

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:

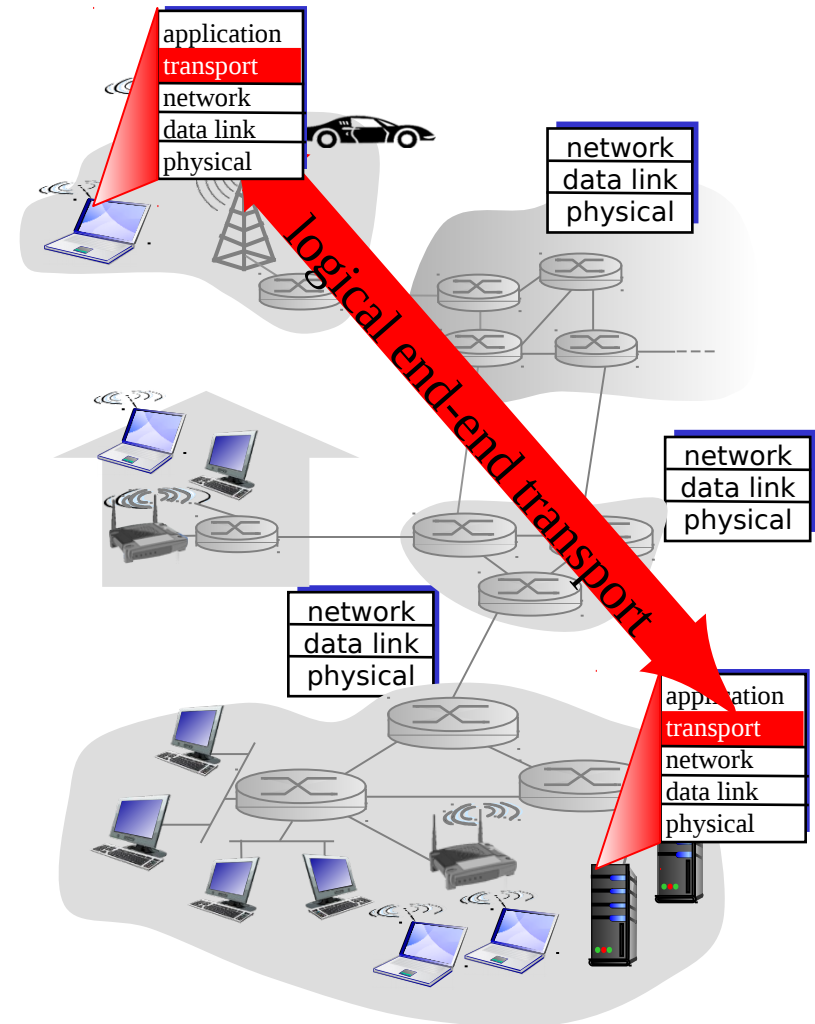
- ❑ Entender los principios detrás de los servicios de la capa transporte:
 - Multiplexing / demultiplexing
 - Transferencia confiable de datos
 - Control de flujo
 - Control de congestión
- ❑ Aprender sobre los protocolos de transporte en la Internet:
 - UDP (**U**ser **D**atagram **P**rotocol): transporte 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 y no son requeridos en equipos internos como routers)
- Hay más de un protocolo de transporte disponible para las aplicaciones
 - Más comunes: **TCP** y **UDP**
 - Hoy también **SCTP** (Stream Control Transmission Protocol, 2000), y **DCCP** (Datagram Congestion Control Protocol, 2005)



