Tecnología Digital IV: Redes de Computadoras

Clase 3: Nivel de Aplicación - Parte 1

Lucio Santi & Emmanuel Iarussi

Licenciatura en Tecnología Digital
Universidad Torcuato Di Tella

13 de marzo de 2025

Agenda

- Principios de las aplicaciones de red
- Protocolos de la capa de aplicación
  - Web: HTTP
    - Conexiones persistentes y no persistentes
    - Mensajes: *requests* y *responses*
    - *Cookies*
    - Cachés
    - HTTP/2 y HTTP/3

Objetivos

- Estudiar **aspectos conceptuales e implementativos** de los protocolos de aplicación
  - Modelos de servicio de la capa de transporte
  - Paradigma cliente-servidor
  - Paradigma *peer-to-peer*
- Analizar **protocolos fundamentales** de esta capa con el objeto de aprender sobre el funcionamiento y la infraestructura de Internet
- Entender cómo **programar aplicaciones de red** y cómo utilizar la API de *sockets*

Algunas aplicaciones en red

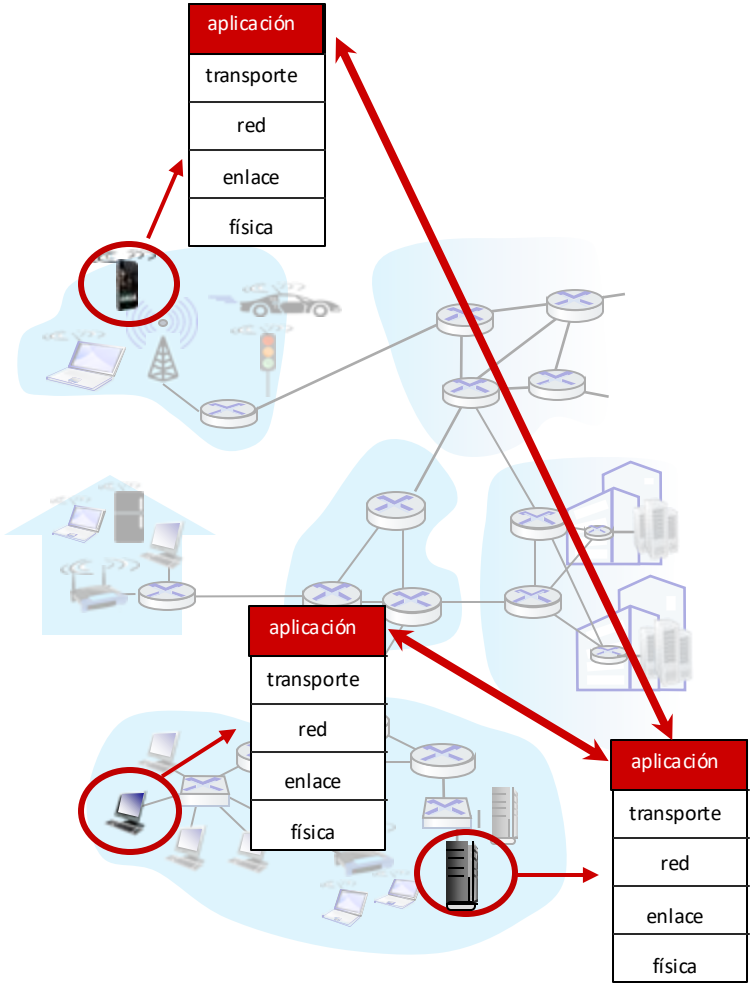
- Web
- Redes sociales (Instagram, Facebook)
- Mensajería instantánea (WhatsApp)
- E-mail
- Videojuegos
- Streaming de video (YouTube, Netflix)
- VoIP (Skype)
- Teleconferencias en tiempo real (Google Meet, Zoom)
- Buscadores (Google)
- Login remoto
- …




Algunas aclaraciones

Aplicaciones en red

- Software que corre en distintos hosts
- La comunicación se da a través de la red
- Por ejemplo, un *browser* web (Chrome) se comunica con un servidor web
- No se requiere el desarrollo de software para dispositivos en el núcleo de la red!
- Estos dispositivos (routers/switches) **no corren aplicaciones de usuario**
- Las apps en los hosts permiten agilizar el desarrollo y la adopción de las mismas



