Capítulo 2
La capa de aplicación

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

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

Capítulo 2: La capa de aplicación

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

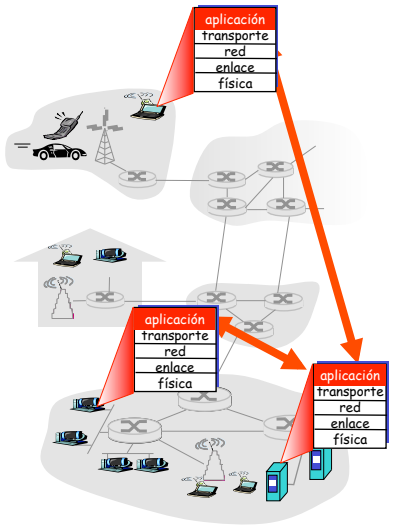
Algunas aplicaciones de red

- 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*
- ...
- ...
-

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



Capítulo 2: La capa de aplicación

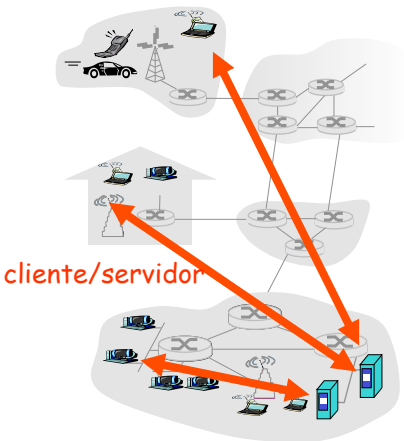
2.1 Principios de las aplicaciones en red
2.2 Web y HTTP
2.3 FTP
2.4 Correo electrónico
- SMTP, POP3, IMAP

2.5 DNS
2.6 Aplicaciones P2P
2.7 Programación de sockets con TCP
2.8 Programación de sockets con UDP

Arquitecturas de las aplicaciones en red

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

Arquitectura cliente-servidor



servidor:

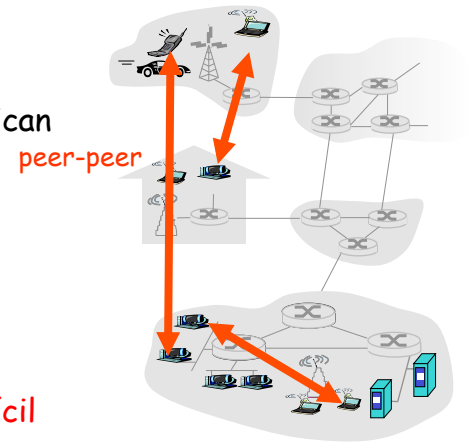
- host siempre activo
- IP permanente
- clusters para mejor escalabilidad

clientes:

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

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



muy escalable, pero difícil de gestionar

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 ususario registra su IP en el servidor central cuando se conecta
  - el usuario contacta con el servidor central para averiguar las IPs de sus contactos

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.

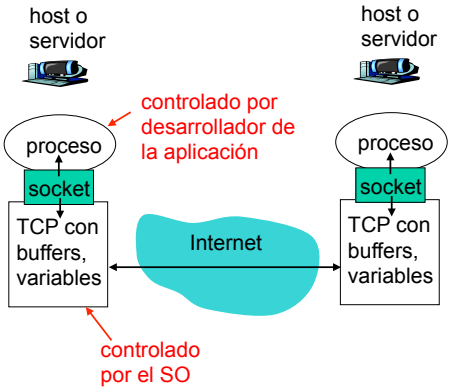
proceso cliente: el que inicia la comunicación

proceso servidor: el que espera a que contacten con él

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

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)

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?

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…*

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

¿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, …

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

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?

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

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

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:

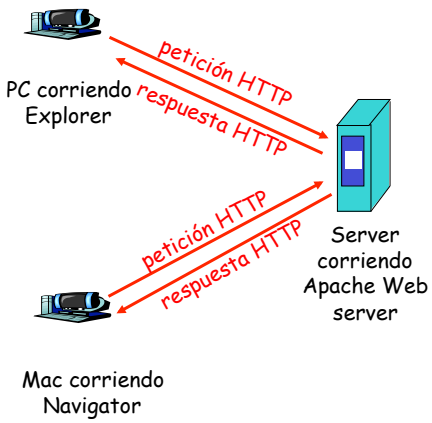
`www.someschool.edu/someDept/pic.gif`

nombre del host — ruta

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



Visión general de HTTP (cont.)

Usa TCP:

- el cliente inicia conexión TCP (crea el socket) con el servidor, puerto 80
- el servidor acepta la conexión TCP del cliente
- se intercambian mensajes HTTP (capa de aplicación) entre el navegador (cliente HTTP) y el servidor Web (servidor HTTP)
- Se cierra la conexión TCP

HTTP es "stateless" (sin estado)

- el servidor no guarda información sobre las peticiones de los clientes

por cierto…

los protocolos que guardan el estado son complejos

- se debe guardar la historia (el estado)
- si se cuelga el servidor o el cliente, sus "estados" pueden ser incongruentes, deben ser unificados.

conexiones HTTP

HTTP no persistente

- como máximo, 1 objeto enviado a través de una conexión TCP.

HTTP persistente

- múltiples objetos se pueden enviar a través de una conexión TCP única entre cliente y servidor.

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 TPC 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

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.

