Capítulo 3: Capa Transporte - I

Este material está basado en:

- material de apoyo al texto *Computer Networking: A Top Down Approach Featuring the Internet 3rd* edition. Jim Kurose, Keith Ross Addison-Wesley, 2004.

Capítulo 3: Capa Transporte

Objetivos:

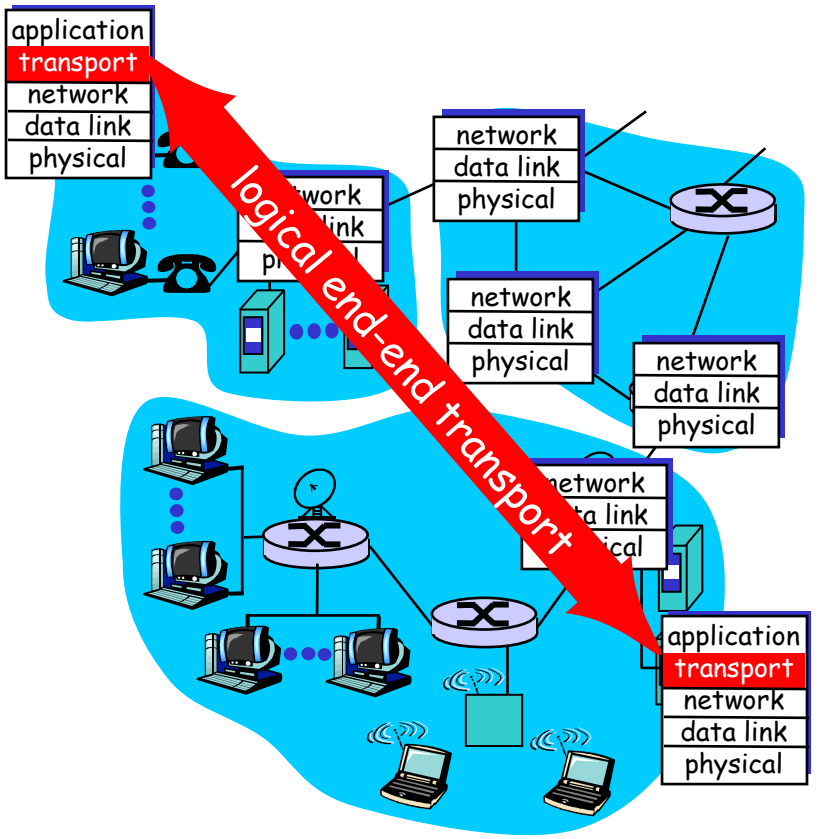
- Entender los principios detrás de los servicios de la capa transporte:
  - Multiplexing / demultiplexing
  - Transferencia de datos confiable
  - Control de flujo
  - Control de congestión
- Aprender sobre los protocolos de transporte en la Internet:
  - UDP: transporte sin conexión
  - TCP: transporte 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 aplicaciónes corriendo en diferentes hosts
- Los protocolos de transporte corren en sistemas terminales (computadores, no equipos internos como routers)
  - Lado Tx: divide el mensaje de la aplicación en segmentos, y los pasa a la capa de red
  - Lado Rx: re-ensambla los segmentos del mensaje , y lo pasa a la capa aplicación
- Hay más de un protocolo de transporte disponible para las aplicaciones
  - Internet: TCP y UDP