Paradigma cliente-servidor

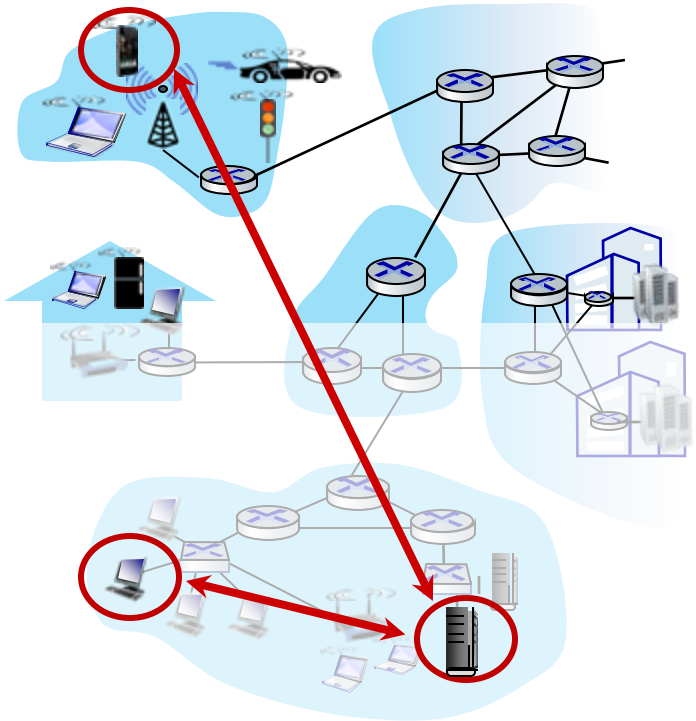
Servidor

- Host “siempre disponible”
- Dirección “permanente” (**IP**)
- Suelen estar en datacenters

Cliente

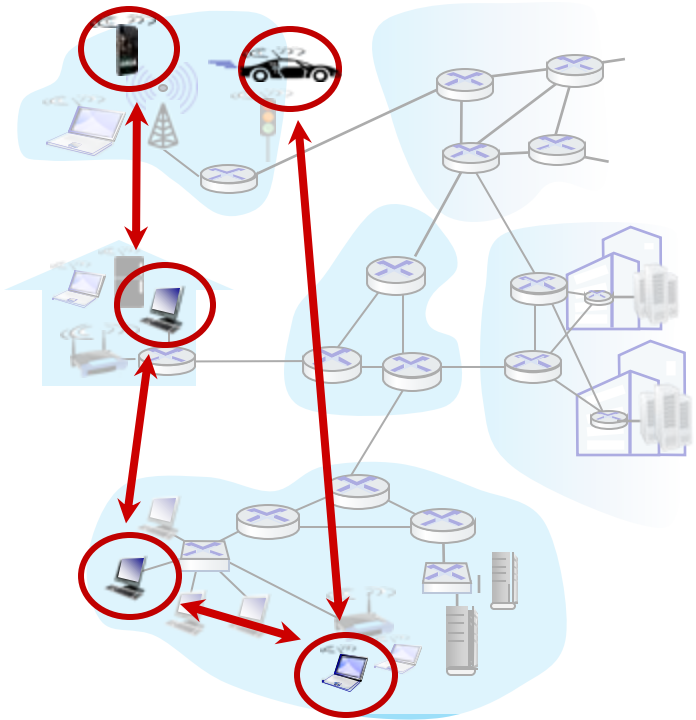
- Se comunica con los servidores
- No siempre conectado
- Puede tener direcciones dinámicas
- No se comunican entre sí

- Algunos protocolos de ejemplo: HTTP, IMAP, FTP



Paradigma *peer-to-peer* (P2P)

- No hay *un* servidor
- Comunicación directa entre hosts/*peers*
- Los peers solicitan/ofrecen servicios a otros peers
- Auto-escalabilidad
- Los peers tienen conectividad intermitente; cambian direcciones IP
- Ejemplo: intercambio de archivos P2P (e.g. BitTorrent)