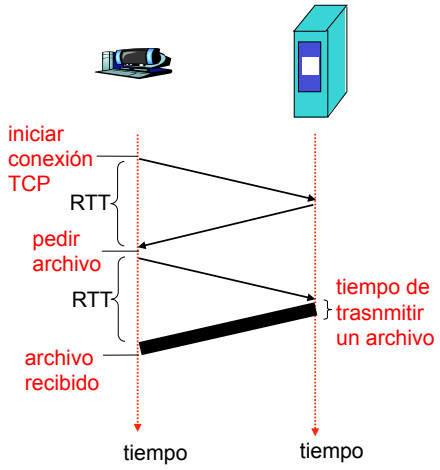
tiempo

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 conexion RTT
- un RTT para la petición HTTP y primeros bits de respuesta HTTP
- tiempo de trasmisión del archivo



total = 2RTT+tiempo de trasnsmisión

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

mensaje de solicitud HTTP

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

retorno de carro
línea nueva

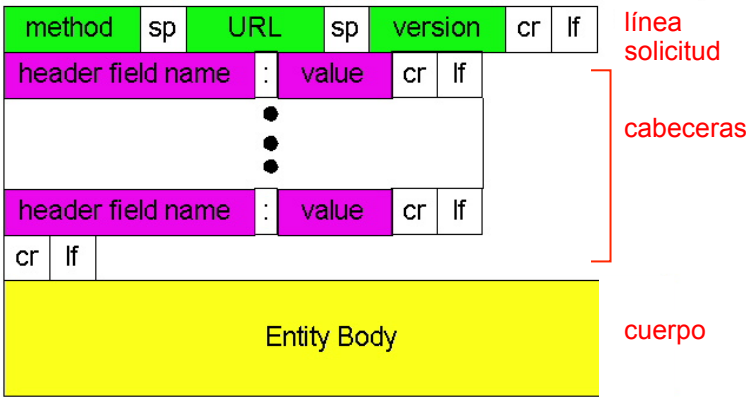
línea de solicitud (comandos GET, POST, HEAD)

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
```


solicitud HTTP : formato general



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
```


