

Capítulo 3: Capa Transporte - I

ELO322: Redes de Computadores
Agustín J. González

Este material está basado en:

- Material de apoyo al texto *Computer Networking: A Top Down Approach Featuring the Internet*. Jim Kurose, Keith Ross.

Capítulo 3: Capa Transporte

Objetivos:

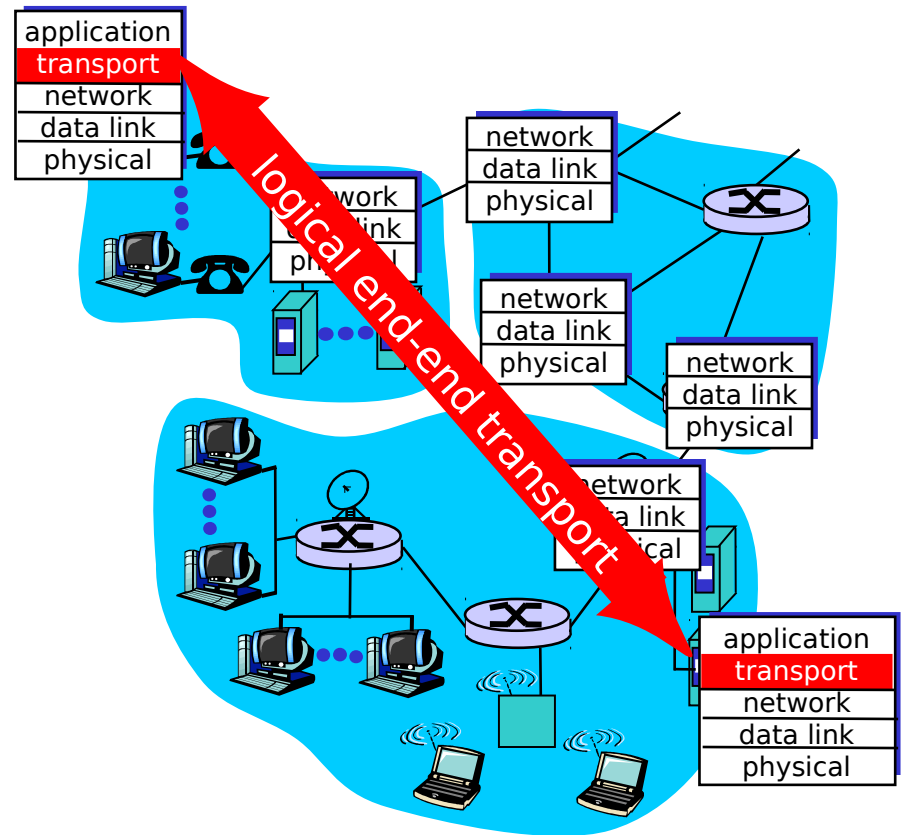
- ❑ Entender los principios detrás de los servicios de la capa transporte:
 - Multiplexing / demultiplexing
 - Transferencia confiable de datos
 - Control de flujo
 - Control de congestión
- ❑ Aprender sobre los protocolos de transporte en la Internet:
 - UDP (**U**ser **D**atagram **P**rotocol): transporte sin conexión
 - TCP (**T**ransmission **C**ontrol **P**rotocol): transporte orientado a la conexión
 - Control de congestión en TCP

Contenido de Capítulo 3

- ❑ 3.1 Servicios de la capa transporte
- ❑ 3.2 Multiplexing y demultiplexing
- ❑ 3.3 Transporte sin conexión: UDP
- ❑ 3.4 Principios de transferencia confiable de datos
- ❑ 3.5 Transporte orientado a la conexión: TCP
 - Estructura de un segmento
 - Transferencia confiable de datos
 - Control de flujo
 - Gestión de la conexión
- ❑ 3.6 Principios del control de congestión
- ❑ 3.7 Control de congestión en TCP

Protocolos y servicios de transporte

- Proveer *comunicación lógica* entre procesos de aplicaciones corriendo en diferentes hosts
- Los protocolos de transporte corren en sistemas terminales (computadores, no equipos internos como routers)
- Hay más de un protocolo de transporte disponible para las aplicaciones
 - Más comunes: TCP y UDP
 - Hoy también SCTP (**S**tream **C**ontrol **T**ransmission **P**rotocol, 2000), DCCP (**D**atagram **C**ongestion **C**ontrol **P**rotocol, 2005)



