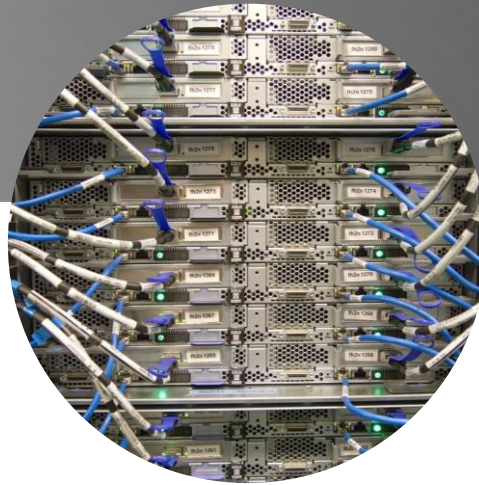


Redes de computadores 2022 -1 (11310052)

David Felipe Celeita Rodriguez



Universidad del
Rosario

Escuela de Ingeniería,
Ciencia y Tecnología

- "You lose your curiosity when you stop learning."

- Katherine Johnson



Universidad del
Rosario

Escuela de Ingeniería,
Ciencia y Tecnología

Programa

Fecha (Sesión)	Tema
Sesión 1-2 24 Ene – 28 Ene	Introducción a redes de computadores Parte 1
Sesión 3-4 31 Ene – 4 Feb	Introducción a redes de computadores Parte 2
Sesión 5-6 7 Feb – 11 Feb	Capa de aplicación Parte 1
Sesión 7-8 14 Feb – 18 Feb	Capa de aplicación Parte 2
Sesión 9-10 21 Feb – 25 Feb	Capa de transporte Parte 1
Sesión 10 21 Feb – 25 Feb	PARCIAL 1





Capítulo 2: Capa de aplicación

Contexto:

Fundamentos de aplicaciones en redes

Servicios Web y HTTP

E-mail, SMTP, IMAP

(Domain Name System) DNS

Aplicaciones P2P (Peer-to-peer)

Streaming de video y redes de distribución de contenido

Programación de socket con UDP y TCP (Taller con Python)

Taller Packet Tracer – Network Services

CAPA DE APLICACIÓN: GENERALIDADES

Objetivos:

- Aspectos conceptuales y de implementación de los protocolos de la capa de aplicación.
- Modelos de servicio de la capa de transporte
 - paradigma cliente-servidor
 - paradigma de P2P
- Aprender acerca de los protocolos examinando la infraestructura de la capa de aplicación:
 - HTTP
 - SMTP,
 - IMAP
 - DNS
 - Sistemas de transmisión de video, CDNs
 - Programación de aplicaciones de red (Socket API)

Apps/Servicios en redes

- Redes sociales
- Web
- Mensajes de texto
- E-mail
- Video juegos multijugador
- Streaming de video (YouTube, Hulu, Netflix)
- Compartir archivos (P2P)
- Voz sobre IP (e.g., Skype)
- Video-conferencias en tiempo real(e.g., Zoom)
- Motores de búsqueda en Internet
- Acceso remoto

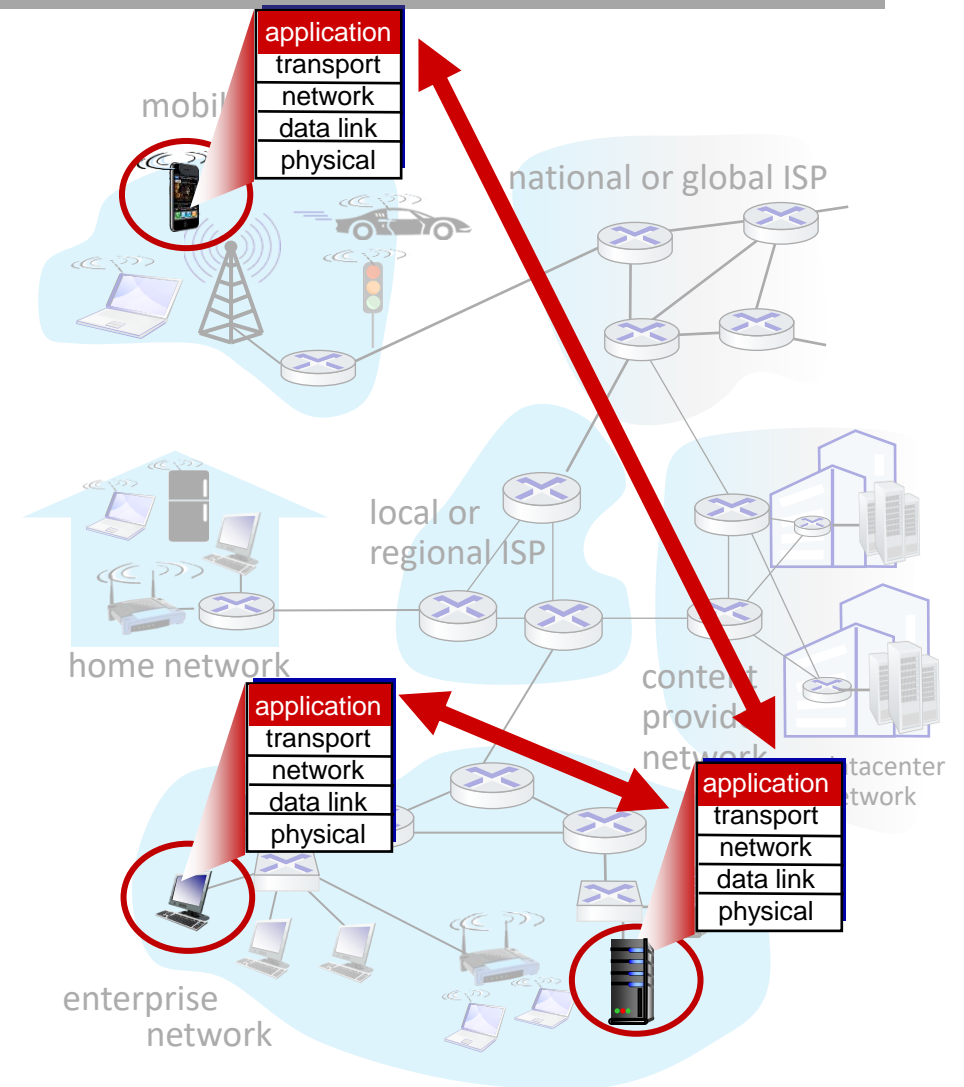
OTROS?

CREANDO UNA APP PARA LA RED

Basado en: Jim Kurose, Keith Ross Pearson, 2020 Slides

Escribir un programa:

- Se ejecuta en diferentes sistemas terminales
- Se comunica sobre la red
- No se escriben programas para ejecutar en dispositivos del núcleo de la red (red central) ya que estos **no ejecutan aplicaciones de usuario**
- Las aplicaciones en sistemas terminales permiten un rápido desarrollo y propagación de aplicaciones



PARADIGMA CLIENTE-SERVIDOR

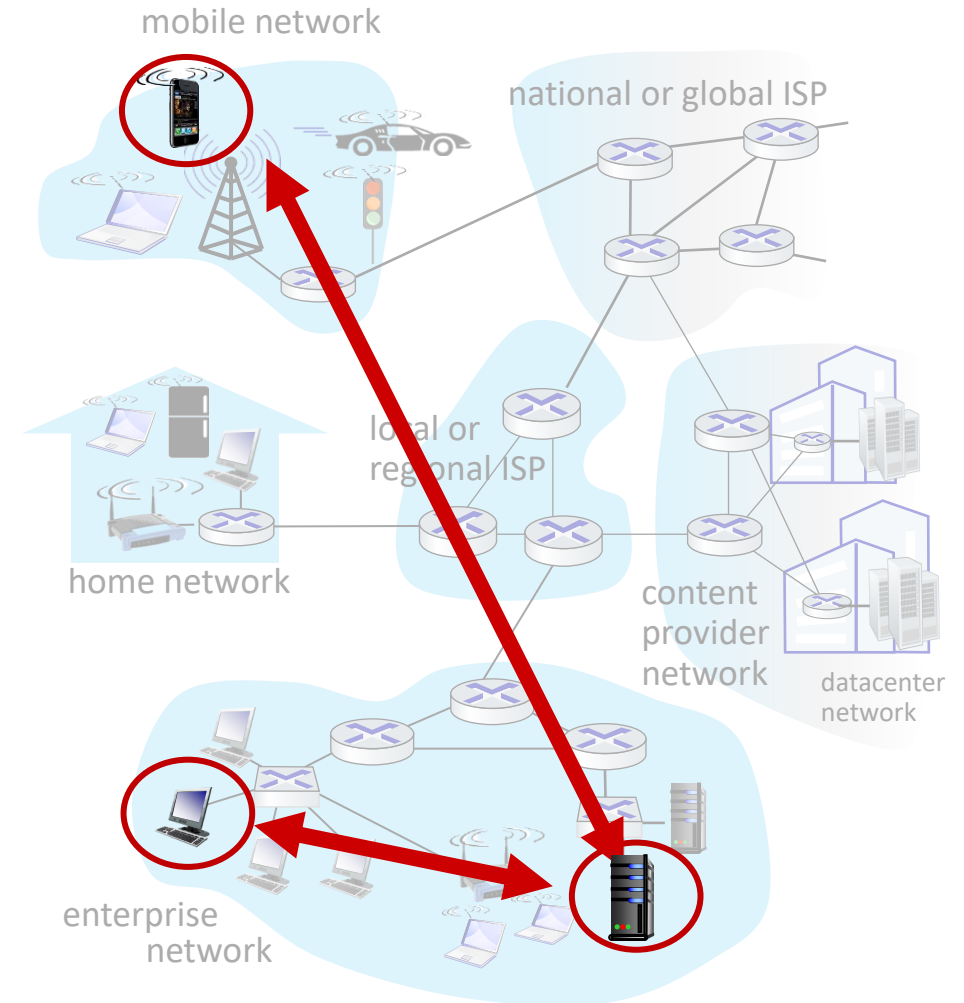
Basado en: Jim Kurose, Keith Ross Pearson, 2020 Slides