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

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 ...
```


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

Prueba de (cliente) por uno mismo

1. 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

2. teclear 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

3. ¡observar la respuesta del servidor HTTP!

(o usar Wireshark!)

estado del usuario: cookies

muchos sitios Web usan cookies

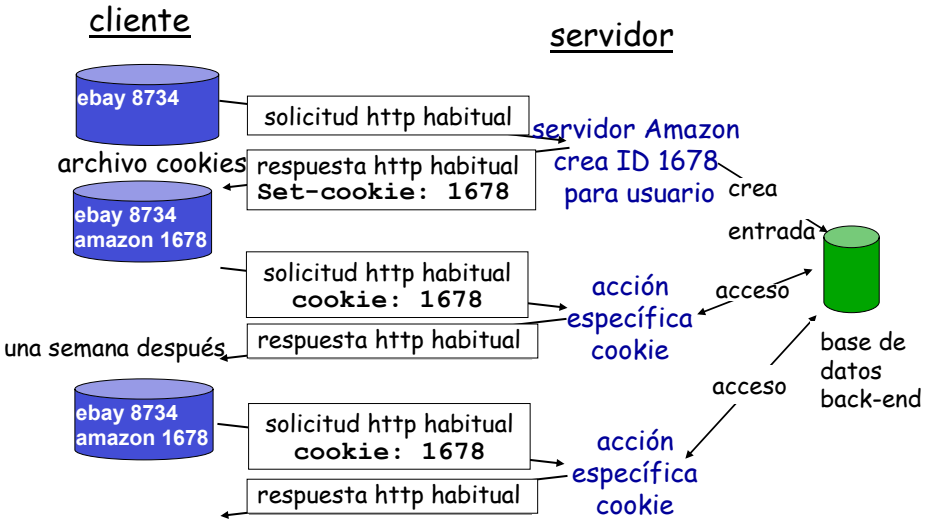
cuatro componentes:

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

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

estado del usuario: cookies (cont.)



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"

por cierto…

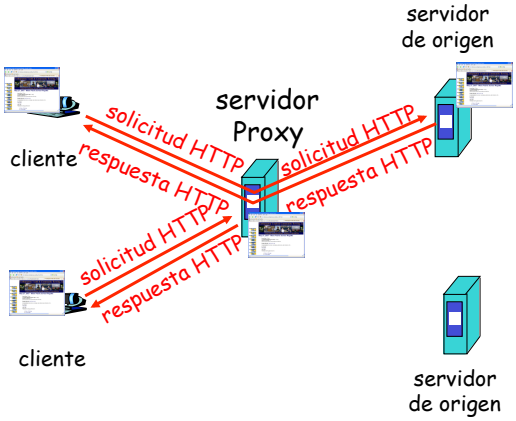
cookies y privacidad:

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

caches Web (servidor proxy)

Objetivo: cumplir la solicitud del cliente sin involucrar al servidor de origen

- configurar navegador: accesos Web por cache
- el navegador envía todas las solicitudes HTTP a la cache:
  - acierto: la cache devuelve el objeto
  - fallo: lo pide al servidor de origen y lo reenvía al cliente



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)

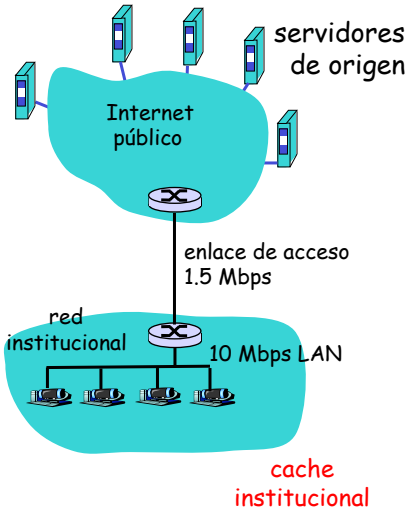
Ejemplo de sistema con cache

supuestos

- tamaño medio objeto = 100.000 bits
- tasa media solicitudes desde navegadores de la institución a servidores de origen = 15/s
- tiempo ida y vuelta desde el router de la institución a cualquier servidor de origen = 2 s

consecuencias

- uso en la LAN = 15%
- uso en el enlace de acceso = 100%
- tiempo total = tiempo Internet + tiempo acceso + tiempo LAN
  = 2 s + minutos + milisegundos



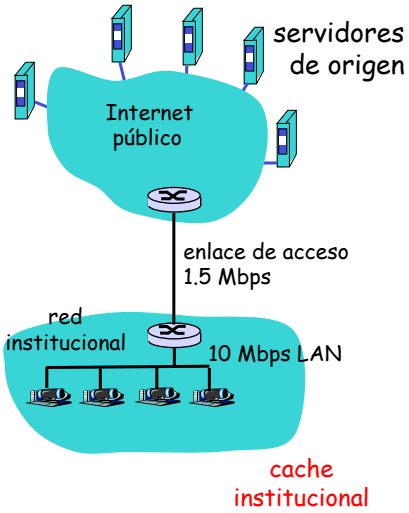
Ejemplo de sistema con cache (cont)

posibe solución

- aumentar ancho de banda del enlace de acceso a, por ejemplo, 10 Mbps

consecuencia

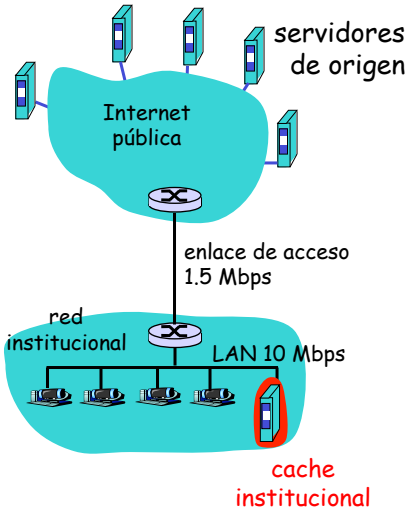
- uso en la LAN = 15%
- uso enlace acceso = 15%
- tiempo total = tiempo Internet + tiempo acceso + tiempo LAN
  = 2 s + milisegs + milisegs
- a menudo mejora muy costosa



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*(2,01) s + 0,4*varios ms < 1,4 s

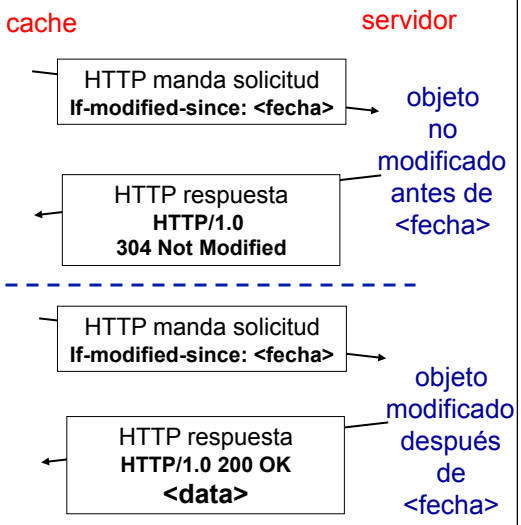


GET Condicional

- 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`

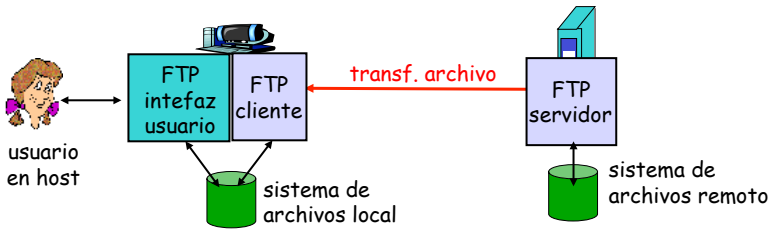


Capítulo 2: La capa de aplicación

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

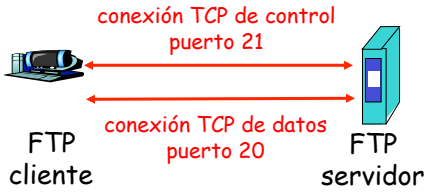
FTP: File Transfer Protocol
(protocolo de transferencia de archivos)



- 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

FTP: conexiones paralelas para control y datos

- El cliente contacta con el servidor en el puerto 21. El protocolo de tranporte es TCP
- se autoriza al cliente en la conexión de control
- el cliente hojea 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

FTP: comandos, respuestas

ejemplos de comando:

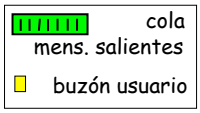
- se envían como texto ASCII por el canal de control
- **`USER`** ***`usuario`***
- **`PASS`** ***`contraseña`***
- **`LIST`** devuelve lista de archivos en directorio actual
- **`RETR archivo`** obtiene (get) archivo
- **`STOR archivo`** almacena (put) archivo en el host remoto

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`**

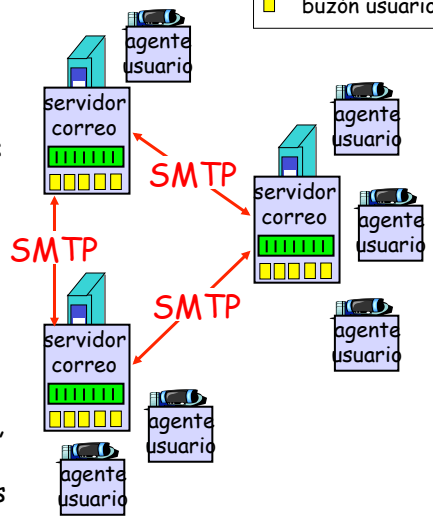
Capítulo 2: La capa de aplicación

Correo electrónico (e-mail)



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

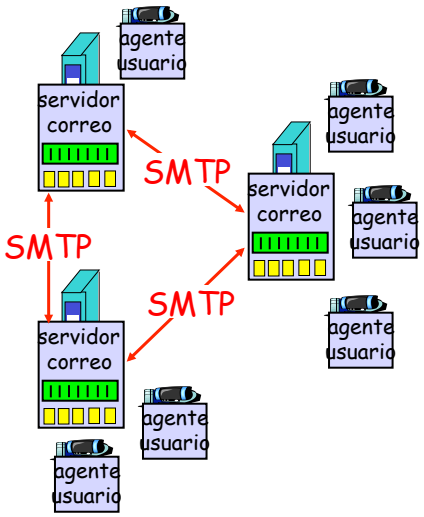
Agente de Usuario
- alias "lector de correo"
- composición, edición, lectura de mensajes de correo
- ej.: Outlook, elm, Thunderbird, iPhone Mail Client
- mensajes entrantes y salientes almacenados en el servidor



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

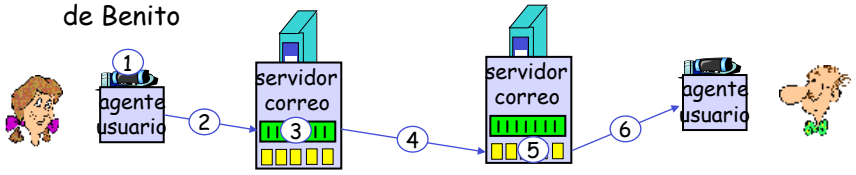


Correo electrónico: SMTP [RFC 2821]

- 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

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



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
```

