Capa de Aplicación

Mag.Ing.Miguel Solinas
miguel.solinas@unc.edu.ar

Capítulo 2: hoja de ruta

1. Principio de aplicaciones de red
   a) Arquitectura de las aplicaciones
   b) Sus requerimientos
2. Web y HTTP
3. FTP
4. Correo electrónico
   a) SMTP, POP3, IMAP
5. DNS
6. Aplicaciones P2P
7. Programación de socket con UDP y TCP

Chapter 2: application layer

Nuestra meta:

- Aspectos conceptuales y de implementación de protocolos de aplicación de red
  - Modelos de servicio de capa de transporte
  - Paradigma cliente-servidor
  - Paradigma peer-to-peer
- Aprender examinando protocolos de aplicaciones populares
  - HTTP
  - FTP
  - SMTP / POP3 / IMAP
  - DNS
- Crear aplicaciones de red
  - socket API

Algunas apps de red

- e-mail
- web
- text messaging
- remote login
- P2P file sharing
- multi-user network games
- streaming stored video (YouTube, Hulu, Netflix)
- voice over IP (e.g., Skype)
- real-time video conferencing
- social networking
- search
- …
- …

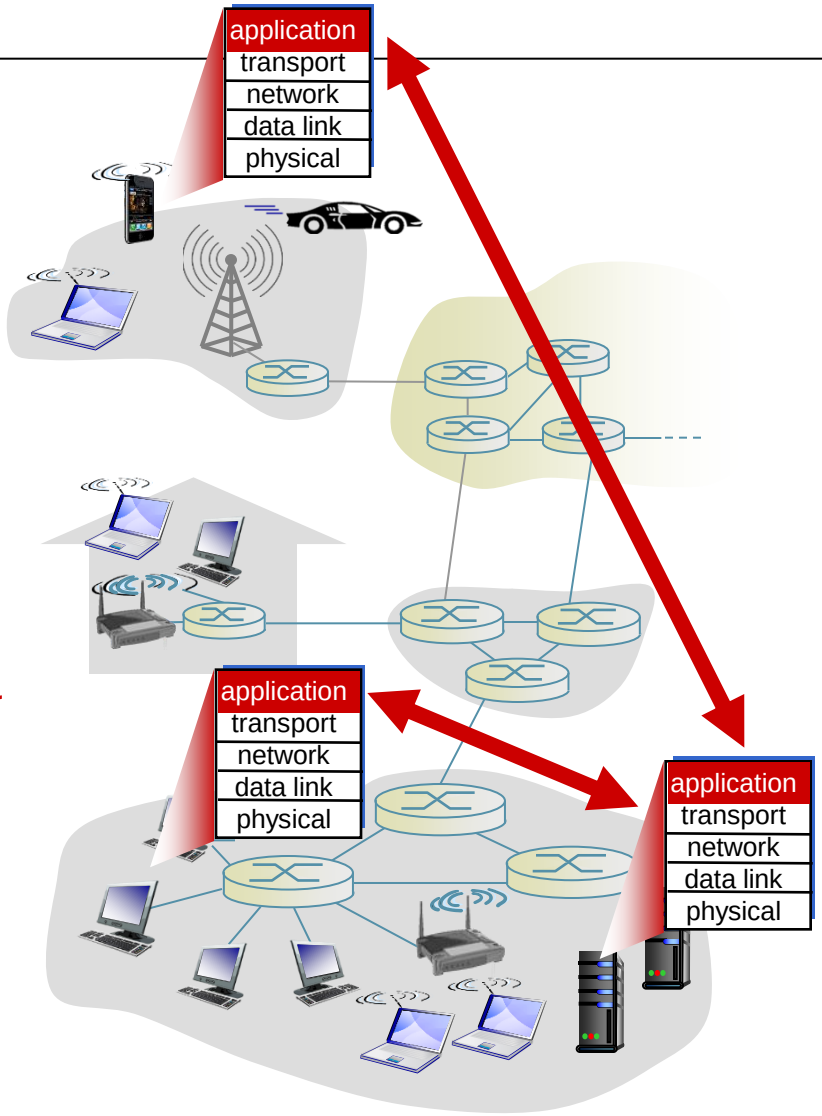
Creando apps de red

Para construir programas que :

- Corran sobre sistemas finales diferentes
- Se comuniquen sobre la red
- Ej.: web server comunicados con browsers

No se necesita escribir software para los dispositivos del núcleo de la red

- No corren aplicaciones de usuario
- Las aplicaciones sobre los sistemas finales permiten un rápido desarrollo, propagación…

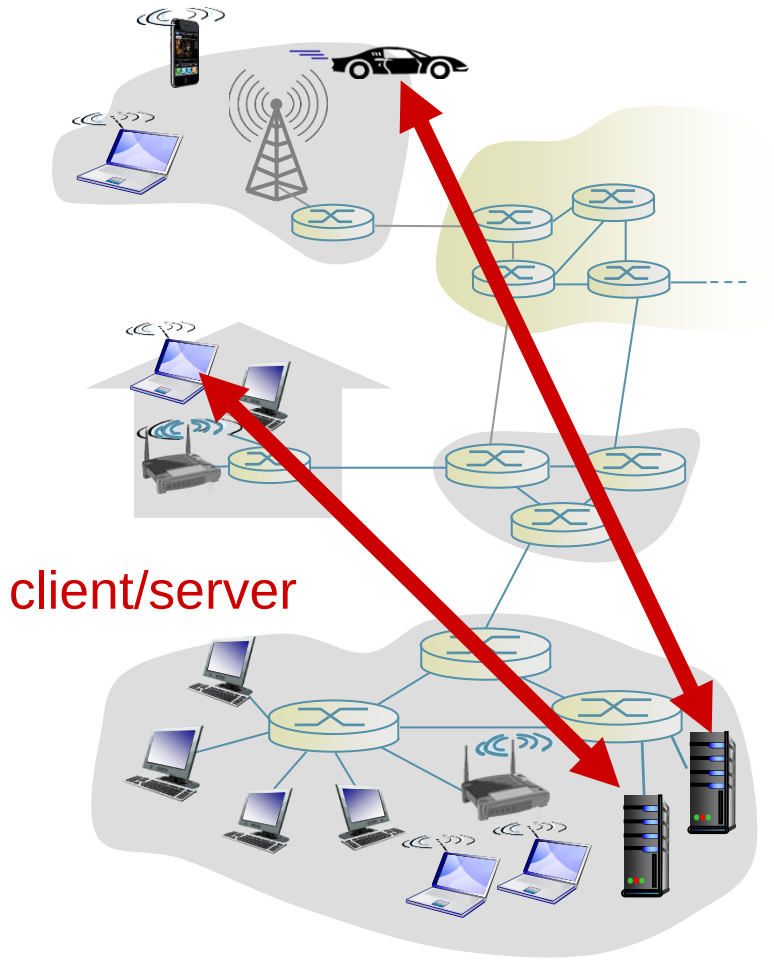


Arquitectura de las aplicaciones

Posibles estructura de las aplicaciones :

- client-server
- peer-to-peer (P2P)

Arquitectura Client – Server



Servidor:

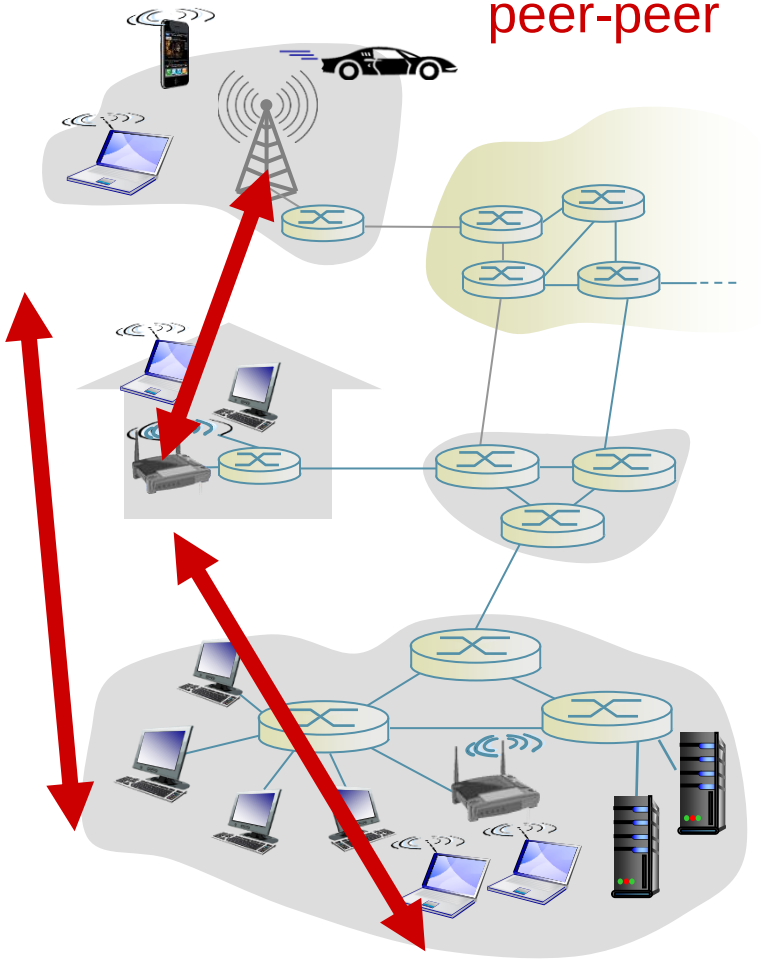
- Siempre on
- Dirección IP permanente
- Escalan en Data Centers

Cliente:

- Se comunica con el servidor
- Se puede conectar de forma intermitente
- Puede tener una IP dinámica
- No se comunican directamente entre sí