Procesos

Un **proceso** es un programa en ejecución en un host

- En un mismo host, dos procesos se comunican vía mecanismos de IPC (a nivel sistema operativo)
- Los procesos en diferentes hosts se comunican vía intercambio de **mensajes**

cliente/servidor

proceso cliente: el que inicia la comunicación

proceso servidor: el que espera a ser contactado

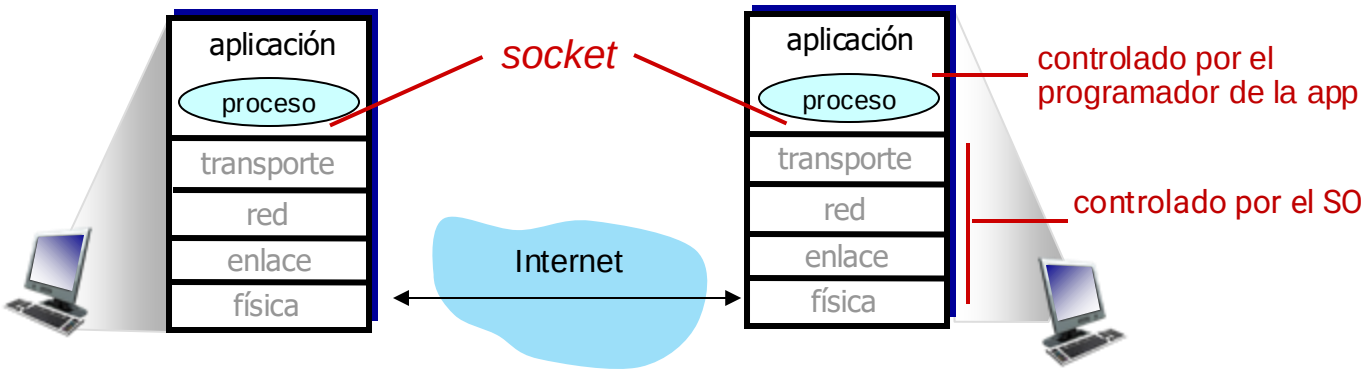
- En arquitecturas P2P hay procesos clientes y procesos servidores (en el mismo host)

Inter-Process Communication (IPC)

- **Files (Archivos)**: En un sistema operativo, un archivo es una colección de datos almacenados que pueden ser leídos o escritos por procesos. Los archivos son la forma principal de almacenamiento persistente de datos.
- **Pipes (Tuberías)**: permite a un proceso enviar datos a otro de forma secuencial. Los datos escritos en una tubería por un proceso pueden ser leídos por otro proceso, permitiendo la comunicación.
- **Memoria compartida**: Es un mecanismo de IPC que permite que múltiples procesos accedan a un mismo segmento de memoria.
- **Queues (Colas)**: Son estructuras de datos que permiten a los procesos enviar y recibir mensajes. Las colas de mensajes permiten almacenar mensajes en un orden secuencial hasta que el proceso receptor esté listo para procesarlos.
- **Sockets**: Son un punto de comunicación entre dos máquinas. Los Unix domain sockets son una variante de sockets que se usan para la comunicación entre procesos en el mismo sistema operativo. Proveen un medio eficiente para el intercambio de datos en

Sockets

- Los procesos envían y reciben mensajes a través de **sockets**
- Interfaz de comunicación que oficia como una *puerta*
  - El proceso emisor despacha mensajes por la puerta
  - Se apoya en la infraestructura de transporte para entregar los mensajes a la *puerta* del proceso receptor
  - Dos sockets involucrados: uno en cada extremo



Sockets

Existen esencialmente tres tipos de sockets:

- Datagram sockets
  - Se montan sobre el protocolo de transporte UDP (no confiable)
- Stream sockets
  - Se montan principalmente sobre TCP (confiable y orientado a conexión)
- Sockets raw
  - Permiten envío directo de paquetes IP
  - Mecanismo para implementar protocolos de transporte a nivel usuario (no kernel del SO)

Más info en la clase práctica!