Servidor:

- Host siempre activo
- Dirección IP permanente
- A menudo en datacenters (escalable)

Cliente:

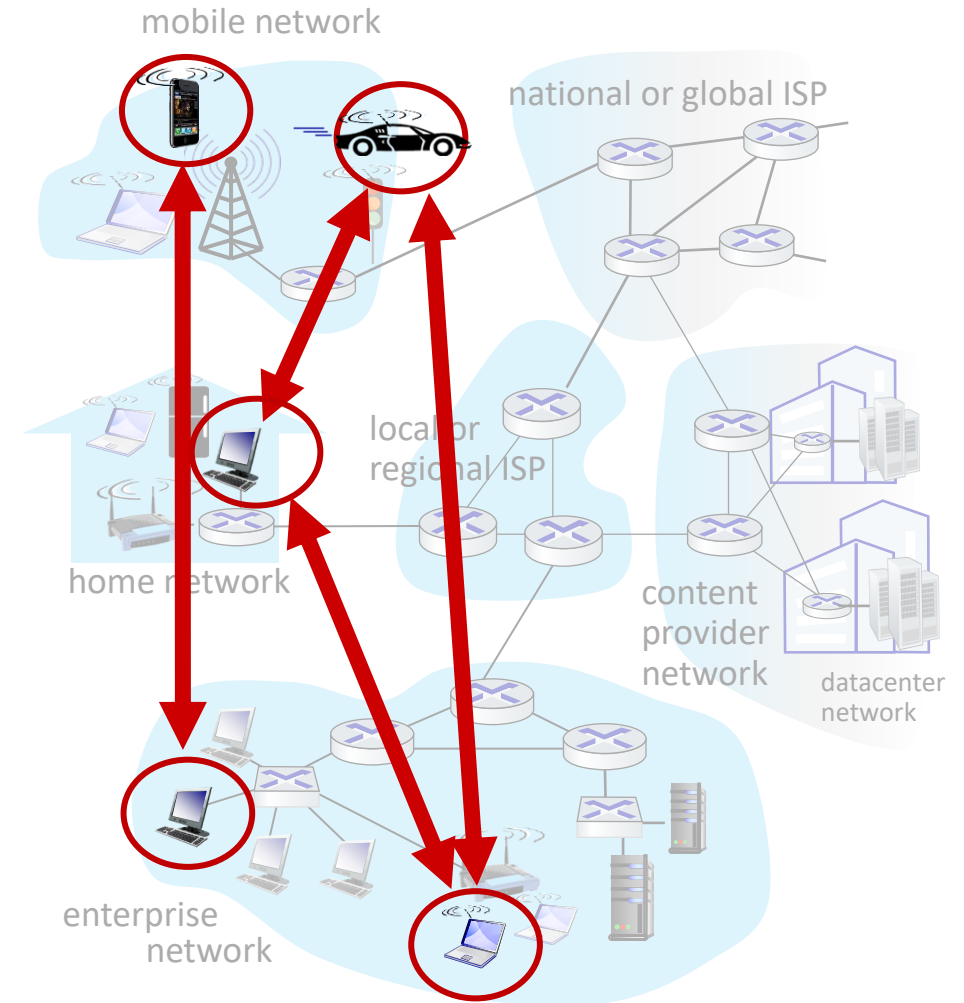
- Contactar/comunicarse con el servidor
- Puede estar conectado de forma intermitente
- Puede tener direcciones IP dinámicas
- No se comunican directamente entre sí (ejemplos HTTP, IMAP, FTP)



ARQUITECTURA P2P

Basado en: Jim Kurose, Keith Ross Pearson, 2020 Slides

- El servidor no siempre está activo
- Sistemas finales arbitrarios se comunican directamente
- “peers request service” de otros pares (peers), brindando un servicio a cambio
- *Auto escalables* – nuevos “peers” traen nuevas capacidades de servicio, así como nuevas demandas en el servicio
- Los “peers” están conectados de forma intermitente y cambian las direcciones IP
- Gestión compleja (ejemplo: intercambio de archivos P2P)



PROCESO DE COMUNICACIÓN

- process*: programa que se ejecuta dentro de un host
- Dentro del mismo host, dos procesos se comunican mediante la **comunicación entre procesos** (definida por el sistema operativo)
 - Los procesos en diferentes hosts se comunican mediante el **intercambio de mensajes**.

clientes, servidores

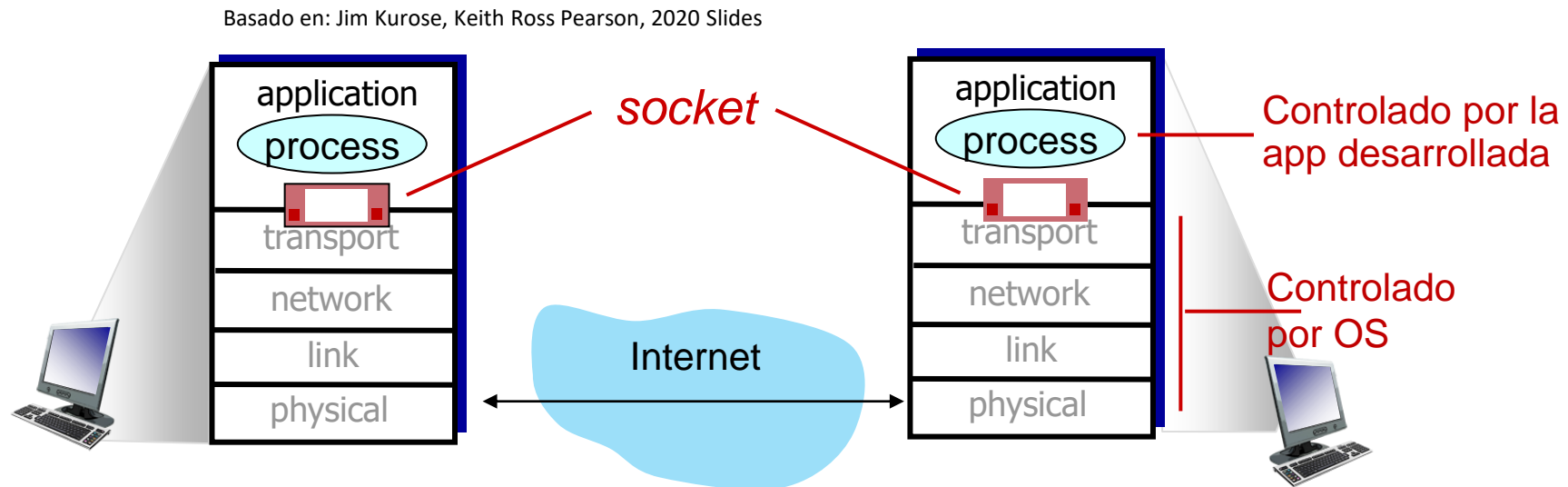
client process: proceso que inicia la comunicación

server process: proceso que espera ser contactado

Las aplicaciones con arquitecturas P2P tienen procesos de cliente y procesos de servidor

Sockets

- El proceso envía/recibe mensajes desde y hacia su propio **socket**
- socket es análogo a una “puerta”
 - El proceso de envío empuja el mensaje a la puerta
 - El proceso de envío se depende de la infraestructura de transporte del otro lado de la puerta para entregar el mensaje al socket en el proceso de recepción.
 - Dos sockets involucradas: una a cada lado



DIRECCIONAMIENTO DE PROCESOS

- Para recibir mensajes, el proceso debe tener un *identifier*
- El host tiene una Dirección IP única de 32-bit
- ¿La IP es suficiente para identificar el host donde se ejecuta el proceso?
- *identifier* incluye ambas: **dirección IP** y **número de puertos** asociados al proceso en el host.
- Ejemplo de números de puerto:
 - HTTP server: 80
 - mail server: 25
- Del **Hands-on 1**: Para enviar un mensaje HTTP a gaia.cs.umass.edu web server:
 - **IP address**: 128.119.245.12
 - **port number**: 80

NO!, El host puede estar ejecutando muchos procesos simultáneamente

LOS PROTOCOLOS EN LA CAPA DE APLICACION DEFINEN:

- Tipos de mensajes intercambiados,
 - (request, response)
- Sintaxis del mensaje:
 - Campos y organización del mensaje
- Semántica del mensaje
 - Significado del mensaje en cada campo
- Reglas cuándo y como los procesos envían mensajes

open protocols:

- Definidos en RFCs, Todos tienen acceso a la definición del protocolo
- Permite interoperabilidad
- (HTTP, SMTP)

proprietary protocols:

- (Skype, Zoom)

¿QUE SERVICIO DE TRANSPORTE NECESITA UNA APLICACIÓN?

data integrity

- Algunas apps (Transferencia de archivos, pagos electrónicos) requieren 100% de confiabilidad en la transferencia de datos
- Otras apps (audio) pueden tolerar algunas pérdidas

timing

- Ejemplo: Telefonía en Internet, video-juegos requieren efectividad de bajos retardos en el servicio

throughput

- Algunas apps (multimedia) requieren un mínimo throughput
- Otras (“apps elásticas”) pueden trabajar con cualquier throughput disponible

seguridad

- encriptado, integridad de datos, ...

REQUERIMIENTOS PARA EL SERVICIO DE TRANSPORTE

Aplicación	data loss	throughput	Timing sensible?
file transfer/download	Sin pérdida	Elástico	no
e-mail	Sin pérdida	Elástico	no
Documentos Web	Sin pérdida	Elástico	no
real-time audio/video	Tolerante	Audio: 5Kbps-1Mbps Video:10Kbps-5Mbps	Si (10's msec)
streaming audio/video	Tolerante	Igual al anterior	Si (~sec)
Video-juegos	Tolerante	Kbps+	Si (10's msec)
Mensajes de texto	Sin pérdida	Elástico	Depende

PROTOCOLOS PARA SERVICIO DE TRANSPORTE

TCP-Transmission Control Protocol:

- *Transporte confinable* entre procesos de envío y recepción
- *Control de flujo*: quien envía no va a sobrecargar al receptor
- *Control de congestión*: Acelerar el remitente cuando la red está sobrecargada
- *Connection-oriented*: configuración requerida entre los procesos del cliente y del servidor
- *NO PROVEE*: timing, throughput mínimo garantizado, ***seguridad***

UDP-User Datagram Protocol:

- *Transferencia de datos no confiable* entre procesos de envío y recepción
- *NO PROVEE*: confiabilidad, control de flujo y congestión, timing, throughput garantizado, seguridad, conexión configurable.



