


Universidad de Alcalá

Departamento de Automática

# Capítulo 2 La capa de aplicación



**A note on the use of these ppt slides:**  
We're making these slides freely available to all (faculty, students, readers). They're in PowerPoint form so you can add, modify, and delete slides (including this one) and slide content to suit your needs. They obviously represent a lot of work on our part. In return for use, we only ask the following:  
 • If you use these slides (e.g., in a class) in substantially unaltered form, that you mention their source (after all, we'd like people to use our book!)  
 • If you post any slides in substantially unaltered form on a www site, that you note that they are adapted from (or perhaps identical to) our slides, and note our copyright of this material.

Thanks and enjoy! JFK/KWR  
All material copyright 1996-2010  
J.F. Kurose and K.W. Ross. All Rights Reserved

*Redes de computadoras: Un enfoque descendente, 5ª edición, Jim Kurose, Keith Ross, Pearson Educación, 2010.*

Raúl Durán, Nacho Pérez, v1.0 Capo de Aplicación 1

Universidad de Alcalá

Departamento de Automática

# Capítulo 2: La capa de aplicación

2.1 Principios de las aplicaciones en red	2.6 Aplicaciones P2P
2.2 Web y HTTP	2.7 Programación de sockets con TCP
2.3 FTP	2.8 Programación de sockets con UDP
2.4 Correo electrónico	
▪ SMTP, POP3, IMAP	
2.5 DNS	

Raúl Durán, Nacho Pérez, v1.0 Capo de Aplicación 2

Universidad de Alcalá

Departamento de Automática

# Capítulo 2: La capa de aplicación

**Objetivos:**

- ❖ aspectos conceptuales y de implementación de los protocolos de aplicaciones en red
  - modelos de servicio de la capa de transporte
  - paradigma cliente-servidor
  - paradigma P2P ('peer-to-peer')
- ❖ conocer los protocolos examinando los más usados de la capa de aplicación
  - HTTP
  - FTP
  - SMTP / POP3 / IMAP
  - DNS
- ❖ programación de aplicaciones en red
  - API de los sockets

Raúl Durán, Nacho Pérez, v1.0 Capo de Aplicación 3

Universidad de Alcalá

Departamento de Automática

# Algunas aplicaciones de red

- ❖ e-mail
- ❖ web
- ❖ mensajería instantánea
- ❖ login remoto
- ❖ compartición de archivos P2P
- ❖ juegos multiusuario en red
- ❖ streaming de video (YouTube)
- ❖ voz sobre IP
- ❖ videoconferencia en tiempo real
- ❖ *cloud computing*
- ❖ ...
- ❖ ...
- ❖ ...

Raúl Durán, Nacho Pérez, v1.0 Capo de Aplicación 4

Universidad de Alcalá

Departamento de Automática

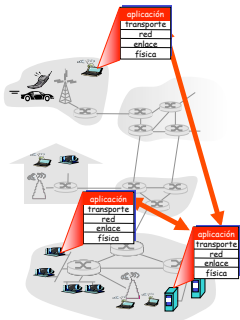
# Creación de una aplicación de red

**programas que**

- corren en (diferentes) sistemas terminales
- se comunican por la red
- p.ej.: servidor web se comunica con el navegador

**No hace falta escribir el código de los dispositivos del núcleo de la red**

- éstos no corren la aplicación del usuario
- el desarrollo de las aplicaciones y su propagación es más rápido



Raúl Durán, Nacho Pérez, v1.0 Capo de Aplicación 5

Universidad de Alcalá

Departamento de Automática

# Capítulo 2: La capa de aplicación

2.1 Principios de las aplicaciones en red	2.6 Aplicaciones P2P
2.2 Web y HTTP	2.7 Programación de sockets con TCP
2.3 FTP	2.8 Programación de sockets con UDP
2.4 Correo electrónico	
▪ SMTP, POP3, IMAP	
2.5 DNS	

Raúl Durán, Nacho Pérez, v1.0 Capo de Aplicación 6

Universidad de Alcalá | Departamento de Automática

## Arquitecturas de las aplicaciones en red

- ❖ cliente-servidor
- ❖ 'peer-to-peer' (P2P)
- ❖ híbrido de cliente-servidor y P2P

Raúl Durán, Nacho Pérez. v1.0 | Copo de Aplicación 7

Universidad de Alcalá | Departamento de Automática

## Arquitectura cliente-servidor



**servidor:**

- host siempre activo
- IP permanente
- clusters para mejor escalabilidad

**clientes:**

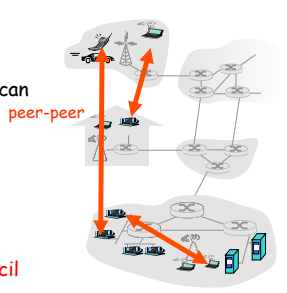
- se comunican con el servidor
- conexión intermitente
- pueden tener IPs dinámicas
- no se comunican entre sí

Raúl Durán, Nacho Pérez. v1.0 | Copo de Aplicación 8

Universidad de Alcalá | Departamento de Automática

## Arquitectura P2P pura

- ❖ no hay un servidor siempre activo
- ❖ sistemas terminales arbitrarios se comunican directamente
- ❖ los pares ('peers') se conectan intermitentemente y cambian de IP



**peer-peer**

muy escalable, pero difícil de gestionar

Raúl Durán, Nacho Pérez. v1.0 | Copo de Aplicación 9

Universidad de Alcalá | Departamento de Automática

## Híbrido de C-S y P2P

**Skype**

- aplicación P2P de voz-sobre-IP
- servidor central: encuentra la dirección de la parte remota
- la conexión cliente-cliente es directa (no a través del servidor)

**Mensajería instantánea**

- el chat entre usuarios es P2P
- centralizado: detección/localización de la presencia del cliente
  - el usuario registra su IP en el servidor central cuando se conecta
  - el usuario contacta con el servidor central para averiguar las IPs de sus contactos

Raúl Durán, Nacho Pérez. v1.0 | Copo de Aplicación 10

Universidad de Alcalá | Departamento de Automática

## Comunicación de procesos

**proceso:** programa que corre en un host

- ❖ en el mismo host, dos procesos se comunican por sistemas definidos por el sistema operativo (**inter-process communication**)
- ❖ los procesos en hosts diferentes se comunican por **mensajes**.

**proceso cliente:** el que inicia la comunicación

**proceso servidor:** el que espera a que contacten con él

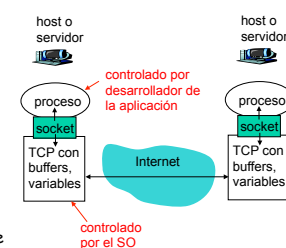
- ❖ aparte: las aplicaciones con arquitectura P2P tienen procesos cliente y procesos servidor

Raúl Durán, Nacho Pérez. v1.0 | Copo de Aplicación 11

Universidad de Alcalá | Departamento de Automática

## Sockets

- ❖ un proceso envía/recibe mensajes a través de/ desde su **socket**
- ❖ un socket es análogo a una puerta
  - el emisor "empuja" el mensaje hacia afuera
  - el emisor cuenta con que la infraestructura de transporte al otro lado de la puerta llevará el mensaje al socket del receptor
- ❖ API: (1) elección del protocolo de transporte; (2) capacidad de fijar algún parámetro (mucho más sobre **esto más adelante**)



host o servidor | host o servidor

proceso | proceso

socket | socket

Internet

TCP con buffers, variables | TCP con buffers, variables

controlado por desarrollador de la aplicación

controlado por el SO

Raúl Durán, Nacho Pérez. v1.0 | Copo de Aplicación 12

Universidad de Alcalá | Departamento de Automática

## Direccionar procesos

- ❖ para recibir mensajes, un proceso debe tener un identificador
- ❖ el host tiene una IP de 32 bits única
- ❖ **P:** ¿es suficiente la IP del host en el que corre el proceso para identificar el proceso?

