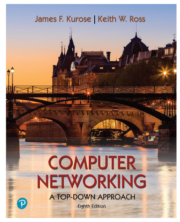
Capítulo 3
Capa de transporte

Una nota sobre el uso de estas diapositivas de PowerPoint:
Nosotros'Estamos poniendo estas diapositivas a disposición de todos (profesores, estudiantes, lectores). Están en forma de PowerPoint para que pueda ver las animaciones; y puede agregar, modificar y eliminar diapositivas (incluida esta) y contenido de diapositivas para satisfacer sus necesidades. Obviamente representan un *lote* de trabajo de nuestra parte. A cambio de su uso, solo pedimos lo siguiente:

- Si usa estas diapositivas (por ejemplo, en una clase) que menciona su fuente (después de todo, nosotros ¡Me gustaría que la gente usara nuestro libro!)
- Si publica diapositivas en un sitio www, debe tener en cuenta que están adaptadas (o tal vez sean idénticas) a nuestras diapositivas, y tenga en cuenta nuestros derechos de autor de este material.

Para obtener un historial de revisión, consulte la nota de diapositiva de esta página.

Gracias y disfruta! JFK / KWR



Redes de computadoras: un enfoque de arriba hacia abajo
8th edición
Jim Kurose, Keith Ross
Pearson, 2020

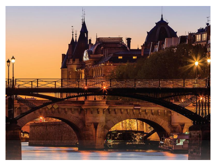
Capa de transporte: descripción general

Nuestro objetivo:

- comprender los principios detrás de los servicios de la capa de transporte:
  - multiplexación, demultiplexación
  - transferencia de datos confiable
  - control de flujo
  - control de congestión
- aprenda sobre los protocolos de la capa de transporte de Internet:
  - UDP: transporte sin conexión
  - TCP: transporte confiable orientado a la conexión
  - Control de congestión TCP

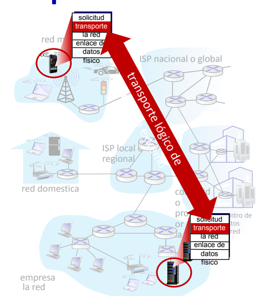
Capa de transporte: hoja de ruta

- Servicios de la capa de transporte
- Multiplexación y demultiplexación
- Transporte sin conexión: UDP
- Principios de la transferencia de datos confiable
- Transporte orientado a la conexión: TCP
- Principios del control de la congestión
- Control de congestión TCP
- Evolución de la funcionalidad de la



Servicios y protocolos de transporte

- proveer *comunicación lógica* entre procesos de aplicación que se ejecutan en diferentes hosts
- acciones de los protocolos de transporte en los sistemas finales:
  - remitente: divide los mensajes de la aplicación en *segmentos*, pasa a la capa de red
  - receptor: vuelve a ensamblar segmentos en mensajes, pasa a la capa de aplicación
- dos protocolos de transporte disponibles para aplicaciones de Internet
  - TCP, UDP



Transporte frente a servicios y protocolos de capa de red

analogía del hogar:

12 niños en Ann 's casa enviando cartas a 12 niños en la casa de Bill:

- hosts = casas
- procesos = niños
- mensajes de la aplicación = cartas en sobres

servicio postal

Transporte frente a servicios y protocolos de capa de red

- capa de red: comunicación lógica entre *Hospedadores*
- capa de transporte: comunicación lógica entre *procesos*
  - confía en los servicios de la capa de red y los mejora

analogía del hogar:

12 niños en Ann 's casa enviando cartas a 12 niños en la casa de Bill:

- hosts = casas
- procesos = niños
- mensajes de la aplicación = cartas en sobres

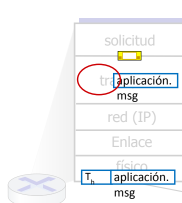
servicio postal

Acciones de la capa de transporte

Remitente:

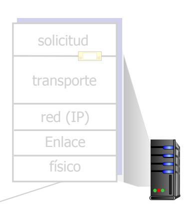
- se pasa un mensaje de capa de aplicación
- determina los valores de los campos de encabezado de segmento
- crea segmento
- pasa segmento a IP

Acciones de la capa de transporte



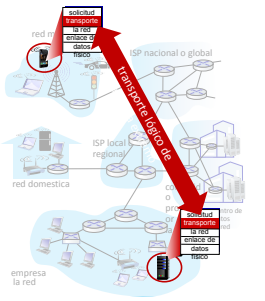
Receptor:

- recibe segmento de IP
- comprueba los valores del encabezado
- extrae el mensaje de la capa de aplicación
- demultiplexa el mensaje hasta la aplicación a través del conector



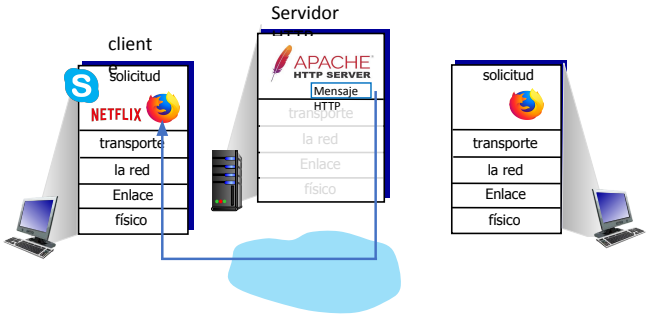
Dos protocolos principales de transporte de Internet

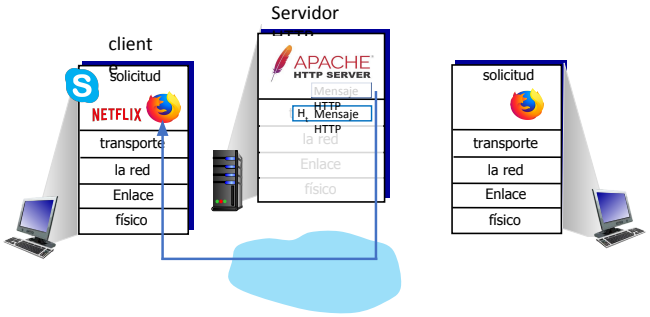
- **TCP:** Protocolo de Control de Transmisión
  - entrega confiable y en orden
  - control de congestión
  - control de flujo
  - configuración de la conexión
- **UDP:** Protocolo de datagramas de usuario
  - entrega no confiable y desordenada
  - extensión sencilla de "IP de mejor esfuerzo
- servicios no disponibles:
  - garantías de demora

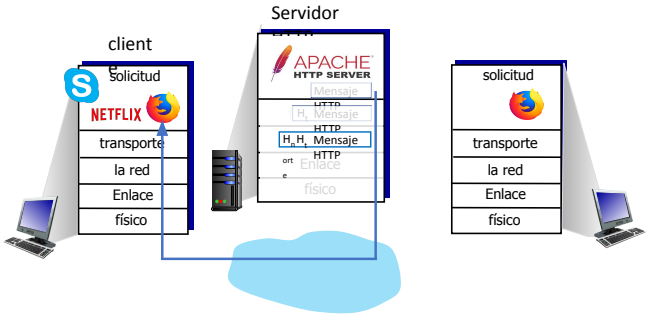


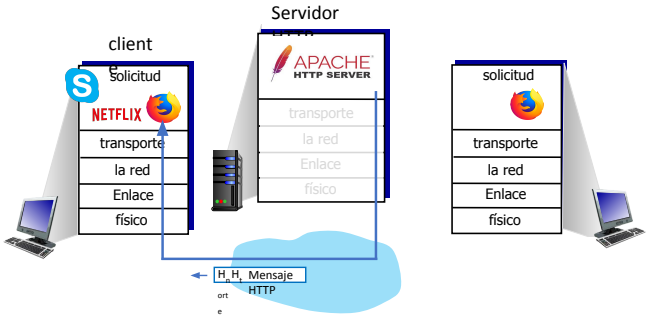
Capítulo 3: hoja de ruta

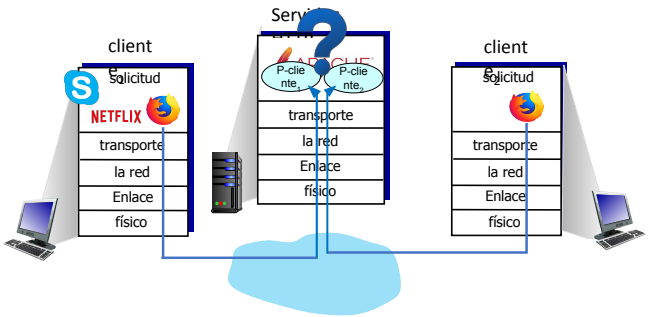
- Servicios de la capa de transporte
- Multiplexación y demultiplexación
- Transporte sin conexión: UDP
- Principios de la transferencia de datos confiable
- Transporte orientado a la conexión: TCP
- Principios del control de la congestión
- Control de congestión TCP
- Evolución de la funcionalidad de la







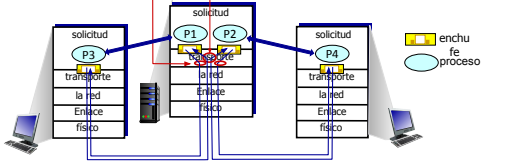




Multiplexación / demultiplexación

multiplexación en el remitente: manejar datos de múltiples sockets, agregue el encabezado de transporte (luego usado para demultiplexar)

demultiplexación en el receptor: usar la información del encabezado para entregar segmentos recibidos para corregir enchufe



¿Cómo demultiplexar worcos

- el host recibe datagramas de IP
  - cada datagrama tiene una dirección IP de origen, una dirección IP de destino
  - cada datagrama lleva un segmento de la capa de transporte
  - cada segmento tiene un número de puerto de origen y destino
- usos del anfitrión *Direcciones IP y números de puerto* para dirigir el segmento al enchufe apropiado