Capa transporte vs. capa red

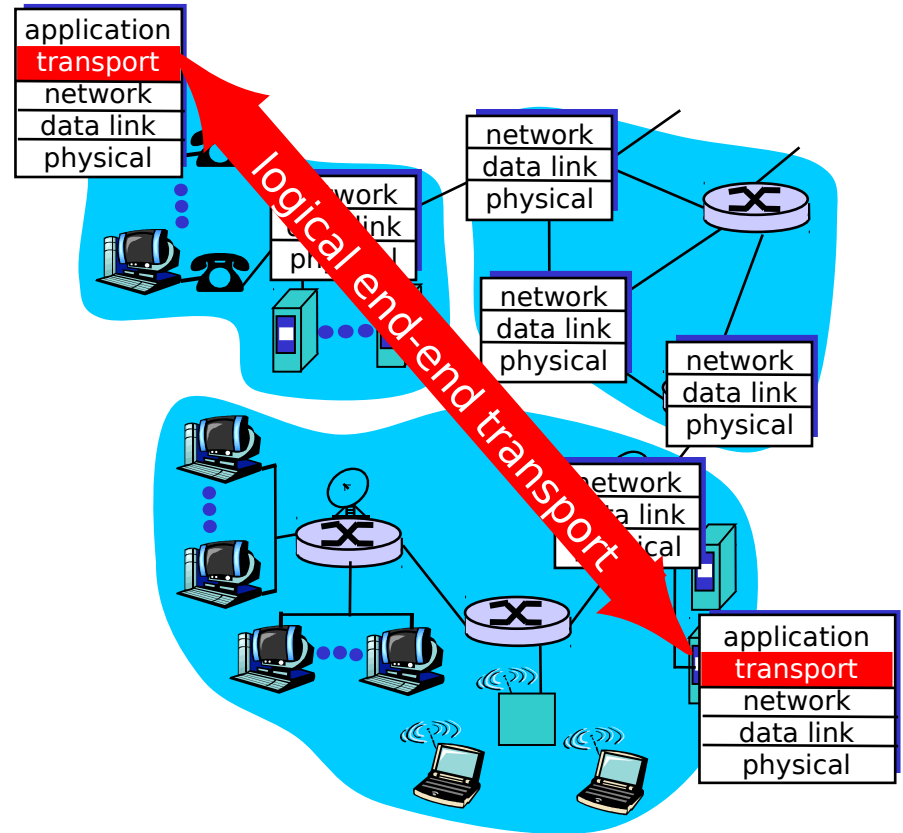
- *Capa transporte:* encargada de la comunicación lógica entre procesos
 - Depende y mejora los servicios de la capa de red
- *Capa de red:* encargada de la comunicación lógica entre hosts

Analogía:

- *12 niños en una casa envían cartas a 12 niños. en otra casa*
- *Ann y Bill recopilan las cartas en cada hogar y las envían por correo. También distribuyen las cartas que llegan.*
- Procesos = niños
- Mensajes aplicación = cartas en sobres
- Hosts = casas
- Protocolo de transporte = Ann y Bill
- Protocolo capa red = servicio de correos

Protocolos de capa transporte en Internet

- TCP: Entrega confiable y en orden
 - Tiene: Control de flujo
 - Tiene: control de congestión
 - Establecimiento de conexión
- UDP: Entrega no confiable, tal vez desordenada:
 - Básicamente el mismo servicio de IP: “mejor esfuerzo” (best-effort)
- Qué servicios **no** se ofrecen:
 - Garantías de retardo
 - Garantías de ancho de banda
 - (Básicamente porque no es fácil -o posible- implementarlo basándose en los servicios de IP)



Contenido de Capítulo 3

- ❑ 3.1 Servicios de la capa transporte
- ❑ 3.2 Multiplexing y demultiplexing
- ❑ 3.3 Transporte sin conexión: UDP
- ❑ 3.4 Principios de transferencia confiable de datos
- ❑ 3.5 Transporte orientado a la conexión: TCP
 - Estructura de un segmento
 - Transferencia confiable de datos
 - Control de flujo
 - Gestión de la conexión
- ❑ 3.6 Principios del control de congestión
- ❑ 3.7 Control de congestión en TCP