APLICACIONES DE INTERNET Y PROTOCOLOS DE TRANSPORTE

application	Protocolo capa de aplicación	Protocolo de transporte
file transfer/download	FTP [RFC 959]	TCP
e-mail	SMTP [RFC 5321]	TCP
Documentos Web	HTTP 1.1 [RFC 7320]	TCP
Telefonía en Internet	SIP [RFC 3261], RTP [RFC 3550], o propio	TCP / UDP
streaming audio/video	HTTP [RFC 7320], DASH	TCP
Video-juegos	WOW, FPS (propio)	UDP / TCP

ASEGURANDO TCP

Vanilla TCP & UDP sockets:

- sin cifrado
- contraseñas de texto sin cifrar enviadas al socket a través de Internet en texto ***sin cifrar***

Transport Layer Security (TLS)

- proporciona conexiones TCP cifradas
- integridad de los datos
- autenticación de punto final

TLS implementada en la capa de aplicación

- las aplicaciones usan bibliotecas TLS, que a su vez usan TCP
- texto sin cifrar enviado a un "socket" a través de Internet cifrado

HTTP (Hypertext Transfer Protocol) y Web

- Una página web consiste en “objetos” *objects*, cada uno de los cuales puede ser almacenado en diferentes servidores Web
- El objeto puede ser un archivo HTML, JPEG imagen, aplicación Java, archivo de audio/video, entre otros.
- Una página web consiste en *un archivo base HTML* el cual incluye *muchos objetos referenciados, y cada uno* direccionado por una *URL (Uniform Resource Locator)*:

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

host name

path name

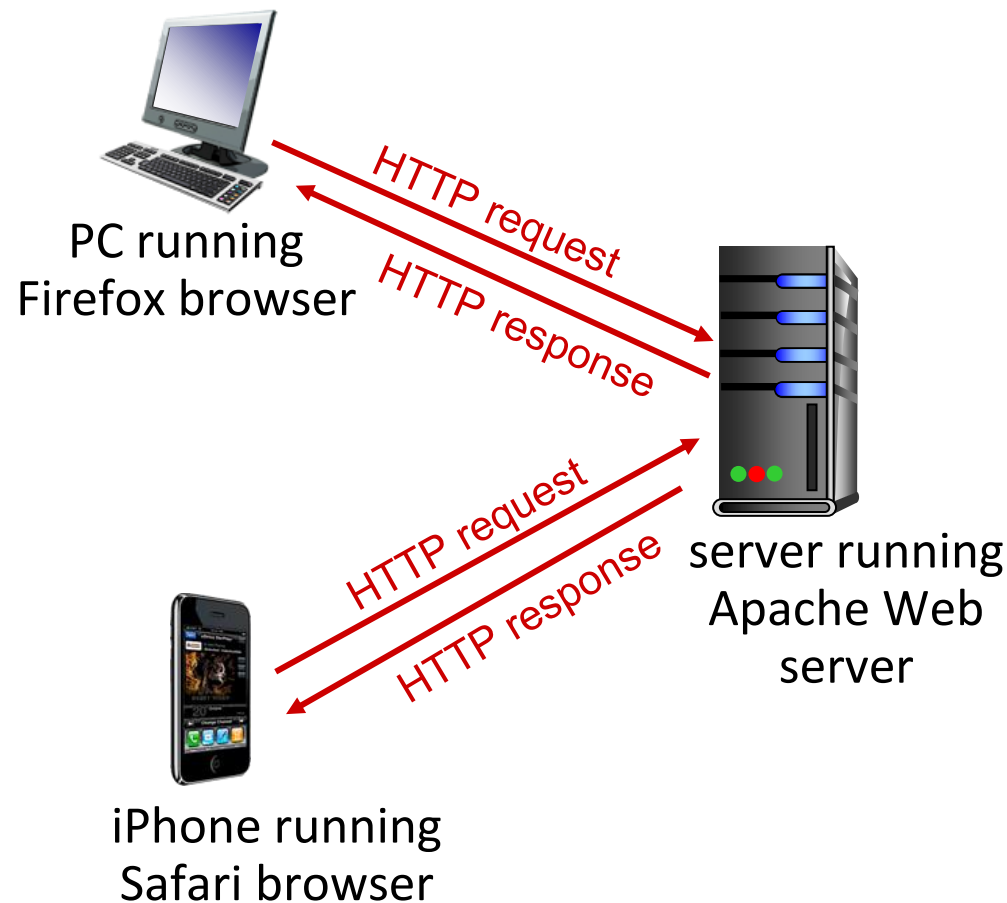
HTTP (generalidades)

Basado en: Jim Kurose, Keith Ross Pearson, 2020 Slides

HTTP:

Protocolo web de la capa de aplicación

- Modelo cliente/servidor:
 - *client*: navegador que solicita/recibe (mediante el protocolo HTTP) y "muestra" objetos web
 - *server*: El servidor web envía (mediante el protocolo HTTP) objetos en respuesta a las solicitudes



HTTP (generalidades)

HTTP usa TCP:

- el cliente inicia la conexión TCP (crea socket) al servidor, puerto 80
- el servidor acepta la conexión TCP del cliente
- Mensajes HTTP (mensajes de protocolo de la capa de aplicación) intercambiados entre el navegador (cliente HTTP) y el servidor web (servidor HTTP)
- Conexión TCP cerrada

HTTP no almacena estados

- El servidor no mantiene información sobre solicitudes de clientes anteriores.

Los protocolos que mantienen el estado son complejos

- la historia pasada (estado) debe mantenerse
- si el servidor / cliente falla, sus puntos de vista del "estado" pueden ser inconsistentes "deben reconciliarse"

HTTP: tipos de conexión

Non-persistent HTTP

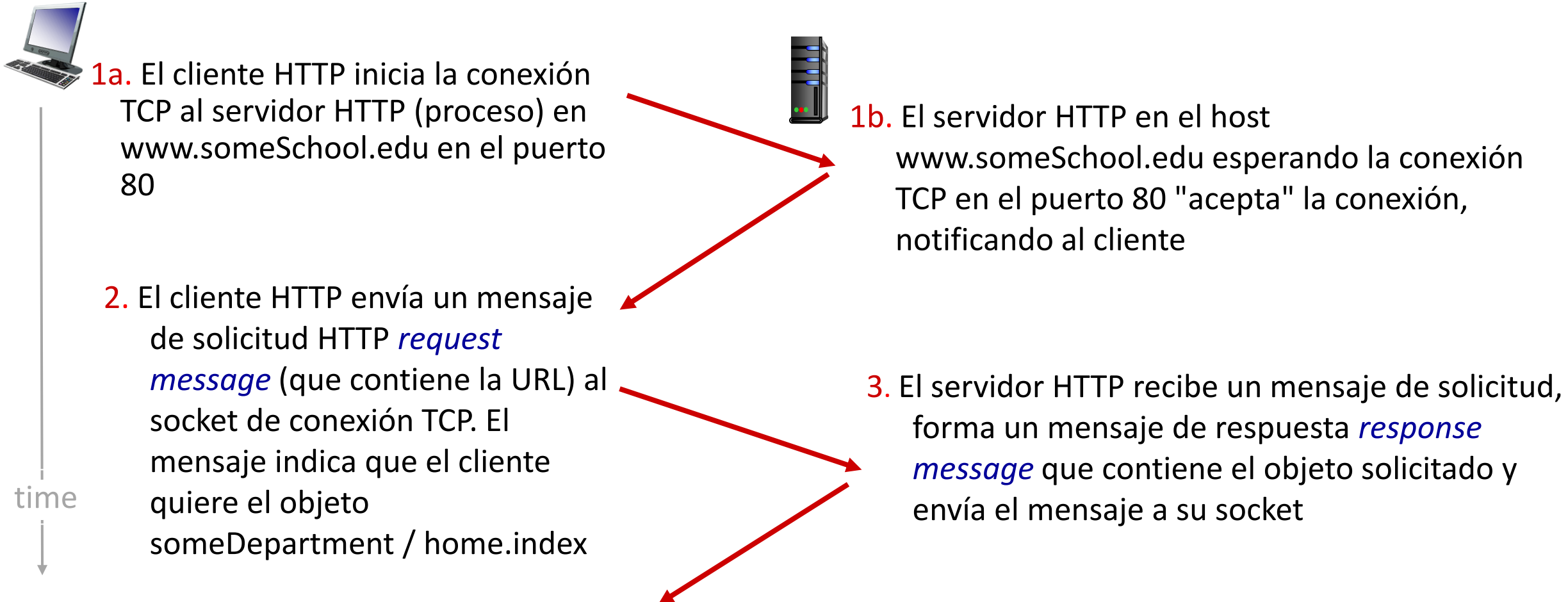
1. Conexión TCP abierta
2. Como máximo un objeto enviado a través de una conexión TCP
3. Conexión TCP cerrada
(cargar varios objetos requirió múltiples conexiones)

Persistent HTTP

- Conexión TCP abierta a un servidor
- Se pueden enviar varios objetos a través de una única conexión TCP entre el cliente y ese servidor.
- Conexión TCP cerrada