¡¡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

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: múltipes objetos enviados en mensaje multiparte

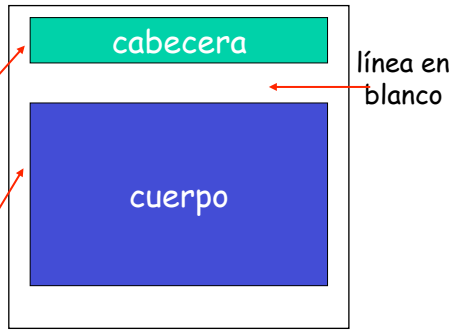
Formato de mensaje de correo

SMTP: protocolo para intercambiar mensajes de correo

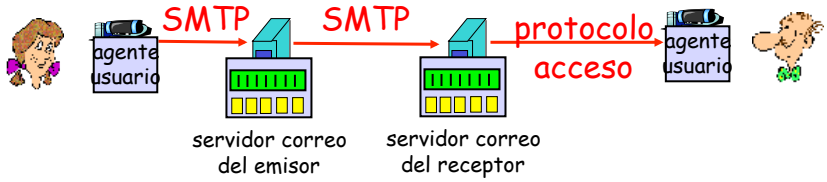
RFC 822: estándar para formato de mensaje de texto:

- cabeceras, p.ej.:
  - To:
  - From:
  - Subject:

  ¡¡diferente de los comandos SMTP!!
- cuerpo
  - el "mensaje", sólo caracteres ASCII-7 bits



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.