Raúl Durán, Nacho Pérez. v1.0 | Copo de Aplicación 13

Universidad de Alcalá | Departamento de Automática

## Direccionar procesos

- ❖ para recibir mensajes, un proceso debe tener un identificador
- ❖ el host tiene una IP de 32 bits única
- ❖ **P:** ¿es suficiente la IP del host en el que corre el proceso para identificar el proceso?
  - **R:** No, más de un proceso puede estar en ejecución en el mismo host
- ❖ el **identificador** incluye tanto la **IP** como el **número de puerto** asociados con el proceso en el host
- ❖ ejemplos:
  - servidor HTTP: 80
  - servidor e-mail: 25
- ❖ para enviar mensaje HTTP al servidor web `gaia.cs.umass.edu` web server:
  - **IP:** 128.119.245.12
  - **Número de puerto:** 80
- ❖ **continúa...**

Raúl Durán, Nacho Pérez. v1.0 | Copo de Aplicación 14

Universidad de Alcalá | Departamento de Automática

## El protocolo de la capa de aplicación define

- ❖ tipos de mensajes intercambiados,
  - p.ej.: petición, respuesta
- ❖ sintaxis de los mensajes:
  - qué campos en qué mensajes y cómo se definen los campos
- ❖ semántica de los mensajes
  - significado de la información contenida en los campos
- ❖ reglas de envío y respuesta de mensajes (cuándo, cómo)

**protocolos de dominio público:**

- ❖ definidos en los RFCs
- ❖ permiten la interoperabilidad
- ❖ p.ej.: HTTP, SMTP

**protocolos propietarios:**

- ❖ p.ej.: Skype

Raúl Durán, Nacho Pérez. v1.0 | Copo de Aplicación 15

Universidad de Alcalá | Departamento de Automática

## ¿Qué servicio de transporte necesita una aplicación?

**Pérdidas de datos**

- ❖ algunas aplicaciones pueden tolerar cierta pérdida (p.ej.: audio)
- ❖ otras necesitan una transferencia 100% fiable (p.ej.: transferencia de archivos, telnet)

**Temporización**

- ❖ algunas aplicaciones (p.ej.: telefonía por Internet, juegos interactivos) requieren bajo retardo para ser "efectivas"

**Tasa de transferencia**

- ❖ algunas aplicaciones requieren una tasa mínima para ser "efectivas" (p.ej.: multimedia)
- ❖ otras aplicaciones ("aplicaciones elásticas") se apañan con la que puedan conseguir.

**Seguridad**

- ❖ encriptación, integridad de datos, ...

Raúl Durán, Nacho Pérez. v1.0 | Copo de Aplicación 16

Universidad de Alcalá | Departamento de Automática

## Requisitos de transporte de algunas aplicaciones

Aplicación	Pérdida	Ancho de banda	Sensible al tiempo
transf. archivos	sin pérdidas	elástica	no
e-mail	sin pérdidas	elástica	no
Docs. web	sin pérdidas	elástica	no
audio/video tiempo real	tolerante	audio: 5kbps-1Mbps video: 10kbps-5Mbps	sí, décimas
audio/video almac.	tolerante	como el anterior	sí, segundos
juegos interactivos	tolerante	desde pocos kbps	sí, décimas
mens. instantánea	sin pérdidas	elástica	sí y no

Raúl Durán, Nacho Pérez. v1.0 | Copo de Aplicación 17

Universidad de Alcalá | Departamento de Automática

## Servicios de los protocolos de transporte de internet

**Servicio TCP:**

- ❖ **orientado a conexión:** requiere negociación entre cliente y servidor
- ❖ **transferencia fiable** entre emisor y receptor
- ❖ **control de flujo:** que el emisor no sature al receptor
- ❖ **control de congestión:** regula el emisor cuando la red se satura
- ❖ **no proporciona:** temporización, ancho de banda garantizado, seguridad

**Servicio UDP:**

- ❖ transferencias no fiables entre emisor y receptor
- ❖ no proporciona: negociación, fiabilidad, control de flujo, control de congestión, temporización, ancho de banda garantizado, seguridad.
- ❖ **P:** ¿qué aporta? ¿por qué hay UDP?

Raúl Durán, Nacho Pérez. v1.0 | Copo de Aplicación 18

Universidad de Alcalá | Departamento de Automática

### protocolos de aplicación y transporte de aplicaciones comunes

Aplicación	Protocolo de aplicación	Protocolo de transporte
e-mail	SMTP [RFC 2821]	TCP
terminal remoto	Telnet [RFC 854]	TCP
Web	HTTP [RFC 2616]	TCP
transf. archivos	FTP [RFC 959]	TCP
streaming multimedia	HTTP (p.ej.: YouTube), RTP [RFC 1889]	TCP o UDP
telefonía Internet	SIP, RTP, propietario (p.ej.: Skype)	típicamente UDP

Raúl Durán, Nacho Pérez. v1.0 | Copia de Aplicación 19

Universidad de Alcalá | Departamento de Automática

## Capítulo 2: La capa de aplicación

<p>2.1 Principios de las aplicaciones en red</p> <p><b>2.2 Web y HTTP</b></p> <p>2.3 FTP</p> <p>2.4 Correo electrónico</p> <ul style="list-style-type: none"> <li>SMTP, POP3, IMAP</li> </ul> <p>2.5 DNS</p>	<p>2.6 Aplicaciones P2P</p> <p>2.7 Programación de sockets con TCP</p> <p>2.8 Programación de sockets con UDP</p>
--	---

Raúl Durán, Nacho Pérez. v1.0 | Copia de Aplicación 20

Universidad de Alcalá | Departamento de Automática

## Web y HTTP

Primero, repaso...

- ❖ una página web consta de objetos
- ❖ un objeto puede ser un archivo HTML, una imagen JPEG, un applet Java, un archivo de audio...
- ❖ una página web consta de un **archivo HTML base**, que incluye objetos referenciados
- ❖ cada objeto es direccionable por una **URL**
- ❖ URL ejemplo:

`www.someschool.edu/someDept/pic.gif`  
 nombre del host     ruta

Raúl Durán, Nacho Pérez. v1.0 | Copia de Aplicación 21

Universidad de Alcalá | Departamento de Automática

## Visión general de HTTP

**HTTP: hypertext transfer protocol**

- ❖ protocolo de la capa de aplicación para la Web
- ❖ modelo cliente/servidor
  - **cliente:** navegador que pide, recibe y muestra objetos Web
  - **servidor:** envía objetos en respuesta a las peticiones

Raúl Durán, Nacho Pérez. v1.0 | Copia de Aplicación 22

Universidad de Alcalá | Departamento de Automática

## Visión general de HTTP (cont.)

<p><b>Usa TCP:</b></p> <ul style="list-style-type: none"> <li>❖ el cliente inicia conexión TCP (crea el socket) con el servidor, puerto 80</li> <li>❖ el servidor acepta la conexión TCP del cliente</li> <li>❖ se intercambian mensajes HTTP (capa de aplicación) entre el navegador (cliente HTTP) y el servidor Web (servidor HTTP)</li> <li>❖ Se cierra la conexión TCP</li> </ul>	<p><b>HTTP es "stateless" (sin estado)</b></p> <ul style="list-style-type: none"> <li>❖ el servidor no guarda información sobre las peticiones de los clientes</li> </ul> <p style="text-align: center;"><b>por cierto...</b></p> <p style="border: 1px solid black; padding: 5px; display: inline-block;"><b>los protocolos que guardan el estado son complejos</b></p> <ul style="list-style-type: none"> <li>❖ se debe guardar la historia (el estado)</li> <li>❖ si se cuelga el servidor o el cliente, sus "estados" pueden ser incongruentes, deben ser unificados.</li> </ul>
--	--

Raúl Durán, Nacho Pérez. v1.0 | Copia de Aplicación 23

Universidad de Alcalá | Departamento de Automática

## conexiones HTTP

<p><b>HTTP no persistente</b></p> <ul style="list-style-type: none"> <li>❖ como máximo, 1 objeto enviado a través de una conexión TCP.</li> </ul>	<p><b>HTTP persistente</b></p> <ul style="list-style-type: none"> <li>❖ múltiples objetos se pueden enviar a través de una conexión TCP única entre cliente y servidor.</li> </ul>
---	--

Raúl Durán, Nacho Pérez. v1.0 | Copia de Aplicación 24

Universidad de Alcalá | Departamento de Automática

## HTTP no persistente

un usuario usa como URL: `www.someSchool.edu/someDepartment/home.index` (contiene texto y referencias a 10 imágenes jpeg)

- 1a. El cliente HTTP inicia conexión TCP con el servidor HTTP en `www.someSchool.edu` en el puerto 80
- 1b. El servidor HTTP en el host `www.someSchool.edu` en espera de conexión TCP en el puerto 80, acepta la conexión, notificándolo al cliente
2. el cliente HTTP envía *mensaje de petición* (con la URL) en el socket TCP. El mensaje indica que el cliente quiere el objeto `someDepartment/home.index`
3. El servidor HTTP recibe mensaje de petición, crea el *mensaje de respuesta*, que contiene el objeto pedido, y lo envía a su socket.

tiempo

Raúl Durán, Nacho Pérez. v1.0 | Copo de Aplicación 25

Universidad de Alcalá | Departamento de Automática

## HTTP no persistente (cont.)

4. El servidor HTTP cierra la conexión TCP.
5. El cliente HTTP recibe el mensaje de respuesta con el archivo html. Lo muestra. Al analizarlo, descubre 10 referencias a objetos jpeg.
6. Se repiten los pasos 1-5 para cada uno de los 10 objetos jpeg.