Multiplexación/demultiplexación

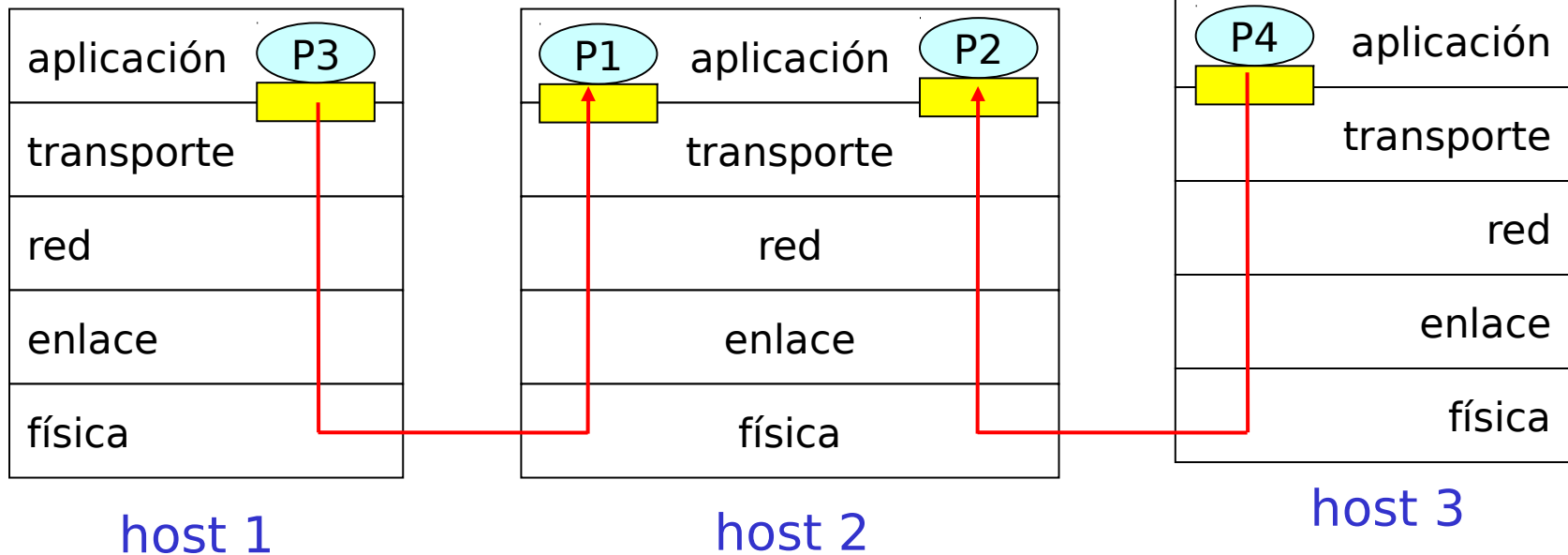
Multiplexación en host Tx:

Capa de transporte:
Recopila datos desde
múltiples sockets, usa info. de
encabezado (puerto e IP
origen, puerto e IP destino)

Demultiplexación en host Rx:

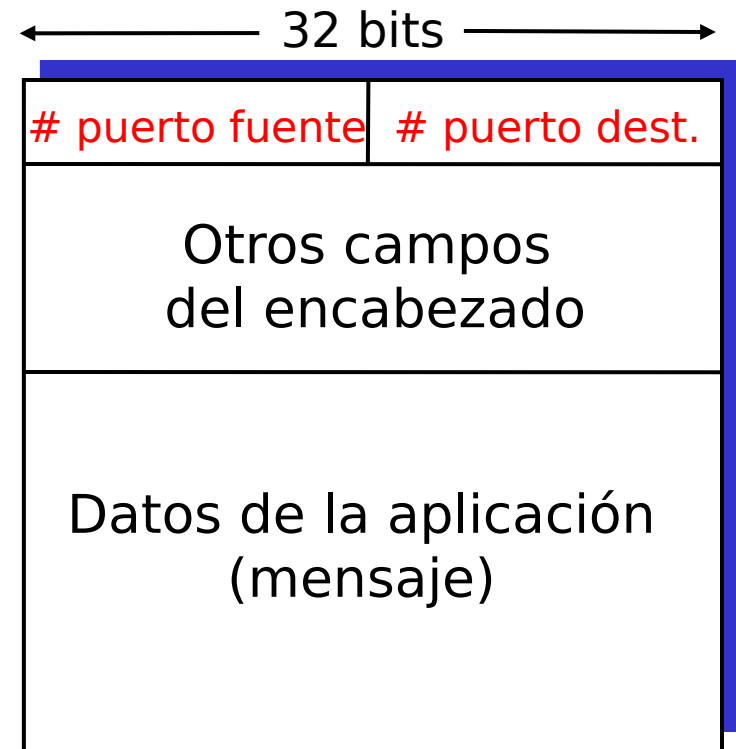
Entrega el segmento recibido
al socket correcto
Basado en puerto (UDP)
Basado en 4-tupla (TCP)