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
```


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 mensajes 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

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

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

DNS

servicios DNS

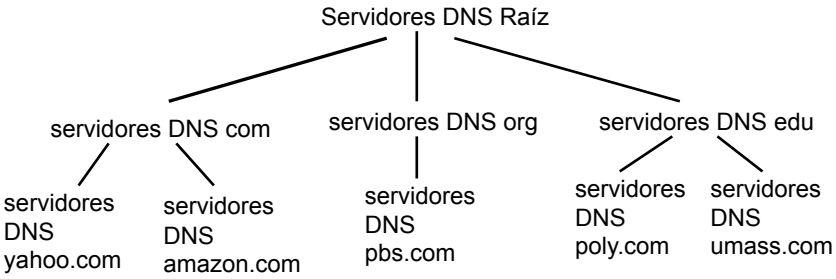
- traducción de nombre de host a IP
- *alias de* host
  - nombre canónico, alias
- *alia* de servidor de correo
- distribución de carga
  servidores Web replicados: conjunto de direcciones IP para un nombre canónico

¿Por qué no centralizar DNS?

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

¡difícil de escalar!

Base de datos distribuida y jerárquica

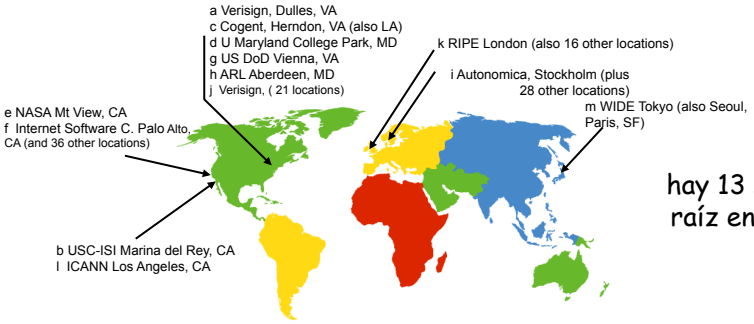


el cliente quiere la IP de www.amazon.com; 1er enfoque:

- 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)

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

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

Servidor local de nombres

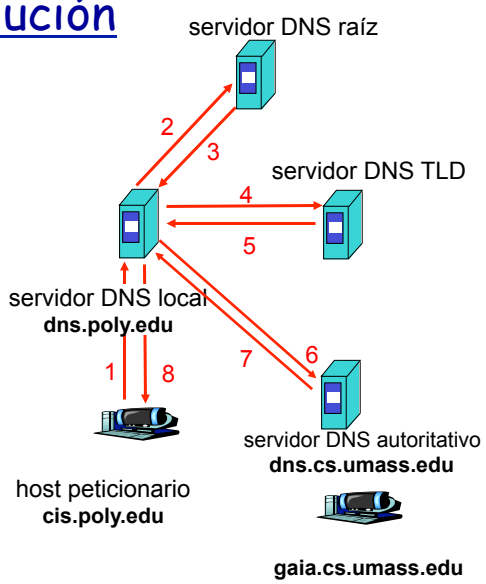
- estrictamente no pertenece a la jerarquía
- cada ISP tiene uno
  - también se le llama "servidor de nombres por omisión"
- cuando un host hace una petición DNS, se envía a su servidor DNS local
  - funciona como proxy, transmite la petición a la jerarquía

ejemplo de resolución de nombre DNS

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

consulta iterativa:

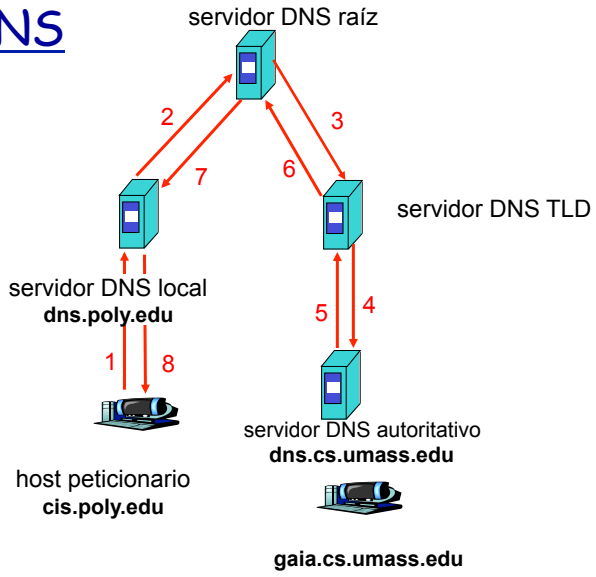
- el servidor contactado responde con el nombre del servidor a contactar
- "Yo no lo sé, pero pregunta a este servidor"



ejemplo de resolución de nombre DNS

consulta recursiva

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



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

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

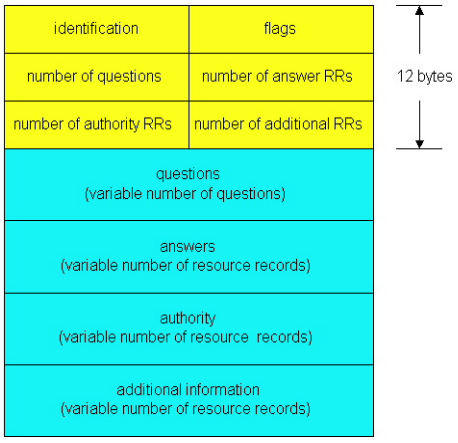
- **`valor`** es el nombre de un servidor de correo asociado con **`nombre`**

DNS: protocolo, mensajes

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

cabecera

- identificación: nº de 16 bits para la consulta, la respuesta usa el mismo nº
- flags:
  - consulta o respuesta
  - se prefiere recursividad
  - recursividad disponible
  - la respuesta es autoritativa



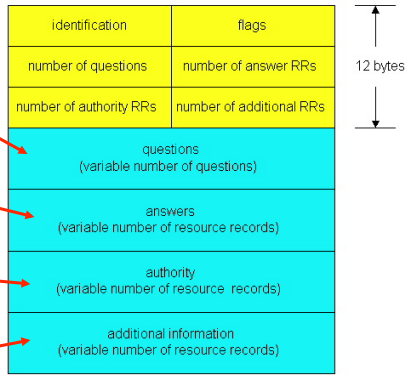
DNS: protocolo, mensajes

campos nombre y tipo para una consulta

RRs de respuesta a la consulta

registros para servidores autoritativos

información adicional que se puede usar



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?

