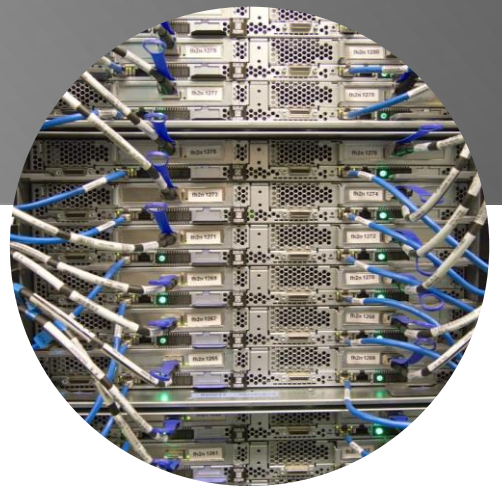
Redes de computadores 2022 -1 (11310052)

David Felipe Celeita Rodriguez

Universidad del Rosario | Escuela de Ingeniería, Ciencia y Tecnología

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

- Katherine Johnson

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

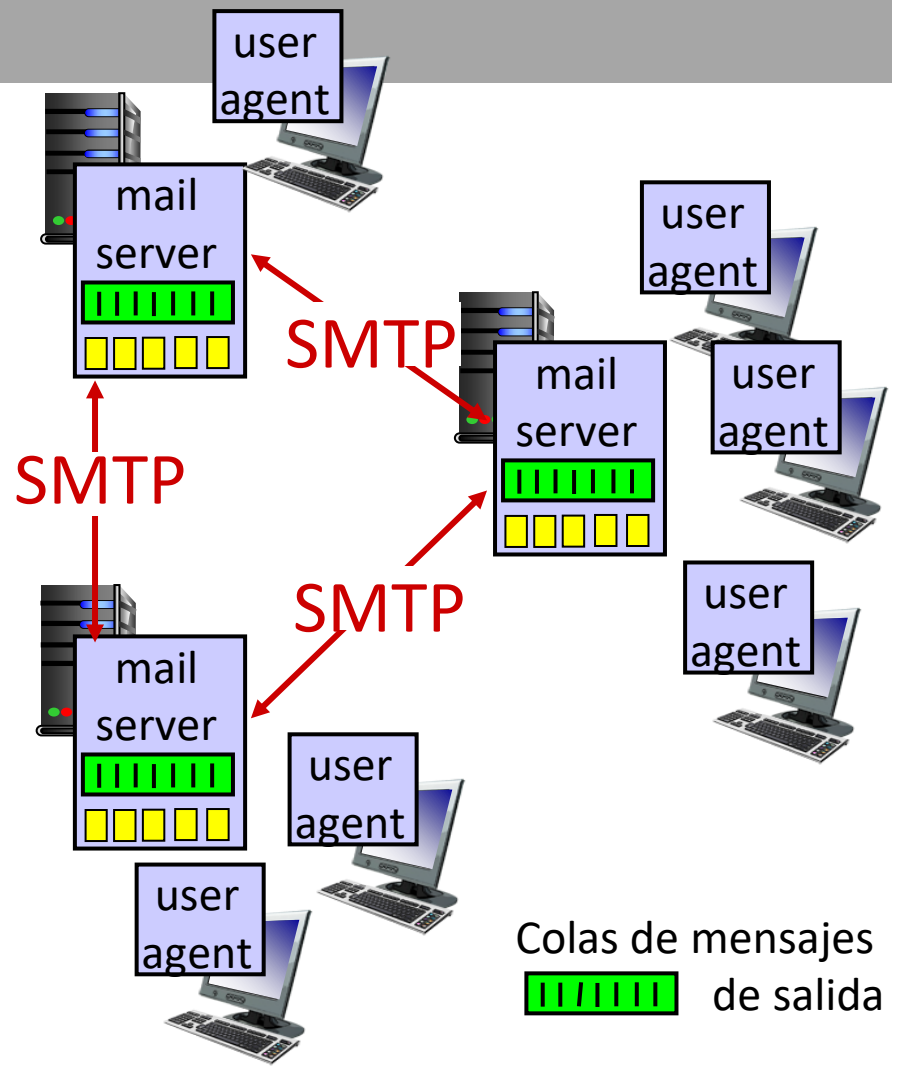
E-mail

Tres grandes componentes:

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

User Agent

- “visualizador de correos”
- Formato, edición, envío/recepción de mensajes
- e.g., Outlook, iPhone mail client
- Mensajes de salida o entrada almacenados en un servidor



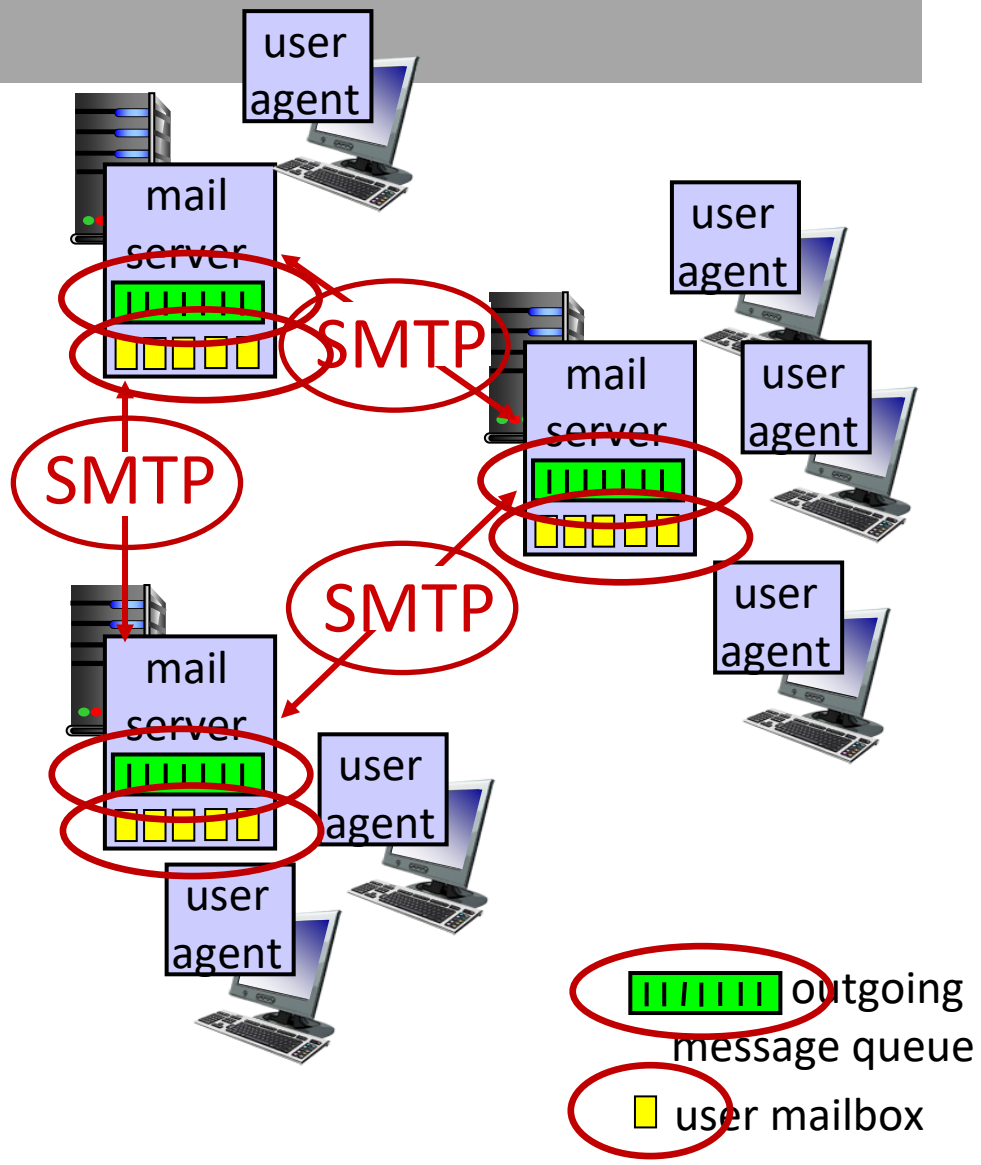
E-mail: mail servers

mail servers:

- *mailbox* Contiene mensajes de entrada por usuario
- *message queue* colas de mensaje de salida (para ser enviados)

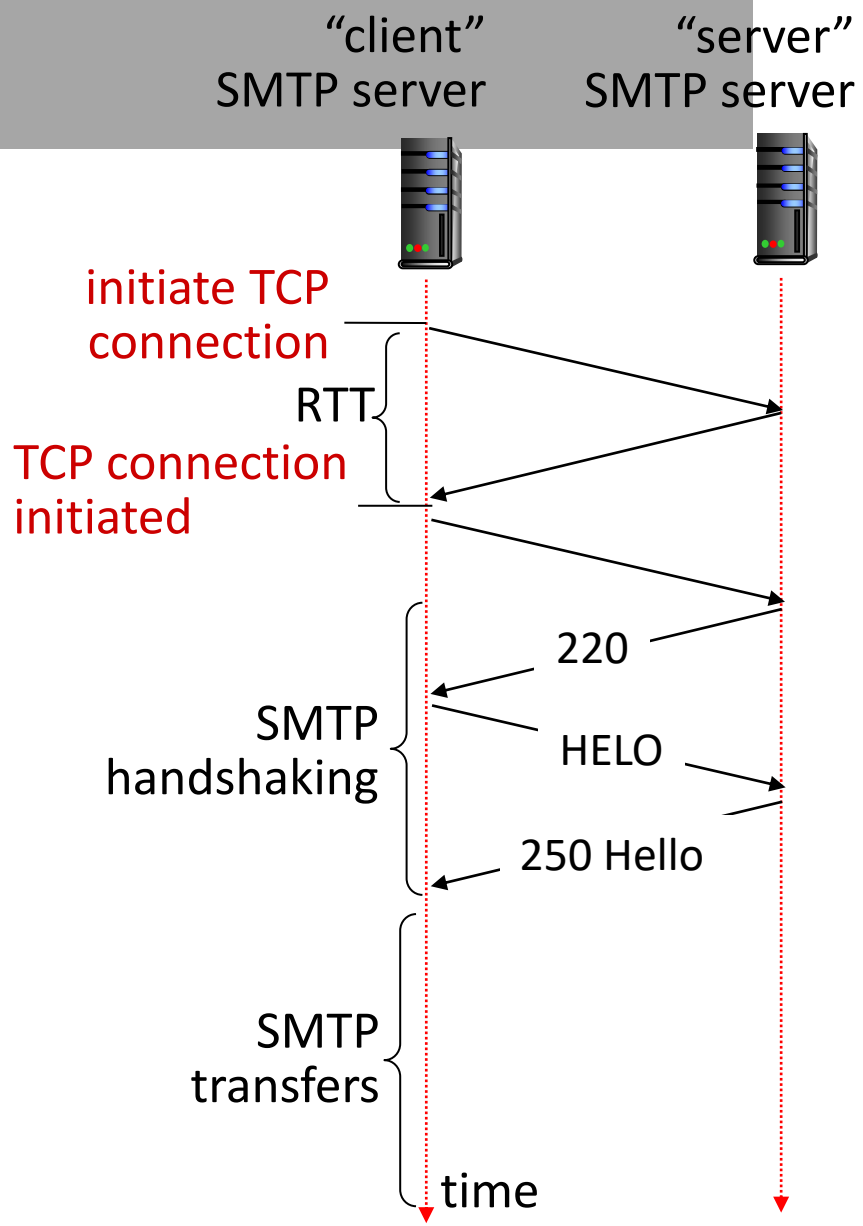
SMTP protocol Opera entre mail servers para enviar mensajes

- "cliente": servidor de envío
- "servidor": servidor de recepción



SMTP RFC (5321)

- **Utiliza TCP** para transferir de forma confiable mensajes de correo electrónico desde el cliente (servidor de correo que inicia la conexión) al servidor de recepción
- **Transferencia directa:** servidor de envío (actuando como cliente) al servidor receptor
- Tres fases de Transferencia:
  - SMTP handshaking (greeting)
  - SMTP transfer of messages
  - SMTP closure
- Interacción command/response (parecido a HTTP)
  - commands: ASCII text
  - response: status code + phrase



Ejemplo: Andrea envía un email a Carlos

1) Andrea usa un UA para redactor un e-mail “a” carlos@urosario.edu.co

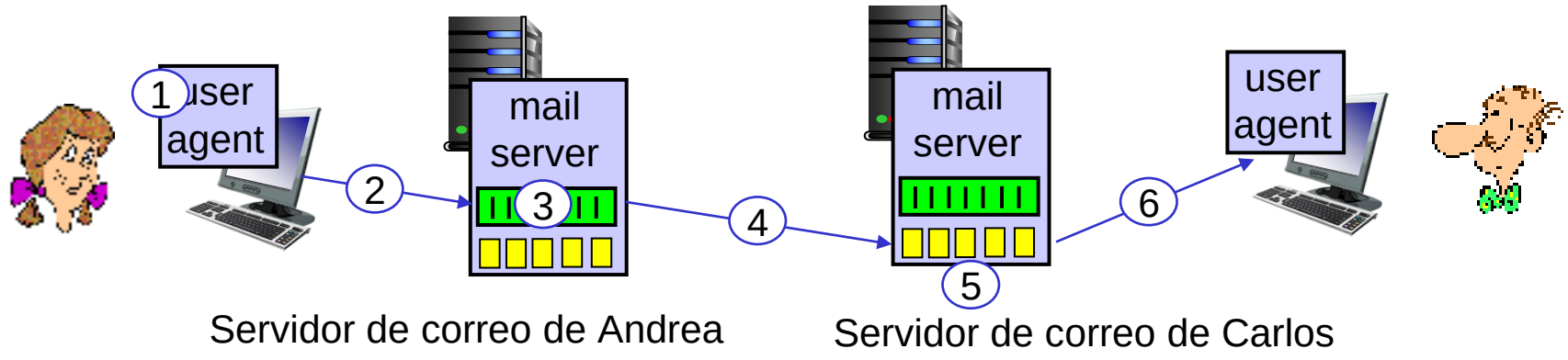
2) El UA de Andrea envía el mensaje usando un servidor SMTP; el mensaje entra a una cola

3) el lado del cliente de SMTP en el servidor de correo abre la conexión TCP con el servidor de correo de Carlos

