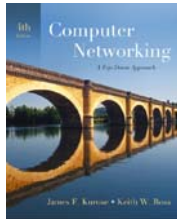


Introducción a las Redes de Computadores

Capítulo 3 Capa de Transporte



Nota acerca de las transparencias del curso:

Estas transparencias están basadas en el sitio web que acompaña el libro y han sido modificadas por los docentes del curso.

All material copyright 1996-2007
J.F. Kurose and K.W. Ross, All Rights Reserved

*Computer Networking:
A Top Down Approach
4th edition.*
Jim Kurose, Keith Ross
Addison-Wesley, July
2007.

Capítulo 3: Capa de Transporte

Objetivos:

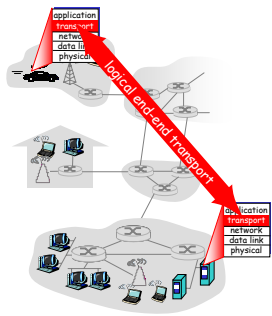
- Entender los principios detrás de los servicios de la capa de transporte:
 - multiplexación/demultiplexación
 - transferencia de datos confiable
 - control de flujo
 - control de congestión
- Aprender acerca de los protocolos de la capa de transporte en Internet:
 - UDP: transporte no orientado a conexión
 - TCP: transporte orientado a conexión
 - control de congestión de TCP

Capítulo 3: agenda

- 3.1 Servicios de la capa de transporte
- 3.2 Multiplexación y demultiplexación
- 3.3 Transporte no orientado a conexión: UDP
- 3.4 Principios de la transferencia de datos confiable
- 3.5 Transporte orientado a conexión: TCP
 - estructura del segmento
 - transferencia de datos confiable
 - control de flujo
 - gestión de la conexión
- 3.6 Principios del control de congestión
- 3.7 Control de congestión de TCP

Protocolos y servicios de transporte

- brinda **comunicación lógica** entre procesos de Aplicación corriendo en *hosts* diferentes
- los protocolos de transporte corren en los *end systems*
 - lado del transmisor: genera **segmentos** a partir de los mensajes de la Aplicación, y los pasa a la capa de red
 - lado del receptor: a partir de los segmentos, los mensajes son pasados a la capa de Aplicación
- más de un protocolo de transporte disponible para las aplicaciones
 - Internet: TCP y UDP



Int. Redes de Computadores - Capa de Transporte 3-4

Capa de Transporte vs. Capa de Red

- **Capa de red:** comunicación lógica entre *hosts*
- **Capa de transporte:** comunicación lógica entre **procesos** ejecutándose en diferentes máquinas
- **Los dispositivos intermedios (routers por ejemplo), implementan protocolos de capa de transporte y de capa de aplicación?**

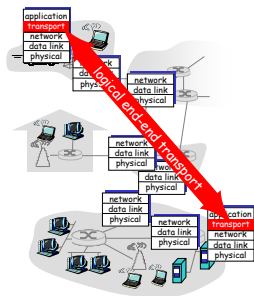
Analogía familiar:

- 6 niños enviando cartas a 6 niños (primos)
- procesos = niños
- Mensajes de aplicación = cartas (en sobres)
- *hosts* = casas
- protocolo de transporte = Ana y Guillermo
- Protocolo de capa de red = servicio de correo postal

Int. Redes de Computadores - Capa de Transporte 3-5

Protocolos de la capa de Transporte de Internet

- confiable, entrega en orden: TCP
 - control de congestión
 - control de flujo
 - establecimiento de la conexión
- no confiable, entrega no ordenada: UDP
 - En relación al "mejor esfuerzo" de IP, poco aporta
- servicios no disponibles:
 - garantías de retardo
 - garantías de ancho de banda



Int. Redes de Computadores - Capa de Transporte 3-6

Capítulo 3: agenda

- 3.1 Servicios de la capa de transporte
- 3.2 Multiplexación y demultiplexación
- 3.3 Transporte no orientado a conexión: UDP
- 3.4 Principios de la transferencia de datos confiable
- 3.5 Transporte orientado a conexión: TCP
 - estructura del segmento
 - Transferencia de datos confiable
 - control de flujo
 - Gestión de la conexión
- 3.6 Principios del control de congestión
- 3.7 Control de congestión de TCP