Arquitectura P2P

- *Servidor no siempre on*
- Los sistemas finales se comunican de forma arbitraria
- Los peer solicitan el servicio de otros peer, prestan servicio a solicitud de otros peer
  - *Auto escalan*: nuevos peers brindan nueva capacidad de servicios, on-demand
- Peers se conectan de forma intermitente y cambian su IP
  - Gestión compleja



Comunicación entre procesos

Proceso: programa corriendo en un host

- Dentro del mismo host, los procesos se comunican utilizando comunicación inter-process (definido por el SO)
- Procesos en hosts diferentes se comunican intercambiando mensajes

clients, servers

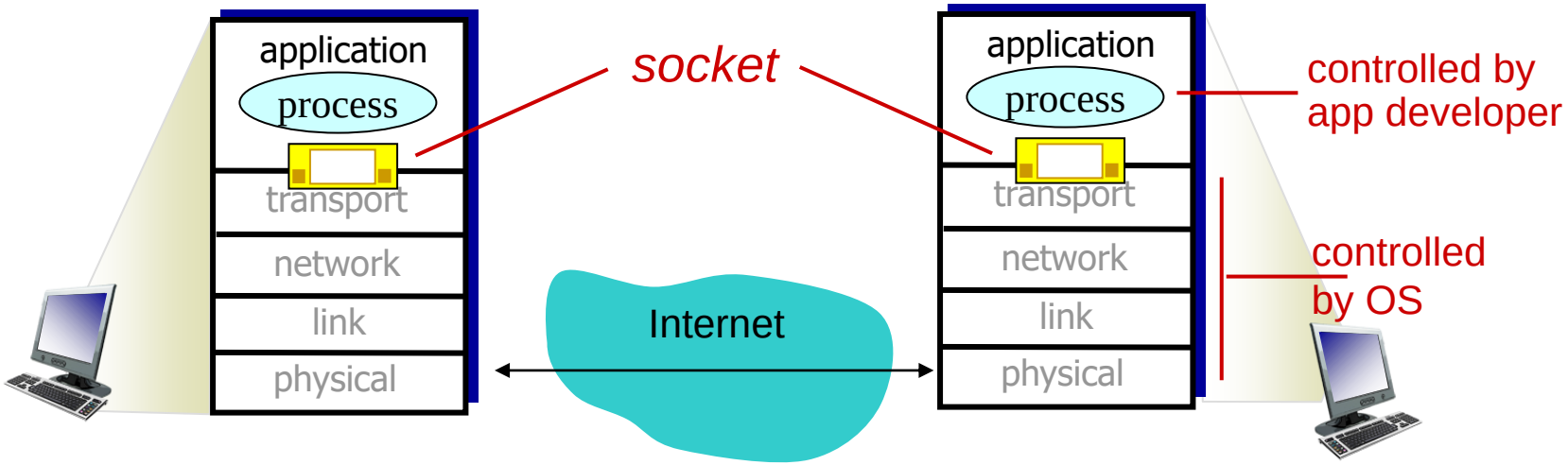
Proceso cliente: el que inicia la comunicación

Proceso server: espera a ser conectado

- Por otra parte: aplicaciones con arquitectura P2P tienen varios procesos clientes & varios procesos server

Sockets

- Proceso envía/recibe mensaje a/desde un socket
- Socket análogo a puerta
  - El proceso que envía, empuja el mensaje a la puerta
  - El proceso que envía confía en la infraestructura de transporte al otro lado de la puerta para hacer llegar el mensaje al socket asociado al proceso receptor



Proceso de direccionado

- Para recibir un mensaje, el proceso debe tener un *identifier*
- Los host tienen una dirección IP de 32-bit única
- *P:* ¿ Es la dirección IP del host, en la cual corre el proceso, suficiente para identificar al proceso ?

- Un *identifier* incluye IP address y port number asociado con el proceso en el host.
- Ej. De números de puerto:
  - HTTP server: 80
  - mail server: 25
- Para enviar un mensaje HTTP al servidor web de gaia.cs.umass.edu :
  - IP address: 128.119.245.12
  - port number: 80
- Continuará…

Un protocolo de la capa de app define

- Tipos de mensajes intercambiados,
  - Ej.: request, response
- Sintáxis de los comandos:
  - Qué campos tiene el mensajes & cómo están delimitados
- Semántica de los mensajes
  - Significado de la información en el campo
- Reglas para saber cuándo y cómo el proceso envía y responde los mensajes

Protocolos abiertos:

- Definidos en RFCs
- Interoperatividad
- Ej.: HTTP, SMTP

Protocolos propietarios:

- Ej.: Skype

¿ Qué servicios de C4 necesita una app ?

Integridad de datos

- Algunas app pueden requerir (Ej.: transferencia de archivos, transacciones web) 100% de confiabilidad en la transferencia de datos
- Otras app (Ej.: audio) pueden tolerar pérdidas

Temporización

- Algunas apps (Ej.: Internet telephony, interactive games) requieren bajos delay para ser “efectivas”

Rendimiento

- Algunas apps (Ej.: multimedia) requieren una cantidad mínima de throughput para ser “efectivas”
- Otras (“elastic apps”) Hacen uso del throughput que obtienen…

Seguridad

- encryption, data integrity, …

Requerimientos de c4s: apps comunes

Aplicación	Perdida de datos	BW	Sensible al tiempo
Transferencia de archivos	Sin pérdida	Elástica	No
Correo eletrónico	Sin pérdida	Elástica	No
Documentos web	Sin pérdida	Elástica (pocos kbps)	No
Telefonía por internet /videoconferencia	Tolerante a pérdidas	Audio: unos pocos kbps-1Mbps Video:10 kbps-5Mbps	Si: decimas de segundo
Audio/video almacenado	Tolerante a pérdidas	Idem anterior	Si: unos pocos seg.
Juegos interactivos	Tolerante a pérdidas	Unos pocos kbps-10Kbps	Si: decimas de segundo
Mensajería instantánea	Sin pérdida	Elástica	Si y No

Servicios procolos de transporte (Internet)

Servicios TCP:

- *Transporte confiable* entre el proceso que envía y el que recibe
- *Control de flujo:* El emisor no desea atorar al receptos
- *Control de congestión:* Transmisor baja pie de acelerador cuando la red está sobrecargada
- *No provee:* timing, minimum throughput guarantee, security
- *Orientado a la conexión:* se requiere un establecimiento de la conexión entre los procesos

Servicios UDP:

- *Transporte no confiable* entre el proceso que envía y el que recibe
- *No provee:* reliability, flow control, congestion control, timing, throughput guarantee, security, or connection setup,

P: ¿ Por qué ambos ? ¿ Por qué existe UDP?

Internet apps: application, transport protocols

application	application layer protocol	underlying transport protocol
e-mail	SMTP [RFC 2821]	TCP
remote terminal access	Telnet [RFC 854]	TCP
Web	HTTP [RFC 2616]	TCP
file transfer	FTP [RFC 959]	TCP
streaming multimedia	HTTP (e.g., YouTube), RTP [RFC 1889]	TCP or UDP
Internet telephony	SIP, RTP, proprietary (e.g., Skype)	TCP or UDP

Asegurando TCP

TCP & UDP

- Ninguna encryption
- Claves en texto plano se envían en socket que atraviesan internet en texto plano

SSL

- Provee encripción a las conexiones TCP
- Integridad de datos
- Autenticación de extremos

SSL está en capa app

- Las apps utilizan librerías SSL, las que “hablan” con TCP

SSL socket API

- Claves en texto plano se envían en socket que atraviesan internet encriptados
- Ver Capítulo 7

Capítulo 2: hoja de ruta

1. Principio de aplicaciones de red
   a) Arquitectura de las aplicaciones
   b) Sus requerimientos
2. Web y HTTP
3. FTP
4. Correo electrónico
   a) SMTP, POP3, IMAP
5. DNS
6. Aplicaciones P2P
7. Programación de socket con UDP y TCP

Web and HTTP

Hagamos una revisión…

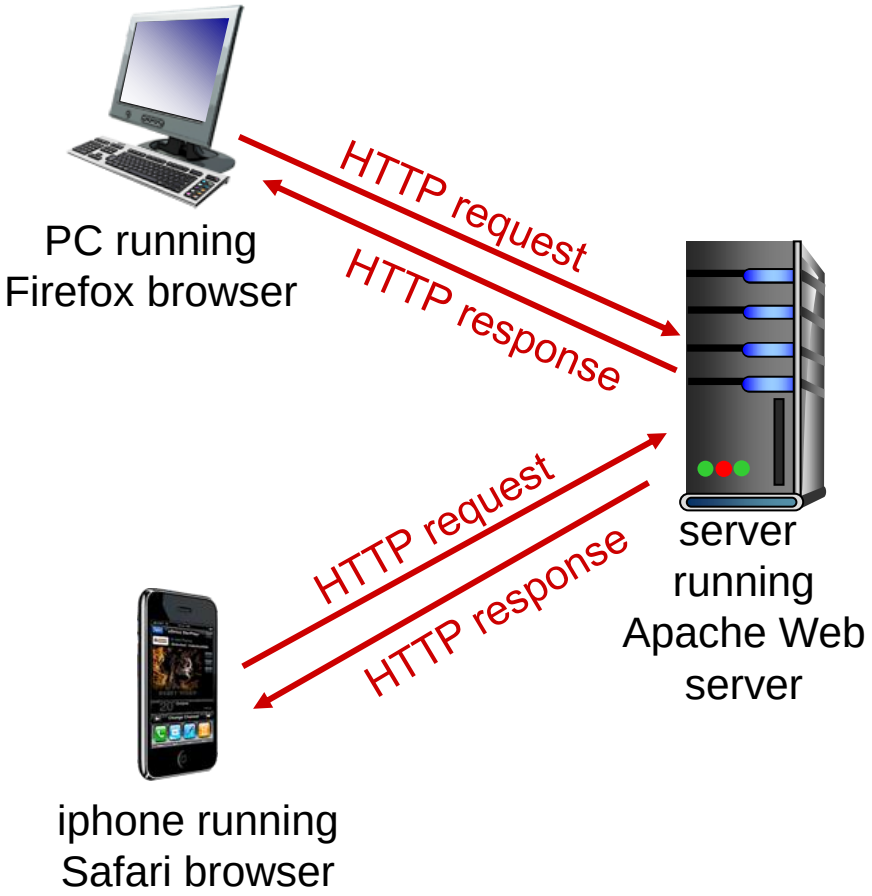
- *Páginas web* están compuestas de *objetos*
- Los objetos pueden ser archivos HTML, imágenes JPEG, applets de Java, archivos de audio,…
- Las páginas web están constituidas por un archivo base HTML que incluye varios objetos referenciados
- Cada objeto es direccionado por una URL, Ej.:

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

host name | path name

HTTP: hypertext transfer protocol

- Web es un protocolo de la capa de aplicación
- Modelo client/server
  - *client:* browser que pide, recibe, (using HTTP) y "muestra" objetos Web
  - *server:* servidor web envía (using HTTP protocol) objetoss en respuesta a peticiones



Introducción a HTTP

Utliza TCP:

- Cliente inicia una conexión TCP (crea socket) con el servidor, puerto 80
- Servidor acepta conexión TCP pedida por el cliente
- Luego mensajes HTTP (application-layer protocol messages) intercambiados entre browser (HTTP client) y servidor Web (HTTP server)
- Para finaliza, la conexión TCP se cierra

HTTP es "sin estado"

- Los servidores no mantienen información sobre las peticiones de los clientes

aparte

Protocolos que mantienen "estados" son complejos!

- Debe mantenerse un historial de navegación (estados)
- Si server/client crashes, la vista de sus estados puede ser inconsistente debe reconciliarse

Conexiones HTTP

HTTP no persistente

- Se puede enviar como máximo un objeto a través de una conexión TCP
  - Luego se cierra
- Descargar múltiples objetos requiere múltiples conexiones

HTTP persistente

- Se puede enviar múltiples objetos sobre una única conexión TCP entre cliente y servidor

HTTP No persistente

Suponga que el usuario ingresa la URL:

`www.someSchool.edu/someDepartment/home.index`

(contiene texto y referencias a 10 imagenes jpeg)

1. Cliente HTTP inicia conexión TCP (proceso) a un Servidor HTTP en el puerto 80 de www.someSchool.edu

2. Servidor HTTP en host www.someSchool.edu espera conexión TCP en el puerto 80. "Acepta" conexión y notifica al cliente

3. Cliente HTTP envía HTTP *request message* (conteniendo URL) al socket TCP. Mensaje indica que el cliente desea el objeto someDepartment/home.index

4. Servidor HTTP recive *request message*, forma un *response message* que contiene objeto solicitado, envía mensaje al socket

Time

HTTP No persistente (cont.)

5. Servidor HTTP cierra conexión TCP.

6. Cliente HTTP recibe *response message* conteniendo archivo html, muestra html. Parsea archivo, encuentra 10 objetos jpeg referenciados

7. Repite pasos 1 a 5 para cada uno de los 10 objetos jpeg.-

Time

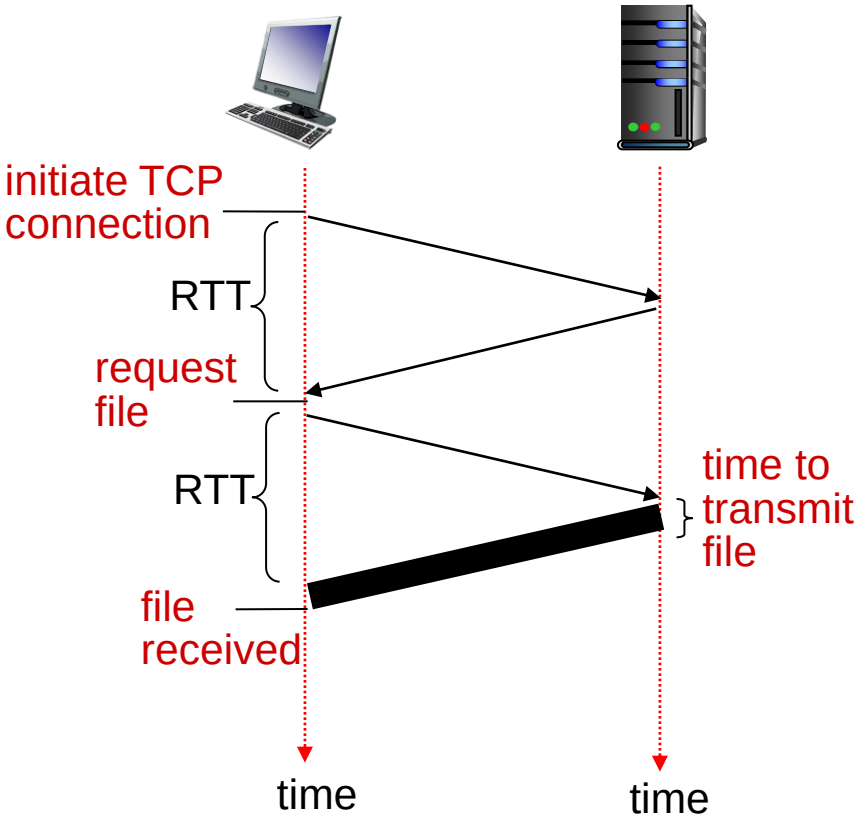
HTTP No persistente (tiempo de respuesta)

RTT (definición): Tiempo que tarda un paquete en ir y volver desde cliente a servidor

Tiempo de respuesta HTTP :

- Un RTT para iniciar conexión TCP
- Un RTT para petición HTTP y recepción de unos pocos bytes de la respuesta HTTP
- Tiempo de transmisión de archivo
- Tiempo total de una conexión HTTP no-persistente =

2RTT+ tiempo transmisión de archivo



HTTP Persistente

Aspectos de una conexión HTTP no persistente:

- Requiere 2 RTTs p/objeto
- Sobrecarga del OS para cada conexión TCP
- Browsers con frecuencia abre conexiones TCP en paralelo para buscar objetos referenciados

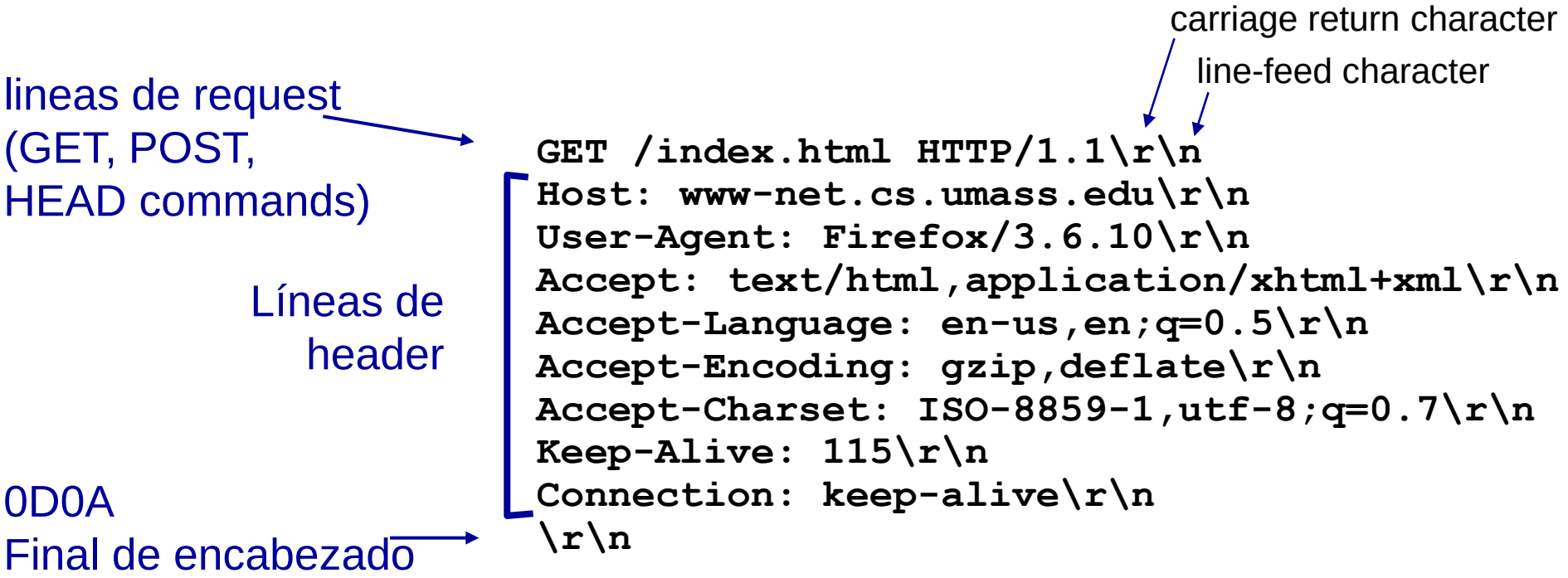
HTTP persistente:

- Servidor deja conexión abierta después de enviar respuesta
- Los mensajes HTTP siguientes entre el mismo cliente/servidor se envían sobre la conexión abierta
- Cliente envía nueva solicitud tan pronto como encuentra un objeto referenciado
- Tan poco como un RTT para c/u de los objetos referenciados

