TEMA 1

INTRODUCCIÓN A LAS TECNOLOGÍAS WEB: EL PROTOCOLO HTTP

APLICACIONES WEB - GIS - CURSO 2017/18

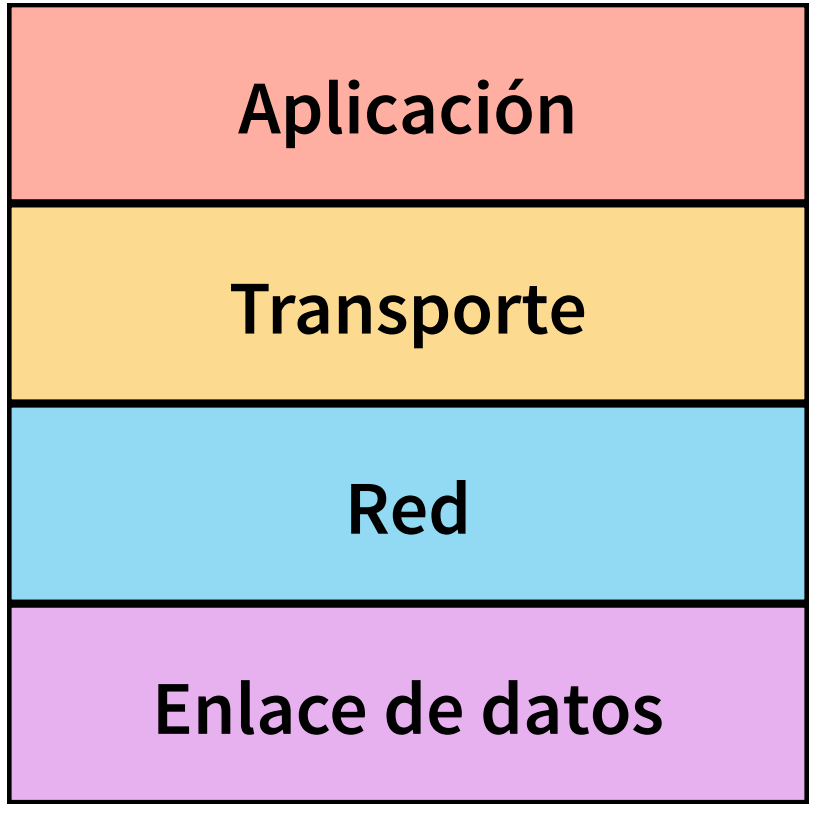
Manuel Montenegro [montenegro@fdi.ucm.es]
Dpto de Sistemas Informáticos y Computación
Facultad de Informática
Universidad Complutense de Madrid

