Tecnología Digital IV: Redes de Computadoras

Clase 7: Nivel de Transporte - Parte 1

Lucio Santi & Emmanuel Iarussi

Licenciatura en Tecnología Digital
Universidad Torcuato Di Tella

1 de abril de 2025

Agenda

Servicios del nivel de transporte
Multiplexación y demultiplexación
Transporte no orientado a conexión: UDP
Transferencia de datos confiable
Transporte orientado a conexión: TCP
Protocolos modernos: QUIC
Control de congestión

Objetivos

Entender los principios detrás de los servicios del nivel de transporte:

- Multiplexación y demultiplexación
- Transferencia de datos confiable
- Control de flujo
- Control de congestión

Estudiar los protocolos de transporte de Internet:

- UDP: transporte no confiable y no orientado a conexión
- TCP: transporte confiable orientado a conexión
- Control de congestión en TCP

Servicios del
Nivel de Transporte

Nivel de transporte

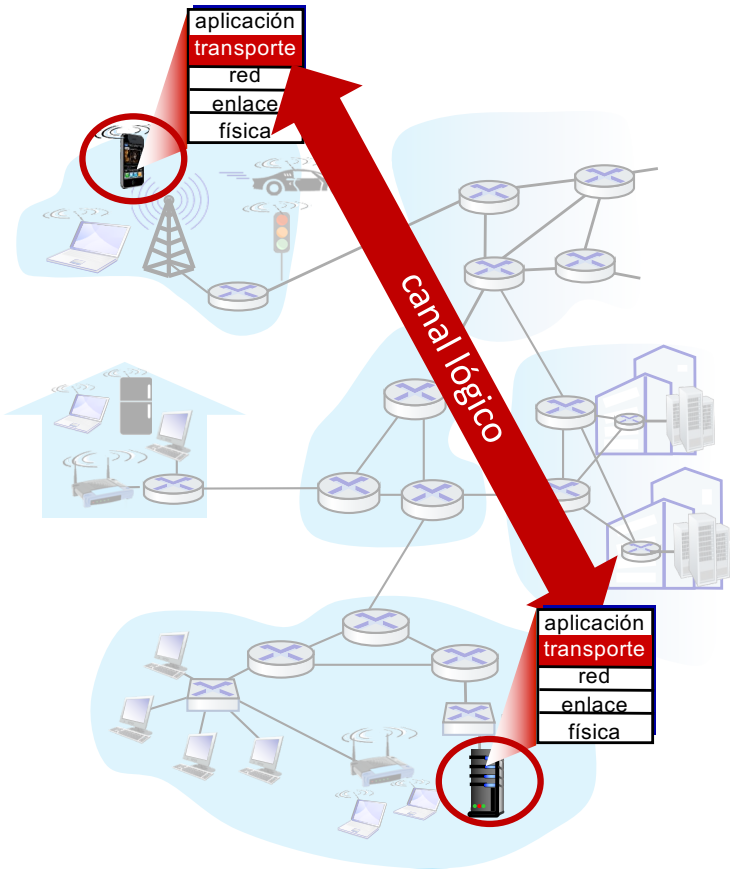
Ofrece un canal de comunicación lógico entre procesos corriendo en diferentes hosts

Acciones de los protocolos en los hosts:

- emisor: divide mensaje de aplicación en **segmentos**; envía los mismos a la capa de red
- receptor: ensambla mensajes a partir de los segmentos; pasa los mismos a la capa de aplicación

Dos protocolos de transporte disponibles para aplicaciones en Internet:

- **TCP** y **UDP**



Contraste con el nivel de red

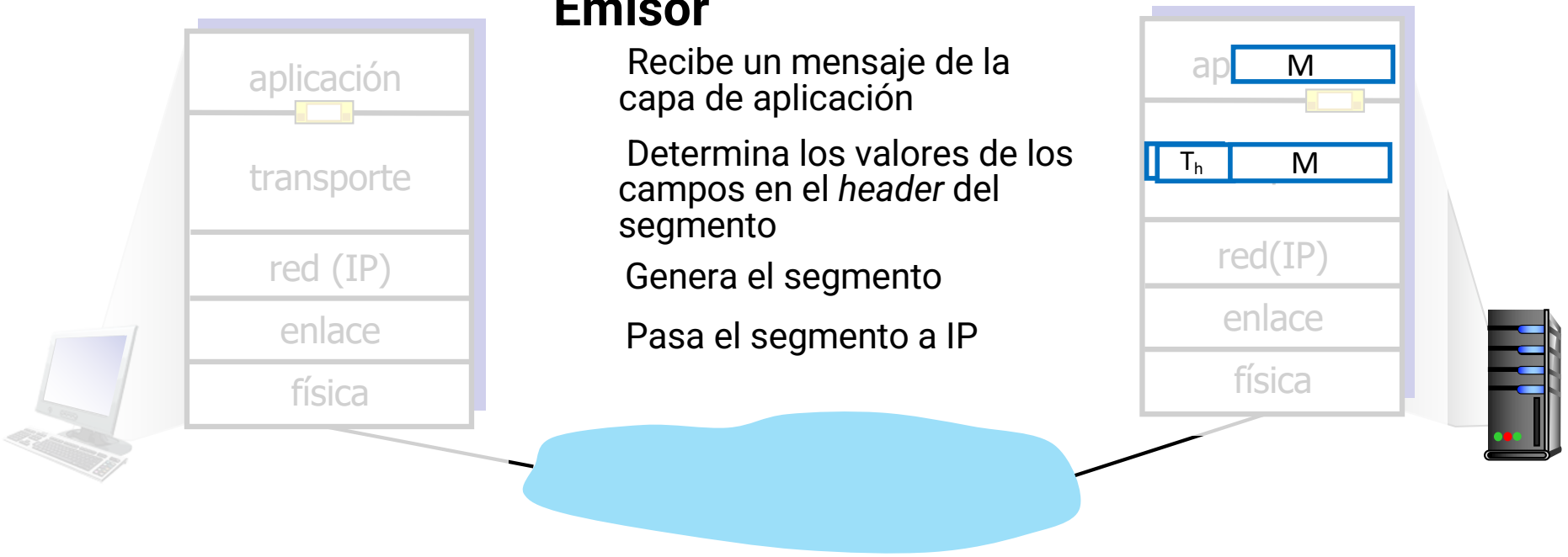
Nivel de transporte

- Canal de comunicación lógico entre procesos
- Se apoya en (y fortalece) los servicios de la capa de red

Nivel de red

- Canal de comunicación lógico entre hosts

Acciones del nivel de transporte



Emisor

- Recibe un mensaje de la capa de aplicación
- Determina los valores de los campos en el *header* del segmento
- Genera el segmento
- Pasa el segmento a IP

Acciones del nivel de transporte

Receptor

Recibe segmento de IP

Verifica el *header*

Extrae el mensaje del protocolo de aplicación

Demultiplexa el mensaje hacia la app a través de un socket

Protocolos de transporte de Internet

TCP: *Transmission Control Protocol*

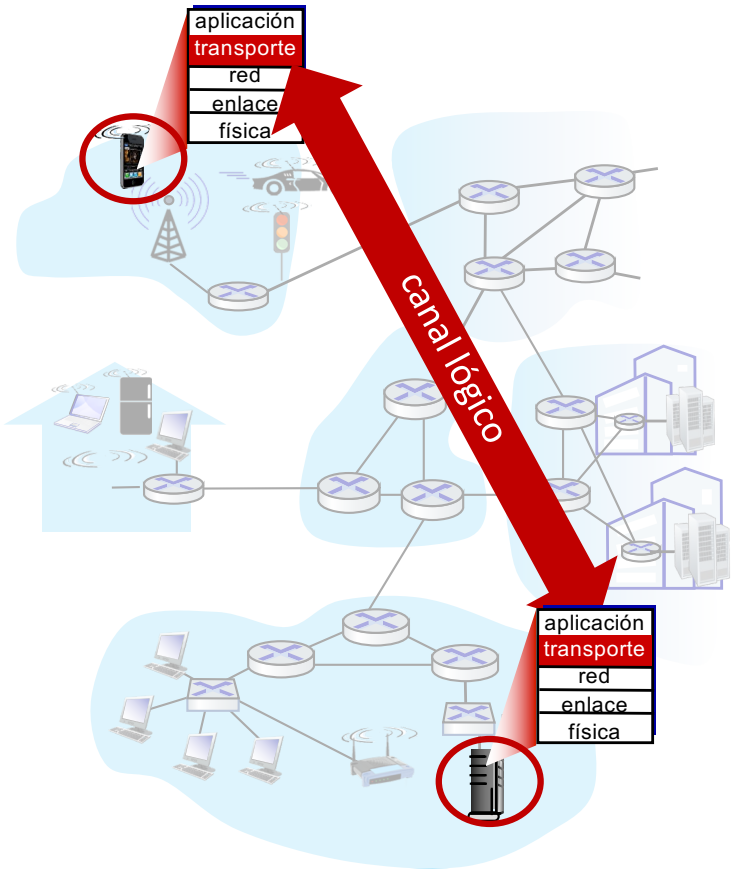
- Entrega confiable y en orden
- Control de congestión
- Control de flujo
- Establecimiento de conexión