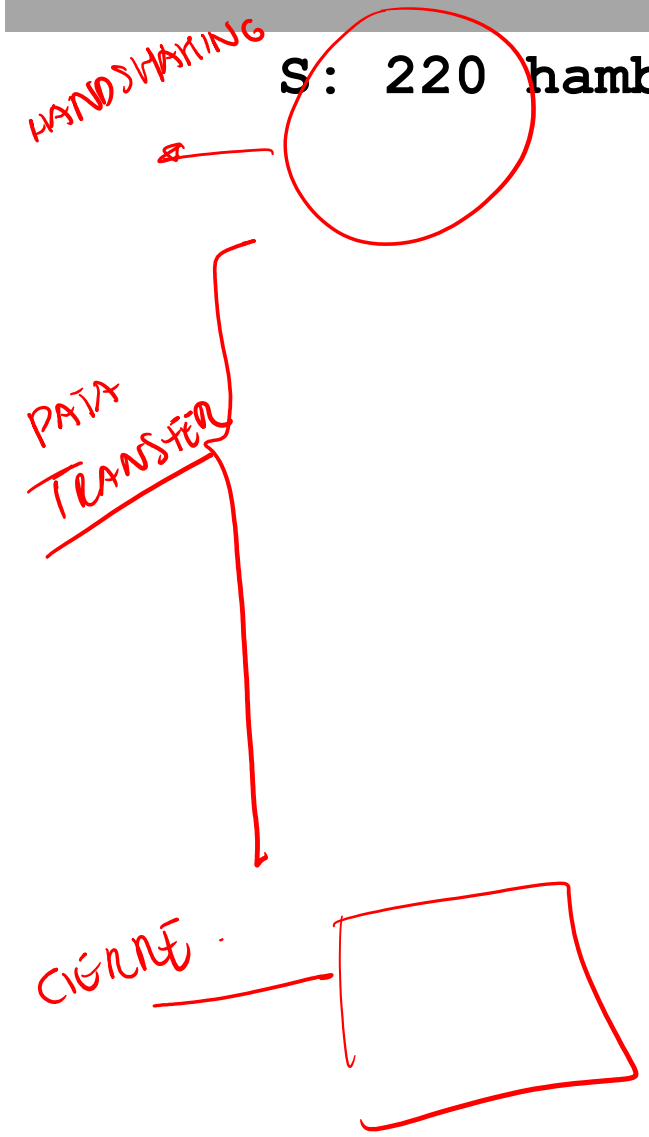
4) El cliente SMTP envía el mensaje de Andrea a través de la conexión TCP

5) El servidor de correo de Carlos coloca el mensaje en el buzón de correo.

6) Carlos invoca/abre a su UA para leer el mensaje



Ejemplo de interacción SMTP



SMTP: características y observaciones

SMTP Vs HTTP:

- HTTP: client pull
- SMTP: client push
- ambos tienen interacción comand/response ASCII, y códigos de estado
- HTTP: cada objeto encapsulado en su propio mensaje de respuesta
- SMTP: varios objetos enviados en mensaje de varias partes
- SMTP usa conexiones persistentes (Slides - Capa de aplicación parte 1)
- SMTP require que el mensaje (header & body) sea de 7-bit ASCII
- El servidor SMTP usa CRLF.CRLF para determinar el fin del mensaje

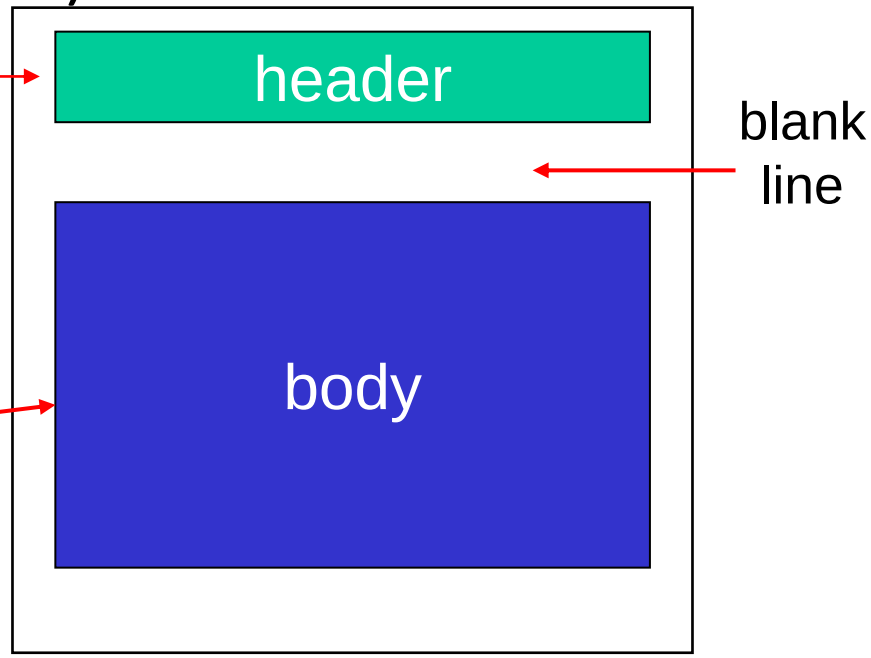
Formato de mensaje de correo

SMTP: protocolo para intercambio de mensajes e-mail, definido en RFC 5321 (así como RFC 7231 definió HTTP)

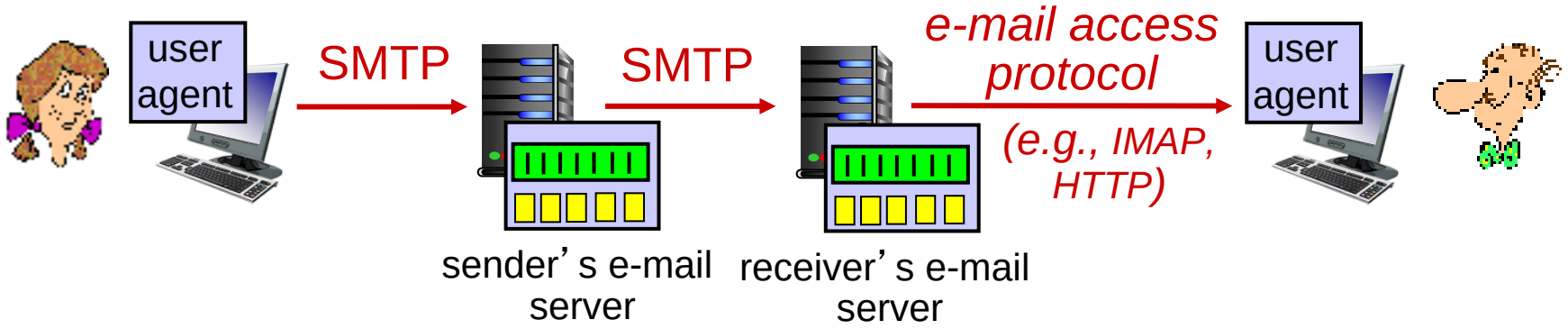
RFC 2822 define la sintáxis del mensaje de correo propiamente (así como HTML define la sintáxis de documentos web)

- header lines, e.g.,
  - To:
  - From:
  - Subject:

  Estas líneas, dentro del cuerpo del mensaje de correo electrónico, son diferentes de los comandos SMTP MAIL FROM:, RCPT TO:
- Body: cuerpo del mensaje, caracteres ASCII únicamente

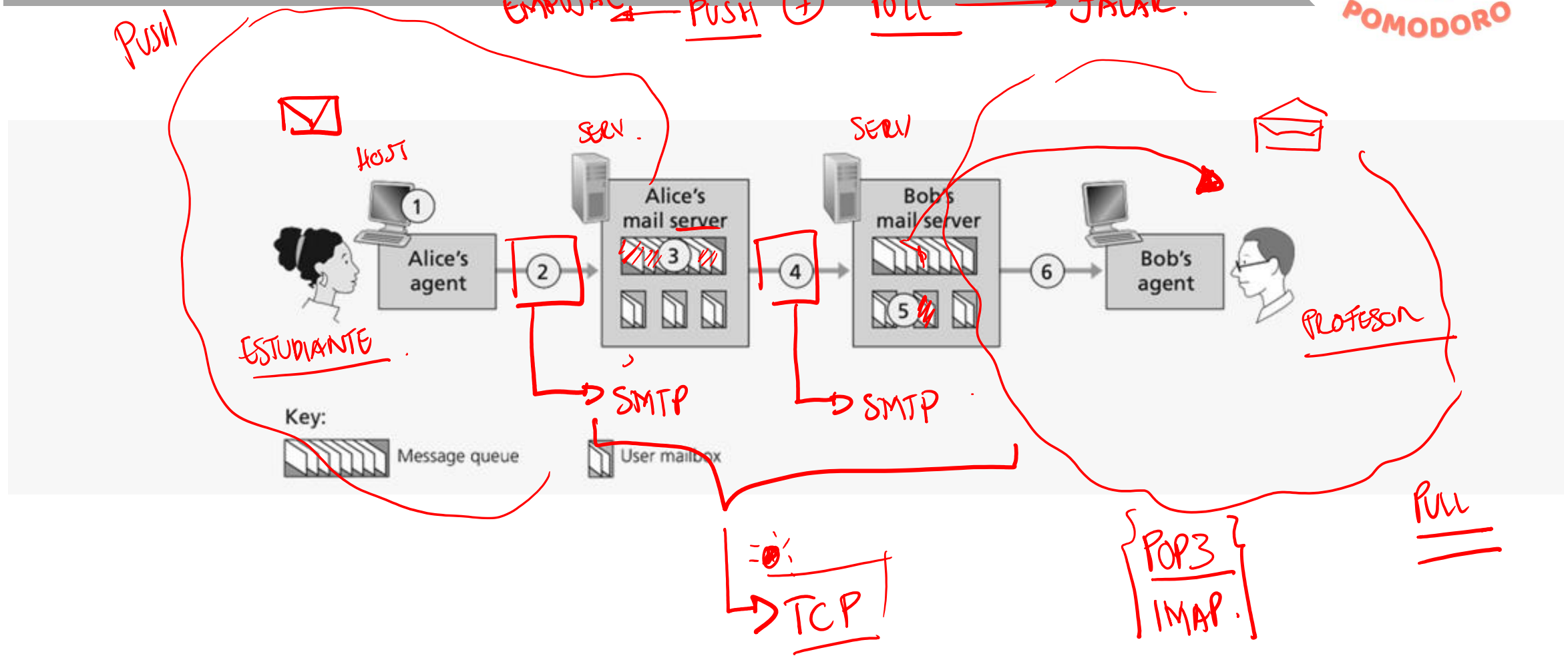


Recuperación de correo electrónico: protocolos de acceso al correo



- SMTP: entrega / almacenamiento de mensajes de correo electrónico al servidor del destinatario
- protocolo de acceso al correo: recuperación del servidor
  - IMAP: Internet Mail Access Protocol [RFC 3501]: mensajes almacenados en el servidor, IMAP proporciona recuperación, eliminación, carpetas de mensajes almacenados en el servidor
- HTTP: gmail, Hotmail, Yahoo!Mail, etc. proporciona una interfaz basada en web sobre STMP (para enviar), IMAP (o POP) para recuperar mensajes de correo electrónico

Resumen de SMTP y servicio E-Mail



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

DNS: Domain Name System

Personas: Muchos IDs:

- SSN/CC, nombre, pasaporte

Internet hosts, routers:

- IP address (32 bit) – usados para direccionar datagramas
- "nombre", (urosario.edu.co)- usados por humanos

Q: Cómo mapear entre direcciones IP y nombres (y viceversa)?

Domain Name System (DNS):

- *Base de datos distribuida* implementada en jerarquía con muchos *nombres de servidores*
- *Protocolo de capa de aplicación:* hosts, servidores DNS se comunican para *resolver* nombres (traducción de dirección/nombre)
  - *NOTA:* es el núcleo de la función de Internet, implementado como un protocolo de la capa de aplicación
  - Es complejo y difícil en la frontera de la red

DNS: servicios y estructura

DNS servicios:

- Traducción hostname-to-IP-address
- host aliasing
  - canónico, nombres alias nombres
- mail server aliasing
- Distribución de carga
  - Servidores Web replicados: muchas direcciones IP corresponden a un nombre

Q: Por qué no centralizar DNS?

- Un solo punto de falla
- Volumen de tráfico
- Base de datos centralizada
- Mantenimiento

A: No es escalable

- Solo servidores DNS de Comcast: 600.000 millones de consultas DNS/día
- Solo servidores DNS de Akamai: 2.2T consultas DNS / día

Características de DNS

Base de datos distribuida GIGANTE:

- ~ billones de registros, cada uno “simple”

Gestiona “trillones” solicicitudes por día:

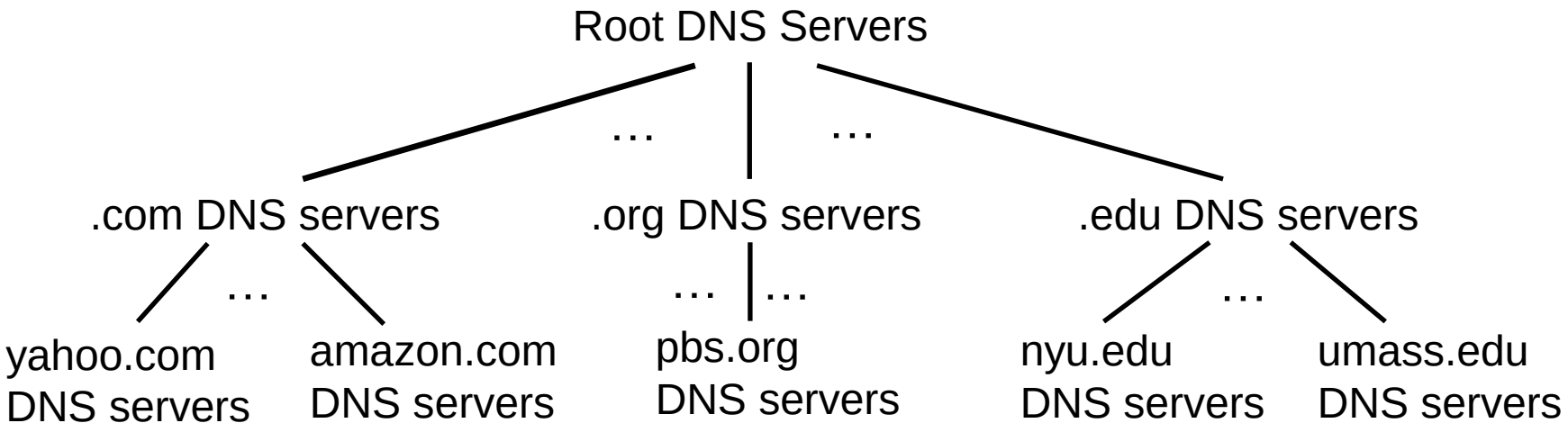
- Más lecturas que escrituras
- *Desempeño:* casi todas las interacciones en internet se hacen con DNS – (msecs)