Capa transporte vs. capa red

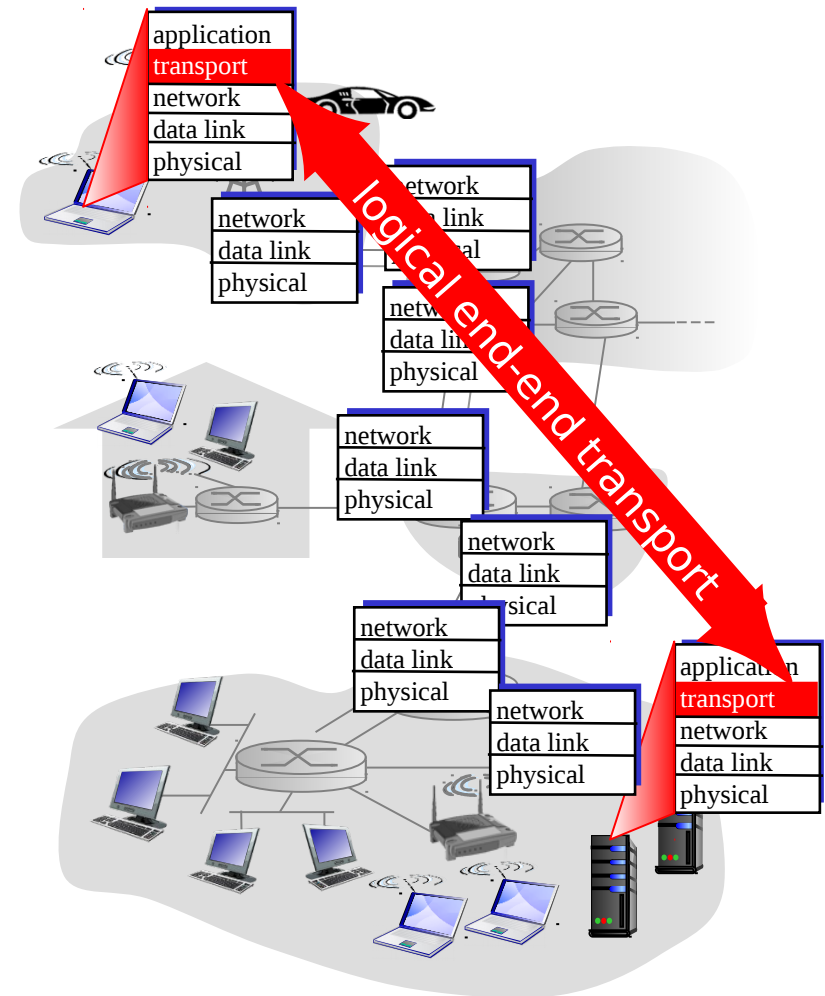
- *Capa transporte:* encargada de la comunicación lógica entre procesos
 - Depende y mejora los servicios de la capa de red
- *Capa de red:* encargada de la comunicación lógica entre hosts

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
 - No se ofrecen 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