Formato de segmento TCP / UDP

Demultiplexación sin conexión

Recordar:

- al crear socket, debe especificar *anfitrión-local* Puerto #:

```
DatagramSocket mySocket1 = nuevo
DatagramSocket (5534);
```

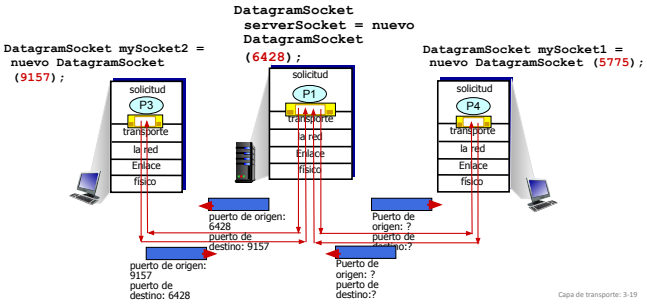
- al crear un datagrama para enviarlo al socket UDP, debe especificar
  - Dirección IP de destino
  - Puerto de destino #

al recibir el host recibe *UDP* segmento:

- comprueba el número de puerto de destino en el segmento
- dirige el segmento UDP al socket con ese puerto #

Datagramas IP / UDP con *mismo dest. Puerto #,* pero se dirigirán a diferentes direcciones IP de origen y / o números de puerto de origen *mismo enchufe* al recibir el anfitrión

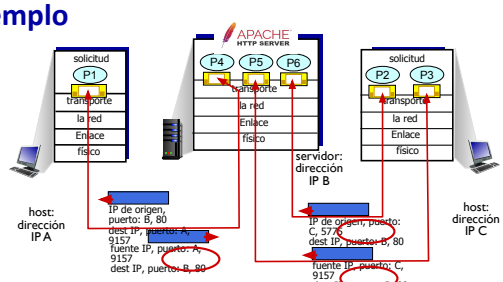
Demultiplexación sin conexión: un ejemplo



Demultiplexación orientada a la conexión

- Socket TCP identificado por 4-tupla:
  - Dirección IP origen
  - número de puerto de origen
  - dest dirección IP
  - dest número de puerto
- demux: usos del receptor *los cuatro valores (4 tuplas)* para dirigir el segmento al enchufe apropiado
- el servidor puede admitir muchos sockets TCP simultáneos:
  - cada zócalo identificado por su propia tupla de 4
  - cada socket asociado con un cliente de conexión diferente

Demultiplexación orientada a la conexión: ejemplo



Tres segmentos, todos destinados a la dirección IP: B,
dest puerto: 80 se demultiplexan a *diferente*

Resumen

- Multiplexación, demultiplexación: basado en segmentos, valores de campo de encabezado de datagrama
- **UDP:** demultiplexación utilizando el número de puerto de destino (solo)
- **TCP:** demultiplexación utilizando 4 tuplas: direcciones IP de origen y destino, y números de puerto
- La multiplexación / demultiplexación ocurre en *todas* capas

Capítulo 3: hoja de ruta

- Servicios de la capa de transporte
- Multiplexación y demultiplexación
- Transporte sin conexión: UDP
- Principios de la transferencia de datos confiable
- Transporte orientado a la conexión: TCP
- Principios del control de la congestión
- Control de congestión TCP
- Evolución de la funcionalidad de la

UDP: Protocolo de datagramas de usuario

- Protocolo de transporte de Internet "sencillo", "básico"
- Servicio de "mejor esfuerzo", los segmentos UDP pueden ser:
  - perdió
  - entregado fuera de servicio a la aplicación
- *sin conexión:*
  - sin protocolo de enlace entre el remitente y el receptor UDP
  - cada segmento UDP se maneja independientemente de los demás

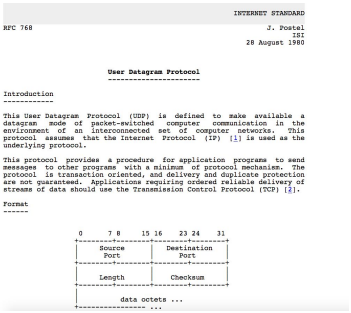
Why ¿hay un UDP?

- sin establecimiento de conexión (lo que puede agregar demora RTT)
- simple: sin estado de conexión en el remitente, el receptor
- tamaño de encabezado pequeño
- sin control de congestión
  - ¡UDP puede disparar tan rápido como se desee!
  - puede funcionar frente a la congestión

UDP: Protocolo de datagramas de usuario

- Uso de UDP:
  - transmisión de aplicaciones multimedia (tolerantes a pérdidas, sensibles a la velocidad)
  - DNS
  - SNMP
  - HTTP / 3
- si se necesita una transferencia confiable a través de UDP (por ejemplo, HTTP / 3):
  - agregue la confiabilidad necesaria en la capa de aplicación
  - agregar control de congestión en la capa de aplicación

UDP: Protocolo de datagramas de usuario [RFC 768]



INTERNET STANDARD

RFC 768 — J. Postel, ISI, 28 August 1980

User Datagram Protocol

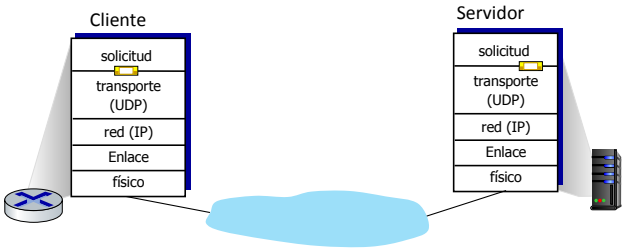
Introduction

This User Datagram Protocol (UDP) is defined to make available a datagram mode of packet-switched computer communication in the environment of an interconnected set of computer networks. This protocol assumes that the Internet Protocol (IP) [1] is used as the underlying protocol.

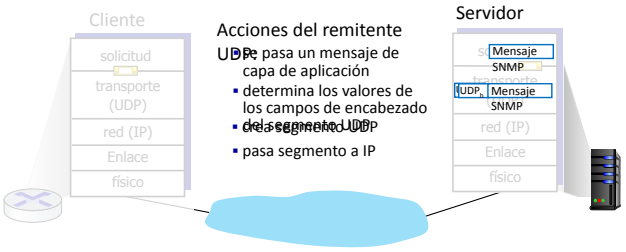
This protocol provides a procedure for application programs to send messages to other programs with a minimum of protocol mechanism. The protocol is transaction oriented, and delivery and duplicate protection are not guaranteed. Applications requiring ordered reliable delivery of streams of data should use the Transmission Control Protocol (TCP) [2].

Format

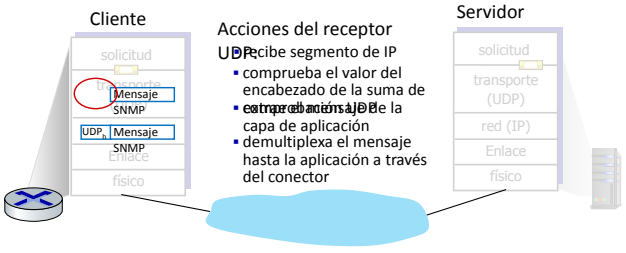
UDP: acciones de la capa de transporte



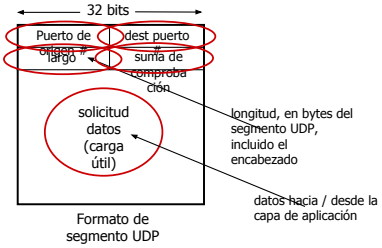
UDP: acciones de la capa de transporte



UDP: acciones de la capa de transporte

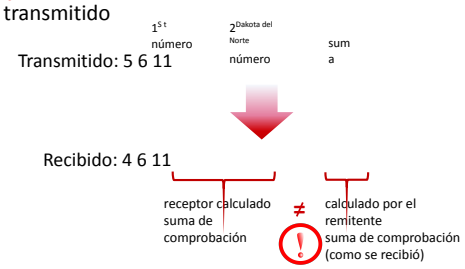


Segmento UDP header



Suma de comprobación UDP

Objetivo: detectar errores*es decir,* bits invertidos) en el segmento transmitido



Suma de comprobación UDP

Objetivo: detectar errores*es decir,* bits invertidos) en el segmento transmitido

remitente:

- tratar el contenido del segmento UDP (incluidos los campos de encabezado UDP y las direcciones IP) como secuencia de enteros de 16 bits
- suma de comprobación: adición (uno's suma del complemento) del contenido del segmento
- valor de suma de comprobación puesto en el

receptor:

- calcular la suma de comprobación del segmento recibido
- compruebe si la suma de comprobación calculada es igual al valor del campo de suma de comprobación:
  - Distinto: error detectado
  - Igual: no se detectó ningún error. *¿Pero tal vez errores de todos modos?* Más tarde ….

Suma de comprobación de Internet: un ejemplo

ejemplo: sume 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
envolver alrededor	1 1 0 1 1 1 0 1 1 1 0 1 1 1 0 1 1
suma	1 0 1 1 1 0 1 1 1 0 1 1 1 1 0 0
suma de comprobación	0 1 0 0 0 1 0 0 0 1 0 0 0 0 1 1

Nota: al sumar números, se debe agregar un arrastre del bit más significativo al resultado

* Consulte los ejercicios interactivos en línea para ver más ejemplos: http: //gaia.cs.umass.edu/kurose_ross/interactivo/

Suma de comprobación de Internet: ¡protección débil!

ejemplo: sume 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
envolver alrededor	1 1 0 1 1 1 0 1 1 1 0 1 1 1 0 1 1
suma	1 0 1 1 1 0 1 1 1 0 1 1 1 1 0 0
suma de comprobación	0 1 0 0 0 1 0 0 0 1 0 0 0 0 1 1

0 1
1 0

A pesar de que los números han cambiado (cambio de bits), *No* cambio en la suma de comprobación!