Organizacionalmente y físicamente descentralizada:

- millones de organizaciones diferentes responsables de sus propios registros

“a prueba de balas”: confiables y segura

DNS: una base de datos jerárquica distribuida



Root

Top Level Domain

Authoritative

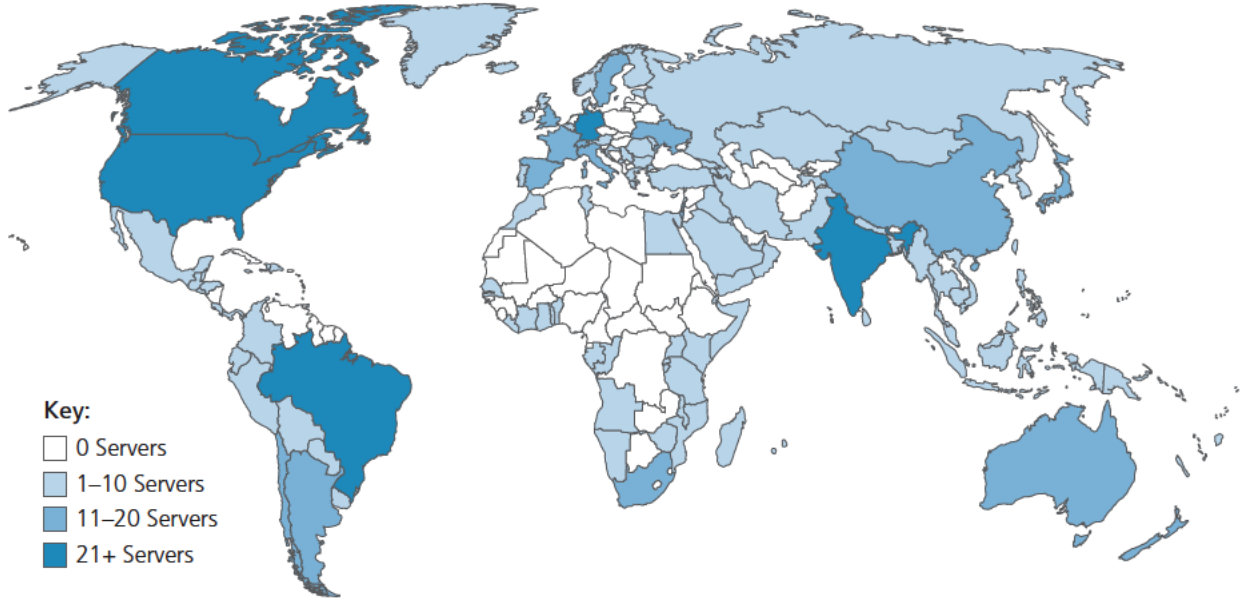
El cliente quiere una dirección IP para www.amazon.com; 1ra aproximación:

- el cliente consulta el servidor raíz para encontrar el servidor DNS .com
- el cliente consulta el servidor DNS .com para obtener el servidor DNS de amazon.com
- el cliente consulta el servidor DNS de amazon.com para obtener la dirección IP de www.amazon.com

DNS: root name servers

- oficial, contacto de último recurso por servidores que no pueden resolver el nombre
- *FUNCIÓN IMPORTANTE de* Internet
  - ¡Internet no podría funcionar sin él!
  - DNSSEC: proporciona seguridad (autenticación, integridad del mensaje)
- ICANN (Internet Corporation for Assigned Names and Numbers) gestiona dominios de root DNS

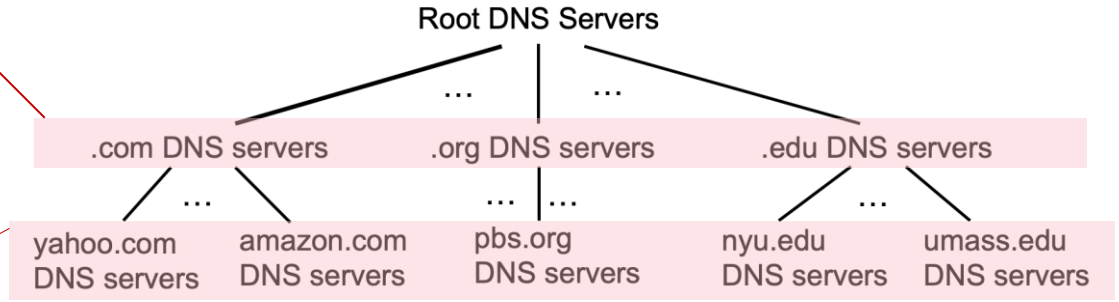
13 "servidores" de nombres raíz lógicos en todo el mundo, cada "servidor" se replicó muchas veces (~ 200 servidores en EE. UU.)



Top-Level Domain – Authorative servers

Top-Level Domain (TLD) servers:

- .com, .org, .net, .edu, .aero, .jobs, .museums .gov, todos los dominios de más alto nivel por país, e.g.: .cn, .uk, .fr, .ca, .jp, .co
- Soluciones de red: registro autorizado por .com, .net TLD
- Dominios de propósito educativo: .edu TLD



authoritative DNS servers:

- los propios servidores DNS de la organización, que proporcionan un nombre de host autorizado a las asignaciones de IP para los hosts con nombre de la organización
- puede ser mantenido por la organización o el proveedor de servicios

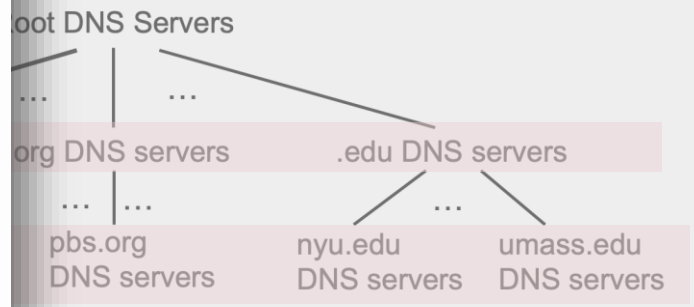
Top-Level Dom ervers

Top-Level Domain

- .com, .org, .net, .edu, s dominios de más alto nivel por país, e.g.: .c
- Soluciones de red: reg
- Dominios de propósit

authoritative DNS

- los propios servidores ionan un nombre de host autorizado a las a nombre de la organización
- puede ser mantenido e servicios



El argentino que compró el dominio de Google por menos de US$3

James Clayton
BBC, reportero de Tecnología en América del Norte

27 abril 2021

Google aún investiga cómo fue que un diseñador web local logró comprar su dominio en Argentina.

El nombre de dominio de Google Argentina fue comprado por un diseñador web mientras el sitio estuvo caído durante dos horas en ese país, el miércoles pasado.

Nicolás Kuroña, de 30 años, dijo que logró comprar Google.com.ar a través de un proceso legal normal.

Nombres de servidores DNS locales

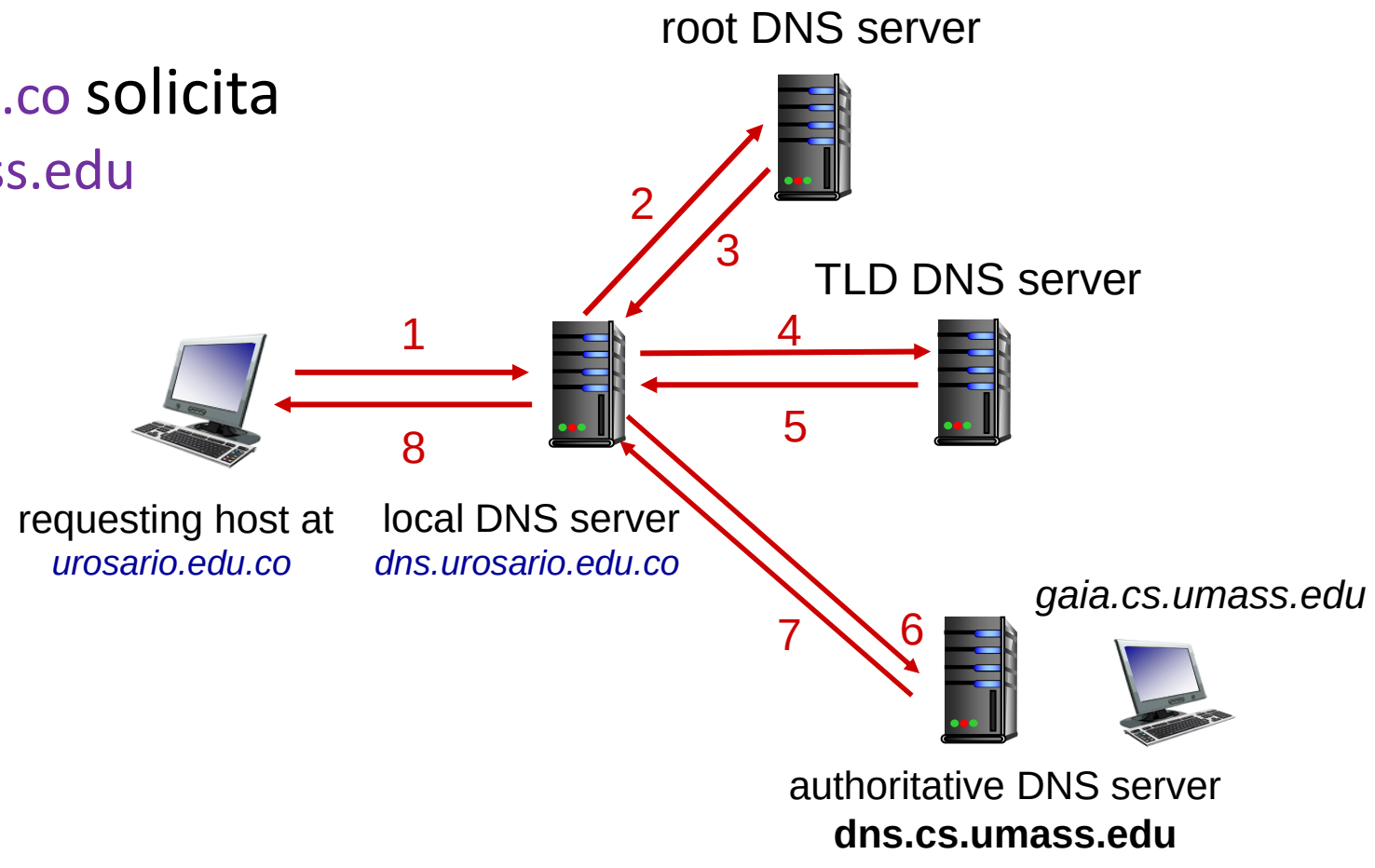
- cuando el host realiza una consulta de DNS, se envía a su servidor DNS local
- El servidor DNS local devuelve la respuesta, respondiendo:
  - de su caché local de traducciones recientes de nombre/dirección (casi siempre desactualizados)
  - reenviar la solicitud a la jerarquía de DNS para su resolución
  - cada ISP tiene un servidor de nombres DNS local; para encontrar el tuyo:
    - MacOS: `% scutil --dns`
    - Windows: `>ipconfig /all`
- el servidor DNS local no pertenece estrictamente a la jerarquía

Resolución de nombres DNS: consulta iterada

Ejemplo: host en urosario.edu.co solicita la dirección IP de gaia.cs.umass.edu

Consulta iterada:

- El servidor contactado responde con el nombre del servidor para contactar
- "No sé este nombre, pero pregúntale a este servidor"

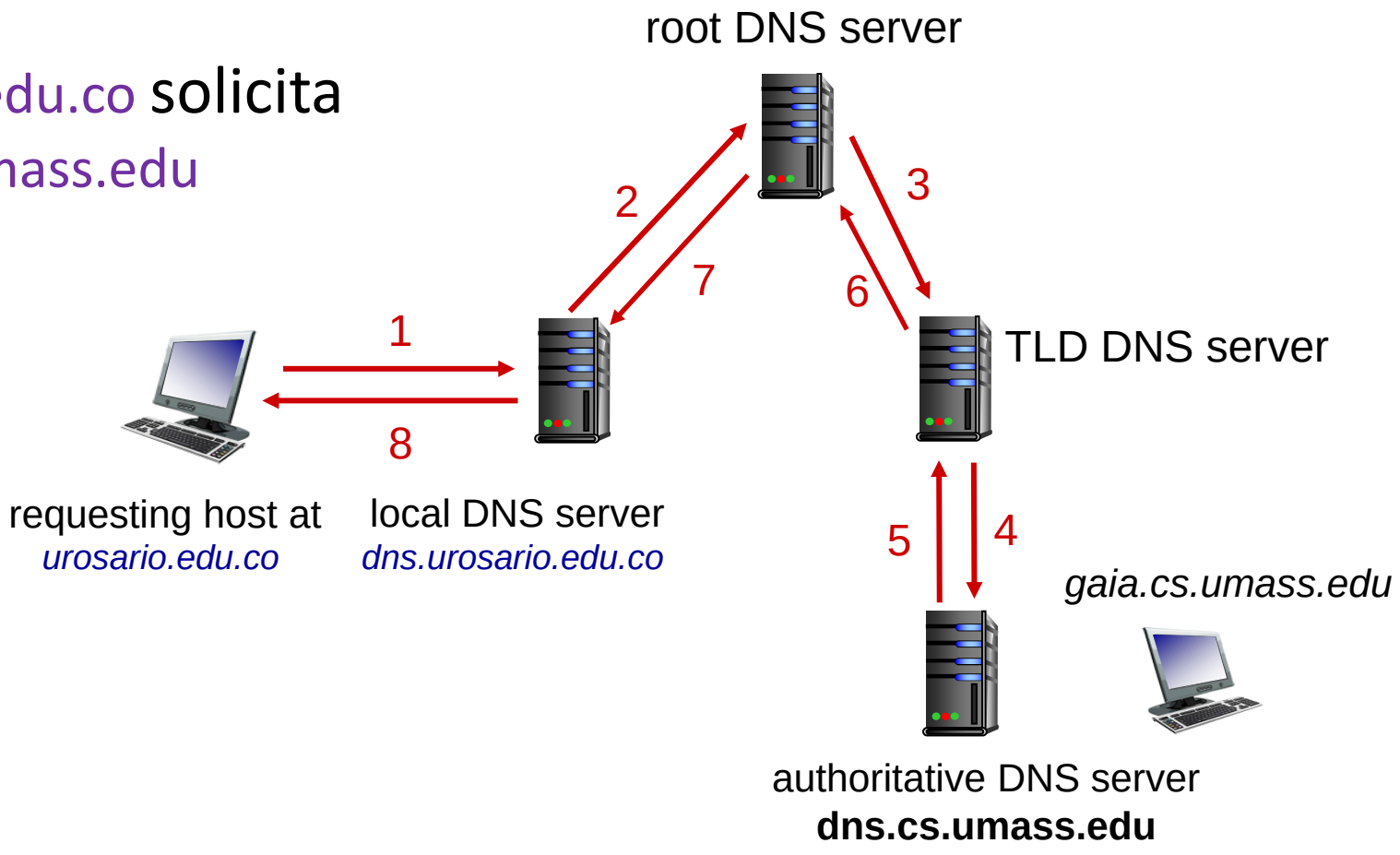


Resolución de nombres DNS: consulta recursiva

Ejemplo: host en urosario.edu.co solicita la dirección IP de gaia.cs.umass.edu

Consulta recursiva:

- pone la carga de la resolución de nombres en el servidor de nombres contactado
- carga pesada en los niveles superiores de jerarquía (?)