UDP: *User Datagram Protocol*

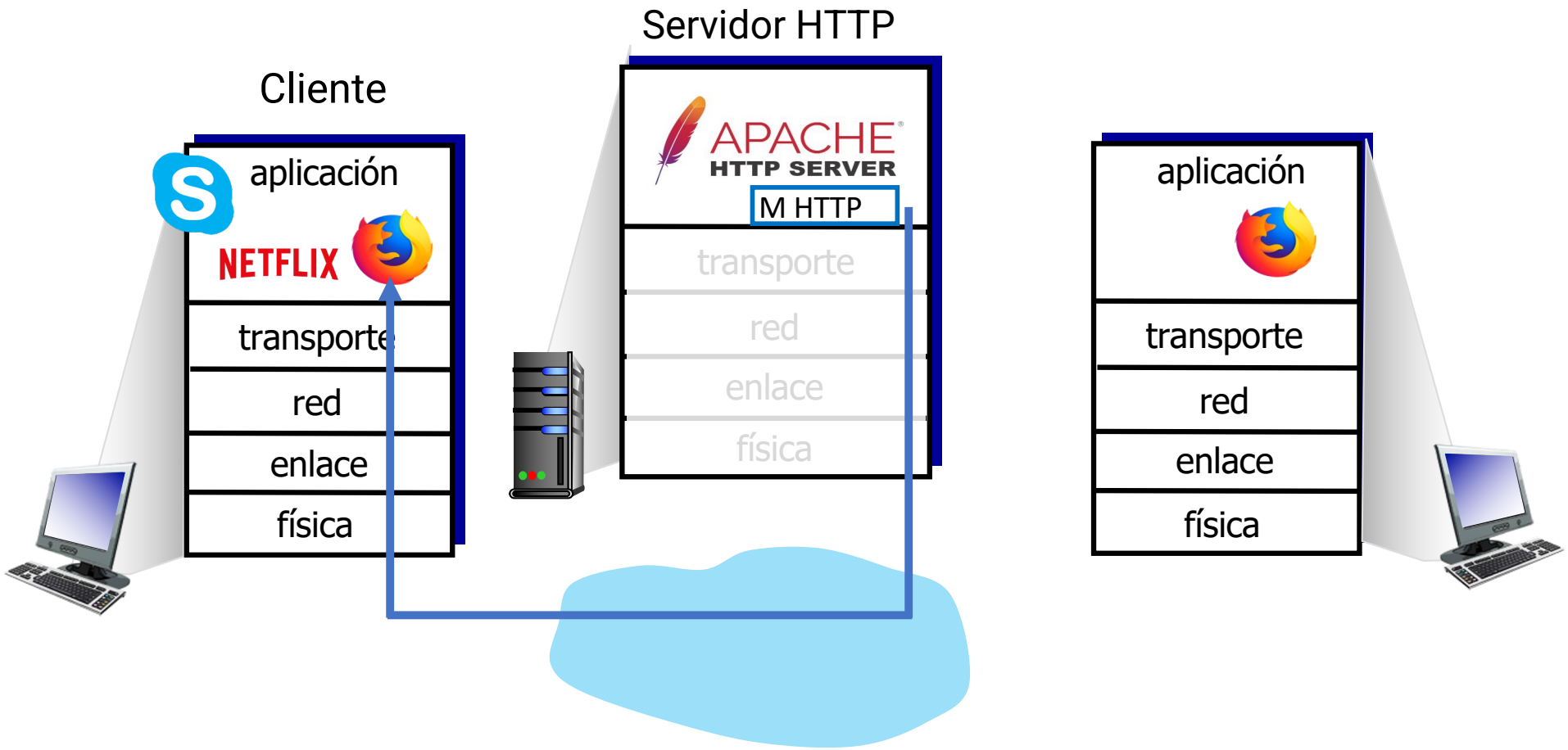
- Entrega no confiable y fuera de orden
- "*Best-effort*" (extensión de IP)

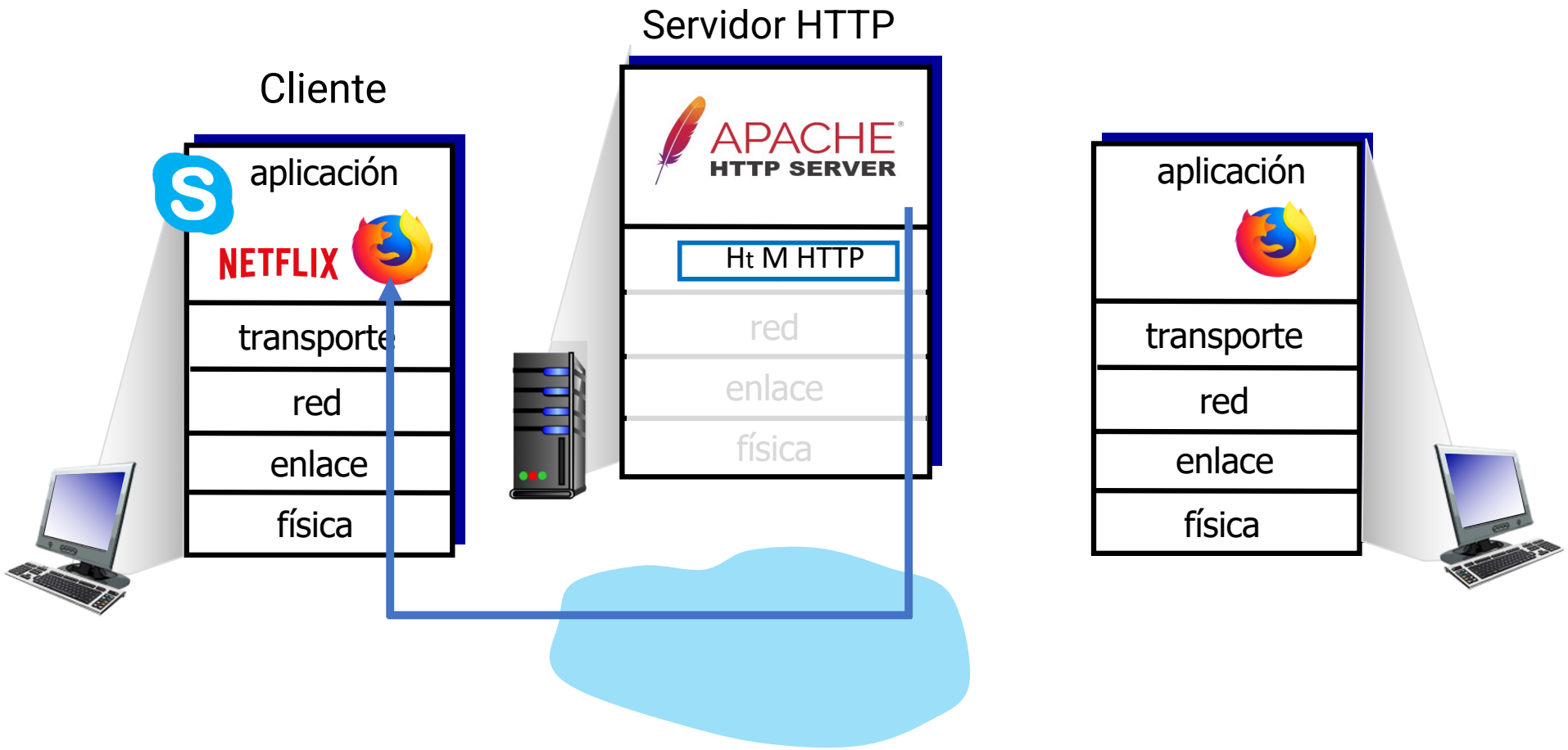
Servicios no disponibles:

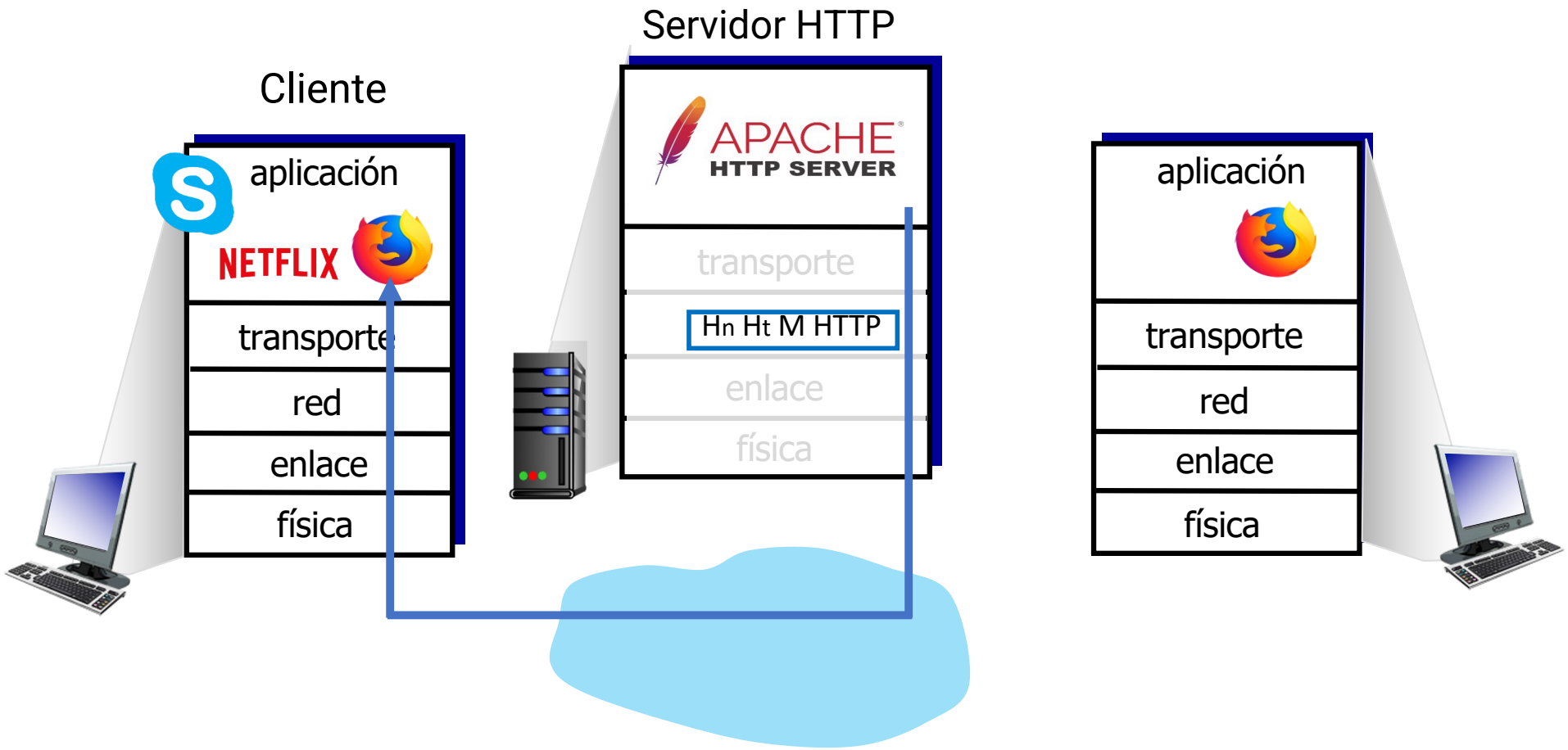
- Garantías de delay
- Garantías de ancho de banda