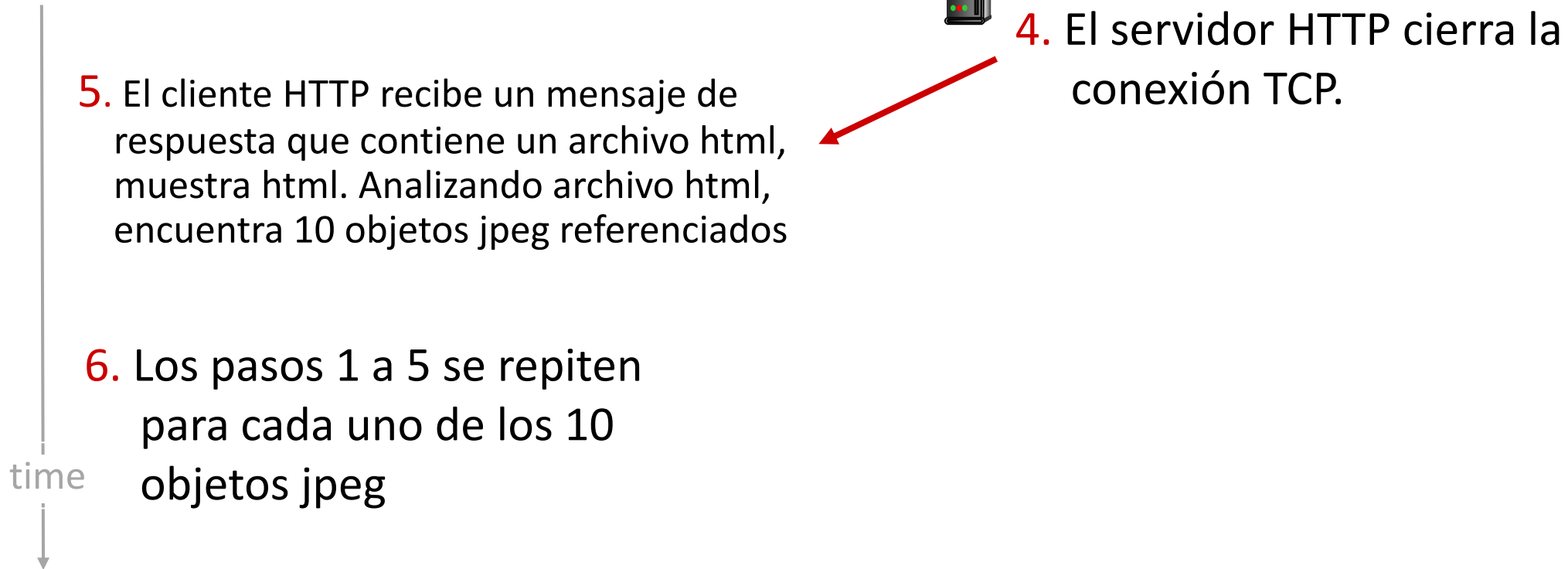
Non-persistent HTTP: Ejemplo

Usuario ingresa a URL: `www.someSchool.edu/someDepartment/home.index`
(contiene texto, referencias, al menos 10 imagenes, etc)



Non-persistent HTTP: Ejemplo

Usuario ingresa a URL: `www.someSchool.edu/someDepartment/home.index`
(contiene texto, referencias, al menos 10 imagenes, etc)



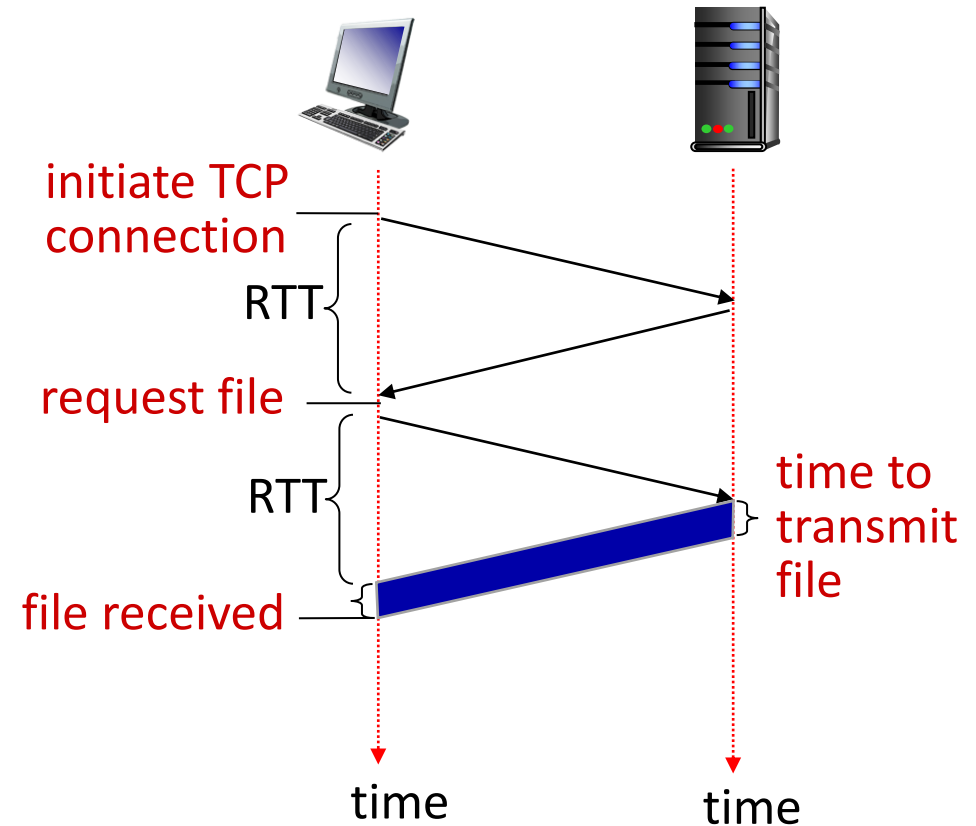
Non-persistent HTTP: tiempo de respuesta

RTT (definición): tiempo para que un pequeño paquete viaje de cliente a servidor y viceversa

HTTP tiempo de RTA (por objeto):

- un RTT para iniciar la conexión TCP
- un RTT para la solicitud HTTP y los primeros bytes de respuesta HTTP para devolver
- tiempo de transmisión de objeto/archivo

Basado en: Jim Kurose, Keith Ross Pearson, 2020 Slides



Non-persistent HTTP (tiempo de respuesta) = $2RTT + \text{file transmission time}$

Persistent HTTP (HTTP 1.1)



Non-persistent HTTP (problemas):

- requiere 2 RTT por objeto
- Sobrecarga del sistema operativo para cada conexión TCP
- Los navegadores a menudo abren múltiples conexiones TCP paralelas para buscar objetos referenciados en paralelo.

Persistent HTTP (HTTP1.1):

- el servidor deja la conexión abierta después de enviar la respuesta
- mensajes HTTP posteriores entre el mismo cliente / servidor enviados a través de una conexión abierta
- el cliente envía solicitudes tan pronto como encuentra un objeto referenciado
- tan solo un RTT para todos los objetos referenciados (reduciendo el tiempo de respuesta a la mitad)

HTTP (request message)

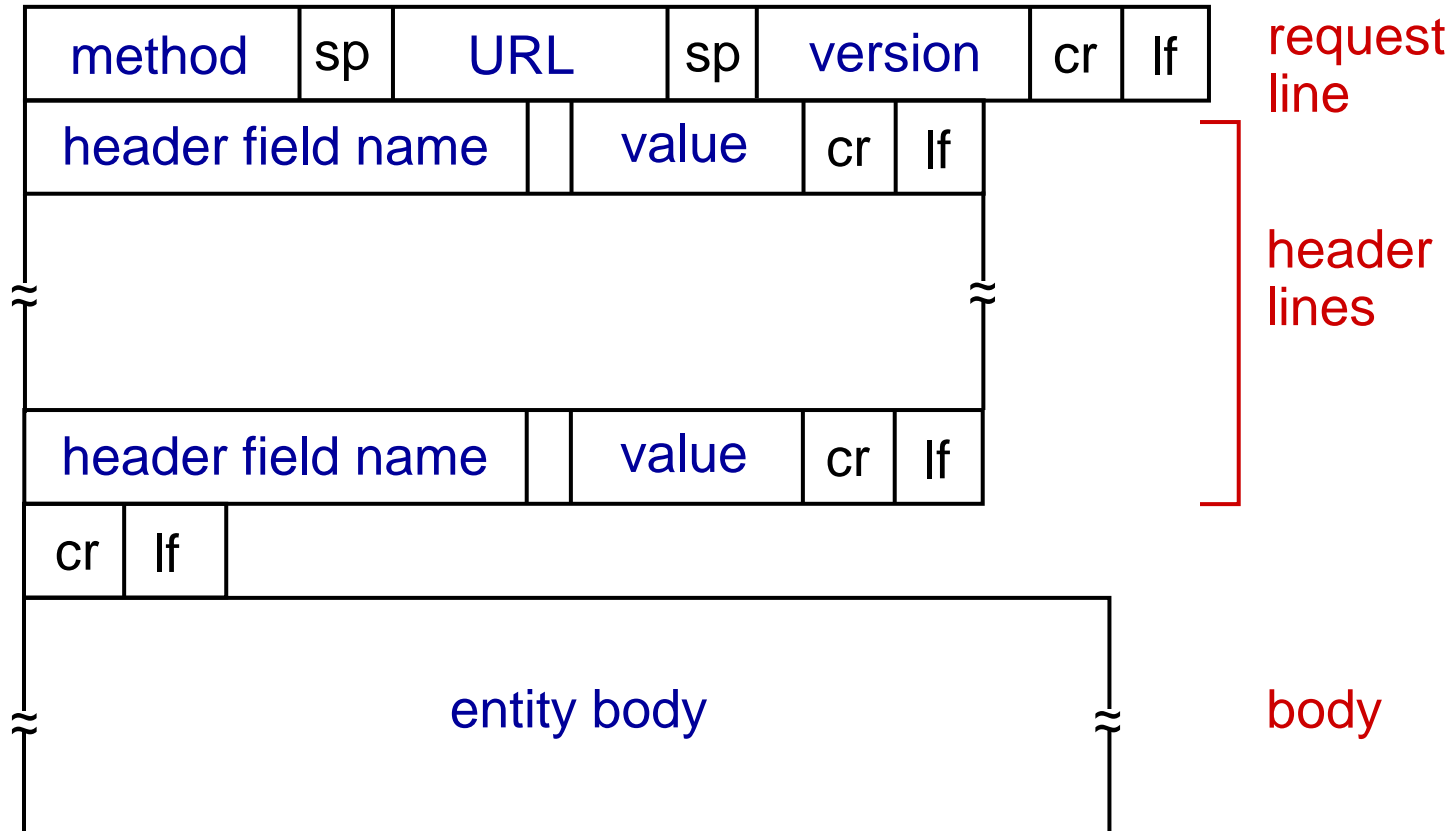
- Dos tipos de mensajes HTTP: *request, response*
- HTTP request message:
 - ASCII (human-readable format)

request line (GET, POST,
HEAD commands) →

/ Tiene el caracter de “return”
/ line-feed character

tiene “return, line feed” →
al inicio, indica el fin de
las líneas de encabezado

HTTP request message: Formato general



Basado en: Jim Kurose, Keith Ross Pearson, 2020 Slides

Otros HTTP request messages

POST method:

- la página web a menudo incluye entrada de formulario
- entrada del usuario enviada del cliente al servidor en el cuerpo de la entidad del mensaje de solicitud HTTP POST

GET method (para enviar datos al servidor):

- incluir datos de usuario en el campo URL del mensaje de solicitud HTTP GET (después de un "?"):

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


HEAD method:

- solicita encabezados (solo) que se devolverían si se solicitara la URL especificada con un método HTTP GET.

PUT method:

- carga un nuevo archivo (objeto) al servidor
- reemplaza completamente el archivo que existe en la URL especificada con contenido en el cuerpo de la entidad del mensaje de solicitud POST HTTP

HTTP response message

status line (protocol  HTTP/1.1 200 OK
status code status phrase)

HTTP response status codes

- El código de estado aparece en la 1ª línea del mensaje de respuesta de servidor a cliente. Algunos códigos de ejemplo:

200 OK

- solicitud realizada correctamente, objeto solicitado más adelante en este mensaje

301 Moved Permanently

- objeto solicitado movido, nueva ubicación especificada más adelante en este mensaje (en el campo Ubicación)

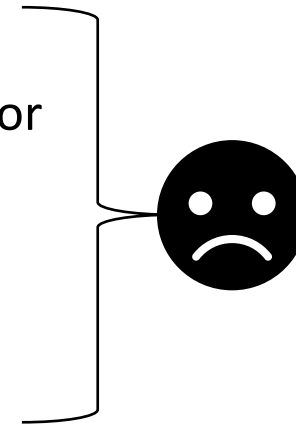
400 Bad Request

- mensaje de solicitud no entendido por el servidor

404 Not Found

- documento solicitado no encontrado en este

505 HTTP Version Not Supported



Mensaje GET HTTP (Wireshark)



1. Ingresar a: gaia.cs.umass.edu

- Utilice Wireshark para rastrear paquetes HTTP cuando realice esta acción en su explorador.

2. Identifique los mensajes GET HTTP request:

No.	Time	Source	Destination	Protocol	Length	Info
167	2021-08-03 20:55:47,558555	192.168.1.17	128.119.245.12	HTTP	608	GET / HTTP/1.1
176	2021-08-03 20:55:47,665577	128.119.245.12	192.168.1.17	HTTP	294	HTTP/1.1 304 Not Modified
190	2021-08-03 20:55:47,777042	192.168.1.17	104.196.134.131	HTTP	459	GET /images/qupmember.gif HTTP/1.1
194	2021-08-03 20:55:47,840429	104.196.134.131	192.168.1.17	HTTP	458	HTTP/1.1 404 Not Found (text/html)

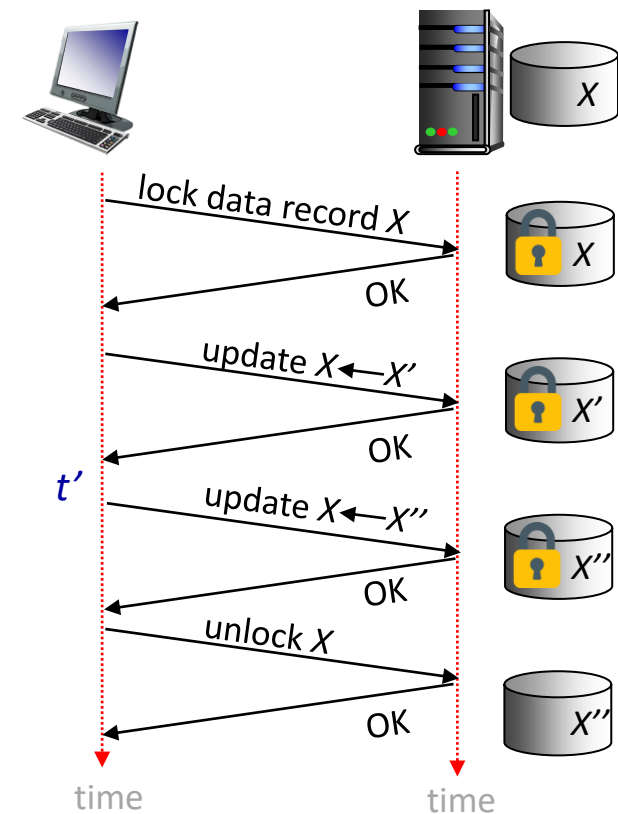
3. Identifique la respuesta del servidor HTTP

Mantener estados de usuario/servidor: cookies

La interacción HTTP GET/response *no guarda estados*

- no hay noción de intercambios de varios pasos de mensajes HTTP para completar una "transacción" web
- no es necesario que el cliente / servidor rastree el "estado" del intercambio de varios pasos
- todas las solicitudes HTTP son independientes entre sí
- no es necesario que el cliente / servidor se "recupere" de una transacción parcialmente completada pero nunca completada por 100%

Un protocolo con estados: el cliente hace dos cambios a X o ninguno.



Mantener estados de usuario/servidor: cookies

Los sitios web y el navegador del cliente utilizan *cookies* para mantener cierto estado entre interacciones.

4 componentes:

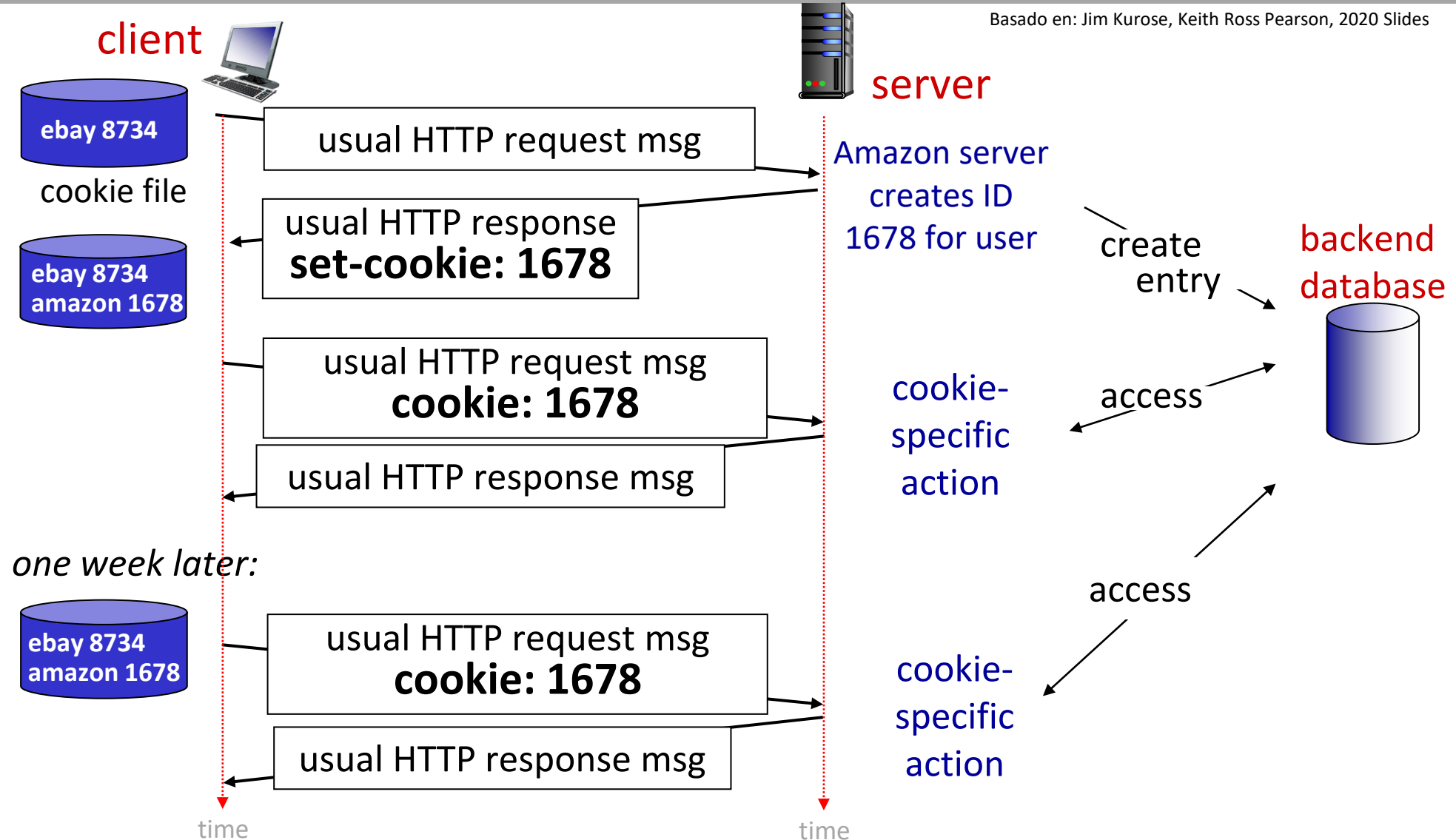
- 1) línea de encabezado de cookie del mensaje de respuesta HTTP
- 2) línea de encabezado de cookie en el siguiente mensaje de solicitud HTTP
- 3) archivo de cookies guardado en el host del usuario, administrado por el navegador del usuario
- 4) base de datos back-end en el sitio web

En palabras:

- David usa el navegador en el portátil, visita un sitio de comercio electrónico específico por primera vez
- cuando las solicitudes HTTP iniciales llegan al sitio, el sitio crea: ID único ("cookie")
- entrada en la base de datos backend para ID
- Las solicitudes HTTP posteriores de David a este sitio contendrán un valor de ID de cookie, lo que permitirá que el sitio "identifique" a David.