1. REPASO DEL MODELO TCP/IP
2. MODELO CLIENTE/SERVIDOR
3. SOCKETS Y PROTOCOLOS
4. PROTOCOLO HTTP
5. REFERENCIAS

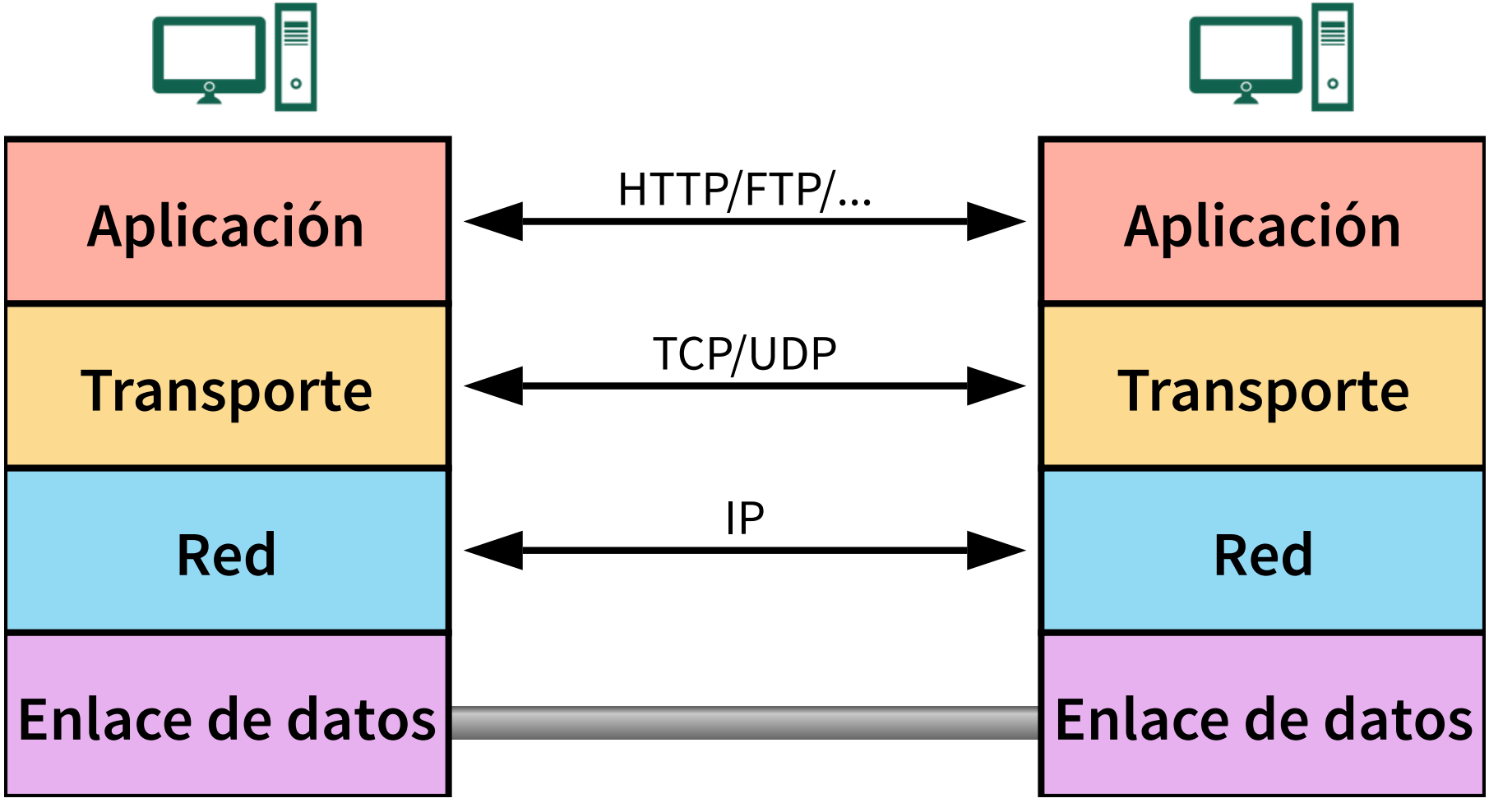
MODELO TCP/IP

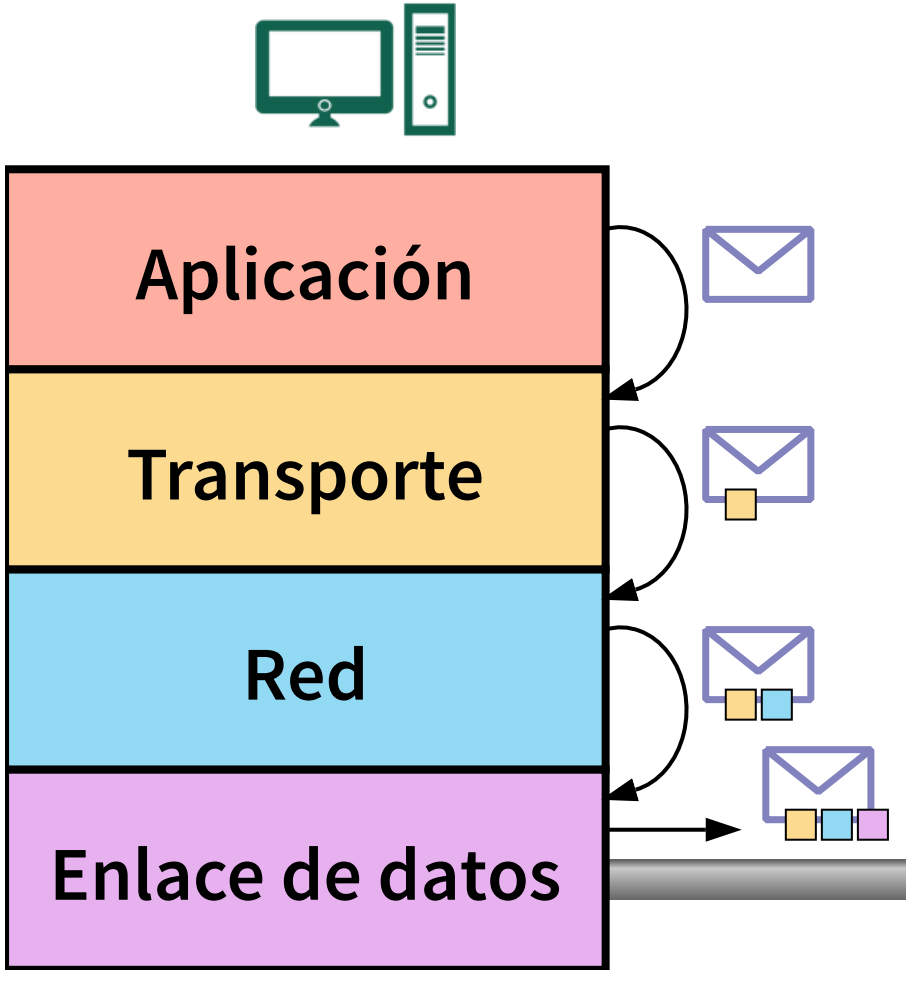
Basado en cuatro capas

Cada capa representa un nivel de abstracción



Cada capa se comunica con su homóloga en otro nodo distinto, utilizando un *protocolo* como lenguaje.





Sin embargo, la comunicación entre capas homólogas no es directa.

Cada capa solicita a su capa inferior que se envíe el mensaje al destino.

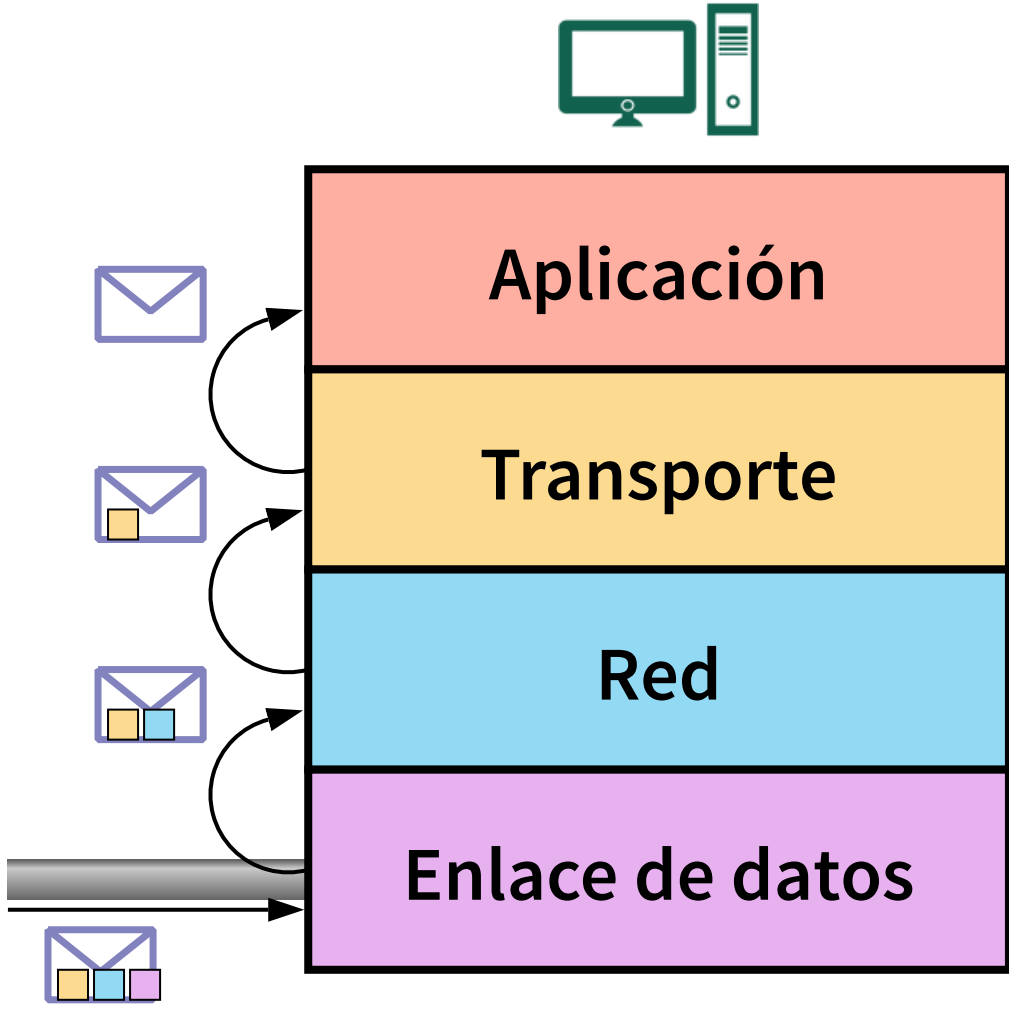
Cada capa extiende el mensaje a enviar con información propia que le permite realizar su cometido

Finalmente, la capa de enlace es la que realiza el envío físico

El nodo receptor recibe el mensaje a través de la capa de enlace.

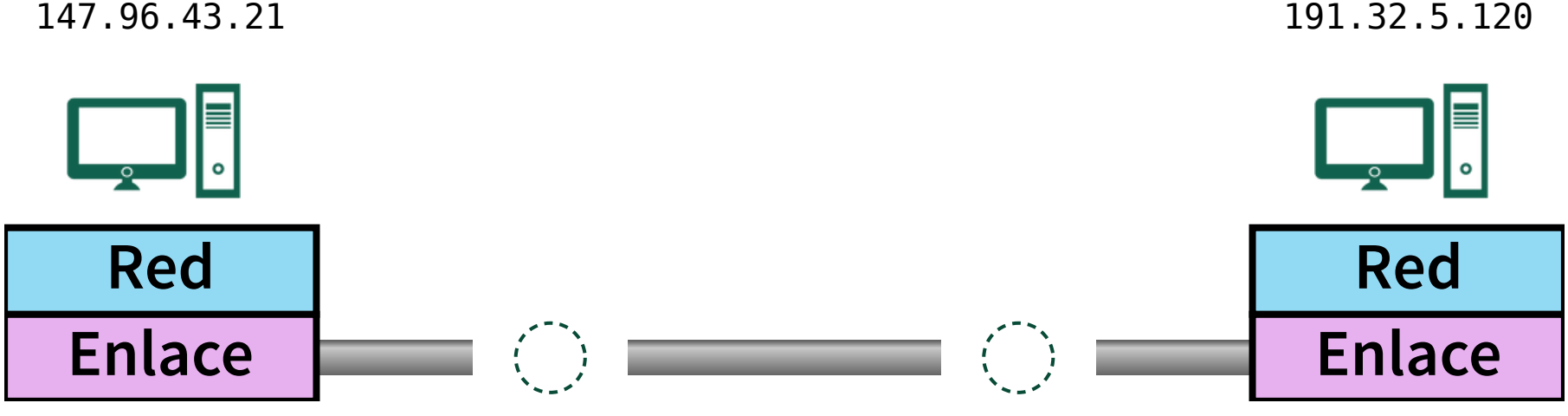
Cada capa interpreta la información adicional introducida por su capa homóloga durante el envío, y acaba remitiendo el mensaje original (sin esta información adicional) a la capa superior.

Finalmente, la capa de aplicación recibe el mensaje tal y como lo envió la capa de aplicación en el origen.