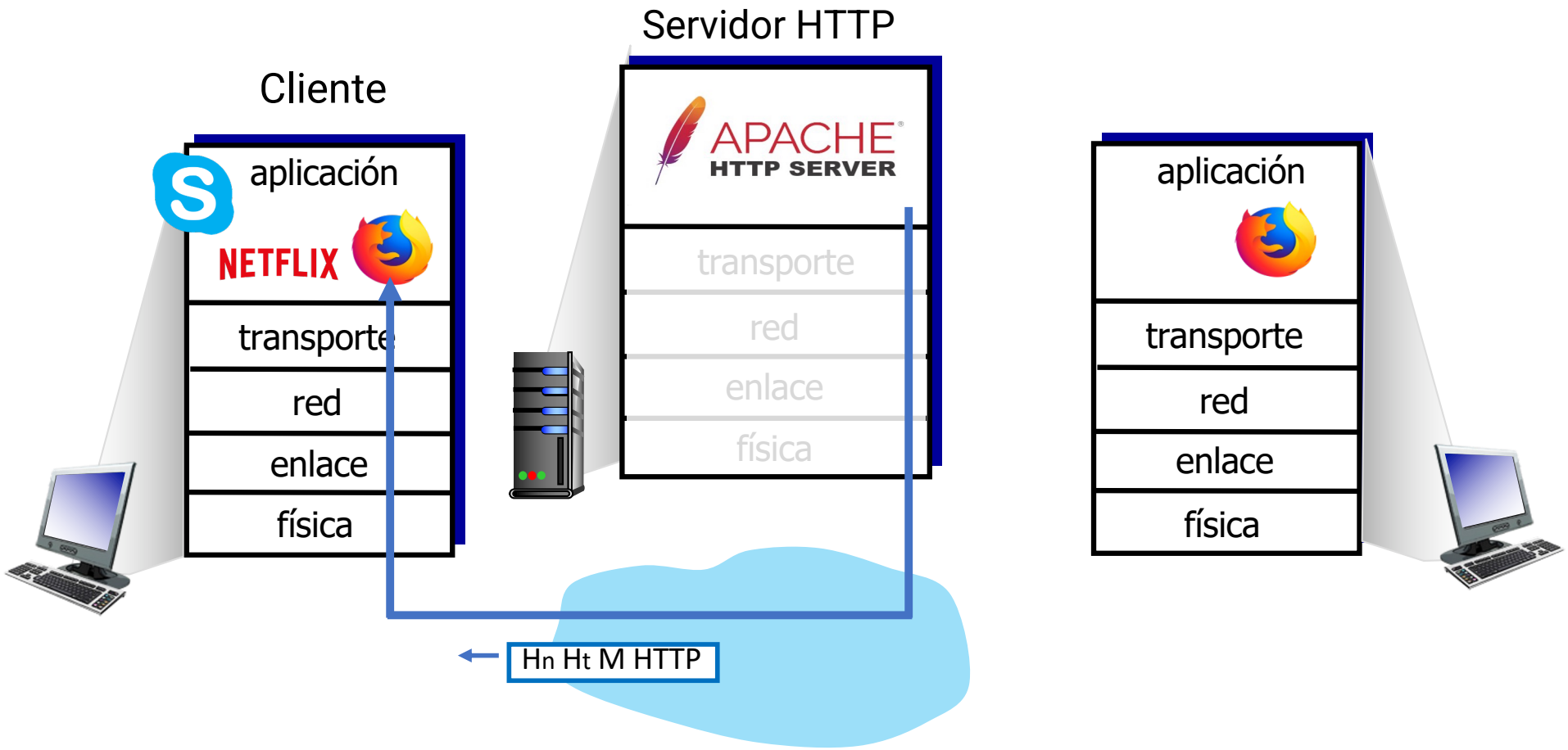


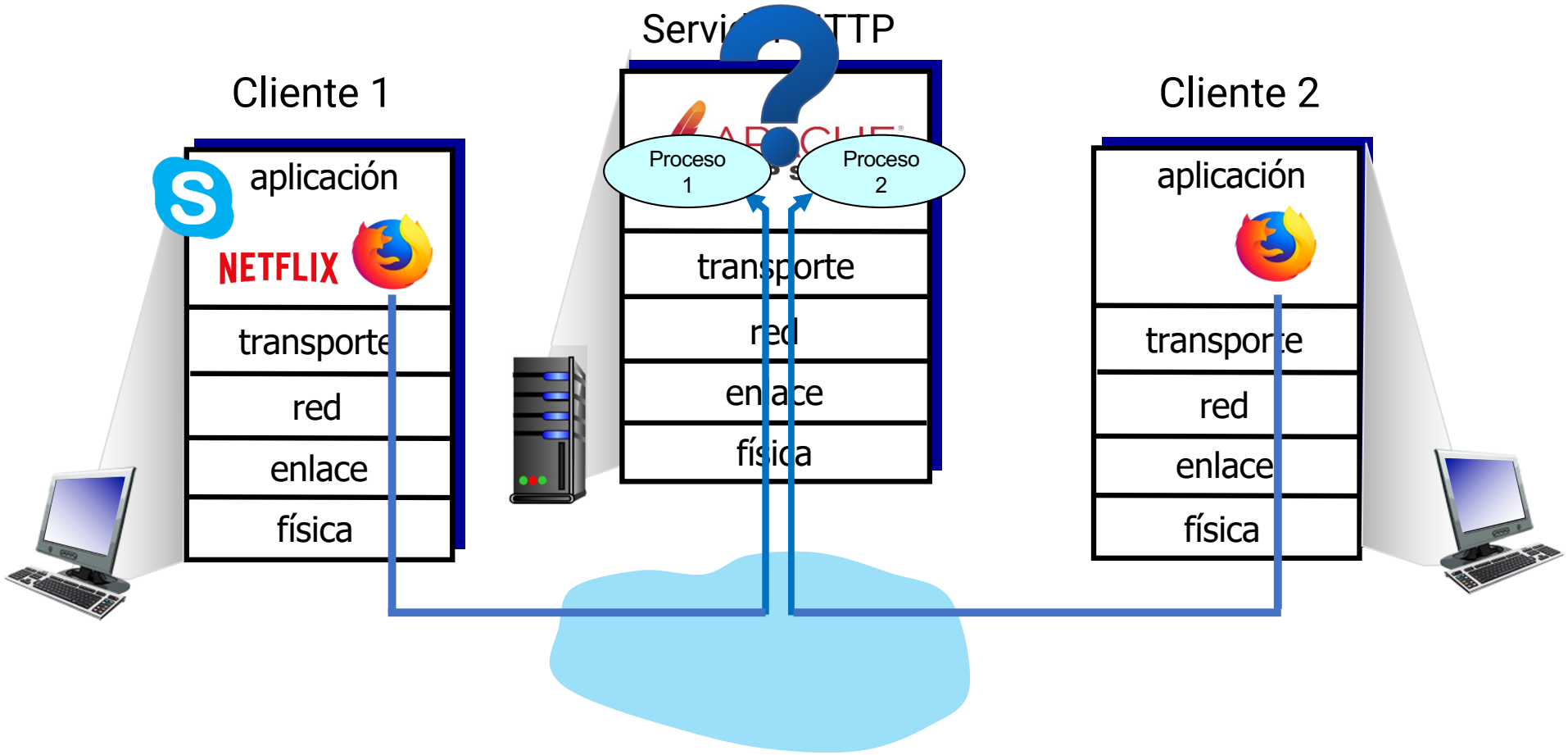
Multiplexación y demultiplexación


Cliente
aplicación
NETFLIX
transporte
red
enlace
física
Servidor HTTP
APACHE
HTTP SERVER
M HTTP
transporte
red
enlace
física
aplicación
transporte
red
enlace
física


Cliente
Servidor HTTP
aplicación
NETFLIX
transporte
red
enlace
física
APACHE
HTTP SERVER
Ht M HTTP
red
enlace
física
aplicación
transporte
red
enlace
física


Cliente
Servidor HTTP
aplicación
NETFLIX
transporte
red
enlace
física
APACHE
HTTP SERVER
transporte
Hn Ht M HTTP
enlace
física
aplicación
transporte
red
enlace
física


Cliente
Servidor HTTP
aplicación
NETFLIX
transporte
red
enlace
física
APACHE
HTTP SERVER
transporte
red
enlace
física
aplicación
transporte
red
enlace
física
Hn Ht M HTTP


Cliente 1
aplicación
NETFLIX
transporte
red
enlace
física
Servi TTP
Proceso 1
Proceso 2
transporte
red
enlace
física
Cliente 2
aplicación
transporte
red
enlace
física

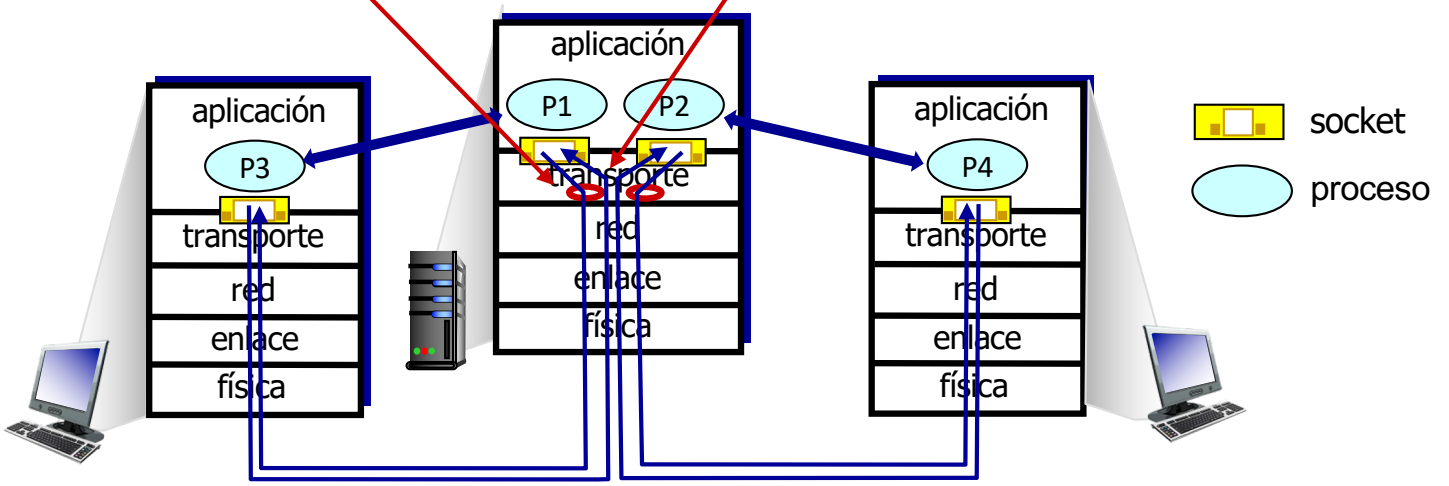
Multiplexación/demultiplexación

multiplexación (emisor)

datos de distintos sockets; se agrega *header* de transporte (utilizado para demultiplexar luego)

demultiplexación (receptor)

Entrega de segmentos recibidos al socket correcto a partir de los datos del *header*

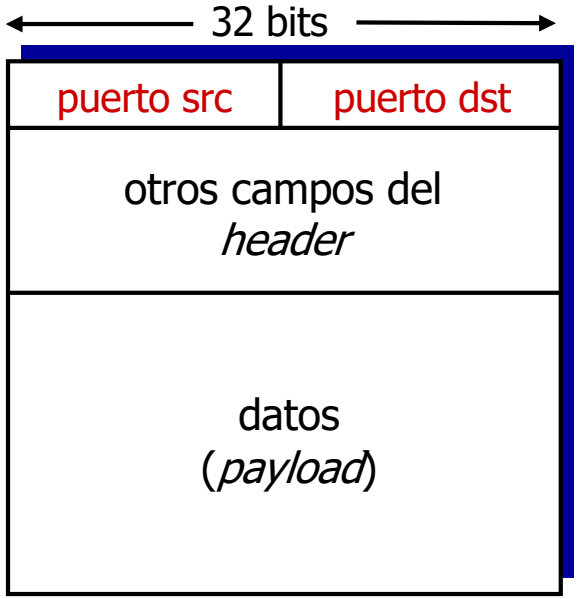


Demultiplexación

El host recibe datagramas IP

- Cada datagrama tiene una dirección IP origen y una dirección IP destino
- Cada datagrama lleva un segmento del nivel de transporte
- Cada segmento tiene un **puerto de origen** y un **puerto de destino**

El host utiliza las direcciones IP y los puertos para dirigir el segmento al socket apropiado



Formato de segmento TCP/UDP

Demultiplexación sin conexión

Al abrir un socket UDP, se debe indicar el puerto local a utilizar:

```
DatagramSocket socket = new DatagramSocket(12534);
```