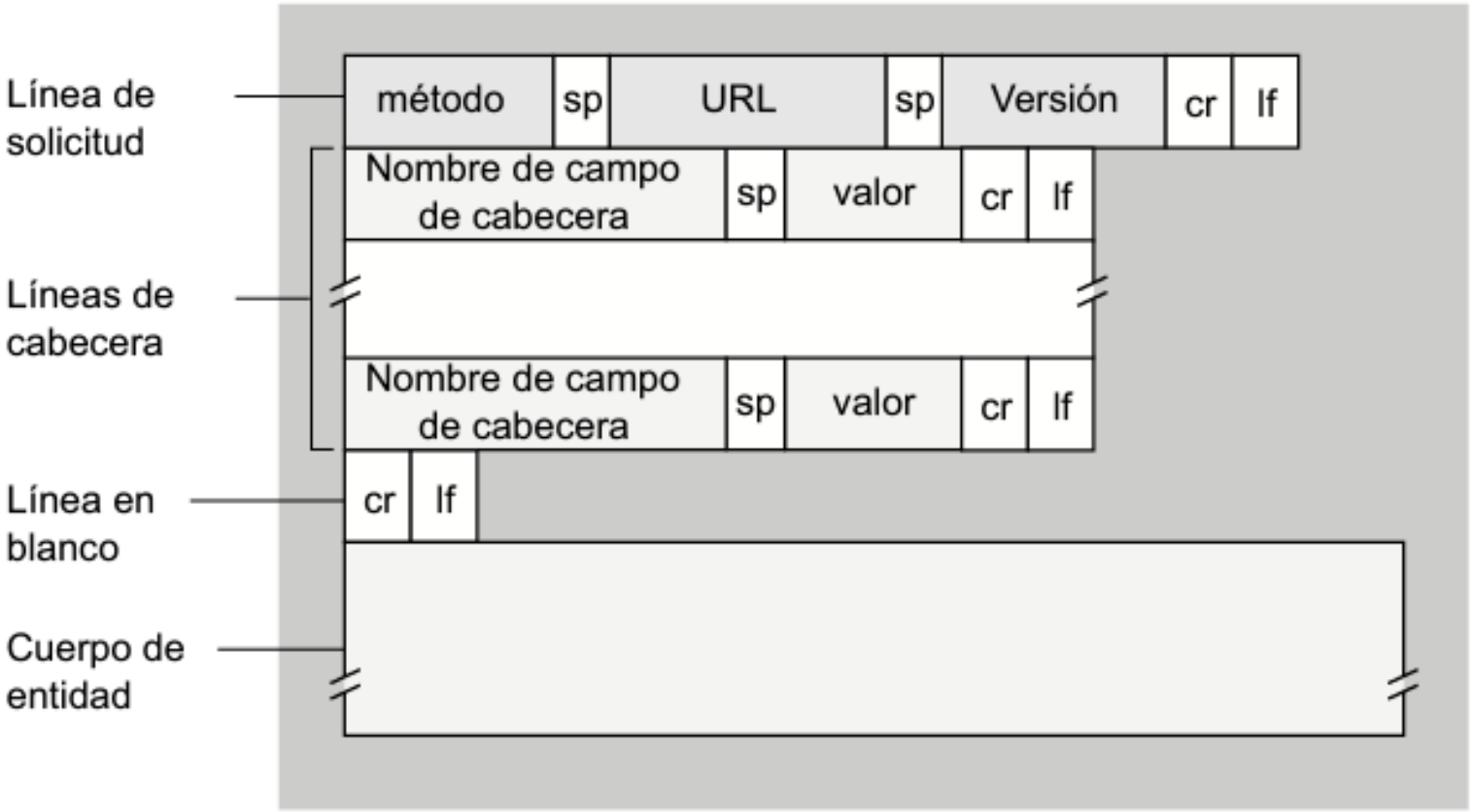
HTTP: mensaje de request

- Dos tipos de mensajes HTTP : *request*, *response*

- HTTP request message:
  - ASCII (human-readable format)



HTTP: mensaje de request (formato general)



Cargando formularios de entrada

Método POST:

- Página web habitualmente incluye entrada de formulario
- Entrada es entregada al server en el “cuerpo de entidad”

Método URL:

- Utiliza método GET
- Entrada es cargada en el campo URL de una línea de petición:

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

Tipos de métodos

HTTP/1.0:

- GET
- POST
- HEAD
  - Pide al servidor que deje el objeto solicitado sin respuesta

HTTP/1.1:

- GET, POST, HEAD
- PUT
  - Carga el archivo en el cuerpo de entidad en la ruta especificada en el campo URL
- DELETE
  - Elimina el arhivo especificado en el campo URL

HTTP: mensaje de response

Líena de status (protocol status code status phrase)

header lines

data, e.g., requested HTML file

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

HTTP: mensaje de response (status codes)

- El código de estátus aparece en la primera línea en un mensaje de respuesta de server-al-cliente.
- Algunos código de ejemplo:

200 OK

- request succeeded, requested object later in this msg

301 Moved Permanently

- requested object moved, new location specified later in this msg (Location:)

400 Bad Request

- request msg not understood by server

404 Not Found

- requested document not found on this server

505 HTTP Version Not Supported

Probando HTTP (client side) Ud. mismo

1. Telnet to your favorite Web server:

```
telnet cis.poly.edu 80
```

opens TCP connection to port 80 (default HTTP server port) at cis.poly.edu. anything typed in sent to port 80 at cis.poly.edu

2. type in a GET HTTP request:

```
GET /~ross/ HTTP/1.1
Host: cis.poly.edu
```

by typing this in (hit carriage return twice), you send this minimal (but complete) GET request to HTTP server

3. look at response message sent by HTTP server!

(or use Wireshark to look at captured HTTP request/response)

User-server state: cookies

Muchos sitios web las utilizan

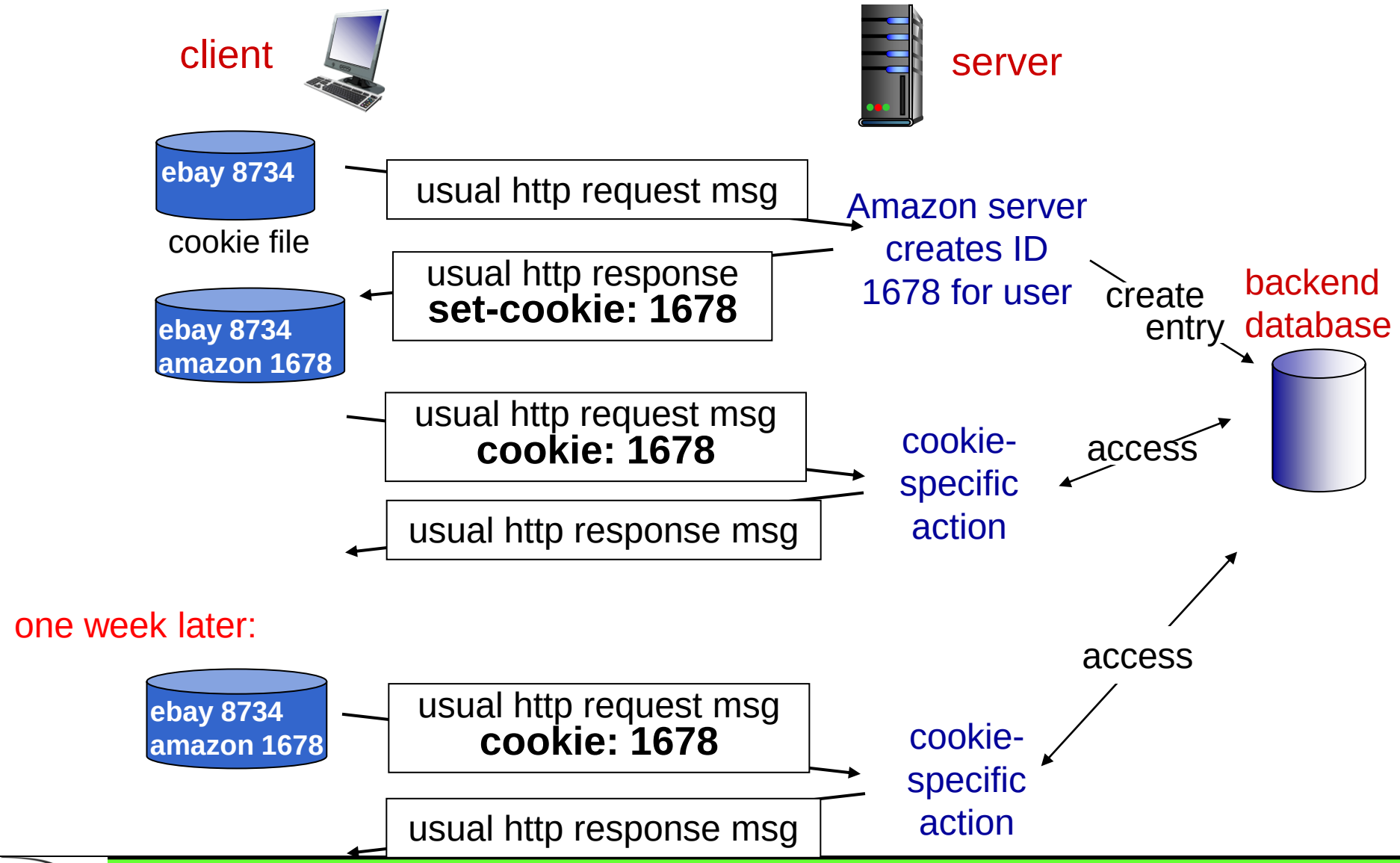
Cuantro componentes:

1) Una línea de cabecera de la cookie en el mensaje *HTTP response*
2) Una línea de cabecera de la cookie en el próximo mensaje *HTTP request*
3) Archivo de cookie guardado en el host del usuario, administrado por su navegador
4) Back-end DB en el sitio Web

Ejemplos:

- Pedro siempre tiene acceso a Internet desde su PC
- Visita un sitio específico de comercio electrónico por primera vez
- Cuando los HTTP request iniciales llegan al sitio, el sitio crea:
  - Un ID único
  - Entrada en la back-end DB para ese ID

Cookies: keeping "state" (cont.)