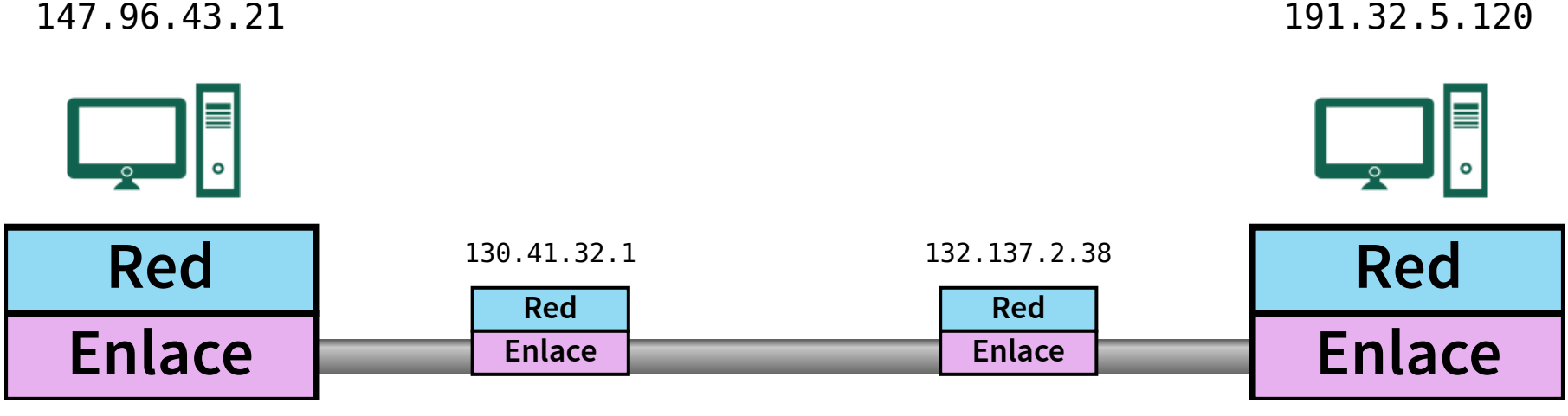


La capa de *enlace de datos* proporciona una vía de transmisión de paquetes de datos entre dos nodos enlazados directamente

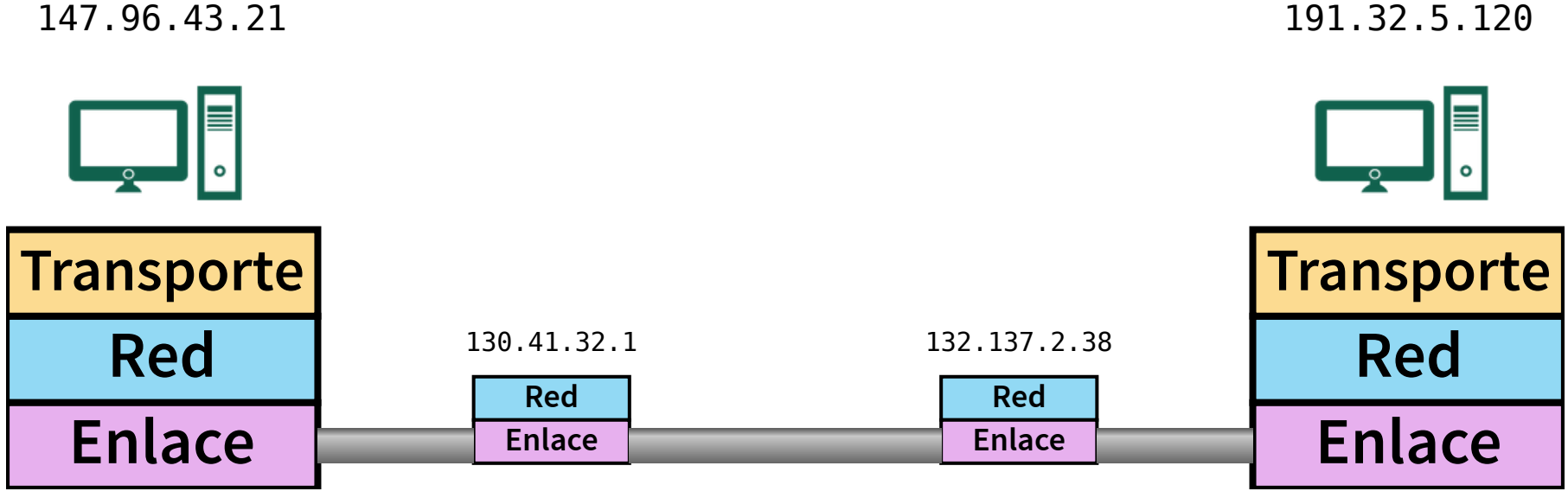
La capa de *red* gestiona el envío de mensajes a través de múltiples nodos intermedios.
Cada nodo está identificado mediante una *dirección IP*



Las capas de red de los nodos intervinientes *enrutan* los paquetes desde el origen hasta el destino



La capa de *transporte* se encarga de la transmisión sin errores desde origen a destino.
Segmentación en paquetes, recepción en orden correcto...



CAPA DE TRANSPORTE

Protocolos:

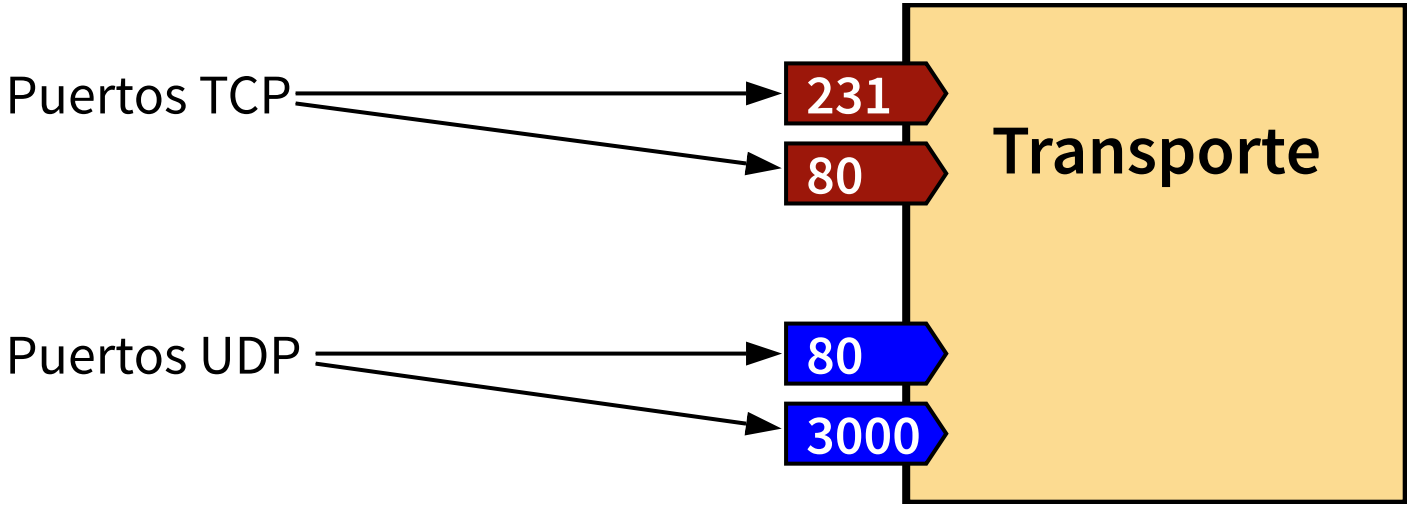
- **TCP**: orientado a conexión, bidireccional, fiable.
- **UDP**: no conexión, no comprobación de paquetes duplicados, extraviados, etc.

La capa de transporte permite varios canales de comunicación entre nodos.

Cada canal está identificado por un *número de puerto*.

Existen *puertos TCP* y *puertos UDP*.

EJEMPLO



PUERTOS TCP/UDP MÁS COMUNES

- **Puerto 20**: FTP (transmisión de archivos).
- **Puerto 22**: SSH (transmisión de archivos segura).
- **Puerto 25**: SMTP (envío de correo).
- **Puerto 80**: HTTP (web).
- **Puerto 110**: POP3 (recepción correo).
- **Puerto 143**: IMAP (recepción correo).
- **Puerto 443**: HTTPS (web segura).

Ver: List of TCP and UDP port numbers (Wikipedia)

1. REPASO DEL MODELO TCP/IP
2. MODELO CLIENTE/SERVIDOR
3. SOCKETS Y PROTOCOLOS
4. PROTOCOLO HTTP
5. REFERENCIAS

MODELO CLIENTE/SERVIDOR

Supongamos que dos nodos quieren conectarse para intercambiar información

¿Cómo y cuándo se establece esta conexión?

