



# Arquitectura de Redes 1: Práctica 1

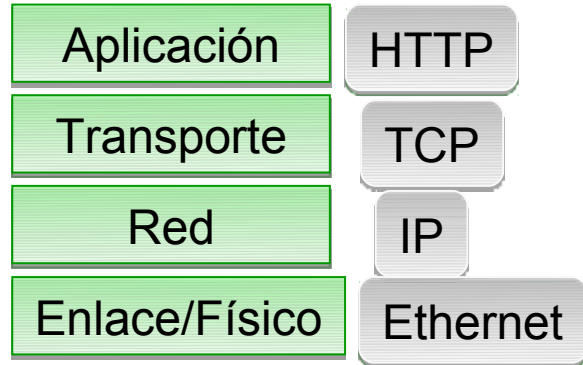
# Práctica 1

- Leer enunciado en la web.
- Objetivo: implementación de un servidor sencillo HTTP.

# Contexto



10.0.0.2



Servidor Web



10.1.0.3

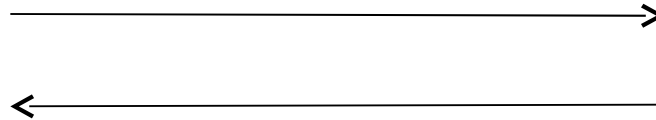


# Introducción

- ¿Qué es HTTP?
  - Protocolo de nivel de aplicación usado para transacciones WWW.
- ¿Por qué es importante el protocolo HTTP?
  - Google, Facebook, Youtube, etc
  - Aplicaciones móviles
  - ~46% del tráfico de Internet es HTTP

# Fundamentos protocolo HTTP

- RFC 2616:<http://www.faqs.org/rfcs/rfc2616.html>
- Funciona sobre TCP puerto 80.
- Texto plano.
- Modelo petición-respuesta:
  - Navegador envía petición de un recurso web.
  - El servidor comprueba la existencia del recurso y, en tal caso, envía la respuesta con el recurso e información del mismo.



Servidor Web



# Petición genérica

MÉTODO RECURSO VERSIÓN\r\n

Opcion1\r\n

Opcion2\r\n

.

.

.

OpcionN\r\n

\r\n

[Data]

# Peticiones

- Método:
  - GET: el método más común.
  - POST: para subir información en formularios
  - PUT, DELETE, ...: más métodos que no implementaremos
- Recurso: identificados por URIs
  - Relativos o absolutos
  - Caso especial:
    - / es la página por defecto
- Versión: HTTP/1.0 ó HTTP/1.1

# Primer ejemplo: petición

GET / HTTP/1.1

Host: www.google.es

Connection: keep-alive

Accept: application/xml,application/xhtml+xml,text/html;q=0.9,text/plain;q=0.8,image/png,\*/\*;q=0.5

User-Agent: Mozilla/5.0 (X11; U; Linux x86\_64; en-US) AppleWebKit/534.7 (KHTML, like Gecko) Chrome/7.0.517.44 Safari/534.7

Accept-Encoding: gzip,deflate,sdch

Accept-Language: es-ES,es;q=0.8

Accept-Charset: ISO-8859-1,utf-8;q=0.7,\*;q=0.3

Cookie: NID=41=uBDgNAUYK-

J0rl6WIVYr9q4urtJ6U7BEEvH8EMIZ4mGA7BYZfC1PFaspJbYwHoN\_6U71aA  
GRdcdfhixCBjPuDUa1-4ST9oQkP0Qjfz\_KI1GeiD8wkGFf20PeTvjcxeW3;



# Respuesta genérica

VERSIÓN CÓDIGO\_ESTADO DESCRIPCIÓN\_CÓDIGO\r\n

Opción1\r\n

Opción2\r\n

.

.

.

