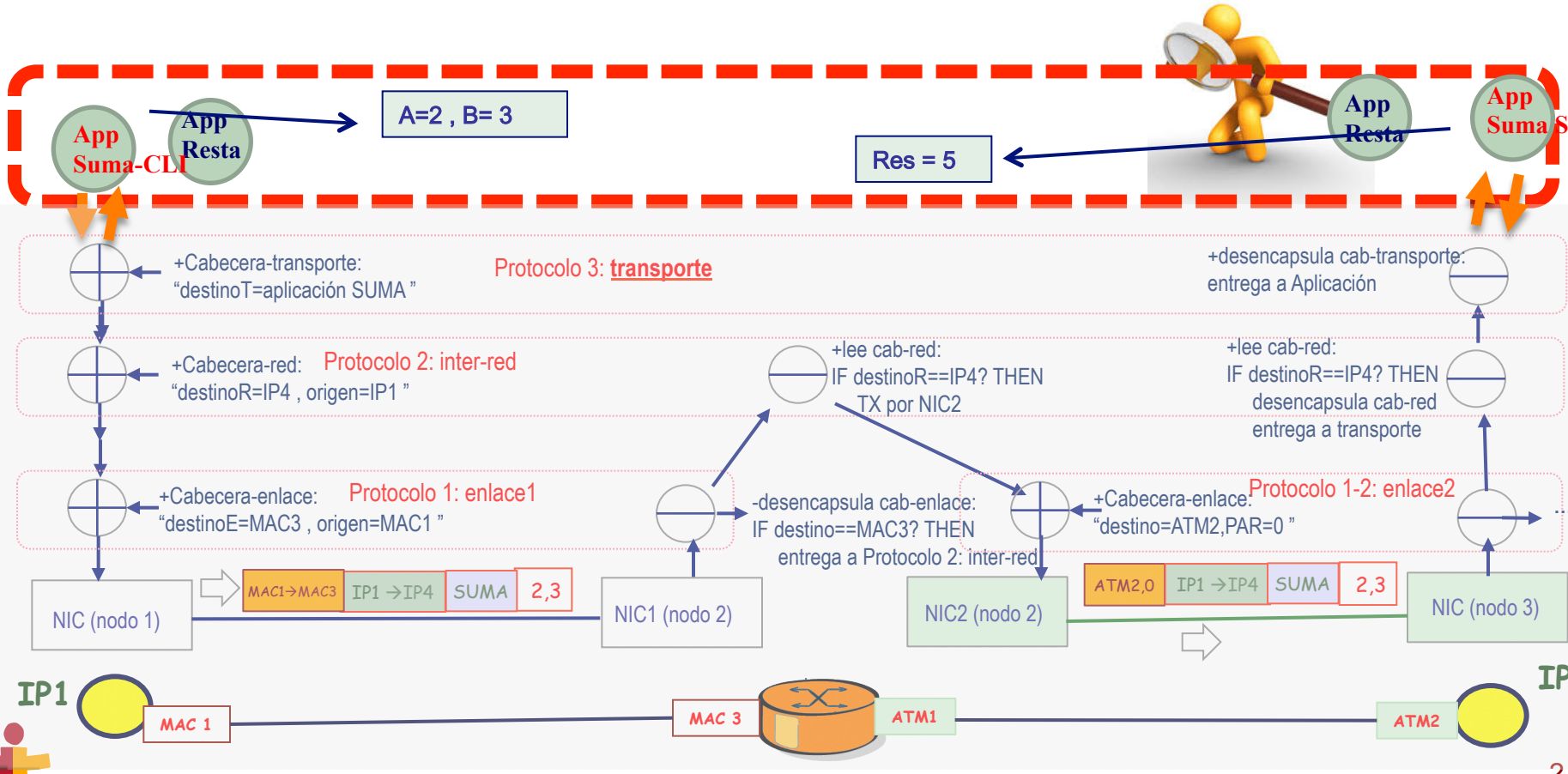
Tema 02
curso 2018/19

Introducción a la Capa de Aplicación

Antonio J. Estepa Alonso

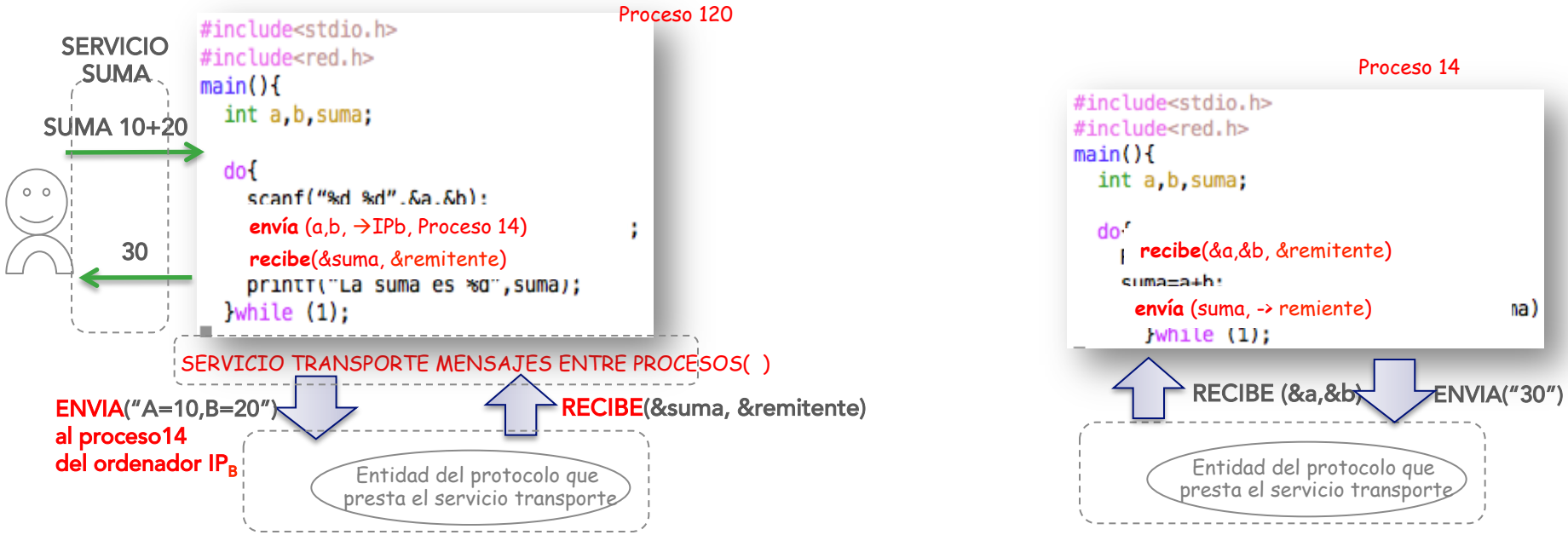
Departamento de
Ingeniería Telemática

Índice del Tema 02



Aplicaciones Distribuidas

- Ofrecen un servicio solicitado por un usuario (humano u otra app)
- Para ofrecer su servicio varios procesos deben cooperar
  - Cada proceso necesita usar primitivas del servicio de transporte de mensajes entre procesos para enviar o recibir mensajes



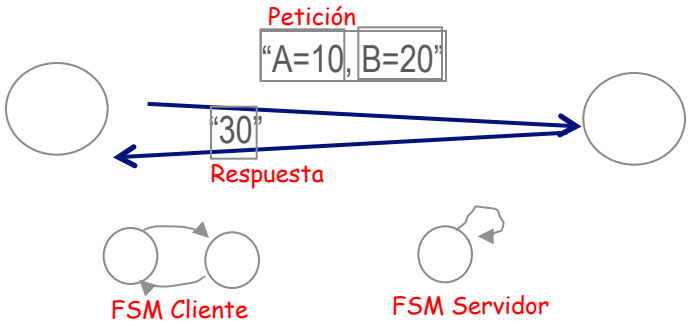
Host A
IP_A

Host B
IP_B

Protocolo en una Aplicación Distribuida

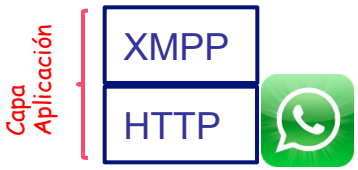
Define

- El tipo de mensajes intercambiados entre los procesos remotos
  - p.e. petición o respuesta
- La sintaxis de cada mensaje (campos y su delimitación)
- La semática de cada campo
- Una serie de reglas que indican cuándo y cómo los procesos envían y responden a los mensajes



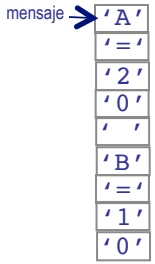
- Pueden ser
  - Propietarios
  - Públicos

- Pueden ser complejos
  - Muchas A-PDUs con muchos campos
  - Protocolos encapsulados en otros protolos



Protocolos orientados a carácter (textual) vs binarios

- Las A-PDUs (mensajes) pueden ser “legibles” por personas o no.
  - Ejemplo de Mensaje “legible” (protocolo orientado a carácter, textual)



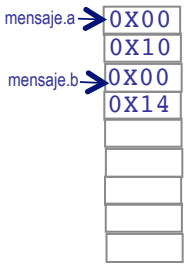

```c
#include<stdio.h>
#include<red.h>

main(){
  char mensaje[]="A=20 B=10";
  ENVIA(ORDENADOR B, PROCESO 14, mensaje);
}
```

(9 CARACTERES+\0)

“A=20 B=10”

```c
#include<stdio.h>
#include<red.h>

main(){
  char mensaje[100];
  RECIBE(ORDENADOR A,PROCESO 120, mensaje);
  /* ahora examinamos la cadena (parse)*/
  if (strncmp(mensaje,"A=",2)==0)
    ...
}
```

  - Ejemplo de Mensaje no “legible” (protocolo binario)




```c
#include<stdio.h>
#include<red.h>

typedef struct{
int a;
int b;
} APDU_req;

main(){
  APDU_req mensaje;
  mensaje.a=10;
  mensaje.b=20;
  ENVIA(ORDENADOR B, PROCESO 14, mensaje);
}
```