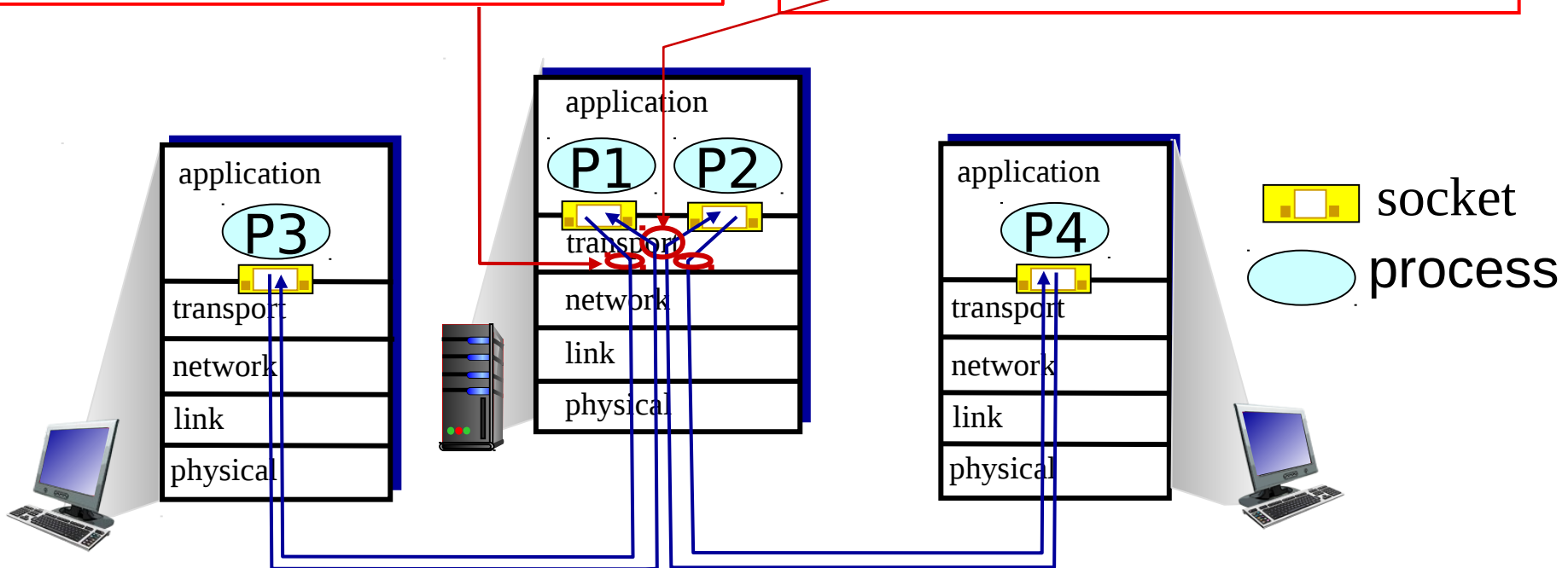
Multiplexación/demultiplexación

Multiplexación en host Tx:

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

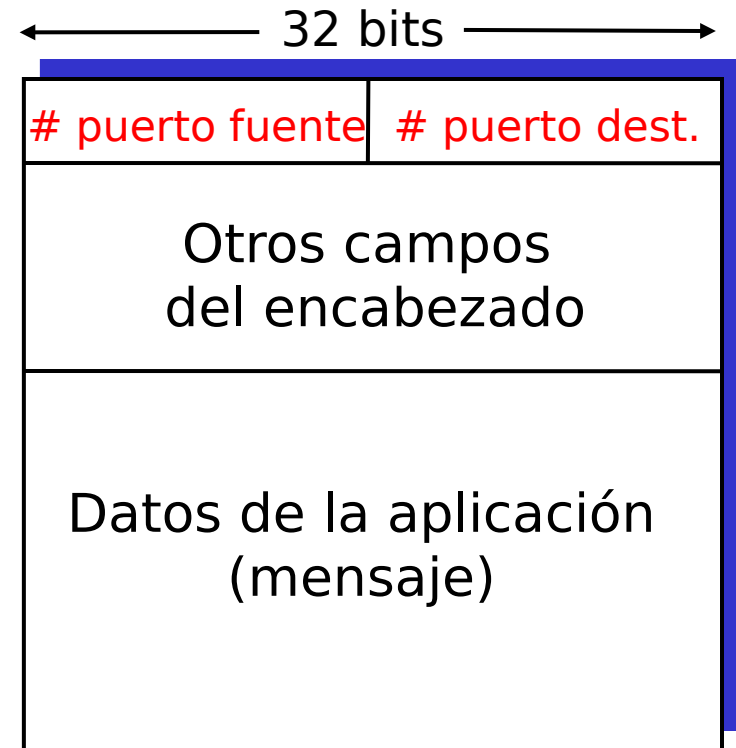
Demultiplexación en host Rx:

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



Cómo trabaja la demultiplexación?

- **El host Rx recibe datagramas IP**
 - Cada datagrama tiene dirección IP fuente y dirección IP destino
 - Cada datagrama lleva 1 segmento de la capa transporte
 - Cada segmento tiene números de puerto fuente y destino
- Un host usa las direcciones **IPs** y **números de puertos** para conducir un segmento al socket apropiado
- Puertos 0-1023 están reservados para protocolos establecidos, HTTP: 80, ssh: 22, están definidos en RFC 1700



Formato de segmento TCP/UDP

Demultiplexación sin conexión (UDP)

- ❑ Capa de transporte del **receptor** 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:

```
serverName = 'hostname'
```

```
serverPort = 12000
```

```
clientSocket = socket(AF_INET,SOCK_DGRAM)
```

```
message = .....
```

```
clientSocket.sendto(message,(serverName, serverPort))
```

- Durante su transporte en la red, el Socket UDP destino queda identificado por (2-tuple):

(Dirección IP destino, Número puerto destino)

Usado por capa de red para conducirlo al computador destino

Usado por capa de transporte para entregarlo al socket adecuado

Re-visitemos: Código Servidor Socket UDP

- ❑ Servidor crea el socket en un puerto específico:

```
serverPort = 12000
```

```
serverSocket = socket(AF_INET, SOCK_DGRAM)
```

```
serverSocket.bind(('', serverPort))
```

```
:
```

- ❑ Recibe paquete del cliente, usa dirección IP y puerto orígenes del mensaje para responder:

```
message, clientAddress = serverSocket.recvfrom(2048)
```

