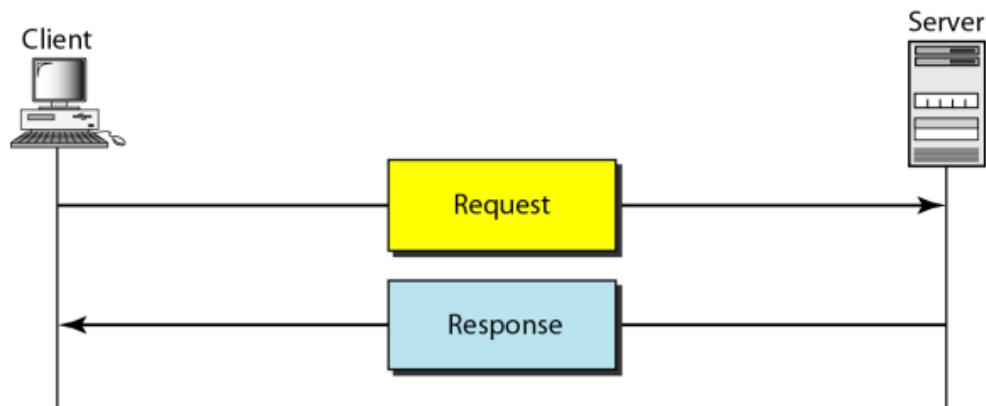
Activos: Programas que se ejecutan en el lado del cliente, el servidor envía un documento activo que se ejecuta en el lado del cliente, JavaScript, Silverlight Flash...

## PROTOCOLO DE TRANSFERENCIA DE HIPERTEXTO HTTP

Permite acceder a los recursos de la web

Utiliza TCP con el puerto 80 o 443 para HTTPS

Es un protocolo sin estado, cada par <petición/respuesta> es independiente al resto.



Un mensaje de petición consta de:

**Una línea de petición** como puede ser URL,versión HTTP

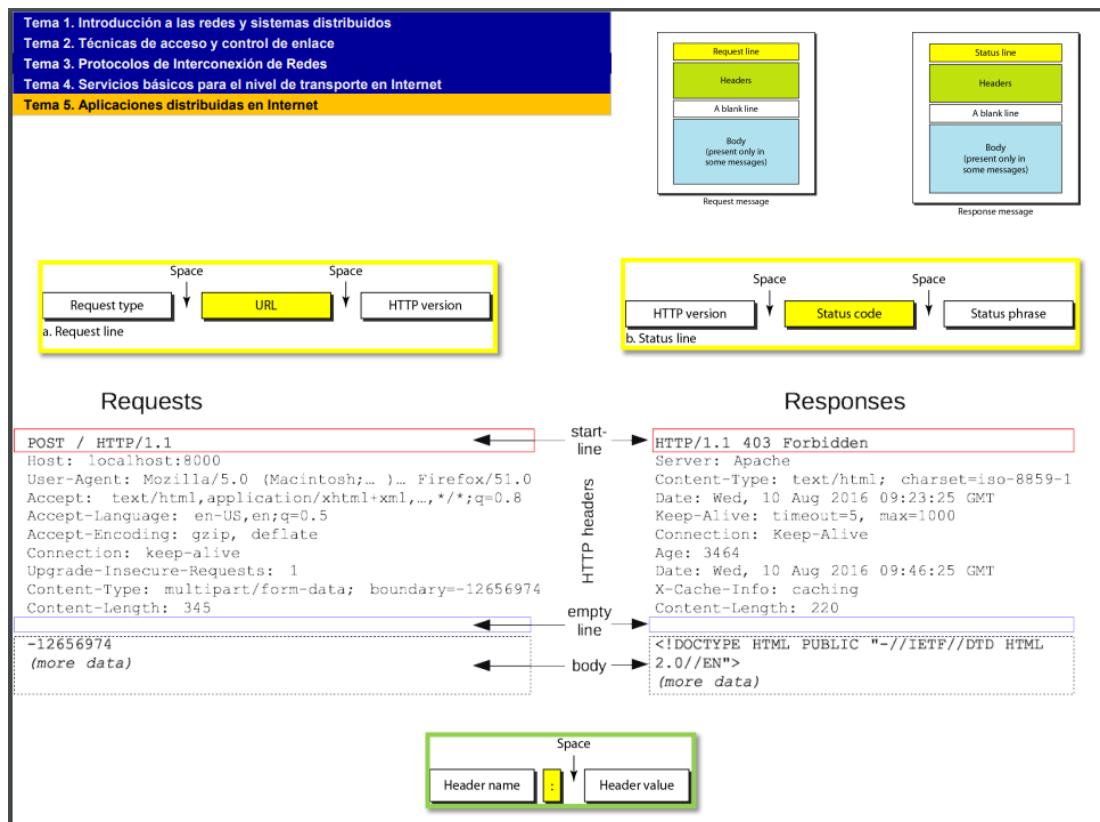
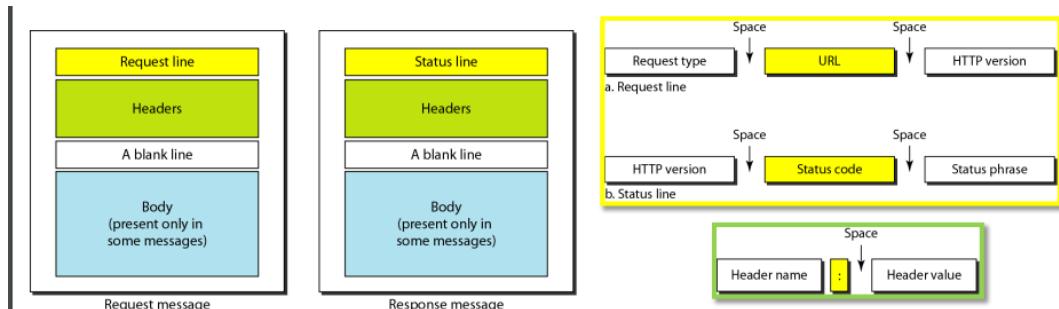
**Una cabecera** formado por un conjunto de líneas en las que se especifica información adicional