Int. Redes de Computadores - Capa de Transporte 3-7

Multiplexación/Demultiplexación

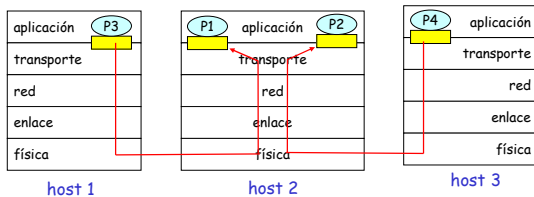
Demultiplexación en el *host* receptor:

Entregando los segmentos recibidos al *socket* correcto

Multiplexación en el *host* transmisor:

Recolectando datos de múltiples *sockets*, incorporando *headers* (después utilizados en la demultiplexación)

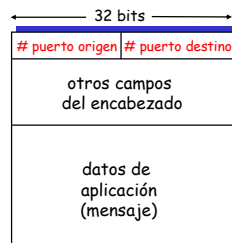
■ = socket ○ = proceso



Int. Redes de Computadores - Capa de Transporte 3-8

¿Cómo trabaja la demultiplexación?

- Un *host* recibe datagramas IP
 - cada datagrama tiene dirección IP de origen, dirección IP de destino
 - cada datagrama lleva 1 segmento de la capa de transporte
 - cada segmento tiene número de puertos de origen y destino
- los *host* utilizan las direcciones IP y los números de puertos para enviar el segmento al *socket* apropiado
- Nro. de puerto: **TSAP** (*Transport Service Access Point*)



Formato del segmento TCP/UDP

Int. Redes de Computadores - Capa de Transporte 3-9

Un poco más sobre puertos

□ IANA (Internet Assigned Numbers Authority)

- Se agrupan en 3 rangos
- *Well-known port numbers*: 0 - 1023
 - aplicaciones de servidor; en general ejecutados por usuarios tipo "root". Se deberían registrar
 - 53: DNS, 80: www HTTP, 20 y 21: ftp, 22: ssh, 161 y 162: SNMP, imaps: 993, imap: 143, nntps: 563
- *Registered ports*: 1024 - 49151
 - aplicaciones de servidor, usuarios comunes. Se deberían registrar
 - 1812, 1813: radius
- *Dynamic and/or Private ports*: 49152 - 65535
 - Para uso temporario. No se deben registrar
- <http://www.iana.org/assignments/port-numbers>

Int. Redes de Computadores - Capa de Transporte 3-10

Demultiplexación no orientada a conexión

□ Crear *sockets* con números de puerto:

```
DatagramSocket e1Socket1 = new
DatagramSocket();
```

Asigna un nro. de puerto libre entre 1024 y 65535

```
DatagramSocket e1Socket2 = new
DatagramSocket(12535);
```

□ UDP *socket* completamente identificado por la dupla:

(dirección IP destino, número puerto destino)

□ Cuando el *host* recibe el segmento UDP:

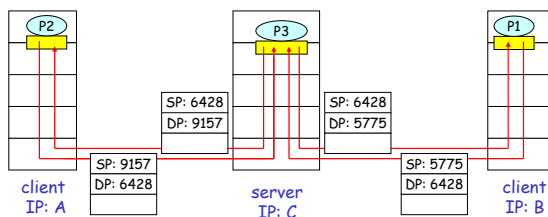
- Chequea el número de puerto destino en el segmento
- Dirige el segmento UDP al *socket* con dicho número de puerto

□ Datagramas IP con diferentes direcciones IP **origen** y/o números de puerto **origen** dirigidos al **mismo socket**

Int. Redes de Computadores - Capa de Transporte 3-11

demux no orientada a conexión (continuación)

```
DatagramSocket serverSocket = new DatagramSocket(6428);
```



SP: Source Port
DP: Destination Port

SP proporciona "dirección de retorno"

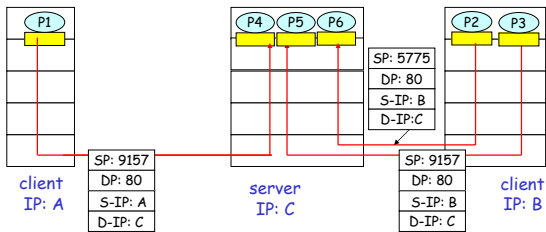
Int. Redes de Computadores - Capa de Transporte 3-12