(4 bytes) – depende del sistema

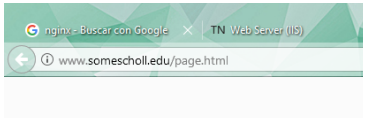
0x00100014

```c
#include<stdio.h>
#include<red.h>

typedef struct{
int a;
int b;
} APDU_req;

main(){
  APDU_req mensaje;
  int suma=0;
  RECIBE(ORDENADOR A, PROCESO 120, &mensaje);
  suma = mensaje.a + mensaje.b;
}
```

Ejemplo

- Protocolo textual, aplicación Web
  - Protocolo HTTP (RFC 2616) petición/respuesta
    - Proceso (a) cliente web (hace peticiones e interpreta respuestas)
    - Proceso (b) servidor web (responde a las peticiones)

(a)



petición

```
GET /somedir/page.html  http/1.1
Host: www.someschool.edu
Connection: close
User-agent: Chrome/4.0
Accept-language: es
```

(b)

Servidor Web (Servidor)

Navegador (Cliente) (browser, user-agent)


Google Chrome


Opera


Safari

```
HTTP/1.1 200 OK
Connection: close
Data: Thu, 03 Jul 2006 12:00:15 GMT
Server: Apache/1.3.0 (Unix)
Last-Modified: Sun, 5 May 2006 09:23:24 GMT
Content-Length: 6821
Content-Type: text/html
(Data,data,data,…)
```

1. Envía ( )
4. Recibe( )

S.O. SERVICIO TRANSPORTE MENSAJES( )

3. Envía( )
2. Recibe( )

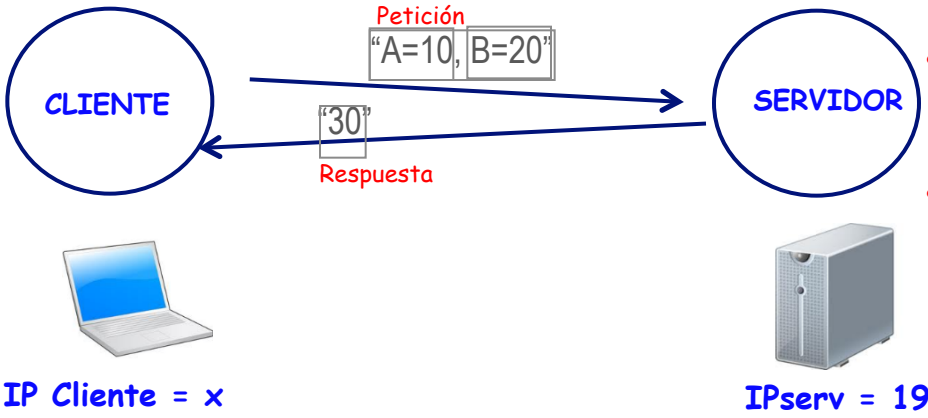
S.O. SERVICIO TRANSPORTE MENSAJES( )

https://www.tutorialspoint.com/http/http_message_examples.htm

Arquitectura de una Aplicación Distribuida

- Indica la organización de la aplicación entre varios procesos
  - Cómo los diferentes procesos que componen la aplicación distribuida se comunican entre sí.
- Terminología: (según el papel de cada proceso)
  - Proceso de tipo cliente (inicia la comunicación y solicita algo)
  - Proceso de tipo servidor (sirve las peticiones del cliente

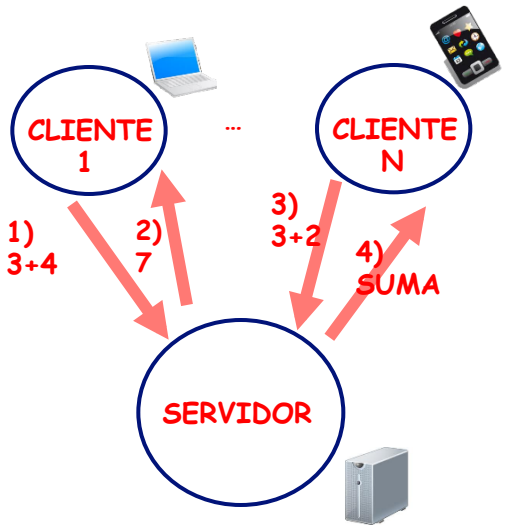
- Se comunica con el servidor para realizar peticiones
- Puede o no estar encendido
- Puede tener direcciones IP dinámicas



- Equipo siempre encendido (el proceso servidor espera a ser contactado)
- Atiende peticiones (responde)
- Dirección IP permanente y conocida por los clientes

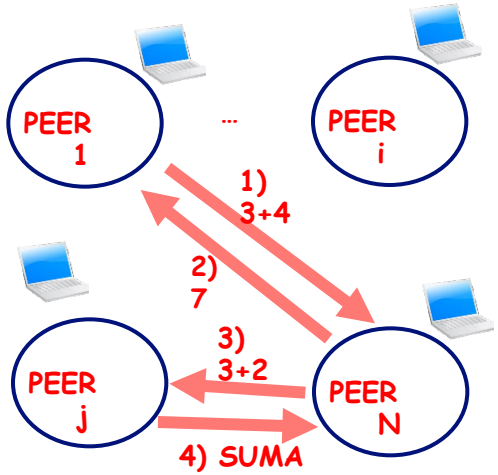
Tres posibles arquitecturas

Cliente/Servidor



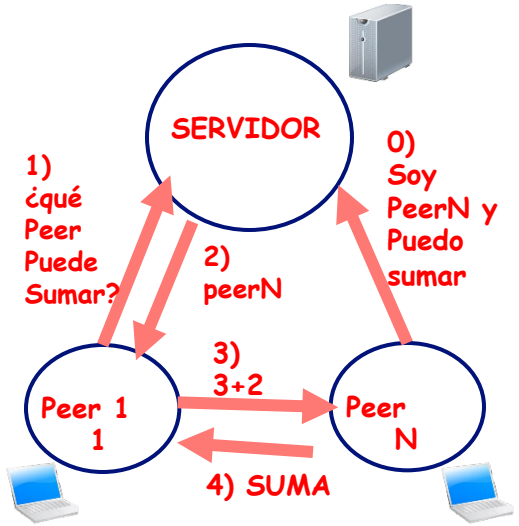
- NO HAY COMUNCACIÓN ENTRE CLIENTES
- UN SOLO SERVIDOR ATIENDE PETICIONES DE MUCHOS CLIENTES
  - ☹ PUNTO ÚNICO DE FALLO
  - ☹ ESCALABILIDAD

Peer2peer



- PEER = CLIENTE + SERVIDOR, pero con carácterísticas de "CLIENTES"
- SÓLO HAY COMUNCACIÓN ENTRE "CLIENTES"
  - ☹ ¿CON QUÍEN ME COMUNICO?
  - ☹ ¿ESTARÁ ENCENDIDO? TENDRÁ LO QUE BUSCO?

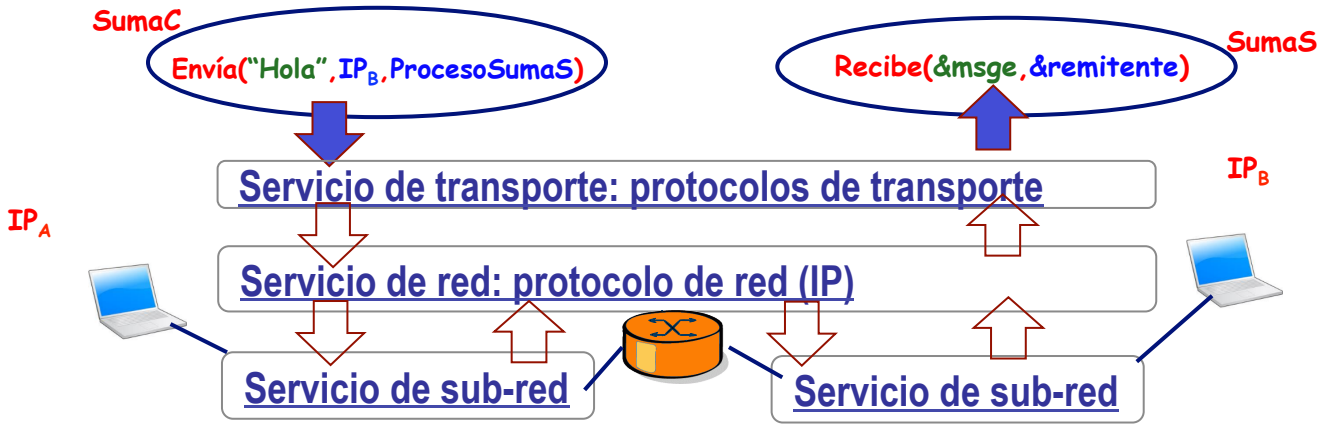
Mixta



- SERVIDORES DE REGISTRO, AUTENTICACIÓN Y LOCALIZACIÓN DE CONTENIDOS
- PEERS CONSULTAN SERVIDORES PARA SABER CON QUÉ PEER TIENEN QUE COMUNICARSE PARA OBTENER LO QUE BUSCAN

Servicio de transporte de Mensajes entre procesos remotos

- Envia/Recibe mensajes desde un proceso que se ejecuta en un host (IP) hasta otro proceso que se ejecuta en otro host



- Pero durante el viaje a través de Internet pueden ocurrir:
  - Pérdidas
  - Retrasos
  - Throughput limitado (caudal entre procesos de aplicación)
  - Seguridad

- **PREGUNTA:** ¿cómo afectarán los factores anteriores al servicio que presta la aplicación?