tiempo

Raúl Durán, Nacho Pérez. v1.0 | Copo de Aplicación 26

Universidad de Alcalá | Departamento de Automática

## HTTP no persistente: Tiempo de respuesta

**definición de RTT (tiempo de ida y vuelta):** tiempo en el que un pequeño paquete va y vuelve del cliente al servidor

**tiempo de respuesta:**

- ❖ un RTT para empezar la conexión RTT
- ❖ un RTT para la petición HTTP y primeros bits de respuesta
- ❖ tiempo de transmisión del archivo

total = 2RTT + tiempo de transmisión

Raúl Durán, Nacho Pérez. v1.0 | Copo de Aplicación 27

Universidad de Alcalá | Departamento de Automática

## HTTP persistente

**cuestiones del HTTP no persistente:**

- ❖ requiere 2 RTTs por objeto
- ❖ recargo del SO para cada conexión TCP
- ❖ los navegadores a menudo abren conexiones TCP en paralelo para conseguir los objetos referenciados

**HTTP persistente:**

- ❖ el servidor deja la conexión abierta tras enviar la respuesta
- ❖ los mensajes HTTP subsiguientes entre el mismo par cliente/servidor se envían por la conexión abierta
- ❖ el cliente envía peticiones en cuanto encuentra un objeto referenciado
- ❖ un RTT como máximo para todos los objetos referenciados

Raúl Durán, Nacho Pérez. v1.0 | Copo de Aplicación 28

Universidad de Alcalá | Departamento de Automática

## mensaje de solicitud HTTP

- ❖ dos tipos de mensajes HTTP: *solicitud*, *respuesta*
- ❖ **solicitud HTTP:**
  - ASCII

línea de solicitud (comandos GET, POST, HEAD)

cabeceras

retorno de carro + línea nueva al inicio de una línea indica final de las cabeceras

```
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
```

retorno de carro línea nueva

Raúl Durán, Nacho Pérez. v1.0 | Copo de Aplicación 29

Universidad de Alcalá | Departamento de Automática

## solicitud HTTP : formato general

method	sp	URL	sp	version	cr	lf	línea solicitud
header field name	:	value	cr	lf	cabeceras		
...							
header field name	:	value	cr	lf	cuerpo		
cr	lf	Entity Body					

Raúl Durán, Nacho Pérez. v1.0 | Copo de Aplicación 30

Universidad de Alcalá | Departamento de Automática

## envío de información de formularios

**Método POST:**

- la página web a menudo incluye formularios
- la información se envía al servidor en el cuerpo de la entidad

**Método URL:**

- usa el método GET
- la información se envía en el campo de URL de la línea de solicitud:

`www.somesite.com/animalsearch?monkeys&banana`

Raúl Durán, Nacho Pérez, v1.0 | Copia de Aplicación | 31

Universidad de Alcalá | Departamento de Automática

## Tipos de métodos

**HTTP/1.0**

- GET
- POST
- HEAD
  - pide al servidor que no incluya el objeto pedido en la respuesta

**HTTP/1.1**

- GET, POST, HEAD
- PUT
  - "sube" el archivo en el cuerpo de la entidad a la ruta especificada en el campo URL
- DELETE
  - borra el archivo especificado en el campo URL

Raúl Durán, Nacho Pérez, v1.0 | Copia de Aplicación | 32

Universidad de Alcalá | Departamento de Automática

## mensaje de respuesta HTTP

línea de estado (protocolo código estado frase estado)

cabeceras

datos, p.ej.: archivo HTML solicitado

```

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 ...
  
```

Raúl Durán, Nacho Pérez, v1.0 | Copia de Aplicación | 33

Universidad de Alcalá | Departamento de Automática

## códigos de estado de respuesta HTTP

- el código de estado aparece en la 1ª línea en el mensaje de respuesta del servidor.
- ejemplos:

- 200 OK**
  - solicitud con éxito, el objeto solicitado está contenido en el mensaje
- 301 Moved Permanently (trasladado permanentemente)**
  - el objeto solicitado se ha trasladado, la nueva ubicación se incluye en este mensaje (Location:)
- 400 Bad Request (solicitud incorrecta)**
  - el mensaje de solicitud no ha sido entendido por el servidor
- 404 Not Found (no encontrado)**
  - documento solicitado no encontrado en este servidor
- 505 HTTP Version Not Supported**

Raúl Durán, Nacho Pérez, v1.0 | Copia de Aplicación | 34

Universidad de Alcalá | Departamento de Automática

## Prueba de (cliente) por uno mismo

- hacer telnet a un servidor Web:
 

```
telnet atc2.aut.uah.es 80
```

abre conexión TCP al puerto 80 (puerto HTTP por omisión) de atc2.aut.uah.es lo que se escriba se enviará al puerto 80 de atc2.aut.uah.es
- teclea una solicitud GET de HTTP:
 

```
GET /~rduran/ HTTP/1.1
Host: atc2.aut.uah.es
```

al teclear esto (con dos retornos), se envía una solicitud mínima (pero completa!) al servidor HTTP
- observar la respuesta del servidor HTTP! (o usar Wireshark!)

Raúl Durán, Nacho Pérez, v1.0 | Copia de Aplicación | 35

Universidad de Alcalá | Departamento de Automática

## estado del usuario: cookies

muchos sitios Web usan cookies

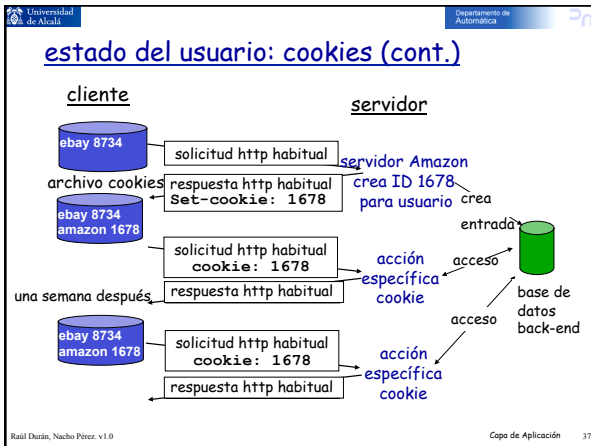
**ejemplo:**

- Susana accede a internet siempre desde su PC
- visita una página de comercio electrónico por primera vez
- con la primera solicitud HTTP, el sitio crea:
  - ID único
  - entrada en la base de datos para el ID

**cuatro componentes:**

- línea de cabecera de la cookie en la respuesta HTTP
- línea de cabecera de la cookie en la solicitud HTTP
- el archivo de cookie está en el host del usuario, gestionado por su navegador
- base de datos de respaldo (back-end) en el sitio Web

Raúl Durán, Nacho Pérez, v1.0 | Copia de Aplicación | 36



**Cookies (cont.)**

**lo que pueden aportar:**

- ❖ autorización
- ❖ carritos de la compra
- ❖ recomendaciones
- ❖ estado de la sesión de usuario (e-mail Web)

**cómo mantener el "estado":**

- ❖ puntos de terminación del protocolo: mantener el estado del emisor/receptor a lo largo de múltiples transacciones
- ❖ cookies: los mensajes http llevan "estado"

**cookies y privacidad:**

- ❖ las cookies permiten a los sitios saber mucho de ti
- ❖ puedes proporcionar nombre y dirección de e-mail a los sitios

por cierto...

Capo de Aplicación 38



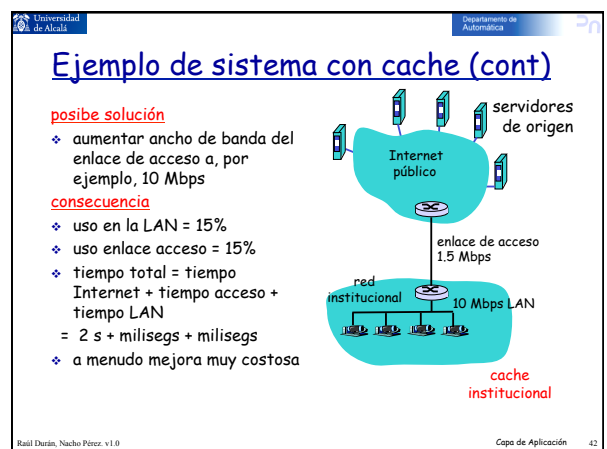
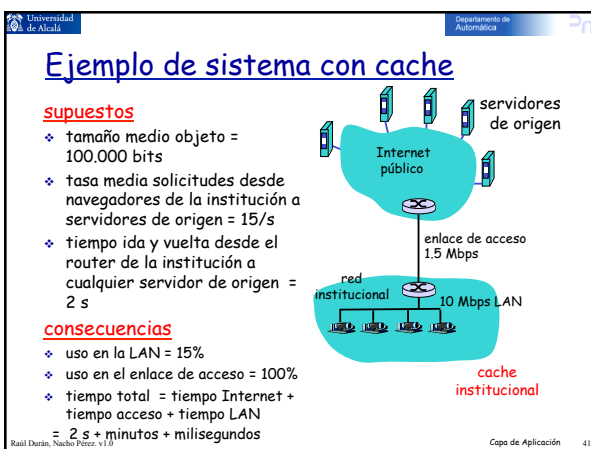
**Más sobre caches web**