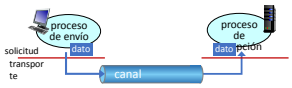
Resumen: UDP

- Protocolo "sin lujos":
  - los segmentos pueden perderse, entregarse fuera de servicio
  - servicio de mejor esfuerzo: "enviar y esperar lo mejor"
- UDP tiene sus ventajas:
  - no se necesita configuración / protocolo de enlace (no se incurre en RTT)
  - puede funcionar cuando el servicio de red está comprometido
  - ayuda con la confiabilidad (suma de verificación)
- construir funcionalidad adicional sobre UDP en la capa de aplicación (por ejemplo, HTTP / 3)

Capítulo 3: hoja de ruta

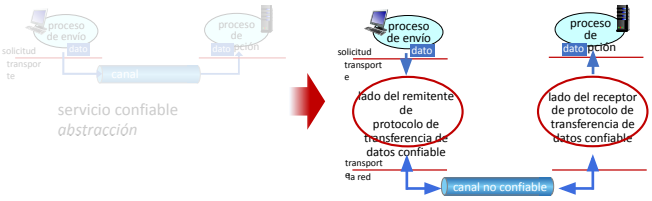
- Servicios de la capa de transporte
- Multiplexación y demultiplexación
- Transporte sin conexión: UDP
- Principios de la transferencia de datos confiable
- Transporte orientado a la conexión: TCP
- Principios del control de la congestión
- Control de congestión TCP
- Evolución de la funcionalidad de la

Principios de confiabilidad Data ttransferir



servicio confiable
abstracción

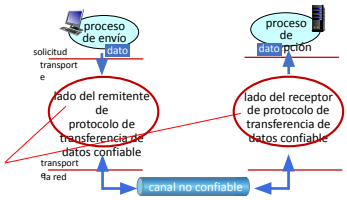
Principios de confiabilidad Data ttransferir



servicio confiable *implementación*

Principios de confiabilidad Data ttransferir

La complejidad de un protocolo de transferencia de datos confiable dependerá (en gran medida) de las características del canal no confiable (¿perder, corromper, reordenar datos?)

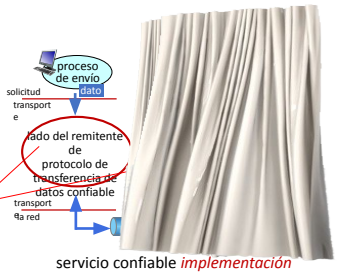


servicio confiable *implementación*

Principios de confiabilidad Data ttransferir

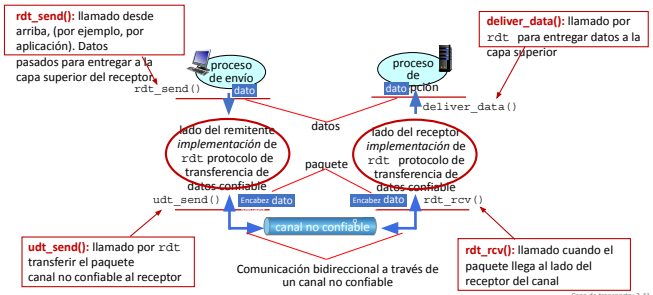
Remitente, receptor hacer *no* conocer el "estado" de cada uno, por ejemplo, ¿se recibió un mensaje?

- a menos que se comunique a través de un mensaje



servicio confiable *implementación*

Protocolo de transferencia de datos confiable (rdt): interfaces



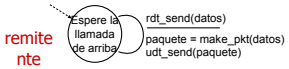
Transferencia de datos confiable: primeros pasos

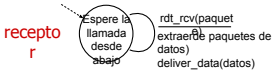
Lo haremos:

- Desarrollar incrementalmente los lados del remitente y del receptor reliable Data tprotocolo de transferenciardt)
- considere solo la transferencia de datos unidireccional
  - ¡pero la información de control fluirá en ambas direcciones!
- utilizar máquinas de estado finito (FSM) para especificar el remitente, el receptor

rdt1.0: transferencia confiable a través de un canal confiable

- canal subyacente perfectamente confiable
  - sin errores de bits
  - sin pérdida de paquetes
- *separar* FSM para emisor, receptor:
  - el remitente envía datos al canal subyacente
  - el receptor lee datos del canal subyacente





rdt2.0: canal con errores de bit

- el canal subyacente puede invertir bits en el paquete
  - suma de comprobación (por ejemplo, suma de comprobación de Internet) para detectar errores de bits
- *la* pregunta: ¿cómo recuperarse de errores?

¿Cómo se recuperan los humanos de "errores " durante la conversación?

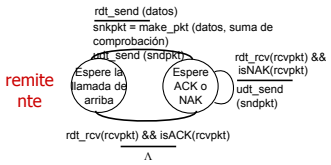
rdt2.0: canal con errores de bit

- el canal subyacente puede invertir bits en el paquete
  - suma de comprobación para detectar errores de bits
- *la* pregunta: ¿cómo recuperarse de errores?
  - *agradecimientos (ACK):* El receptor le dice explícitamente al remitente que el paquete recibió OK
  - *reconocimientos negativos (NAK):* el receptor le dice explícitamente al remitente que el paquete tenía errores
  - remitente *retransmite* paquete al recibir NAK

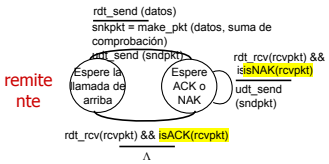
detente y espere

el remitente envía un paquete, luego espera la respuesta del receptor

rdt2.0: especificaciones FSM



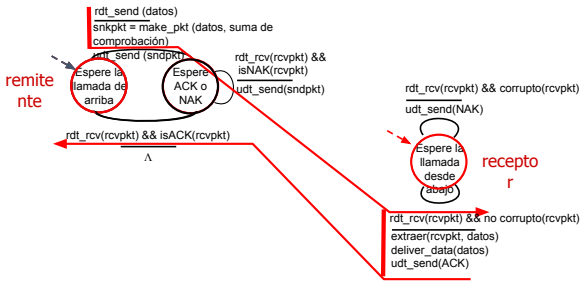
rdt2.0: especificación FSM



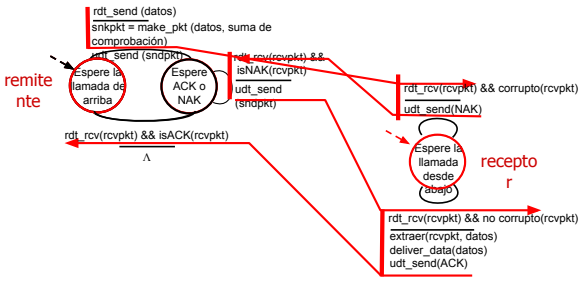
Nota: El "estado" del receptor (¿el receptor recibió mi mensaje correctamente?) No es conocido por el remitente a menos que se comunique de alguna manera del receptor al remitente.

- ¡por eso necesitamos un protocolo!

rdt2.0: operación sin errores



rdt2.0: escenario de paquete dañado



¡rdt2.0 tiene un defecto fatal!

¿Qué sucede si ACK / NAK está dañado??

- el remitente no¡No sé lo que pasó en el receptor!
- lata't solo retransmitir: posible duplicado

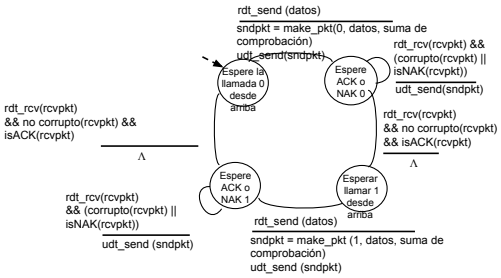
manejo de duplicados:

- el remitente retransmite el paquete actual si ACK / NAK está dañado
- el remitente agrega *secuencia de números* a cada paquete
- el receptor descarta (not entregar) paquete duplicado

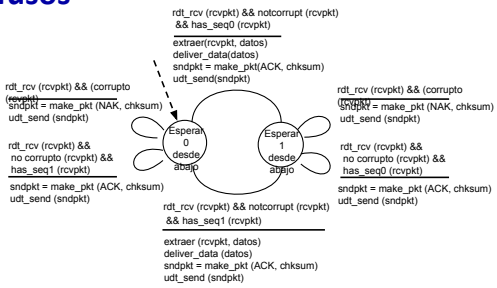
detente y espera

el remitente envía un paquete, luego espera la respuesta del receptor

rdt2.1: remitente, manejo de ACK / NAK confusos



rdt2.1: receptor, manejo de ACK / NAK confusos



rdt2.1: discusión

remitente:

- seq # agregado al paquete
- dos seq. #s (0,1) será suficiente. ¿Por qué?
- debe verificar si el ACK / NAK recibido está dañado
- el doble de estados
  - el estado debe "recuerde "si el paquete" esperado "debe tener un número de secuencia de 0 o 1

receptor:

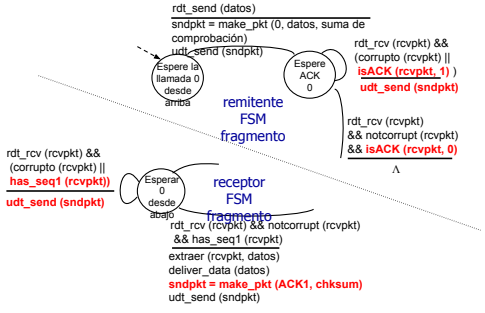
- debe verificar si el paquete recibido está duplicado
  - estado indica si se espera 0 o 1 pkt seq #
- nota: el receptor puede *no* saber si su último ACK / NAK recibió OK en el remitente

rdt2.2: un protocolo libre de NAK

- Misma funcionalidad que rdt2.1, usando solo ACK
- en lugar de NAK, el receptor envía ACK para el último paquete recibido OK
  - el receptor debe *explícitamente* incluir el número de secuencia del paquete que es ACKED
- ACK duplicado en el remitente da como resultado la misma acción que NAK: *retransmitir el paquete actual*