Respuesta

- **Depende de cada tipo de aplicación**
  - **Aunque la seguridad es siempre importante**
- **Libro Kurose.**

Aplicación	Tolera Pérdida de datos?	Throughput	Sensible al retardo?
Transf. fich.	NO	flexible	no
e-mail	NO	flexible	no
Web docs.	NO	flexible	no
tiempo real audio/video	SI	audio: 5kbps-1Mbps video:10kbps-5Mbps	si, 100’s msec
Almac. audio/video	SI	Idem (arriba)	si, pocos secs
Juegos interactiv.	SI	desde pocos kbps	si, 100’s msec
Mensajería instan.	NO	flexible	si y no

… y qué servicios de transporte ofrece Internet?

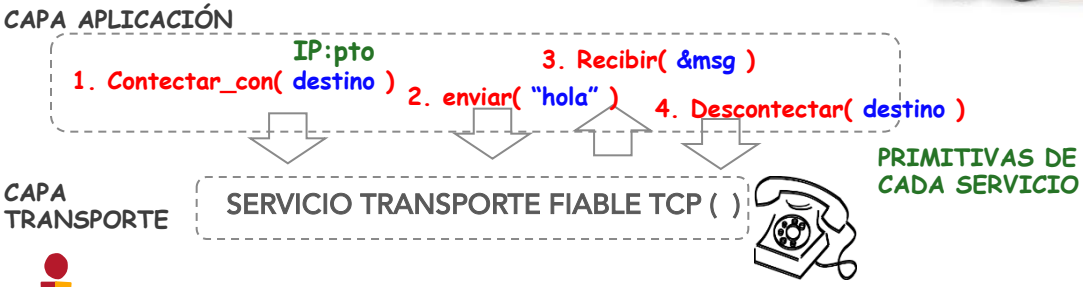
Servicio TCP: (características)

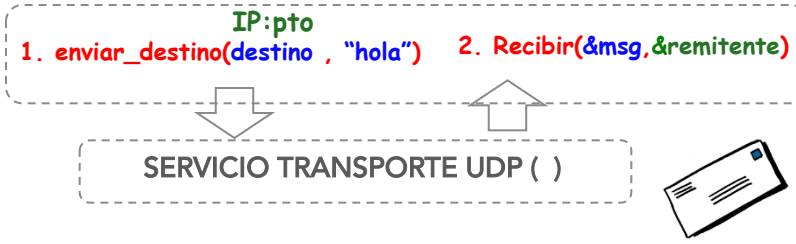
- *Orientado-a-conexión:* se requiere el establecimiento de una "asociación" entre los procesos cliente y servidor antes del intercambio de mensajes
- *Transporte Fiable* de mensajes entre los procesos que se comunican
- *Control de Flujo:* el proceso emisor no "agobiará" al proceso receptor
- *Control de Congestión:* ralentiza la emisión si hay congestión en la red
- *NO OFRECE:* retardo acotado, garantías sobre throughput mínimo, securidad

Servicio UDP:

- Tranferencia de mensajes NO FIABLE entre los procesos origen y destino
- NO OFRECE: establecimiento de conexión, control de flujo, control de congestión, retardo acotado, garantías sobre throughput, seguridad

¿Tiene UDP alguna ventaja ?





Protocolos de aplicación y transporte usados por algunas aplicaciones de Internet

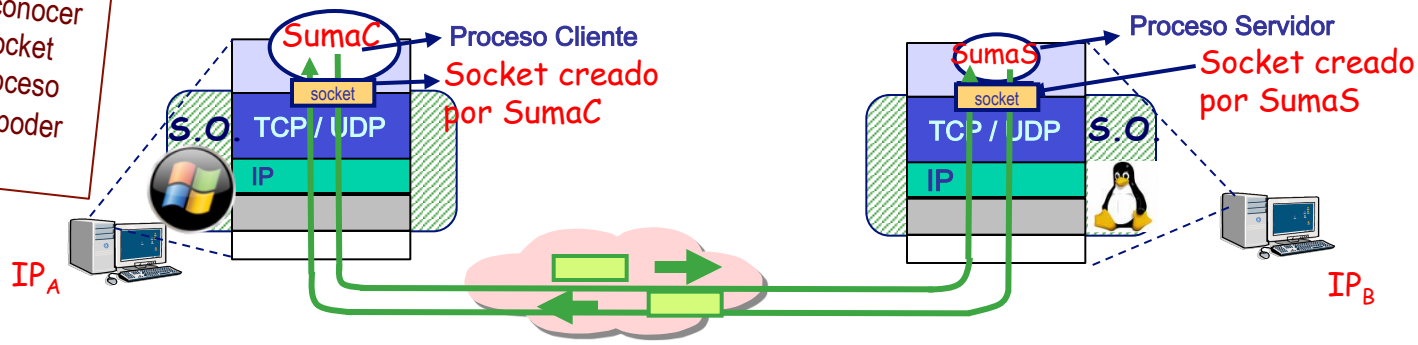
- **Cada aplicación elige el servicio de transporte que más se adecúe a sus necesidades.**

APLICACIÓN	Protoco de nivel de Aplicación	Servicio elegido para transporte
e-mail	SMTP [RFC 2821]	TCP
terminal remoto	Telnet [RFC 854]	TCP
Web	HTTP [RFC 2616]	TCP
transferencia ficheros	FTP [RFC 959]	TCP
streaming multimedia	HTTP (eg Youtube), RTP [RFC 1889]	TCP or UDP
Telefonía en Internet	SIP, RTP, proprietarios (e.g., Skype)	Normalmente UDP

¿cómo solicitan tus aplicaciones los servicios del protocolo de transporte elegido?

- **Socket**: API para usar la red (ofrecida por el S.O.)
  - Las aplicaciones crean socket (t-SAP) y atrés de ellos acceden a los servicios de transporte (usando primitvas del servicio).
  - El socket creado es identificado de forma única localmente

Un cliente debe conocer la dirección del socket que ha creado proceso servidor antes de poder enviarle mensajes



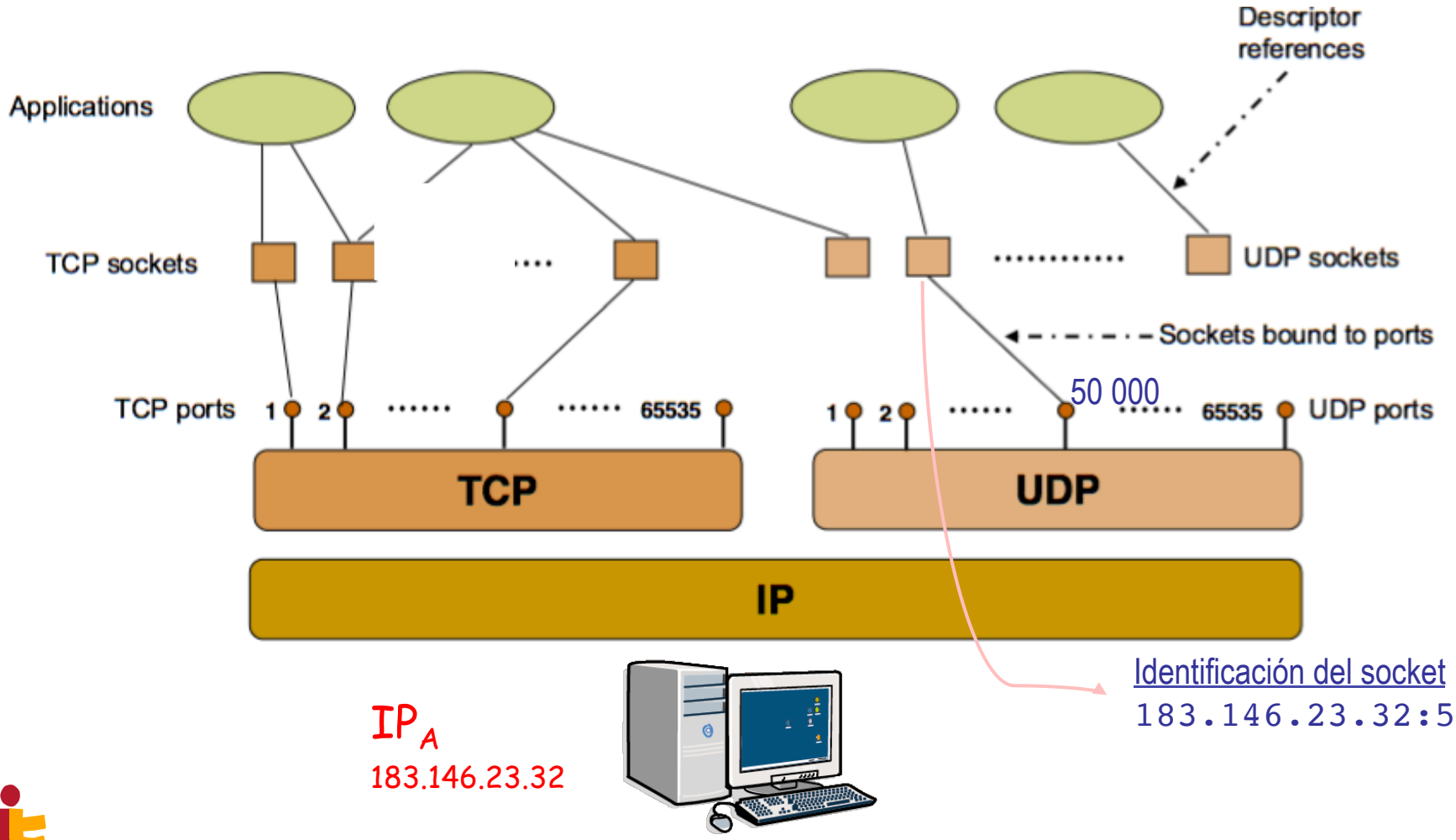
- **Primitivas de Acceso al Servicio de transporte → Llamadas usada por tu aplicación para solicitar servicios de transporte en Internet**