Al crear un paquete para enviar al socket, se debe indicar:

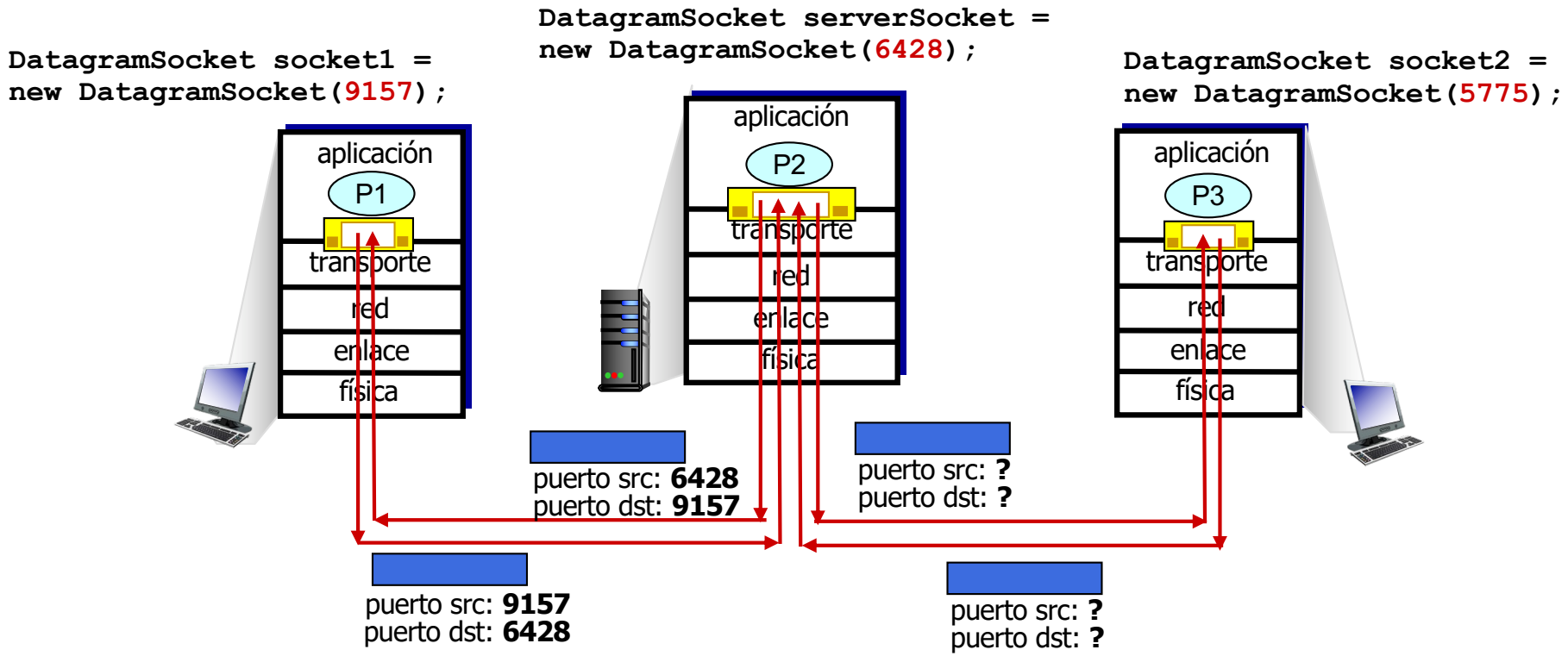
- La dirección IP destino
- El puerto destino

Cuando un host recibe un segmento UDP:

- Verifica el puerto destino en el segmento
- Redirige el segmento al socket con dicho puerto

Los datagramas IP/UDP con el **mismo puerto destino** (pero distinta dirección IP y/o puerto origen) son redirigidos al **mismo socket** en el host receptor

Demultiplexación sin conexión



Demultiplexación con conexión

Un socket TCP se identifica por la **4**-upla

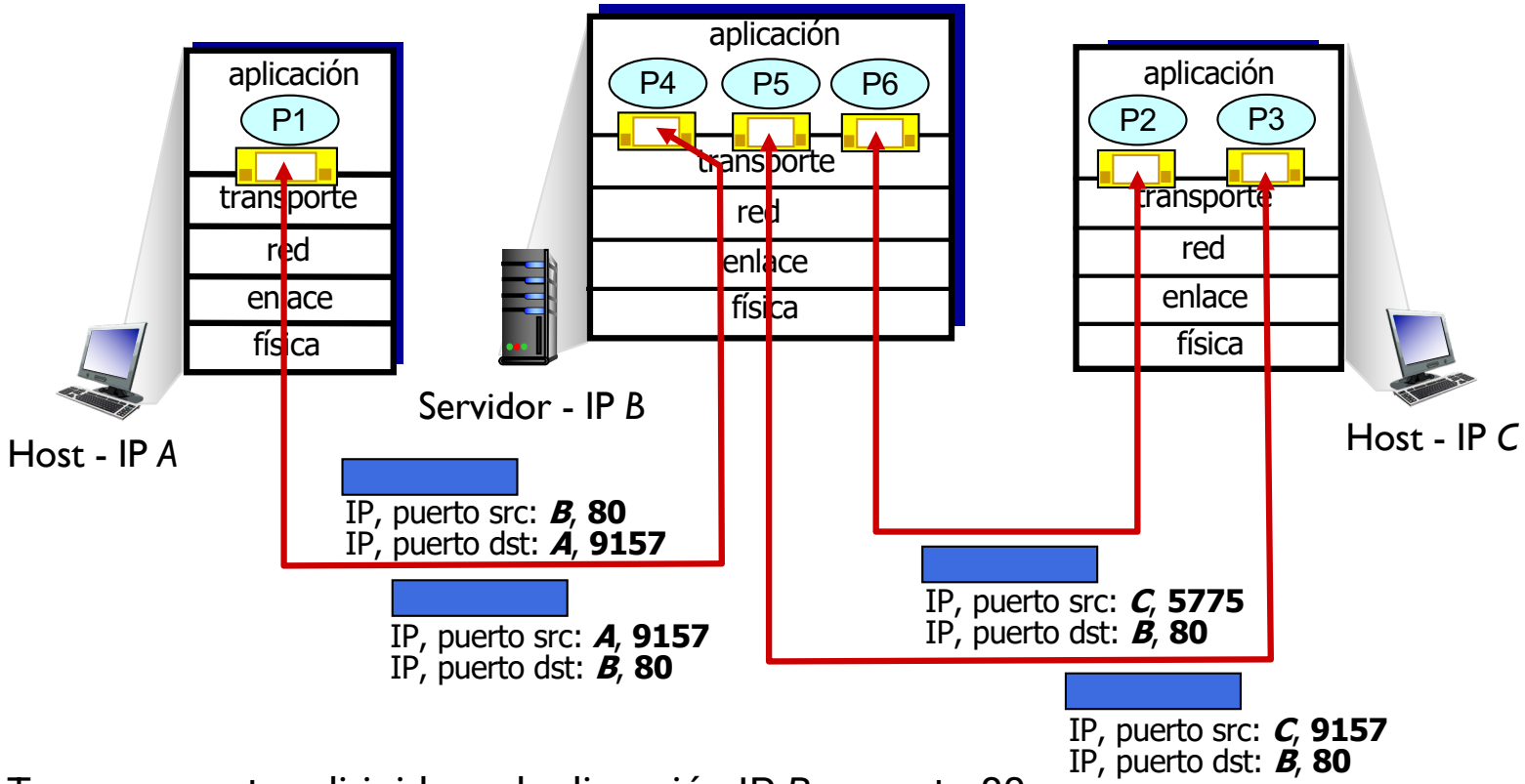
- Dirección IP origen
- Puerto origen
- Dirección IP destino
- Puerto destino

El receptor emplea los cuatro valores para redirigir (demultiplexar) el segmento al socket correcto

Un servidor puede manejar simultáneamente muchos sockets TCP:

- Cada socket se identifica por su 4-upla
- Cada socket está asociado a un cliente distinto

Demultiplexación con conexión



Tres segmentos dirigidos a la dirección IP *B* y puerto 80: demultiplexados hacia **sockets distintos**

Resumen

La multiplexación/demultiplexación se basa en los valores de los *headers* del datagrama y el segmento

UDP: demultiplexación utilizando sólo el puerto destino

TCP: demultiplexación utilizando la 4-upla conformada por las direcciones IP y los puertos origen y destino

El protocolo UDP

UDP: *User Datagram Protocol*

Protocolo de transporte de Internet **básico**

Servicio ***best effort***: los segmentos pueden

- Extraviarse en la red
- Entregarse fuera de orden

No orientado a conexión

- Sin *handshaking* entre emisor y receptor (*connectionless*)
- Cada segmento es procesado independientemente de los otros

¿Por qué existe UDP?