Información DNS almacenada en Caché

- una vez que (cualquier) servidor de nombres aprende el mapeo, *almacena en caché* el mapeo e *inmediatamente* devuelve un mapeo en caché en respuesta a una consulta
  - el almacenamiento en caché mejora el tiempo de respuesta
  - tiempo de espera de las entradas de caché (desaparecen) después de algún tiempo (TTL)
  - Los servidores TLD normalmente se almacenan en caché en servidores de nombres locales.
- Los datos en cache pueden estar *desactualizados*
  - Si el host con nombre cambia la dirección IP, es posible que no se conozca en todo Internet hasta que expiren todos los TTL.
  - *¡Mejor traducción de nombre/dirección!*

Registros DNS

DNS: base de datos distribuida que almacena registros de recursos (RR)

RR format: (`name, value, type, ttl`)

type=A

- `name` = hostname
- `value` = IP address

type=NS

- `name` = domain (e.g., foo.com)
- `value` = hostname of authoritative name server for this domain

type=CNAME

- `name` = nombre de alias para algún nombre "canónico" (el nombre real)
- www.ibm.com ES REALMENTE servereast.backup2.ibm.com
- `value` nombre canónico

type=MX

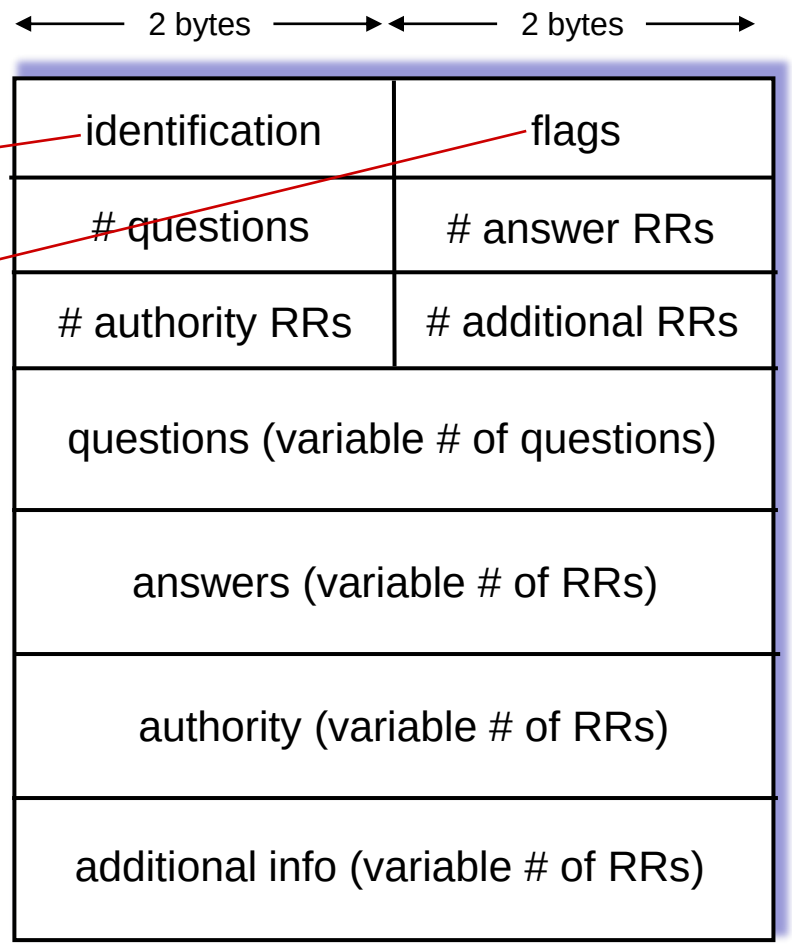
- `value` = nombre del servidor de correo SMTP asociado con el nombre (`name`)

DNS: Protocolo de mensajes

Solicitud y *respuesta* de mensajes DNS, comparten el mismo *formato:*

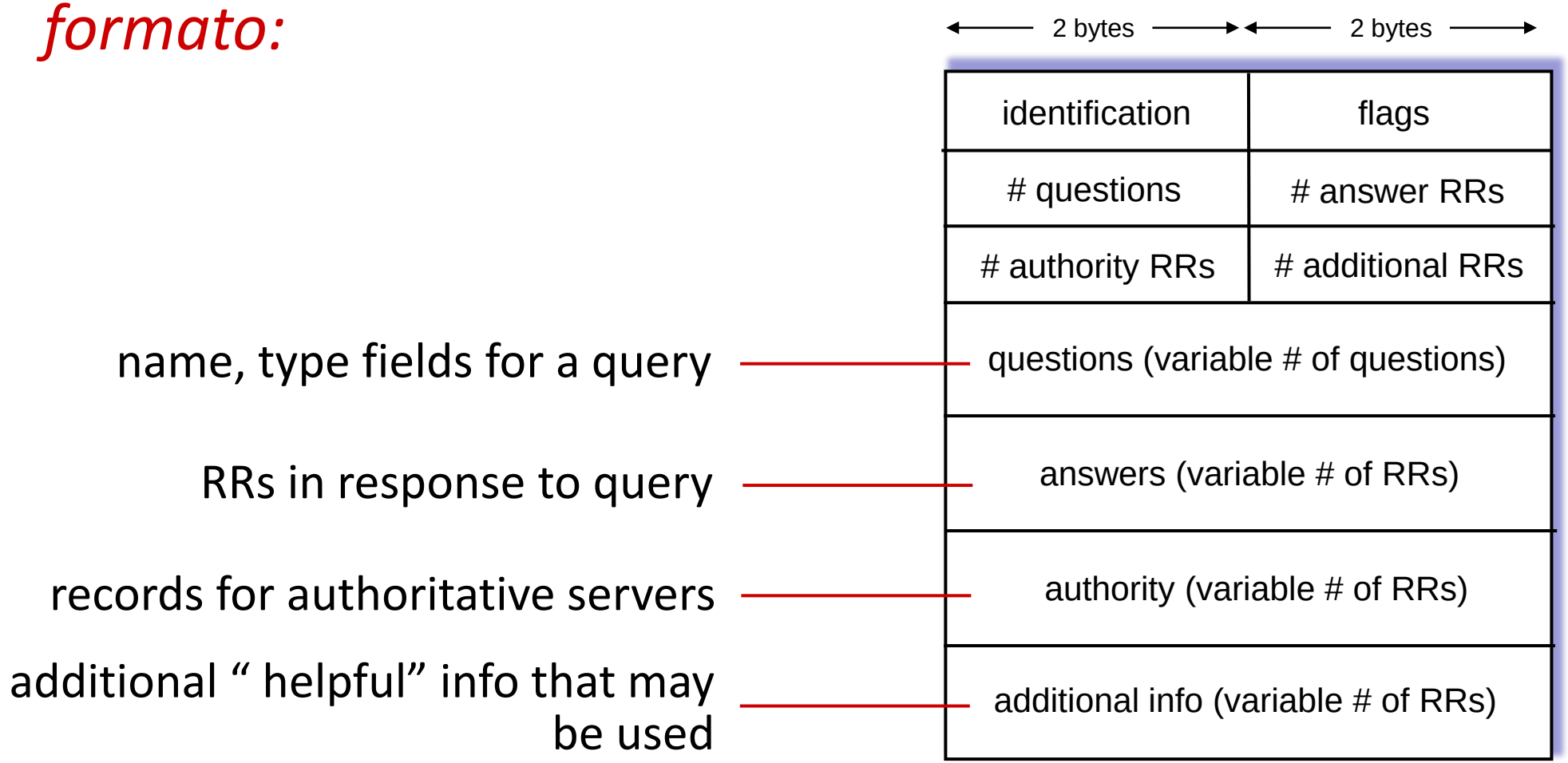
message header:

- identification: 16 bit # por solicitud, y la respuesta usa el mismo #
- flags:
  - query or reply
  - recursion desired
  - recursion available
  - reply is authoritative



DNS: Protocolo de mensajes

Solicitud y *respuesta* de mensajes DNS, comparten el mismo *formato:*



Ingresar su información en el DNS

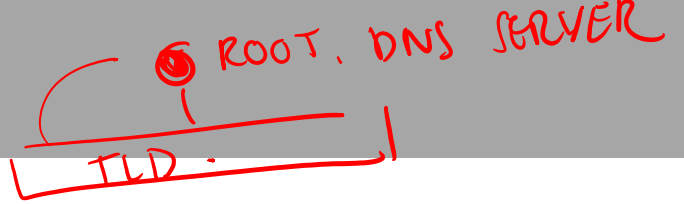
Ejemplo: nueva startup "MACCSHINE"

- Registrar el nombre macchine.com en un *registro DNS* (e.g., Network Solutions) GO DADDY | COLOMBIA HOSTING . . .
  - proporcionar nombres, direcciones IP del servidor de nombres autorizado (primario y secundario)
  - El registrador inserta NS, A RRs en el servidor TLD .com:

(maccshine.com, dns1.maccshine.com, NS)
(dns1.maccshine.com, 212.212.212.1, A)

- crear un servidor autorizado localmente con dirección IP 212.212.212.1
  - Registro type A para www. maccshine.com
  - Registro type MX para maccshine.com

DNS: Seguridad



Ataques DDoS (distributed denial-of-service)

- bombardear servidores raíz con tráfico
  - no ha tenido éxito hasta la fecha
  - filtrado de tráfico
  - Los servidores DNS locales almacenan en caché las direcciones IP de los servidores TLD, lo que permite omitir el servidor raíz.
- Bombardear servidores TLD
  - Potencialmente más peligroso

Spoofing

- Intercepta solicitudes DNS devolviendo respuestas falsas
  - DNS envenamiento de caché
  - RFC 4033: DNSSEC autenticación de servicios

HANDS-ON

- 0_PTTA_Intro_1
- 0_PTTA_Intro_2

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)

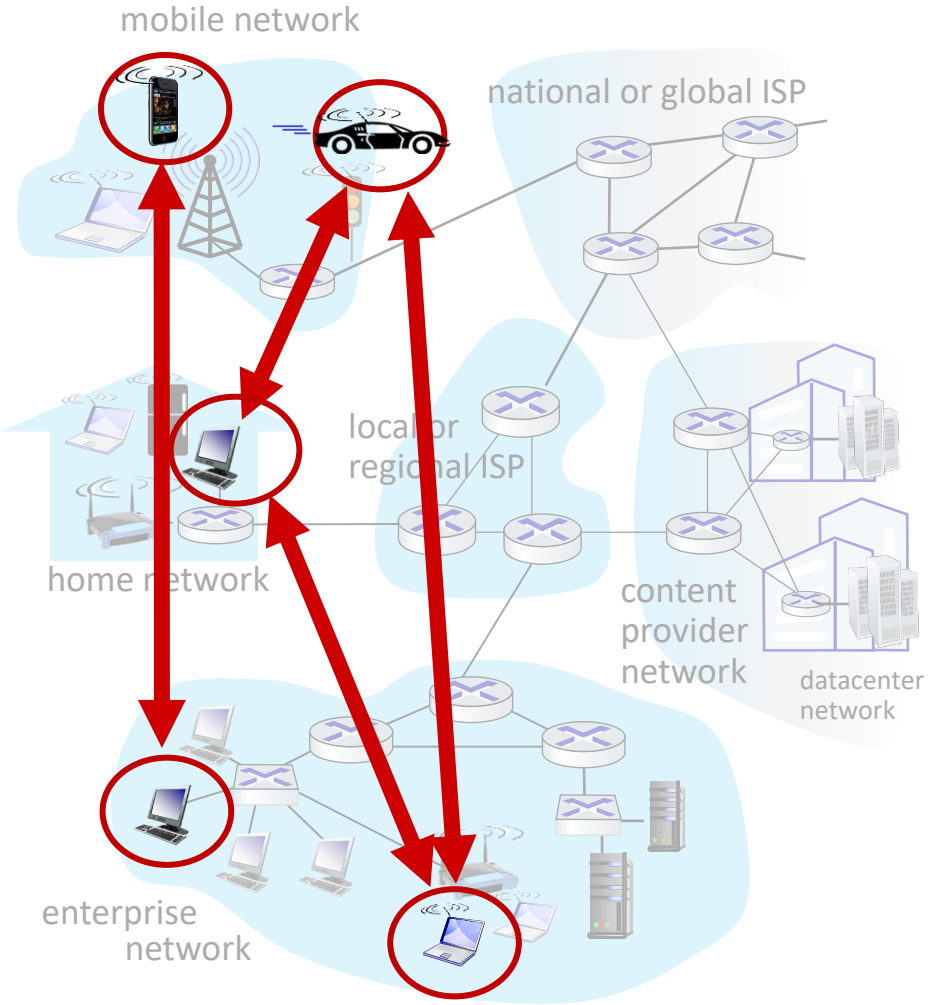
Streaming de video y redes de distribución de contenido