**Un cuerpo** (opcional)

Un mensaje de respuesta consta de:

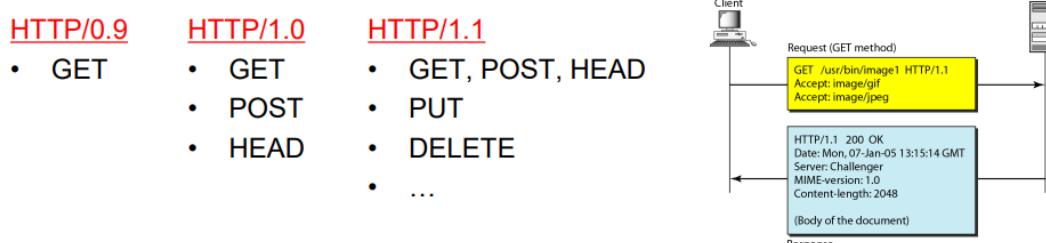
**Una linea de estado** puede ser versión HTTP, código de estado (3 dígitos) frase de estado

**Una cabecera y un cuerpo (opcional)**



## Petición: Métodos HTTP

Método	Acción
<b>GET</b>	Solicitud de un documento a un servidor
<b>HEAD</b>	Solicitud de información sobre un documento (no el documento en sí)
<b>POST</b>	Envío de información al servidor para que sean procesados por el recurso.
<b>PUT</b>	Envío de un documento del servidor al cliente
<b>TRACE</b>	Eco de petición entrante
<b>CONNECT</b>	Se utiliza para saber si se tiene acceso a un host
<b>OPTION</b>	Solicitud de algunas opciones disponibles
<b>DELETE</b>	Elimina un recurso del servidor



## Petición: Cabeceras

Cabecera	Descripción
Accept	Contenido aceptado por el navegador (ej., texto/html). <b>MIME</b>
Accept-Charset	Juego de caracteres que el navegador espera
Accept-Encoding	Codificación de datos que el navegador acepta
Accept-Language	Idioma que el navegador espera
Authorization	Identificación del navegador en el servidor
Content-Encoding	Tipo de codificación para el cuerpo de la solicitud
Content-Language	Tipo de idioma en el cuerpo de la solicitud
Content-Length	Extensión del cuerpo de la solicitud
Content-Type	Tipo de contenido del cuerpo de la solicitud (por ejemplo, texto/html). <b>MIME</b>
Date	Fecha en que comienza la transferencia de datos
Forwarded	Utilizado por equipos intermediarios entre el navegador y el servidor
Host	Nodo destino
User-Agent	Cadena con información sobre el cliente, por ejemplo, el nombre y la versión del navegador y el sistema operativo
Upgrade	Cambiar de protocolo a usar (HTTP/2)

## Respuestas: Cabeceras

Cabecera	Descripción
Cache-control	Qué objetos cachear, durante cuanto tiempo, etc..
Content-Encoding	Tipo de codificación para el cuerpo de la respuesta
Content-Language	Tipo de idioma en el cuerpo de la respuesta
Content-Length	Extensión del cuerpo de la respuesta
Content-Type	Tipo de contenido del cuerpo de la respuesta (por ejemplo, texto/html). <b>MIME</b>
Connection	Controla si la conexión tcp permanece o no abierta finalizada la transacción (conexión persistente) close vs keep-alive
Date	Fecha en que comienza la transferencia de datos
Expires	Fecha límite de uso de los datos
Forwarded	Utilizado por equipos intermediarios entre el navegador y el servidor
Location	Redirección a una nueva dirección URL asociada con el documento
Server	Características del servidor que envió la respuesta
Strict-Transport-Security	Obliga al uso de conexión segura (HSTS). Convierte los http en https
X-...	Cabeceras no estándar definidas por el servidor