Puertos y direccionamiento de procesos

- Para intercambiar mensajes, es necesario identificar procesos
- El host tiene una dirección IP de 32 bits
- **No alcanza** para identificar a los (múltiples) procesos del host

- El identificador incluye tanto la dirección IP como el puerto asociado al proceso en su host
- Algunos puertos conocidos:
  - Servidor HTTP: 80
  - Servidor de mail: 25
- Para enviar mensajes HTTP al servidor web `www.utdt.edu`:
  - Dirección IP: `104.22.27.107`
  - Puerto: 80 (ó 443)

Socket que escucha

```
import socket

server_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)  # Socket TCP
server_socket.bind(("0.0.0.0", 12345))  # IP y puerto
server_socket.listen(5)  # Escucha hasta 5 conexiones

print("Servidor esperando conexiones...")
client_socket, addr = server_socket.accept()  # Acepta conexión
print(f"Conexión establecida con {addr}")

data = client_socket.recv(1024).decode()  # Recibe datos
print(f"Mensaje recibido: {data}")

client_socket.send("Mensaje recibido".encode())  # Responde
client_socket.close()
server_socket.close()
```

Socket cliente

- import socket

- client_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM) # Socket TCP
- client_socket.connect(("127.0.0.1", 12345)) # Conecta al servidor

- client_socket.send("Hola, Servidor".encode()) # Envía mensaje
- response = client_socket.recv(1024).decode() # Recibe respuesta
- print(f"Respuesta del servidor: {response}")

- client_socket.close()

Protocolos de aplicación

Especifican:

- Tipos de mensajes
  - e.g., request, response
- Sintaxis de mensajes
  - Estructura, campos y tamaños
- Semántica de mensajes
  - Cómo interpretar los datos en los campos
- Reglas que determinan cuándo y cómo los procesos envían y responden mensajes

Protocolos abiertos

- Definidos en RFCs; todos tienen acceso a las especificaciones
- Permite **interoperabilidad**
- e.g., HTTP, SMTP

Protocolos propietarios

- e.g., Skype, Zoom

Servicios requeridos del nivel de transporte

Integridad

- Algunas apps (e.g., transferencia de archivos o transacciones web) necesitan transferencias **confiables**
- Otras (e.g., audio) pueden tolerar pérdidas

Delay

- Algunas apps (e.g., telefonía IP, videojuegos) necesitan bajo delay

Throughput

- Algunas apps (e.g., multimedia) necesitan un mínimo de throughput para ser efectivas
- Algunas pueden emplear el throughput disponible indistintamente (elásticas)

Seguridad

- Encriptación, integridad, etc.

Servicios de la capa de transporte: Internet

Servicio de TCP

- Transporte confiable
- Control de flujo: evita *saturar* al receptor
- Control de congestión: regula la tasa de emisión en función de la carga en la red
- Orientado a conexión: etapa de reserva de recursos pre-comunicación (*handshaking*)
- **Sin garantías** de tiempo, seguridad o throughput

Servicio de UDP

- Transporte no confiable y sin conexión
- No ofrece control de flujo, de congestión, garantías de tiempo, seguridad o throughput

Servicios requeridos del nivel de transporte

aplicación	pérdida	throughput	delay
descarga de archivos	no	elástico	-
e-mail	no	elástico	-
web	no	elástico	-
video en tiempo real	tolerante	audio: 5Kbps-1Mbps video:10Kbps-5Mbps	Decenas de ms
streaming audio/video	tolerante	ídem anterior	Pocos segundos
videojuegos	tolerante	Kbps+	Decenas de ms
mensajes de texto	no	elástico	-

Ejercicio!

aplicación	protocolo de nivel de aplicación	protocolo de transporte
descarga de archivos	FTP [RFC 959]	?
e-mail	SMTP [RFC 5321]	?
web	HTTP/1.1 [RFC 7230]	?
telefonía IP	SIP [RFC 3261], RTP [RFC 3550], o proprietario	?
streaming audio/video	DASH, HLS, RTSP	?
videojuegos	WOW (proprietario)	?