■ = socket ○ = proceso



Cómo trabaja la demultiplexación?

- **El host Rx recibe datagramas IP**
 - Cada datagrama tiene dirección IP fuente y dirección IP destino
 - Cada datagrama incluye 1 segmento de la capa transporte
 - Cada segmento tiene números de puerto fuente y destino
- En general el host usa las direcciones **IPs** y **números de puertos** para conducir un segmento al socket apropiado
- Puertos 0-1023 reservados por protocolos establecidos, HTTP: 80, FTP: 21, están definidos en RFC 1700



Formato de segmento TCP/UDP

Demultiplexación sin conexión (UDP)

- ❑ Capa de transporte reconoce Socket destino por puerto destino y la aplicación reconocer remitente por la IP y puerto origen.
- ❑ Cuando la capa transporte recibe un segmento UDP (lado receptor):
 - Chequea número de puerto destino en segmento
 - Dirige el segmento UDP al socket con ese número de puerto
- ❑ Datagramas IP con direcciones IP y/o números de puerto origen diferentes pueden ser dirigidos al mismo socket destino.

Re-visitemos: Código Cliente Socket UDP

- Cliente: Creamos socket y datagrama con núm. de puerto:

```
DatagramSocket clientSocket = new DatagramSocket();
```

```
...
```

```
InetAddress IPAddress = InetAddress.getByName("hostname");
```

```
...
```

```
DatagramPacket sendPacket =
```

```
new DatagramPacket(sendData, sendData.length, IPAddress, 9876);
```

```
...
```

```
clientSocket.send(sendPacket);
```

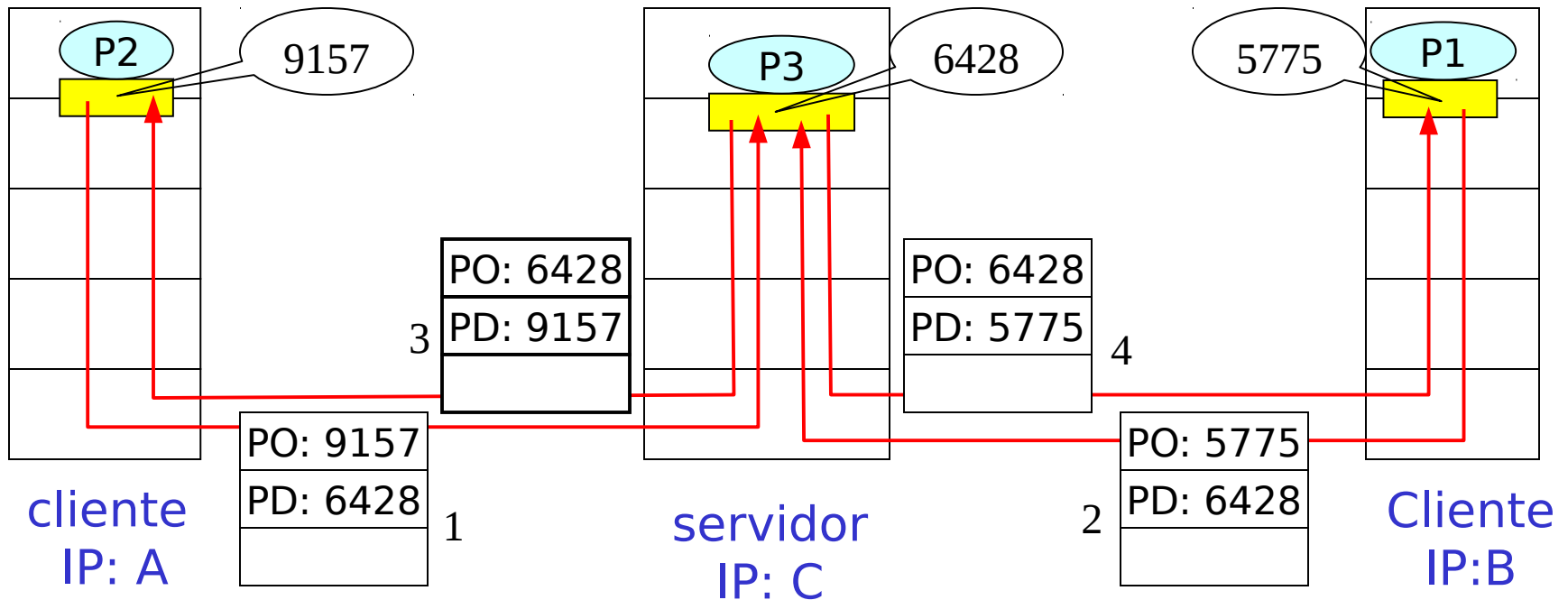
- En la red, el Socket UDP destino queda identificado por (2-tuple):

