Redes de computadoras

Capa de transporte - Introducción

Las diapositivas están basadas en en libro:
“Redes de Computadoras – Un enfoque descendente”
de James F. Kurose & Keith W. Ross

La capa de transporte

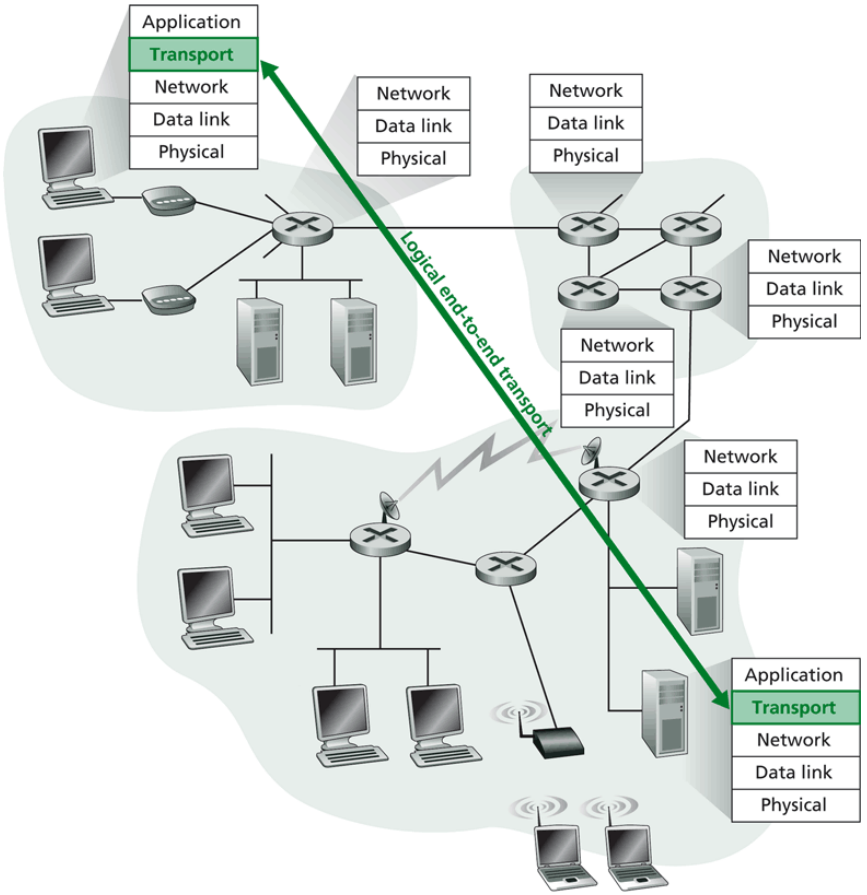

Figure 3.1 ♦ The transport layer provides logical rather than physical communication between application processes.

La capa de transporte proporciona una comunicación lógica en lugar de física entre los procesos de aplicación.

La capa de transporte

La capa de transporte proporciona servicios de comunicación a los procesos que se ejecutan en distintos hosts.

La capa de red, sobre la que se encuentra, cumple el rol de proporcionar una comunicación lógica entre hosts, mientras que la de transporte lo hará entre procesos de distintos hosts.

Los protocolos de transporte se ejecutan en los sistemas terminales.

- Lado del emisor: genera segmentos a partir de los mensajes de la aplicación y los pasa a la capa de red.
- El receptor: A partir de los segmentos, los mensajes son pasados a la capa de aplicación.

Protocolos a estudiar: TCP y UDP.

La capa de transporte

Los protocolos disponibles desde la capa de transporte a la capa de aplicación en Internet son:

TCP (Transmission Control Protocol)

– Confiable, orientado a la conexión, entrega en orden.

- control de congestión
- control de flujo
- establecimiento de la conexión

UDP (User Datagram Protocol)

– No confiable, entrega no ordenada

- No aporta mucho más de lo que ya hace el protocolo IP de la capa de red.

Servicios no disponibles:

- Garantías de retardo
- Garatías de ancho de banda

La capa de transporte

A los paquetes de la capa de transporte se los denomina “Segmentos”

Sin embargo en muchos documentos, incluídos los RFC utilizan para los paquetes UDP el termino “Datagramas” al igual que a los paquetes de la capa de red.