Ejercicio!

aplicación	protocolo de nivel de aplicación	protocolo de transporte
descarga de archivos	FTP [RFC 959]	TCP
e-mail	SMTP [RFC 5321]	TCP
web	HTTP/1.1 [RFC 7230]	TCP
telefonía IP	SIP [RFC 3261], RTP [RFC 3550], o proprietario	TCP o UDP
streaming audio/video	DASH, HLS, RTSP	TCP o UDP
videojuegos	WOW (proprietario)	UDP o TCP

Seguridad en TCP

Sockets TCP y UDP:

- No utilizan encriptación
- Passwords viajan por Internet en **texto plano**

Transport Layer Security (TLS)

- Provee conexiones TCP encriptadas
- Ofrece garantías de integridad y autenticación

TLS está implementado a nivel de aplicación

- Las apps emplean bibliotecas de TLS (que a su vez se apoyan en TCP)
- Los datos enviados al *socket* TLS viajan por Internet **encriptados**

El protocolo HTTP

La web y HTTP

- Una página web consta de **objetos** (que pueden estar almacenados en distintos servidores web)
- Un **objeto** puede ser un archivo HTML, una imagen JPG o un archivo de audio, entre otros
- Las páginas tienen un archivo HTML base que referencia otros objetos, cada uno direccionable por una **URL**:

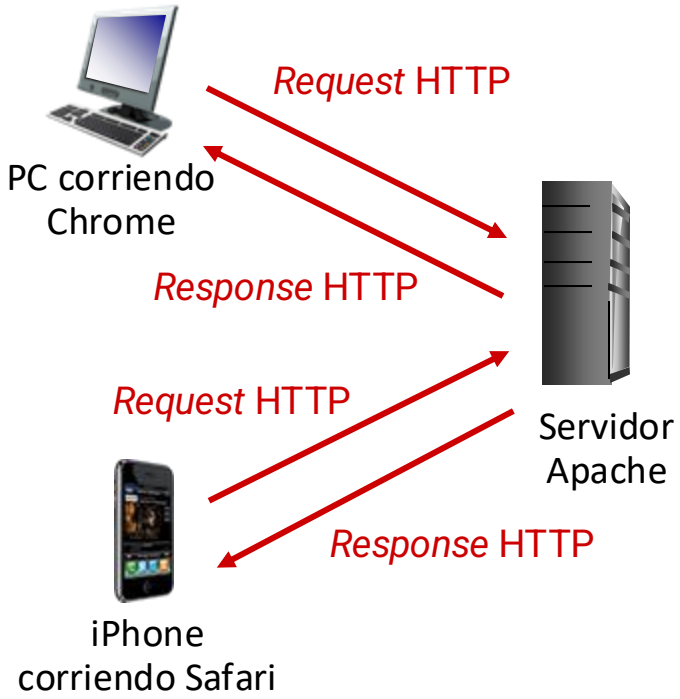
`www.utdt.edu/Images/20190422_og_campus.jpg`

nombre del host — *path*

El protocolo HTTP

HTTP: *HyperText Transfer Protocol*

- El protocolo de aplicación de la web
- Modelo cliente/servidor
  - Cliente: el navegador que solicita, recibe (vía HTTP) y "muestra" los objetos web
  - Servidor: el dispositivo que envía objetos (vía HTTP) como respuesta a las solicitudes recibidas



El protocolo HTTP

HTTP utiliza TCP

- El cliente abre un socket e inicia una conexión TCP al puerto 80 del servidor
- El servidor acepta la conexión
- Se intercambian mensajes HTTP entre el navegador (cliente HTTP) y el servidor web (servidor HTTP)
- Se cierra la conexión TCP

HTTP es *stateless*

- El servidor **no mantiene información** de requests anteriores

Los protocolos que preservan estado son **complejos**

- Ante una caída del cliente o el servidor, pueden quedar inconsistencias que deben reconciliarse

Conexiones HTTP

Conexiones no persistentes

1. Establecimiento de conexión TCP
2. Envío de a lo sumo un objeto
3. Cierre de conexión TCP

Conexiones persistentes

1. Establecimiento de conexión TCP
2. Envío de múltiples objetos
3. Cierre de conexión TCP

Con HTTP no persistente, la transferencia de múltiples objetos demanda **múltiples conexiones**

Ejemplo: HTTP no persistente

En la PC se accede a: **`www.utdt.edu/td4/home.html`**
(contiene texto y referencias a 10 imágenes JPG)