(Dirección IP destino, Número puerto destino)

Re-visitemos: Código Servidor Socket UDP

- ❑ Servidor crea el socket en un puerto específico:
DatagramSocket **serverSocket** = new DatagramSocket(9876);
...
❑ Recibe paquete del cliente, usa IP y puerto del mensaje para responder:
DatagramPacket receivePacket =
 new DatagramPacket(receiveData, receiveData.length);
serverSocket.receive(receivePacket); // paquete recibido
...
InetAddress IPAddress = **receivePacket.getAddress();** //saca IP y
int port = **receivePacket.getPort();** // puerto origen
DatagramPacket sendPacket =
 new DatagramPacket(sendData, sendData.length, IPAddress, port);
...
❑ Servidor responde usando la información obtenida del paquete:
serverSocket.send(sendPacket);

Demultiplexación sin conexión (cont)



PO: puerto origen, junto a IP origen proveen “dirección de retorno”

Un mismo socket recibe mensajes desde distintos hosts. La aplicación los distingue por la IP y puerto origen de cada mensaje.

Demultiplexación orientada a la conexión (TCP)

- ❑ Un socket maneja solo un cliente.
- ❑ Sockets TCP queda definido por (4-tupla):
 - Dirección IP Origen
 - Número de puerto Origen
 - Dirección IP Destino
 - Número de puerto Destino
- ❑ Host receptor usa los cuatro valores para dirigir los segmentos al socket apropiado.
- ❑ Un host servidor puede soportar muchos sockets TCP en un mismo puerto simultáneos:
 - Cada socket es identificado por su 4-tupla propia
- ❑ Un servidores Web tiene un socket diferente por cada cliente conectado
 - HTTP no-persistente tendrá diferentes sockets por cada petición de datos