En el modelo *cliente/servidor* cada nodo adquiere un rol:

- El *servidor* permanece conectado a la espera de información que llegue por un determinado puerto. Se dice que el servidor *escucha* en dicho puerto.
- El *cliente* inicia la comunicación enviando información al servidor.

SERVIDOR

- Pasivo: espera a que un cliente envíe una petición
- Responde (sirve) las peticiones de varios clientes
- A cada tipo de petición se le llama *servicio*

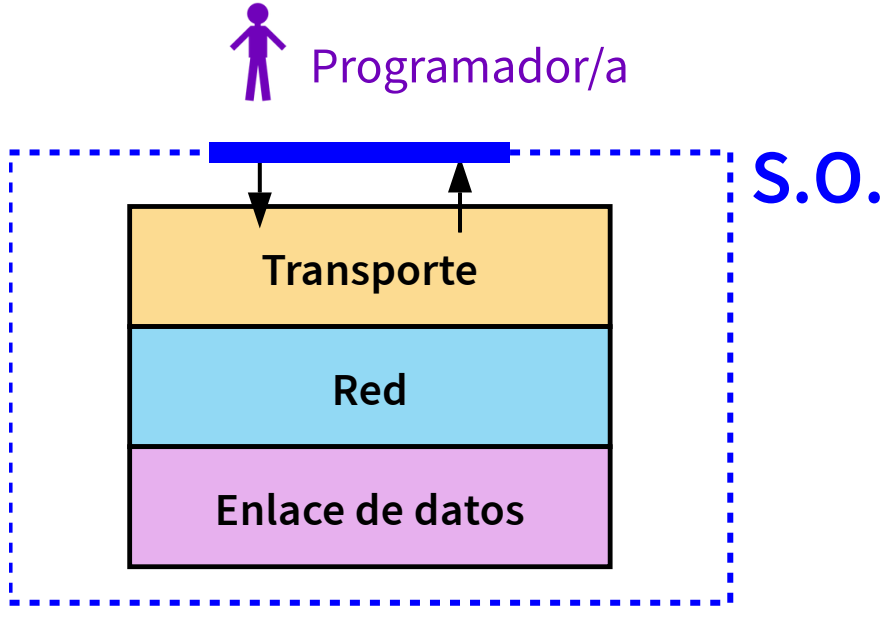
CLIENTE

- Activo: toma la iniciativa en la comunicación con el servidor
- Ha de conocer la dirección IP del servidor, y el puerto al que enviar la información (petición)

1. REPASO DEL MODELO TCP/IP
2. MODELO CLIENTE/SERVIDOR
3. SOCKETS Y PROTOCOLOS
4. PROTOCOLO HTTP
5. REFERENCIAS

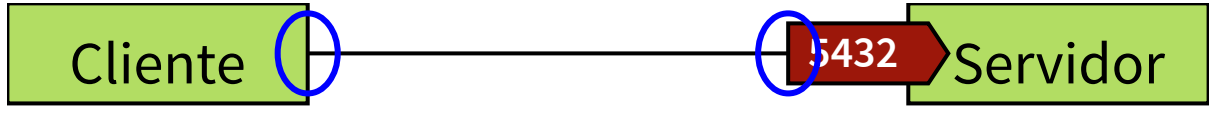
SOCKETS

El sistema operativo proporciona un mecanismo de acceso a la capa de transporte.



Este mecanismo se basa en el uso de *sockets*.

Supongamos que un servidor está escuchando en un determinado puerto y que un cliente se conecta al mismo:



Esto crea un canal de comunicación bidireccional entre cliente y servidor.

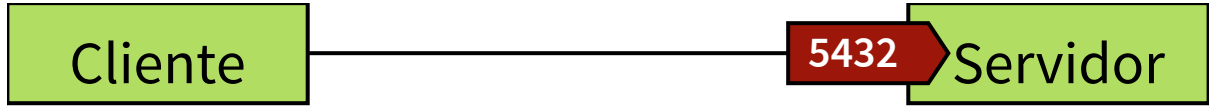
A cada uno de los extremos de este canal de comunicación se le llama *socket*

Cada socket tiene asociados dos flujos:

- **Flujo de entrada** (*input stream*)
  La información que llegue desde el otro extremo se lee a través de este flujo.

- **Flujo de salida** (*output stream*)
  Todo lo que se escriba en este flujo es enviado al otro extremo.

El programador lee y escribe en estos flujos como si de ficheros se tratase



SOCKET DE SERVIDOR

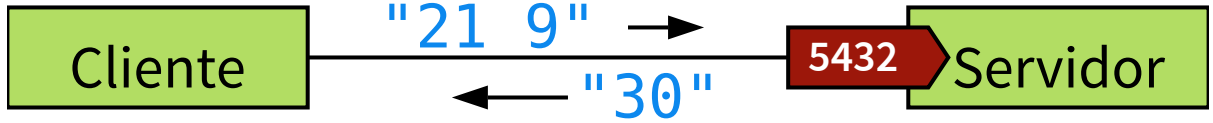
- Recibe las peticiones de a través de su flujo de entrada.
- Envía las respuestas a través del flujo de salida.

SOCKET DE CLIENTE

- Envía las peticiones a través del flujo de salida.
- Recibe las respuestas a través del flujo de entrada.

EJEMPLO: SERVIDOR CALCULADORA

Implementamos en Java un sencillo servidor que realiza la suma de dos números.



El cliente envía al servidor una cadena de texto con dos números enteros.

El servidor responde con la suma de dichos números.

IMPLEMENTACIÓN DEL SERVIDOR

```
public class ArithServer {
    // Número de puerto
    private static final int PORT_NUMBER = 5432;

    public static void main(String... args) {
        try {
            ServerSocket server = new ServerSocket(PORT_NUMBER);
            Socket s = server.accept();
            servePetition(s); // Implementado más adelante
        } catch (IOException e) {
            ...
        }
    }

    private static void servePetition(Socket s) { ... }
}
```

```
ServerSocket server = new ServerSocket(PORT_NUMBER);
```

Crea un objeto `ServerSocket`, que permite recibir conexiones a través del puerto dado.

```
Socket s = server.accept();
```

Queda a la espera de que se conecte un cliente.
El método `accept()` **detiene** la ejecución del programa hasta que reciba la conexión de un cliente.

El objeto de la clase `Socket` es el que permite acceder a los flujos de entrada y salida del socket:

```
public InputStream getInputStream();
public OutputStream getOutputStream();
```

El método `servePetition` recibe la petición y envía la respuesta a través de los flujos del socket.

```
private static void servePetition(Socket socket) {
  try (
      Scanner sc = new Scanner(socket.getInputStream());
      PrintWriter out =
        new PrintWriter(
          new OutputStreamWriter(socket.getOutputStream())
        )) {
    // Leer sumandos a partir del flujo de entrada
    int sum1 = sc.nextInt();
    int sum2 = sc.nextInt();

    // Escribir en el flujo de salida
    out.println(sum1 + sum2);
  } catch (IOException e) { ... }
}
```

Más información: Java try-with-resources

IMPLEMENTACIÓN DEL CLIENTE

```
public static void main(String... args) {
  try {
    Socket s = new Socket("localhost", PORT_NUMBER);

    // Enviar información al servidor y recibir respuesta
    int res = add(s, 21, 45);

    System.out.println("21 + 45 = " + res);
  } catch (IOException e) {
    System.err.println("Error de conexión: " + e.getMessage());
  }
}
```

La conexión a un servidor se realiza creando directamente una instancia de la clase `Socket`, indicando el servidor al que conectarse (`localhost` = propia máquina) y el puerto.