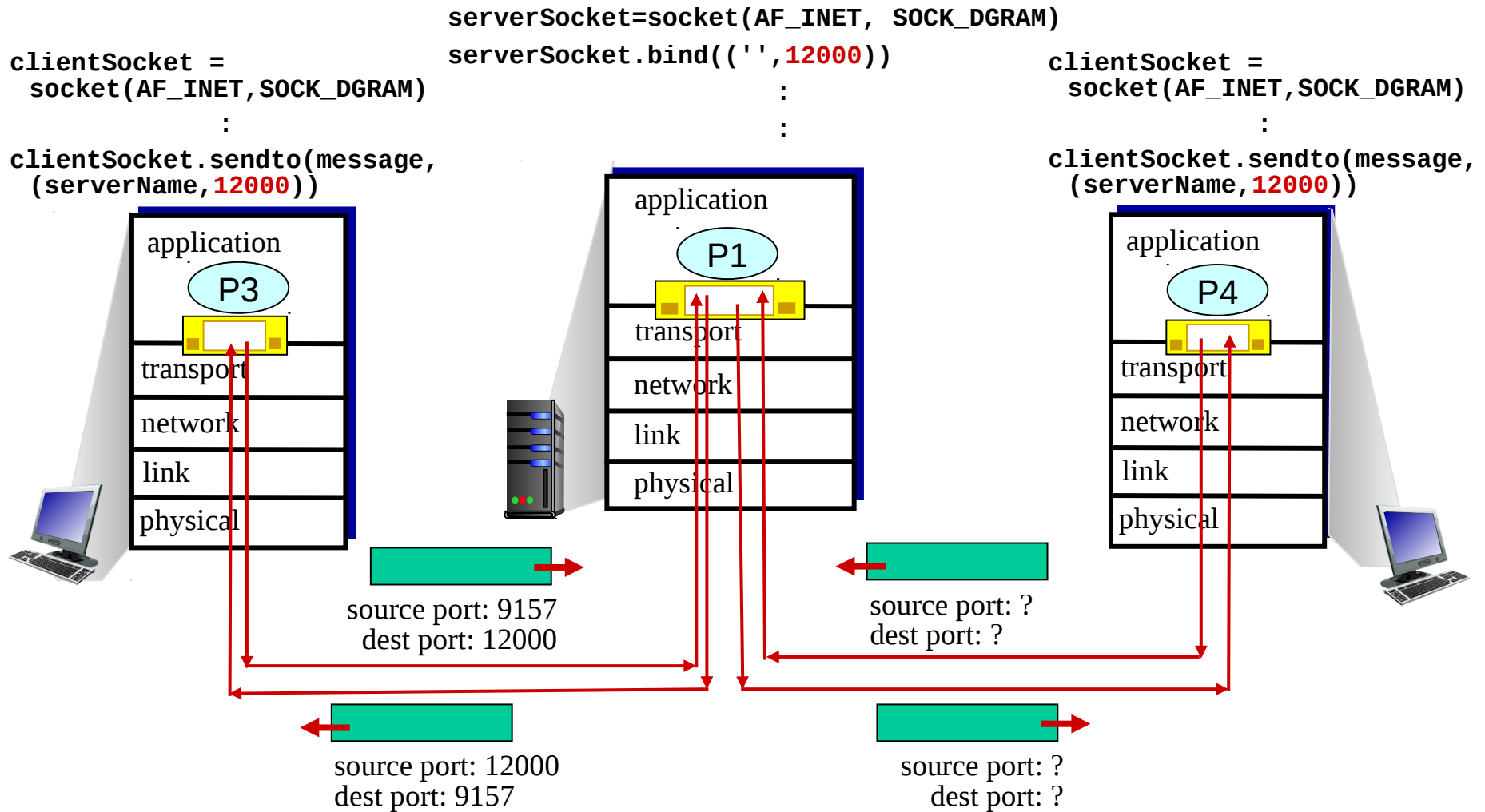
```
:
```

- ❑ Servidor responde usando la información obtenida del paquete:

```
serverSocket.sendto(modifiedMessage, clientAddress)
```



Demultiplexión sin conexión (ejemplo)



Un mismo socket recibe mensajes desde distintos hosts. La aplicación los distingue por la dirección 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 servidor puede mantener muchos sockets TCP simultáneos en un mismo puerto:
 - 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, todos ellos con puerto 80 de su lado.