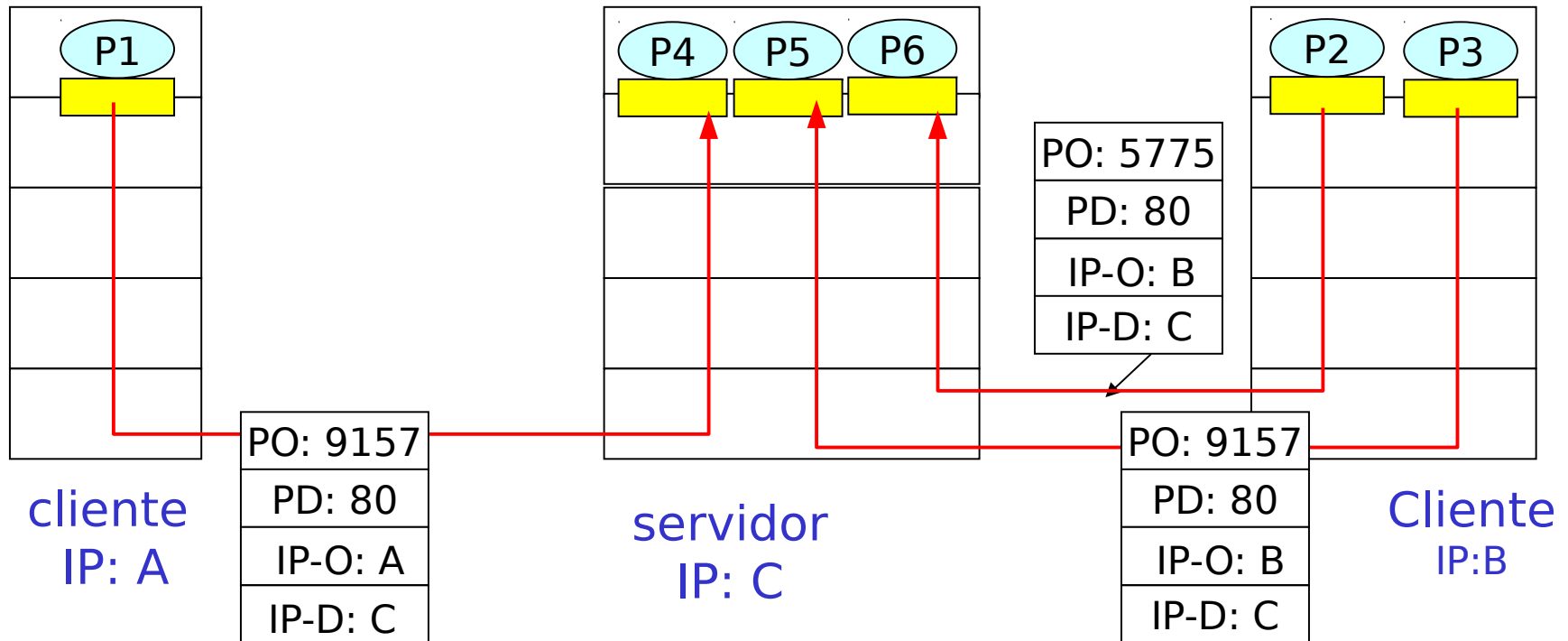
Re-visitemos: Código Cliente Socket TCP

- Cliente: Creamos socket con IP y núm. de puerto:
Socket `clientSocket` = new Socket("hostname", 6789);
...
DataOutputStream `outToServer` = new
DataOutputStream(`clientSocket.getOutputStream()`);
- La capa de transporte asigna un puerto local para el socket del cliente.