La función `add()` es la que envía la petición al servidor:

```
private static int add(Socket s, int n1, int n2) {
  try (PrintWriter out = new PrintWriter(
          new OutputStreamWriter(s.getOutputStream()));
       Scanner sc = new Scanner(s.getInputStream())) {
    // Enviar sumandos al servidor
    out.print(n1);
    out.print(" ");
    out.println(n2);
    out.flush();
    // Recibir resultado
    return sc.nextInt();
  } catch (IOException e) {
    System.err.println("Error de E/S: " + e.getMessage());
    return -1;
  }
}
```

Para ejecutar este ejemplo:

- Ejecutamos método `main()` del servidor.
- Ejecutamos método `main()` del cliente, obteniendo como salida:

```
21 + 45 = 66
```

- Tras esto, tanto el cliente como el servidor **finalizan**.

Si queremos que el servidor siga atendiendo peticiones de otros clientes, podemos hacerlo mediante un bucle que se repita indefinidamente:

```
ServerSocket server = new ServerSocket(PORT_NUMBER);
while (true) {
    Socket s = server.accept();
    servePetition(s);
}
```

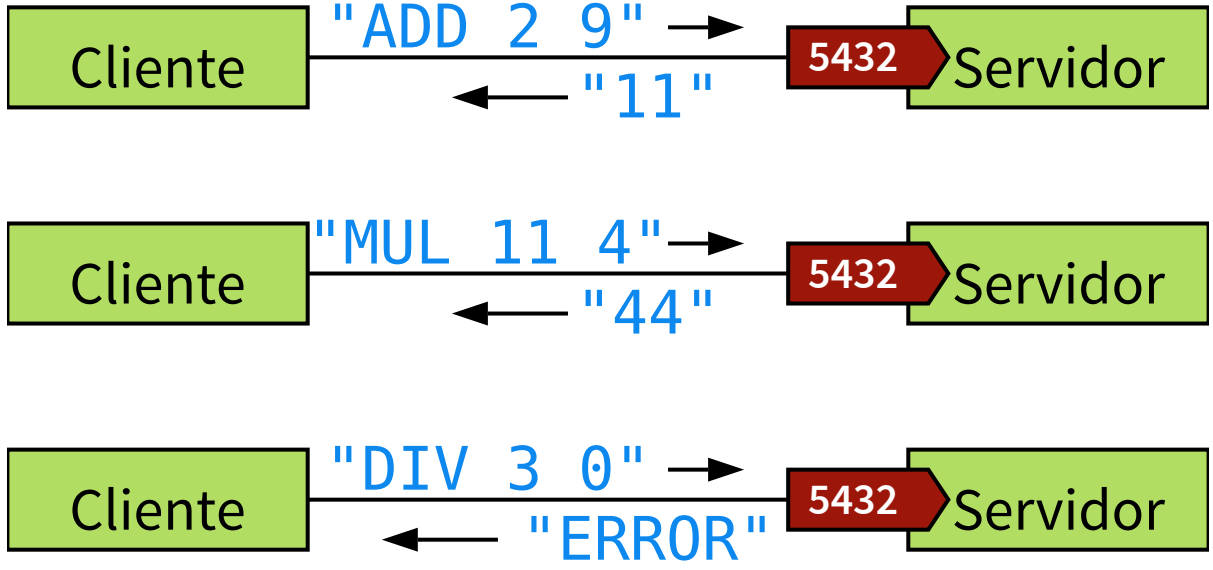
EXTENSIÓN DEL SERVIDOR

Modificamos el servidor para que pueda realizar otro tipo de operaciones aritméticas.

El servidor recibirá mensajes de la forma:

- `ADD` *n1 n2*
- `SUB` *n1 n2*
- `MUL` *n1 n2*
- `DIV` *n1 n2*
- `SQRT` *n1*

EJEMPLOS


Cliente
"ADD 2 9"
"11"
5432
Servidor
Cliente
"MUL 11 4"
"44"
5432
Servidor
Cliente
"DIV 3 0"
"ERROR"
5432
Servidor

CÓDIGO DEL SERVIDOR

```
private static void servePetition(Socket socket) {
  try (Scanner sc = new Scanner(socket.getInputStream());
       PrintWriter out = new PrintWriter(
          new OutputStreamWriter(socket.getOutputStream())
       )) {
    String op = sc.next();
    switch (op) {
      case "ADD"  : serveAddition(sc, out);       break;
      case "SUB"  : serveSubstraction(sc, out);   break;
      case "MUL"  : serveMultiplication(sc, out); break;
      case "DIV"  : serveDivision(sc, out);       break;
      case "SQRT" : serveSquareRoot(sc, out);     break;
      default     : out.println("ERROR");
    }
  } catch (IOException e) {
    System.err.println("Error al obtener el flujo de entrada: " + e.getMessage());
  }
}
```

EJEMPLO: DIVISIÓN

```
private static void serveDivision(Scanner sc, PrintWriter out) {
  int n1 = sc.nextInt();
  int n2 = sc.nextInt();
  if (n2 != 0) {
    System.out.println("Dividiendo " + n1 + " entre " + n2);
    out.println((double)n1 / n2);
  } else {
    out.println("ERROR");
  }
}
```

PETICIÓN DEL CLIENTE

```
public static void main(String... args) throws IOException {
    Socket s = new Socket("localhost", 5432);
    BufferedWriter bw = new BufferedWriter(
          new OutputStreamWriter(s.getOutputStream()));
    bw.write("MUL 4 12\n");
    bw.flush();
    BufferedReader br = new BufferedReader(
          new InputStreamReader(s.getInputStream()));
    System.out.println(br.readLine());
    bw.close();
    br.close();
}
```

Resultado:

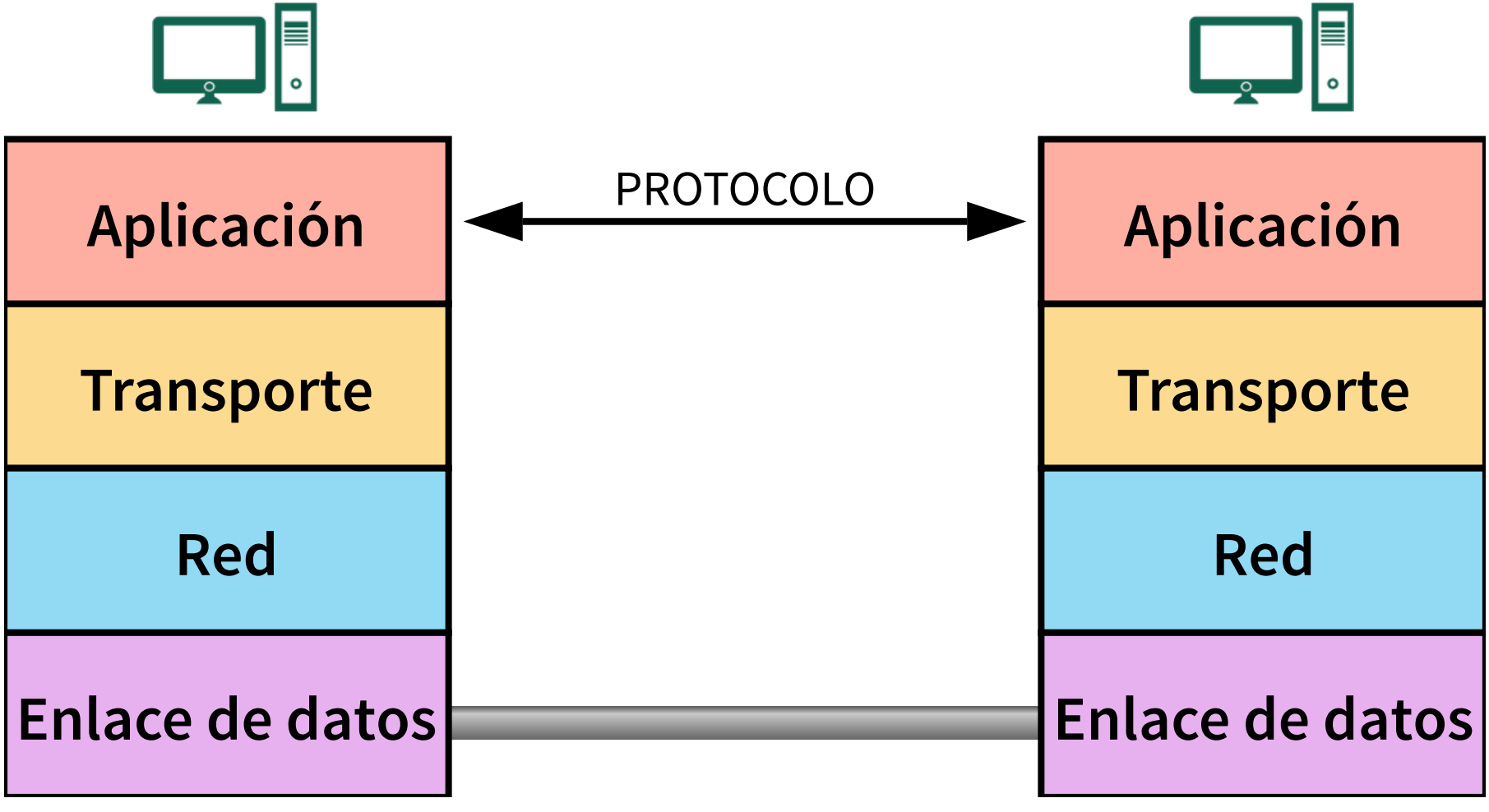
48

RESUMEN: COMUNICACIÓN CON EL SERVIDOR

Petición	Respuesta
`ADD` *n1 n2*	*n1* + *n2*
`SUB` *n1 n2*	*n1* - *n2*
`MUL` *n1 n2*	*n1* * *n2*
`DIV` *n1 n2*	*n1* / *n2* (si n1 ≠ 0) `ERROR` (e.o.c.)
`SQRT` *n*	$\sqrt{n}$ (si n ≥ 0) `ERROR` (e.o.c.)

Esta tabla describe un **protocolo**

Hemos implementado la *capa de aplicación* de la calculadora.



EXPERIMENTO

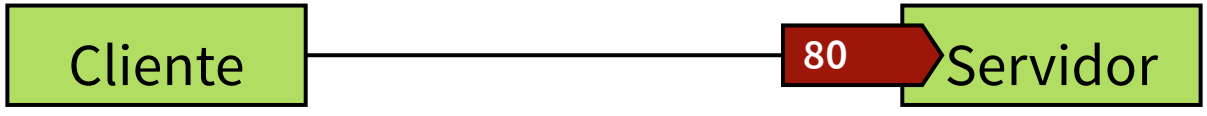
Conectamos con el servidor de la FdI en el puerto 80:

```
public static void main(String... args) throws IOException {
  Socket s = new Socket("informatica.ucm.es", 80);
  PrintWriter pw = new PrintWriter(new OutputStreamWriter(s.getOutputStream()));

  // Petición
  pw.println("GET / HTTP/1.0");
  pw.println("Host: informatica.ucm.es\n");
  pw.flush();

  // Respuesta
  Scanner sc = new Scanner(s.getInputStream());
  while (sc.hasNextLine()) {
    System.out.println(sc.nextLine());
  }
}
```

Respuesta obtenida:

```
HTTP/1.1 200 OK
Date: Wed, 10 Aug 2016 16:40:14 GMT
Server: Apache
Expires: Thu, 19 Nov 1981 08:52:00 GMT
Cache-Control: no-store, no-cache, must-revalidate, post-check=0,
               pre-check=0
Pragma: no-cache
Content-Type: text/html; charset=UTF-8
Set-Cookie: ucmweb=7rbkr7h03pld7rkugaod7u28a6; path=/;
            domain=ucm.es
Connection: close

<!DOCTYPE html>
<html lang="es">
<head>
        <title>Facultad de Informática. Universidad Complutense de
  ...
```

APLICACIONES WEB

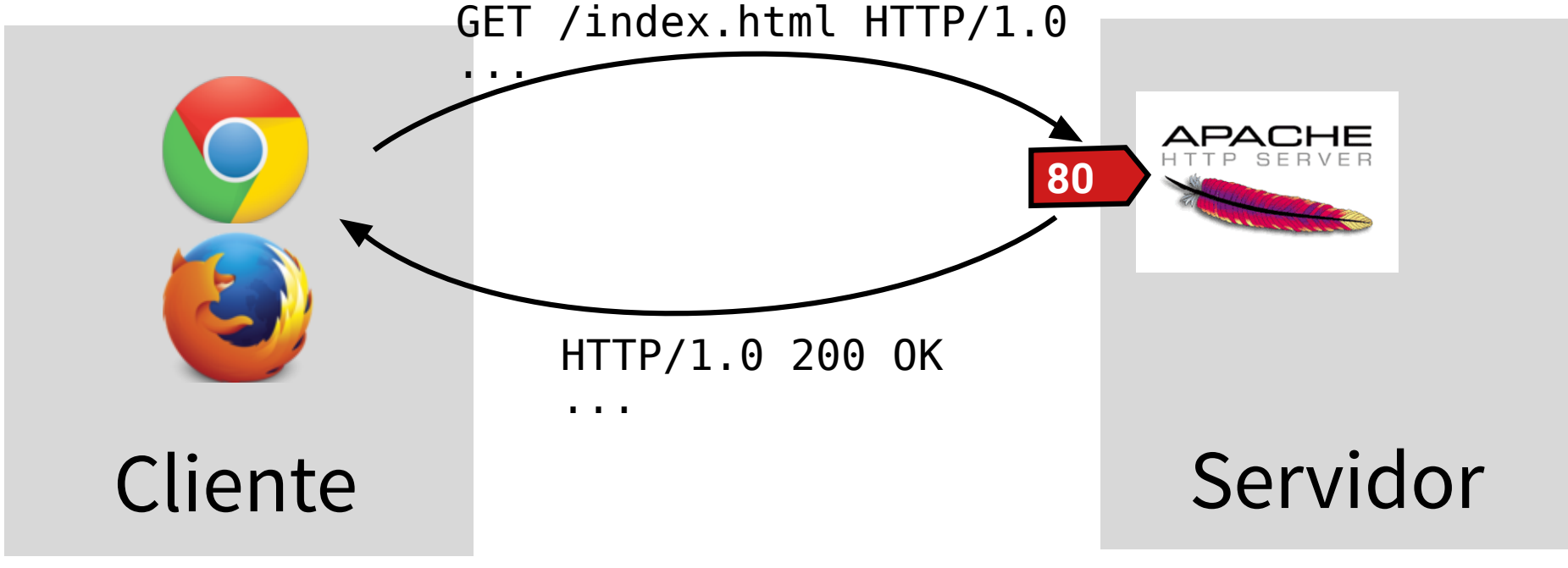


Las aplicaciones web también utilizan el modelo cliente-servidor:

- **Cliente:** Navegador web.
  Firefox, Chrome, Opera, etc.
- **Servidor:** Servidor web.
  Apache HTTP server, Tomcat, Nginx, etc.
- **Protocolo:** HTTP

1. REPASO DEL MODELO TCP/IP
2. MODELO CLIENTE/SERVIDOR
3. SOCKETS Y PROTOCOLOS
4. PROTOCOLO HTTP
5. REFERENCIAS

ESQUEMA GENERAL



ESTRUCTURA DE UNA PETICIÓN HTTP

```
GET /register.html HTTP/1.1
Host: www.foo.com
Accept: text/html
Accept-Language: en-US, es-ES
Connection: keep-alive
If-Modified-Since: Fri, 26 Feb 2016 03:09:05 GMT
                    (línea en blanco)
name=Francisco%20Sanchez&age=12
...
```

Línea de petición

Cabeceras

Cuerpo de la petición (opcional)

LÍNEA DE PETICIÓN

Tiene el siguiente formato:

```
[método] [recurso] [versión_HTTP]
```

- `[método]`: Acción a realizar (GET, POST, HEAD, PUT, ...)
- `[recurso]`: Nombre del recurso (URI) sobre el que se accederá.
- `[versión_HTTP]`: HTTP/1.0 o HTTP/1.1.

Ejemplo:

```
GET /index.html HTTP/1.1
```

ALGUNOS MÉTODOS DISPONIBLES

- `GET`: Obtener un recurso del servidor.
- `HEAD`: Igual que GET, pero el servidor no devolverá el recurso; sólo las cabeceras de la respuesta.
- `POST`: Enviar información al servidor.
- `PUT`: Guardar información en el servidor.
- `DELETE`: Eliminar información del servidor.
- `OPTIONS`: Obtener la lista de métodos soportados por el servidor.