Sin establecimiento de conexión (agrega delay por RTT)

Sencillo: sin estado tanto en emisor como en receptor

Header más chico

Sin control de congestión

UDP puede ir tan “rápido” como quiera

Puede funcionar ante la presencia de congestión

UDP: *User Datagram Protocol*

Usos de **UDP**:

- Aplicaciones de streaming (tolerantes a pérdidas y sensibles a la tasa de transferencia)
- DNS
- SNMP (protocolo para monitorear y administrar dispositivos de red)
- HTTP/3 (más detalle luego)

En caso de necesitar transporte confiable sobre UDP (como HTTP/3):

- Agregar confiabilidad necesaria en la capa de aplicación
- Ídem control de congestión

UDP: *User Datagram Protocol* [RFC 768]

INTERNET STANDARD

RFC 768

J. Postel
ISI
28 August 1980

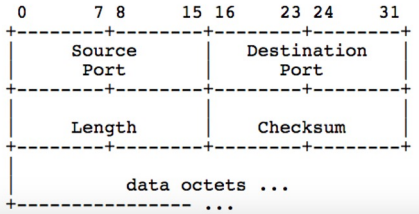
User Datagram Protocol

Introduction

This User Datagram Protocol (UDP) is defined to make available a datagram mode of packet-switched computer communication in the environment of an interconnected set of computer networks. This protocol assumes that the Internet Protocol (IP) [1] is used as the underlying protocol.

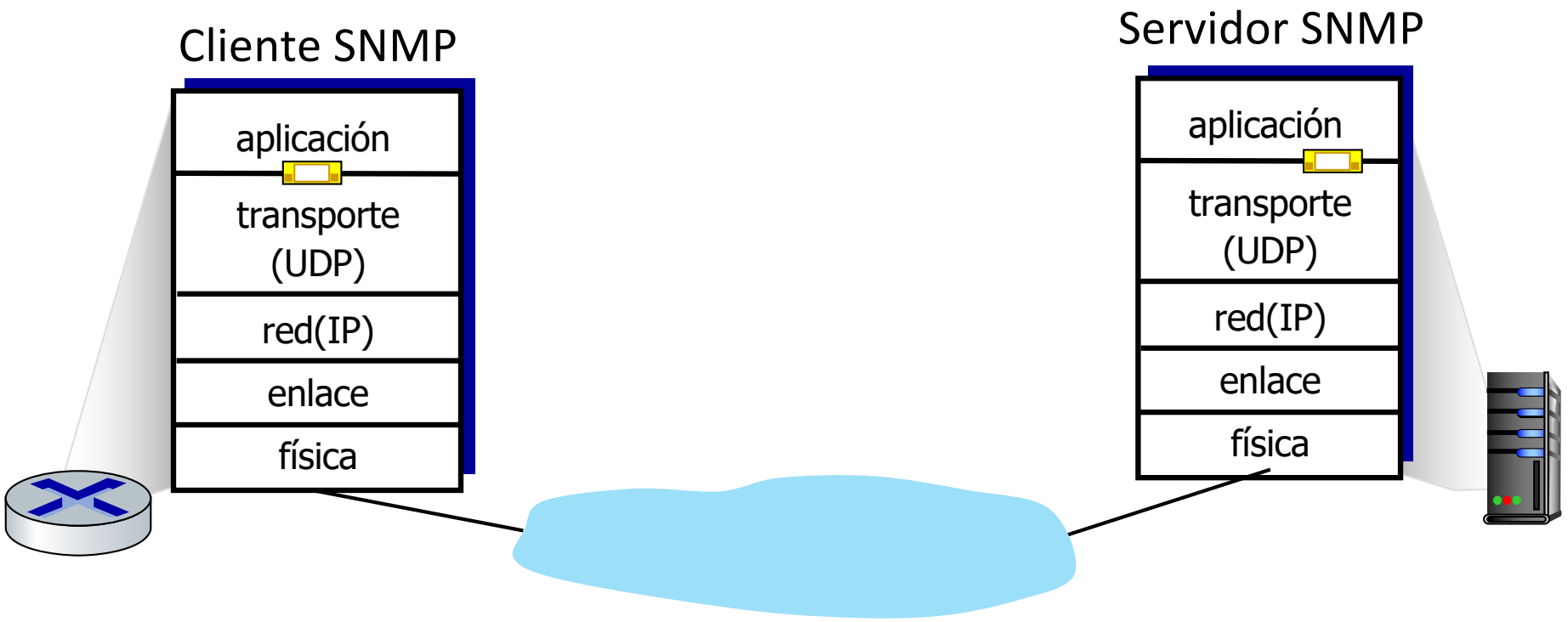
This protocol provides a procedure for application programs to send messages to other programs with a minimum of protocol mechanism. The protocol is transaction oriented, and delivery and duplicate protection are not guaranteed. Applications requiring ordered reliable delivery of streams of data should use the Transmission Control Protocol (TCP) [2].