Cookies (continued)

¿ Para qué utilizar cookies ?:

- Autorización
- Carritos de compra
- Recomendaciones
- Estado de la sesión de usuario (Web e-mail)

cookies and privacy:

- cookies permit sites to learn a lot about you
- you may supply name and e-mail to sites

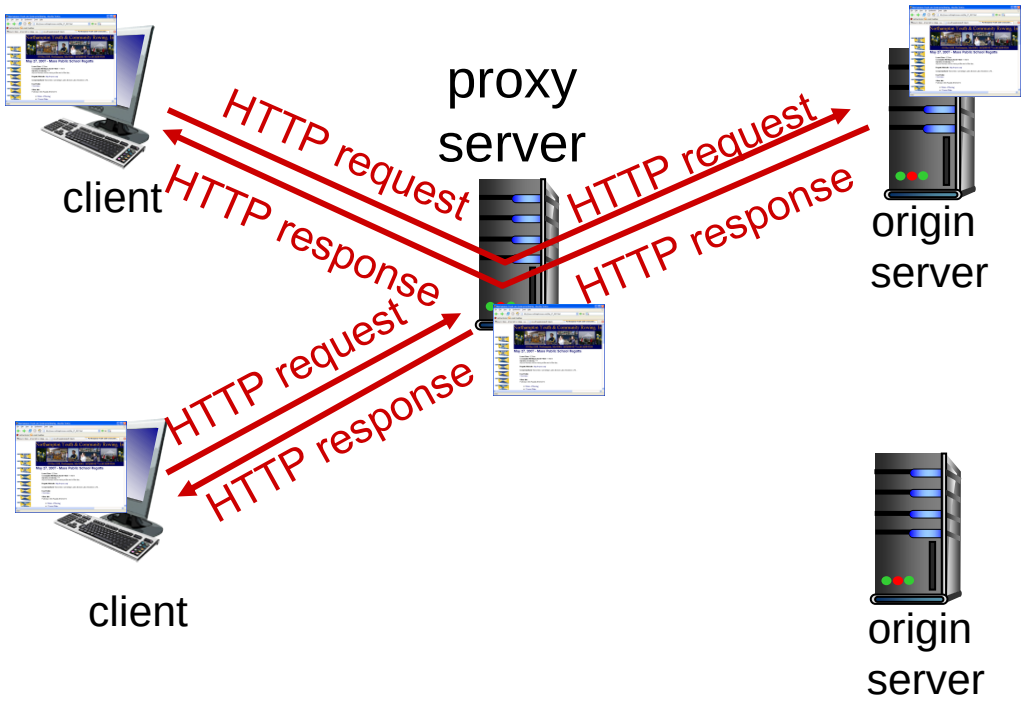
Cómo mantener el "state" de un navegador:

- Protocolo en extremos: registrando el estado sender/receiver sobre múltiples transacciones
- cookies: mantienen el estado con mensajes HTTP

Cache Web ó servidor proxy

Objetivo: Satisfacer solicitudes HTTP en nombre de un servidor web de origen. Es una entidad de red.

- Usuario configura browser: acceso web via cache
- Browser envía todas las peticiones HTTP al cache
  - Objeto en cache: cache retorna objeto
  - Sino, cache solicita objeto a servidor original, luego retorna objeto a cliente



Algo mas sobre cache web

- Cache actúa como servidor y como cliente
  - Servidor para peticiones originales del cliente
  - Cliente para servidor original
- Típicamente cache es instalado por ISP (university, company, residential ISP)

¿ Por qué un cache web ?

- Reducir tiempo de respuesta de solicitud del cliente
- Reducir tráfico en enlace de acceso de una institución
- Internet dense with caches: enables “poor” content providers to effectively deliver content (so too does P2P file sharing)

Ejemplo de cache:

Suposiciones:

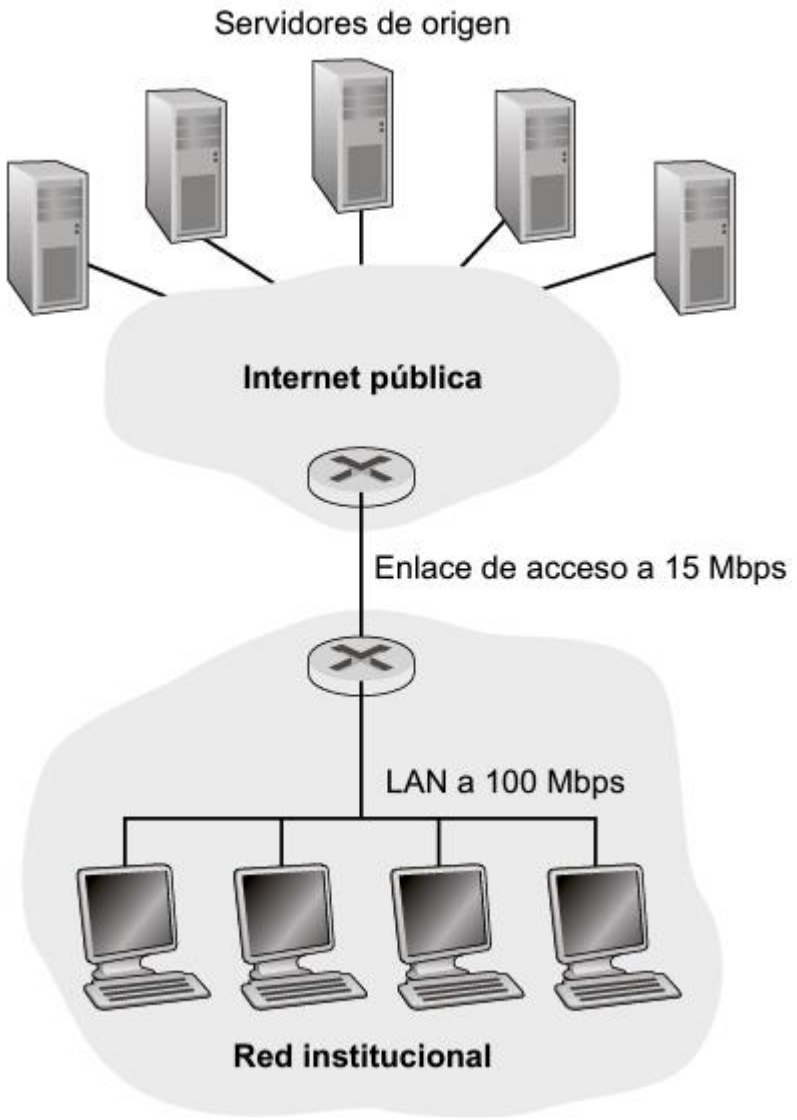
- avg object size: 1 Mbits
- avg request rate from browsers to origin servers:15/sec
- LAN utilization: 15%
- *(retardos en el orden de los ms)*

(15 req/sec) (1 Mbits/req)/(100 Mbps) = 0,15

- Access link utilization = 100%
- *(retardos en el orden de los min)*

(15 req/sec) (1 Mbits/req)/(15 Mbps) = 1

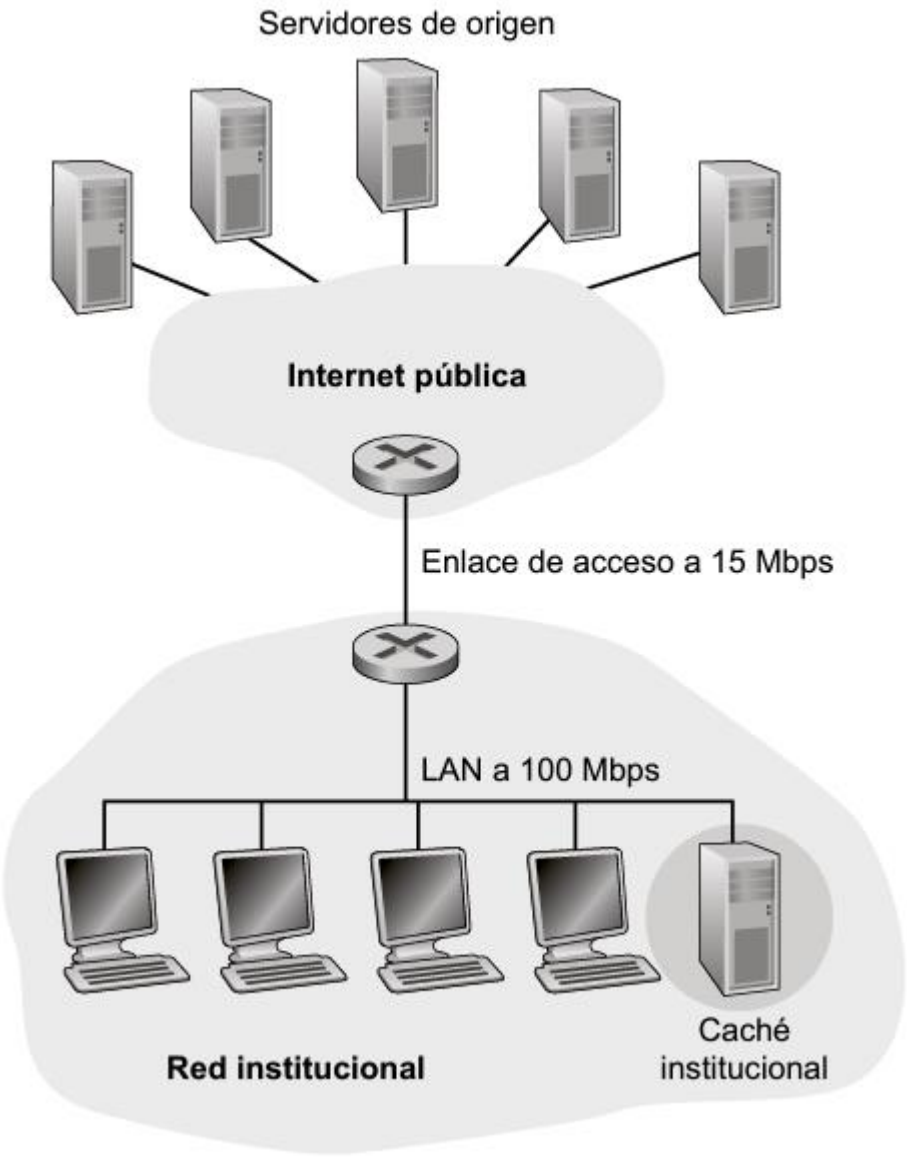
¿ SOLUCIÓN ?