Demux orientada a conexión

- *Socket* TCP identificado por una tupla de 4 elementos:
 - dirección IP origen
 - número puerto origen
 - dirección IP destino
 - número puerto destino
- El *host* destino utiliza los 4 valores para dirigir el segmento al *socket* apropiado
- El *host* servidor debería soportar varios *sockets* TCP simultáneos:
 - cada *socket* identificado por su propia tupla de 4 elementos
- Los servidores Web tienen diferentes *sockets* para cada cliente que se conecta
 - HTTP no persistente debe tener diferentes *sockets* para cada request

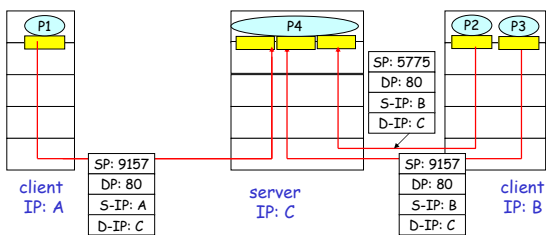
Int. Redes de Computadores - Capa de Transporte 3-13

Demux orientada a conexión (continuación)



Int. Redes de Computadores - Capa de Transporte 3-14

Demux orientada a conexión: Threaded Web Server



Int. Redes de Computadores - Capa de Transporte 3-15

Port scanning

- ❑ Es relativamente sencillo conocer qué aplicaciones están escuchando en qué puertos en un *end system* o en un conjunto de *end systems*
- ❑ Un programa para ello
 - Nmap (*Network Mapper*)
 - <http://insecure.org/nmap>
- ❑ Si detectamos algún *host* corriendo determinada aplicación de la que conocemos alguna vulnerabilidad, esto puede ser el punto de partida de un ataque

Int. Redes de Computadores - Capa de Transporte 3-16

Capítulo 3: agenda

- ❑ 3.1 Servicios de la capa de transporte
- ❑ 3.2 Multiplexación y demultiplexación
- ❑ 3.3 Transporte no orientado a conexión: UDP
- ❑ 3.4 Principios de la transferencia de datos confiable
- ❑ 3.5 Transporte orientado a conexión: TCP
 - estructura del segmento
 - Transferencia de datos confiable
 - control de flujo
 - Gestión de la conexión
- ❑ 3.6 Principios del control de congestión
- ❑ 3.7 Control de congestión de TCP

Int. Redes de Computadores - Capa de Transporte 3-17

UDP: User Datagram Protocol [RFC 768]

- ❑ Protocolo de transporte de Internet "esquelético"
- ❑ Servicio "*best effort*"; los segmentos UDP podrían:
 - perderse
 - entregarse fuera de orden a la aplicación
- ❑ **no orientado a conexión:**
 - no hay *handshaking* UDP entre el *sender* y el *receiver*
 - cada segmento UDP es manejado independientemente de los otros

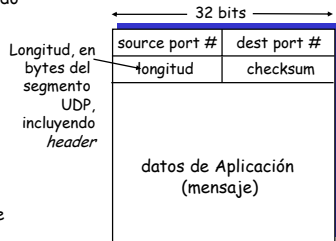
¿Por qué existe UDP?

- ❑ no hay establecimiento de conexión (lo cual puede agregar retardo)
- ❑ simple: no hay estado de conexión ni en el *sender* ni en el *receiver*
- ❑ encabezado del segmento pequeño
- ❑ no hay control de congestión

Int. Redes de Computadores - Capa de Transporte 3-18

UDP: más

- frecuentemente utilizado para aplicaciones de *streaming* multimedia
 - tolerante a las pérdidas
 - sensible a la *rate*
- otros usos de UDP
 - DNS
 - SNMP
 - NTP
- transferencia confiable sobre UDP: agregar confiabilidad en la capa de Aplicación



Formato del segmento UDP

El checksum de UDP

Objetivo: detectar errores (p.e., bits cambiados) en el segmento transmitido

Sender:

- trata el contenido de cada segmento como una secuencia de enteros de 16-bit
- *checksum*: complemento a 1 de la suma de los contenidos del segmento
- El transmisor pone el valor del *checksum* UDP en el campo *checksum*

Receiver:

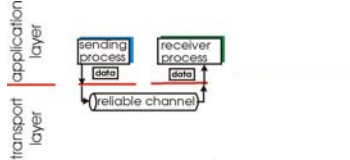
- calcula el *checksum* del segmento recibido
- Chequea si el *checksum* calculado es igual al valor del campo *checksum*.
 - NO - error detectado
 - SI - no se ha detectado error. *Pero, no obstante, ¿podrían haber errores?*