Format

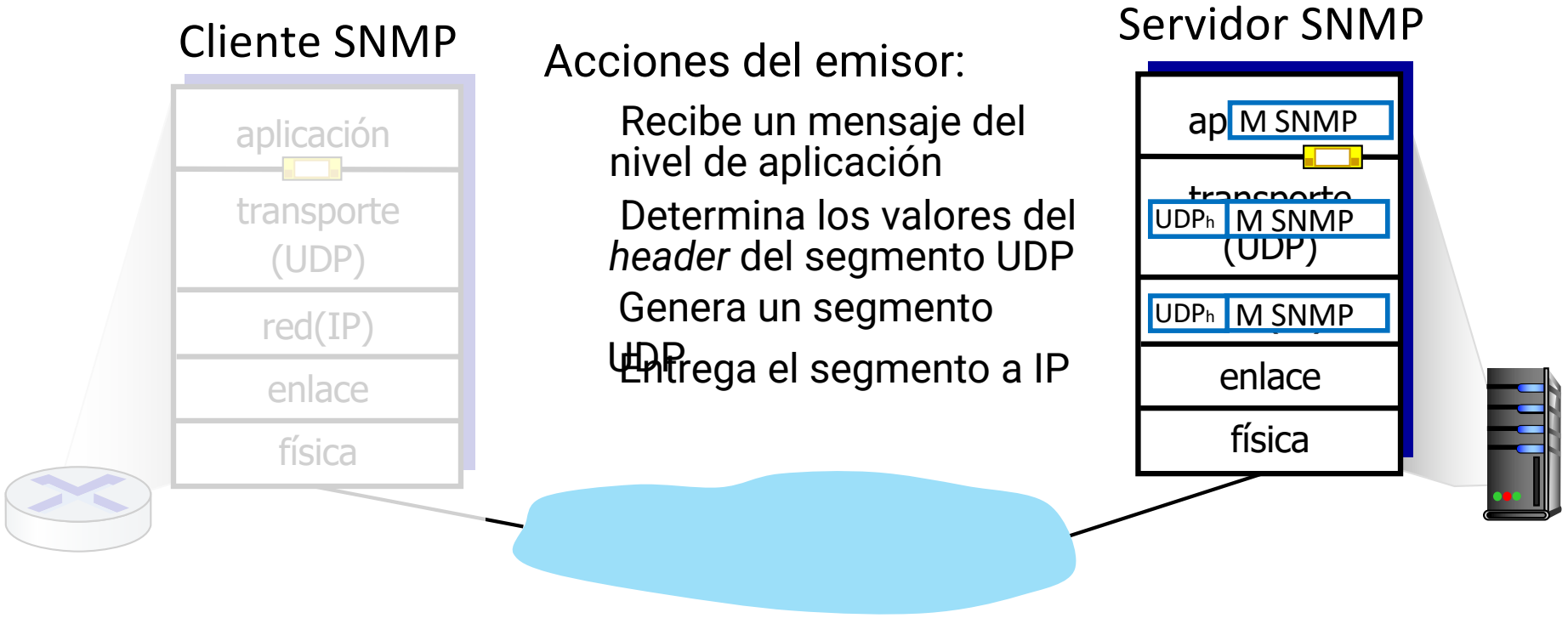


https://www.rfc-es.org/rfc/rfc0768-es.txt

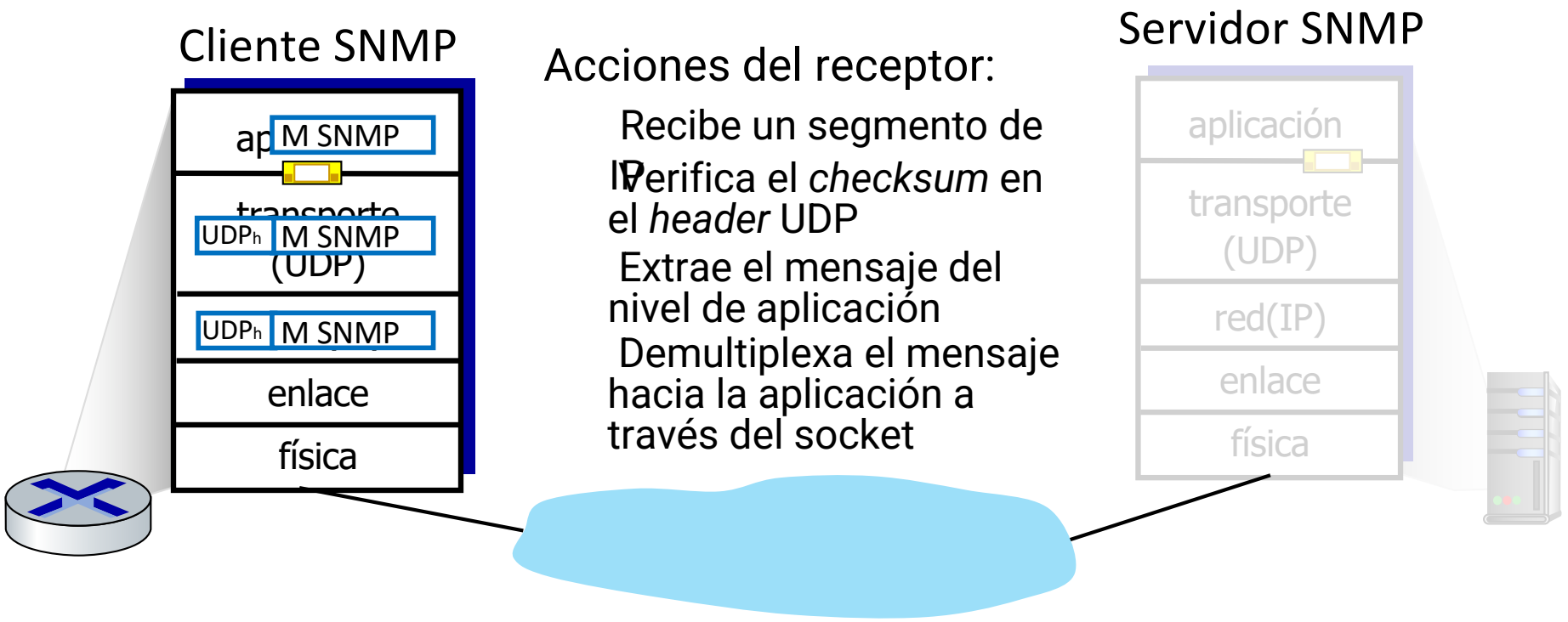
UDP: acciones del nivel de transporte



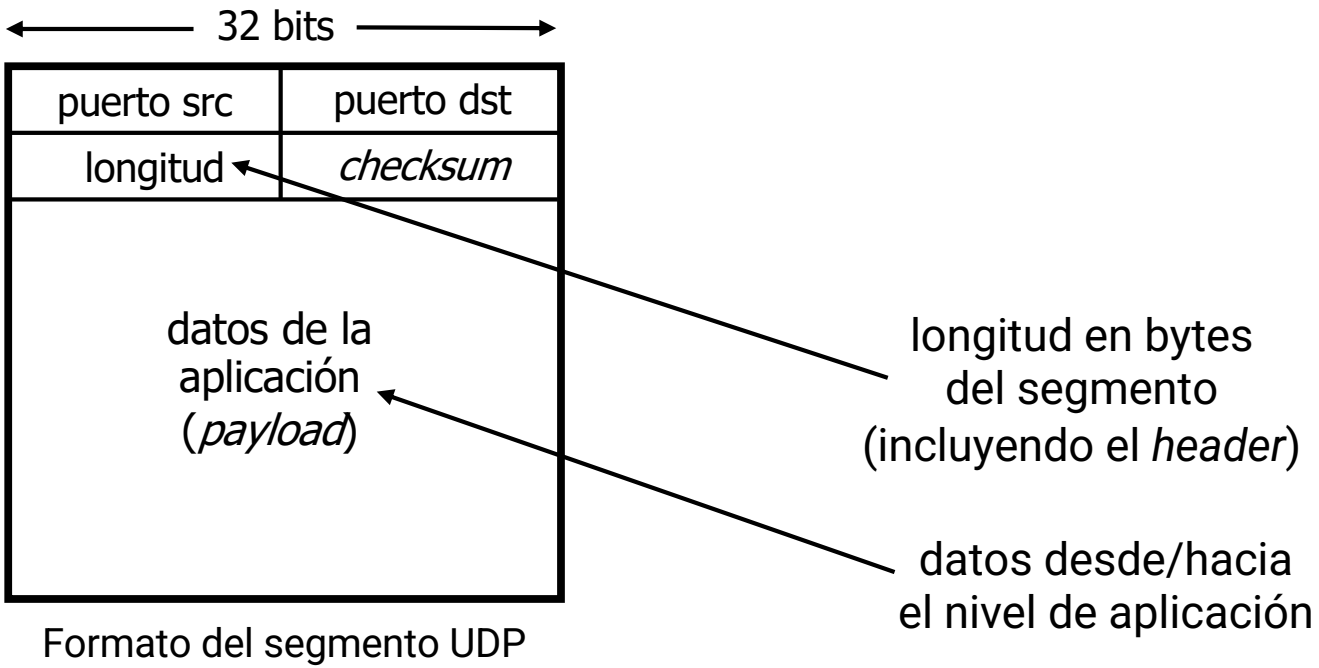
UDP: acciones del nivel de transporte



UDP: acciones del nivel de transporte



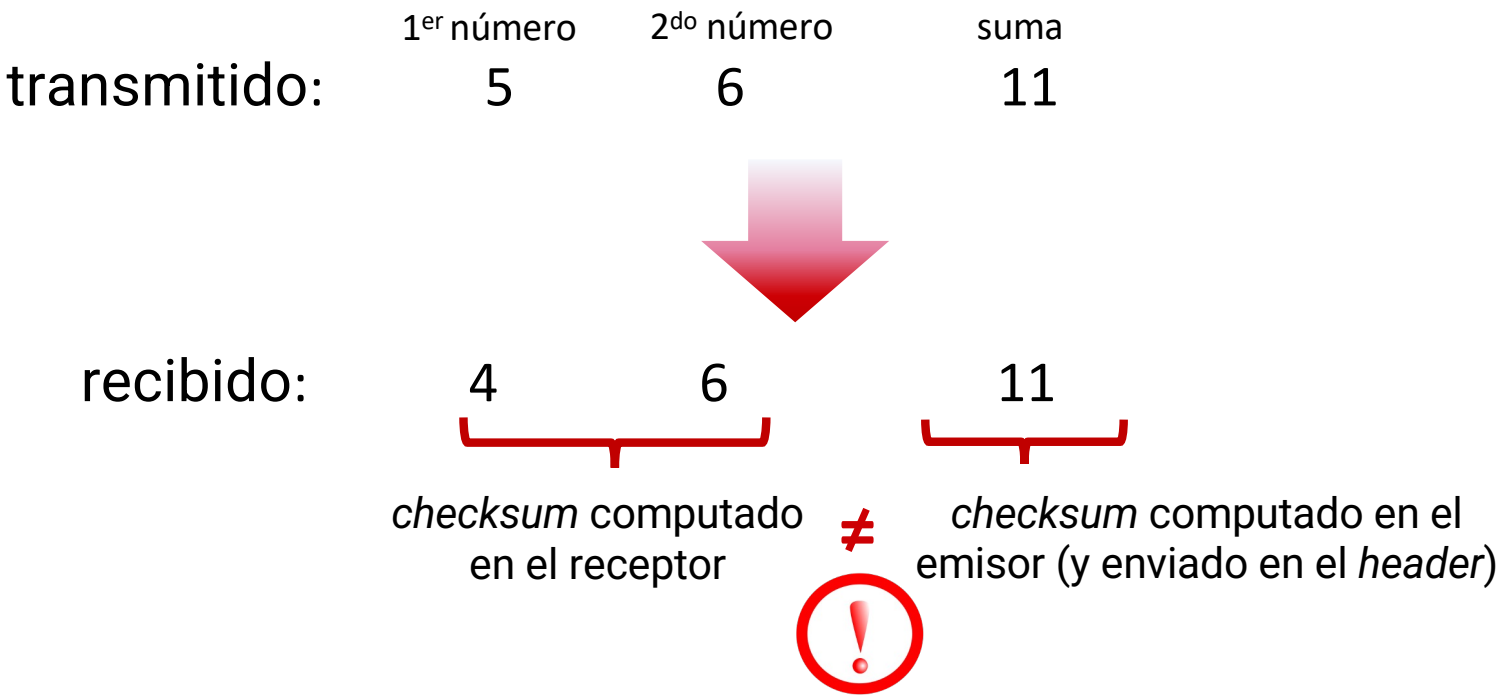
Header de UDP



Formato del segmento UDP

Checksum de UDP

Objetivo: **detectar errores** en los bits del segmento transmitido



Checksum de UDP

Objetivo: **detectar errores** en los bits del segmento transmitido

Emisor

Interpreta el contenido del segmento como una secuencia de enteros de 16 bits (incluyendo ***header* + direcciones IP**)

checksum: complemento a uno de la suma del contenido del segmento

El valor del *checksum* se coloca en el *header* del segmento

Receptor

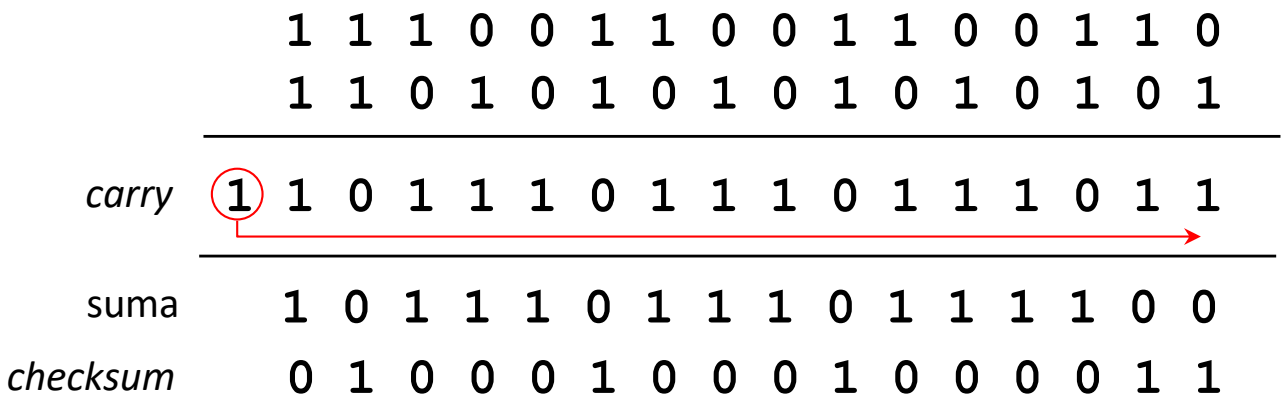
Calcula la suma de las palabras de 16 bits del segmento recibido (incluido el *checksum*)

Comprueba que el complemento a uno de este valor sea 0

- Si no es cero: error detectado!
- Si es cero: no se detectaron errores (**...pero podría haber errores de todos modos?**)

Checksum: ejemplo

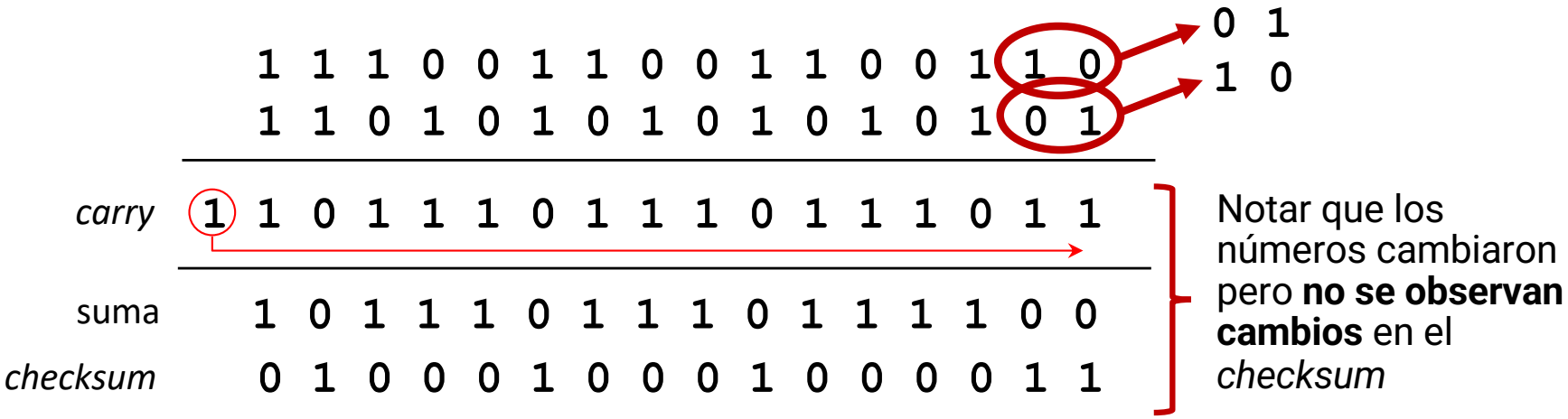
Ejemplo: suma (complemento a uno) de dos enteros de 16 bits



Nota: al sumar números en complemento a uno, el bit de carry debe sumarse al resultado

Protección del checksum

Ejemplo: suma (complemento a uno) de dos enteros de 16 bits



Demo!

Armemos un paquete UDP "de juguete" en Scapy