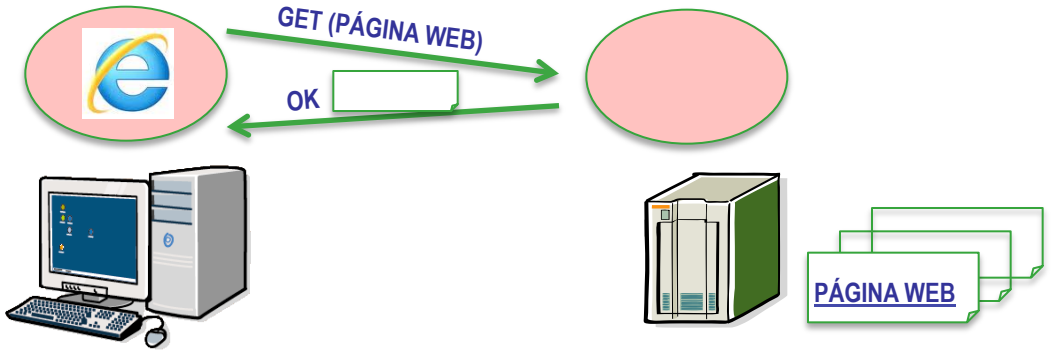
```
Socket  ( … )   /* creación de sockets */
Bind (…)  /* asociación de socket a un identificador local*/
Connect ( … )  /* solicitud de conexión*/
Send( … ) /*envío a través de una conexión*/
SendTo( …) /*envío sin conexión */
Etc. ..
```

Librerías con "primitivas de servicio" para el uso de los sockets (#include<socket.h>)

sockets

- Dentro de un host puede haber multiples aplicaciones
- Cada aplicación puede haber creado varios sockets



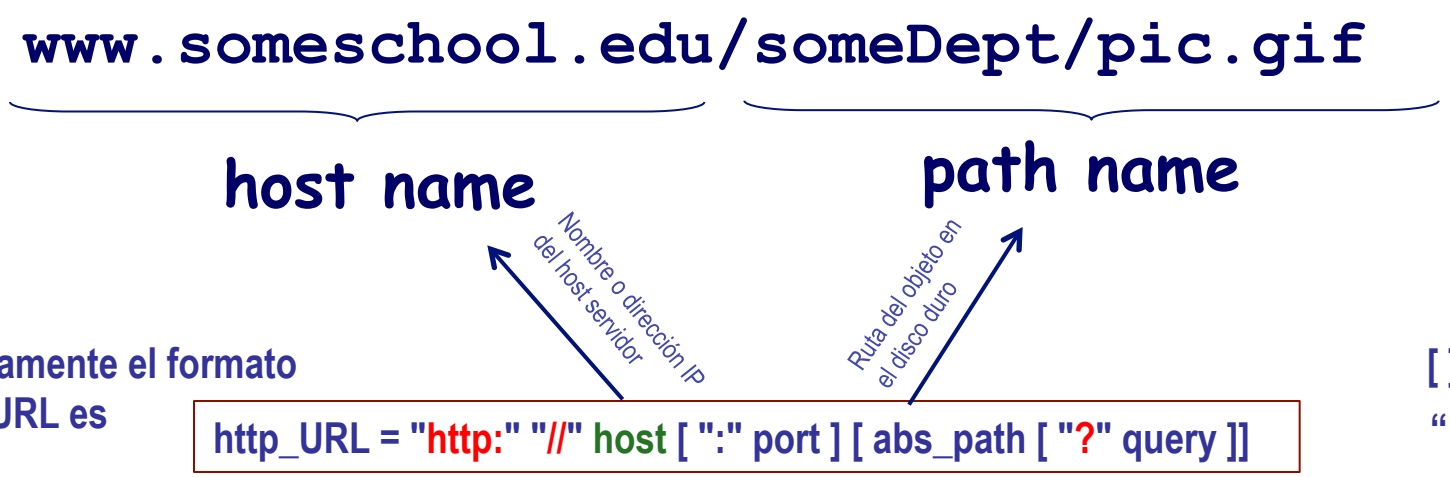


Aplicaciones de Ejemplo

La web clásica (práctica 1)

Web y HTTP: conceptos previos

- Página Web: es un documento que está compuesto por **objetos**.
- Una página web consiste en un fichero base en formato HTML que incluye referencia a otros objetos que están también almacenados como ficheros de tipo imágenes JPEG, applets de java, audio, etc..
- Cada objeto es direccionable (accesible) por una URL (Universal Resource Locator)
- Ejemplo de URL:



Genéricamente el formato de una URL es

http_URL = "http:" "//" host [":" port] [abs_path ["?" query]]

[ ] opcional

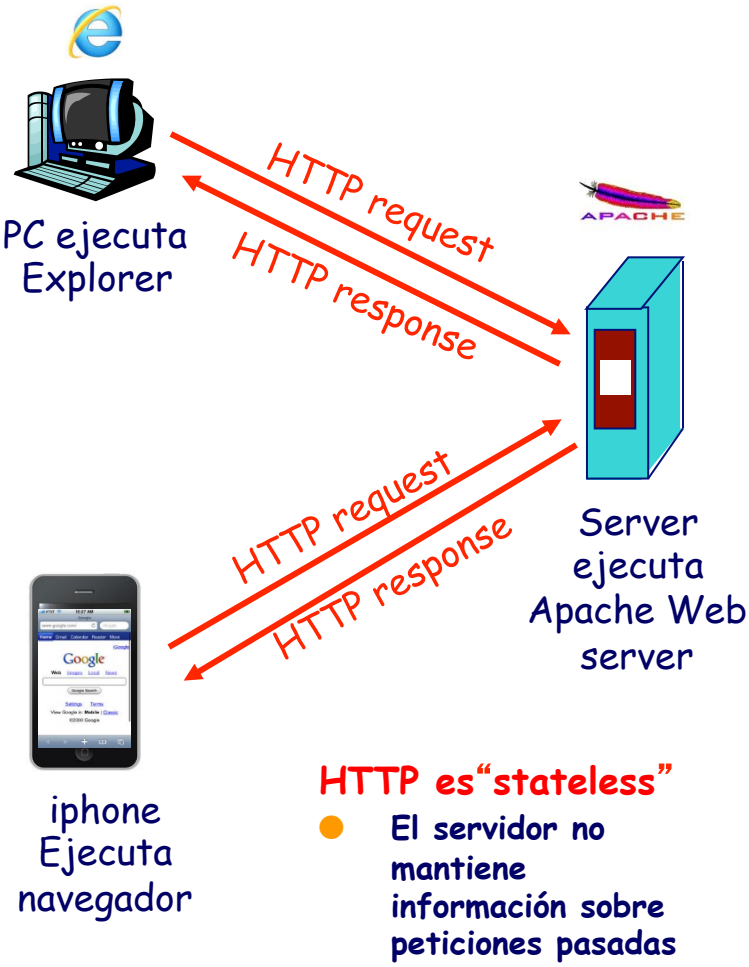
" " literal

HTTP (Hypertext Transfer Protocol)

- Protocolo para la aplicación Web.
  - Tipo textual
  - Dos tipos de mensajes HTTP
    - Petición (request)
    - Respuesta (response)

- Arquitectura C/S
  - **Cliente**: (navegador) envía peticiones y recibe y representa los objetos web
  - **Servidor Web**: envía objetos en respuesta a las peticiones del cliente

- La aplicación web usa el servicio de TCP para el transporte de mensajes
  - TCP es un protocolo orientado a conexión

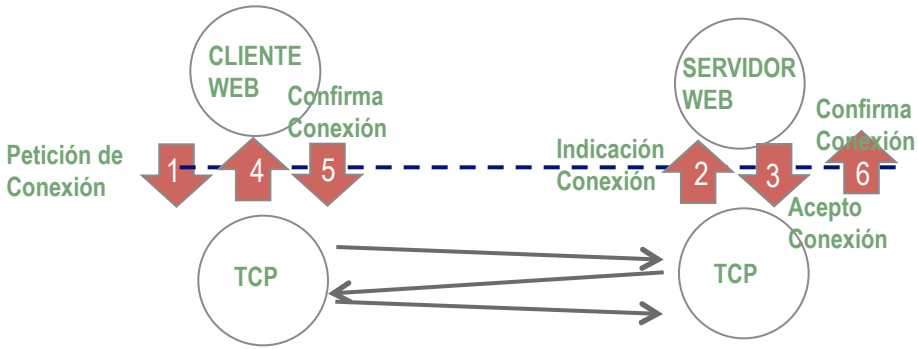


HTTP es "stateless"

- El servidor no mantiene información sobre peticiones pasadas

Conexiones usadas por HTTP

- La conexión debe ser solicitada por el cliente y aceptada por el servidor (puerto 80 TCP) previamente al envío de datos.
  - El cliente crea un socket TCP y le pide que se conecte al socket del servidor



(el servicio de creación y cierre de conexión es doblemente confirmado)

- Una vez establecida la conexión, los dos procesos de aplicación pueden intercambiar mensajes del protocolo HTTP
- Una vez finalizado el intercambio, cierran la conexión

HTTP No persistente

- Por una conexión TCP sólo se puede enviar como máximo un objeto
- Cada objeto requiere su propia conexión TCP.