RECURSOS HTTP

```
[método] [recurso] [versión_HTTP]
```

El nombre de recurso tiene la aparencia de un nombre de fichero, pero **no tiene por qué corresponder con un fichero físico almacenado en el servidor**

Un servidor web puede hace corresponder las URIs con llamadas a programas (por ejemplo, servlets), accesos a recursos físicos almacenados con un nombre distinto, etc.

CABECERAS EN UNA PETICIÓN

Especifican información adicional en una petición

```
Host: www.foo.com
Accept: text/html
Accept-Language: en-US, es-ES
Connection: keep-alive
If-Modified-Since: Fri, 26 Feb 2016 03:09:05 GMT
```

ALGUNOS EJEMPLOS DE CABECERAS

- `Host: www.ucm.es`
  Obligatorio en HTTP/1.1. Indica el servidor virtual al que acceder.
- `Accept: text/html, image/gif, */*`
  Tipos de contenido (MIME types) que se espera recibir (Lista de tipos MIME)
- `Accept-Language: en-US`
  Idioma(s) en los que se espera recibir el recurso (Idiomas y Regiones)
- `Accept-Charset: UTF-8`
  `Accept-Encoding: x-gzip`
  Codificación de caracteres.

ALGUNOS EJEMPLOS DE CABECERAS

- `User-Agent: Mozilla/5.0 ...`
  Navegador o cliente que envía la petición.
- `Content-Type: text/html`
  En peticiones POST, indica el tipo de contenido que se adjunta en el cuerpo de la petición.
- `Authorization: Basic bWFudWVsOm1hbnVlbA==`
  Nombre de usuario y contraseña (para acceso a recursos protegidos por contraseña).
- `If-Modified-Since: Fri, 26 Feb 2016 03:09:05 GMT`
  Indica al servidor que sólo devuelva el recurso si ha sido modificado después de la fecha dada.

ESTRUCTURA DE UNA RESPUESTA HTTP

```
HTTP/1.1 200 OK
Date: Mon, 15 Aug 2016 14:05:30 GMT
Server: Apache
Set-Cookie: ucmweb=a8bcbkqfml72df71mee86drgj7; domain=ucm.es
Expires: Thu, 19 Nov 1981 08:52:00 GMT
Content-Type: text/html; charset=utf-8
(línea en blanco)
<!DOCTYPE html>
<html lang="es">
<head>
  <title>UCM-Universidad Complutense de Madrid</title>
...
```

Línea de estado

Cabeceras de la respuesta

Cuerpo de la respuesta

LÍNEA DE ESTADO

```
HTTP/1.1 200 OK
```

Indica la versión del protocolo HTTP utilizada, un código de estado y un texto con la descripción dicho código.

CÓDIGOS DE ESTADO

- `200 OK`: Petición recibida y servida con éxito.
- `3xx`: Redirección: el recurso ha cambiado de dirección.
  - `301 Move Permanently`
  - `302 Move Temporarily`
  - `304 Not modified`

En los códigos 301 y 302 se indica en la cabecera de la respuesta la nueva dirección del recurso. El código 304 se devuelve en el caso en el que el cliente haya incluido `If-Modified-Since` y el recurso no haya cambiado desde la fecha indicada.

CÓDIGOS DE ESTADO

- `4xx`: Errores atribuibles al cliente.
  - `400 Bad Request`: Petición incorrecta.
  - `401 Authentication Required`: Se requiere identificación (nombre y contraseña).
  - `403 Forbidden`: Identificación incorrecta.
  - `404 Not Found`: Recurso no encontrado en el servidor.
  - `405 Method not Allowed`: Acción (GET, POST, etc.) no permitida.
- `5xx`: Errores atribuibles al servidor.
  - `500 Internal Server Error`
  - `503 Service Unavailable`

CUERPO DE LA RESPUESTA

```
<!DOCTYPE html>
<html lang="es">
<head>
  <title>UCM-Universidad Complutense de Madrid</title>
...
```

Tal y como indica la cabecera `Content-Type: text/html`, el cuerpo de la respuesta contiene un texto en formato HTML que define la estructura de la página web recuperada.

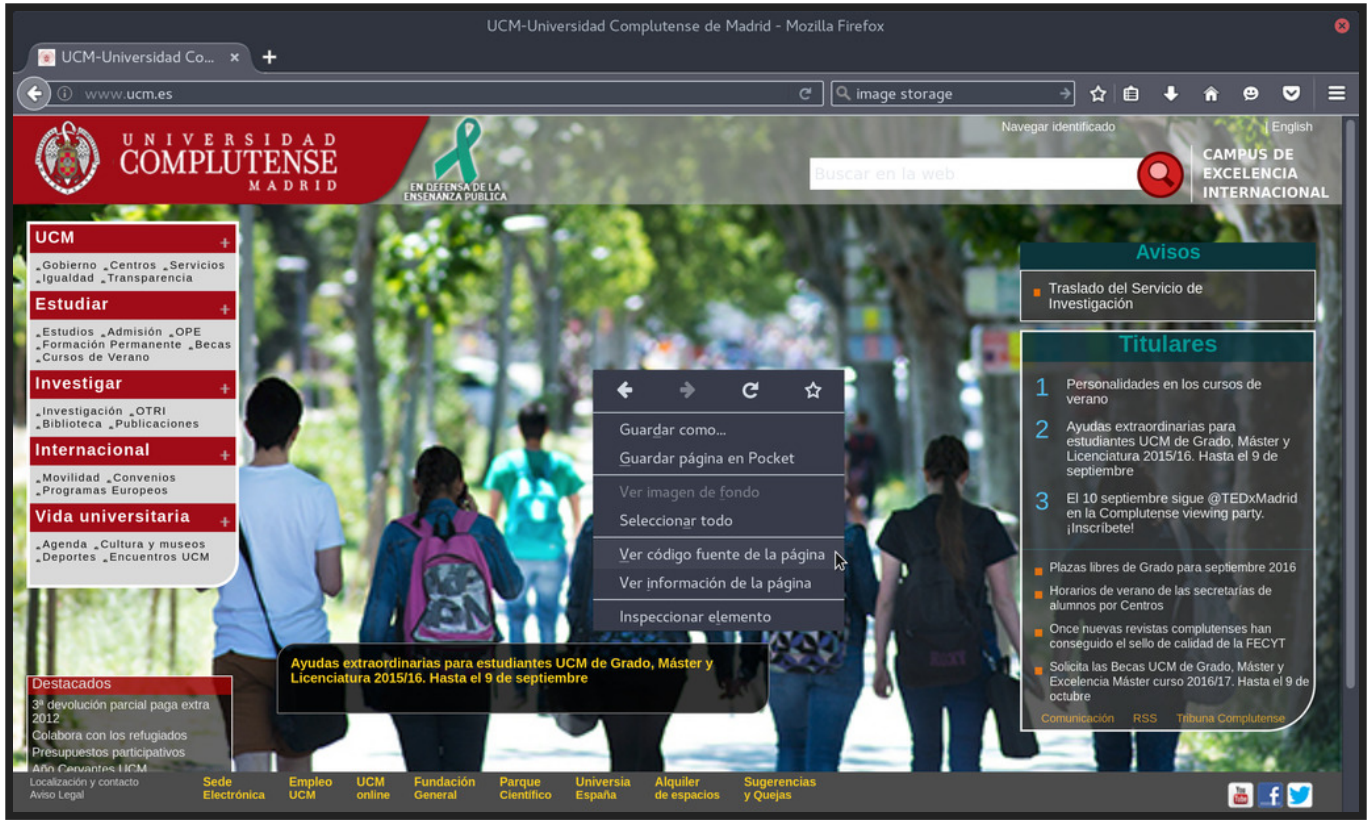
El código HTML puede hacer referencia a otros recursos:

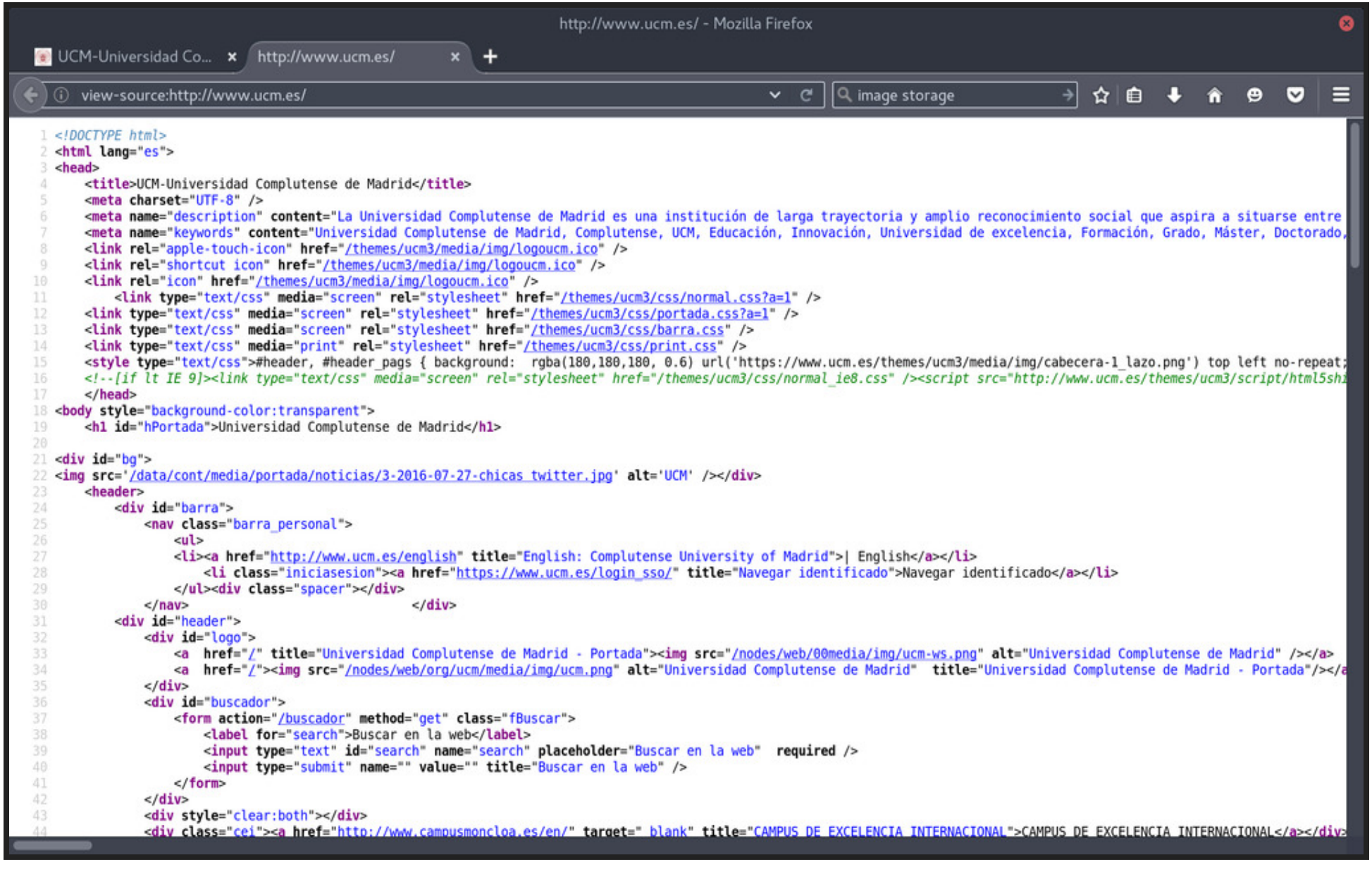
```
...
<link type="text/css" media="screen" rel="stylesheet"
href="/themes/ucm3/css/portada.css?a=1" />
...
<img src="/nodes/web/00media/img/ucm-ws.png"/>
...
```

Cada uno de estos recursos deberá ser obtenido por el navegador Web en sucesivas peticiones HTTP al servidor.

VISUALIZAR CÓDIGO HTML EN FIREFOX Y CHROME

- Botón derecho → *Ver código fuente*, o bien,
- Ctrl+U






```
<!DOCTYPE html>
<html lang="es">
<head>
    <title>UCM-Universidad Complutense de Madrid</title>
    <meta charset="UTF-8" />
    <meta name="description" content="La Universidad Complutense de Madrid es una institución de larga trayectoria y amplio reconocimiento social que aspira a situarse entre
    <meta name="keywords" content="Universidad Complutense de Madrid, Complutense, UCM, Educación, Innovación, Universidad de excelencia, Formación, Grado, Máster, Doctorado,
    <link rel="apple-touch-icon" href="/themes/ucm3/media/img/logoucm.ico" />
    <link rel="shortcut icon" href="/themes/ucm3/media/img/logoucm.ico" />
    <link rel="icon" href="/themes/ucm3/media/img/logoucm.ico" />
        <link type="text/css" media="screen" rel="stylesheet" href="/themes/ucm3/css/normal.css?a=1" />
    <link type="text/css" media="screen" rel="stylesheet" href="/themes/ucm3/css/portada.css?a=1" />
    <link type="text/css" media="screen" rel="stylesheet" href="/themes/ucm3/css/barra.css" />
    <link type="text/css" media="print" rel="stylesheet" href="/themes/ucm3/css/print.css" />
    <style type="text/css">#header, #header_pags { background:  rgba(180,180,180, 0.6) url('https://www.ucm.es/themes/ucm3/media/img/cabecera-1_lazo.png') top left no-repeat;
    <!--[if lt IE 9]><link type="text/css" media="screen" rel="stylesheet" href="/themes/ucm3/css/normal_ie8.css" /><script src="http://www.ucm.es/themes/ucm3/script/html5shi
    </head>
<body style="background-color:transparent">
    <h1 id="hPortada">Universidad Complutense de Madrid</h1>

<div id="bg">
<img src='/data/cont/media/portada/noticias/3-2016-07-27-chicas_twitter.jpg' alt='UCM' /></div>
    <header>
        <div id="barra">
            <nav class="barra_personal">
                <ul>
                <li><a href="http://www.ucm.es/english" title="English: Complutense University of Madrid">| English</a></li>
                    <li class="iniciasesion"><a href="https://www.ucm.es/login_sso/" title="Navegar identificado">Navegar identificado</a></li>
                </ul><div class="spacer"></div>
            </nav>                                  </div>
        <div id="header">
            <div id="logo">
                <a  href="/" title="Universidad Complutense de Madrid - Portada"><img src="/nodes/web/00media/img/ucm-ws.png" alt="Universidad Complutense de Madrid" /></a>
                <a  href="/"><img src="/nodes/web/org/ucm/media/img/ucm.png" alt="Universidad Complutense de Madrid"  title="Universidad Complutense de Madrid - Portada"/></a
            </div>
            <div id="buscador">
                <form action="/buscador" method="get" class="fBuscar">
                    <label for="search">Buscar en la web</label>
                    <input type="text" id="search" name="search" placeholder="Buscar en la web"  required />
                    <input type="submit" name="" value="" title="Buscar en la web" />
                </form>
            </div>
            <div style="clear:both"></div>
            <div class="cei"><a href="http://www.campusmoncloa.es/en/" target="_blank" title="CAMPUS DE EXCELENCIA INTERNACIONAL">CAMPUS DE EXCELENCIA INTERNACIONAL</a></div>
```

OBSERVAR CABECERAS HTTP EN FIREFOX

Herramientas → *Desarrollador Web* → *Red* (Ctrl+Mayus+Q)

https://youtu.be/RaeyLZE6Yo0

OBSERVAR CABECERAS HTTP EN CHROME

Menú *Más Herramientas* → *Herramientas para desarrolladores* (Ctrl+Mayus+I)

https://youtu.be/B4vQHymBP1A

1. REPASO DEL MODELO TCP/IP
2. MODELO CLIENTE/SERVIDOR
3. SOCKETS Y PROTOCOLOS
4. PROTOCOLO HTTP
5. REFERENCIAS

REFERENCIAS

- Wikipedia - *Internet Protocol Suite*
  https://en.wikipedia.org/wiki/Internet_protocol_suite
- *An Introduction to HTTP Basics*
  https://www.ntu.edu.sg/home/ehchua/programming/webprogramming/HTTP_Basics.html