Multiplexación y Demultiplexación

Aplicaciones de red corriendo en simultaneo en un mismo host

HTTP, telnet, FTP, SMTP

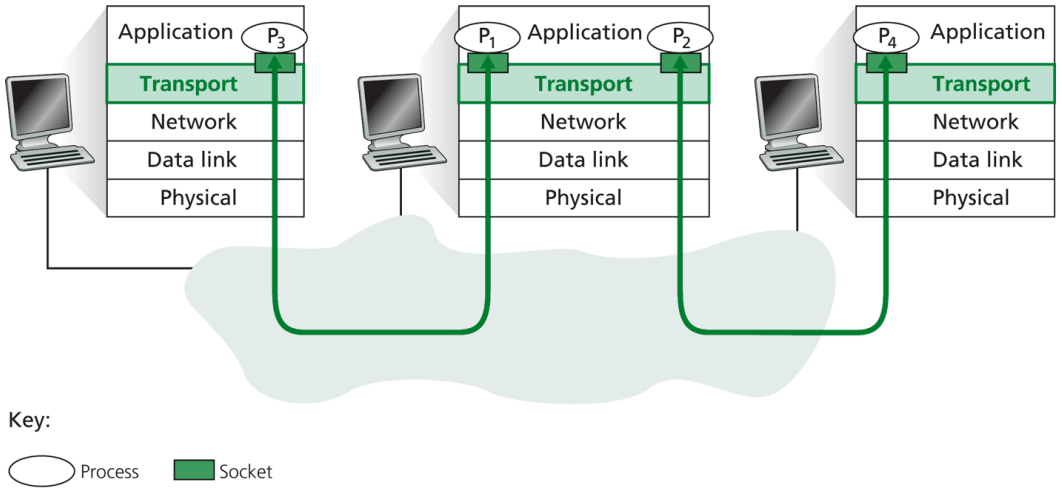

Figure 3.2 ♦ Transport-layer multiplexing and demultiplexing

Cuando la capa de transporte recibe datos procedentes de la capa de red, tiene que dirigir los datos a el proceso correcto.

La capa de transporte

Un proceso puede tener uno o más de un socket

La capa de transporte no entrega los datos directamente al proceso, sino que los “deposita” en un socket

Los sockets tienen un número identificador único

El formato del identificador varía entre sockets UDP y TCP

Demultiplexación

Cada segmento contiene un conjunto de campos para vincularlo al socket destino.

Al llegar al host receptor la capa de transporte examina dichos campos y envía el segmento al socket correspondiente.

Esta tarea es la que se denomina Desmultiplexación.

Multiplexación

En el host emisor la capa debe reunir los fragmentos de datos desde los sockets

Encapsular cada fragmento de datos con información de cabecera

- La información de la cabecera (header) es lo que luego se utilizará en el proceso de desmultiplexación.

Una vez creado el segmento es pasado a la capa de red

Multiplexación y demultiplexación

Un host recibe datagramas IP

- Cada datagrama contiene
  - Una dirección IP de origen
  - Una dirección IP de destino
- Cada datagrama lleva 1 segmento de la capa de transporte
- Cada segmento tiene puertos de origen y destino

Los host utilizan las direcciones IP y los números de puertos para enviar el segmento al socket apropiado.

- Número de puerto: TSAP (Transport Service Access Point)

Puertos

IANA (Internet Assigned Numbers Authority)

Se agrupan en 3 rangos

- Bien conocidos (Well-known port numbers): 0 – 1023
  - Aplicaciones de servidor.
  - Ejemplos: DNS 53, HTTP 80, FTP 20 y 21, SSH 22
- Registrados (Registered ports): 1024 – 49151
  - Aplicaciones de servidor, usuarios comunes
  - Radius 1812, 1813
- Dinámicos (Dynamic and-or private ports): 49152 – 65535
  - Para uso temporario, no se deben registrar

http://www.iana.org/assignments/port-numbers

Segmento

Los números de puerto son números de 16 bits (0 – 65535)
del 0 al 1023 números de puertos bien conocidos

(RFC 3232)