Como veremos, TCP usa este enfoque para estar libre de NAK

rdt2.2: fragmentos de emisor, receptor



rdt3.0: canales con errores *y* pérdida

Supuesto de nuevo canal: El canal subyacente también puede *perder* paquetes (datos, ACK)

- La suma de comprobación, los números de secuencia, los ACK, las retransmisiones serán de ayuda ... pero no lo suficiente

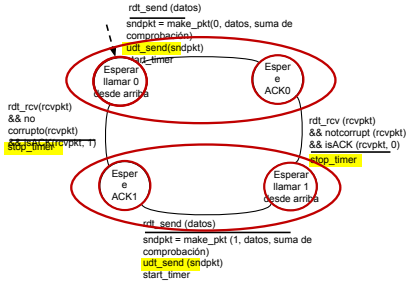
Q: Como hacer *humanos* manejar las palabras perdidas de remitente a receptor en una conversación?

rdt3.0: canales con errores y pérdida

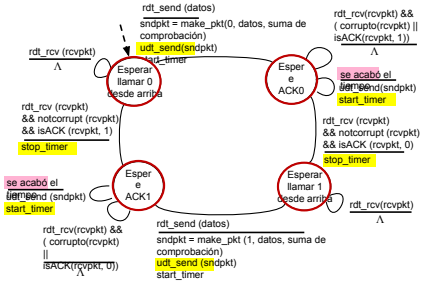
Acercarse: el remitente espera Cantidad de tiempo "razonable" para ACK

- retransmite si no se recibe ACK en este tiempo
- si pkt (o ACK) solo se retrasa (no se pierde):
  - la retransmisión será duplicada, pero seq #¡s ya maneja esto!
  - El receptor debe especificar el número de secuencia del paquete que se ACKED
- use el temporizador de cuenta regresiva para interrumpir después de una cantidad de tiempo "razonable"

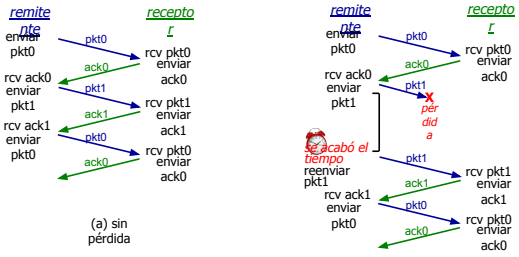
remitente rdt3.0



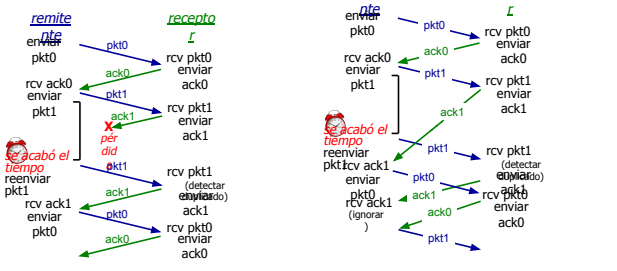
remitente rdt3.0



rdt3.0 en acción



rdt3.0 en acción

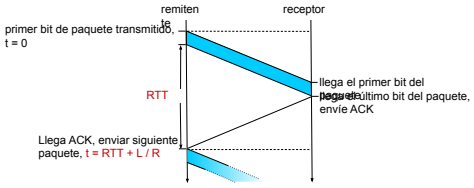


Rendimiento de rdt3.0 (parar y esperar)

- $U_{remitente}$: *utilización* - fracción de tiempo que el remitente está ocupado enviando
- ejemplo: enlace de 1 Gbps, 15 Sraapuntalar. retraso, paquete de 8000 bits
  - tyo me para transmitir paquetes al canal:

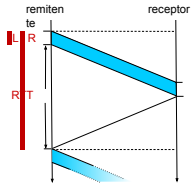
$$D_{trans} = \frac{L}{R} = \frac{8000\ bits}{10^9\ bits/seg} = 8\ microsegundos$$

rdt3.0: operación de parada y espera



rdt3.0: operación de parada y espera

$$U_{remite\ nte} = \frac{L/R}{RTT + L/R} = \frac{.008}{30.008} = 0{,}00027$$

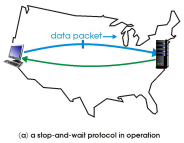


- rdt ¡El rendimiento del protocolo 3.0 apesta!
- El protocolo limita el rendimiento de la infraestructura subyacente (canal)

rdt3.0: operación de protocolos canalizados

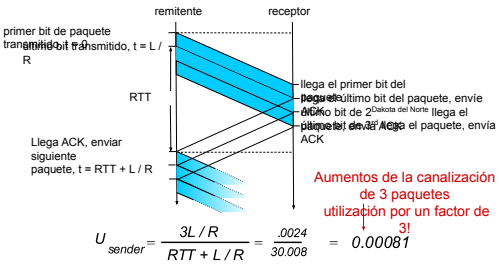
canalización: el remitente permite múltiples, "en vuelo ", paquetes aún por confirmar

- se debe aumentar el rango de números de secuencia
- almacenamiento en búfer en el emisor y / o receptor



(a) a stop-and-wait protocol in operation

Canalización: mayor utilización



Aumentos de la canalización de 3 paquetes utilización por un factor de 3!

$$U_{sender} = \frac{3L / R}{RTT + L / R} = \frac{.0024}{30.008} = 0.00081$$

Go-Back-N: remitente

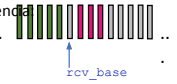
- remitente: "ventana" de hasta N, transmitida consecutivamente pero desembarazado paquetes
  - # de secuencia de k-bit en el encabezado del paquete

- *ACK acumulativo:* ACK (*norte*): ACK todos los paquetes hasta, incluido el número de secuencia *norte*
  - al recibir ACK (*norte*): move ventana adelante para comenzar en *n + 1*
- temporizador para el paquete en vuelo más antiguo
- *tiempo de espera (n):* retransmitir el paquete ny todos los paquetes de

Go-Back-N: receptor

- Solo ACK: envíe siempre ACK para el paquete recibido correctamente hasta el momento, con el más alto *en orden* seq #
  - puede generar ACK duplicados
  - solo necesito recordar `rcv_base`
- al recibir un paquete fuera de servicio:
  - puede descartar (don 't buffer) o buffer: una decisión de implementación
  - re-ACK pkt con el número de secuencia en orden más alto

Vista del receptor del espacio del número de secuencia

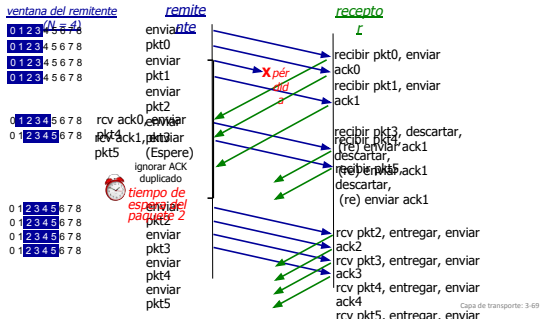


recibido y ACKED

Fuera de servicio: recibido pero no ACKED

No recibido

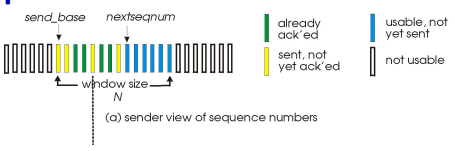
Go-Back-N en acción



Repetición selectiva

- receptor *individualmente* reconoce todos los paquetes recibidos correctamente
  - almacena los paquetes, según sea necesario, para una eventual entrega en orden a la capa superior
- El remitente agota el tiempo de espera / retransmite individualmente para desembarazado paquetes
  - el remitente mantiene el temporizador para cada desembarazado pit
- ventana del remitente
  - *norte* # de secuencia consecutivas
  - limita el número de secuencias de envío, desembarazado paquetes

Repetición selectiva: ventanas de emisor, receptor



Repetición selectiva: emisor y receptor

remite nte

datos de arriba:

- si el próximo número de secuencia disponible en la ventana, envíe el paquete

se acabó el tiempo(*norte*):

- reenviar paquete *norte*, reiniciar el temporizador

ACK (*norte*) en [sendbase, sendbase + N]:

- paquete de marca *norte* como recibido
- si n el más pequeño desembarazado paquete,

recepto r

paquete *norte* en [rcvbase, rcvbase + N-1]

- enviar ACK (*norte*)
- fuera de servicio: búfer
- en orden: entregar (también entregar paquetes en orden almacenados en búfer), ventana de avance al siguiente paquete aún no recibido

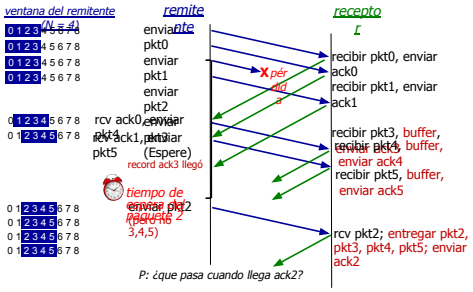
paquete *norte* en [rcvbase-N, rcvbase-1]

- ACK (*norte*)

de lo contrario:

- ignorar

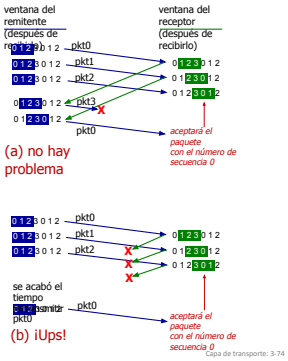
Repetición selectiva en acción



Repetición selectiva: ¡un dilema!

ejemplo:

- seq #s: 0, 1, 2, 3 (contando base 4)
- tamaño de la ventana = 3

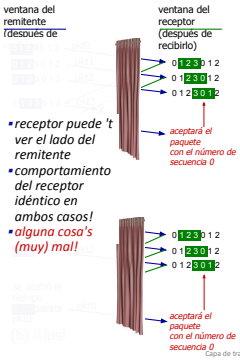


Repetición selectiva: ¡un dilema!

ejemplo:

- seq #s: 0, 1, 2, 3 (contando base 4)
- tamaño de la ventana = 3

Q: ¿Qué relación se necesita entre el tamaño del número de secuencia y el tamaño de la ventana para evitar problemas en el escenario (b)?