Mantener estados de usuario/servidor: cookies

Basado en: Jim Kurose, Keith Ross Pearson, 2020 Slides



HTTP cookies

Para qué se pueden utilizar las cookies: Autorización

- carritos de compra
- Recomendaciones
- estado de la sesión del usuario (correo electrónico web)

Reto: cómo mantener el estado?

- *Protocolos en puntos terminales:* mantener el estado en el remitente / receptor en múltiples transacciones
- *En mensajes:* las cookies en los mensajes HTTP llevan el estado

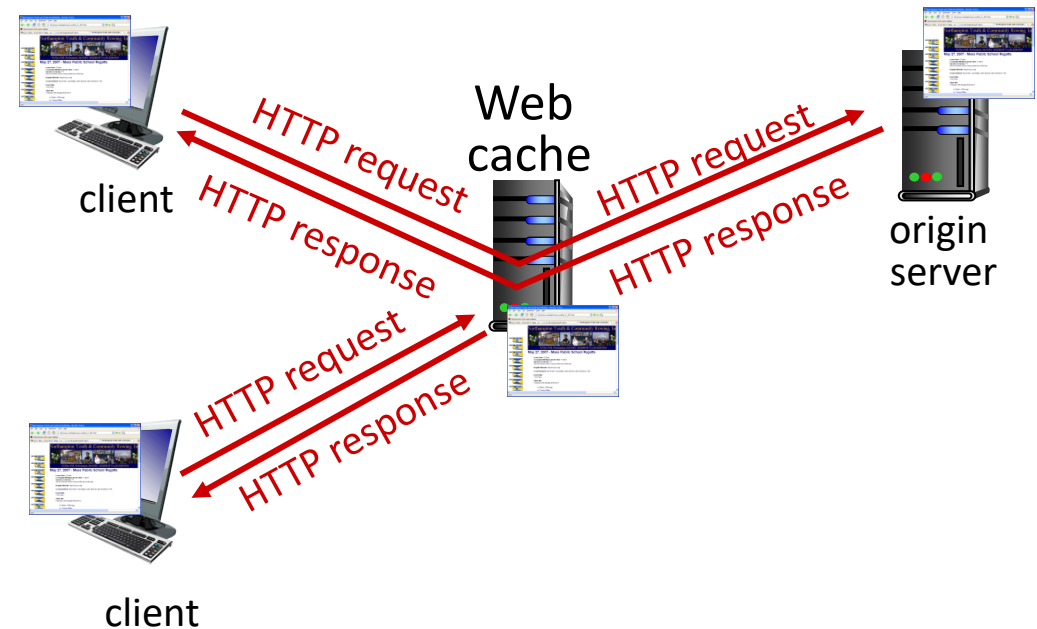
cookies y privacidad:

- Las cookies permiten que los sitios aprendan mucho sobre usted en su sitio.
- Las cookies persistentes de terceros (cookies de seguimiento) permiten rastrear la identidad común (valor de la cookie) en varios sitios web.

Web caches

Objetivo: satisfacer las solicitudes de los clientes sin involucrar al servidor de origen

- El usuario configura el navegador para que apunte a una caché web (local)
- El navegador envía todas las solicitudes HTTP a la caché
- **Si** el objeto está en la caché: la caché devuelve el objeto al cliente
- **De lo contrario**, la caché solicita el objeto del servidor de origen, almacena en caché el objeto recibido y luego devuelve el objeto al cliente



Web caches (proxy servers)

- La caché web actúa como cliente y servidor:
 - servidor para el cliente solicitante original
 - cliente al servidor de origen

El servidor le dice a la caché sobre el almacenamiento en caché permitido del objeto en el encabezado de respuesta:

```
Cache-Control: max-age=<seconds>
```

```
Cache-Control: no-cache
```

¿Por qué usarla?

- reducir el tiempo de respuesta a la solicitud del cliente (la caché está más cerca del cliente)
- reducir el tráfico en el enlace de acceso de una institución
- Internet está lleno de cachés
- permite a los proveedores de contenido "deficientes" entregar contenido de manera más eficaz

Almacenamiento en cache (Ejemplo)

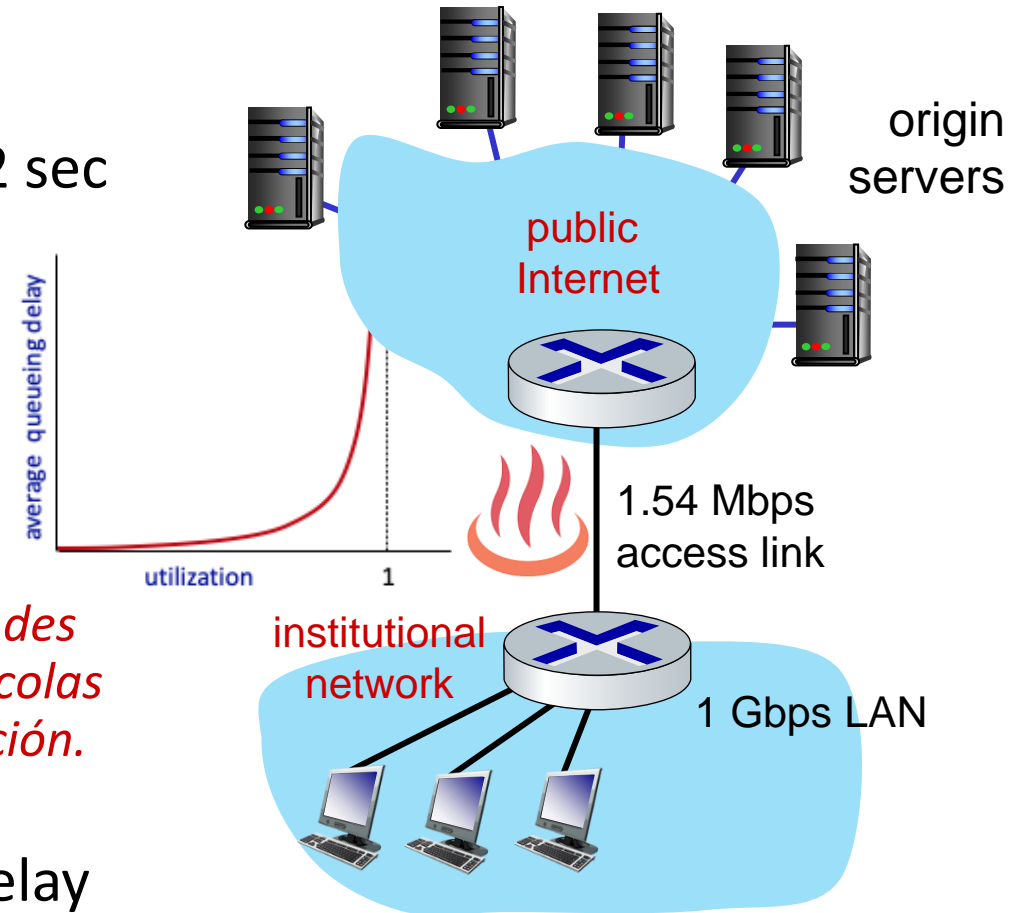
Contexto:

- access link rate: 1.54 Mbps
- RTT desde router institucional a servidor: 2 sec
- web object size: 100K bits
- average request rate desde buscadores a servidor de origen: 15/sec
 - avg data rate a buscadores: 1.50 Mbps

Desempeño:

- access link utilization = .97
- LAN utilization: .0015
- end-end delay = Internet delay +
access link delay + LAN delay
= 2 sec + minutes + usecs

Problema: grandes retrasos en las colas con alta utilización.



Basado en: Jim Kurose, Keith Ross Pearson, 2020 Slides

Opción 1: pagar un enlace de acceso más rápido

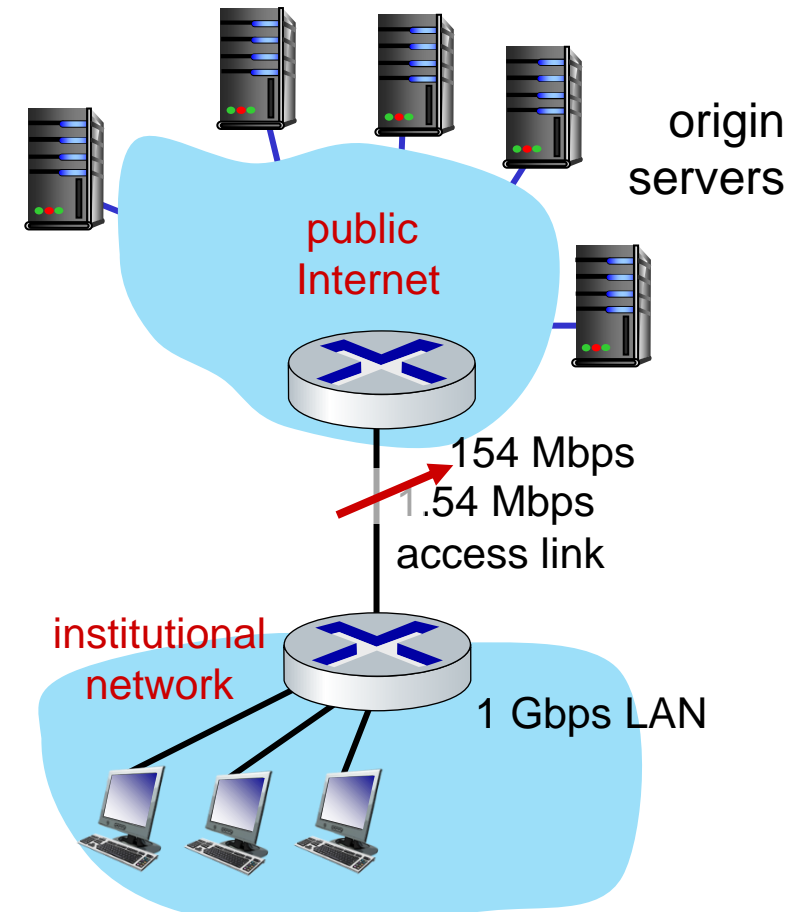
Contexto:

- access link rate: ~~1.54~~ 154 Mbps
- RTT desde router institucional a servidor : 2 sec
- web object size: 100K bits
- average request rate desde buscadores a servidor de origen: 15/sec
 - avg data rate a buscadores: 1.50 Mbps

Desempeño:

- access link utilization = ~~.97~~ .0097
- LAN utilization: .0015
- end-end delay = Internet delay +
access link delay + LAN delay
= 2 sec + ~~minutes~~ + usecs

Costo: Mejor enlace *Más caro!* → msecs



Basado en: Jim Kurose, Keith Ross Pearson, 2020 Slides

Opción 2: instalar cache Web

Contexto:

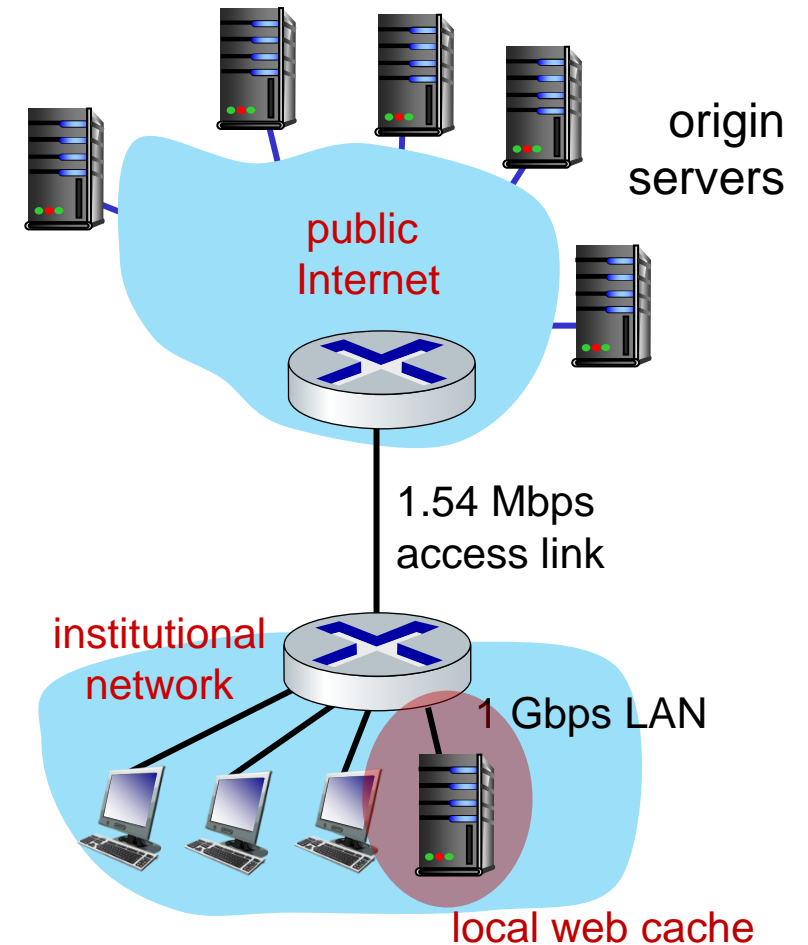
- access link rate: 1.54 Mbps
- RTT desde router institucional a servidor: 2 sec
- web object size: 100K bits
- average request rate desde buscadores a servidor de origen: 15/sec
 - avg data rate a buscadores: 1.50 Mbps

Costo: web cache (más económico)

Desempeño:

- LAN utilization: .?
- access link utilization = ?
- average end-end delay = ?

¿Cómo calcular?

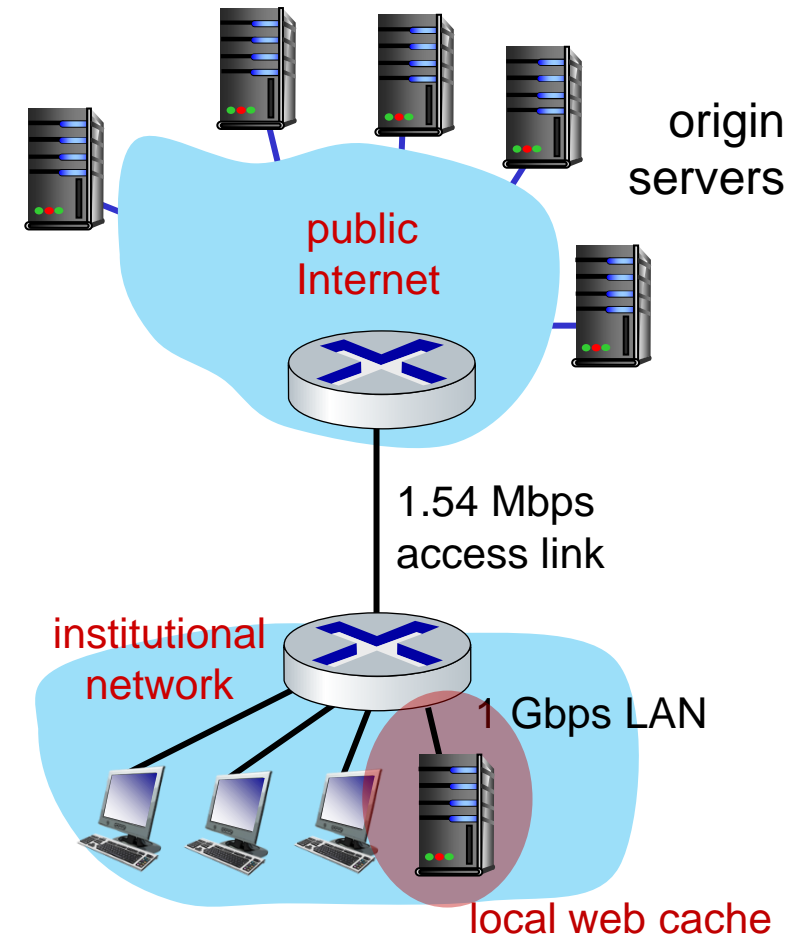


Basado en: Jim Kurose, Keith Ross Pearson, 2020 Slides

Cálculos cuando usamos cache Web:

- Supongamos que la tasa de aciertos de caché es 0.4:
40% de solicitudes atendidas por caché, con un retraso bajo (mseg)
- 60% de solicitudes satisfechas en origen
- calificar a los navegadores a través del enlace de acceso = $0.6 * 1.50 \text{ Mbps} = 0.9 \text{ Mbps}$
- utilización del enlace de acceso = $0.9 / 1.54 = .58$ significa:
Retardo de cola bajo (mseg) en el enlace de acceso
- average end-end delay:
= $0.6 * (\text{retardo desde servidores en origen})$
+ $0.4 * (\text{retardo cuando supe una caché})$
= $0.6 (2.01) + 0.4 (\sim \text{msecs}) = \sim 1.2 \text{ secs}$

Retardo final promedio más bajo que con el enlace de 154 Mbps (¡y más barato también!)



Basado en: Jim Kurose, Keith Ross Pearson, 2020 Slides

Conditional GET

Basado en: Jim Kurose, Keith Ross Pearson, 2020 Slides

client



server



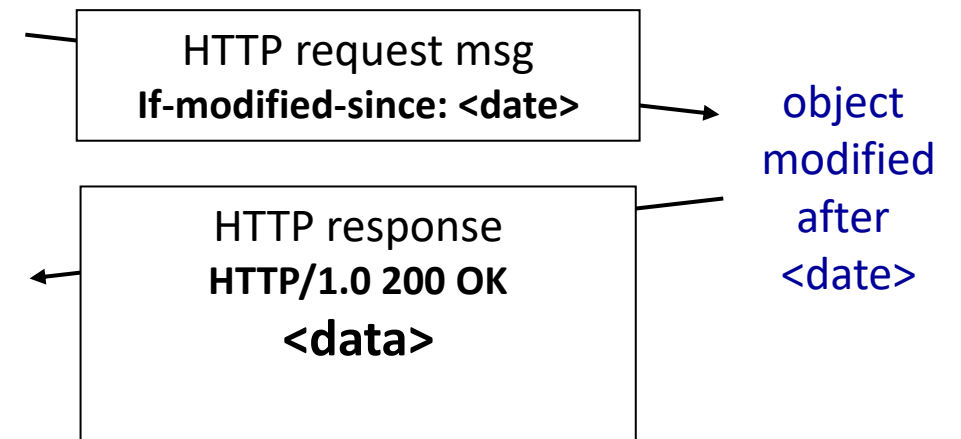
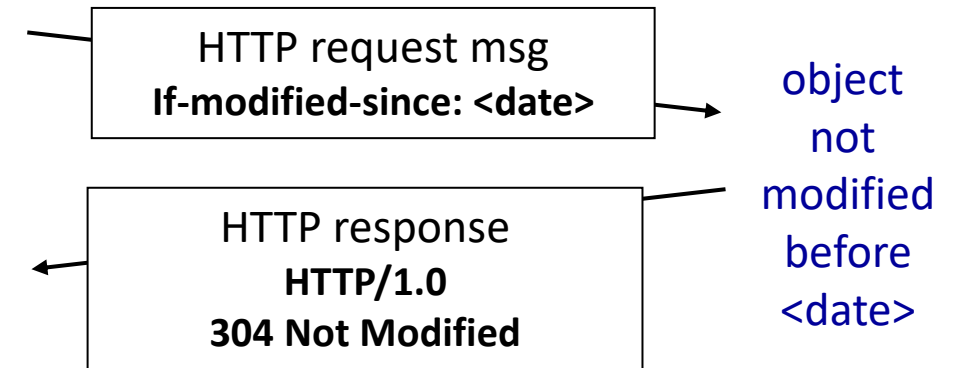
Objetivo: No enviar objeto si la cache tiene que actualizar su versión