## Petición: Envío de datos.

**GET:** los datos se codifican en la URL:

```
GET /suma.html?op1=1&op2=2
HTTP/1.1
...
[Línea en blanco]
```

**POST:** Los datos se envían en el cuerpo de la petición:

```
POST /suma.html HTTP/1.1
...
[Línea en blanco]
op1=1&op2=2
[Línea en blanco]
```

Además de la forma de envío Get es cacheable, se puede añadir a favoritos y se mantiene en el historial del navegador, Post tiene longitud ilimitada y permite datos binarios.

### **Respuestas: Tipos**

Mensajes de respuesta (estados y frases) + contenido solicitado:

- 1xx Confirmación preliminar
  - 2xx Confirmación
  - 3xx Se necesitan más acciones por parte del cliente
  - 4xx Error en la petición
  - 5xx Error en el servidor
- 
- o Ejemplos
    - 101 Switching
    - 200 OK
    - 301 Moved Permanently
    - 400 Bad Request
    - 404 Not Found
    - 505 HTTP Version Not Supported

### **HTTP: Documentos web**

Tres tipos de documentos:

Estáticos : recurso fijo del servidor

Dinámicos: El documento se crea en el servidor cuando llega una petición

Activo: el programa se ejecuta en el lado del cliente,el servidor envía un documento activo que se ejecuta en el lado del cliente.

Documentos web (HTML):

Contienen múltiples objetos como código en JS, CSS ,imágenes, videos.

### **HTTP: Control de sesiones: Cookies**

Las cookies son piezas de información que permite conservar info entre peticiones (compras, personalizaciones, análisis)

El servidor establece la opción Set-Cookie (se puede indicar una duración)

En algunos casos de código JavaScript las pueden tratar (si la opción HTTPONLY esta desactivada)

El cliente en cada petición en las opciones enviará su contenido en la opción Cookie



Propiedades:

4KB máx

50 por dominio

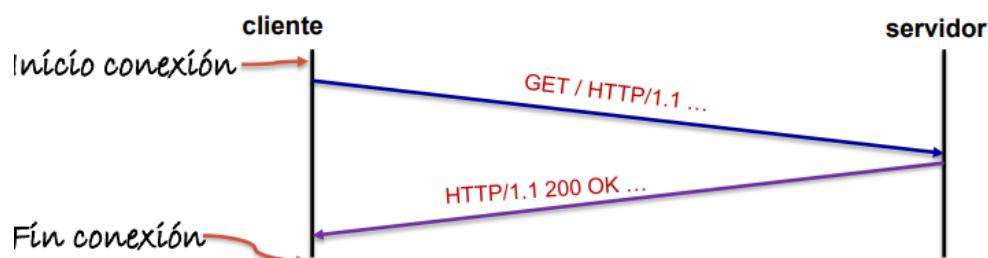
3000 en total

Problemas:

Cookies de terceros, privacidad, acceso indebido o falsificación

### **HTTP: Uso ineficiente de conexiones TCP**

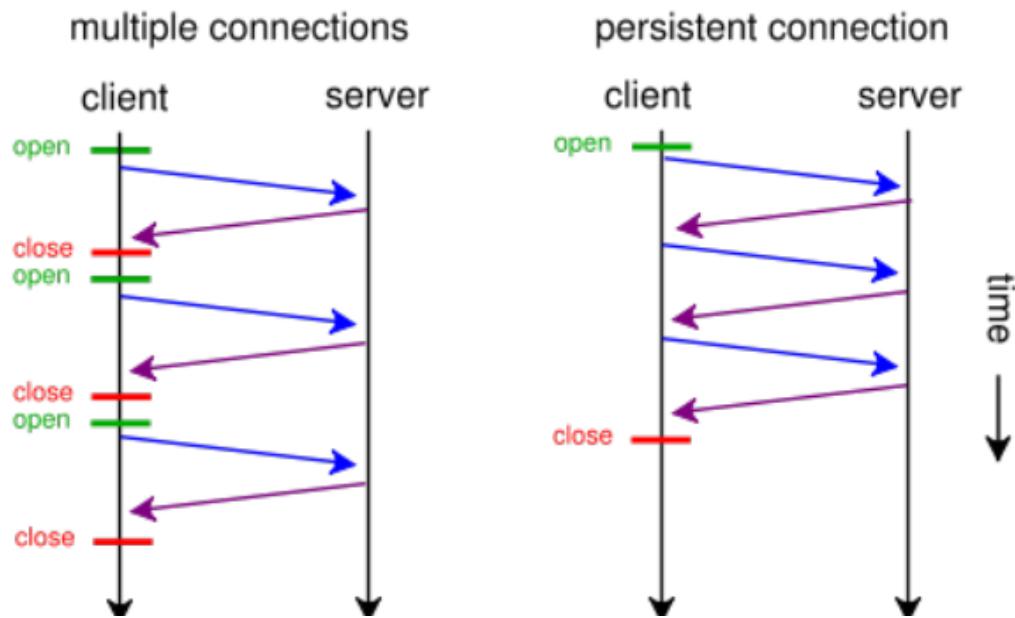
una petición /respuesta por conexión TCP