1a. El cliente inicia una conexión TCP al puerto 80 del host

1b. El servidor HTTP, a la espera de conexiones TCP, acepta dicha conexión y notifica al cliente

2. El cliente envía un *request HTTP* a través del socket (este request es un mensaje que indica que el cliente solicita el objeto **`td4/home.html`**)

3. El servidor recibe el mensaje, genera una *respuesta HTTP* con el objeto solicitado y envía este mensaje a través de su socket

tiempo

Ejemplo: HTTP no persistente

En la PC se accede a: **`www.utdt.edu/td4/home.html`**
(contiene texto y referencias a 10 imágenes JPG)

4. El servidor cierra la conexión TCP

5. El cliente recibe la respuesta con el archivo HTML, lo interpreta y descubre referencias a las diez imágenes JPG

6. Se repiten los pasos 1-5 para cada una de las diez imágenes

tiempo

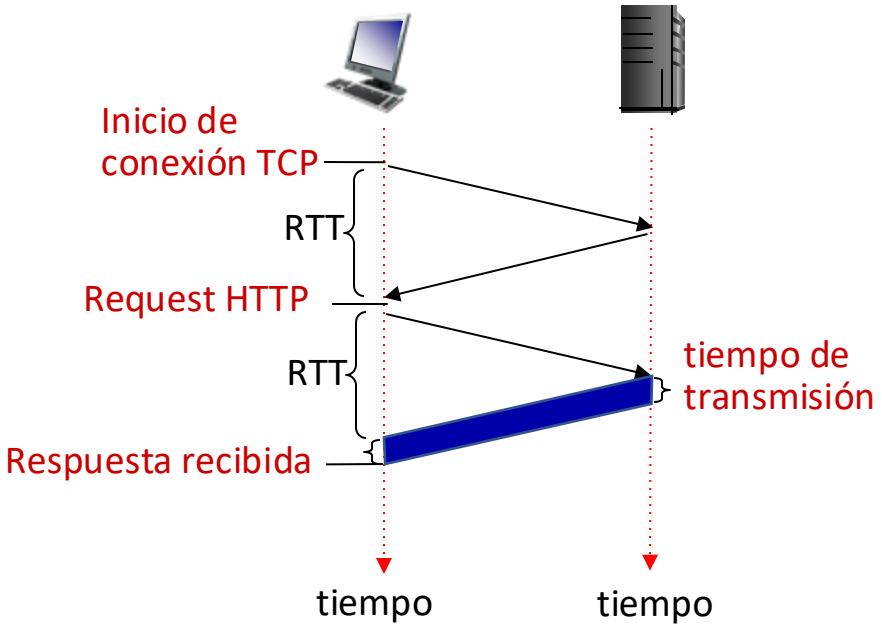
Tiempo de respuesta en HTTP no persistente

RTT (*Round Trip Time*)

- Tiempo que tarda un mensaje (pequeño) en viajar del cliente al servidor y volver

Tiempo de respuesta (por objeto):

- Un RTT para establecer la conexión TCP
- Un RTT para el request HTTP y los primeros bytes de la respuesta
- Tiempo de transmisión del objeto



Tiempo total = 2RTT + tiempo de transmisión del objeto

HTTP persistente (HTTP/1.1)