HTTP persistente

- Por una conexión TCP se pueden enviar multiples objetos

Mensajes de Petición HTTP

- Escritos en ASCII (legibles por personas)

método URL versión

carriage return

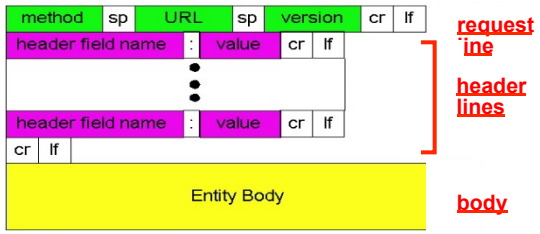
line-feed

Línea de petición (comandos GET, POST, HEAD)

Líneas de cabecera

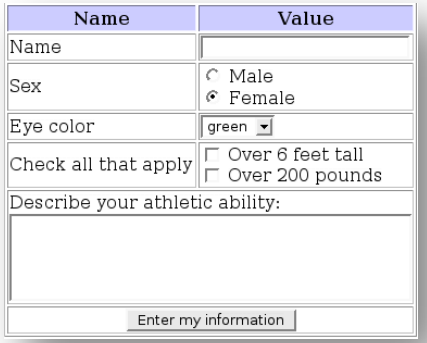
carriage return, line feed al comienzo de línea indica el final de las líneas de cabecera y el comienzo del cuerpo del mensaje (Body)

```
GET /index.html HTTP/1.1\r\n
Host: www-net.cs.umass.edu\r\n
User-Agent: Firefox/3.6.10\r\n
Accept: text/html,application/xhtml+xml\r\n
Accept-Language: en-us,en;q=0.5\r\n
Accept-Encoding: gzip,deflate\r\n
Accept-Charset: ISO-8859-1,utf-8;q=0.7\r\n
Keep-Alive: 115\r\n
Connection: keep-alive\r\n
\r\n
```



Enviando al servidor información de formularios

- Los mensajes de petición solicitan objetos pero…
- HTML permite el uso de formularios dentro de una página web
  - El formulario esta compuesto por objetos de recolección de datos de usuario (cuadros de texto, selectores, etc..)
  - La información que entra el usuario debe ser subida al servidor para su procesamiento (C→S)
    - Normalmente a través de algún lenguaje de programación (p.e. php) .. Páginas dinámicas .
- HTTP permite subir esta información (C→S)
  - Método POST
    - Sube la información en el cuerpo del mensaje
  - Método de la URL (usa el método GET)
    - Sube la información en el campo URL de la línea de petición:
      - www.miweb.com/search?Name=Antonio&EyeColor=2



Métodos de las peticiones según el protocolo

HTTP/1.0 (RFC 1945)

- GET (solicita objeto)
  - Sube información de formularios en URL
- POST(solicita objeto)
  - Sube información de formularios en el cuerpo
- HEAD(solicita objeto)
  - Pide al servidor que no incluya el objeto pedido en la respuesta

HTTP/1.1 (RFC 2616)

- GET, POST, HEAD
- PUT
  - Sube fichero en el cuerpo a la ruta espcificada en URL
- DELETE
  - Borra fichero especificado en el campo URL

Mensaje de Respuesta HTTP

200 OK
request succeeded, requested object later in this msg
301 Moved Permanently
requested object moved, new location specified later in this msg (Location:)
400 Bad Request
request msg not understood by server
404 Not Found
requested document not found on this server
505 HTTP Version Not Supported

Ahora en Inglés!

Status Line (protocol status code status phrase)

header lines

data, e.g., requested HTML file

```
HTTP/1.1 200 OK\r\n
Date: Sun, 26 Sep 2010 20:09:20 GMT\r\n
Server: Apache/2.0.52 (CentOS)\r\n
Last-Modified: Tue, 30 Oct 2007 17:00:02 GMT\r\n
ETag: "17dc6-a5c-bf716880"\r\n
Accept-Ranges: bytes\r\n
Content-Length: 2652\r\n
Keep-Alive: timeout=10, max=100\r\n
Connection: Keep-Alive\r\n
Content-Type: text/html; charset=ISO-8859-1\r\n
\r\n
data data data data data ...
```

Servidores “con estado”: cookies

RFC 6265

- Los extremos del protocolo pueden mantener el estado durante múltiples transacciones (el servidor quiere “conocerte”)
  - P.e. Para saber tu actividad pasada o personalizar una página
- Mecanismo

Requiere dos líneas de cabecera:

Set-cookie:

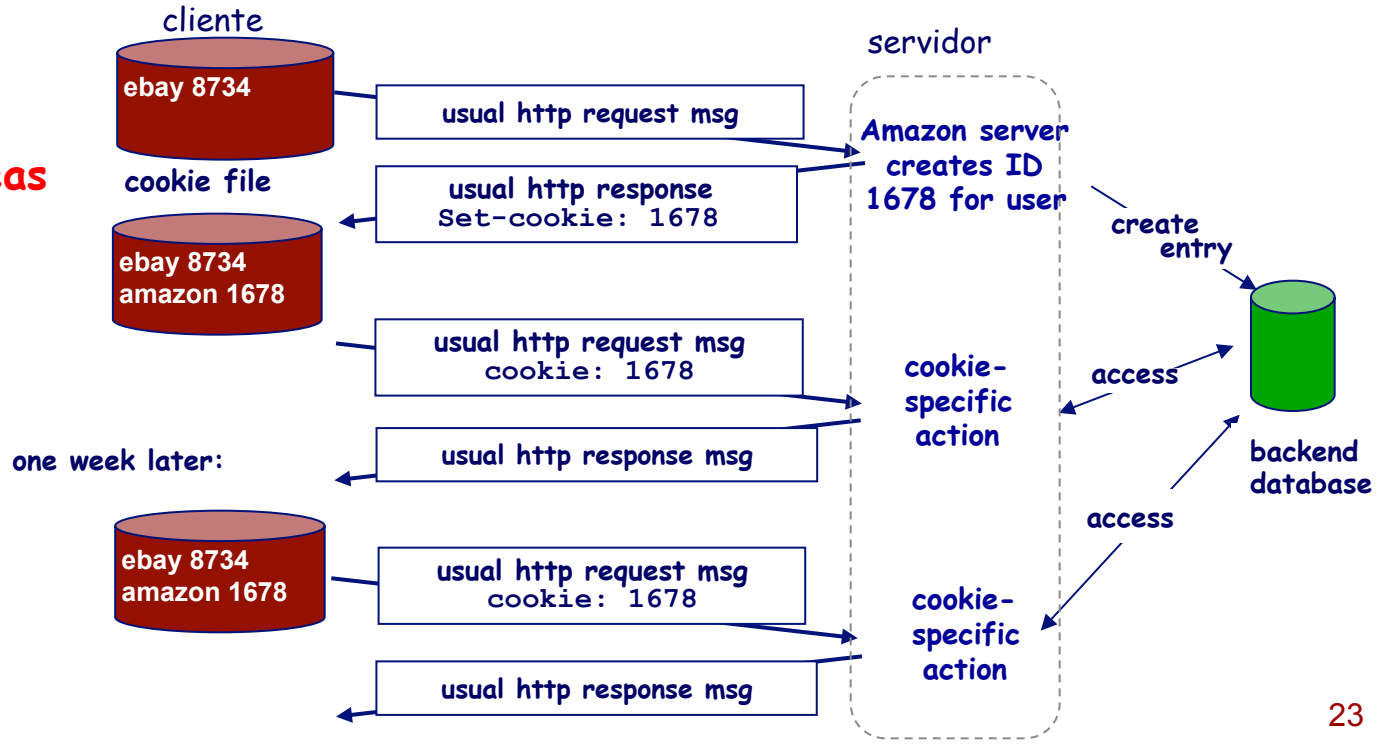
Cookie:

Internet Engineering Task Force (IETF) — A. Barth
Request for Comments: 6265 — U.C. Berkeley
Obsoletes: 2965 — April 2011
Category: Standards Track
ISSN: 2070-1721

HTTP State Management Mechanism

Abstract

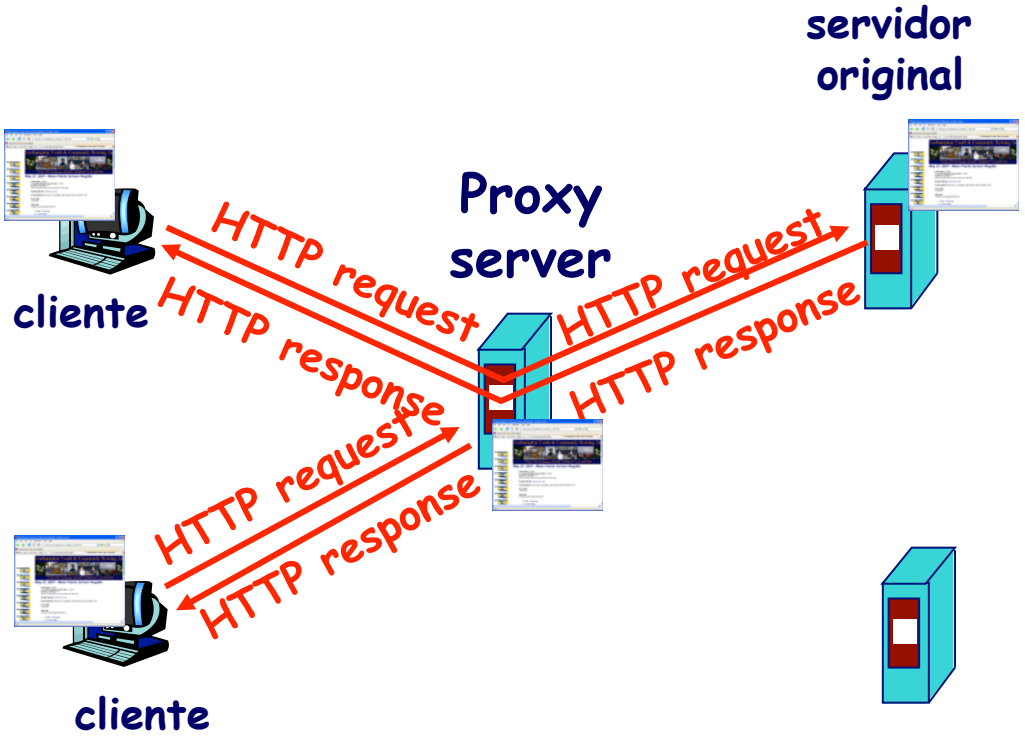
This document defines the HTTP Cookie and Set-Cookie header fields. These header fields can be used by HTTP servers to store state (called cookies) at HTTP user agents, letting the servers maintain a stateful session over the mostly stateless HTTP protocol. Although cookies have many historical infelicities that degrade their security and privacy, the Cookie and Set-Cookie header fields are widely used on the Internet. This document obsoletes RFC 2965.