- ❖ la cache actúa como cliente y como servidor
- ❖ típicamente instalada por el ISP (universidad, compañía, ISP residencial)

**¿por qué?**

- ❖ reduce tiempo de respuesta
- ❖ reduce tráfico en el enlace de la institución
- ❖ permite a proveedores con pocos recursos distribuir contenidos de modo efectivo (la compartición de archivos P2P también lo hace)

Capo de Aplicación 40





Universidad de Alcalá | Departamento de Automática

## Ejemplo de sistema con cache (cont)

**posible solución:**

- ❖ instalar cache

**consecuencia:**

- ❖ suponer tasa acierto 0,4
  - 40% de solicitudes se sirven casi de inmediato
  - 60% de solicitudes servidas por el servidor de origen
- ❖ uso del enlace de acceso reducido al 60%; resultado, tiempos mínimos (p.ej.: 10 ms)
- ❖ tiempo medio total = tiempo Internet + tiempo acceso + tiempo LAN =  $0,6 \cdot (2,01) s + 0,4 \cdot \text{varios ms} < 1,4 s$

Raúl Darán, Nacho Pérez. v1.0 | Copo de Aplicación 43

Universidad de Alcalá | Departamento de Automática

## GET Condicional

- ❖ **Objetivo:** no enviar objeto si la cache tiene la versión actualizada
- ❖ **cache:** especifica la fecha de la copia en cache en la solicitud  
If-modified-since: <fecha>
- ❖ **servidor:** la respuesta no contiene objeto si la versión en cache está al día:  
HTTP/1.0 304 Not Modified

**cache**      **servidor**

Raúl Darán, Nacho Pérez. v1.0 | Copo de Aplicación 44

Universidad de Alcalá | Departamento de Automática

## Capítulo 2: La capa de aplicación

- 2.1 Principios de las aplicaciones en red
- 2.2 Web y HTTP
- 2.3 FTP
- 2.4 Correo electrónico
  - SMTP, POP3, IMAP
- 2.5 DNS
- 2.6 Aplicaciones P2P
- 2.7 Programación de sockets con TCP
- 2.8 Programación de sockets con UDP

Raúl Darán, Nacho Pérez. v1.0 | Copo de Aplicación 45

Universidad de Alcalá | Departamento de Automática

## FTP: File Transfer Protocol (protocolo de transferencia de archivos)

- ❖ transferir archivo de/hacia host remoto
- ❖ modelo cliente/servidor
  - **cliente:** parte que inicia la transferencia (puede ser hacia o desde el host remoto)
  - **servidor:** host remoto
- ❖ ftp: RFC 959
- ❖ servidor ftp: puerto 21

Raúl Darán, Nacho Pérez. v1.0 | Copo de Aplicación 46

Universidad de Alcalá | Departamento de Automática

## FTP: conexiones paralelas para control y datos

- ❖ El cliente contacta con el servidor en el puerto 21. El protocolo de transporte es TCP
- ❖ se autoriza al cliente en la conexión de control
- ❖ el cliente hace el directorio remoto con comandos a través de la conexión de control
- ❖ al recibir un comando de transferencia de archivo, el servidor abre una 2ª conexión TCP al cliente para el archivo
- ❖ después de transferir un archivo, el servidor cierra la conexión de datos

- ❖ el servidor abre otra conexión TCP para transferir otro archivo
- ❖ la conexión de control está "fuera de banda"
- ❖ el servidor FTP mantiene el "estado": directorio actual, autenticación

Raúl Darán, Nacho Pérez. v1.0 | Copo de Aplicación 47

Universidad de Alcalá | Departamento de Automática

## FTP: comandos, respuestas

**ejemplos de comando:**

- ❖ se envían como texto ASCII por el canal de control
- ❖ **USER usuario**
- ❖ **PASS contraseña**
- ❖ **LIST** devuelve lista de archivos en directorio actual
- ❖ **RETR** archivo obtiene (get) archivo
- ❖ **STOR** archivo almacena (put) archivo en el host remoto

**ejemplos de códigos**

- ❖ código de estado y frase (como en HTTP)
- ❖ 331 Username OK, password required
- ❖ 125 data connection already open; transfer starting
- ❖ 425 Can't open data connection
- ❖ 452 Error writing file

Raúl Darán, Nacho Pérez. v1.0 | Copo de Aplicación 48



Universidad de Alcalá | Departamento de Automática

## Capítulo 2: La capa de aplicación

2.1 Principios de las aplicaciones en red

2.2 Web y HTTP

2.3 FTP

2.4 Correo electrónico

- SMTP, POP3, IMAP

2.5 DNS

2.6 Aplicaciones P2P

2.7 Programación de sockets con TCP

2.8 Programación de sockets con UDP

Raúl Darán, Nacho Pérez. v1.0 | Capa de Aplicación 49

Universidad de Alcalá | Departamento de Automática

## Correo electrónico (e-mail)

cola mens. salientes

buzón usuario

**3 componentes principales:**

- ❖ agentes de usuario
- ❖ servidores de correo
- ❖ Simple Mail Transfer Protocol: SMTP (protocolo sencillo de transferencia de correo)

**Agente de Usuario**

- ❖ alias "lector de correo"
- ❖ composición, edición, lectura de mensajes de correo
- ❖ ej.: Outlook, elm, Thunderbird, iPhone Mail Client
- ❖ mensajes entrantes y salientes almacenados en el servidor

Raúl Darán, Nacho Pérez. v1.0 | Capa de Aplicación 50

Universidad de Alcalá | Departamento de Automática

## Correo electrónico: servidores

**Servidores de correo**

- ❖ **buzón** contiene mensajes entrantes para el usuario
- ❖ **cola de mensajes salientes** (pendientes de ser enviados)
- ❖ **protocolo SMTP** entre servidores para enviar mensajes
  - cliente: servidor que envía el mensaje
  - "servidor": servidor que recibe el mensaje

Raúl Darán, Nacho Pérez. v1.0 | Capa de Aplicación 51

Universidad de Alcalá | Departamento de Automática

## Correo electrónico: SMTP [RFC 2821]

- ❖ usa TCP para transferir mensajes de e-mail de modo fiable de cliente a servidor, puerto 25
- ❖ transferencia directa: del servidor emisor al receptor
- ❖ tres fases en la transferencia
  - *handshaking* ("negociación": establecimiento de conexión)
  - transferencia de mensajes
  - cierre
- ❖ interacción de comandos y respuestas
  - **comandos:** texto ASCII
  - **respuesta:** código de estado y frase
- ❖ los mensajes deben ir en ASCII de 7 bits

Raúl Darán, Nacho Pérez. v1.0 | Capa de Aplicación 52

Universidad de Alcalá | Departamento de Automática

## Escenario: Alicia envía mensaje a Benito

- 1) Alicia usa un AU para escribir mensaje a benito@alu.uah.es
- 2) El AU de Alicia envía el mensaje a su servidor de correo; se queda en la cola de mensajes
- 3) La parte de cliente de SMTP abre conexión TCP con el servidor de correo de Benito
- 4) El cliente SMTP envía el mensaje de Alicia por la conexión TCP
- 5) El servidor de Benito pone el mensaje en el buzón de Benito
- 6) Benito ejecuta su agente de usuario para leer el mensaje

Raúl Darán, Nacho Pérez. v1.0 | Capa de Aplicación 53

Universidad de Alcalá | Departamento de Automática

## Ejemplo de interacción SMTP

```

S: 220 hamburger.edu
C: HELO crepes.fr
S: 250 Hello crepes.fr, pleased to meet you
C: MAIL FROM: <alice@crepes.fr>
S: 250 alice@crepes.fr... Sender ok
C: RCPT TO: <bob@hamburger.edu>
S: 250 bob@hamburger.edu ... Recipient ok
C: DATA
S: 354 Enter mail, end with "." on a line by itself
C: Te gusta el ketchup?
C: Que tal pepinillos?
C: .
S: 250 Message accepted for delivery
C: QUIT
S: 221 hamburger.edu closing connection
  
```

Raúl Darán, Nacho Pérez. v1.0 | Capa de Aplicación 54

Universidad de Alcalá | Departamento de Automática

## ¡¡Inténtalo tú mismo!!

- ❖ telnet servername 25
- ❖ observar la respuesta 220 del servidor
- ❖ teclear los comandos HELO, MAIL FROM, RCPT TO, DATA, QUIT

manualmente permite enviar correos sin usar el cliente de correo

Raúl Durán, Nacho Pérez. v1.0 | Copo de Aplicación | 55

Universidad de Alcalá | Departamento de Automática

## SMTP: palabras finales

- ❖ SMTP usa conexiones persistentes
- ❖ SMTP requiere que el mensaje (encabezado y cuerpo) esté en ASCII de 7 bits
- ❖ El servidor SMTP usa CRLF.CRLF (retorno de carro-línea nueva, retorno de carro-línea nueva) para determinar el final de un mensaje