Capa transporte vs. capa red

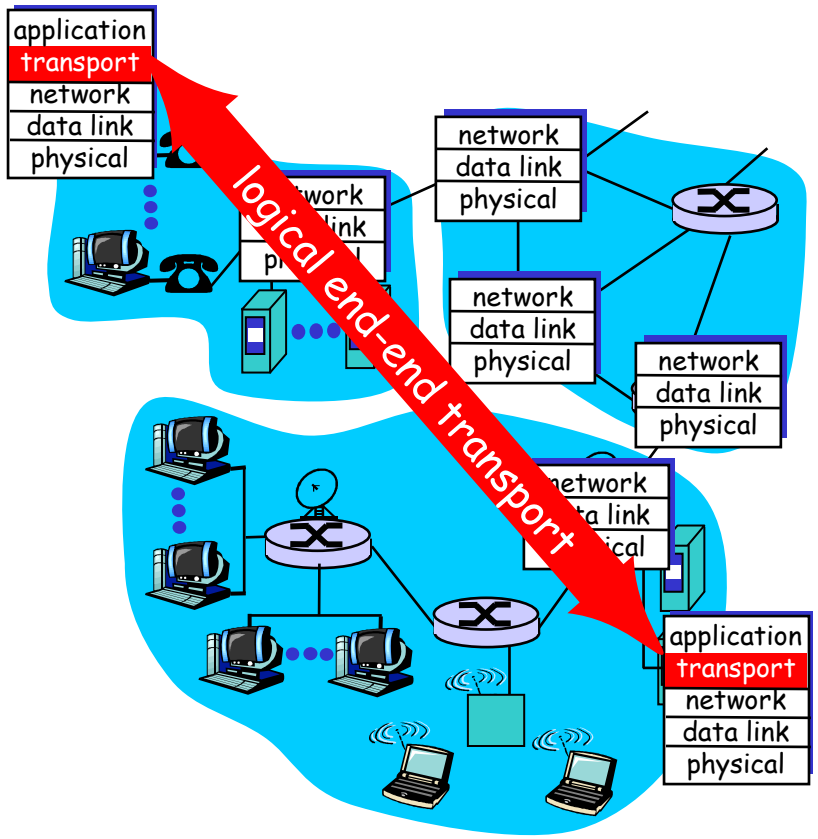
- *Capa de red:* encargada de la comunicación lógica entre hosts
- *Capa transporte:* encargada de la comunicación lógica entre procesos
  - Depende de y mejora los servicios de la capa de red

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

- Entrega confiable y en orden (TCP)
  - Tiene: control de congestión
  - Control de flujo
  - Establecimiento de conexión
- Entrega no confiable, talvez desordenada: (UDP)
  - Básicamente el mismo servicio que "mejor esfuerzo ("best-effort") IP
- Qué servicios no se ofrecen:
  - Garantías de retardo
  - Garantías de ancho de banda
  - (Básicamente porque no es fácil 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