Cache Web (Servidores Proxy)

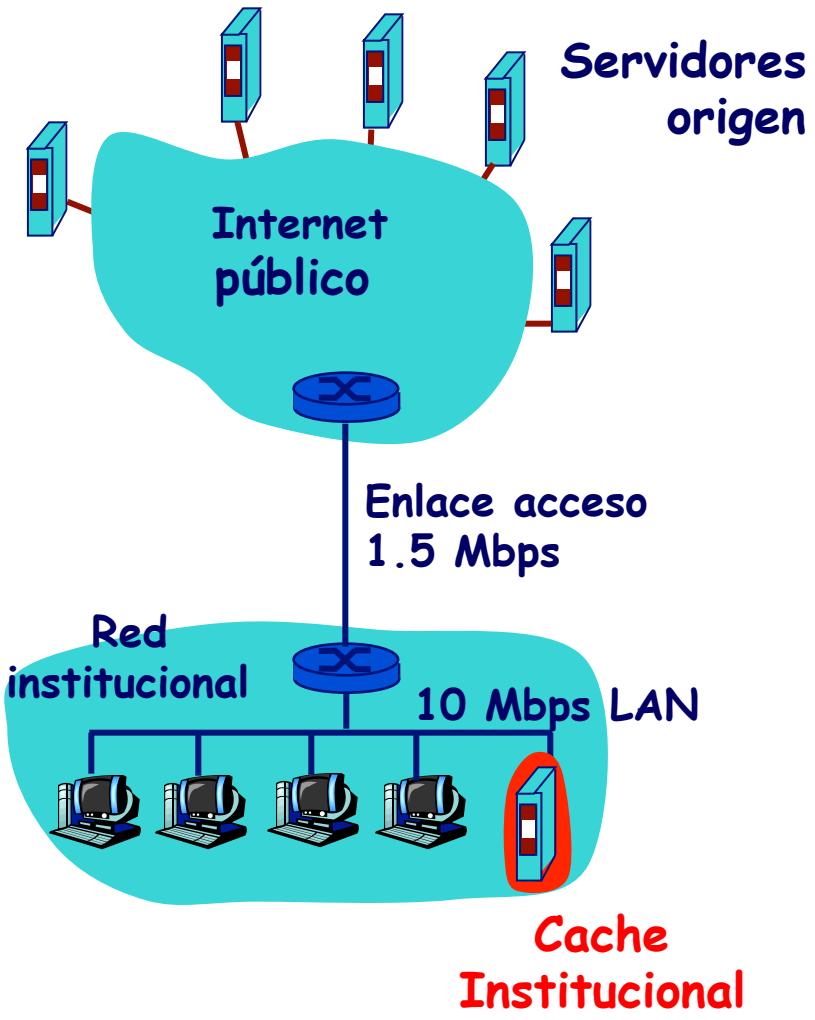
- Objetivo: satisfacer las peticiones de los clientes sin involucrar a los servidores

- **Usuarios**: configuran el navegador para usar el proxy
- El navegador envía todas las peticiones al proxy
  - Si el proxy las tiene en caché las envía a los usuarios
  - Si no, las solicita al servidor original antes de enviársela al cliente.



Cache Webs

- El Proxy o cache web hace las veces de cliente y servidor
- Suele ser instalado por los ISPs (universidad, empresa, etc..)
- ¿Qué ventajas tiene?
  - Reduce el tiempo de respuesta para atender la petición del cliente
  - Reduce el tráfico en la red de acceso del cliente
  - Incrementa la seguridad



… y si cambia algún objeto en el Servidor Web ?

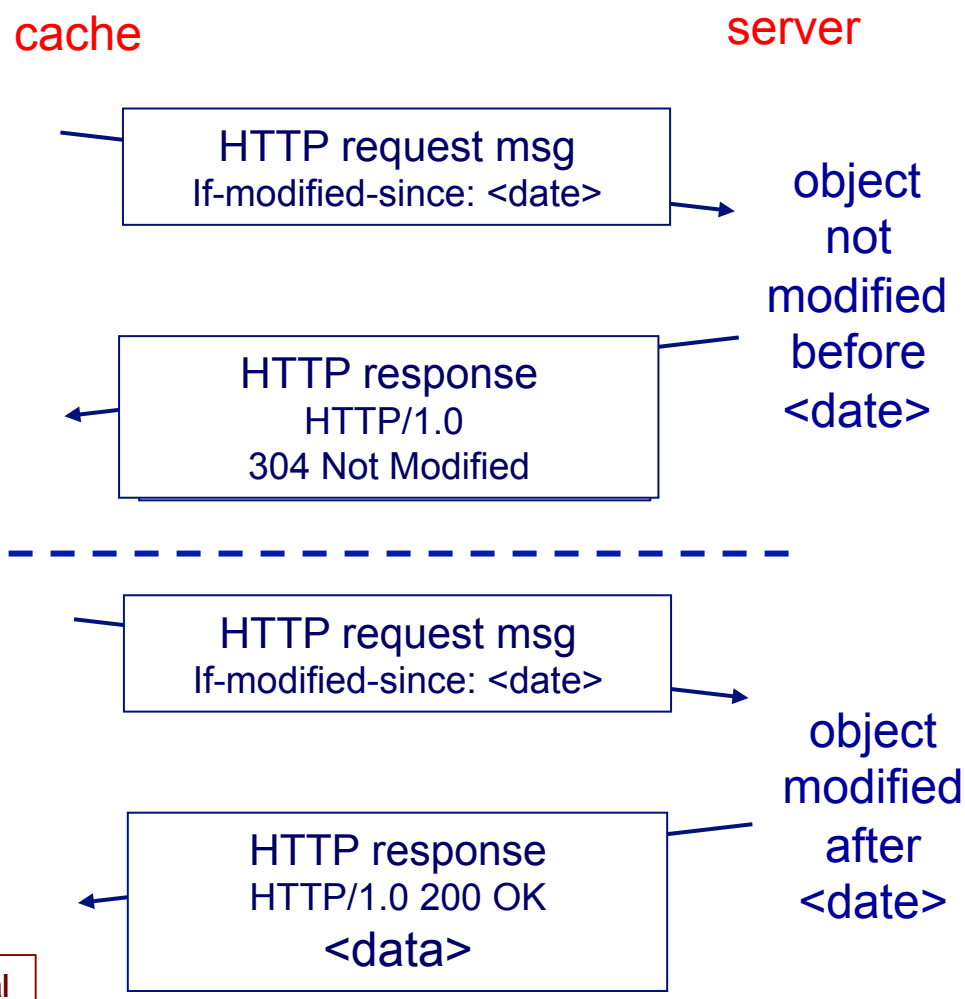
- Conditional GET
- El servidor no envía el objeto si el proxy tiene una versión actualizada
  - La fecha de la versión en cache se envía como una cabecera

```
If-modified-since: <date>
```

  - El servidor comprueba la fecha del caché con la del objeto que almacena. Si esta actualizada, le envía

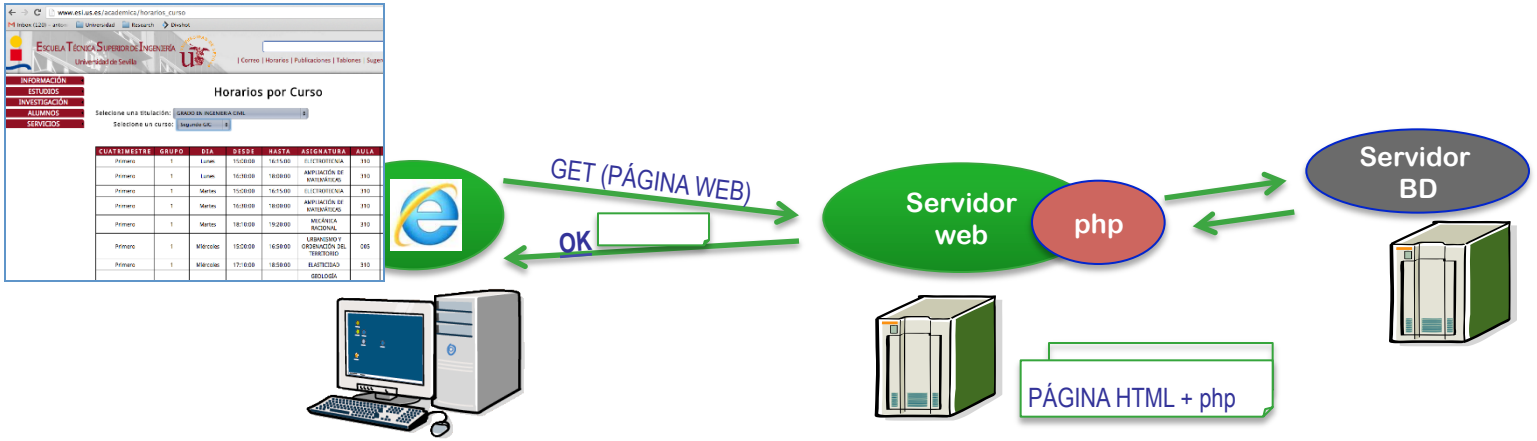
```
HTTP/1.0 304 Not Modified
```

Los propios navegadores suelen mantener un cache local y usan conditional Get para asegurarse de que su cache esta actualizada



La Web para usos “no clásicos”

- Los servidores y clientes web podrían ejecutar una lógica que utilizase los datos de los mensajes HTTP
  - P.e. El contenido de la línea de petición, líneas de cabecera o cuerpo del mensaje (información de formularios)
- Una vez ejecutada esa lógica, el servidor devolverá un mensaje de respuesta HTTP
  - el cuerpo del mensaje no tiene por qué ser un fichero estático sino algo “generado” ad-hoc (página dinámica)
- Otras veces, el servidor enviará un trozo de código al para que éste lo ejecute (javascript)



Aplicaciones de Ejemplo

DNS – Sistema de nombres de dominio (práctica 2)

Aplicación DNS: Domain Name System

- Es mas fácil para las personas recordar nombres que direcciones numéricas

- Sin embargo, para las máquinas es al revés (dirección IP: 182.12.52.88)
  - Los routers usan la dirección IP para identificar al destino.