**comparación con HTTP:**

- ❖ HTTP: pull (extracción)
- ❖ SMTP: push (inserción)
- ❖ ambos tienen interacción comando/respuesta y códigos de estado
- ❖ HTTP: cada objeto encapsulado en su propio mensaje de respuesta
- ❖ SMTP: múltiples objetos enviados en mensaje multiparte

Raúl Durán, Nacho Pérez. v1.0 | Copo de Aplicación | 56

Universidad de Alcalá | Departamento de Automática

## Formato de mensaje de correo

SMTP: protocolo para intercambiar mensajes de correo

RFC 822: estándar para formato de mensaje de texto:

- ❖ cabeceras, p.ej.:
  - To:
  - From:
  - Subject:*¡diferente de los comandos SMTP!!*
- ❖ cuerpo
  - el "mensaje", sólo caracteres ASCII-7 bits

Raúl Durán, Nacho Pérez. v1.0 | Copo de Aplicación | 57

Universidad de Alcalá | Departamento de Automática

## protocolos de acceso para correo

- ❖ SMTP: entrega/reparto al servidor del receptor
- ❖ protocolo de acceso para correo: retirada del servidor
  - POP: Post Office Protocol [RFC 1939]
    - autorización (agente <-->servidor) y descarga
  - IMAP: Internet Mail Access Protocol [RFC 1730]
    - más características (más complejo)
    - manipulación de mensajes en el propio servidor
  - HTTP: gmail, Hotmail, Yahoo! Mail, etc.

Raúl Durán, Nacho Pérez. v1.0 | Copo de Aplicación | 58

Universidad de Alcalá | Departamento de Automática

## protocolo POP3

**fase autorización**

- ❖ comandos cliente:
  - user: declarar usuario
  - pass: contraseña
- ❖ respuestas servidor:
  - +OK
  - -ERR

**fase transacción, cliente:**

- ❖ list: listar números mens.
- ❖ retr: obtener mensaje por número
- ❖ dele: borrar
- ❖ quit: cerrar

```

S: +OK POP3 server ready
C: user bob
S: +OK
C: pass hungry
S: +OK user successfully logged on

C: list
S: 1 498
S: 2 912
S: .
C: retr 1
S: <message 1 contents>
S: .
C: dele 1
C: retr 2
S: <message 1 contents>
S: .
C: dele 2
C: quit
S: +OK POP3 server signing off
  
```

Raúl Durán, Nacho Pérez. v1.0 | Copo de Aplicación | 59

Universidad de Alcalá | Departamento de Automática

## POP3 (más) e IMAP

**más sobre POP3**

- ❖ el ejemplo usa el modo "descargar y borrar".
- ❖ Bob no puede releer el mensaje si cambia de cliente
- ❖ "descargar y mantener": copias de clientes en clientes diferentes
- ❖ POP3 es 'stateless' (sin estado) de sesión en sesión

**IMAP**

- ❖ mantiene todos los mensajes en un sólo sitio: el servidor
- ❖ permite al usuario organizar sus mensajes en carpetas
- ❖ mantiene el estado del usuario de sesión en sesión:
  - nombres de carpetas y correspondencia entre IDs de mensaje y nombre de carpeta

Raúl Durán, Nacho Pérez. v1.0 | Copo de Aplicación | 60

Universidad de Alcalá | Departamento de Automática

## Capítulo 2: La capa de aplicación

- 2.1 Principios de las aplicaciones en red
- 2.2 Web y HTTP
- 2.3 FTP
- 2.4 Correo electrónico
  - SMTP, POP3, IMAP
- 2.5 DNS
- 2.6 Aplicaciones P2P
- 2.7 Programación de sockets con TCP
- 2.8 Programación de sockets con UDP

Raúl Durán, Nacho Pérez. v1.0 | Capa de Aplicación 61

Universidad de Alcalá | Departamento de Automática

## DNS: Domain Name System (Sistema de Nombres de Dominio)

**gente:** varios identificadores:
 

- DNI, nombre, pasaporte

**hosts y routers de Internet:**

- dir. IP (32 bits) - usada para direccionar datagramas
- "nombre", p.ej.: www.yahoo.com - usado por humanos

**P:** ¿correspondencia IP <-> nombre (y viceversa)?

**Domain Name System:**

- ❖ *base de datos distribuida* implementada en jerarquía de *servidores de nombres*
- ❖ *protocolo de capa de aplicación* host, routers, servidores de nombres deben comunicarse para *resolver* nombres (traducción dirección/nombre)
  - nota: es una función del núcleo de la red, implementada como un protocolo de la capa de aplicación
  - la complejidad está en la frontera de la red

Raúl Durán, Nacho Pérez. v1.0 | Capa de Aplicación 62

Universidad de Alcalá | Departamento de Automática

## DNS

**servicios DNS**

- ❖ traducción de nombre de host a IP
- ❖ *alias de host*
  - nombre canónico, alias
- ❖ *alia de servidor de correo*
- ❖ distribución de carga servidores Web replicados: conjunto de direcciones IP para un nombre canónico

**¿Por qué no centralizar DNS?**

- ❖ punto crítico para fallos
- ❖ volumen de tráfico
- ❖ base de datos centralizada lejana
- ❖ mantenimiento

idifícil de escalar!

Raúl Durán, Nacho Pérez. v1.0 | Capa de Aplicación 63

Universidad de Alcalá | Departamento de Automática

## Base de datos distribuida y jerárquica

```

  graph TD
    Root[Servidores DNS Raíz] --> Com[servidores DNS com]
    Root --> Org[servidores DNS org]
    Root --> Edu[servidores DNS edu]
    Com --> Yahoo[servidores DNS yahoo.com]
    Com --> Amazon[servidores DNS amazon.com]
    Org --> PBS[servidores DNS pbs.com]
    Edu --> Poly[servidores DNS poly.com]
    Edu --> Umass[servidores DNS umass.com]
  
```

**el cliente quiere la IP de www.amazon.com; 1er enfoque:**

- ❖ el cliente pregunta a un servidor de raíz por un servidor com
- ❖ el cliente pregunta al servidor com por el servidor amazon.com
- ❖ el cliente pregunta al servidor amazon.com por la IP de www.amazon.com
- ❖ (todos los servidores = servidores DNS)

Raúl Durán, Nacho Pérez. v1.0 | Capa de Aplicación 64

Universidad de Alcalá | Departamento de Automática

## DNS: Servidores raíz

- ❖ contactados por servidores locales de nombres que no pueden resolver un nombre
- ❖ servidor raíz de nombre:
  - contacta con un servidor de nombres autoritativo si no se conoce la correspondencia del nombre
  - obtiene la correspondencia
  - devuelve la correspondencia al servidor local

hay 13 servidores raíz en el mundo

Raúl Durán, Nacho Pérez. v1.0 | Capa de Aplicación 65

Universidad de Alcalá | Departamento de Automática

## TLDs y servidores autoritativos

**'Top-level domain' (TLD) servers (servidores de nivel superior):**

- responsables de los dominios com, org, net, edu, y los dominios de nivel superior de los distintos países, p.ej.: es, uk, fr, ca, jp
- Network Solutions mantiene los servidores DNS com
- Educause lo hace para edu

**Servidores DNS autoritativos:**

- servidores DNS de una organización; proporcionan correspondencias nombre/IP para los servidores de la organización (Web, mail, etc)
- pueden ser mantenidos por la organización o por un proveedor de servicios

Raúl Durán, Nacho Pérez. v1.0 | Capa de Aplicación 66

Universidad de Alcalá | Departamento de Automática

## Servidor local de nombres

- ❖ estrictamente no pertenece a la jerarquía
- ❖ cada ISP tiene uno
  - también se le llama "servidor de nombres por omisión"
- ❖ cuando un host hace una petición DNS, se envía a su servidor DNS local
  - funciona como proxy, transmite la petición a la jerarquía

Raúl Durán, Nacho Pérez. v1.0 | Copo de Aplicación 67

Universidad de Alcalá | Departamento de Automática

## ejemplo de resolución de nombre DNS

- ❖ un host de cis.poly.edu quiere la IP de gaia.cs.umass.edu

**consulta iterativa:**

- ❖ el servidor contactado responde con el nombre del servidor a contactar
- ❖ "Yo no lo sé, pero pregunta a este servidor"

Raúl Durán, Nacho Pérez. v1.0 | Copo de Aplicación 68

Universidad de Alcalá | Departamento de Automática

## ejemplo de resolución de nombre DNS

**consulta recursiva**

- ❖ pone la carga de la resolución en el servidor contactado
- ❖ ¿mucho carga?