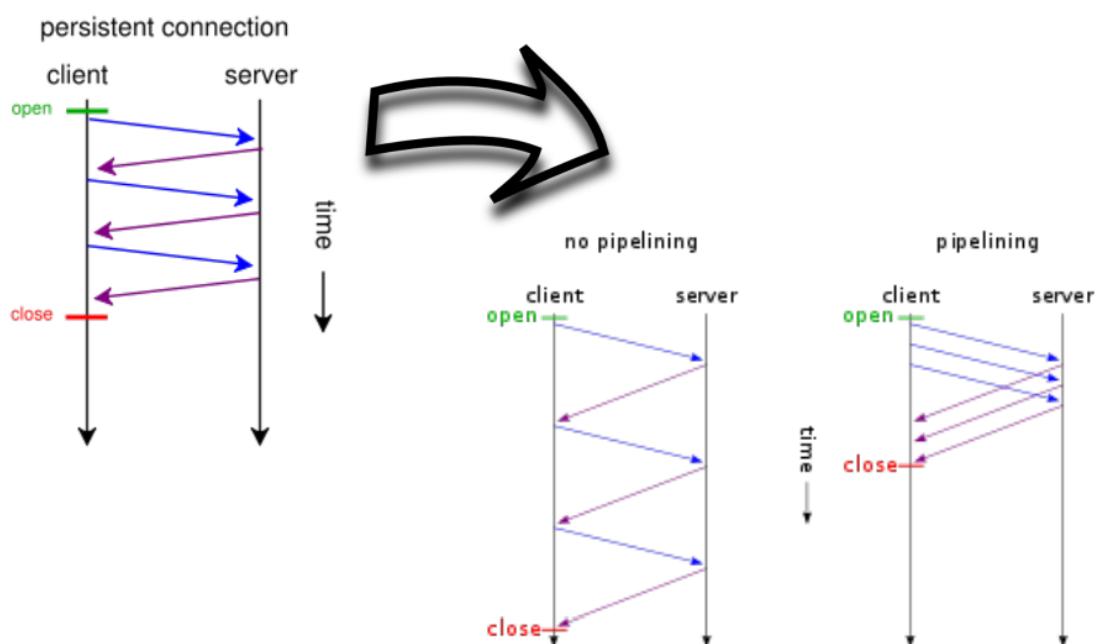
Problemas: Inicio/fin de conexión (6/7 mensajes)

Control de la congestión (Slow Start)

Para mejorar la eficiencia se usa la técnica de persistencia, es decir usar la misma conexión para varios envíos de modo que solo se inicia/finaliza la conexión una vez



- En HTTP/1.0 con la opción Connection: Keep-Alive
  - HTTP/1.1 por defecto activa
  - Se cierra con Connection: Close



HTTP Problema de la mejora de eficiencia

Reducir los rtts: Solapando peticiones,

Pipelining: un cliente puede enviar múltiples peticiones sin esperar a cada respuesta individual, un servidor debe responder en el mismo orden que fueron recibidas.

Múltiples conexiones: Clientes que usan conexiones persistentes deberán limitar le número de conexiones simultaneas que mantienen con un servidor, un mismo usuario no deberá tener mas de dos conexiones con ningún servidor o proxy.

Estas guías tienen la intención de mejorar el tiempo de respuesta de HTTP y evitar la congestión.

## **HTTP/1.1: Problemas**

- HTTP/1.1
  - Persistencia:
    - La misma conexión para multiples recursos
    - Problema: dependencia entre recursos (head of the line blocking), esperas (no pipelining)
  - Optimizaciones:
    - Descargas en paralelo:
    - Múltiple conexiones  
(Ej, 6 por dominio - Chrome)
    - “Trucos” de desarrollo:
      - Concatenar ficheros, incrustar ficheros (Ej, meter imagen en un fichero CSS), etc.
      - Uso de varios dominios por servidor web (media: 18)
    - Problema: complejidad, peticiones DNS, problemas con el uso de caché, modularidad, ...
  - ¿Es necesario una nueva versión de HTTP?
    - Evitar trucos que incrementen la eficiencia
    - Solucionar otros problemas que tiene
      - Objetivos de HTTP/2
      - Aumentar la eficiencia
      - No requerir múltiples conexiones y evitar otros problemas
      - Compatibilidad HTTP/1.1 (no romper la web ya existente)
    - Aprobado el 18 de febrero de 2015:

- HTTP/2 (RFC 7540) – 76 páginas
- HPACK: Compresión de la cabecera (RFC 7541) – 55 págs.
- Dos implementaciones:
- HTTP/2 sobre TLS (h2)
- HTTP/2 sobre TCP plano (h2c)

## **HTTP/2**

La idea es mantener la semántica de la versión 1.1, mismos métodos, cabeceras, casos de uso

Mantiene el modelo cliente/servidor

Novedades: Envío de tramas en binario, compresión de las cabeceras, una sola conexión:

Múltiples flujos , se pueden limitar y se pueden cerrar sin cerrar conexión (RST\_STREAM)

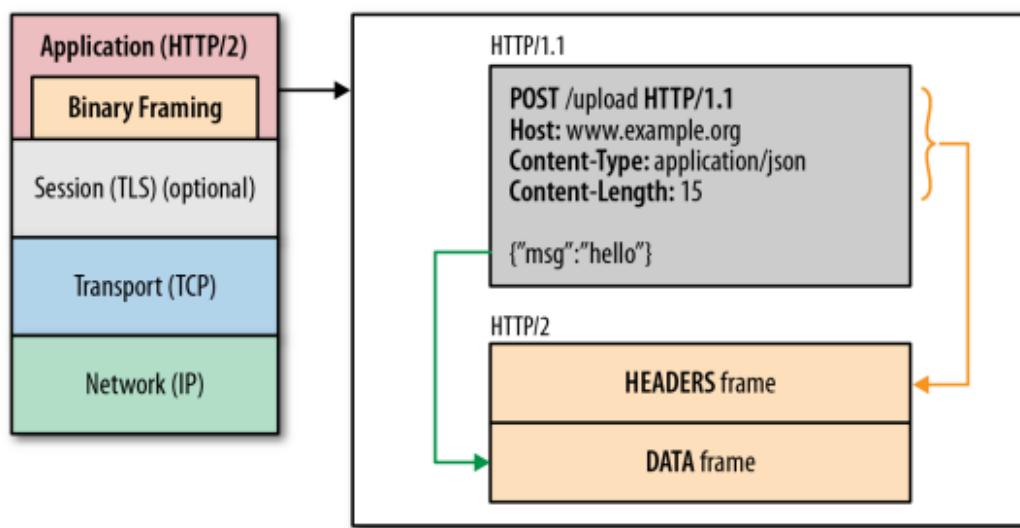
Prioridades/dependencias de flujos y cambios dinámicos → recursos "fuera de orden"

Server push: Envío de información sin ser solicitada por parte del servidor

## **HTTP/2: Capa de Tramado Binario**

La semántica de HTTP es la misma, pero la manera en la que los paquetes HTTP se transmiten y transportan cambian gracias a la capa de tramado binario (BINARY FRAMING)

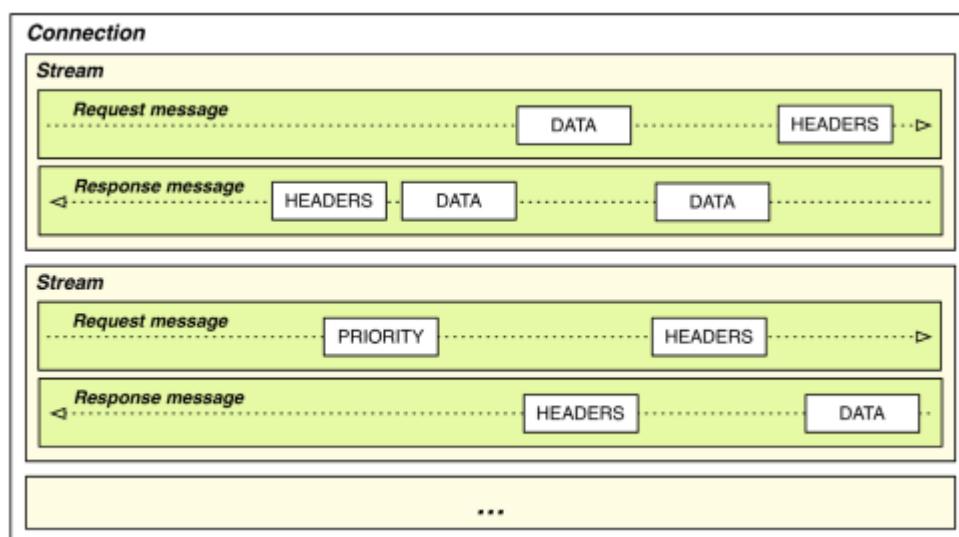
Los mensajes HTTP son divididos en tramas independientes y reensamblados en el destino