- *Header* IP nulo y *payload* con los bytes `hola`

Vamos a inspeccionar su contenido, en particular el *checksum*

Luego, cambiamos un campo (e.g. el puerto de origen) y volvemos a inspeccionar el contenido

- ¿Cómo cambió el *checksum*?

Resumen: **UDP**

Protocolo **básico**

- Pueden perderse segmentos y ser entregados fuera de orden
- Servicio *best effort*: enviar y “esperar lo mejor”

UDP tiene sus beneficios:

- No precisa *setup* inicial: evita penalización de RTT
- Puede funcionar cuando el servicio de la red está degradado
- Ayuda con la confiabilidad (*checksum*)

Es posible agregar funcionalidad adicional sobre UDP en la capa de aplicación (e.g., HTTP/3)

Ejercicio!

UDP y TCP usan el complemento 1 para sus *checksums*. Supongamos que tenemos los siguientes tres bytes: 01010011, 01100110, 01110100.

- ¿Cuál es el complemento a 1 de la suma de estos bytes?
- ¿Por qué UDP toma el complemento a 1 de la suma? es decir, ¿por qué no usar simplemente la suma?
- ¿Es posible que un error de 1 bit pase desapercibido? ¿Qué tal un error de 2 bits?

Ejercicio!

UDP y TCP usan el complemento 1 para sus *checksums*. Supongamos que tenemos los siguientes tres bytes: 01010011, 01100110, 01110100.

$$
\begin{array}{r}
0\ 1\ 0\ 1\ 0\ 0\ 1\ 1 \\
+\ 0\ 1\ 1\ 0\ 0\ 1\ 1\ 0 \\
\hline
1\ 0\ 1\ 1\ 1\ 0\ 0\ 1
\end{array}
$$

$$
\begin{array}{r}
1\ 0\ 1\ 1\ 1\ 0\ 0\ 1 \\
+\ 0\ 1\ 1\ 1\ 0\ 1\ 0\ 0 \\
\hline
0\ 0\ 1\ 0\ 1\ 1\ 1\ 0
\end{array}
$$

$$C1 = 1\ 1\ 0\ 1\ 0\ 0\ 0\ 1$$