Capítulo 3: agenda

- 3.1 Servicios de la capa de transporte
- 3.2 Multiplexación y demultiplexación
- 3.3 Transporte no orientado a conexión: UDP
- 3.4 Principios de la transferencia de datos confiable
- 3.5 Transporte orientado a conexión: TCP
 - estructura del segmento
 - Transferencia de datos confiable
 - control de flujo
 - Gestión de la conexión
- 3.6 Principios del control de congestión
- 3.7 Control de congestión de TCP

Principios de la transferencia de datos confiable

- importante en app., transport, link layers
- i En la lista de los temas más importantes del *networking*!



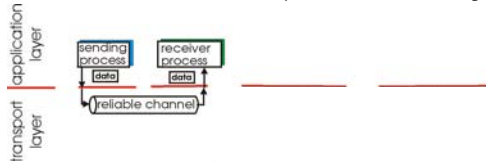
(a) provided service

- las características del canal no confiable determinan la complejidad del protocolo transferencia de datos confiable (rdt: *reliable data transfer*)

Int. Redes de Computadores - Capa de Transporte 3-22

Principios de la transferencia de datos confiable

- importante en app., transport, link layers
- i En la lista de los 10 temas más importantes del *networking*!



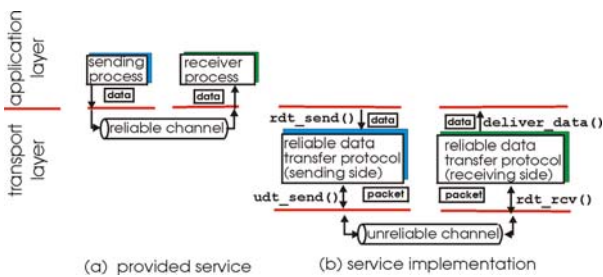
(a) provided service

(b) service implementation

- las características del canal no confiable determinan la complejidad del protocolo transferencia de datos confiable (rdt: *reliable data transfer*)

Int. Redes de Computadores - Capa de Transporte 3-23

Principios de la transferencia de datos confiable



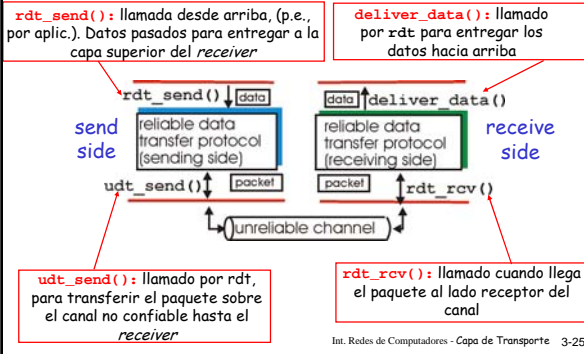
(a) provided service

(b) service implementation

udt: *unreliable data transfer*

Int. Redes de Computadores - Capa de Transporte 3-24

Transferencia de datos confiable: comenzando



Transferencia de datos confiable: comenzando

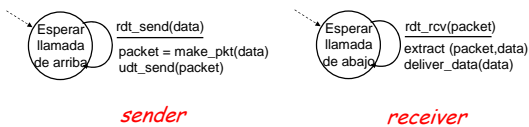
- Desarrollo incremental del transmisor y del receptor de un protocolo de transferencia confiable de datos (rdt)
- consideramos solamente transferencia unidireccional de datos
 - ¡ Pero la información de control y de datos fluye en ambas direcciones !
- para especificar el emisor y el receptor utilizamos una máquina de estados finita (FSM)

evento que causa la transición de estado
acciones tomadas en la transición de estado



rdt1.0: transferencia confiable sobre un canal confiable

- El canal subyacente es perfectamente confiable
 - Sin errores en bits
 - Sin pérdida de paquetes
- FSMs separadas para sender y receiver.
 - El emisor envía datos dentro del canal subyacente
 - El receptor lee datos del canal subyacente



Protocolos ARQ

- Implica la presencia de reconocimientos positivos ("Correcto") y negativos ("Repita, por favor")
- Los reconocimientos son mensajes de control a través de los cuales el receptor realimenta al emisor respecto a las PDU que recibe.
- Los protocolos de transferencia de datos que implementan esto, se conocen como protocolos ARQ
 - Automatic Repeat reQuest