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

Taller Packet Tracer – Network Services

Arquitectura P2P

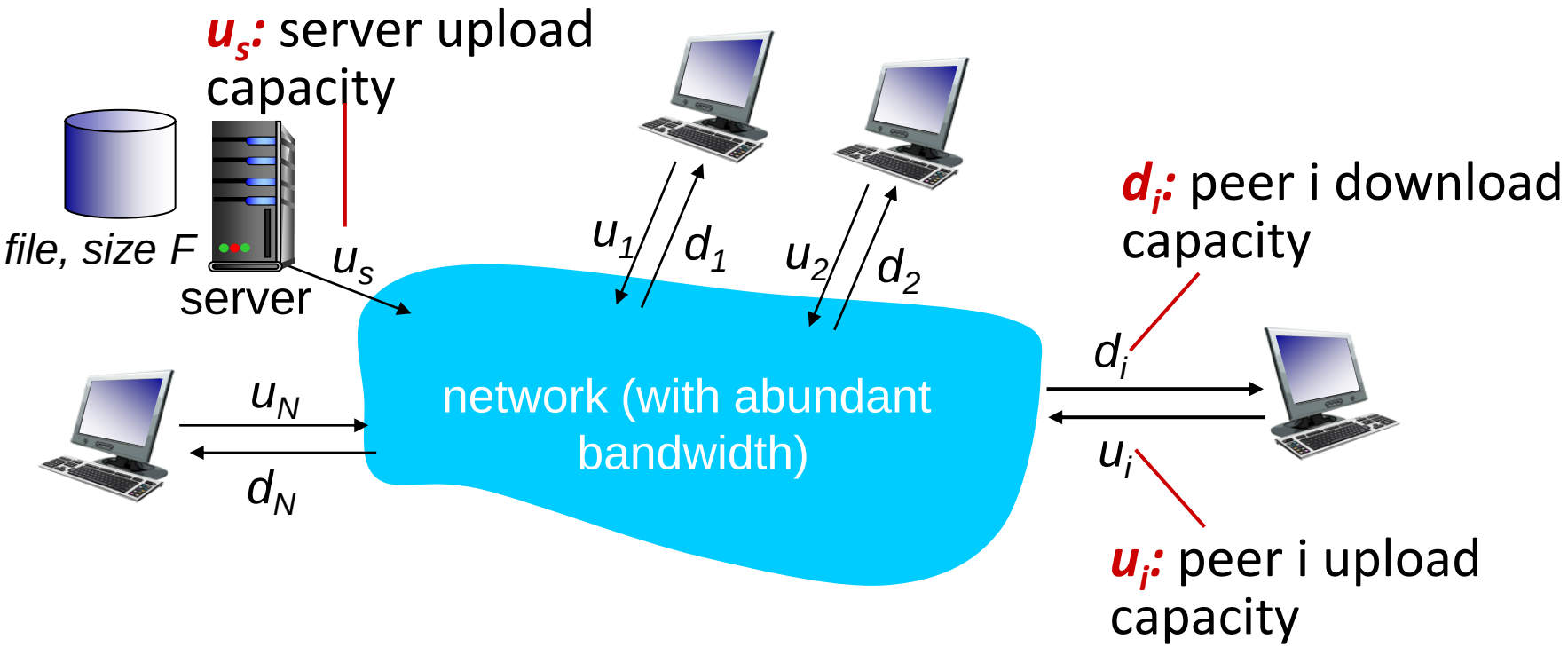
- *sin servidor siempre activo*
- *los sistemas finales arbitrarios se comunican directamente*
- *los pares solicitan el servicio de otros pares, brindan servicio a cambio de otros pares*
- *autoescalabilidad: los nuevos pares aportan nueva capacidad de servicio y nuevas demandas de servicio*
- Los pares están conectados de forma intermitente y cambian las direcciones IP
- gestión compleja
- **ejemplos:** intercambio de archivos P2P (BitTorrent), transmisión (KanKan), VoIP (Skype)



Distribución de archivos: Client-server vs P2P

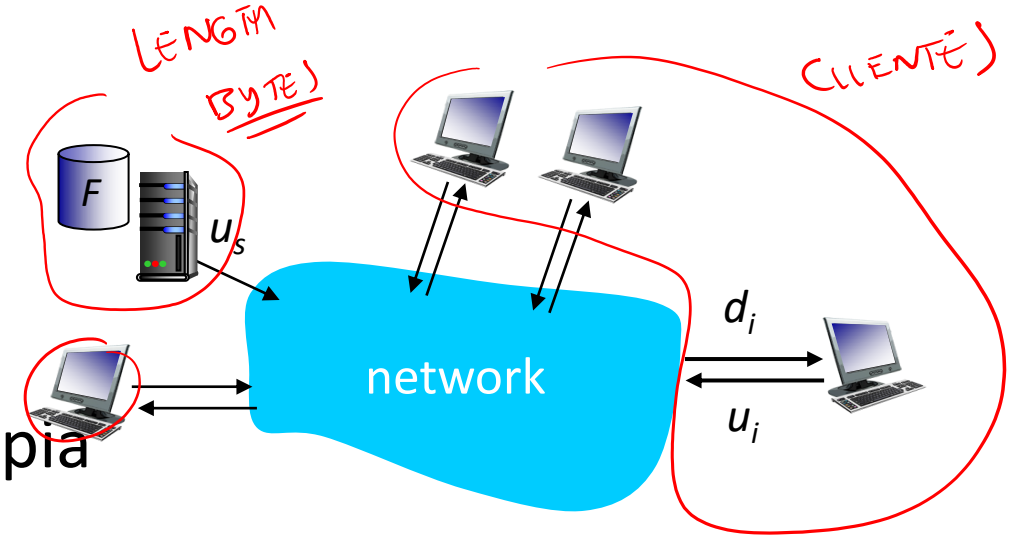
Q: ¿Cuánto tiempo tarda distribuir el archivo (tamaño F) desde un servidor a N pares?

- la capacidad de carga / descarga de pares es un recurso limitado



Distribución de archivos: Client-server vs P2P

- *server transmission:* debe enviar (cargar) de archivos secuencialmente: N copias
  - Tiempo para enviar una copia: F/u_s
  - Tiempo para enviar N copias: NF/u_s
- *client:* cada cliente debe descargar la copia del archivo
  - d_{min} = tasa min de descarga del cliente
  - Tiempo min de descarga del cliente: F/d_{min}



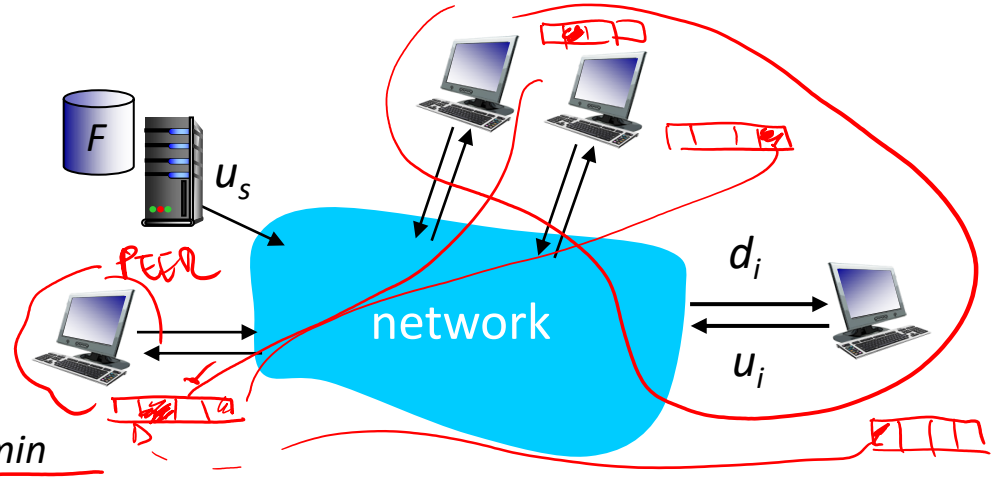
Tiempo para distribuir F a N clientes usando Arquitectura client-server

$$D_{c\text{-}s} \geq max\{NF/u_s, F/d_{min}\}$$

Incrementa linealmente en N

Distribución de archivos: Client-server vs P2P

- *server transmission:* debe cargar al menos una copia:
  - Tiempo para enviar una copia: F/u_s
- *client:* cada cliente debe descargar una copia
  - Tiempo ´min de descarga por cliente: F/d_{min}
- *clients:* como agregado debe descargar NF bits
  - Max tasa de carga (limitando la max tasa de descarga) es $u_s + \Sigma u_i$



Tiempo para distribuir F a N clientes usando P2P

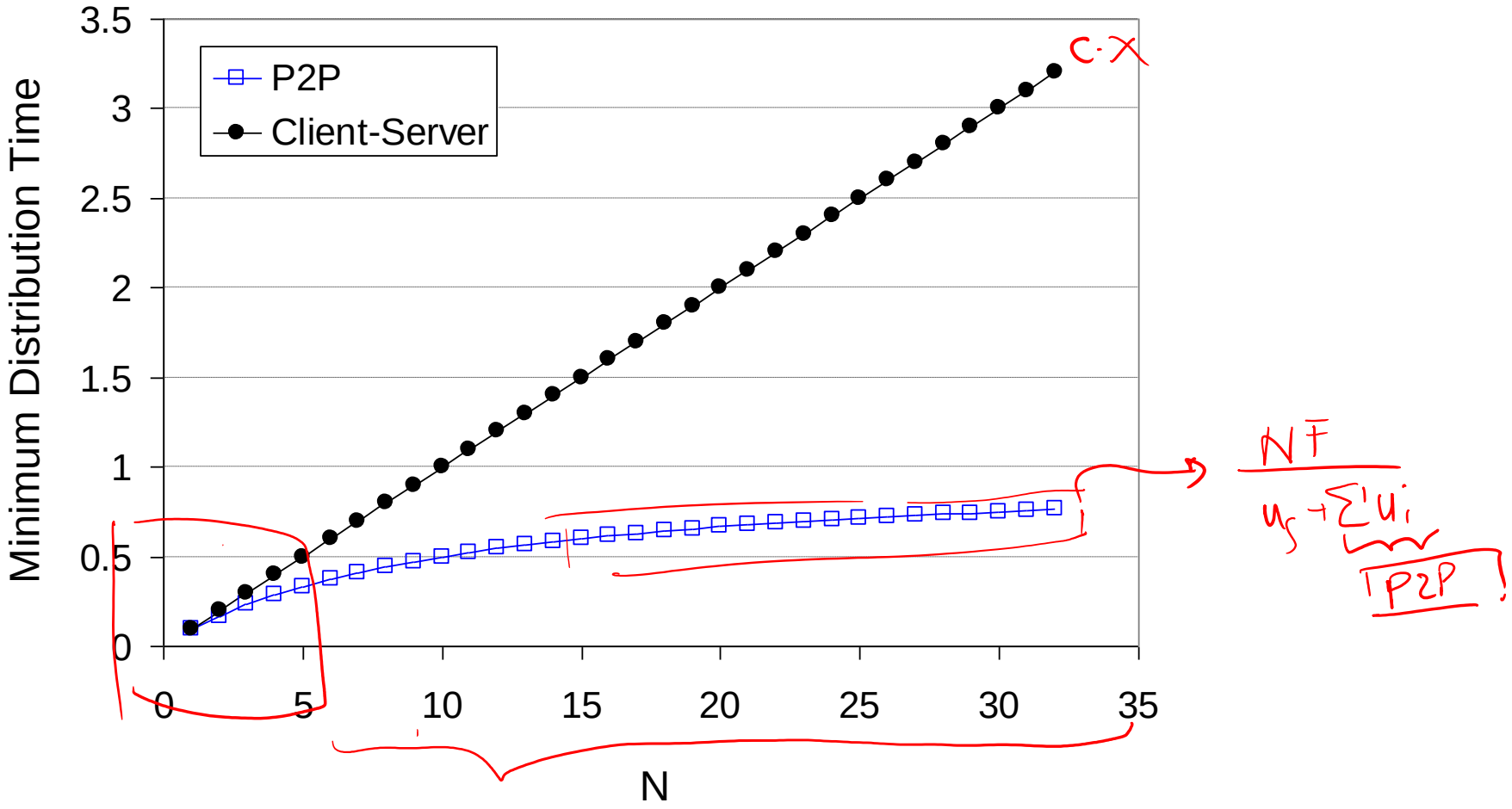
$$D_{P2P} \geq max\{F/u_s, F/d_{min}, NF/(u_s + \Sigma u_i)\}$$

Incrementa linealmente en N …

… pero también disminuye, ya que cada par aporta capacidad de servicio

Client-server vs. P2P: ejemplo

Tasa de carga por cliente= u, F/u = 1 hour, $u_s = 10u$, $d_{min} \geq u_s$

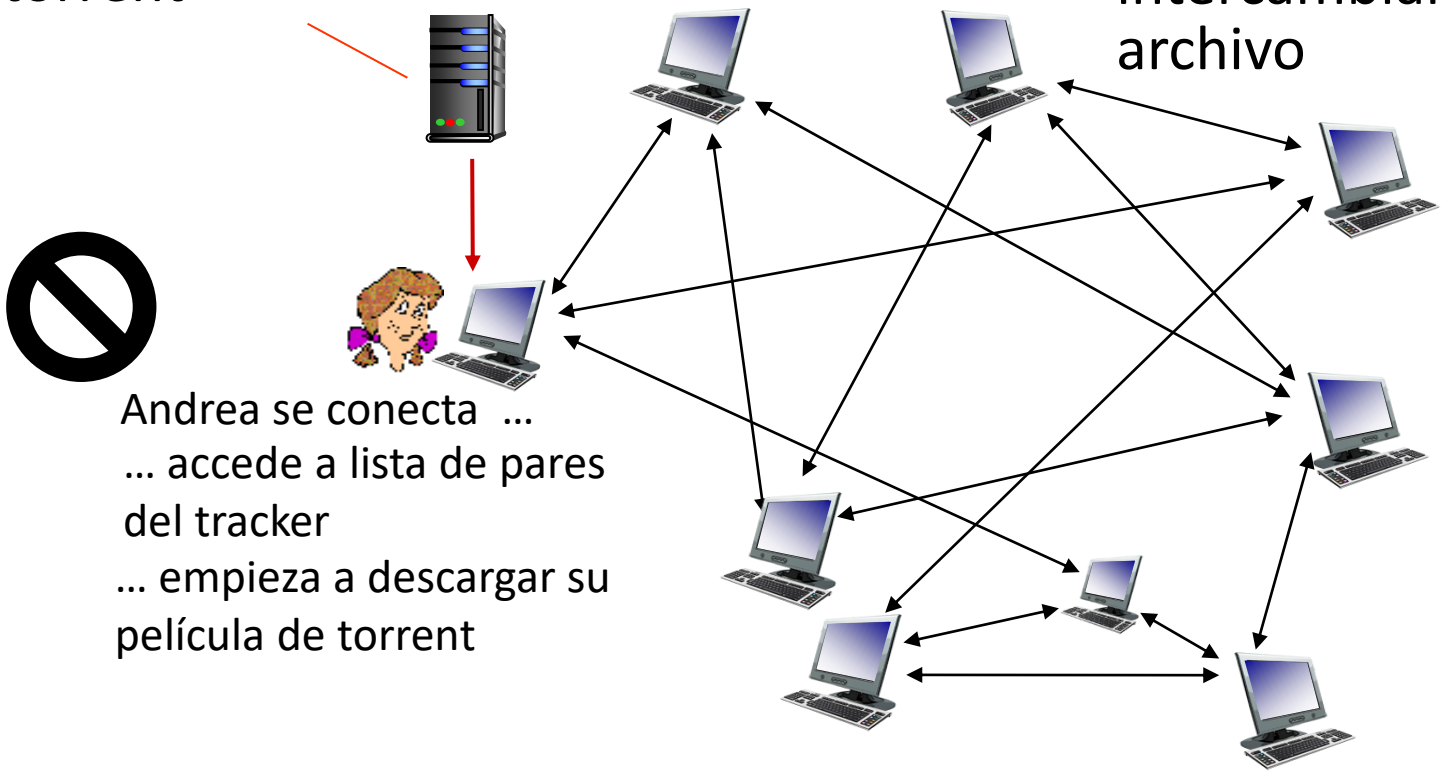


P2P distribución de archivos: BitTorrent

- archivo dividido en trozos de 256 Kb
- pares en fragmentos de archivos de envío/recepción de torrent