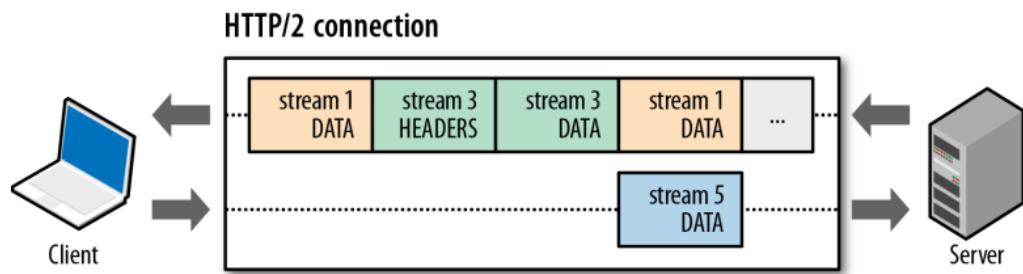
## HTTP/2: Flujos, mensajes y tramas

En una misma conexión podemos tener varios flujos en paralelo, cada flujo está identificado de manera única y puede transportar mensajes bidireccionalmente y tiene dependencias a otros flujos y una prioridad

Cada mensaje es una respuesta o solicitud HTTP y se divide en una o varias tramas, la trama es la unidad mínima de comunicación y se puede entrelazar con otras tramas de otros flujos de comunicación. Son asignadas en destino a su flujo gracias al uso de identificadores

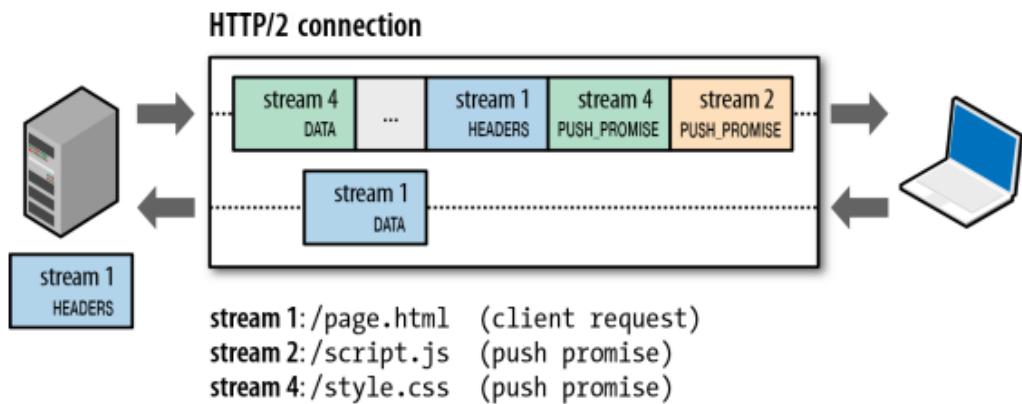


El uso de una misma conexión con múltiples flujos permite el multiplexado de solicitudes y respuestas. Las tramas correspondientes a distintos flujos se intercalan en la misma conexión. Elimina el HOL y la necesidad de múltiples conexiones

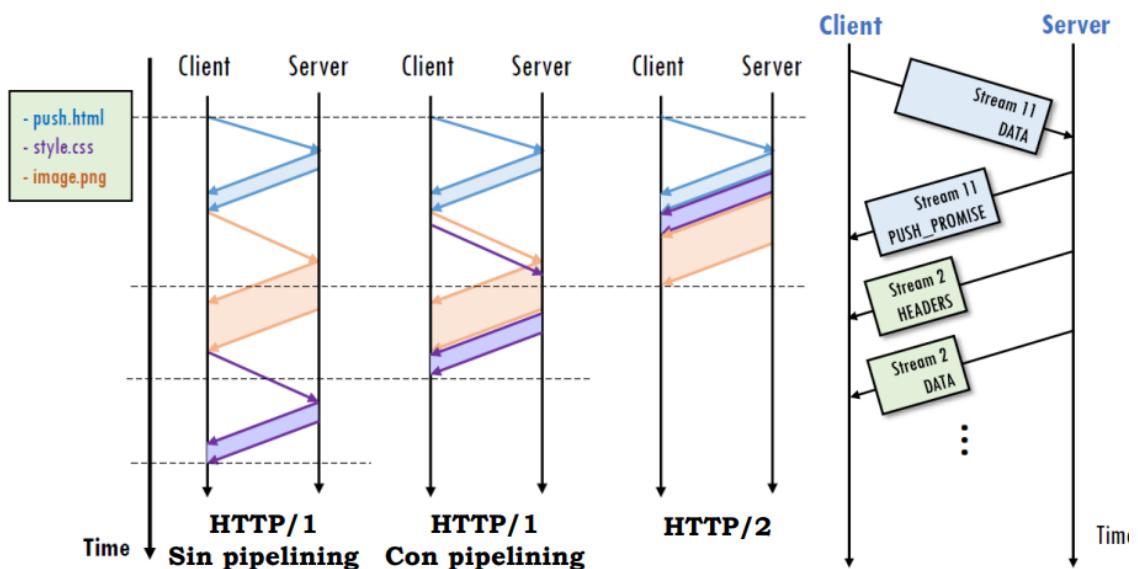


## HTTP/2: Server Push

- Las versiones anteriores de HTTP solo permiten que el servidor mande información cuando se le solicita
  - En HTTP/2 se permite que el servidor mande múltiples respuestas a una solicitud.
    - El servidor puede hacer PUSH de recursos que el cliente no le ha solicitado de manera explícita, pero que sabe que va a necesitar