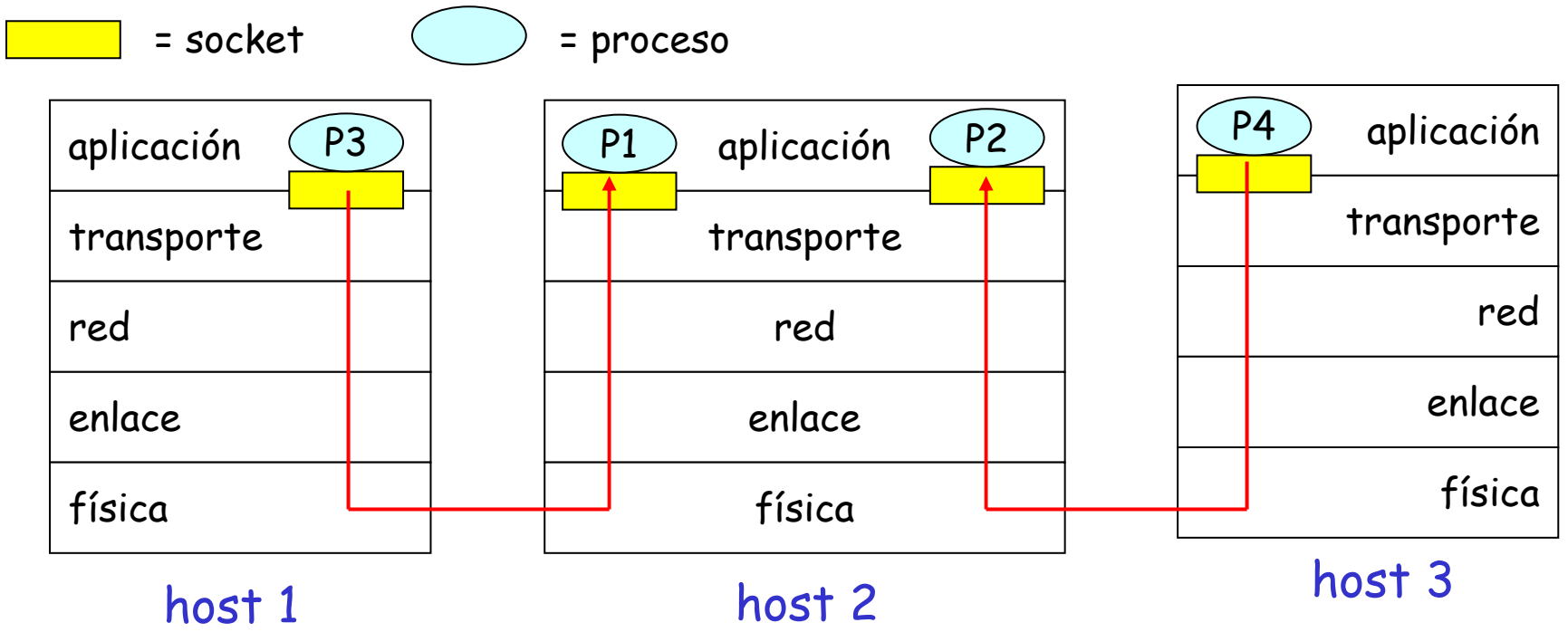
Multiplexación/demultiplexación

Multiplexación en host Tx:

Recolección de datos desde múltiples sockets, incorporar encabezado (luego usado para demulteplexación

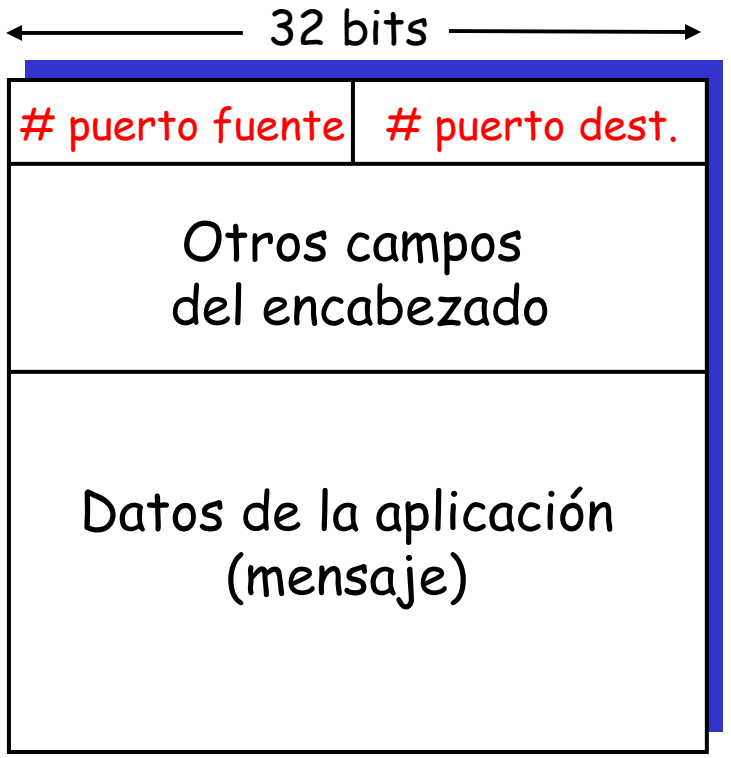
Demultiplexación en host Rx:

Entrega el segmento recibido al socket correcto



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 (recordar: hay números de puerto conocidos para aplicaciones específicas)
- El host usa direccion IP y número de puerto para conducir un segmento al socket apropiado
- Puertos 0-1023 reservados por protocolos establecidos (well known port numbers), HTTP: 80, FTP: 21, ver RFC 1700