Re-visitemos: Código **Cliente** Socket TCP

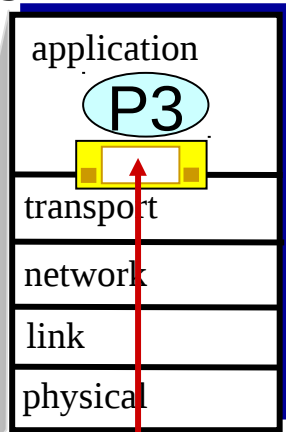
- ❑ Cliente: Creamos socket con direc. IP y núm. de puerto:
serverName = 'hostname'
serverPort = 12000
`clientSocket = socket(AF_INET, SOCK_STREAM)`
`clientSocket.connect((serverName,serverPort))`
:
- ❑ La capa de transporte asigna un puerto local para el socket del cliente.
- ❑ Para enviar información vía un socket conectado sólo se envía el dato (no requiere indicar destino cada vez)
sentence = raw_input('Input lowercase sentence: ')
`clientSocket.send(sentence)`

Re-visitemos: Código Servidor Socket TCP

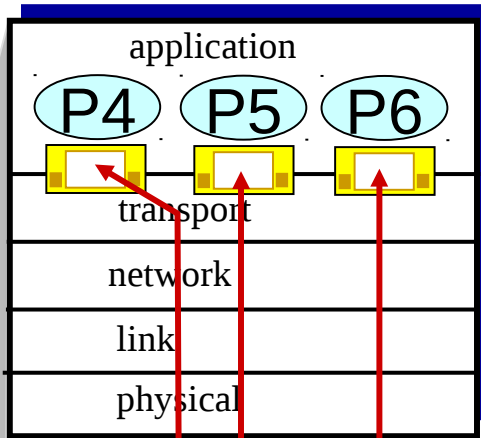
- ❑ Servidor crea el “welcome socket” en un puerto específico:
serverPort = 12000
serverSocket = socket(AF_INET,SOCK_STREAM) #welcome socket
serverSocket.bind(('',serverPort))
serverSocket.listen(1)
- ❑ Al aceptar una conexión se crea un nuevo socket para este cliente específico:
connectionSocket, addr = serverSocket.accept()
- ❑ Se lee desde socket de conexión:
sentence = connectionSocket.recv(1024)
- ❑ Se responde usando el mismo socket de conexión:
connectionSocket.send(capitalizedSentence)

Demultiplexación orientada a la conexión (cont.)