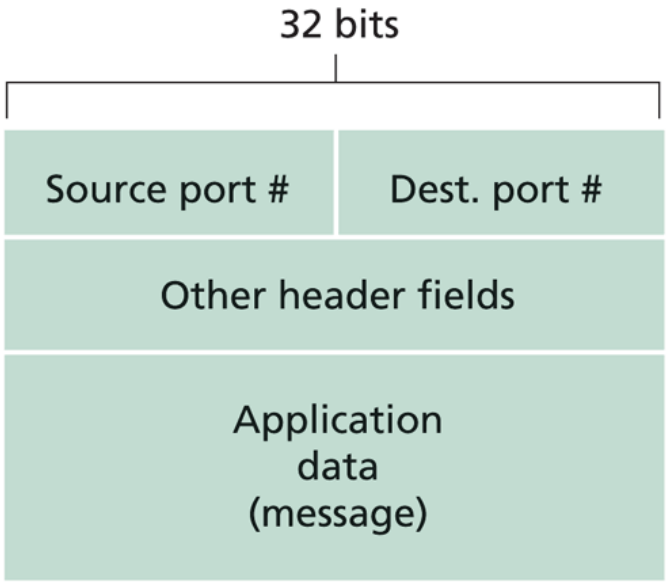

Figure 3.3 ♦ Source and destination port-number fields in a transport-layer segment

Multiplexación y demultiplexación sin conexión

Cuando se crea un socket UDP sin especificarle el número de puerto, este es asignado de forma automática por la capa de transporte, en el rango de 1024 a 65535

```
16          DatagramSocket socketCliente = new DatagramSocket();
```

Alternativamente es posible asignarlo de forma manual:

```
0           DatagramSocket socketServidor = new DatagramSocket(9876);
```

Normalmente el lado del cliente de la aplicación permite asignar a la capa de transporte el número de puerto de forma automática, mientras que en el servidor se asignará un puerto específico.

Multiplexación y demultiplexación sin conexión

Para que un host pueda ser identificado desde un host remoto basta la tupla **dirección IP, número de puerto puerto**.

Así un proceso en el **host A**, con el puerto UDP 28340, se podría comunicar con el **host B** al puerto 9876.

- La capa de transporte del host A crea un segmento incluyendo los datos de la aplicación, puerto origen y puerto destino.
- Se pasa el segmento a la capa de red la cual lo encapsula en un datagrama, haciendo el máximo esfuerzo por entregar el segmento al host receptor.
- En caso de llegar la capa de transporte del host B examina el número de puerto y entrega el segmento al socket identificado con el puerto.