Los flujos PUSH son señalizados con las tramas PUSH\_PROMISE



Cada solicitud HTTP lleva una serie de cabeceras que describen el recurso transportado y sus propiedades

Las cabeceras añaden desde 500Bytes hasta varios Kbytes en cada transacción HTTP

Para reducir esta sobrecarga HTTP/2 comprime las cabeceras usando el formato HPACK

Los campos de la cabecera son codificados usando códigos Huffman

Clientes y servidores mantienen y actualizan tablas de traducción para entender y codificar campos de cabeceras

Uso de tablas de cabeceras:

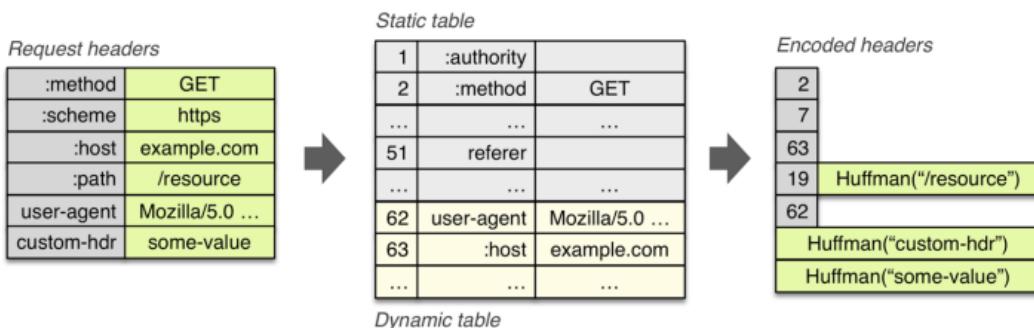
Se crea con habituales (estática) o con usadas previamente (dinámicas)

Se envían los índices de esas tablas

Los valores /cabeceras nuevas se codifican con Huffman

Caracteres más usados se envían usando menos bits

- 'e' -> 00101
- '\> 11111111111111110000



- 9 Bytes + parte variable según el tipo:

Bit	+0..7	+8..15	+16..23	+24..31
0		Length	Type	Flags
32	R		Stream Identifier	
...			Frame Payload	

Length: Longitud de la trama (sin incluir los 9 bytes iniciales)

Type: Tipo de trama

Flags: definidos atendiendo al tipo de trama

R: Reservado (0)

Stream Identifier: Identificador de flujo (petición/respuesta):

0: reservado para control de la conexión en general

1: para la conexión HTTP/1.1 inicial (si la hay)