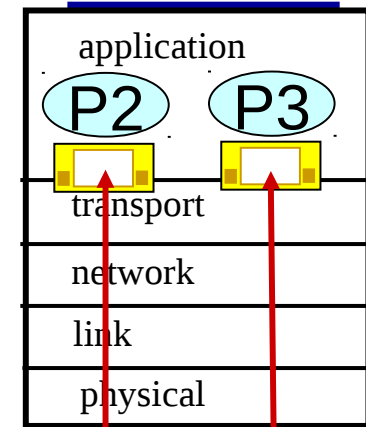
host: IP address=A



server: IP address=B



host: IP address=C



source IP,port: A,9157
dest IP, port: B,80

source IP,port: C,5775
dest IP,port: B,80

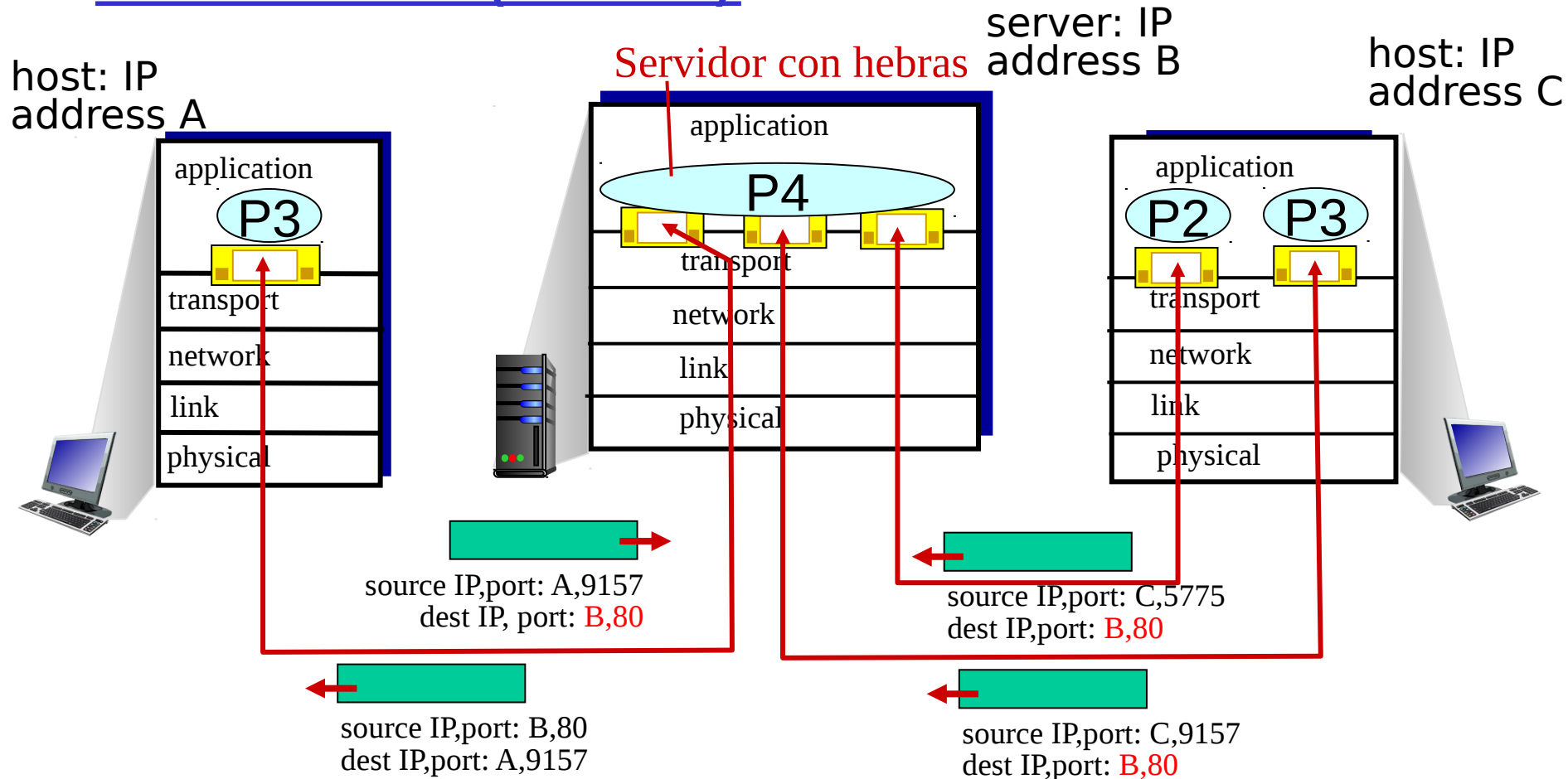
source IP,port: B,80
dest IP,port: A,9157

source IP,port: C,9157
dest IP,port: B,80

Tres segmentos destinados a IP address: B,
dest port: 80 son demultiplexados a sockets *diferentes*.

Esto es muy distinto a UDP

Demultiplexación orientada a la conexión (cont.)



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

Un autodidacta de Internet dice “Como el campo número de puerto es de 16 bits, un servidor TCP no pueden atender más de 2^{16} clientes simultáneamente.” ¿Está usted de acuerdo? Justifique.



- No es correcto. El rango para el número de puerto no limita el número de clientes conectados a un servidor. En general los clientes vienen desde distintas máquinas y el servidor creará un nuevo socket (no puerto) por cada cliente. Ciertamente hay un límite pero no está relacionado con el rango para los puertos pues todos los sockets usan un único puerto en el servidor.
- Cada conexión TCP queda definida por la 4-tupla: (IP_Origen, Puerto_origen, IP_destino, Puerto_destino)

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.

Usos de UDP

- ❑ Streaming de multimedios
 - toleran pérdidas
 - les afecta la tasa de transmisión
- ❑ DNS
- ❑ SNMP (Simple Network Management Protocol)
- ❑ Transferencia confiable sobre UDP:
 - Confiabilidad implementada en aplicación