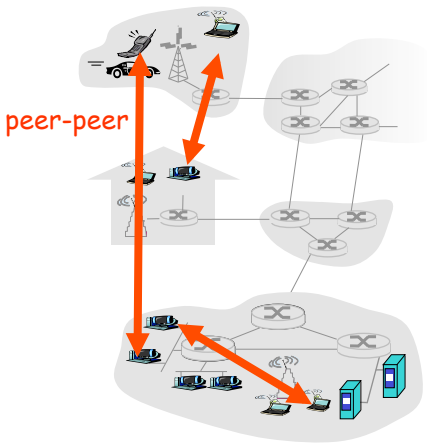
Capítulo 2: La capa de aplicación

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

Arquitectura P2P pura

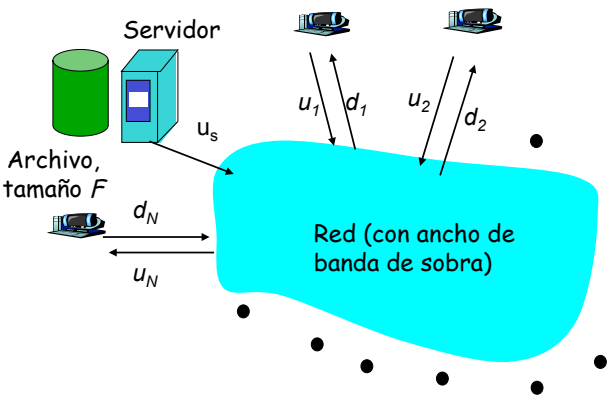
- no hay servidor siempre activo
- sistemas terminales arbitrarios se comunican directamente
- los pares se conectan intermitentemente y cambian de IP



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

Distribución de archivos: cliente-servidor frente a P2P

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



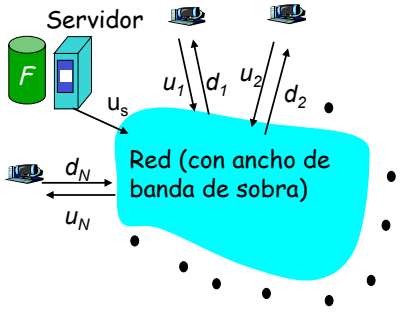
u_s: ancho de banda de carga del *servidor*

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

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

Tiempo de distribución: cliente-servidor

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

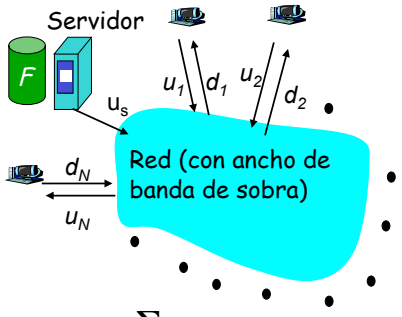


Tiempo en distribuir F a N clientes usando C/S = $d_{cs} = \max\{ NF/u_s, F/\min_i(d_i) \}$

aumenta linealmente con N (para N grande)

Tiempo de distribución: P2P

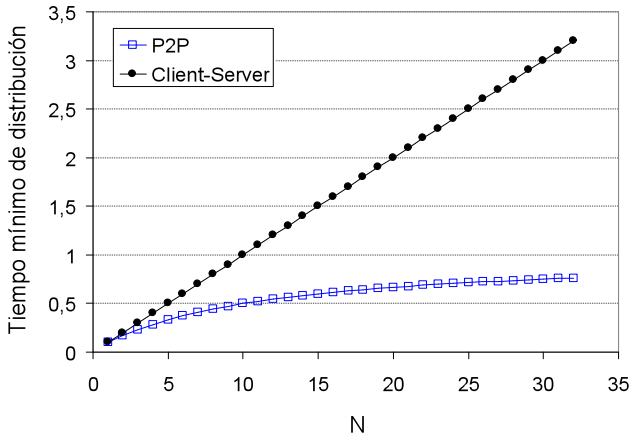
- el servidor debe enviar una copia, tiempo: F/u_s
- al cliente *i* le lleva F/d_i descargarlo
- se deben descargar NF bits (tiempo acumulado)
  - tasa máxima posible de carga: $u_s + \Sigma u_i$



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

Cliente-servidor frente a P2P: ejemplo

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

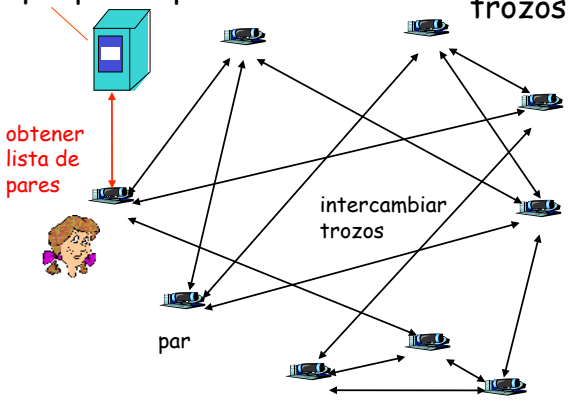


Distribución de archivos: BitTorrent

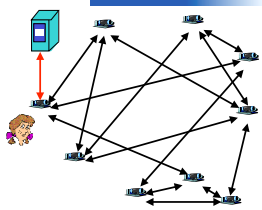
Distribución de archivos P2P

tracker: registra pares que participan en un torrente

torrente: grupo de pares que intercambian trozos de un archivo



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

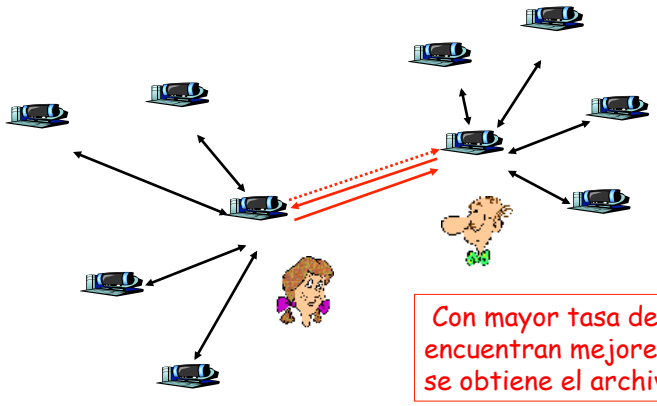
- en cada momento, pares diferentes tienen diferentes subconjuntos de trozos
- periódicamente, cada par pide a cada vecino la lista de lo que tienen
- entonces solicita los trozos que le faltan
  - el menos común primero

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

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)

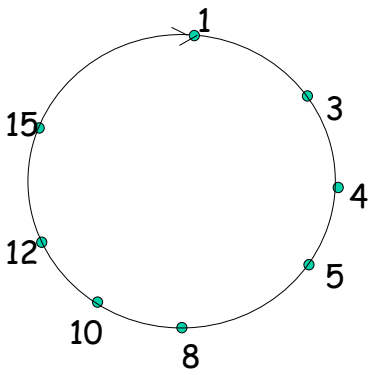
Identificadores DHT

- 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

¿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

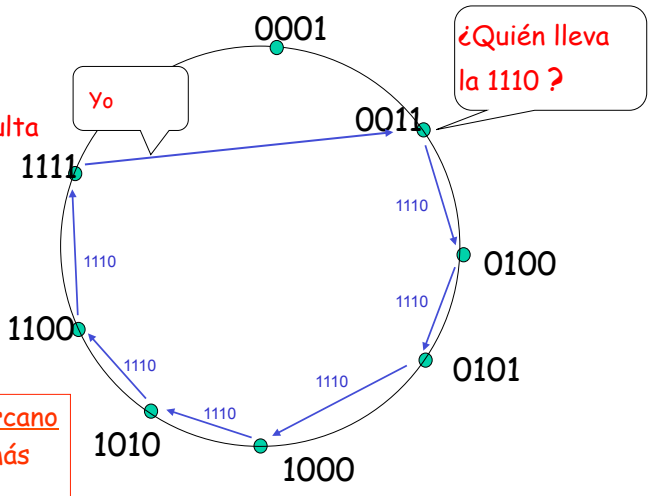
DHT Circular (1)



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

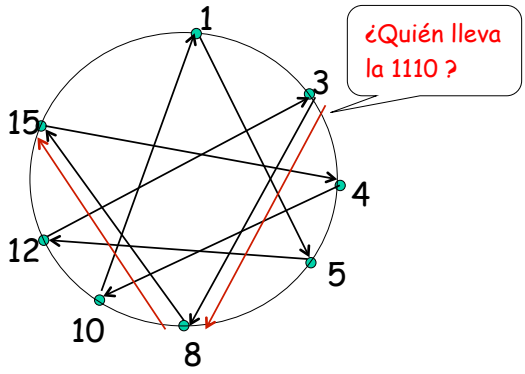
DHT Circular (2)

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