Formato de segmento TCP/UDP

Demultiplexación sin conexión (UDP)

- Cuando la capa transporte en un host Rx recibe un segmento UDP:
  - 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

Repaso: 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);
```

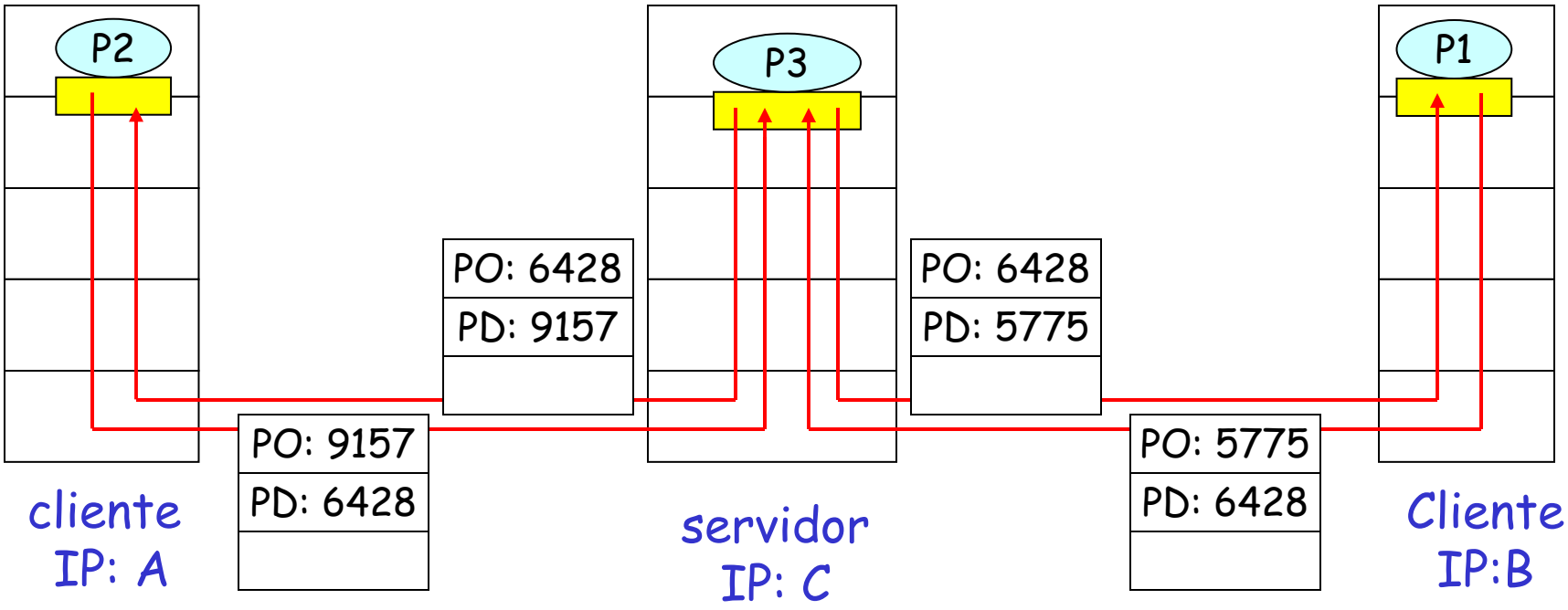
- El Socket UDP queda identificado por (2-tuple):

  (Dirección IP destino, Número puerto destino)

Repaso: Código Servidor Socket UDP

- Servidor crea el socket en un puerto especifico:
  DatagramSocket serverSocket = new DatagramSocket(9876);
  ...
- Recibe paquete del cliente y saca ip y puerto para responder:
  DatagramPacket receivePacket =
  new DatagramPacket(receiveData, receiveData.length);
  serverSocket.receive(receivePacket);
  ...
  InetAddress IPAddress = receivePacket.getAddress();
  int port = receivePacket.getPort();
  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, provee "dirección de retorno"

Demultiplexación orientada a la conexión

- Sockets TCP queda definido por (4-tuple):
  - 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 simultáneos:
  - Cada socket es identificado por su 4-tupla propia
- Servidores Web tiene sockets diferentes por cada cliente conectado
  - HTTP no-persistente tendrá diferentes sockets por cada petición de datos

Repaso: 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());
```