Ejemplo de cache:

Suposiciones:

- Un cache web satisface entre un 20 % y un 70 % de solicitudes.
- Suponiendo un 40%, la utilización del 100% de enlace se reduce a 60%.

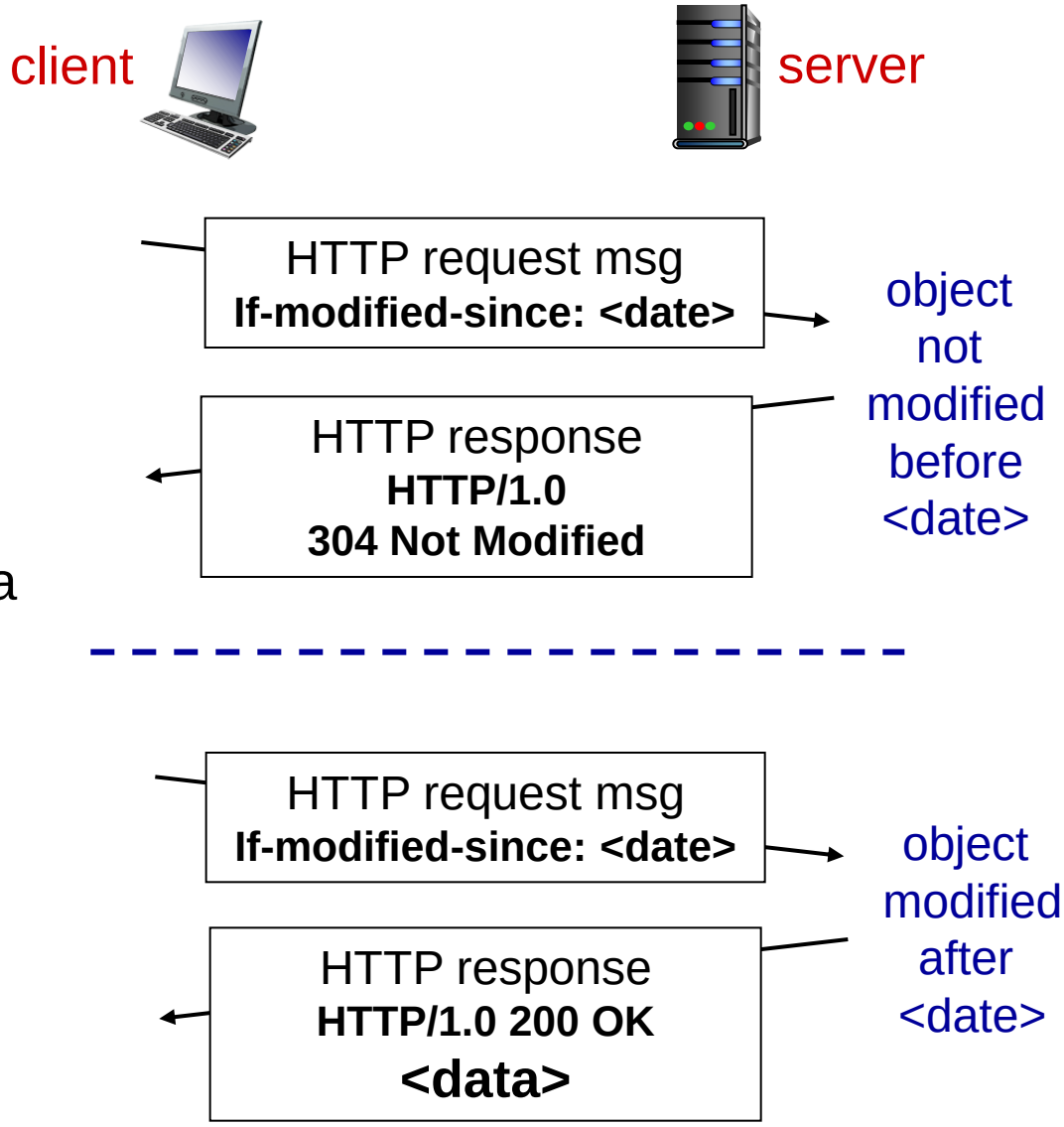


GET Condicional

- *Goal:* No enviar objeto si la memoria caché tiene una versión actualizada en caché
  - No hay retardo de transmisión de objeto
  - Baja utilización de enlace
- *cache:* Especificar la fecha de la copia en caché en HTTP request

  `If-modified-since: <date>`
- *server:* La respuesta no contiene ningún objeto si copia en caché está actualizada :

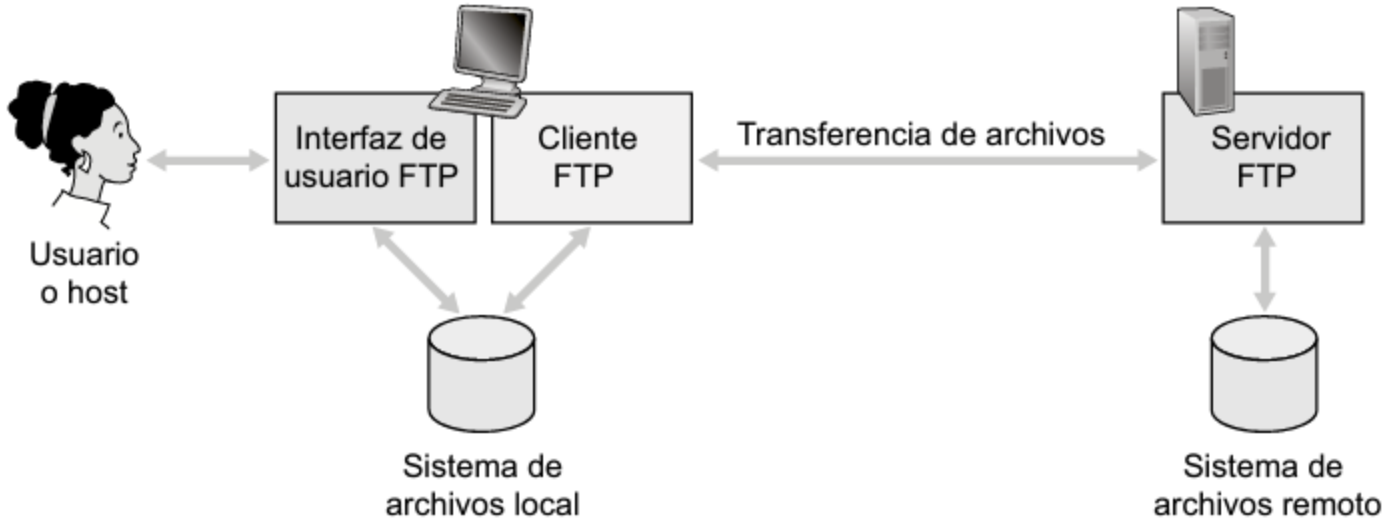
  `HTTP/1.0 304 Not Modified`



Capítulo 2: hoja de ruta

1. Principio de aplicaciones de red
   a) Arquitectura de las aplicaciones
   b) Sus requerimientos
2. Web y HTTP
3. FTP
4. Correo electrónico
   a) SMTP, POP3, IMAP
5. DNS
6. Aplicaciones P2P
7. Programación de socket con UDP y TCP

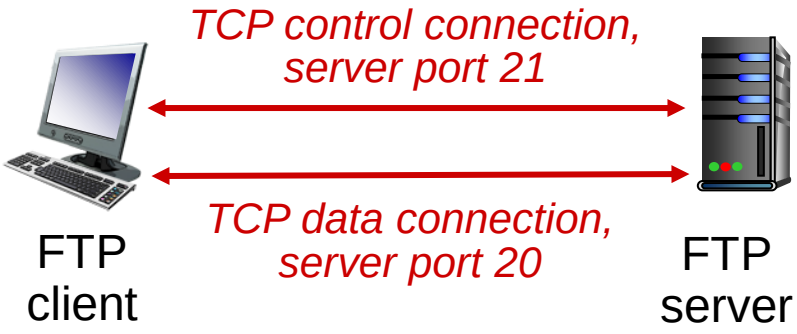
FTP: File Transfer Protocol



- Transferencia de archivos hacia/desde un host remoto
- Modelo client/server
  - *client:* lado que inicia transferencia (ya sea hacia/desde host remoto)
  - *server:* host remoto
- FTP: RFC 959
- FTP server: port 21

FTP: Separa control de conexión de datos

- Cliente FTP contacta Servidor FTP en puerto 21 + TCP
- Cliente autorizado sobre la conexión de control
- Cliente navega directorio remoto, envía comandos a través de conexión de control
- Cuando servidor recibe comandos de transferencia de archivo, *server* abre *2nd conexión* TCP para datos (archivo) hacia el cliente
- Después de transferir un archivo, server cierra la conexión de datos



- Server abre otra conexión TCP si necesita enviar otro archivo
- Conexión de control: *"out of band"*
- Servidor FTP mantiene "estado": directorio actual, autenticación temprana

FTP commands, responses

Comandos :

- sent as ASCII text over control channel
- **USER *username***
- **PASS *password***
- **LIST** return list of file in current directory
- **RETR filename** retrieves (gets) file
- **STOR filename** stores (puts) file onto remote host