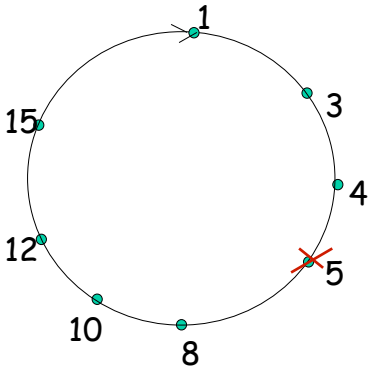
Definir más cercano como sucesor más cercano

DHT Circular con Atajos



- cada par controla las IPs del predecesor, el sucesor y los atajos
- reduce de 6 a 2 mensajes
- es posibe diseñar atajos para O(logN) vecinos, O(logN) mensajes por consulta

Abandono de los pares

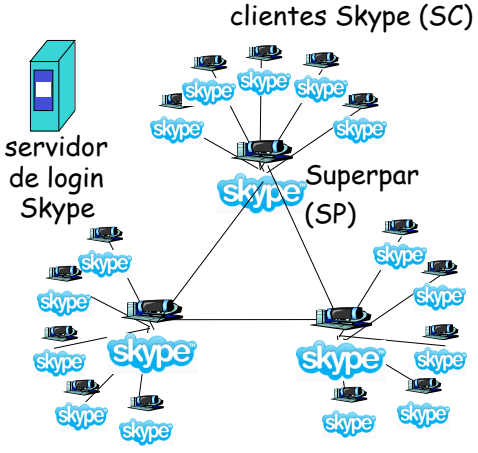


- Para manejar el abandono, hay que exigir que cada par sepa la IP de sus 2 sucesores
- cada par hace *ping* periódicamente a sus 2 sucesores, para ver si siguen vivos

- 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?

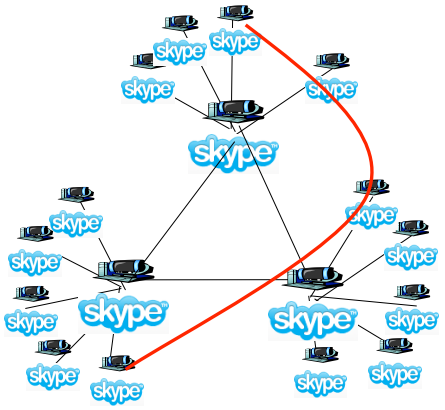
Caso de estudio P2P: Skype

- 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



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



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

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 ciente/servidor
- 2 tipos de servicio de transporte en el API de socket:
  - datagrama no fiable
  - fiable, orientado a flujo de bytes

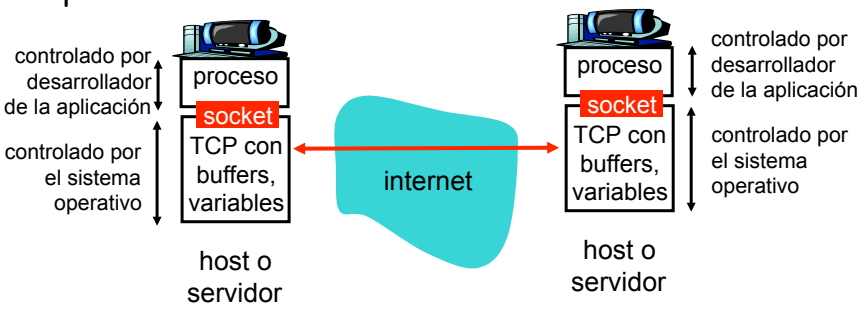
socket

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

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



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:

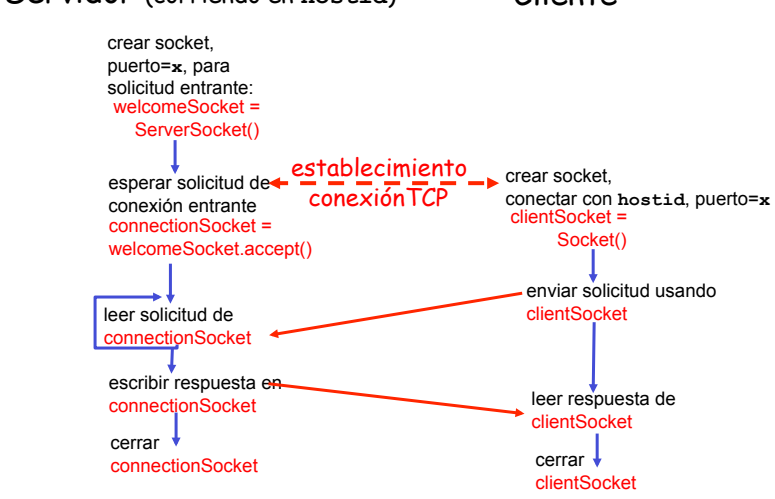
- al crear su socket TCP,
- especificar IP y puerto del proceso servidor,
- cuando el cliente crea el socket: el TCP del cliente se conecta con el del servidor

- 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

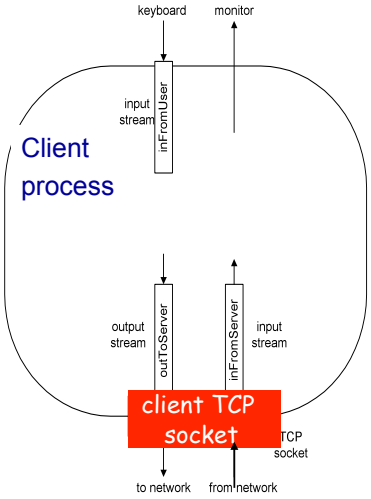
TCP proporciona transferencia de bytes fiable y en orden (un "pipe") entre cliente y servidor

interacción de sockets cliente/servidor: TCP



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



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`**)

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

nº puerto servidor

crear flujo entrada

crear objeto clientSocket de tipo Socket, conectar con servidor

crear flujo salida asociado al socket

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 (¡dejar todo limpio!)

Ejemplo: servidor Java (TCP)

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

class TCPServer {

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

     ServerSocket welcomeSocket = new ServerSocket(6789);

     while(true) {

       Socket connectionSocket = welcomeSocket.accept();

       BufferedReader inFromClient =
         new BufferedReader(new
         InputStreamReader(connectionSocket.getInputStream()));
```