Raúl Durán, Nacho Pérez. v1.0 | Copo de Aplicación 69

Universidad de Alcalá | Departamento de Automática

## DNS: cacheado y almacenamiento de registros

- ❖ una vez que un servidor de nombres (cualquiera) conoce una correspondencia, la guarda en su cache
  - las entradas de cache caducan en un tiempo
  - los servidores TLD típicamente están cacheados en servidores de nombres locales
    - De esta manera los servidores raíz no se visitan muy a menudo
- ❖ mecanismos de actualización/notificación propuestos en el IETF standard
  - RFC 2136

Raúl Durán, Nacho Pérez. v1.0 | Copo de Aplicación 70

Universidad de Alcalá | Departamento de Automática

## registros DNS

**DNS:** base de datos distribuida que almacena registros de recursos ('resource records', RR)

formato RR: (nombre, valor, tipo, ttl)

**Tipo=A**

- nombre es un nombre de host
- valor es su IP

**Tipo=NS**

- nombre es un dominio (p.ej.: foo.com)
- valor es el nombre del servidor autoritativo para ese dominio

**Tipo=CNAME**

- nombre es el alias de un nombre "canónico" (el real)
- www.ibm.com es en realidad servereast.backup2.ibm.com
- valor es el nombre canónico

**Tipo=MX**

- valor es el nombre de un servidor de correo asociado con nombre

Raúl Durán, Nacho Pérez. v1.0 | Copo de Aplicación 71

Universidad de Alcalá | Departamento de Automática

## DNS: protocolo, mensajes

**protocolo DNS:** mensajes de *consulta* y *respuesta*, ambos con el mismo *formato*

**cabecera**

- ❖ **identificación:** n° de 16 bits para la consulta, la respuesta usa el mismo n°
- ❖ **flags:**
  - consulta o respuesta
  - se prefiere recursividad
  - recursividad disponible
  - la respuesta es autoritativa

identification	flags	12 bytes
number of questions	number of answer RRs	
number of authority RRs	number of additional RRs	
questions (variable number of questions)		
answers (variable number of resource records)		
authority (variable number of resource records)		
additional information (variable number of resource records)		

Raúl Durán, Nacho Pérez. v1.0 | Copo de Aplicación 72

Universidad de Alcalá | Departamento de Automática

## DNS: protocolo, mensajes

campos nombre y tipo para una consulta

RRs de respuesta a la consulta

registros para servidores autoritativos

información adicional que se puede usar

12 bytes

identification flags

number of questions number of answer RRs

number of authority RRs number of additional RRs

questions (variable number of questions)

answers (variable number of resource records)

authority (variable number of resource records)

additional information (variable number of resource records)

Raúl Durán, Nacho Pérez. v1.0 | Copo de Aplicación 73

Universidad de Alcalá | Departamento de Automática

## Insertar registros en DNS

- ❖ ejemplo: nueva empresa "Churros Alcalá"
- ❖ registrar el nombre `churros-alcala.com` en un **registrador DNS** (p.ej.: Network Solutions)
  - proporcionar nombres, IPs del servidor autoritativo (primario y secundario)
  - el registrador inserta 2 RRs en el servidor TLD del dominio `com`:
 

```
(churros-alcala.com, dns.churros-alcala.com, NS)
(dns.churros-alcala.com, 212.212.212.1, A)
```
- ❖ crear registros en el servidor autoritativo: de Tipo **A** para `www.churros-alcala.com`, y de Tipo **MX** para `@churros-alcala.com`
- ❖ **¿Cómo se consigue una IP para tu sitio Web?**

Raúl Durán, Nacho Pérez. v1.0 | Copo de Aplicación 74

Universidad de Alcalá | Departamento de Automática

## Capítulo 2: La capa de aplicación

- 2.1 Principios de las aplicaciones en red
- 2.2 Web y HTTP
- 2.3 FTP
- 2.4 Correo electrónico
  - SMTP, POP3, IMAP
- 2.5 DNS

- 2.6 Aplicaciones P2P
- 2.7 Programación de sockets con TCP
- 2.8 Programación de sockets con UDP

Raúl Durán, Nacho Pérez. v1.0 | Copo de Aplicación 75

Universidad de Alcalá | Departamento de Automática

## Arquitectura P2P pura

- ❖ no hay servidor siempre activo
- ❖ sistemas terminales arbitrarios se comunican directamente
- ❖ los pares se conectan intermitentemente y cambian de IP

peer-peer

3 cuestiones:

- distribución de archivos
- búsqueda de información
- caso de estudio: Skype

Raúl Durán, Nacho Pérez. v1.0 | Copo de Aplicación 76

Universidad de Alcalá | Departamento de Automática

## Distribución de archivos: cliente-servidor frente a P2P

**Pregunta:** ¿Tiempo en distribuir un archivo de un servidor a N pares?

Servidor

Archivo, tamaño  $F$

Red (con ancho de banda de sobra)

Red (con ancho de banda de sobra)

$u_s$ : ancho de banda de carga del servidor

$u_i$ : ancho de banda de carga del par  $i$

$d_i$ : ancho de banda de descarga del par  $i$

Raúl Durán, Nacho Pérez. v1.0 | Copo de Aplicación 77

Universidad de Alcalá | Departamento de Automática

## Tiempo de distribución: cliente-servidor

- ❖ el servidor envía  $N$  copias secuencialmente:
  - tiempo:  $NF/u_s$
- ❖ al cliente  $i$  le lleva  $F/d_i$  descargarlo

Tiempo en distribuir  $F$  a  $N$  clientes =  $d_{cs} = \max \{ NF/u_s, F/\min(d_i) \}$  usando C/S

aumenta linealmente con  $N$  (para  $N$  grande)

Raúl Durán, Nacho Pérez. v1.0 | Copo de Aplicación 78

**Tiempo de distribución: P2P**

- ❖ el servidor debe enviar una copia, tiempo:  $F/u_s$
- ❖ al cliente  $i$  le lleva  $F/d_i$ , descargarlo
- ❖ se deben descargar  $NF$  bits (tiempo acumulado)
  - tasa máxima posible de carga:  $u_s + \sum u_i$

$$d_{P2P} = \max \{ F/u_s, F/\min(d_i), NF/(u_s + \sum u_i) \}$$

**Cliente-servidor frente a P2P: ejemplo**

tasa de carga del cliente =  $u$ ,  $F/u = 1$  hora,  $u_s = 10u$ ,  $d_{\min} \geq u_s$

N	Client-Server (Time)	P2P (Time)
0	0	0
5	0.5	0.75
10	1.0	0.75
15	1.5	0.75
20	2.0	0.75
25	2.5	0.75
30	3.0	0.75
35	3.5	0.75

**Distribución de archivos: BitTorrent**

Distribución de archivos P2P

**tracker:** registra pares que participan en un torrente

**torrente:** grupo de pares que intercambian trozos de un archivo

obtener lista de pares

intercambiar trozos

par

**BitTorrent (1)**

- ❖ los archivos se dividen en **trozos** ('chunks') de 256KB
- ❖ un par que se une a un torrente:
  - no tiene trozos, pero los acumulará con el tiempo
  - se registra en el tracker para obtener lista de pares, se conecta a un subconjunto ("vecinos")
- ❖ mientras descarga, el par carga trozos a otros pares
- ❖ los pares aparecen y desaparecen
- ❖ cuando un par completa el archivo, puede retirarse (egoístamente) o seguir conectado (generosamente)

**BitTorrent (2)**

**Extraer trozos**

- ❖ en cada momento, pares diferentes tienen diferentes subconjuntos de trozos
- ❖ periódicamente, cada par pide a cada vecino la lista de lo que tienen
- ❖ entonces solicita los trozos que le faltan
  - el menos común primero

**Envío de trozos: ojo por ojo**

- ❖ Un par envía trozos a los 4 vecinos que le envían a él a la **máxima velocidad**
  - el "top 4" se reevalúa cada 10 segundos
- ❖ cada 30 segundos se elige aleatoriamente otro par y se le empieza a enviar
  - el nuevo par elegido puede pasar al "top 4"
  - "no filtrado de forma optimista"

**BitTorrent: ojo por ojo**

- (1) Alice "desfiltra optimistamente" a Bob
- (2) Alice se convierte en uno de los "top 4" de Bob; Bob corresponde
- (3) Bob entra en el "top 4" de Alice

Con mayor tasa de carga se encuentran mejores pares y se obtiene el archivo antes

Universidad de Alcalá | Departamento de Automática

## Distributed Hash Table (DHT) (Tabla Hash Distribuida)

- ❖ DHT: base de datos P2P distribuida
- ❖ la base tiene duplas (**clave, valor**)
  - clave: DNI; valor: nombre
  - clave: tipo de contenido; valor: IP
- ❖ los pares **consultan** la BD con la clave
  - la BD devuelve valores que coinciden con la clave
- ❖ los pares también pueden **insertar** duplas (clave,valor)

Raúl Durán, Nacho Pérez. v1.0 | Copia de Aplicación 85

Universidad de Alcalá | Departamento de Automática

## Identificadores DHT

- ❖ asignar un ID entero a cada par en el rango  $[0, 2^n - 1]$ .
  - cada ID se representa con n bits
- ❖ exigir que cada clave sea un entero del **mismo rango**
- ❖ para obtener claves enteras, hacer un hash de la original
  - p.ej.: clave= h("Led Zeppelin IV")
  - por eso se llama tabla "hash" distribuida

Raúl Durán, Nacho Pérez. v1.0 | Copia de Aplicación 86

Universidad de Alcalá | Departamento de Automática

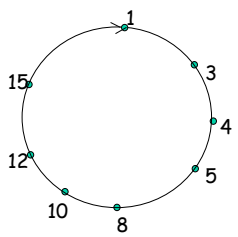
## ¿Cómo asignar claves a pares?