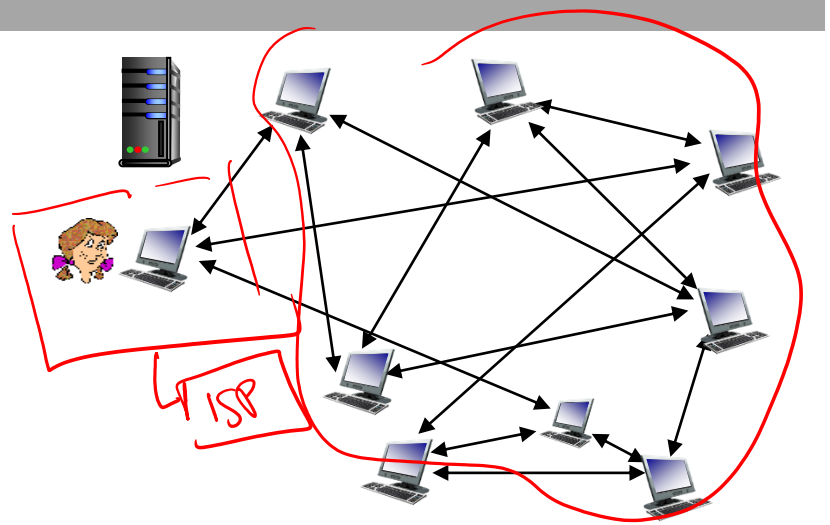
tracker: rastrea pares vinculados al torrent

torrent: agrupa pares intercambiando trozos del archivo



Andrea se conecta …
… accede a lista de pares del tracker
… empieza a descargar su película de torrent

P2P distribución de archivos: BitTorrent



- El par cuando se une a torrent:
  - no tiene fragmentos, pero los acumulará con el tiempo de otros pares
  - se registra con el rastreador para obtener una lista de pares, se conecta a un subconjunto de pares ("vecinos")
- durante la descarga, los pares cargan fragmentos a otros pares
- el par puede cambiar a los pares con los que intercambia fragmentos
- *abandono:* los compañeros pueden ir y venir
- una vez que el par tiene el archivo completo, puede (egoístamente) irse o (altruistamente) permanecer en torrent

BitTorrent: solicitando y enviando trozos de archivos

Solicitud:

- en un momento dado, diferentes pares tienen diferentes subconjuntos de fragmentos de archivos
- periódicamente, Andrea le pide a cada compañero una lista de los fragmentos que tienen
- Andrea solicita fragmentos faltantes a sus compañeros, los más raros primero

Envío: tit-for-tat (“ojo por ojo”)

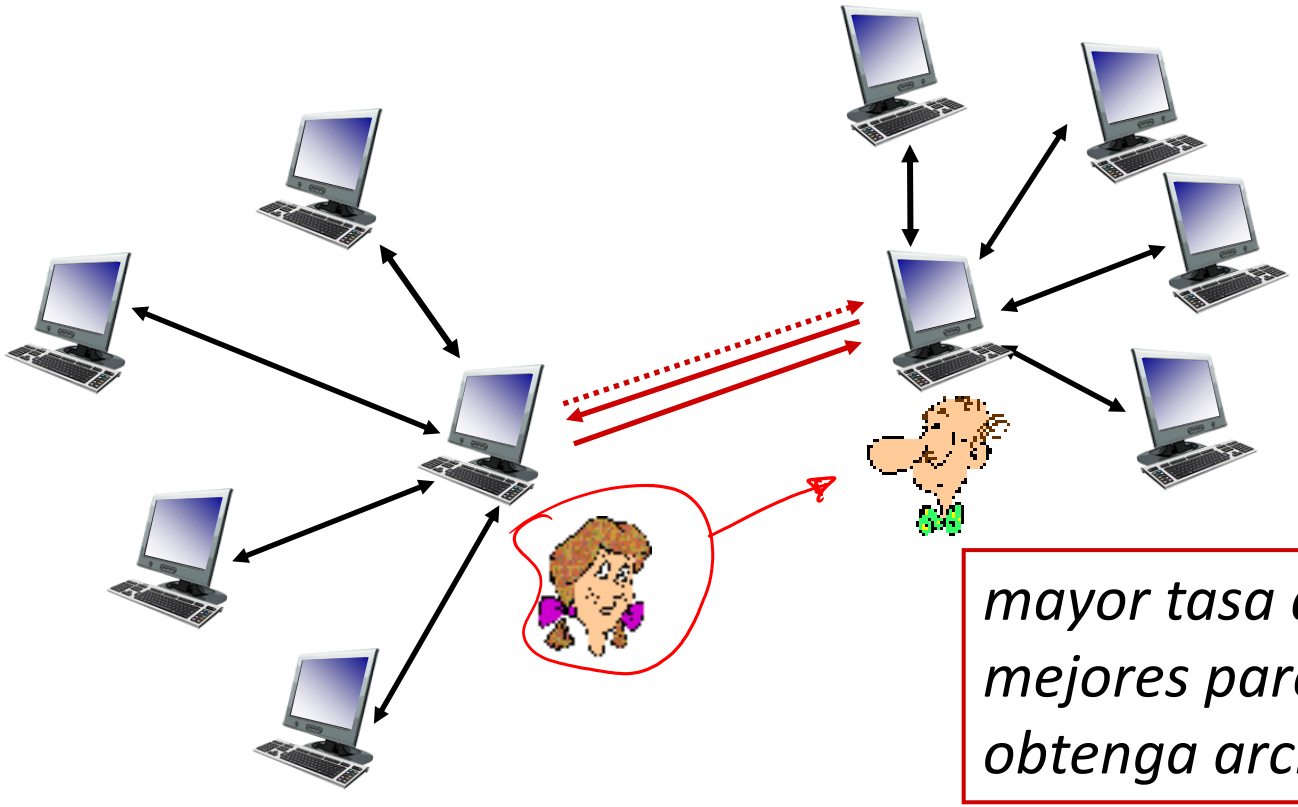
- Andrea envía fragmentos a esos cuatro compañeros que actualmente envían sus fragmentos a la tasa más alta.
- otros compañeros son ahogados por andrea (no reciben trozos de ella)
- reevaluar los 4 primeros cada 10 segundos
- cada 30 segundos: selecciona aleatoriamente otro par, comienza a enviar fragmentos“
- Libera de forma optimista" a este compañero
- el compañero recién elegido puede unirse al top 4

BitTorrent: tit-for-tat ("ojo por ojo")

(1) Andrea "de manera optimista desbloquea" a Carlos

(2) Andrea es ahora una de las top-4

(3) Carlos también entra en el top.4 de Andrea



mayor tasa de carga: encuentre mejores pares de intercambio, obtenga archivos más rápido!

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

Video Streaming y CDNs: contexto

- stream video traffic: es el mayor consume de ancho de banda de internet
  - Netflix, YouTube, Amazon Prime: 80% del trafico residencial (ISP 2020)
- *Retos:*
- Escalabilidad: Cantidad de usuarios aumenta
- Heterogeneidad: diferentes usuarios tienen diferentes capacidades (p. ej., cableado versus móvil; ancho de banda grande Vs ancho de banda pobre)

NETFLIX

迅雷看看 www.kankan.com 网络高清影院

Akamai

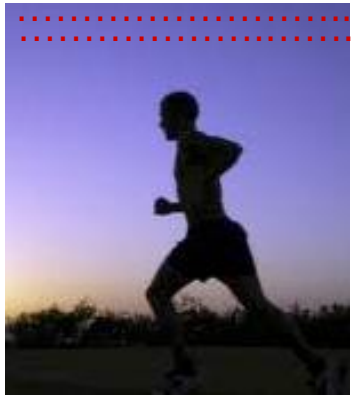
YouTube

Solución: Infrastructura de nivel de aplicación distribuida

Multimedia: video

- video: secuencia de imágenes mostradas a velocidad constante
  - p. ej., 24 imágenes/seg.
- imagen digital: matriz de píxeles donde cada píxel es representado por bits
- codificación: usa la redundancia *dentro y entre* imágenes para disminuir # bits utilizados para codificar la imagen
  - espacial (dentro de la imagen)
  - temporal (de una imagen a la siguiente)

Ejemplo de codificación espacial: en lugar de enviar N valores del mismo color (todo morado), envía solo dos valores: valor de color (morado) y número de valores repetidos (N)



frame *i*

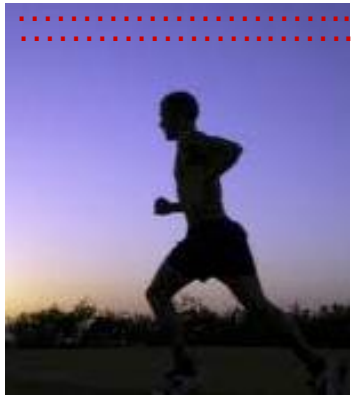
frame *i+1*

Ejemplo de codificación temporal: en lugar de enviar una trama completa en i + 1, envía solo las diferencias de la trama i

Multimedia: video

- CBR: (constant bit rate): tasa de codificación de video fija
- VBR: (variable bit rate): La tasa de codificación de video cambia a medida que la cantidad de codificación espacial y temporal cambian
- Ejemplos:
  - MPEG 1 (CD-ROM) 1.5 Mbps
  - MPEG2 (DVD) 3-6 Mbps
  - MPEG4 (usado en internetInternet, 64Kbps – 12 Mbps)

Ejemplo de codificación espacial: en lugar de enviar N valores del mismo color (todo morado), envía solo dos valores: valor de color (morado) y número de valores repetidos (N)



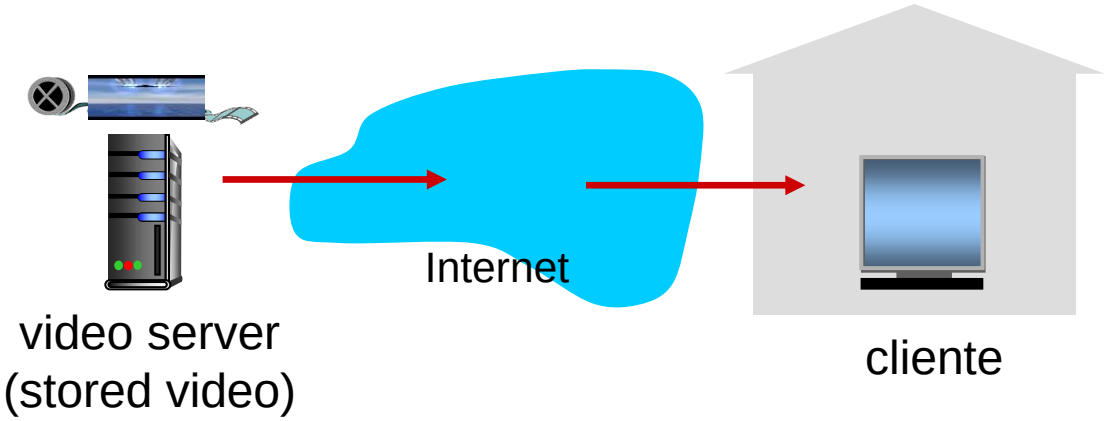
frame *i*

frame *i+1*

Ejemplo de codificación temporal: en lugar de enviar una trama completa en i + 1, envía solo las diferencias de la trama i