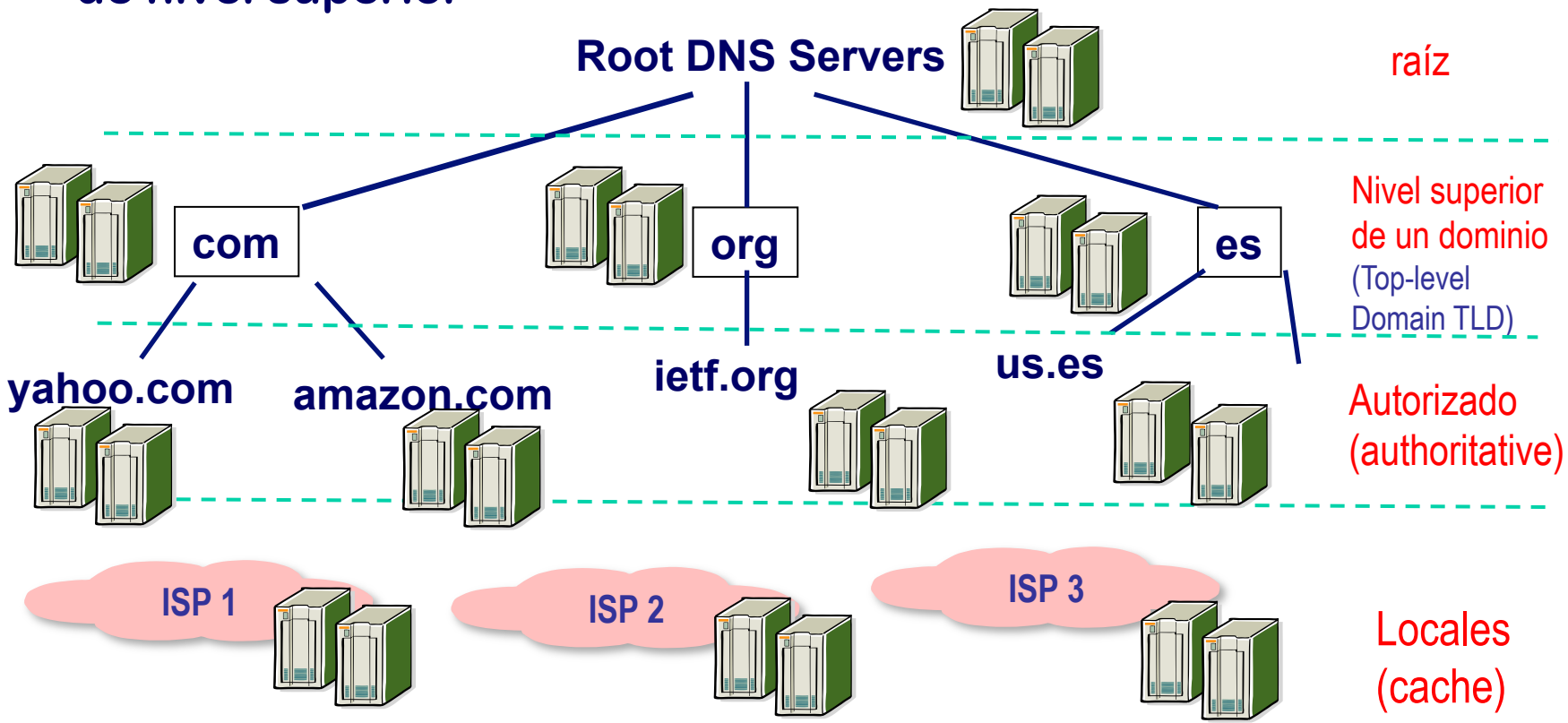
- El sistema de Nombres de dominio (DNS) es una aplicación distribuida
- Compuesta por:
  - Una base de datos **distribuida** que asocia los nombres de las máquinas → direcciones IP
  - Un protocolo de aplicación para la consulta (**resolución**) de la base de datos
    - Usa servicio UDP (puerto 53)
- Es básica en Internet
  - Reside en el borde de la red

Aplicación DNS: Domain Name System

- Tiene varias aplicaciones (además de la evidente)
  - Traducción de nombre a dirección IP
  - Hosts Aliasing (múltiples nombres para la misma máquina)
    - Nombre canónico, nombres alias.
  - Alias de servidores de email
  - Distribución de carga
    - Servidores Web replicados con diferentes IPs y bajo el mismo nombre.
- Es un aplicación C/S, pero no hay un único servidor. Los datos se distribuyen entre varios servidores
  - Evita
    - Único punto de fallo
    - Excesivo volumen de tráfico
    - Excesivo tiempo de respuesta
    - Excesivo Mantenimiento (añadir, cambiar, borrar datos)

La Base de Datos de DNS es distribuida y jerárquica

- Varios servidores DNS en cada nivel de la jerarquía.
- Los servidores de un nivel conocen, al menos la dirección IP de los servidores dentro de su dominio y la de algún servidor de nivel superior



Jerarquía de Servidores DNS

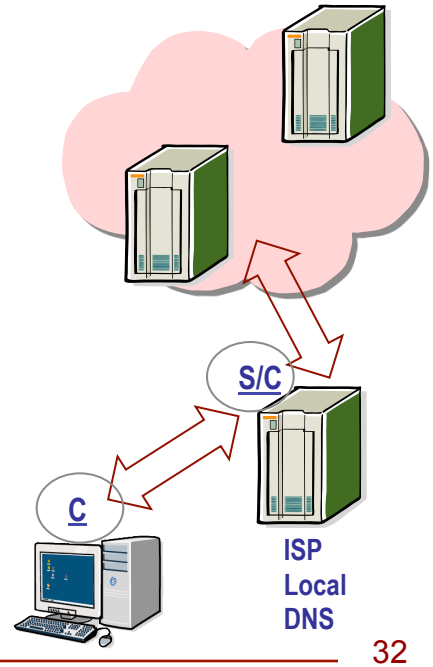
- Raíz (root)
  - Los servidores Locales pueden contactar con él si no tienen otra alternativa.
  - Contacta con servidor de nombres autorizado (authoritative) si no conoce el mapeo para resolver un nombre
  - Hay 13 distribuidos por el mundo (con réplicas)

http://www.root-servers.org

- Nivel superior del dominio (TLD)
  - Responsables de dominios de nivel superior (com , net, org) y y todos los dominios de nivel superior nacional (uk, es, etc..)
    - Network solutions mantiene com http://www.networksolutions.com

- Servidores Autorizados (authoritative)
  - Servidores DNS de una organización (dominio).
  - Ofrecen mapeos "autorizados" o fiables para los servidores de un dominio
  - Pueden ser mantenidos por la propia organización o un SP (prov.)

- Servidores de Nombre Locales (no pertenecen a la jerarquía)
  - Cada ISP tiene al menos uno (default name server)
  - Los clientes DNS de los hosts realizan su consulta siempre a los servidores Locales (actúan como una especie de proxy)



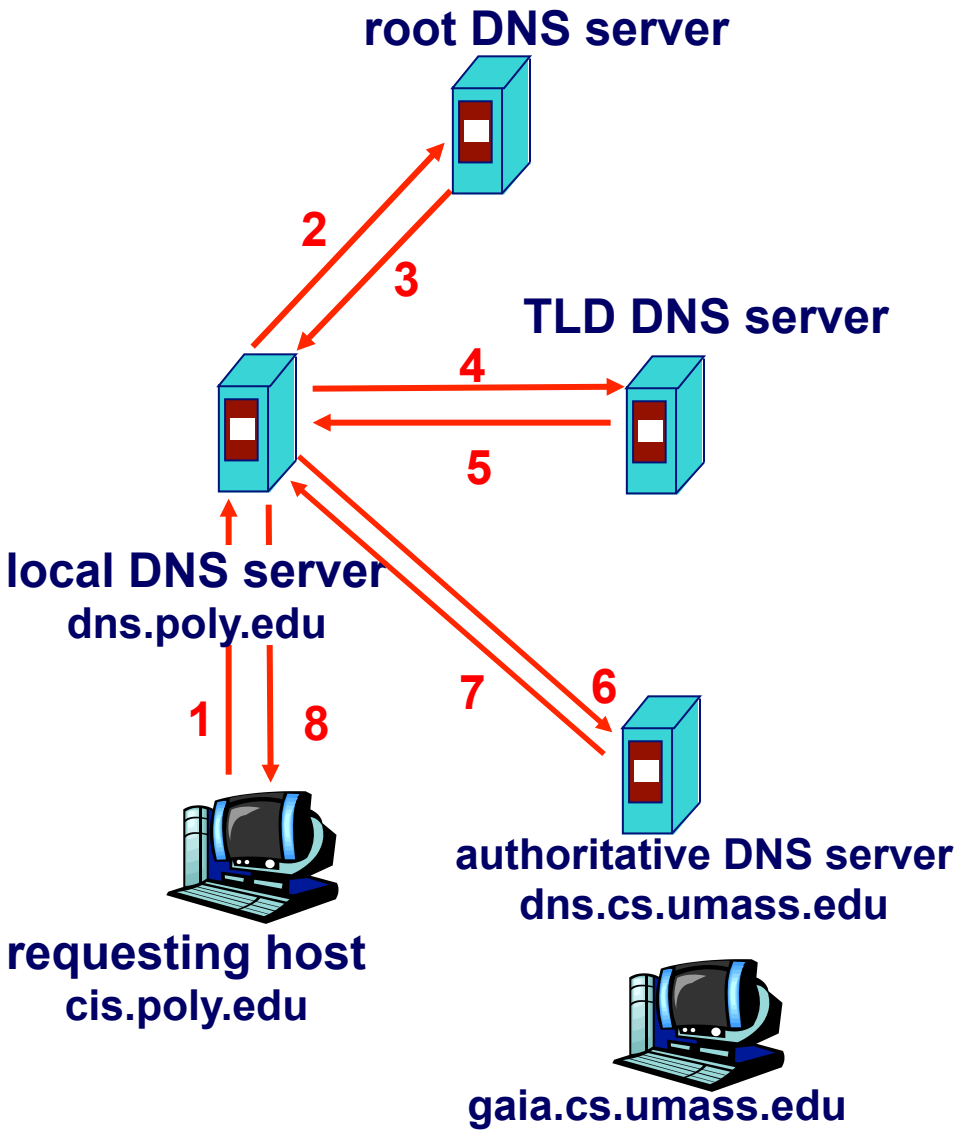
Ejemplo Resolución DNS

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

Consulta ITERATIVA :