Multiplexación y demultiplexación sin conexión

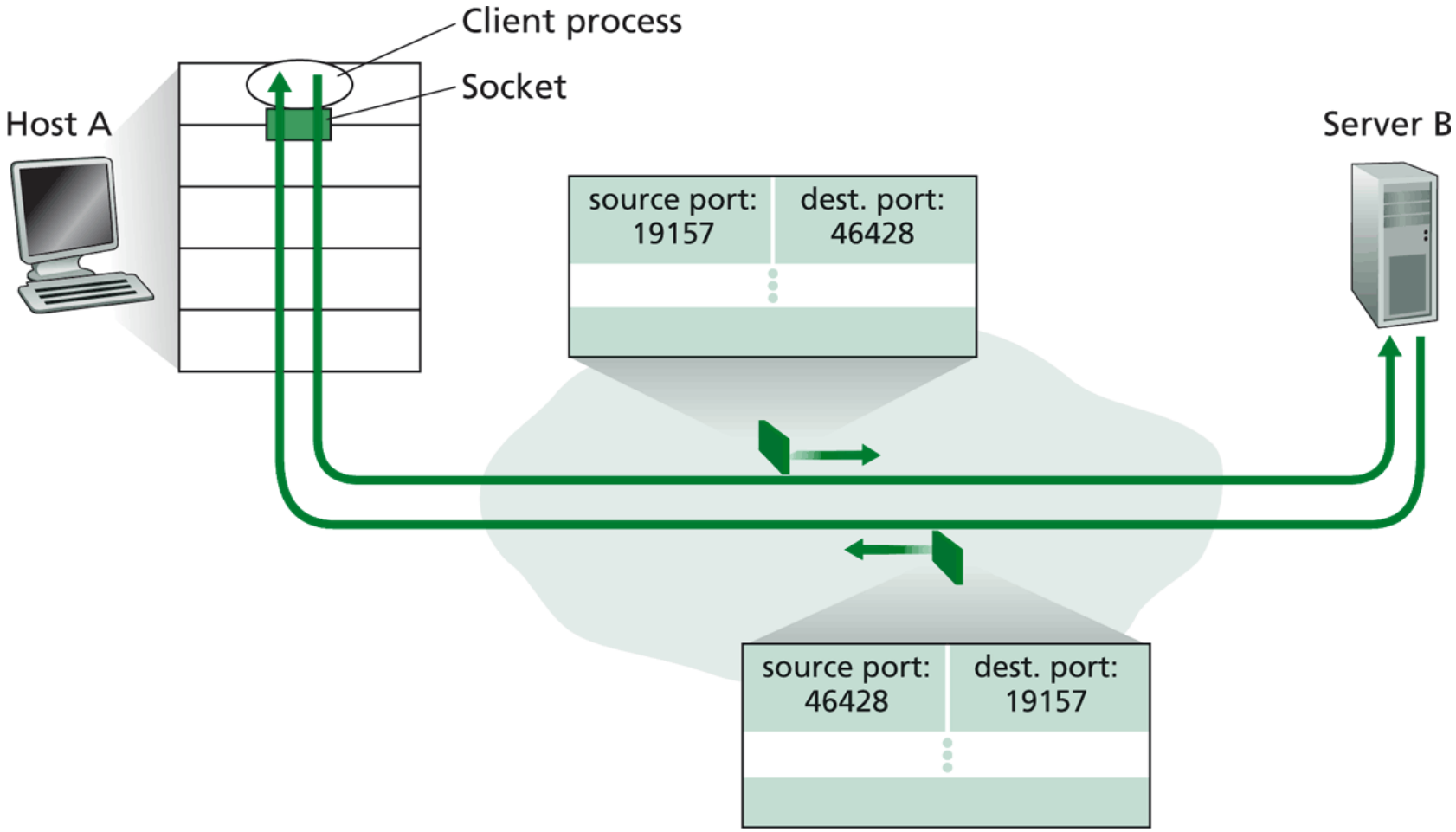

Figure 3.4 ♦ The inversion of source and destination port numbers

Multiplexación y demultiplexación orientada a la conexión

El Socket TCP se identifica por 4 elementos.

- Dirección IP origen
- Número de puerto origen
- Dirección IP destino
- Número de puerto destino

- El host destino utiliza los 4 valores para dirigir el segmento al socket apropiado