Streaming stored video

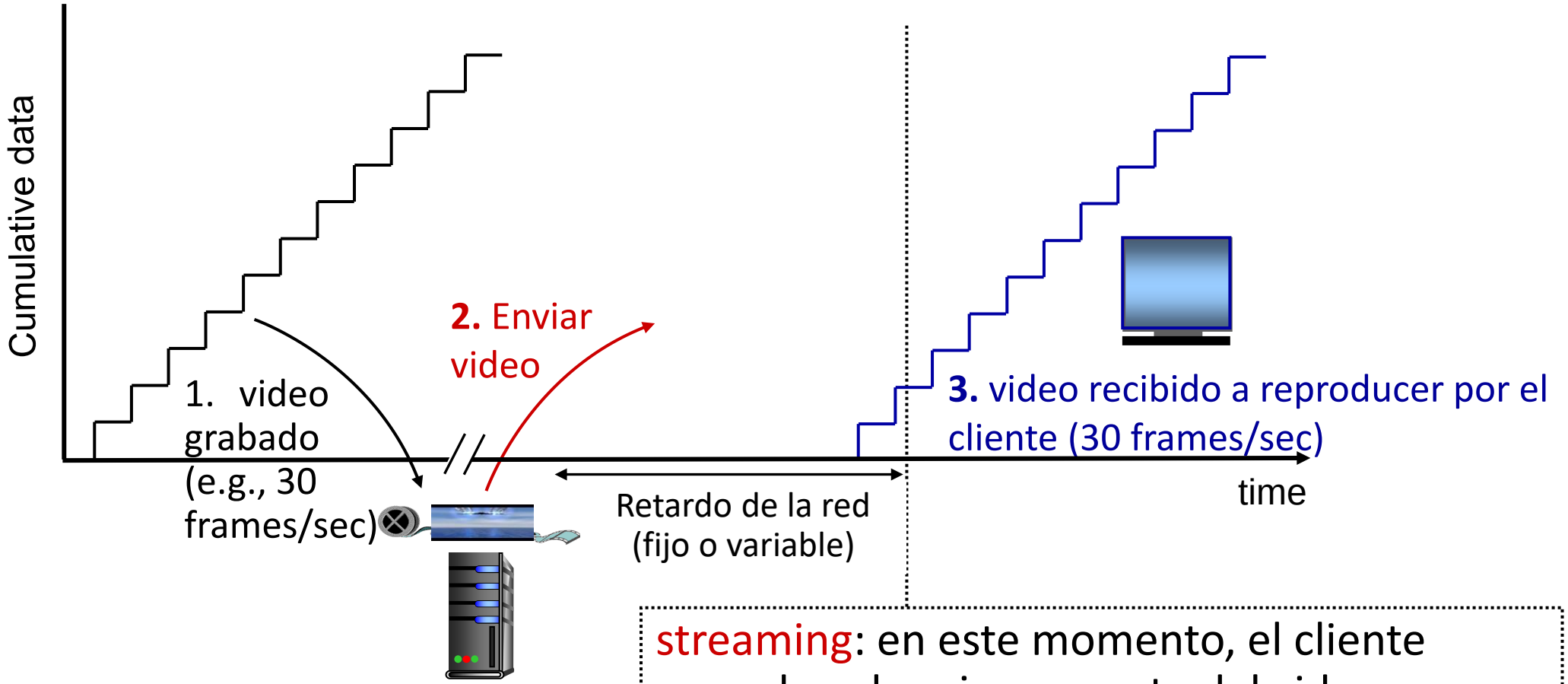
Escenario básico:



Retos:

- El ancho de banda de servidor a cliente variará con el tiempo, con cambios en los niveles de congestión de la red (interna, red de acceso, núcleo de red, servidor de video)
- La pérdida de paquetes, el retraso debido a la congestión retrasará la reproducción o dará como resultado una mala calidad de video

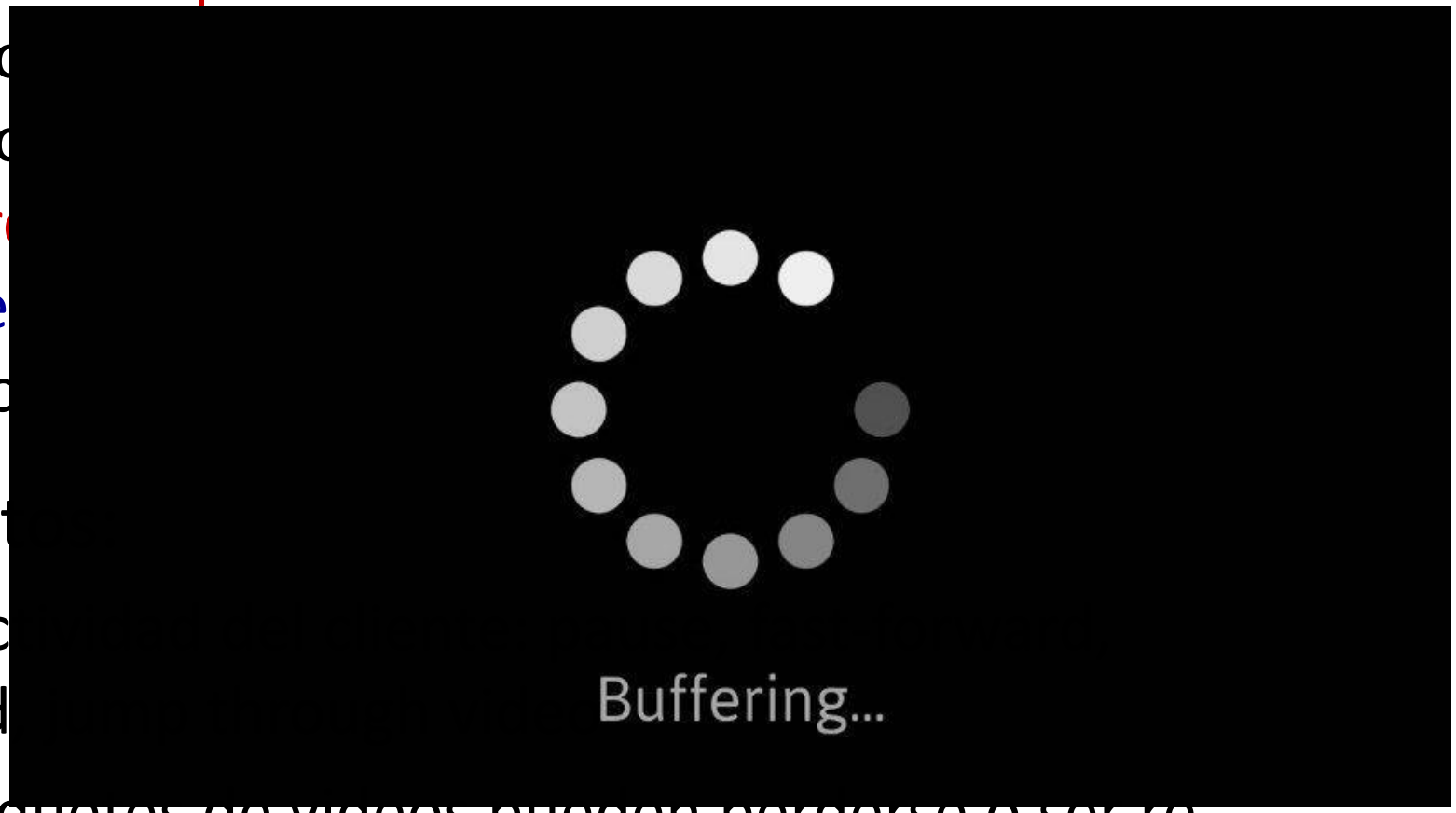
Streaming stored video



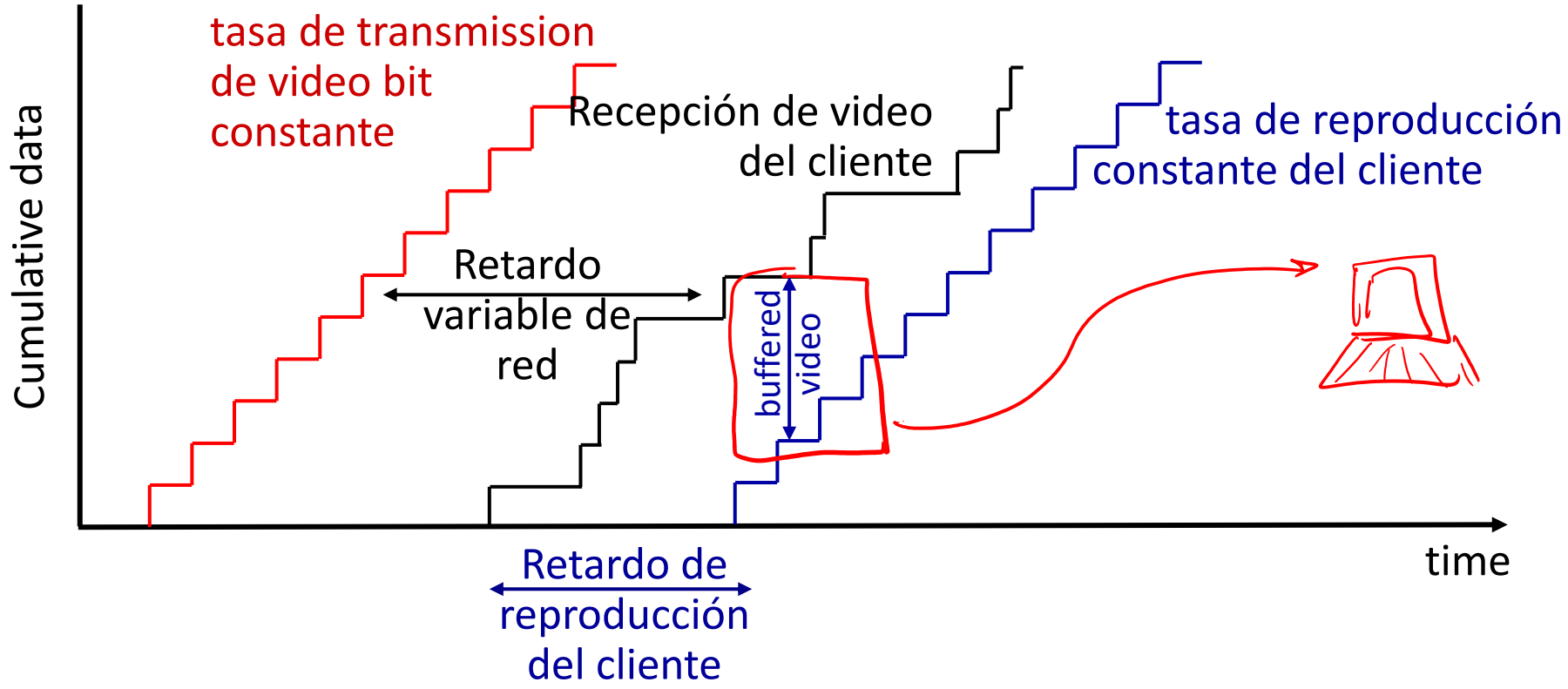
streaming: en este momento, el cliente reproduce la primera parte del video, mientras que el servidor aún envía la parte posterior del video

Streaming stored video: retos

- restricción de reproducción continua: durante la reproducc
  reproducc
  pero los r
  que se ne
  coincida c
- Otros re
  - Interac
    rewind
  - Los paquetes de videos pueden perderse o ser re
    transmitidos



Streaming stored video: playout buffering



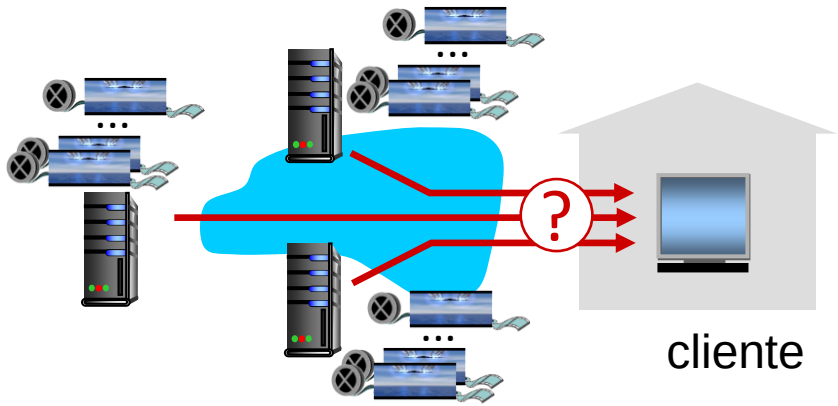
- *búfer del lado del cliente y retardo de reproducción:* compensar el retardo agregado a la red, la fluctuación de retardo (jitter)

Streaming multimedia: DASH

*D*ynamic, *A*daptive
*S*treaming over *H*TTP

servidor:

- divide el archivo de video en varios fragmentos
- cada fragmento codificado a varias velocidades diferentes
- diferentes codificaciones de velocidad almacenadas en diferentes archivos
- archivos replicados en varios nodos CDN
- **archivo de manifiesto:** proporciona URL para diferentes fragmentos

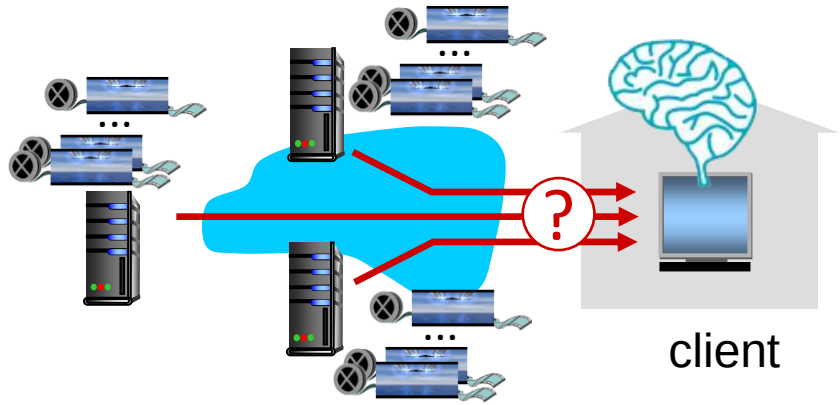


cliente:

- estima periódicamente el ancho de banda de servidor a cliente
- consulta el **manifiesto**, solicita un fragmento a la vez
  - elige la tasa de codificación máxima sostenible dado el ancho de banda actual
  - Puede elegir diferentes tasas de codificación en diferentes momentos (dependiendo del ancho de banda disponible en ese momento) y de diferentes servidores.

Streaming multimedia: DASH

- *"inteligencia"* en el cliente: determina
  - *Cuándo* solicitar un fragmento (para que no se produzca falta de búfer o desbordamiento)
  - *Qué tasa de codificación* solicitar (mayor calidad cuando hay más ancho de banda disponible)
  - *Dónde* solicitar un fragmento (puede solicitarlo desde el servidor de URL que está "cerca" del cliente o tiene un alto ancho de banda disponible)



Streaming video = encoding + DASH + playout buffering

Content distribution networks (CDNs)

Reto: ¿Cómo transmitir contenido (seleccionado entre millones de videos) a cientos de miles de usuarios simultáneos?