Re-visitemos: Código Servidor Socket TCP

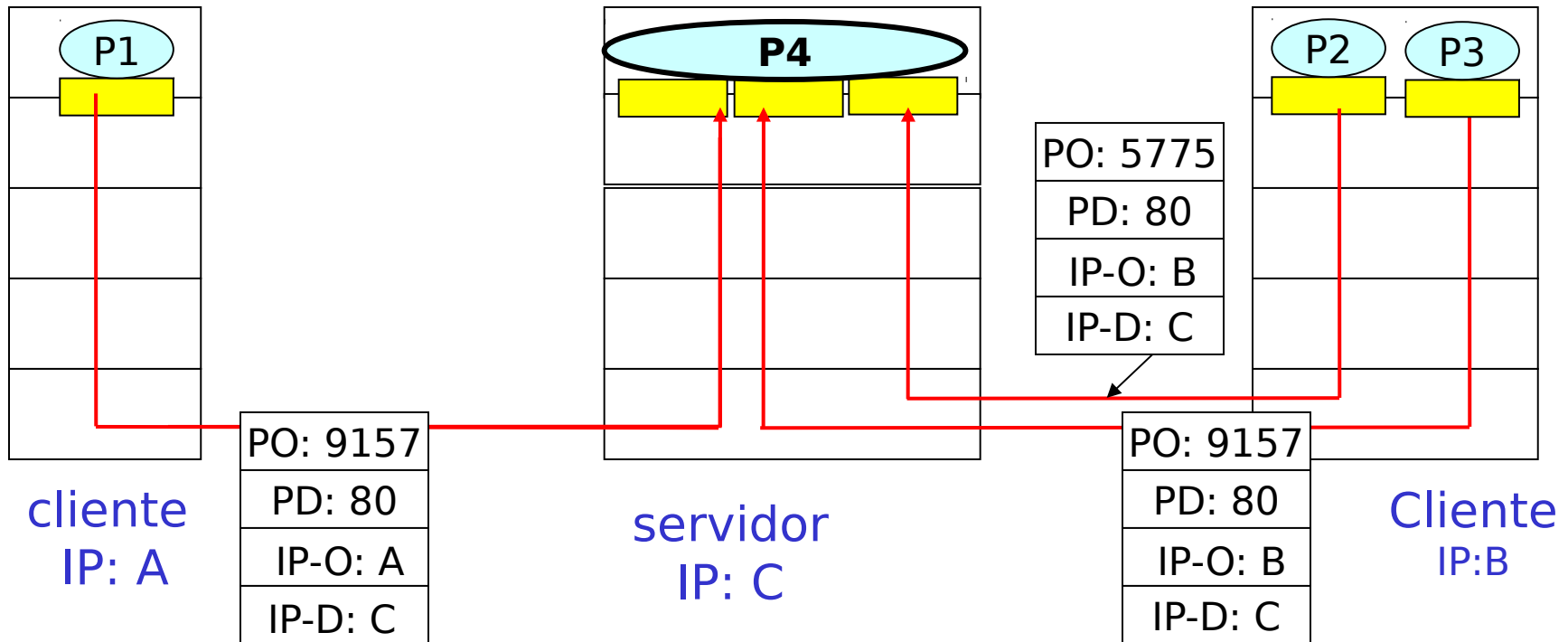
- ❑ Servidor crea el welcome socket en un puerto específico:
`ServerSocket welcomeSocket = new ServerSocket(6789);`
- ❑ Al aceptar una conexión se crea un nuevo socket para este cliente específico:
`Socket connectionSocket = welcomeSocket.accept();`
- ❑ Se lee desde socket de conexión:
`BufferedReader inFromClient = new BufferedReader(
 new InputStreamReader(connectionSocket.getInputStream()));`
- ❑ Se responde usando el mismo socket de conexión:
`DataOutputStream outToClient =
 new DataOutputStream(connectionSocket.getOutputStream());`

Demultiplexación orientada a la conexión (cont.)



Notar puede haber igual puerto Origen en A y B.
Igual puerto destino en ambas conexiones desde B.

Demultiplexación orientada a la conexión: Servidor Web con hebras (cont.)



Aquí un único proceso maneja **distintos sockets** para cada cliente. La programación de esto se estudia en ELO330

Contenido de Capítulo 3

- ❑ 3.1 Servicios de la capa transporte
- ❑ 3.2 Multiplexing y demultiplexing
- ❑ 3.3 Transporte sin conexión: UDP
- ❑ 3.4 Principios de transferencia confiable de datos
- ❑ 3.5 Transporte orientado a la conexión: TCP
 - Estructura de un segmento
 - Transferencia confiable de datos
 - Control de flujo
 - Gestión de la conexión
- ❑ 3.6 Principios del control de congestión
- ❑ 3.7 Control de congestión en TCP

UDP: User Datagram Protocol [RFC 768]

- ❑ Básicamente agrega puerto a paquete IP
- ❑ Servicio de “mejor esfuerzo”, un segmento UDP se puede:
 - perder
 - Entregar a la aplicación fuera de orden
- ❑ **Sin conexión:**
 - No hay handshaking (establecimiento de conexión) entre servidor y receptor UDP
 - Cada segmento UDP es manejado en forma independiente de los otros

“Mejor esfuerzo”: Haré todo lo posible por entregarlo, pero no lo garantizo.