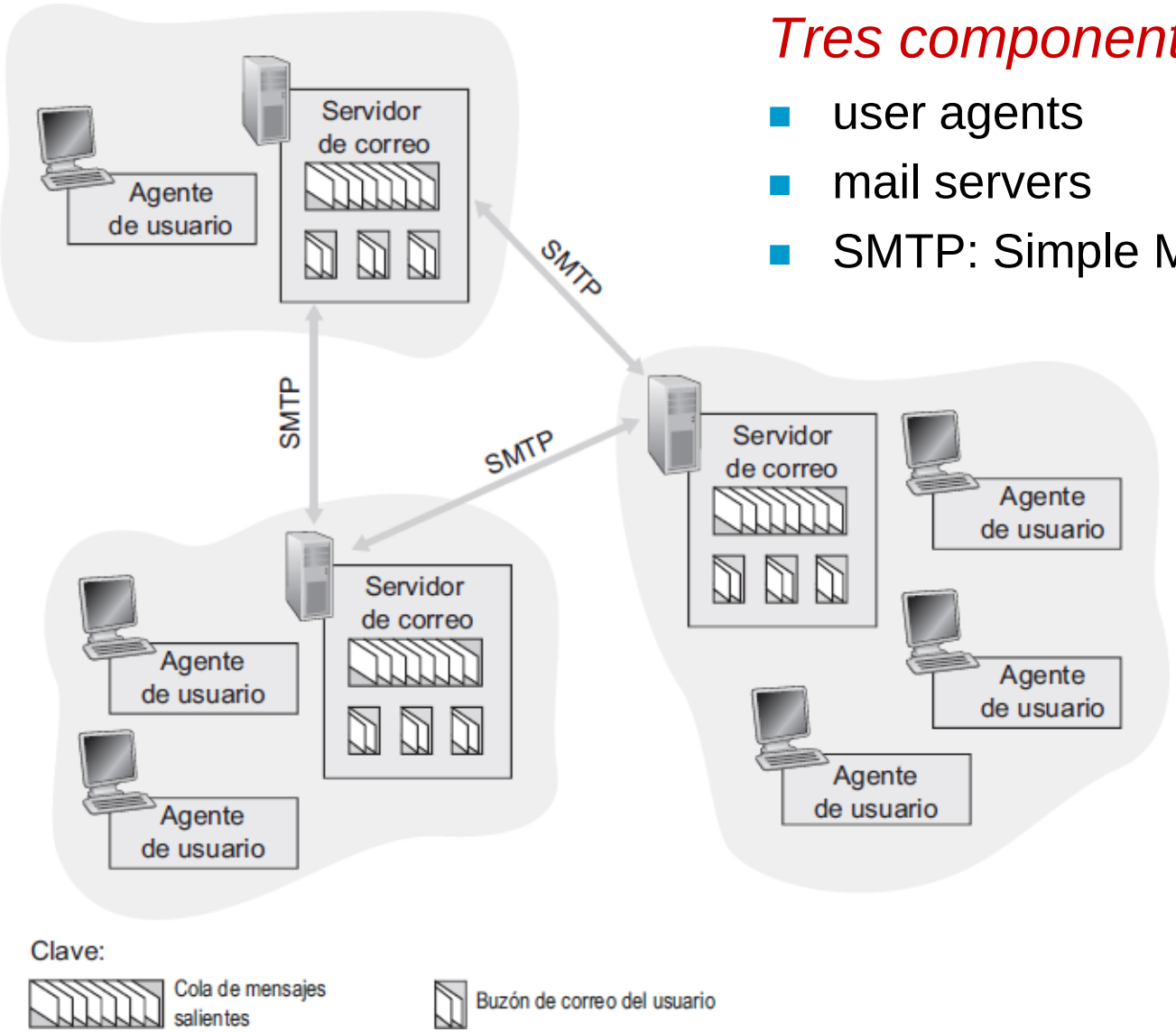
Código de retorno:

- status code and phrase (as in 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: hoja de ruta

1. Principio de aplicaciones de red
   a) Arquitectura de las aplicaciones
   b) Sus requerimientos
2. Web y HTTP
3. FTP
4. Correo electrónico
   a) SMTP, POP3, IMAP
5. DNS
6. Aplicaciones P2P
7. Programación de socket con UDP y TCP

Correo electrónico



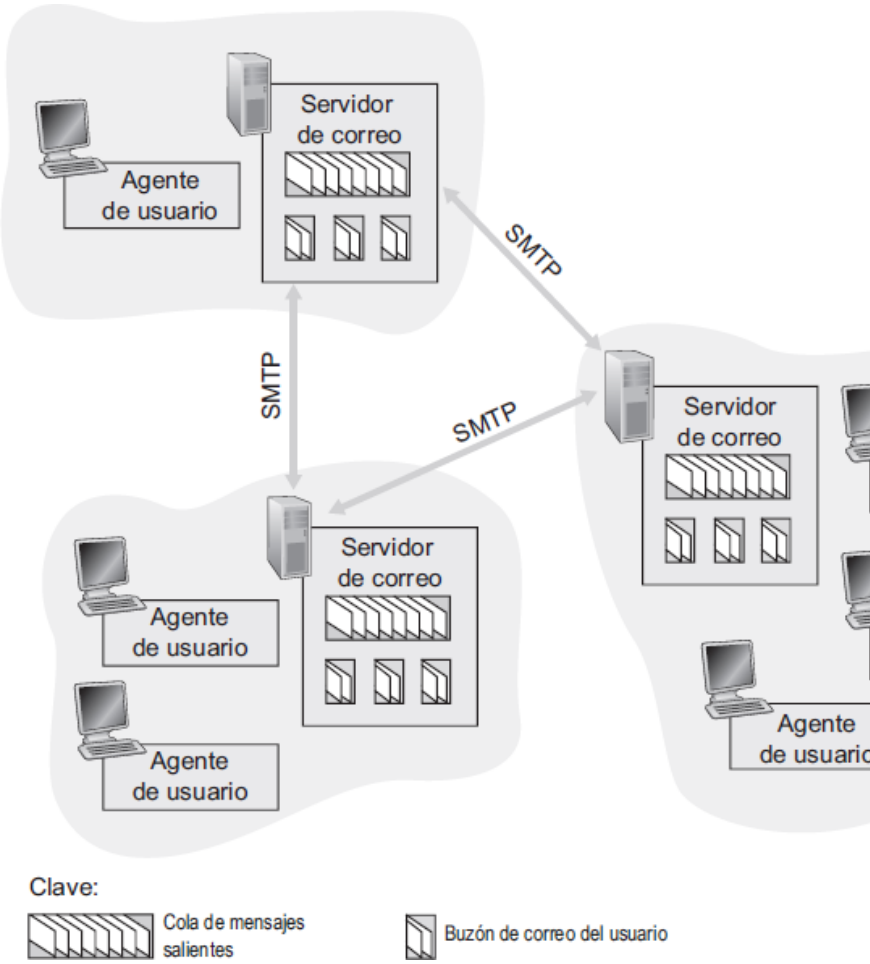
Tres componentes principales:

- user agents
- mail servers
- SMTP: Simple Mail Transfer Protocol

Correo electrónico

User Agent

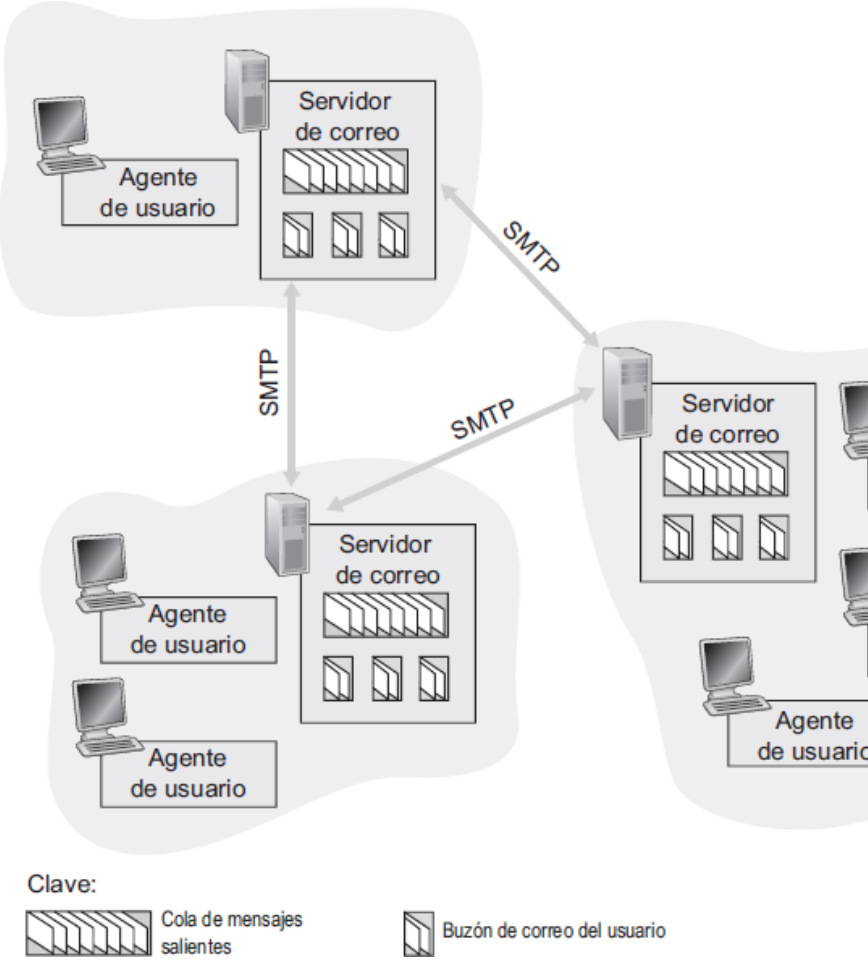
- Conocido como “mail reader”
- Composición, edición, lectura de mensajes de correo
- Ej.: Outlook, Thunderbird, iPhone mail client
- Mensajes salientes y entrantes almacenados en el servidor



Correo electrónico

Servidor de correo:

- *mailbox* contiene mensajes para el usuario
- *message queue* cola de mensajes de correo a ser enviado
- *SMTP protocol* Entre servidores de correo para enviar mensajes de correo electrónico
  - client: Enviar al servidor de correo
  - "server": recibir del servidor de correo