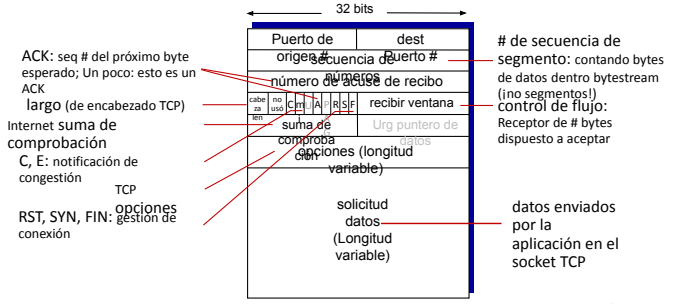
Capítulo 3: hoja de ruta

- Servicios de la capa de transporte
- Multiplexación y demultiplexación
- Transporte sin conexión: UDP
- Principios de la transferencia de datos confiable
- Transporte orientado a la conexión: TCP
  - estructura del segmento
  - transferencia de datos confiable
  - control de flujo
  - gestión de conexión
- Principios del control de la

TCP: descripción general RFC: 793,1122, 2018, 5681, 7323

- punto a punto:
  - un remitente, un receptor
- confiable, en orden *byte de vapor:*
  - No "límites del mensaje "
- datos full duplex:
  - flujo de datos bidireccional en la misma conexión
  - MSS: tamaño máximo de segmento
- ACK acumulativos
- canalización:
  - Tamaño de ventana establecido de control de flujo y congestión de TCP
- orientado a la conexión:
  - apretón de manos (intercambio de mensajes de control) inicializa el estado del remitente y del receptor antes del intercambio de datos
- flujo controlado:
  - el remitente no abrumará al

Estructura del segmento TCP



Números de secuencia TCP, ACK

Números de secuencia:

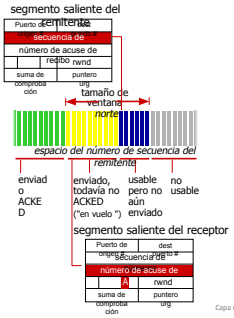
- flujo de bytes "número "del primer byte en los datos del segmento

Agradecimientos:

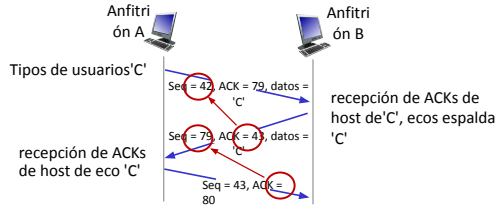
- seq # del próximo byte esperado del otro lado
- ACK acumulativo

Q: cómo el receptor maneja los segmentos desordenados

- A: La especificación TCP no't digo, - hasta el implementador



Números de secuencia TCP, ACK



escenario telnet simple

Tiempo de ida y vuelta de TCP, tiempo de espera

Q: ¿Cómo configurar el valor de tiempo de espera de TCP?

- más largo que RTT, ¡pero RTT varía!
- *demasiado corto:* tiempo de espera prematuro, retransmisiones innecesarias
- *demasiado largo:* reacción lenta a la pérdida de segmento

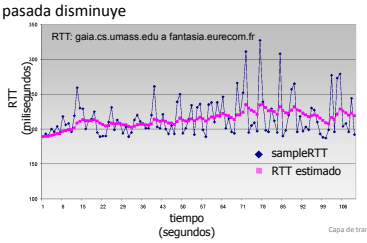
Q: ¿Cómo estimar RTT?

- `SampleRTT`: Medido tiempo desde la transmisión del segmento hasta la recepción ACK
  - ignorar retransmisiones
- `SampleRTT` variará, desea RTT estimado "mas suave"
  - promedio de varios *reciente* mediciones, no solo de corriente `SampleRTT`

Tiempo de ida y vuelta de TCP, tiempo de espera

```
RTT estimado = (1- α) *RTT estimado +
               α*SampleRTT
```

- miexponencial wmirado metrooving apromedio (EWMA)
- la influencia de la muestra pasada disminuye exponencialmente rápido
- valor típico: α = 0,125



Tiempo de ida y vuelta de TCP, tiempo de espera

- intervalo de tiempo de espera: **`RTT estimado`** más "margen de seguridad"
  - gran variación en **`RTT estimado`**: quiere un mayor margen de seguridad

```
TimeoutInterval = RTT estimado + 4 *DevRTT
```

RTT estimado — "margen de seguridad"

- **`DevRTT`**: EWMA de **`SampleRTT`** desviación de **`RTT estimado`**:

```
DevRTT = (1-β) *DevRTT + β* |MuestraRTT-EstimatedRTT|
```

(típicamente, β = 0,25)

* Consulte los ejercicios interactivos en línea para ver más ejemplos: http: //gaia.cs.umass.edu/kurose_ross/interactivo/

Remitente TCP (simplificado)

evento: datos recibidos de la aplicación

- crear segmento con seq #
- seq # es el número de flujo de bytes del primer byte de datos en el segmento
- iniciar el temporizador si aún no está funcionando
  - piensa en el temporizador como si fuera el más viejo desembarazado segmento
  - intervalo de caducidad: **`TimeOutInterval`**

evento: tiempo de espera

- retransmitir segmento que provocó el tiempo de espera
- reiniciar el temporizador

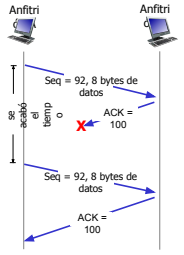
evento: ACK recibido

- si ACK reconoce previamente desembarazado segmentos
  - actualizar lo que se sabe que es ACKED
  - iniciar el temporizador si aún quedan desembarazado segmentos

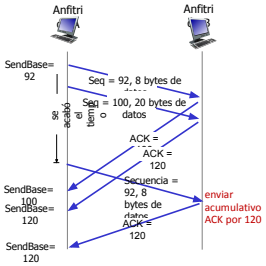
Receptor TCP: generación de ACK [RFC 5681]

Evento en el receptor	*Acción del receptor TCP*
los datos hasta	enviar ACK
llegada del segmento en orden	ACK, ACKing ambos segmentos en
el segmento tiene ACK	enviar inmediatamente *ACK duplicado.*
servicio secuencia superior a la	enviar ACK inmediato, siempre que

TCP: escenarios de retransmisión

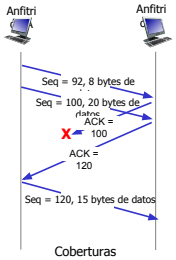


escenario ACK perdido



tiempo de espera prematuro

TCP: escenarios de retransmisión



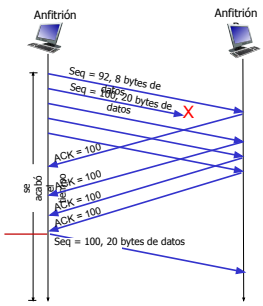
Coberturas acumulativas de ACK para ACK perdidos anteriormente

Retransmisión rápida TCP

Retransmisión rápida TCP

si el remitente recibe 3 ACK adicionales por los mismos datos ("ACK duplicados triples"), reenviar desembarazado segmento con el número de secuencia más pequeño

- probablemente eso desembarazado segmento perdido, así que noNo espere el tiempo de espera
  - la recepción de tres ACK duplicados indica que se recibieron 3 segmentos después de un segmento faltante; es probable que se pierda un segmento. ¡Así que

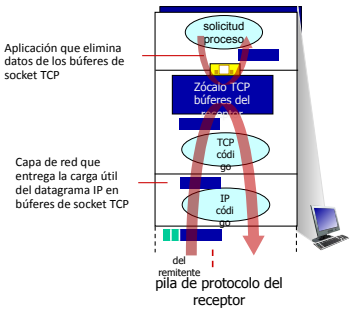


Capítulo 3: hoja de ruta

- Servicios de la capa de transporte
- Multiplexación y demultiplexación
- Transporte sin conexión: UDP
- Principios de la transferencia de datos confiable
- Transporte orientado a la conexión: TCP
  - estructura del segmento
  - transferencia de datos confiable
  - control de flujo
  - gestión de conexión
- Principios del control de la

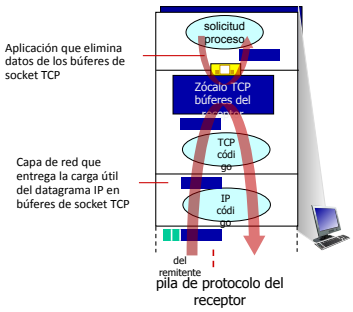
Control de flujo de TCP

Q: ¿Qué sucede si la capa de red entrega datos más rápido que la capa de aplicación elimina datos de los búferes de socket?



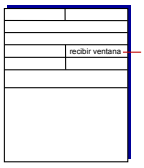
Control de flujo de TCP

Q: ¿Qué sucede si la capa de red entrega datos más rápido que la capa de aplicación elimina datos de los búferes de socket?