Repaso: Código Servidor Socket TCP

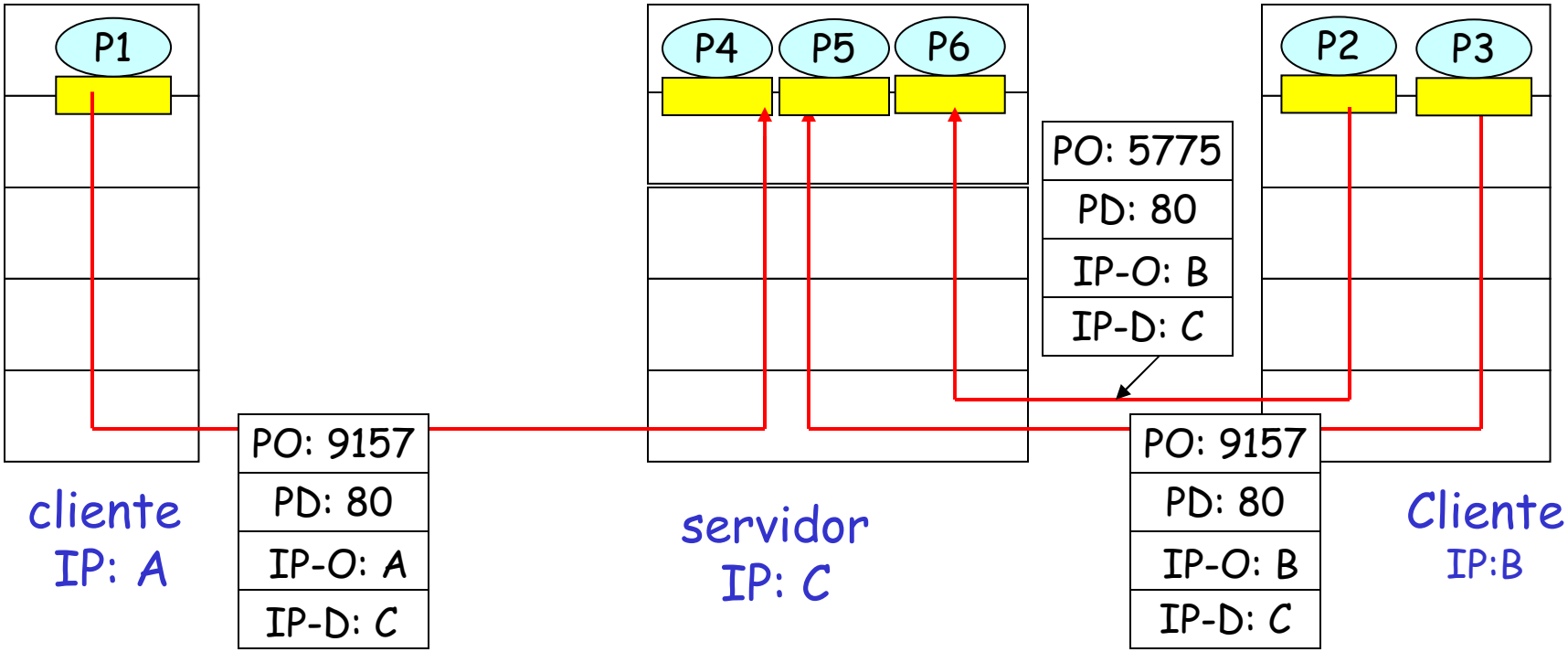
- Servidor crea el welcome socket en un puerto especifico:
  ServerSocket welcomeSocket = new ServerSocket(6789);

- Se asocia el welcome socket al socket de conexión:
  Socket connectionSocket = welcomeSocket.accept();