HTTP no persistente: problemas

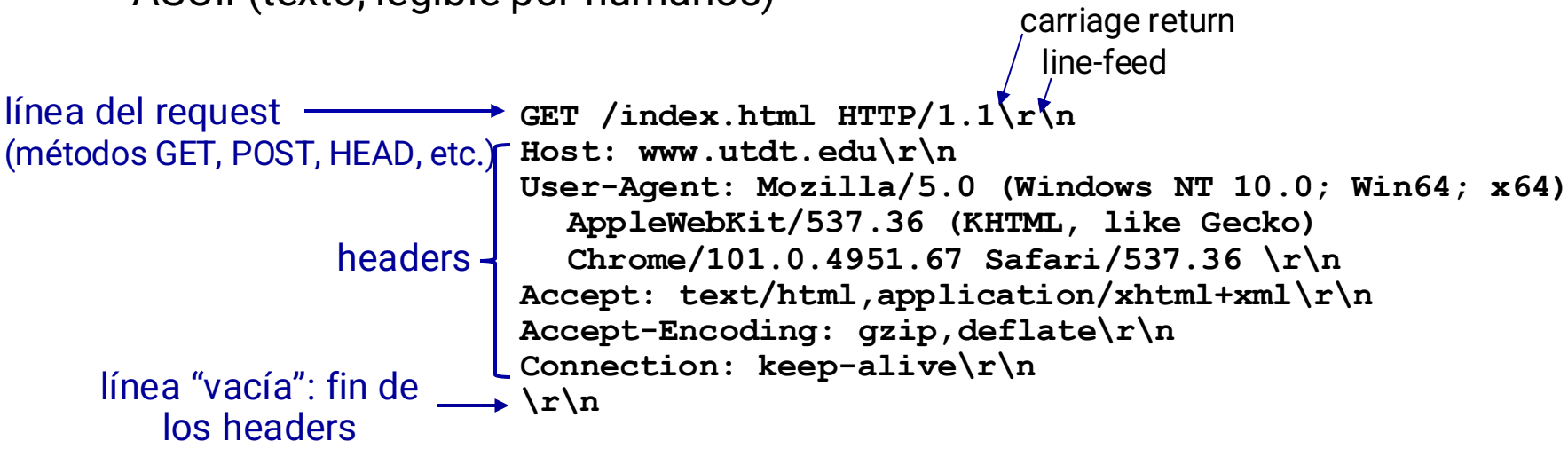
- Demanda 2 RTTs por objeto
- Sobrecarga del SO por cada conexión
- Los navegadores suelen abrir conexiones TCP en paralelo para buscar los objetos

HTTP persistente

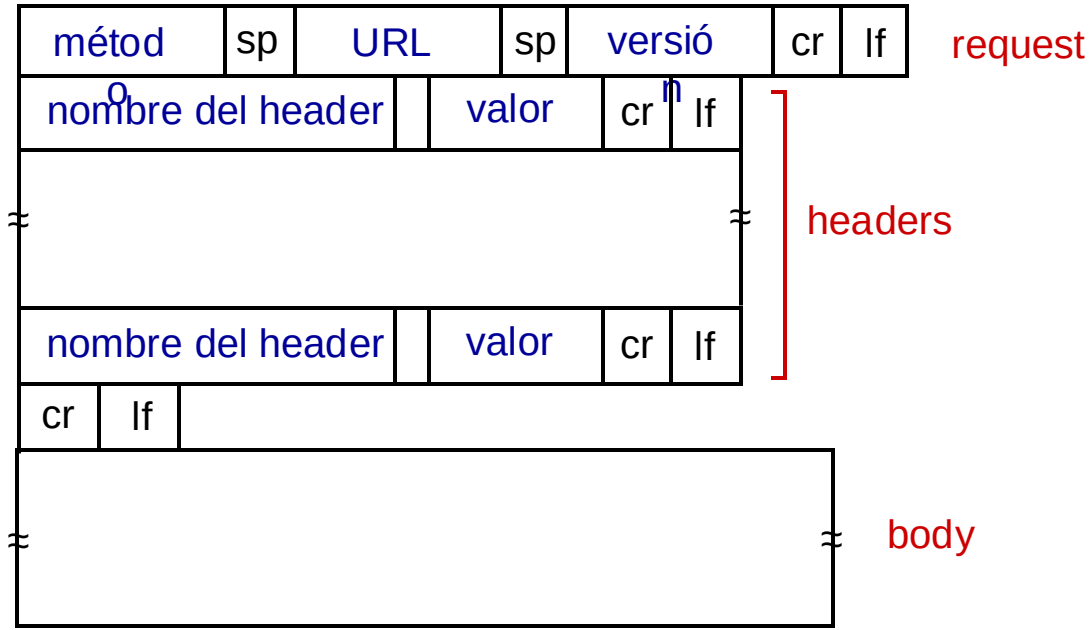
- El servidor deja la conexión abierta luego de la respuesta
- Los mensajes subsiguientes se envían sobre la misma conexión
- El cliente envía requests en cuanto encuentra objetos referenciados en el HTML
- Sólo un RTT para todos los objetos referenciados