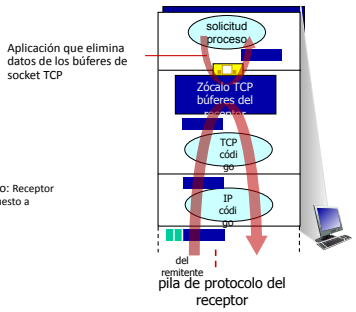


Control de flujo de TCP

Q: ¿Qué sucede si la capa de red entrega datos más rápido que la capa de aplicación elimina datos de los búferes de socket?



control de flujo: Receptor de # bytes dispuesto a aceptar

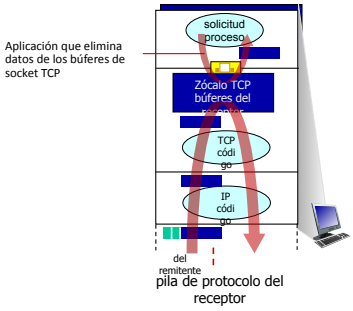


Control de flujo de TCP

Q: ¿Qué sucede si la capa de red entrega datos más rápido que la capa de aplicación elimina datos de los búferes de socket?

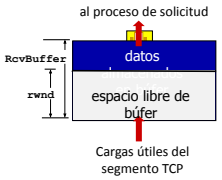
control de flujo

el receptor controla al remitente, por lo que el remitente ganó 't desbordar el búfer del receptor ~~transmitiendo demasiado,~~ demasiado rápido



Control de flujo de TCP

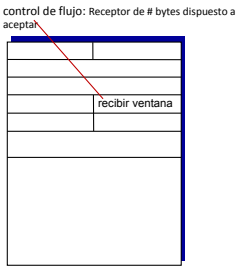
- Receptor TCP "anuncia "espacio de búfer libre en **`rwnd`** campo en el encabezado TCP
  - **`RcvBuffer`** tamaño establecido a través de las opciones de socket (el valor predeterminado típico es 4096 bytes)
  - muchos sistemas operativos auto ajuste **`RcvBuffer`**
- el remitente limita la cantidad de desembarazado ("en vuelo ") datos recibidos **`rwnd`**
- garantiza que el búfer de recepción



Almacenamiento en búfer del lado del receptor TCP

Control de flujo de TCP

- Receptor TCP "anuncia "espacio de búfer libre en **`rwnd`** campo en el encabezado TCP
  - **`RcvBuffer`** tamaño establecido a través de las opciones de socket (el valor predeterminado típico es 4096 bytes)
  - muchos sistemas operativos auto ajuste **`RcvBuffer`**
- el remitente limita la cantidad de desembarazado ("en vuelo ") datos recibidos **`rwnd`**
- garantiza que el búfer de recepción

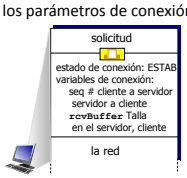


Formato de segmento TCP

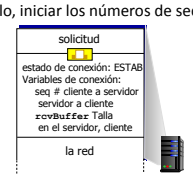
Gestión de conexiones TCP

antes de intercambiar datos, emisor / receptor "apretón de manos":

- acuerda establecer conexión (cada uno conociendo al otro dispuesto a establecer conexión)
- acordar los parámetros de conexión (por ejemplo, iniciar los números de secuencia)



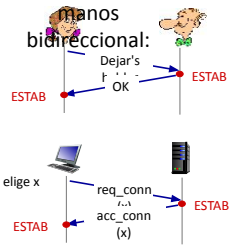

```
Enchufe clientSocket =
 newSocket("nombre de host "," puerto
 número");
```




```
Enchufe conexión =
 welcomeSocket.accept();
```

Aceptar establecer una conexión

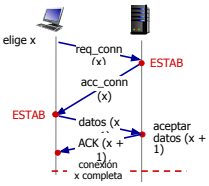
Apretón de manos bidireccional:



Q: ¿Funcionará siempre el protocolo de enlace bidireccional en red?

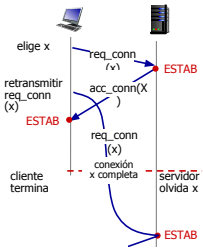
- retrasos variables
- mensajes retransmitidos (p. ej. req_conn(x)) debido a la pérdida del mensaje
- reordenación de mensajes
- Cun't "Ver" el otro lado

Escenarios de protocolo de enlace bidireccional



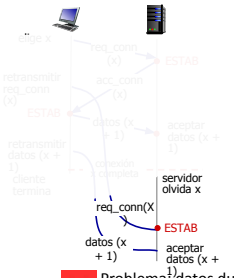
¡No hay pro[illegible]a!

Escenarios de protocolo de enlace bidireccional



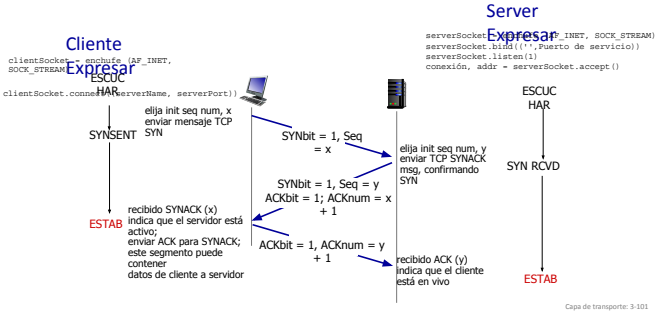
Problema: conexión medio abierta. (sin cliente)

Escenarios de protocolo de enlace bidireccional



Problema: datos du ¡aceptado!

Protocolo de enlace de 3 vías TCP



Un protocolo de apretón de manos de 3 vías humano

Cerrar una conexión TCP

- cliente, servidor cada uno cierra su lado de conexión
  - enviar segmento TCP con FIN bit = 1
- responder al FIN recibido con ACK
  - al recibir FIN, ACK se puede combinar con FIN propio
- se pueden gestionar intercambios FIN simultáneos

Capítulo 3: hoja de ruta

- Servicios de la capa de transporte
- Multiplexación y demultiplexación
- Transporte sin conexión: UDP
- Principios de la transferencia de datos confiable
- Transporte orientado a la conexión: TCP
- Principios del control de la congestión
- Control de congestión TCP
- Evolución de la funcionalidad de la

Principios del control de la congestión

Congestión:

- informalmente: "demasiadas fuentes que envían demasiados datos demasiado rápido para *la red* manejar"
- manifestaciones:
  - retrasos prolongados (hacer cola en los búferes de
  - pagacket pérdida (desbordamiento del búfer en lo
- diferente del control de flujo!
- un problema top-10!

control de la congestión: demasiados remitentes, enviando demasiado rápido

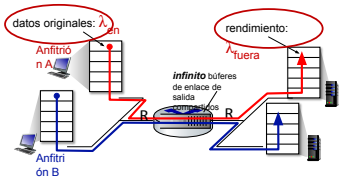
control de flujo: un remitente demasiado rápido para un receptor

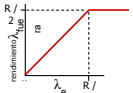
Causas / costos de la congestión: escenario 1

Escenario más simple:

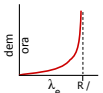
- un enrutador, búferes infinitos
- dos flujos
- entrada capacidad del enlace de salida: R
- no se necesitan retransmisiones

Q: Que pasa como tasa de llegada λ_{en} se acerca a R / 2?





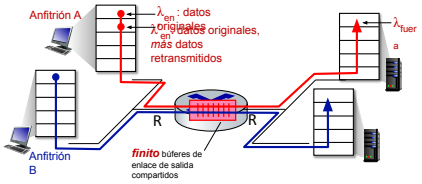
rendimiento máximo por conexión: R / 2



grandes retrasos como tasa de llegada λ_{en} se acerca a la capacidad

Causas / costos de la congestión: escenario 2

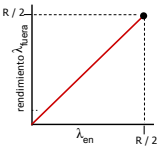
- un enrutador, *finito* amortiguadores
- el remitente retransmite el paquete perdido, agotado
  - entrada de la capa de aplicación = salida de la capa de aplicación: $\lambda_{en} = \lambda_{fuera}$
  - la entrada de la capa de transporte incluye *retransmisiones* : $\lambda'_{en} \geq \lambda_{en}$

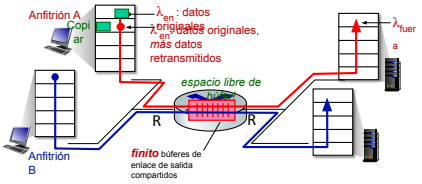


Causas / costos de la congestión: escenario 2

Idealización: conocimiento perfecto

- el remitente envía solo cuando los búferes del enrutador están disponibles





Causas / costos de la congestión: escenario 2

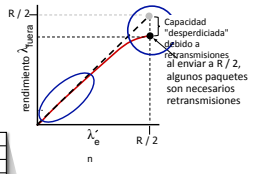
Idealización: *algunos* conocimiento perfecto

- los paquetes se pueden perder (caer en el enrutador) debido a los búferes llenos
- el remitente sabe cuándo se ha descartado el paquete: solo se reenvía si el paquete *conocido* está perdido

Causas / costos de la congestión: escenario 2

Idealización: *algunos* conocimiento perfecto

- los paquetes se pueden perder (caer en el enrutador) debido a los búferes llenos
- el remitente sabe cuándo se ha descartado el paquete: solo se reenvía si el paquete *conocido* está perdido

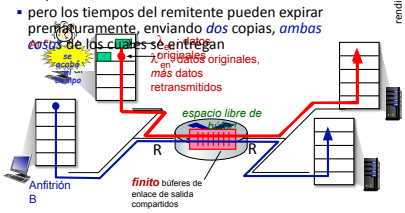


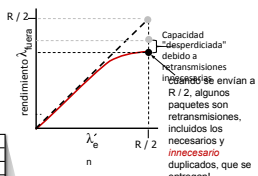
cuando se envían a R / 2, algunos paquetes son necesarios retransmisiones

Causas / costos de la congestión: escenario 2

Escenario realista: *innecesario duplicados*

- los paquetes se pueden perder, descartar en el enrutador debido a los búferes llenos, lo que requiere retransmisiones
- pero los tiempos del remitente pueden expirar prematuramente, enviando *dos* copias, *ambas costas* de los cuales se entregan



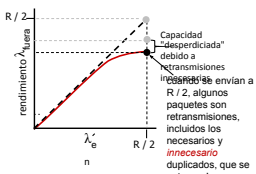


cuando se envían a R / 2, algunos paquetes son retransmisiones, incluidos los necesarios y *innecesario* duplicados, que se entregan!