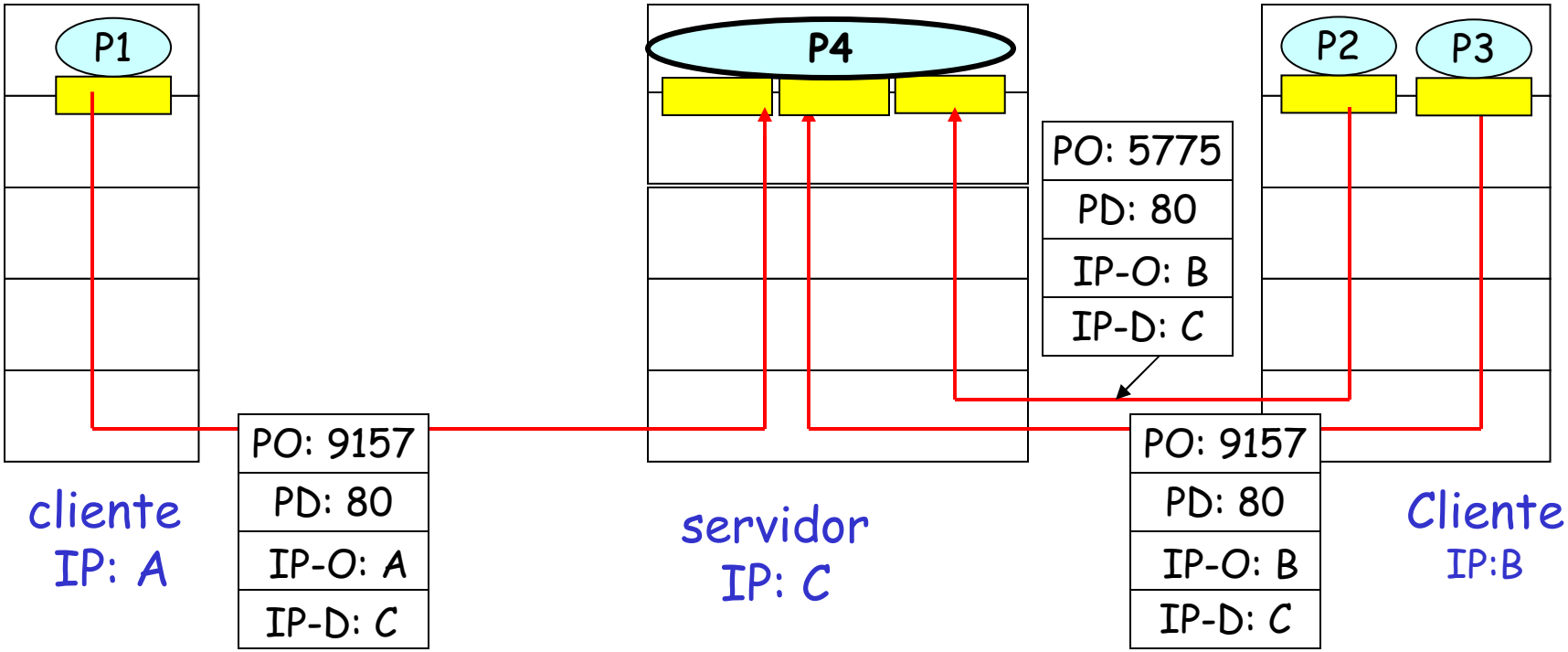
- Se lee del socket de conexion:
  BufferedReader inFromClient =
  new BufferedReader(new
  InputStreamReader(connectionSocket.getInputStream()));

- Se responde usando el mismo socket de conexion:
  DataOutputStream outToClient =
  new DataOutputStream(connectionSocket.getOutputStream());

Demultiplexación orientada a la conexión (cont.)