- El host servidor debe soportar varios sockets TCP simultáneos

- Los servidores web tienen diferentes sockets para cada cliente que se conecta

Multiplexación y demultiplexación orientada a la conexión

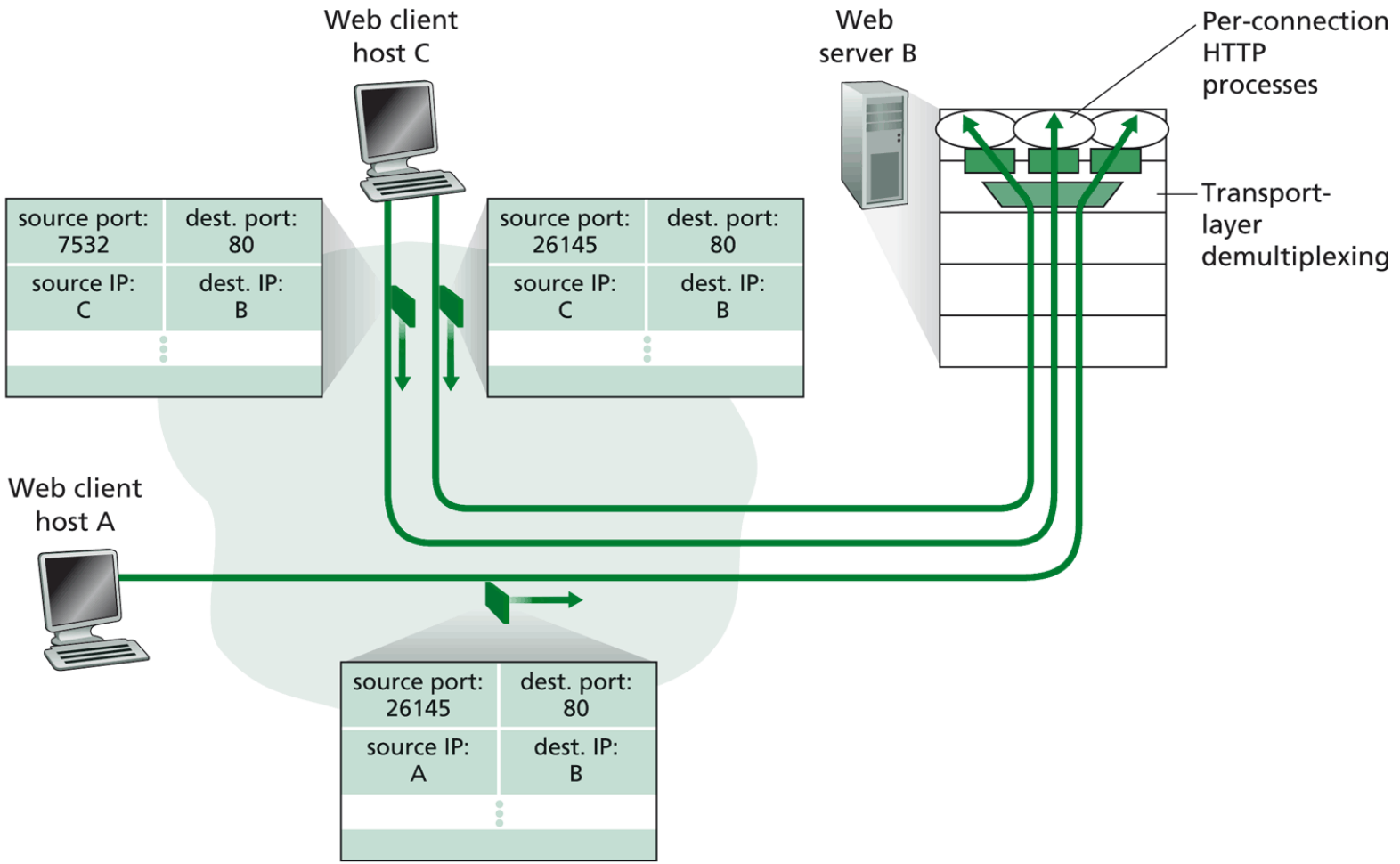

Figure 3.5 ♦ Two clients, using the same destination port number (80) to communicate with the same Web server application

Multiplexación y demultiplexación orientada a la conexión

En el caso de TCP el servidor utiliza un socket de bienvenida

```
13        ServerSocket socketBienvenida = new ServerSocket(6789);
```

El cliente crea un segmento de establecimiento de conexión al crear el socket

```
14        Socket socketCliente = new Socket("nombrehost", 6789);
```

El segmento contiene el puerto destino, el de origen y otras banderas para establecer la conexión.

Cuando el S.O. recibe el segmento, localiza al proceso asociado al puerto y crea un nuevo socket.

```
17            Socket socketConexion = socketBienvenida.accept();
```

Multiplexación y demultiplexación orientada a la conexión

La capa de transporte registra cuatro valores contenidos en el segmento de solicitud de conexión:

- número de puerto origen
- dirección IP origen
- número de puerto destino
- propia dirección IP

De este modo el nuevo socket queda identificado.

Los segmentos que lleguen y coincidan con estos valores serán dirigidos a este socket

Queda establecida la conexión y los host pueden intercambiar datos

Multiplexación y demultiplexación orientada a la conexión

De este modo un servidor puede brindar soporte a multiples sockets de forma simultanea

Los 4 valores anteriormente mencionados son utilizados para demultiplexar de forma apropiada al socket correspondiente

Servidor Web TCP

Un servidor web recibirá peticiones al puerto 80

Tanto segmentos para establecer comunicación como peticiones de datos

Se generan multiples sockets de conexión a través de los que recibirá y responderá peticiones HTTP.

Cada socket puede tener por detrás un proceso o hilo particular

- En caso de usar HTTP persistente mientras dure la conexión se enviaran multiples peticiones HTTP por el mismo socket.
- De lo contrario se crea un socket para cada petición

Enlaces

Class java.net.Socket

https://courses.cs.washington.edu/courses/cse341/98au/java/jdk1.2beta4/docs/api/java/net/Socket.html