- ❖ cuestión central:
  - asignar duplas (clave,valor) a pares
- ❖ regla: asignar clave al par que tenga el ID **más cercano**
- ❖ convención de lectura: el más cercano es el inmediato sucesor de la clave
- ❖ p.ej.: n=4; pares: 1,3,4,5,8,10,12,14;
  - clave= 13, entonces sucesor par = 14
  - clave= 15, entonces sucesor par = 1

Raúl Durán, Nacho Pérez. v1.0 | Copia de Aplicación 87

Universidad de Alcalá | Departamento de Automática

## DHT Circular (1)



- ❖ cada par sólo conoce a sus inmediatos sucesor y predecesor
- ❖ "red solapada"

Raúl Durán, Nacho Pérez. v1.0 | Copia de Aplicación 88

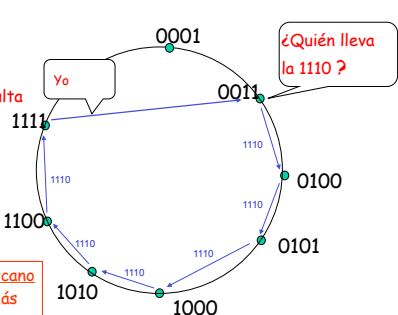
Universidad de Alcalá | Departamento de Automática

## DHT Circular (2)

con N pares, en promedio  $O(N)$  mensajes para resolver la consulta

Yo

¿Quién lleva la 1110 ?

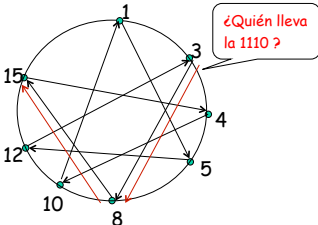


Definir **más cercano** como sucesor más cercano

Raúl Durán, Nacho Pérez. v1.0 | Copia de Aplicación 89

Universidad de Alcalá | Departamento de Automática

## DHT Circular con Atajos



¿Quién lleva la 1110 ?

- ❖ cada par controla las IPs del predecesor, el sucesor y los atajos
- ❖ reduce de 6 a 2 mensajes
- ❖ es posible diseñar atajos para  $O(\log N)$  vecinos,  $O(\log N)$  mensajes por consulta

Raúl Durán, Nacho Pérez. v1.0 | Copia de Aplicación 90



**Abandono de los pares**

- ❖ Para manejar el abandono, hay que exigir que cada par sepa la IP de sus 2 sucesores
- ❖ cada par hace *ping* periódicamente a sus 2 sucesores, para ver si siguen vivos
- ❖ el par 5 sale del sistema
- ❖ el par 4 se da cuenta; hace a 8 su inmediato sucesor; le pregunta a 8 su inmediato sucesor, y le hace su segundo sucesor
- ❖ ¿Qué pasa si el par 13 se quiere unir?

Raúl Durán, Nacho Pérez. v1.0 Copa de Aplicación 91

**Caso de estudio P2P: Skype**

- ❖ inherentemente P2P: pares de usuarios se comunican
- ❖ protocolo de capa de aplicación propietario (deducido por ingeniería inversa)
- ❖ superposición jerárquica con super-pares (super-nodos)
- ❖ un índice relaciona usuarios con IPs; se distribuye sobre los super-pares

Raúl Durán, Nacho Pérez. v1.0 Copa de Aplicación 92

**Pares retransmisores**

- ❖ hay un problema cuando Alice y Bob están detrás de NATs
  - Un NAT impide a un par de fuera iniciar una llamada a uno de dentro
- ❖ solución:
  - usando los SPs de Alice y Bob, se elige un **retransmisor**
  - cada par inicia sesión con el retransmisor
  - los pares ahora se comunican a través de los NATs por medio del retransmisor

Raúl Durán, Nacho Pérez. v1.0 Copa de Aplicación 93

**Capítulo 2: La capa de aplicación**

- 2.1 Principios de las aplicaciones en red
- 2.2 Web y HTTP
- 2.3 FTP
- 2.4 Correo electrónico
  - SMTP, POP3, IMAP
- 2.5 DNS

- 2.6 Aplicaciones P2P
- 2.7 Programación de sockets con TCP
- 2.8 Programación de sockets con UDP

Raúl Durán, Nacho Pérez. v1.0 Copa de Aplicación 94

**Programación con Sockets**

**Objetivo:** aprender a crear una aplicación cliente/servidor que se comunique por sockets

**API de Socket**

- ❖ introducida en Unix BSD4.1, 1981
- ❖ se crea, usa, libera explícitamente por la aplicación
- ❖ paradigma cliente/servidor
- ❖ 2 tipos de servicio de transporte en el API de socket:
  - datagrama no fiable
  - fiable, orientado a flujo de bytes

**socket**

un *interfaz local, creado por la aplicación, controlado por el SO*, (una "puerta") en la que un proceso puede *enviar y recibir* mensajes a/desde otro proceso

Raúl Durán, Nacho Pérez. v1.0 Copa de Aplicación 95

**Programación de sockets con TCP**

**Socket:** una puerta entre un proceso de la aplicación y un protocolo de transporte de extremo a extremo (TCP o UCP)

**servicio TCP:** transferencia fiable de bytes de un proceso al otro

Raúl Durán, Nacho Pérez. v1.0 Copa de Aplicación 96

**programación de sockets con TCP**

- El cliente debe contactar con el servidor
  - ❖ proceso servidor debe primero estar corriendo
  - ❖ el servidor debe haber creado socket que responda al contacto del cliente
- El cliente contacta con el servidor:
  - ❖ al crear su socket TCP,
  - ❖ especificar IP y puerto del proceso servidor,
  - ❖ cuando el cliente crea el socket: el TCP del cliente se conecta con el del servidor

❖ cuando el cliente contacta con él, el TCP del servidor crea un socket nuevo para que el proceso servidor se comunique con el cliente

- esto permite al servidor hablar con varios clientes
- hay puertos de origen distintos para distinguir clientes (más en cap. 3)

desde el punto de vista de la aplicación

TCP proporciona transferencia de bytes fiable y en orden (un "pipe") entre cliente y servidor

Raúl Durán, Nacho Pérez. v1.0 Copa de Aplicación 97

**interacción de sockets cliente/servidor: TCP**

**Servidor (corriendo en hostid)**

```

crear socket,
puerto=x, para
solicitud entrante:
welcomeSocket =
ServerSocket()
esperar solicitud de
conexión entrante
connectionSocket =
welcomeSocket.accept()
leer solicitud de
connectionSocket
escribir respuesta en
connectionSocket
cerrar
connectionSocket
                    
```

**Cliente**

```

crear socket,
conectar con hostid, puerto=x
clientSocket =
Socket()
enviar solicitud usando
clientSocket
leer respuesta de
clientSocket
cerrar
clientSocket
                    
```

establecimiento conexión TCP

Raúl Durán, Nacho Pérez. v1.0 Copa de Aplicación 98

**jerga de flujos**

- ❖ flujo ('stream') es una secuencia de caracteres que fluyen a o desde un proceso
- ❖ flujo de entrada está asociado a alguna fuente de entrada para el proceso, como un teclado o un socket
- ❖ flujo de salida está asociado a una fuente de salida, como un monitor o un socket

Raúl Durán, Nacho Pérez. v1.0 Copa de Aplicación 99

**Programación de sockets con TCP**

**aplicación cliente-servidor ejemplo:**

- 1) el cliente lee una línea de entrada estándar (flujo `inFromUser`), la envía al servidor por el socket (flujo `outToServer`)
- 2) el servidor lee línea del socket
- 3) el servidor la convierte a mayúsculas y la devuelve al cliente
- 4) el cliente lee la línea del socket y la muestra (flujo `inFromServer`)

Raúl Durán, Nacho Pérez. v1.0 Copa de Aplicación 100

**Ejemplo: cliente Java (TCP)**

```

import java.io.*;
import java.net.*;
class TCPClient {

    public static void main(String argv[]) throws Exception
    {
        String sentence;
        String modifiedSentence;

        BufferedReader inFromUser =
        new BufferedReader(new InputStreamReader(System.in));
        Socket clientSocket = new Socket("hostname", 6789);
        DataOutputStream outToServer =
        new DataOutputStream(clientSocket.getOutputStream());
                    
```