Requests HTTP

- HTTP define dos tipos de mensajes: *requests* y *responses*
- Formato del request HTTP
  - ASCII (texto, legible por humanos)



Request HTTP: estructura



Métodos HTTP

POST

- La página suele incluir datos de forms
- El input del usuario se envía en el body del mensaje

GET

- Solicitud de datos del servidor
- Puede incluir parámetros en la URL del request (luego de un caracter `?`)

```
www.utdt.edu/td4?clase=3&teorica=true
```

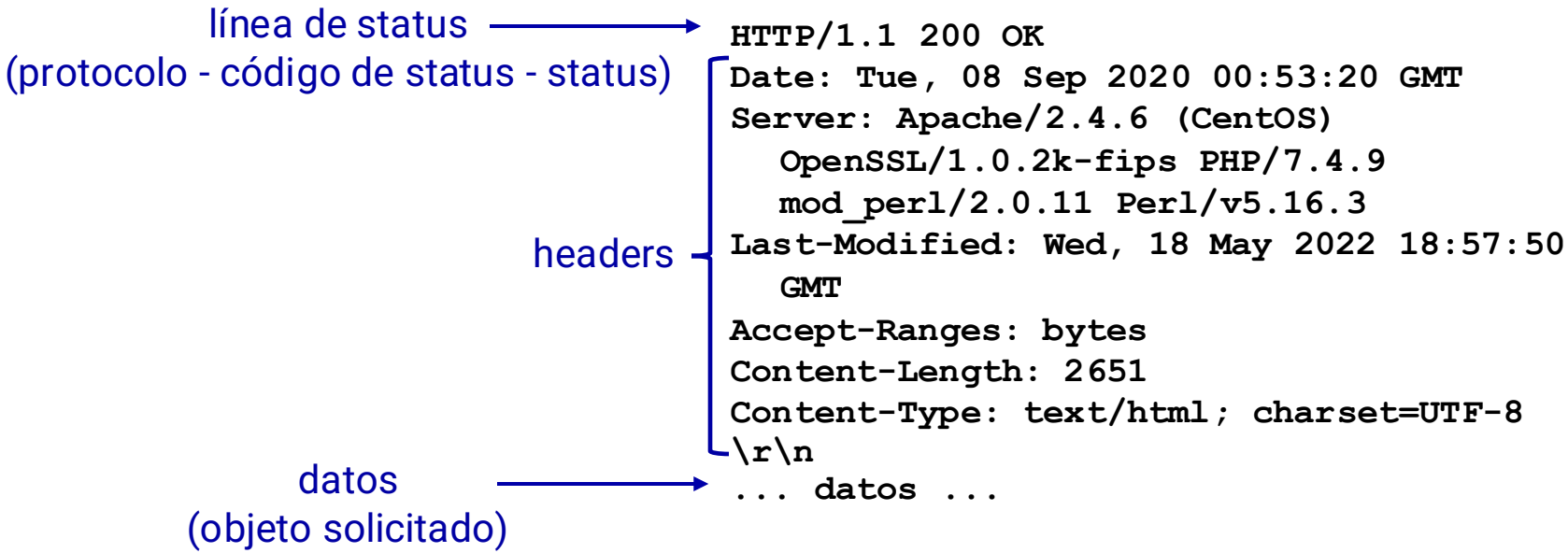
HEAD

- Solicita únicamente los headers que se devolverían en caso de acceder a la URL vía GET

PUT

- Sube un nuevo archivo al servidor
- Reemplaza al archivo de la URL con el contenido en el body del request

Responses HTTP



Códigos de status de HTTP

- Aparecen en la primera línea de las respuestas del servidor HTTP
- Algunos ejemplos:

200 OK
- El request fue exitoso; el objeto se transmite en el mensaje

301 Moved Permanently
- El objeto solicitado cambió de ubicación (la misma se indica en el header `Location:`)

400 Bad Request
- El mensaje del request no pudo ser interpretado por el servidor

404 Not Found
- El objeto solicitado no pudo encontrarse en el servidor

505 HTTP Version Not Supported