- *Opción 1:* "mega-servidor" único y grande
- punto único de fallo
- punto de congestión de la red
- camino largo (y posiblemente congestionado) a clientes distantes

Solución simple y básica pero…*no es escalable!*

Content distribution networks (CDNs)

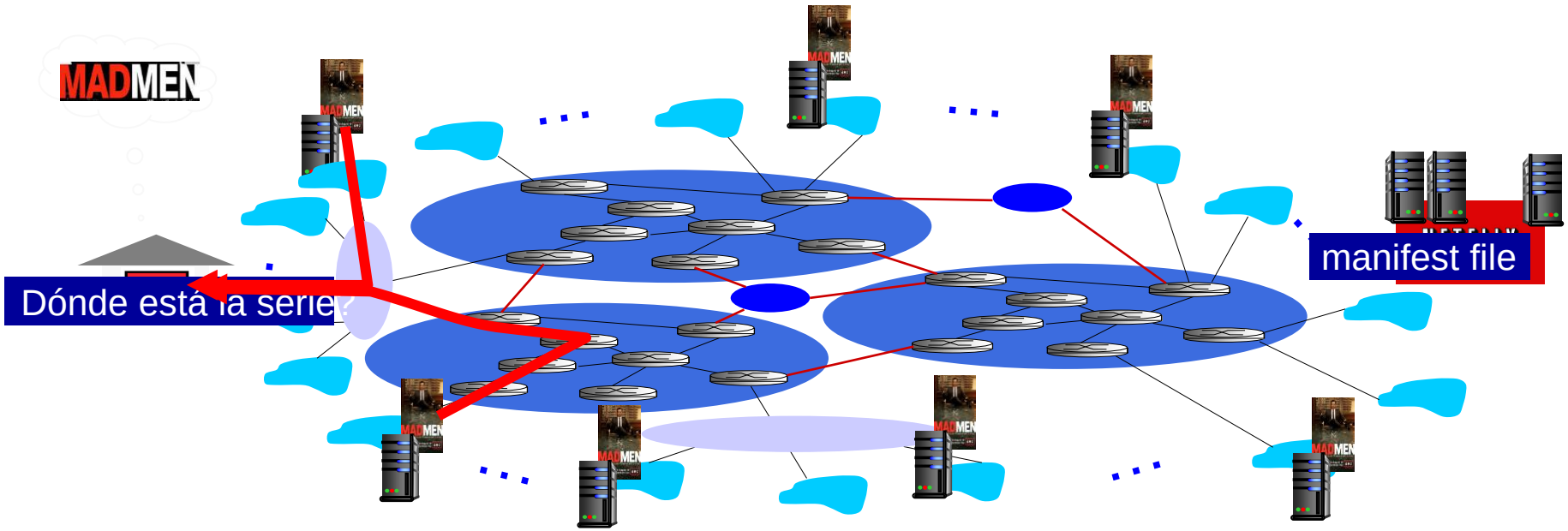
Reto: ¿Cómo transmitir contenido (seleccionado entre millones de videos) a cientos de miles de usuarios simultáneos?

- *Opción 2:* almacenar/despachar múltiples copias de videos en múltiples sitios distribuidos geográficamente *(CDN)*
  - *enter deep:* Empuje los servidores CDN profundamente en muchas redes de acceso
    - Cerca a usuarios
    - Akamai: 240,000 servidores en > 120 países (2015)

  - *bring home:* números más pequeños (decenas) de grupos más grandes en COP cerca de redes de acceso
  - Ejemplo Limelight

Content distribution networks (CDNs)

- CDN: almacena copias de contenido en los nodos CDN
- el suscriptor solicita contenido, el proveedor de servicios devuelve el manifiesto
  - al usar el manifiesto, el cliente recupera el contenido a la tasa más alta admitida
    - puede elegir una velocidad diferente o copiar si la ruta de la red está congestionada



Content distribution networks (CDNs)

Comunicación de Internet host-host como servicio

OTT (retos): hacer frente a una Internet congestionada desde el “la frontera“

¿Qué contenido colocar en qué nodo CDN?

¿De qué nodo CDN recuperar contenido? ¿A qué velocidad?

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

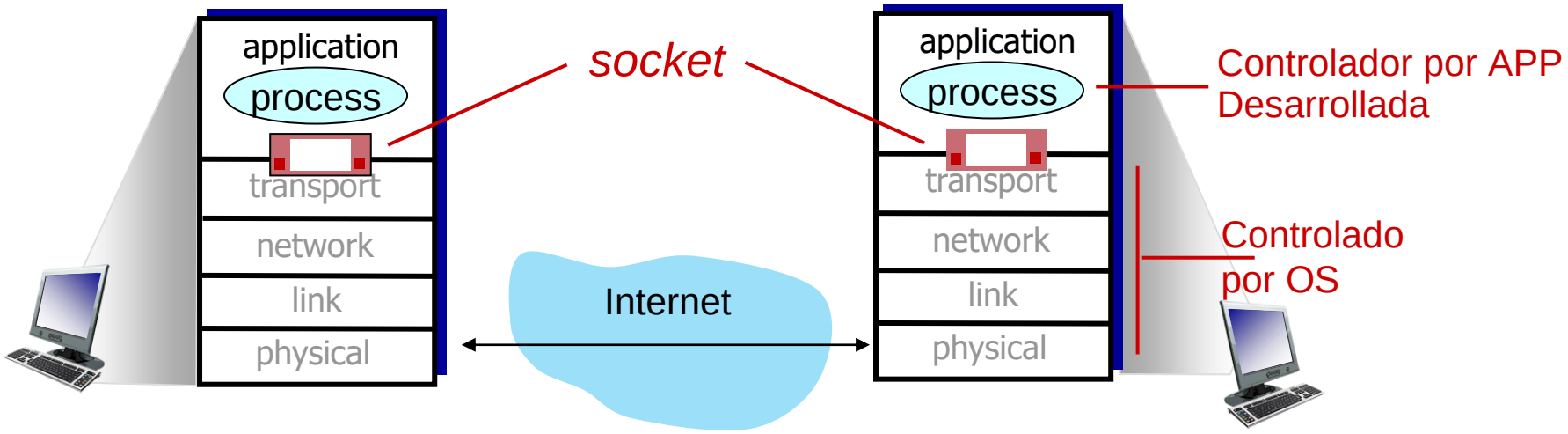
Programación de socket con UDP y TCP (Taller con Python)

Taller Packet Tracer – Network Services

Programación de sockets

objetivo: *aprender a crear aplicaciones cliente / servidor que se comuniquen mediante sockets*

socket: *puerta entre el proceso de aplicación y el protocolo de transporte de extremo a extremo*



Programación de sockets

Dos tipos de enchufes para dos servicios de transporte:

- *UDP:* datagrama no confiable
- *TCP:* confiable, basado en tasa de envío de bytes

Ejemplo básico de Taller 2:

1. el cliente lee una línea de caracteres (datos) de su teclado y envía datos al servidor
2. el servidor recibe los datos y convierte los caracteres a mayúsculas
3. el servidor envía datos modificados al cliente
4. el cliente recibe datos modificados y muestra una línea en su pantalla

Programación de sockets (UDP)

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

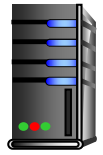
- sin “hand-shaking” antes de enviar datos
- el remitente adjunta explícitamente la dirección IP de destino y el número de puerto a cada paquete
- el receptor extrae la dirección IP del remitente y el número de puerto del paquete recibido

UDP: los datos transmitidos pueden perderse o recibirse fuera de servicio

Nota:

- UDP proporciona una transferencia no confiable de grupos de bytes ("datagramas") entre los procesos del cliente y del servidor

Client/server socket: interacción UDP



server (running on serverIP)

create socket, port= x:
serverSocket =
socket(AF_INET,SOCK_DGRAM)

read datagram from
serverSocket

write reply to
serverSocket
specifying
client address,
port number

client

create socket:
clientSocket =
socket(AF_INET,SOCK_DGRAM)

Create datagram with serverIP address
And port=x; send datagram via
clientSocket

read datagram from
clientSocket

close
clientSocket

Ejemplo: UDP client

Python UDPClient

include Python's socket library →
```
from socket import *
```
```
serverName = 'hostname'
serverPort = 12000
```
create UDP socket for server →
```
clientSocket = socket(AF_INET,
                      SOCK_DGRAM)
```
get user keyboard input →
```
message = raw_input('Input lowercase sentence:')
```
attach server name, port to message; send into socket →
```
clientSocket.sendto(message.encode(),
                    (serverName, serverPort))
```
read reply characters from socket into string →
```
modifiedMessage, serverAddress =
                    clientSocket.recvfrom(2048)
```
print out received string and close socket →
```
print modifiedMessage.decode()
clientSocket.close()
```

Ejemplo app: UDP server

Python UDPServer

```
from socket import *
serverPort = 12000
serverSocket = socket(AF_INET, SOCK_DGRAM)
serverSocket.bind(('', serverPort))
print ("The server is ready to receive")
while True:
    message, clientAddress = serverSocket.recvfrom(2048)
    modifiedMessage = message.decode().upper()
    serverSocket.sendto(modifiedMessage.encode(),
                        clientAddress)
```

create UDP socket →

bind socket to local port number 12000 →

loop forever →

Read from UDP socket into message, getting client's address (client IP and port) →

send upper case string back to this client →

Programación de sockets (TCP)

El cliente debe contactar al servidor

- El proceso del servidor debe estar ejecutándose primero
- el servidor debe haber creado un socket (puerta) que dé la bienvenida al contacto del cliente

El cliente se pone en contacto con el servidor por:

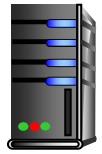
- Creación de un socket TCP, especificando la dirección IP, el número de puerto del proceso del servidor
- cuando el cliente crea el socket: el TCP del cliente establece una conexión con el TCP del servidor

- cuando el cliente lo contacta, el TCP del servidor crea un nuevo socket para que el proceso del servidor se comunique con ese cliente en particular
- permite que el servidor hable con varios clientes/números de puerto de origen utilizados para distinguir clientes (Detalles en capa de transporte)

Nota

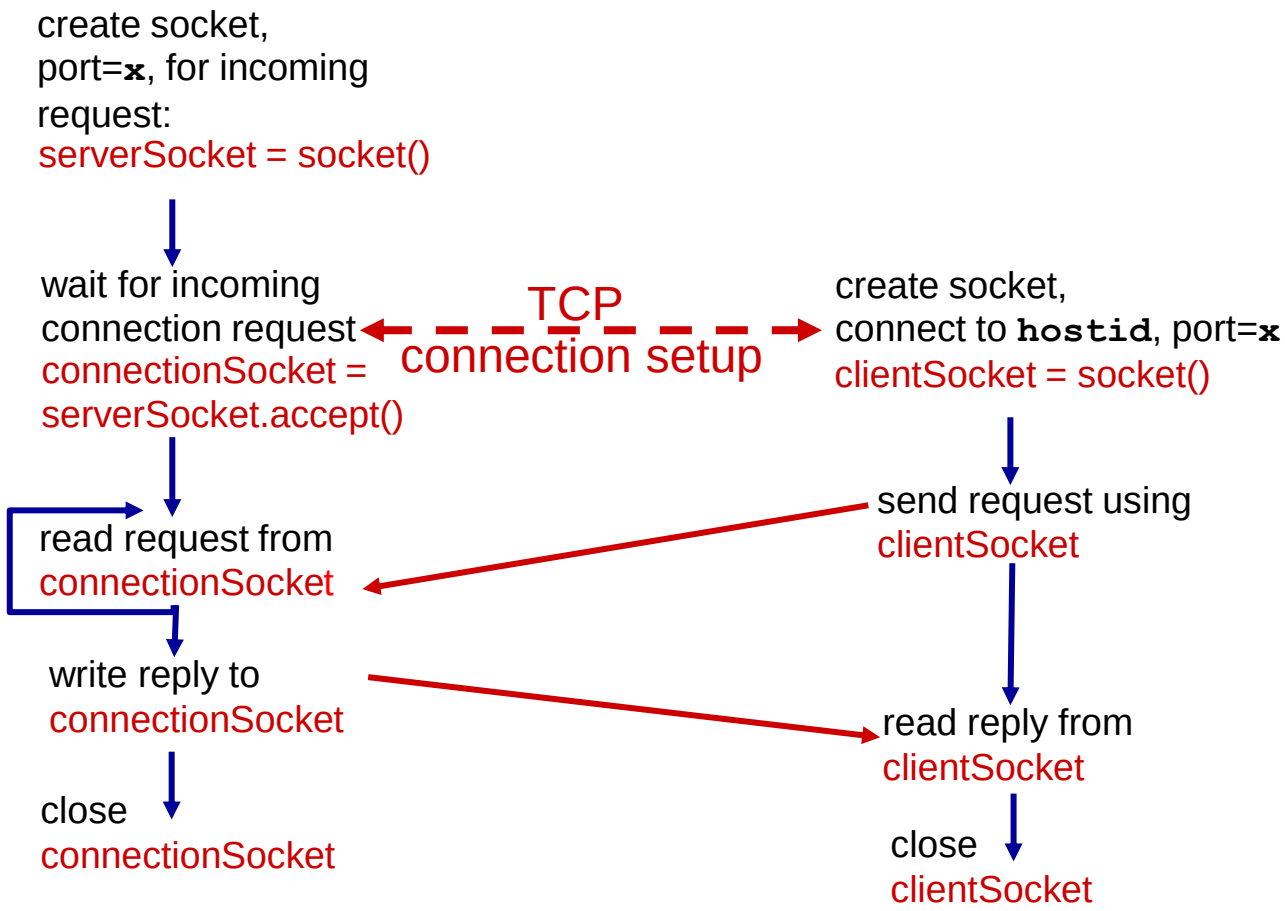
TCP proporciona confiabilidad, en orden de transferencia de flujo de bytes entre los procesos cliente y servidor

Client/server socket: Interacción TCP



server (running on hostid)

client 



Ejemplo: TCP client

Python TCPClient

```
from socket import *
serverName = 'servername'
serverPort = 12000
clientSocket = socket(AF_INET, SOCK_STREAM)
clientSocket.connect((serverName,serverPort))
sentence = raw_input('Input lowercase sentence:')
clientSocket.send(sentence.encode())
modifiedSentence = clientSocket.recv(1024)
print ('From Server:', modifiedSentence.decode())
clientSocket.close()
```

create TCP socket for server, remote port 12000

No need to attach server name, port

Ejemplo: TCP server

Python TCPServer

```
 from socket import *
serverPort = 12000
serverSocket = socket(AF_INET,SOCK_STREAM)
serverSocket.bind(('',serverPort))
serverSocket.listen(1)
print 'The server is ready to receive'
while True:
    connectionSocket, addr = serverSocket.accept()

    sentence = connectionSocket.recv(1024).decode()
    capitalizedSentence = sentence.upper()
    connectionSocket.send(capitalizedSentence.
                                        encode())
    connectionSocket.close()
```

create TCP welcoming socket →

server begins listening for incoming TCP requests →

loop forever →

server waits on accept() for incoming requests, new socket created on return →

read bytes from socket (but not address as in UDP) →

close connection to this client (but *not* welcoming socket) →

HANDS-ON

- Python TCP / UDP