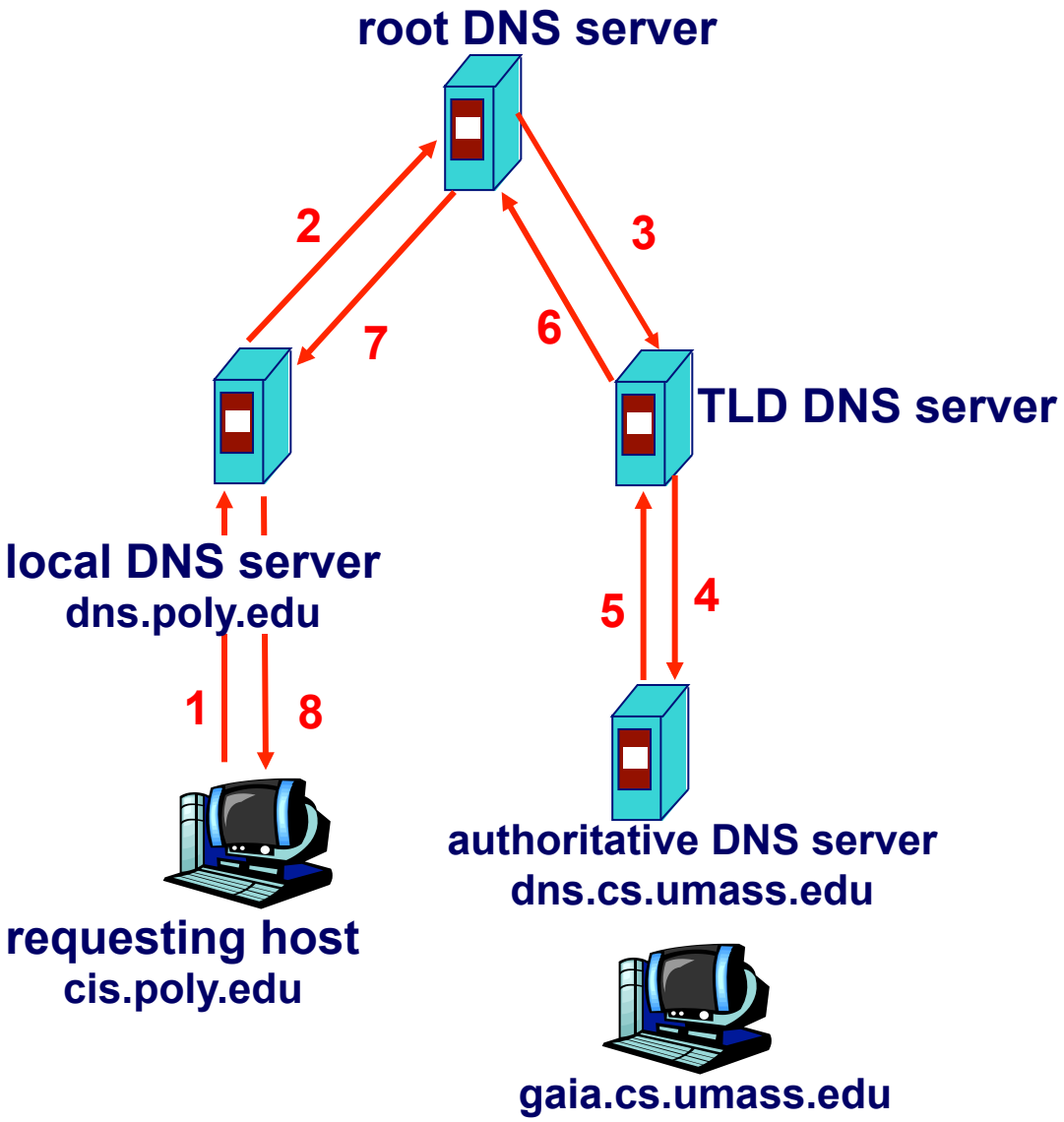
- El Servidor contactado responde con el nombre del servidor con el que contactar
- "No lo se, pero pregunta a este"

También podría haberse resuelto de forma recursiva



Ejemplo Resolución DNS (II)

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



Consulta RECURSIVA:

- El Servidor contactado es el responsable de resolver

Hay una caché local en cada servidor donde guarda las resoluciones aprendidas (cada nueva entrada caduca después de un tiempo)

Los TLD suelen estar en la caché de los locales (el root no se suele usar)

Registros de Recursos DNS (DNS Resource Record)

- Es la unidad información que se almacena en la BBDD distribuida

Formato RR: `(name, value, type, ttl)`

- Cuatro tipos de Registros de Recursos (principales)

Type=A

- `name` →hostname
- `value` → dirección IP

Type=NS

- `name` →dominio (p.e. foo.com)
- `value` → hostname de un servidor DNS autorizado para el dominio

Type=CNAME

- `name` → alias de algún nombre "canonico" (el real)
- `www.ibm.com` es realmente `servereast.backup2.ibm.com`
- `value` → nombre "canonico"

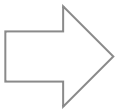
Type=MX

- `value` → nombre del servidor de email asociado al dominio (`name`)

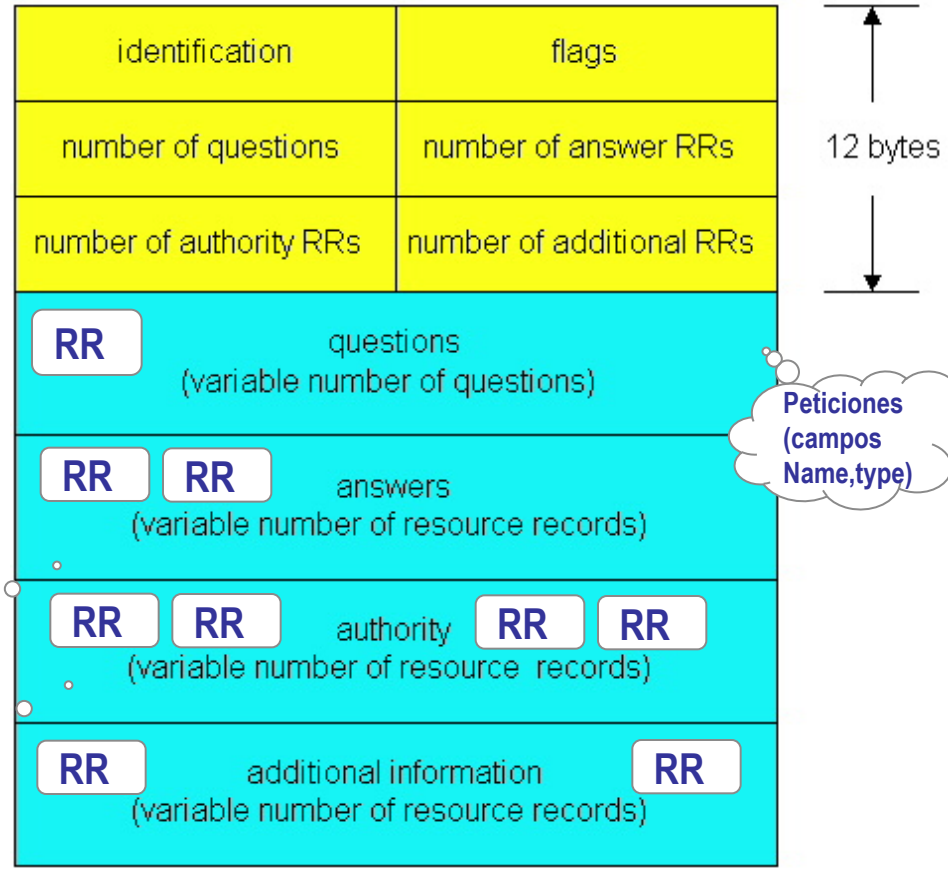
Protocolo de la aplicación DNS: mensajes

- Protocolo de petición/ respuesta (cliente/servidor)
- Usa el servicio de transporte UDP (pto 53)
- Ambos tipos de mensajes con igual formato
- Cabecera (A-PCI)
  - Identificador: # 16bits usado para correlacionar req/res
  - Flags:
    - Req / Res (Query o Reply)
    - Se desea recursividad
    - Se dispone de recursividad
    - La respuesta es autorizada (fiable)

RFC 882
RFC 883
(1983)



RFC 1034
RFC 1035
(1987)



¿cómo inserto un nuevo registro en el DNS?

- Por ejemplo: nueva startup "botijos.com"
- Registro el nombre "botijos.com" en un Registrar DNS. (p.e network solutions)
  - Le doy el nombre, dirección IP de los servidores autorizados (primario y secundario)
  - El registrar inserta estos dos RRs en el servidor com TLD

```
(botijos.com, dns1.botijos.com, NS)
(dns1.botijos.com, 212.212.212.1, A)
```

- Configuro los Servidores Autorizados (o lo alquilo) para mi dominio
  - Un registro tipo A para www.botijos.com; (servidor web)
  - Un registro tipo MX para botijos.com (servidor email)

Lista de Registrars http://www.internic.net/alpha.html