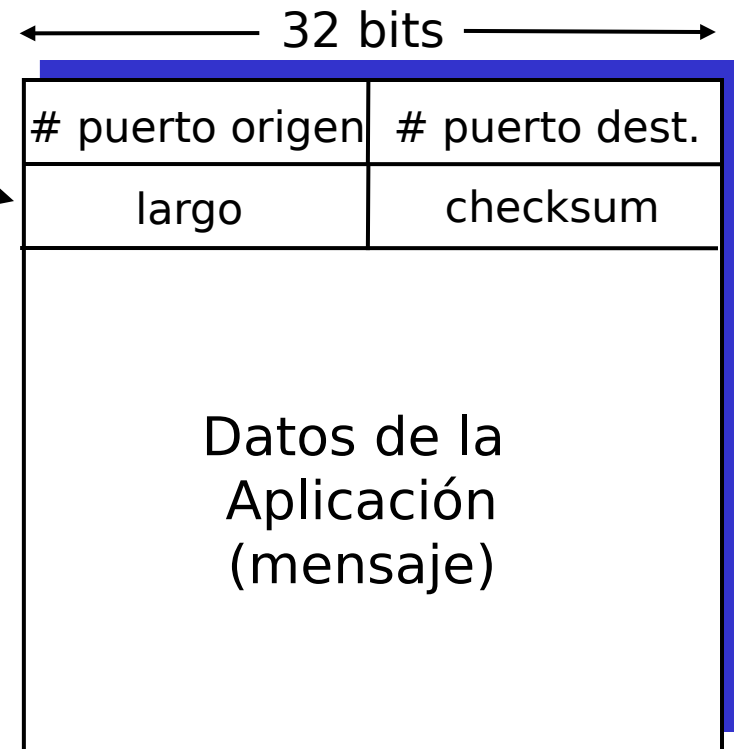
Por qué existe UDP?

- ❑ No requiere establecimiento de conexión (lo cual agregaría retardo)
- ❑ Simple: no se requiere mantener estado en el Tx y el Rx
- ❑ Pequeño segmento de encabezado => menor overhead
- ❑ No hay control de congestión: UDP puede transmitir tan rápido como se desee

UDP: más detalle

- A menudo es usado por flujos (streaming) multimedia en aplicaciones que:
 - Toleran pérdida de datos
 - Son sensibles a la tasa de transmisión
- Otros usos de UDP
 - DNS
 - SNMP (Simple Network Management Protocol)
- Es posible lograr transferencias confiables sobre UDP: se debe programar la confiabilidad en la capa aplicación
 - En este caso la recuperación de errores depende de la aplicación!

Largo, en bytes del segmento UDP, incluyendo encabezado



Formato segmento UDP

Checksum UDP (suma de chequeo)

Objetivo: detectar “errores” (e.g., bits cambiados) en segmentos transmitidos

Transmisor:

- ❑ Trata el contenido de cada segmento como una secuencia de enteros de 16 bits
- ❑ checksum: “suma” del contenido del segmento y luego invierte cada bits (se conoce como tomar el complemento 1).
- ❑ Transmisor pone el valor del **checksum** en el campo checksum del datagrama UDP
- ❑ Incluye algunos campos de IP (para los que quieren comprobarlo con wireshark).

Receptor:

- ❑ Calcula **el checksum** del segmento recibido
- ❑ Chequea si el **checksum calculado corresponde** al valor de **checksum recibido** en el campo:
 - NO corresponde => error detectado
 - SI => no hay error detectado. *Pero podrían haber errores sin embargo!*

Ejemplo Checksum en Internet

□ Notar

- Cuando sumamos números, la reserva del bit más significativo debe ser sumada al resultados
- Tomar el complemento 1 no es más que invertir los bits

□ Ejemplo: sumar dos enteros de 16-bits

	1 1 1 0 0 1 1 0 0 1 1 0 0 1 1 0
	1 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1
	<hr/>
Sumar reserva	1 1 0 1 1 1 0 1 1 1 0 1 1 1 0 1 1
	<hr/>
“suma”	1 0 1 1 1 0 1 1 1 0 1 1 1 1 0 0
checksum	0 1 0 0 0 1 0 0 0 1 0 0 0 0 1 1

Contenido de Capítulo 3

- ❑ 3.1 Servicios de la capa transporte
- ❑ 3.2 Multiplexing y demultiplexing
- ❑ 3.3 Transporte sin conexión: UDP
- ❑ 3.4 Principios de transferencia confiable de datos
- ❑ 3.5 Transporte orientado a la conexión: TCP
 - Estructura de un segmento
 - Transferencia confiable de datos
 - Control de flujo
 - Gestión de la conexión
- ❑ 3.6 Principios del control de congestión
- ❑ 3.7 Control de congestión en TCP