Causas / costos de la congestión: escenario 2

Escenario realista: *innecesario duplicados*

- los paquetes se pueden perder, descartar en el enrutador debido a los búferes llenos, lo que requiere retransmisiones
- pero los tiempos del remitente pueden expirar prematuramente, enviando *dos* copias, *ambas cosas* de los cuales se entregan



cuando se envían a R / 2, algunos paquetes son retransmisiones, incluidos los necesarios y *innecesario* duplicados, que se entregan!

"Costos" de la congestión:

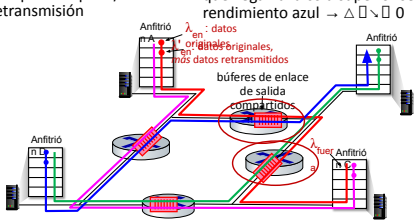
- más trabajo (retransmisión) para un receptor dado rendimiento
- retransmisiones innecesarias: el enlace lleva varias copias de un paquete
  - disminución del rendimiento máximo alcanzable

Causas / costos de la congestión: escenario 3

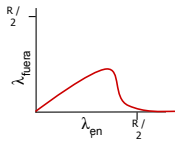
- *cuatro* remitentes
- *multi-salto* rutas
- tiempo de espera / retransmisión

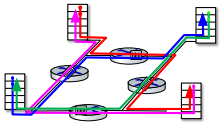
Q: que pasa como λ_{en} y λ_{en}' incrementar ?

A: como rojo λ_{en}' aumenta, todos los paquetes azules que llegan a la cola superior se eliminan, el rendimiento azul → △□↘□ 0



Causas / costos de la congestión: escenario 3



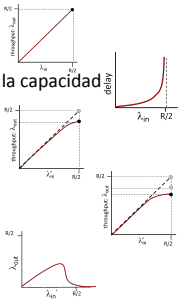


otro "costo ”de la congestión:

- cuando el paquete se cae, cualquier ¡La capacidad de transmisión ascendente y el almacenamiento en búfer utilizados para ese paquete se desperdiciaron!

Causas / costos de la congestión: conocimientos

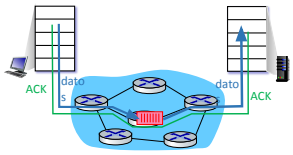
- el rendimiento nunca puede exceder la capacidad
- el retraso aumenta a medida que se acerca la capacidad
- la pérdida / retransmisión disminuye el rendimiento efectivo
- los duplicados innecesarios disminuyen aún más el rendimiento efectivo
- Capacidad de transmisión ascendente / almacenamiento en búfer desperdiciado por paquetes perdidos en sentido descendente



Enfoques hacia el control de la congestión

micontrol de congestión de nd-end:

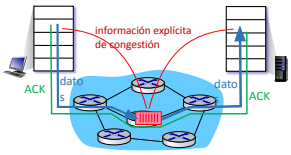
- sin comentarios explícitos de la red
- congestión *inferido* de pérdida observada, retraso
- enfoque adoptado por TCP



Enfoques hacia el control de la congestión

Control de congestión asistido por red:

- los enrutadores proporcionan *directo* retroalimentación a los hosts de envío / recepción con flujos que pasan a través del enrutador congestionado
- puede indicar el nivel de congestión o establecer explícitamente la tasa de envío
- TCP ECN, cajero automático, DECbit protocolos



Capítulo 3: hoja de ruta

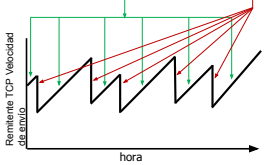
- Servicios de la capa de transporte
- Multiplexación y demultiplexación
- Transporte sin conexión: UDP
- Principios de la transferencia de datos confiable
- Transporte orientado a la conexión: TCP
- Principios del control de la congestión
- Control de congestión TCP
- Evolución de la funcionalidad de la

Control de congestión TCP: AIMD

- *Acercarse:* los remitentes pueden aumente la velocidad de envío hasta que se produzca la pérdida de paquetes (congestión), luego disminuya la velocidad de envío en caso de pérdida

Aditivo laumentar
aumentar la tasa de envío por 1 tamaño máximo de segmento cada RTT hasta que se detecte una pérdida

METROultiplicativo Daumentar
reducir la tasa de envío a la mitad en cada evento de pérdida



AIMD diente de sierra comportamiento: *sondeo* para ancho de

TCP AIMD: más

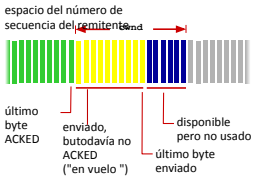
Disminución multiplicativa detalle: la tasa de envío es

- Reducir a la mitad la pérdida detectada por ACK duplicado triple (TCP Reno)
- Corte a 1 MSS (tamaño máximo de segmento) cuando se detecte una pérdida por tiempo de espera (TCP Tahoe)

Por qué AIMETROD?

- AIMD, un algoritmo distribuido y asincrónico, se ha demostrado que:
  - ¡Optimice los caudales congestionados en toda la red!
  - tienen propiedades de estabilidad deseables

Control de congestión TCP: detalles



Comportamiento de envío de TCP:

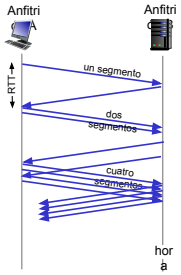
- *aproximadamente:* enviar `cwnd` bytes, espere RTT [illegible]s

$$\text{Tasa de TCP} \approx \frac{\text{cwnd}}{\text{RTT}} \text{ bytes / seg}$$

- El remitente TCP limita la transmisión: `LastByteSent- LastByteAcked ≤ cwnd`
- `cwnd` se ajusta dinámicamente en respuesta a la congestión de la red observada (implementando el control de congestión de TCP)

Inicio lento de TCP

- cuando comience la conexión, aumente la tasa exponencialmente hasta el primer evento de pérdida:
  - inicialmente **`cwnd`** = 1 MSS
  - doble **`cwnd`** cada RTT
  - hecho aumentando **`cwnd`** por cada ACK recibido
- *resumen:* la tasa inicial es lenta, pero aumenta exponencialmente rápido



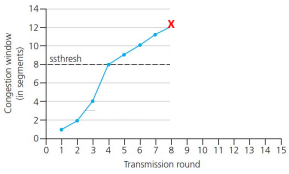
TCP: desde un inicio lento hasta evitar la congestión

Q: ¿Cuándo debería cambiar el aumento exponencial a lineal?

A: Cuándo **`cwnd`** llega a la mitad de su valor antes del tiempo de espera.

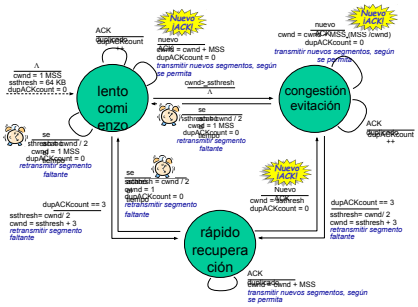
Implementación:

- variable **`ssthresh`**
- en caso de pérdida, **`ssthresh`** está ajustado a la mitad de **`cwnd`** justo antes del evento de pérdida



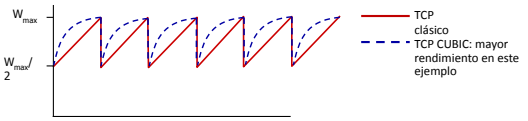
* Consulte los ejercicios interactivos en línea para ver más ejemplos: http: //gaia.cs.umass.edu/kurose_ross/interactivo/

Resumen: control de congestión de TCP



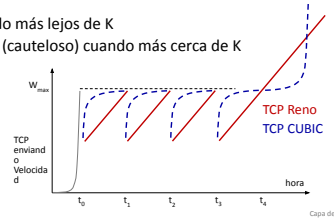
TCP CUBIC

- ¿Existe una forma mejor que AIMD de "sondear" el ancho de banda utilizable?
- Perspicacia / intuición:
  - W_{max}: velocidad de envío a la que se detectó la pérdida por congestión
  - el estado de congestión del enlace de cuello de botella probablemente (?) no ha cambiado mucho
  - después de cortar la tasa / ventana a la mitad en la pérdida, inicialmente rampa a W_{max} *más rápido*, pero luego acércate W_{max} más *lentamente*



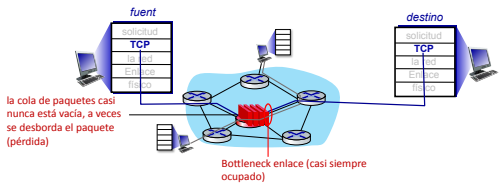
TCP CUBIC

- K: momento en el que alcanzará el tamaño de la ventana TCP W_{max}
  - K en sí mismo es sintonizable
- aumentar W en función de la *cubo* de la distancia entre la hora actual y K
  - mayores aumentos cuando más lejos de K
  - aumentos más pequeños (cauteloso) cuando más cerca de K
- TCP CUBIC predeterminado en Linux, el TCP más popular para servidores web populares