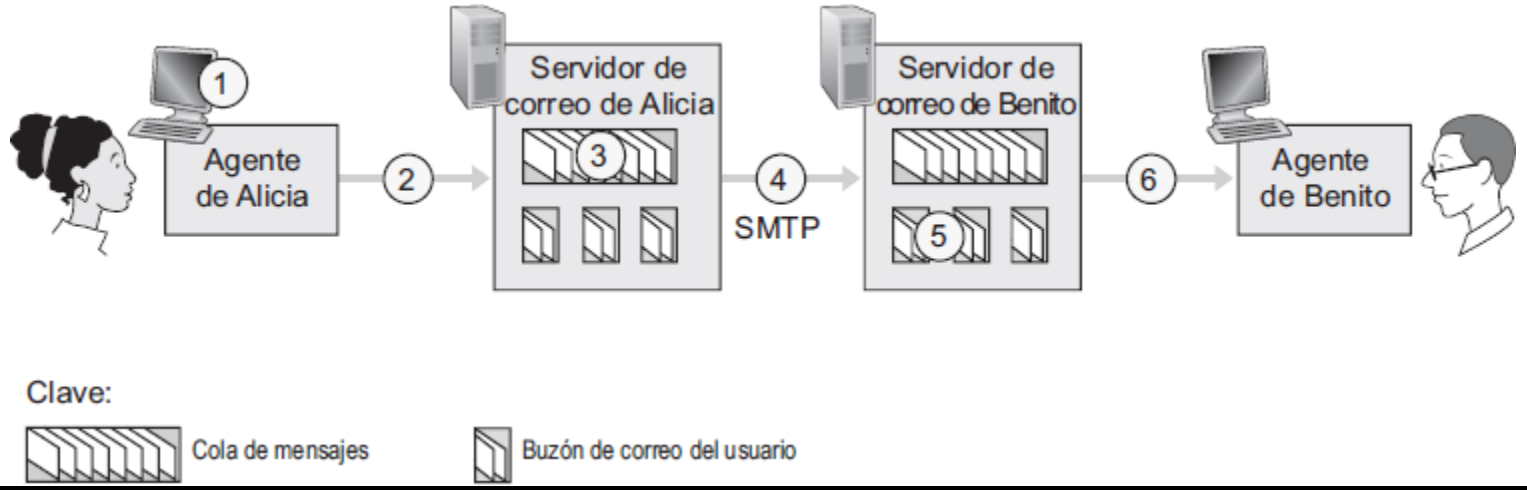


Electronic Mail: SMTP [RFC 2821]

- Utiliza TCP para una transferencia confiable de mensajes de correo entre cliente y servidor *(port 25)*
- Transferencia directa: servidor de envío al servidor de recepción
- Transferencia en tres fases
  - handshaking (greeting)
  - Transferencia de mensajes
  - Cierre
- Interacción command/response (como HTTP, FTP)
  - commands: ASCII text
  - response: status code and phrase
- Los mensajes deben utilizar ASCI de 7-bit

Escenario: Alicia envía mensaje a Benito

1) Alicia utiliza UA para escibir un mensaje «to» `bob@school.edu`
2) El UA de Alicia envía mensaje a su servidor de correo; mensaje es ubicado en cola de mensajes
3) “Cliente” SMTP de lado del Alicia abre una conexión TCP con servidor de correo de Benito
4) “Cliente” SMTP envía mensajes de Alicia sobre conexión TCP
5) Servidor de correos de Benito guarda mensajes en su mailbox
6) Benito invoca su agente para leer mensajes



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 benito@hamburger.edu ... Recipient ok
C: DATA
S: 354 Enter mail, end with "." on a line by itself
C: ¿ Te gusta el ketchup?
C: ¿ Te gustan los pickles?
C: .
S: 250 Message accepted for delivery
C: QUIT
S: 221 hamburger.edu closing connection
```

Try SMTP interaction for yourself:

- **`telnet servername 25`**
- see 220 reply from server
- enter HELO, MAIL FROM, RCPT TO, DATA, QUIT commands

above lets you send email without using email client (reader)

SMTP: palabras finales

- SMTP utiliza conexiones persistentes
- SMTP requiere mensajes (header & body) en 7-bit ASCII
- SMTP server utiliza `CRLF.CRLF` para determinar el fin de un mensaje

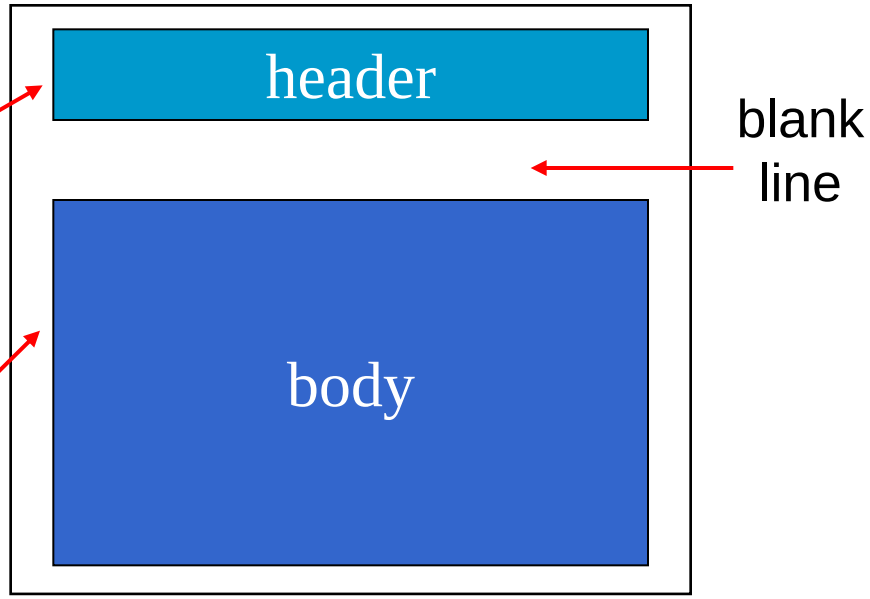
comparison with HTTP:

- HTTP: pull
- SMTP: push
- both have ASCII command/response interaction, status codes
- HTTP: each object encapsulated in its own response msg
- SMTP: multiple objects sent in multipart msg

Formato de los mensajes de correo

SMTP: Protocolo para intercambio de mensajes de correo

RFC 822: Estándar para el formato de los mensajes

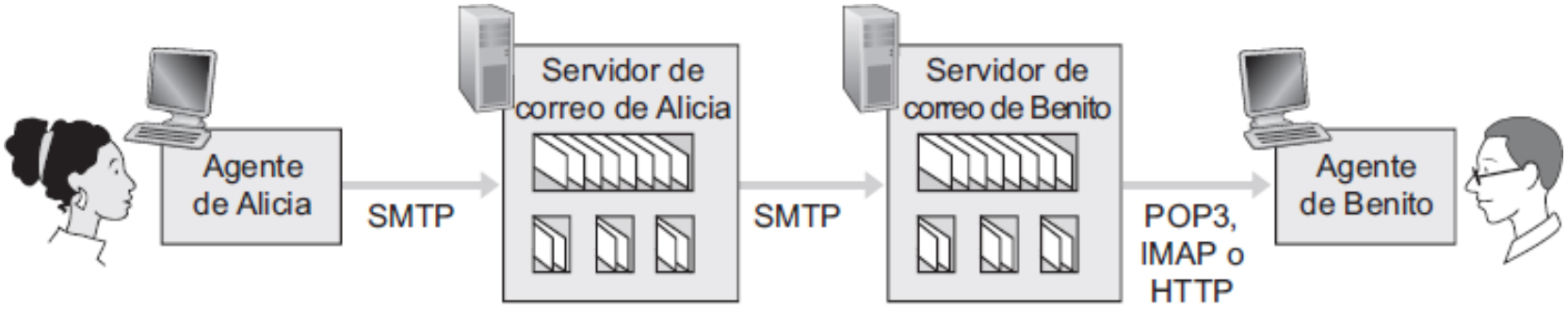


- Lineas de cabecera, Ej.:,
  - To:
  - From:
  - Subject:

(*Diferentes a los comandos SMTP*
MAIL FROM: RCPT TO:)

- Body: del “mensaje”
  - Sólo caracteres ASCII

Protocolos de acceso a correo



- SMTP: Entrega/almacenamiento al/en servidor del receptor
- Protocolos de acceso a correo: recuperan desde servidor
  - POP: Post Office Protocol [RFC 1939]: autorización, descarga
  - IMAP: Internet Mail Access Protocol [RFC 1730]: mas funcionalidades, incluyendo manipulación de mensajes almacenados en servidor
  - HTTP: gmail, Hotmail, Yahoo! Mail, etc.

Protocolo POP3

Fase autorización

- client commands:
  - **`user:`** declare username
  - **`pass:`** password
- server responses
  - **`+OK`**
  - **`-ERR`**

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

Fase transacción, client:

- **`list:`** lista los número de mensajes
- **`retr:`** recupera mensajes por nro.
- **`dele:`** elimina
- **`quit`**

```
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 (algo mas) é IMAP

Algo mas sobre POP3

- Ej. previo utiliza modo “download & delete” de POP3
  - Benito no puede volver a leer correo si cambia de cliente (casa / oficina)
- POP3 “download-and-keep”: permite copia de mensajes en diferentes clientes
- POP3 no registra estados de conexiones.

IMAP

- Mantiene todos los mensajes en un lugar: servidor
- Permite a usuario organizar mensajes en carpetas
- Mantiene estado de usuario a traves de las sesiones:
  - Nombres de carpetas y asignaciones entre ID de mensaje y nombre de carpeta

Capítulo 2: hoja de ruta

1. Principio de aplicaciones de red
   a) Arquitectura de las aplicaciones
   b) Sus requerimientos
2. Web y HTTP
3. FTP
4. Correo electrónico
   a) SMTP, POP3, IMAP
5. DNS
6. Aplicaciones P2P
7. Programación de socket con UDP y TCP

CONTINUARÁ...