OpciónN\r\n

\r\n

Data

# Respuestas

- Código y descripción:
  - RFC 2626: <http://www.faqs.org/rfcs/rfc2616.html>
  - Los más importantes:
    - 200: OK
    - 404: Not found
    - 403: Forbidden
    - 500: Internal error

# Respuestas

- Opciones:
  - Content-Length: Longitud en bytes de los datos (no tiene en cuenta la cabecera). Si no hay datos, se obvia esta opción.
  - Content-Type: Tipo MIME de los datos. Se obtiene a partir de la extensión del archivo.
    - Lo leeremos del fichero /etc/mime.types
    - El caso por defecto (si no se encuentra):  
application/octet-stream.
  - Server: información del servidor.
  - Host: dirección IP o nombre del servidor al que se realiza la petición.
  - Max-Forwards: limitar número the redirecciones.

# Primer ejemplo: respuesta

**HTTP/1.1 200 OK**

Date: Tue, 14 Dec 2010 15:46:06 GMT

Expires: -1

Cache-Control: private, max-age=0

**Content-Type: text/html; charset=UTF-8**

Set-Cookie:

PREF=ID=024ab96d378d5a4b:U=71882ebd72f1d8f8:FF=0:LD=es:NR=10:T  
M=1277722809:LM=1292341566:SG=1:S=wLPiV-268UMeOOun;  
expires=Thu, 13-Dec-2012 15:46:06 GMT; path=/; domain=.google.es

Content-Encoding: gzip

Server: gws

**Content-Length: 8360**

X-XSS-Protection: 1; mode=block

**<html>.....</html>**

# Sockets

- Existen 3 tipos:
  - Socket STREAM: TCP (práctica 1 servidor HTTP).
  - Socket DATAGRAM: UDP (práctica 2 cliente DNS).
  - Socket RAW: Nivel 2 (no se usan en estas prácticas).
  
- Sockets TCP STREAM, leemos y escribimos del socket “como si fuera” un fichero.

# Sockets STREAM

- Primitivas básicas:

- Crear el socket:

```
sockfd = socket(AF_INET, SOCK_STREAM, 0);
```

- Para enlazar el socket con un puerto:

```
serv_addr.sin_family = AF_INET;
```

```
serv_addr.sin_addr.s_addr = INADDR_ANY;
```

```
serv_addr.sin_port = htons(port);
```

```
bind(sockfd, (struct sockaddr *) &serv_addr, sizeof(serv_addr))
```

- Para habilitar el socket para recibir conexiones:

```
listen(sockfd, max_conn);
```

- Para aceptar la conexión:

```
newsockfd = accept(sockfd, (struct sockaddr *) &cli_addr, &clilen);
```

# Pseudocódigo servidor

- **Leer configuración**
- Capturar Ctrl+C
- Abrir socket
- Enlazar puerto
- While (TRUE)
  - Esperar conexión
  - **procesar petición**
  
- Se proporciona un *template* de manejo de sockets TCP como servidor, que podéis tomar como base para el desarrollo del servidor.

# Template Sockets

- Ampliar servidor para que cuando el servidor web recibe una petición esta se analice:
  - Comprobar que es un GET.
  - Extraer el recurso solicitado.
  - Comprobar que existe el recurso.
  - Construir cabecera de respuesta.
    - Para rellenar content-type buscamos en /etc/mime.types
  - Enviar respuesta: cabecera + datos.



Puerto  
efímero

Puerto 80

Servidor Web





# ¿Cómo empezar?

- Descargarse el programa de ejemplo y entender cómo funciona:
  - Compilamos y ejecutamos.
  - Desde el navegador: <http://localhost:6000>
  - Ver en el servidor la petición HTTP.
- Modificaciones:
  - Al principio, lectura de configuración.
  - Abrir el puerto leído en la configuración (en vez del 6000 como ahora).
  - Modificar `process_request` para procesar peticiones HTTP (ver que es GET, que la versión es HTTP/1.1 y comprobar existencia recurso).
  - Modificar `process_request` para enviar respuesta: cabecera, `content-length`, `content-type` y datos.