crear socket recepción en el puerto 6789

espera, el método accept() del socket de recepción crea un socket nuevo

crear flujo entrada, asociado al socket

Ejemplo: servidor Java (TCP), cont

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

       clientSentence = inFromClient.readLine();

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

       outToClient.writeBytes(capitalizedSentence);
     }
   }
}
```

crear flujo salida, asociado al socket

leer línea del socket

escribir línea al socket

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

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

Programación de sockets con UDP

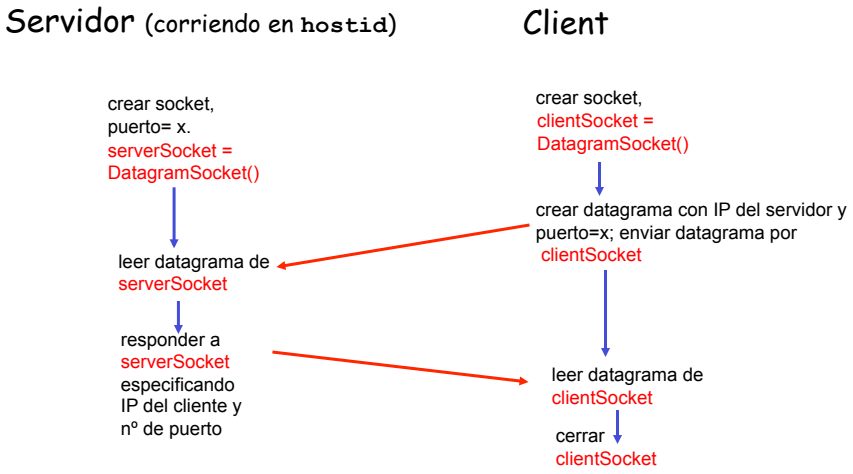
UDP: ho hay "conexión" entre cliente y servidor
- sin establecimiento de conexión
- el emisor incluye IP y puerto de destino en cada paquete
- el servidor debe extraer la IP y nº de puerto del emisor del paquete recibido

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

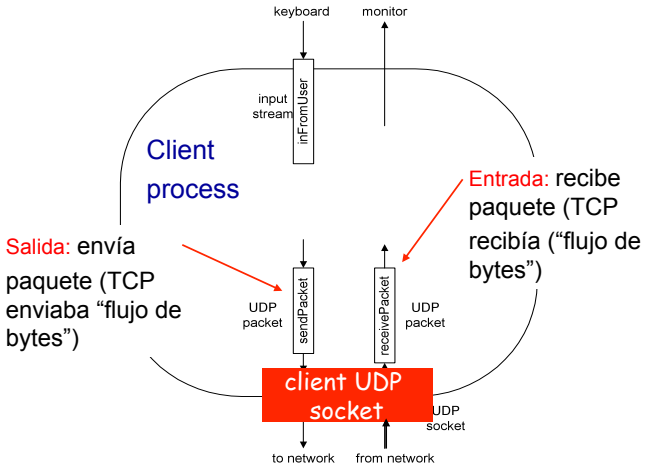
desde el punto de vista de la aplicación:

UDP proporciona transferencias no fiables de grupos de bytes ("datagramas") entre cliente y servidor

interacción de sockets cliente/servidor : UDP



Ejemplo: cliente Java (UDP)



Ejemplo: cliente Java (UDP)

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

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

      DatagramSocket clientSocket = new DatagramSocket();

      InetAddress IPAddress = InetAddress.getByName("hostname");

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

      String sentence = inFromUser.readLine();

      sendData = sentence.getBytes();
```

crear flujo entrada

crear socket cliente

traducir nombre host a IP usando DNS

Ejemplo: cliente Java (UDP), cont.

```
      DatagramPacket sendPacket =
        new DatagramPacket(sendData, sendData.length, IPAddress, 9876);

      clientSocket.send(sendPacket);

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

      clientSocket.receive(receivePacket);

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

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

crear datagrama con datos, longitud, IP, puerto

enviar datagrama al servidor

leer datagrarma del servidor

Ejemplo: servidor Java (UDP)

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

class UDPServer {
  public static void main(String args[]) throws Exception
   {
     DatagramSocket serverSocket = new DatagramSocket(9876);

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

     while(true)
      {
        DatagramPacket receivePacket =
          new DatagramPacket(receiveData, receiveData.length);
        serverSocket.receive(receivePacket);
```

crear socket para datagramas en el puerto 9876

crear espacio para datagrama recibido

recibir datagrama

Ejemplo: servidor Java (UDP), cont

```
        String sentence = new String(receivePacket.getData());
        InetAddress IPAddress = receivePacket.getAddress();
        int port = receivePacket.getPort();
                String capitalizedSentence = sentence.toUpperCase();
        sendData = capitalizedSentence.getBytes();
        DatagramPacket sendPacket =
          new DatagramPacket(sendData, sendData.length, IPAddress,
                  port);
        serverSocket.send(sendPacket);
      }
   }
}
```

obtener IP y nº de puerto del emisor

crear datagrama para el cliente

escribir datagrama al socket

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

Capítulo 2: Resumen

estudio de las aplicaciones en red: ¡completado!

- arquiteturas 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

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"