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



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]

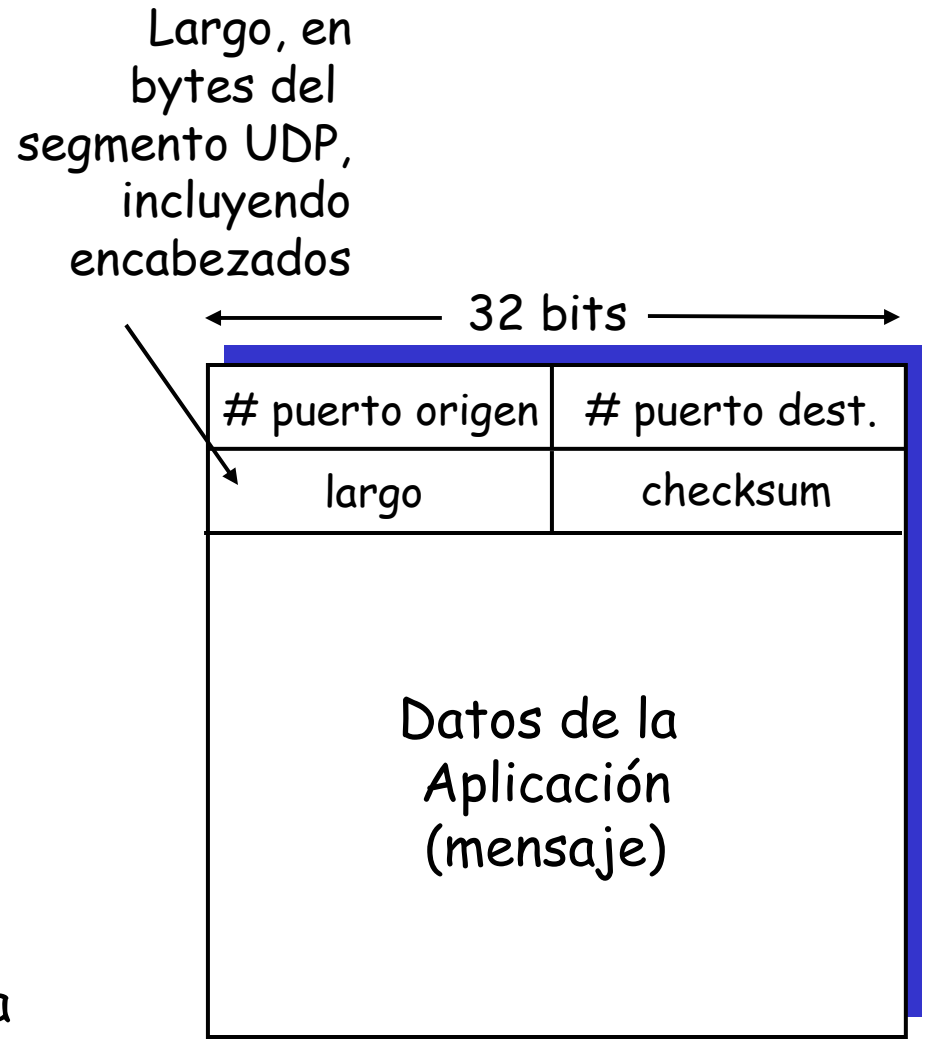
- Protocolo Internet de transporte "sin costos adicionales"
- Servicio de "mejor esfuerzo", un segmento UDP puede ser:
  - perdido
  - Entregado a la aplicación fuera de orden
- *Sin conexión:*
  - No hay handshaking (establecimiento de conexión) entre servidor y receptor UDP
  - Cada segmento UDP en manejado en forma independiente de los otros

Por qué existe UDP?

- No requiere establecimiento de conexión (lo cual agrega 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

- A menudo es usado por flujos (streaming) multimedia en aplicaciones por:
  - Tolerancia a pérdida
  - Sensibilidad a la tasa de transmisión
- Otros usos de UDP
  - DNS
  - SNMP (Simple Network Management Protocol)
- Transferencia confiable sobre UDP: agrega confiabilidad en la capa aplicación
  - Recuperación de errores específicos según la aplicación!



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 tomamos el complemento 1.
- Transmisor pone el valor del checksum en el campo checksum del datagrama UDP

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?*

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