- sin retraso en la transmisión de objetos (o uso de recursos de red)
- *client*: especifica la fecha de la copia en caché en la solicitud HTTP

If-modified-since: <date>

- *server*: La respuesta no contiene ningún objeto si la copia en caché está actualizada:

HTTP/1.0 304 Not Modified



HTTP/2

Objetivo: Disminuir retardos para solicitudes HTTP de múltiples objetos

HTTP1.1: introdujo múltiples GET (pipeline) canalizados a través de una sola conexión TCP

- el servidor responde en orden (FCFS: programación por orden de llegada) a las solicitudes GET
- con FCFS, es posible que los objetos pequeños tengan que esperar la transmisión (**head-of-line (HOL) blocking**) detrás de objetos grandes
- la recuperación de pérdidas (retransmitir segmentos TCP perdidos) detiene la transmisión de objetos

HTTP/2

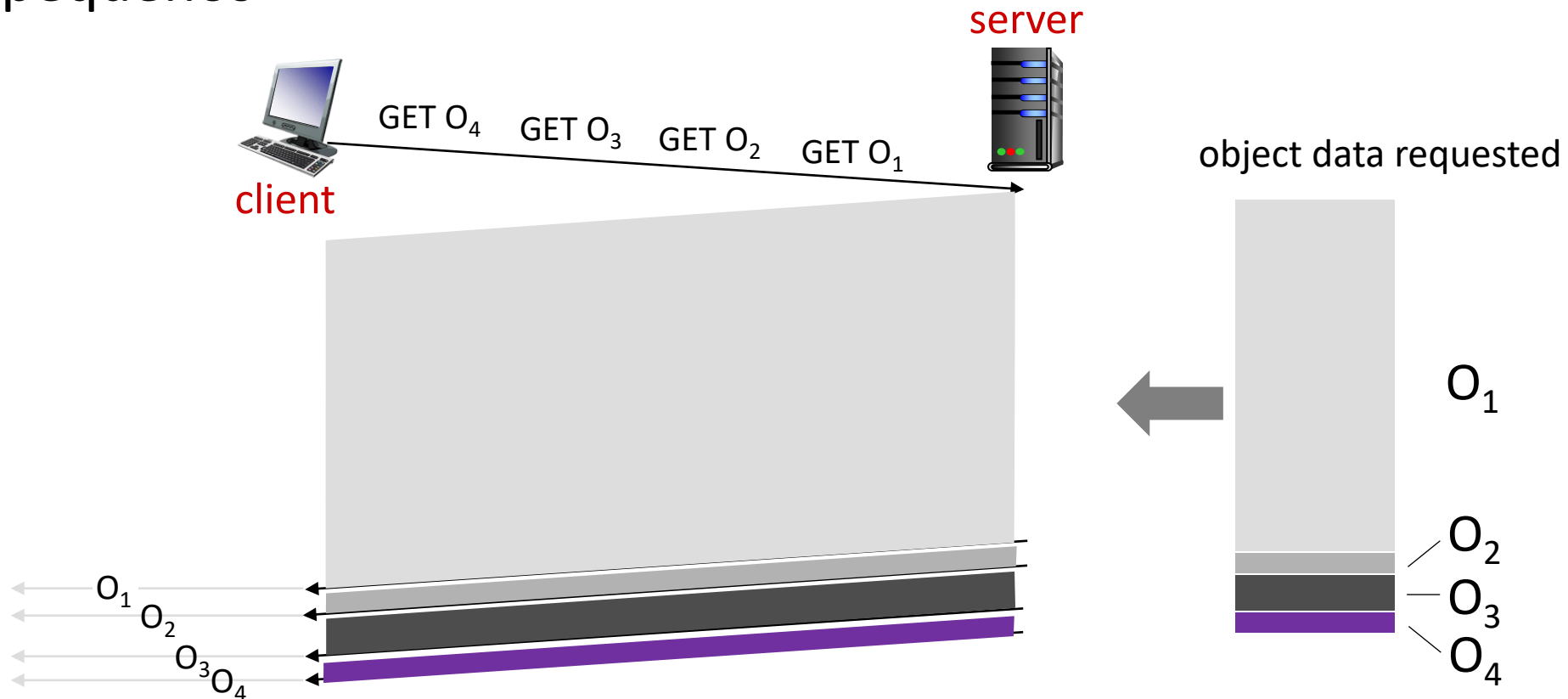
Objetivo: Disminuir retardos para solicitudes HTTP de multiples objetos

HTTP/2: [RFC 7540, 2015] Mayor flexibilidad en el servidor para enviar objetos al cliente:

- métodos, códigos de estado, la mayoría de los campos de encabezado sin cambios desde HTTP 1.1
- orden de transmisión de los objetos solicitados según la prioridad del objeto especificada por el cliente (no necesariamente FCFS)
- enviar objetos no solicitados al cliente
- dividir objetos en frames, programar frames para mitigar el bloqueo HOL

HTTP/2: mitigando el bloqueo HOL

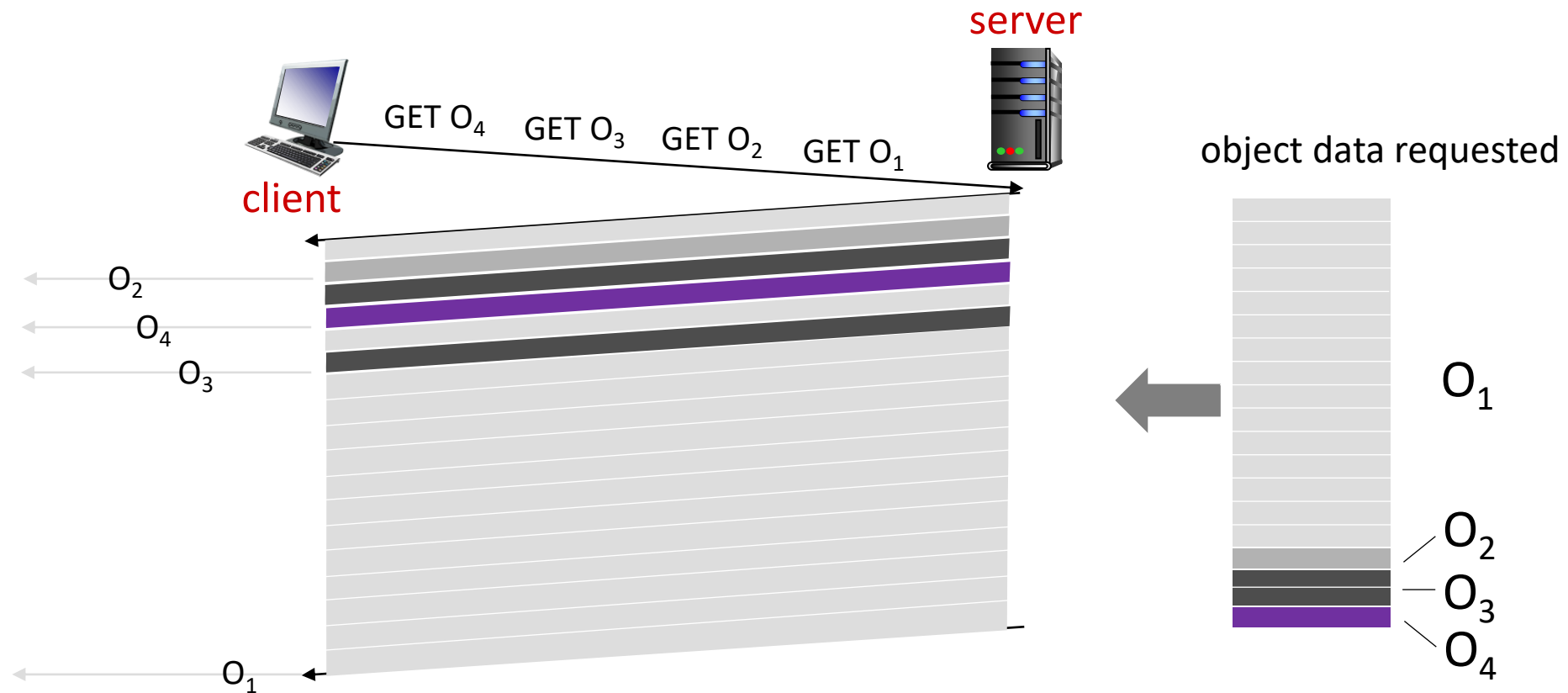
HTTP 1.1: el cliente solicita 1 objeto grande (archivo de video) y 3 objetos más pequeños



Objetos despachados en el orden: O_2 , O_3 , O_4 esperan detrás de O_1

HTTP/2: mitigando el bloqueo HOL

HTTP/2: Objetos divididos en frames, transmisión de frames intercalados.



O₂, O₃, O₄ se despachan rápido, O₁ ligeramente retrasado

HTTP/2 migrando a HTTP/3

HTTP / 2 a través de una sola conexión TCP significa:

- la recuperación de la pérdida de paquetes aún detiene todas las transmisiones de objetos
- Como en HTTP 1.1, los navegadores tienen incentivos para abrir múltiples conexiones TCP paralelas para reducir el estancamiento y aumentar el rendimiento general (throughput).
- sin seguridad sobre la conexión TCP vanilla
- **HTTP / 3:** agrega seguridad, error por objeto y control de congestión (más canalización) a través de UDP



HANDS-ON