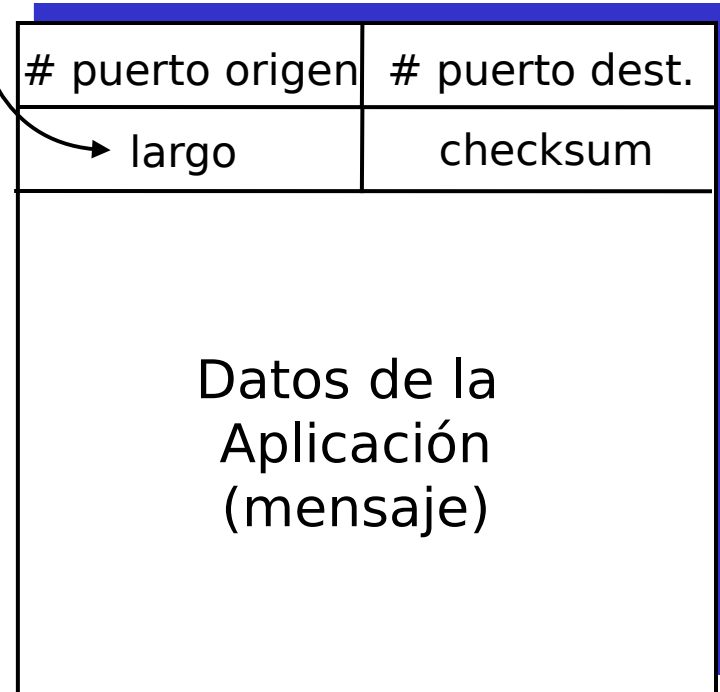
UDP: más detalle

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
- ❑ Encabezado pequeño => menor overhead
- ❑ Al no tener control de congestión puede tratar de transmitir tanto como se desee.

Largo, en bytes del segmento UDP, incluyendo encabezado

← 32 bits = 4 bytes →



Formato segmento UDP
“muy simple”

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 del Protocolo IP (para los que quieren comprobarlo con wireshark).

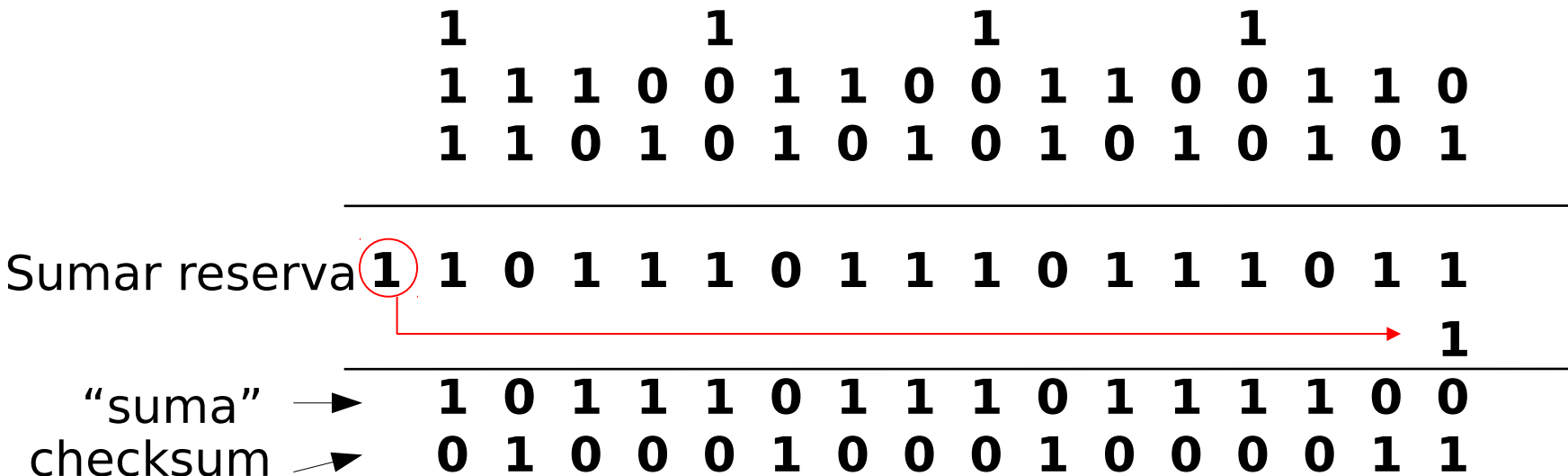
Receptor:

- ❑ Calcula **el checksum** del segmento recibido
- ❑ Chequea si el **checksum calculado corresponde** al valor de **checksum recibido** en el campo:
 - NO corresponde => error detectado, se descarta paquete
 - Sí => 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
- Luego se invierte cada bit (esto es tomar el complemento 1 del resultado).
- Ejemplo: sumar dos enteros de 16-bits



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