Este paquete define las clases Socket() y ServerSocket()

nombre del servidor, p.ej.: www.umass.edu

puerto servidor

crear flujo entrada

crear objeto clientSocket de tipo Socket, conectar con servidor

crear flujo salida asociado al socket

Raúl Durán, Nacho Pérez. v1.0 Copa de Aplicación 101

**Ejemplo: cliente Java (TCP), cont.**

```

        BufferedReader inFromServer =
        new BufferedReader(new
        InputStreamReader(clientSocket.getInputStream()));
        sentence = inFromUser.readLine();
        outToServer.writeBytes(sentence + '\n');
        modifiedSentence = inFromServer.readLine();
        System.out.println("DEL SERVIDOR: " + modifiedSentence);
        clientSocket.close();
    }
}
                    
```

crear flujo entrada asociado al socket

enviar línea al servidor

leer línea del servidor

cerrar socket (de por todo limpiar)

Raúl Durán, Nacho Pérez. v1.0 Copa de Aplicación 102

### Ejemplo: servidor Java (TCP)

```

import java.io.*;
import java.net.*;

class TCPServer {
    public static void main(String argv[]) throws Exception
    {
        String clientSentence;
        String capitalizedSentence;

        crear socket recepción en el puerto 6789 → ServerSocket welcomeSocket = new ServerSocket(6789);

        while(true) {
            espera, el método accept() del socket de recepción crea un socket nuevo → Socket connectionSocket = welcomeSocket.accept();

            crear flujo entrada, asociado al socket → BufferedReader inFromClient =
                new BufferedReader(new
                    InputStreamReader(connectionSocket.getInputStream()));
    }
}

```

Raúl Durán, Nacho Pérez. v1.0 Copa de Aplicación 103

### Ejemplo: servidor Java (TCP), cont

```

        DataOutputStream outToClient =
            new DataOutputStream(connectionSocket.getOutputStream());

        leer línea del socket → clientSentence = inFromClient.readLine();

        capitalizedSentence = clientSentence.toUpperCase() + "\n";

        escribir línea al socket → outToClient.writeBytes(capitalizedSentence);
    }
}

```

**final del while, volver al principio y esperar otra conexión de un cliente**

Raúl Durán, Nacho Pérez. v1.0 Copa de Aplicación 104

## Capítulo 2: La capa de aplicación

2.1 Principios de las aplicaciones en red	2.6 Aplicaciones P2P
2.2 Web y HTTP	2.7 Programación de sockets con TCP
2.3 FTP	<b>2.8 Programación de sockets con UDP</b>
2.4 Correo electrónico	
▪ SMTP, POP3, IMAP	
2.5 DNS	

Raúl Durán, Nacho Pérez. v1.0 Copa de Aplicación 105

## Programación de sockets *con UDP*

**UDP: no hay "conexión" entre cliente y servidor**

- sin establecimiento de conexión
- el emisor incluye IP y puerto de destino en cada paquete
- el servidor debe extraer la IP y nº de puerto del emisor del paquete recibido

**desde el punto de vista de la aplicación:**  
*UDP proporciona transferencias no fiables de grupos de bytes ("datagramas") entre cliente y servidor*

**UDP: los datos transmitidos pueden ser recibidos fuera de orden, o incluso perderse**

Raúl Durán, Nacho Pérez. v1.0 Copa de Aplicación 106

### interacción de sockets cliente/servidor : UDP

Servidor (corriendo en <i>hostid</i> )	Client
crear socket, puerto= x, <b>serverSocket = DatagramSocket()</b> leer datagrama de <b>serverSocket</b> responder a <b>serverSocket</b> especificando IP del cliente y nº de puerto	crear socket, <b>clientSocket = DatagramSocket()</b> crear datagrama con IP del servidor y puerto=x; enviar datagrama por <b>clientSocket</b> leer datagrama de <b>clientSocket</b> cerrar <b>clientSocket</b>

Raúl Durán, Nacho Pérez. v1.0 Copa de Aplicación 107

### Ejemplo: cliente Java (UDP)

Raúl Durán, Nacho Pérez. v1.0 Copa de Aplicación 108

Universidad de Alcalá | Departamento de Automática

## Ejemplo: cliente Java (UDP)

```

import java.io.*;
import java.net.*;

class UDPClient {
    public static void main(String args[]) throws Exception
    {
        // crear flujo entrada
        BufferedReader inFromUser =
            new BufferedReader(new InputStreamReader(System.in));

        // crear socket cliente
        DatagramSocket clientSocket = new DatagramSocket();

        // traducir nombre host a IP usando DNS
        InetAddress IPAddress = InetAddress.getByName("hostname");

        byte[] sendData = new byte[1024];
        byte[] receiveData = new byte[1024];

        String sentence = inFromUser.readLine();
        sendData = sentence.getBytes();
    }
}

```

Raúl Durán, Nacho Pérez. v1.0 | Copo de Aplicación | 109

Universidad de Alcalá | Departamento de Automática

## Ejemplo: cliente Java (UDP), cont.

```

// crear datagrama con datos, longitud, IP, puerto
DatagramPacket sendPacket =
    new DatagramPacket(sendData, sendData.length, IPAddress, 9876);

// enviar datagrama al servidor
clientSocket.send(sendPacket);

DatagramPacket receivePacket =
    new DatagramPacket(receiveData, receiveData.length);

// leer datagrama del servidor
clientSocket.receive(receivePacket);

String modifiedSentence =
    new String(receivePacket.getData());

System.out.println("DEL SERVIDOR:" + modifiedSentence);
clientSocket.close();
}
}

```

Raúl Durán, Nacho Pérez. v1.0 | Copo de Aplicación | 110

Universidad de Alcalá | Departamento de Automática

## Ejemplo: servidor Java (UDP)

```

import java.io.*;
import java.net.*;

class UDPServer {
    public static void main(String args[]) throws Exception
    {
        // crear socket para datagramas en el puerto 9876
        DatagramSocket serverSocket = new DatagramSocket(9876);

        byte[] receiveData = new byte[1024];
        byte[] sendData = new byte[1024];

        // crear espacio para datagrama recibido
        while(true)
        {
            // recibir datagrama
            DatagramPacket receivePacket =
                new DatagramPacket(receiveData, receiveData.length);
            serverSocket.receive(receivePacket);
        }
    }
}

```

Raúl Durán, Nacho Pérez. v1.0 | Copo de Aplicación | 111

Universidad de Alcalá | Departamento de Automática

## Ejemplo: servidor Java (UDP), cont.

```

String sentence = new String(receivePacket.getData());

// obtener IP y nº de puerto del emisor
InetAddress IPAddress = receivePacket.getAddress();
int port = receivePacket.getPort();

String capitalizedSentence = sentence.toUpperCase();

sendData = capitalizedSentence.getBytes();

// crear datagrama para el cliente
DatagramPacket sendPacket =
    new DatagramPacket(sendData, sendData.length, IPAddress,
        port);

// escribir datagrama al socket
serverSocket.send(sendPacket);
}
}

```

fin del bucle while, volver al principio y esperar otro datagrama

Raúl Durán, Nacho Pérez. v1.0 | Copo de Aplicación | 112

Universidad de Alcalá | Departamento de Automática

## Capítulo 2: Resumen

estudio de las aplicaciones en red: **icompletado!**

- ❖ arquitecturas de aplicación
  - cliente-servidor
  - P2P
  - híbridas
- ❖ requisitos de servicio de las aplicaciones:
  - fiabilidad, ancho de banda, tiempo (retardo)
- ❖ modelo de servicio de transporte de Internet
  - orientado a conexión, fiable: TCP
  - no fiable, datagramas: UDP
- ❖ protocolos específicos:
  - HTTP
  - FTP
  - SMTP, POP, IMAP
  - DNS
  - P2P: BitTorrent, Skype
- ❖ programación de sockets

Raúl Durán, Nacho Pérez. v1.0 | Copo de Aplicación | 113

Universidad de Alcalá | Departamento de Automática

## Capítulo 2: Resumen

**lo más importante: hemos aprendido protocolos**

- ❖ intercambio típico de solicitud/respuesta:
  - el cliente solicita información o servicio
  - el cliente responde con datos, código de estado
- ❖ formatos de mensaje:
  - cabeceras: campos que dan información sobre los datos
  - datos: la información que se comunica

**Cuestiones importantes:**

- ❖ mensajes de control / mensajes de datos
  - en banda, fuera de banda
- ❖ centralizados / descentralizados
- ❖ con / sin estado
- ❖ transf. de mensajes fiable / no fiable
- ❖ "complejidad en la frontera de la red"

Raúl Durán, Nacho Pérez. v1.0 | Copo de Aplicación | 114