Int. Redes de Computadores - Capa de Transporte 3-28

Protocolos ARQ

- ¿Qué capacidades precisan estos protocolos?
 - Detección de errores
 - Realimentación desde el receptor
 - Retransmisión
- Volveremos sobre estos protocolos en las próximas clases

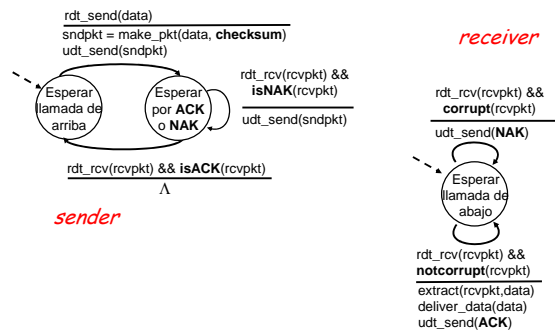
Int. Redes de Computadores - Capa de Transporte 3-29

rdt2.0: canal con errores en bits

- no hay pérdida de paquetes
- el canal subyacente puede modificar bits en el paquete
 - checksum para detectar errores en bits
- ¿Cómo nos recuperamos de los errores?
 - **ACKnowledgements (ACKs)**: el receptor explícitamente le dice al transmisor que el paquete se ha recibido OK
 - **Negative ACKnowledgements (NAKs o NACKs)**: el receptor explícitamente le dice al transmisor que el paquete tiene errores
 - El transmisor retransmite el paquete ante la recepción de un NAK
- nuevo mecanismo en rdt2.0 (respecto a rdt1.0):
 - detección de error
 - realimentación desde el receptor (receiver → sender): mensajes de control (ACK, NAK)

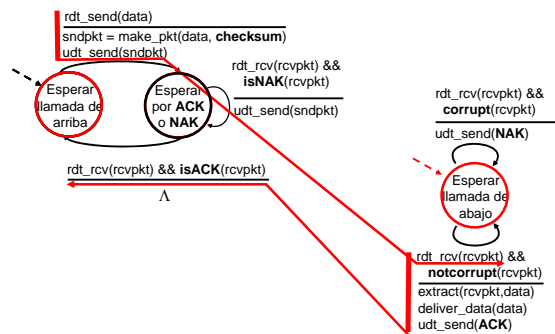
Int. Redes de Computadores - Capa de Transporte 3-30

rdt2.0: Especificación FSM



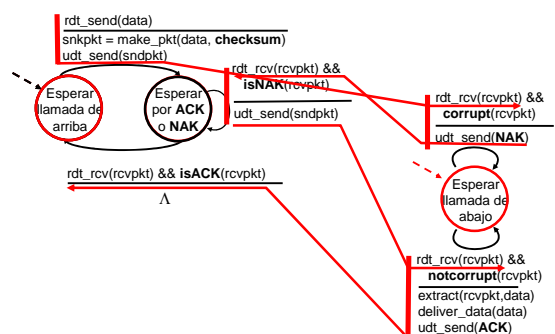
Int. Redes de Computadores - Capa de Transporte 3-31

rdt2.0: operación sin errores



Int. Redes de Computadores - Capa de Transporte 3-32

rdt2.0: escenario de error



Int. Redes de Computadores - Capa de Transporte 3-33

rdt2.1: discusión

Emisor:

- número de secuencia agregado al paquete
- ¿dos números de seq. (0,1) es suficiente?
- debe chequear si el ACK/NAK recibido está corrupto

Receptor:

- debe chequear si el paquete recibido está duplicado
- nota: el receptor no puede saber si el último ACK/NAK se recibió OK en el transmisor

Int. Redes de Computadores - Capa de Transporte 3-37

rdt2.2: un protocolo libre de NAK

- recordemos que los paquetes no se pierden (todavía)
- las mismas funcionalidades que rdt2.1, pero utilizando solamente ACKs
- en lugar de NAK, el receptor envía ACK para el último paquete recibido OK
 - el receptor debe incluir explícitamente el número de secuencia del paquete que está ACKed
- ACK duplicado (dos ACKs para el mismo paquete) en el emisor resulta en la misma acción que el NAK: *retransmitir el paquete actual*

Int. Redes de Computadores - Capa de Transporte 3-38