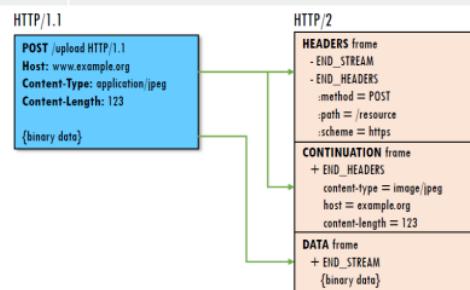
Iniciados por el cliente: impares

Iniciados por el servidor: pares

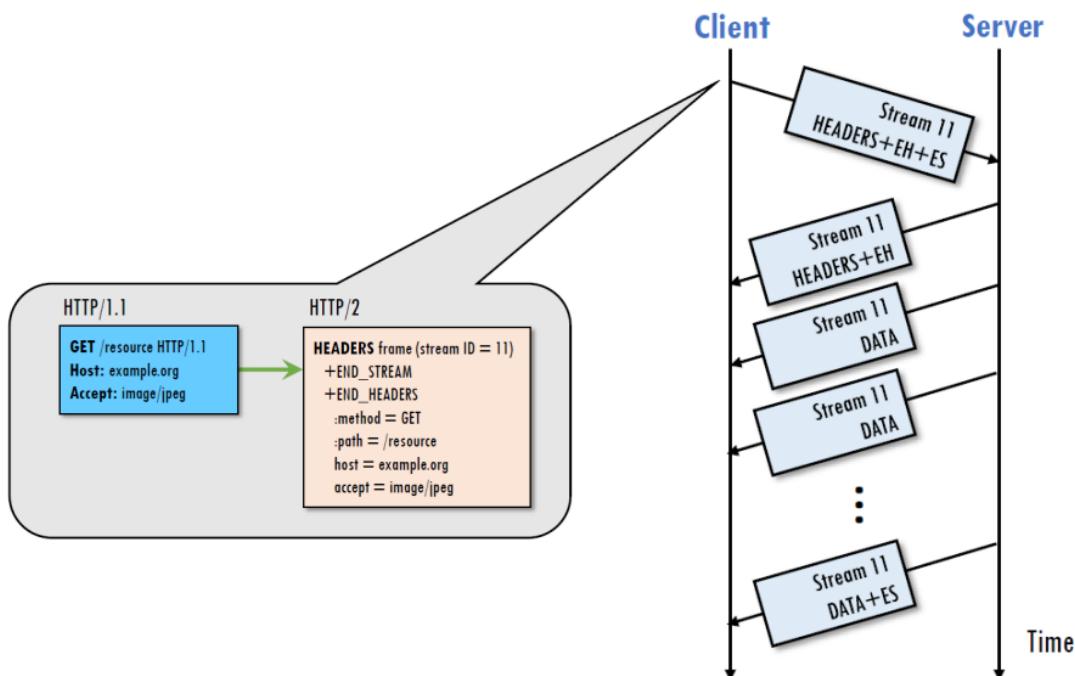
Frame Payload: Datos (depende del tipo de trama)

## HTTP/2: Tipos de trama

Frame type	Description
<b>DATA</b>	<b>Cuerpo HTTP</b>
<b>HEADERS</b>	<b>Cabeceras</b>
PRIORITY	Prioridad de un flujo
RST_STREAM	Finalización de un flujo
<b>SETTINGS</b>	<b>Parámetros de configuración de la conexión (solo flujo 0)</b>
PUSH_PROMISE	Envío de datos (sin haber sido pedidos)
PING	Para medir el rtt y comprobar que el otro extremo está activo
GOAWAY	Informa al otro extremo que no cree más flujos en esta conexión
WINDOW_UPDATE	Control de flujo (flujo concreto o conexión global – flujo 0)
CONTINUATION	Continúa enviado cabeceras



## HTTP/2: Trama: Ejemplo de comunicación



Antes de que cualquier dato pueda ser enviado, se tiene que crear un flujo e indicar sus características y las cabeceras HPACK que van a utilizarse.

- La información del flujo se indica en tramas tipo HEADERS

```

▼ HyperText Transfer Protocol 2
  ▼ Stream: HEADERS, Stream ID: 1, Length 20
    Length: 20
    Type: HEADERS (1)
    ▼ Flags: 0x05
      .... ...1 = End Stream: True
      .... .1.. = End Headers: True
      .... 0... = Padded: False
      ..0. .... = Priority: False
      00.0 ..0. = Unused: 0x00
      0.... .... .... .... .... .... = Reserved: 0x00000000
      .000 0000 0000 0000 0000 0000 0001 = Stream Identifier: 1
      [Pad Length: 0]
      Header Block Fragment: 8682418aa0e41d139d09b8f01e078453032a2f2a
      [Header Length: 100]
      ▶ Header: :scheme: http
      ▶ Header: :method: GET
      ▶ Header: :authority: localhost:8080
      ▶ Header: :path: /
      ▼ Header: accept: */
        Name Length: 6
        Name: accept
        Value Length: 3
        Value: */
        Representation: Literal Header Field with Incremental Indexing – Indexed Name
        Index: 19
  
```

common frame header

HPACK encoded headers

- Los datos se envían en tramas de tipo DATA
  - Los datos de un proceso emisor pueden dividirse en varias tramas que son enviadas por el mismo flujo
  - La trama con el indicador END\_STREAM indica el final de los datos y del flujo

---

```

▼ HyperText Transfer Protocol 2
  ▼ Stream: DATA, Stream ID: 1, Length 5
    Length: 5
    Type: DATA (0)
    ▼ Flags: 0x00
      .... ...0 = End Stream: False
      .... 0... = Padded: False
      0000 .00. = Unused: 0x00
      0.... .... .... .... .... = Reserved: 0x00000000
      .000 0000 0000 0000 0000 0000 0001 = Stream Identifier: 1
      [Pad Length: 0]
      Data: 48656c6c6f
  
```

0000	02 00 00 00 45 00 00 42 89 06 40 00 40 06 00 00	....E..B ..@.@...
0010	7f 00 00 01 7f 00 00 01 1f 90 d8 eb 8a 94 78 19	..... .....x.
0020	7d b6 67 50 80 18 23 dd fe 36 00 00 01 01 08 0a	}.gP..#. .6.....
0030	6a 78 1f ec 6a 78 1f ec 00 00 05 00 00 00 00 00	jx..jx... .....
0040	01 48 65 6c 6c 6f	.Hello