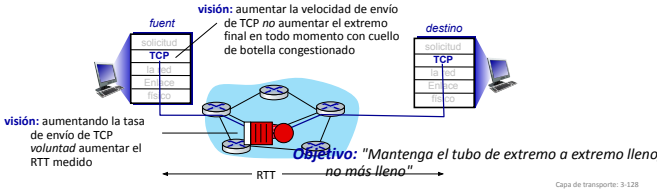
TCP y el "vínculo de cuello de botella" congestionado

- TCP (clásico, CUBIC) aumenta la velocidad de envío de TCP hasta que se produce la pérdida de paquetes en la salida de algún enrutador: el *enlace de cuello de botella*



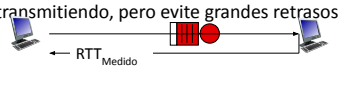
TCP y el "vínculo de cuello de botella" congestionado

- TCP (clásico, CUBIC) aumenta la velocidad de envío de TCP hasta que se produce la pérdida de paquetes en la salida de algún enrutador: el *enlace de cuello de botella*
- comprender la congestión: útil para centrarse en el vínculo de cuello de botella congestionado



TCP basado en retardo Congestión Control

Mantener el conducto de remitente a receptor "lo suficientemente lleno, pero no más lleno": mantenga ocupado el enlace de cuello de botella transmitiendo, pero evite grandes retrasos / almacenamiento en búfer



$$\text{Medido rendimiento} = \frac{\text{\# bytes enviados en el último intervalo RTT}}{\text{RTT Medido}}$$

Enfoque basado en retrasos:

- RTT_{min} - RTT mínimo observado (camino no congestionado)
- rendimiento no congestionado con ventana de congestión `cwnd` es cwnd/RTT_{min}

```
si el rendimiento medido es "muy cercano" al rendimiento no
congestionado
 incrementar cwnd linealmente / * ya que la ruta no está
congestionada * /
de lo contrario, si el rendimiento medido "muy por debajo" no
```

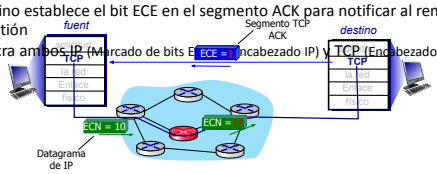

TCP basado en retardo Congestión Control

- control de la congestión sin inducir / forzar pérdidas
- maximizando todo ("manteniendo la tubería justa llena ...") mientras se mantiene el retraso bajo ("... pero no más lleno")
- varios TCP implementados adoptan un enfoque basado en retrasos
  - BBR implementado en la red troncal (interna) de Google

Notificación de congestión explícita (ECN)

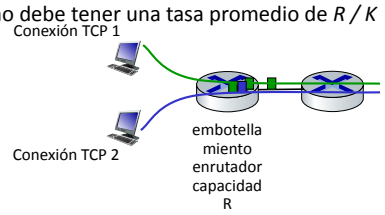
Las implementaciones de TCP a menudo implementan *asistido por red* control de la congestión:

- dos bits en el encabezado IP (ToS campo) marcado *por enrutador de red* para indicar congestión
  - *política* para determinar el marcado elegido por el operador de red
- indicación de congestión llevada al destino
- el destino establece el bit ECE en el segmento ACK para notificar al remitente de la congestión
- involucra ambos IP (Marcado de bits E en el encabezado IP) y TCP (Encabezado TCP C, marca de bit E)



Equidad de TCP

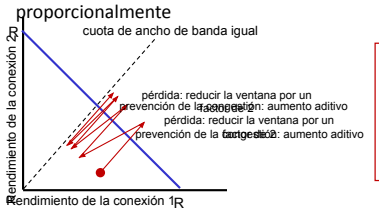
Objetivo de equidad: Si *K* Las sesiones de TCP comparten el mismo enlace de cuello de botella de ancho de banda *R*, cada uno debe tener una tasa promedio de *R / K*



P: ¿TCP es justo?

Ejemplo: dos sesiones de TCP en competencia:

- el aumento aditivo da una pendiente de 1, ya que a lo largo aumenta
- la disminución multiplicativa disminuye el rendimiento proporcionalmente



Es ¿TCP Justo?

Sí, bajo supuestos idealizados:

- mismo RTT
- número fijo de sesiones solo para evitar la congestión

Equidad: ¿todas las aplicaciones de red deben ser "justas"?

Equidad y UDP

- las aplicaciones multimedia a menudo no usan TCP
  - no quiero que la tasa se vea limitada por el control de la congestión
- en su lugar use UDP:
  - enviar audio / video a una velocidad constante, tolerar la pérdida de paquetes
- no existe una "policía de Internet" que controle el uso del control de la congestión

Equidad, conexiones TCP paralelas

- la aplicación puede abrir *múltiple* conexiones paralelas entre dos hosts
- Los navegadores web hacen esto, por ejemplo, un enlace de tasa R con 9 conexiones existentes:
  - la nueva aplicación solicita 1 TCP, obtiene la tasa R / 10
  - nueva aplicación solicita 11 TCP, obtiene R / 2

Capa de transporte: hoja de ruta

- Servicios de la capa de transporte
- Multiplexación y demultiplexación
- Transporte sin conexión: UDP
- Principios de la transferencia de datos confiable
- Transporte orientado a la conexión: TCP
- Principios del control de la congestión
- Control de congestión TCP
- Evolución de la funcionalidad de la

Evolución de la funcionalidad de la capa de transporte

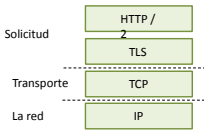
- TCP, UDP: principales protocolos de transporte durante 40 años
- diferentes "sabores" de TCP desarrollados, para escenarios específicos:

Guión	Desafíos
Tubos largos y gordos (grandes transferencias de datos)	Muchos paquetes "en vuelo"; la pérdida cierra la tubería
Redes inalámbricas	Pérdida debido a enlaces inalámbricos ruidosos, movilidad; TCP trata esto como pérdida por congestión
Enlaces de larga demora	RTT extremadamente largos
Redes de centros de datos	Sensible a la latencia
Flujos de tráfico de fondo	Flujos de TCP "en segundo plano" de baja prioridad

- mover las funciones de la capa de transporte a la capa de aplicación, además de UDP
  - HTTP / 3: QUIC

QUIC: Conexiones rápidas a Internet UDP

- protocolo de capa de aplicación, además de UDP
  - aumentar el rendimiento de HTTP
  - implementado en muchos servidores de Google, aplicaciones (Chrome, aplicación móvil de YouTube)



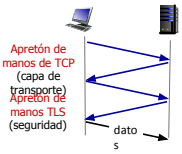
HTTP / 2 sobre TCP

QUIC: Conexiones rápidas a Internet UDP

adopta enfoques que hemos estudiado en este capítulo para el establecimiento de conexiones, control de errores, control de congestión

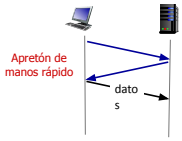
- **control de errores y congestión:** "Los lectores familiarizados con la detección de pérdidas y el control de congestión de TCP encontrarán aquí algoritmos que son paralelos a los conocidos de TCP". [de la especificación QUIC]
- **establecimiento de conexión:** confiabilidad, control de congestión, autenticación, encriptación, estado establecido en un RTT
- múltiples "flujos" a nivel de aplicación multiplexados a través de una sola conexión QUIC
  - transferencia de datos confiable separada, seguridad
  - control de congestión común

QUIC: establecimiento de conexión



TCP (confiabilidad, estado de control de congestión) + TLS (autenticación, estado criptográfico)

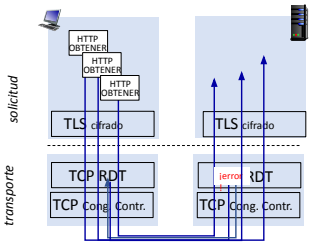
- 2 apretones de manos en serie



QUIC: confiabilidad, control de congestión, autenticación, estado criptográfico

- 1 apretón de manos

QUIC: streams: paralelismo, sin bloqueo HOL



(a) HTTP 1.1

Capítulo 3: resumen

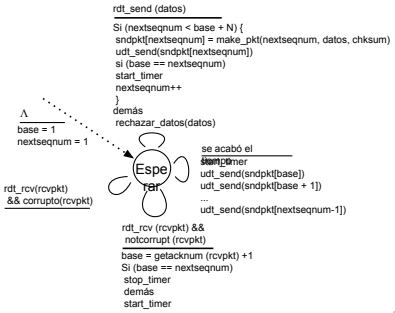
- principios detrás de los servicios de la capa de transporte:
  - multiplexación, demultiplexación
  - transferencia de datos confiable
  - control de flujo
  - control de congestión
- instanciación, implementación en Internet
  - UDP
  - TCP

Hasta la próxima:

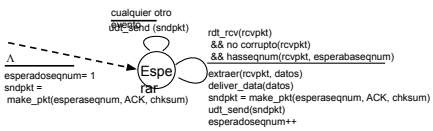
- dejando la red "borde" (aplicación, capas de transporte)
- en la red "centro"
- dos capítulos de capa de red:
  - plano de datos
  - plano de control

Diapositivas adicionales del Capítulo 3

Go-Back-N: FSM extendido del remitente



Go-Back-N: FSM extendido del receptor



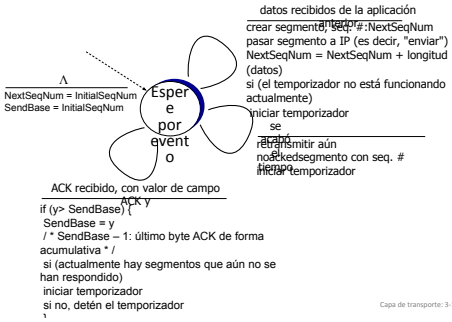
Solo ACK: envíe siempre ACK para el paquete recibido correctamente con la mayor *en orden* seq #

- puede generar ACK duplicados
- solo necesito recordar **`esperadoseqnum`**

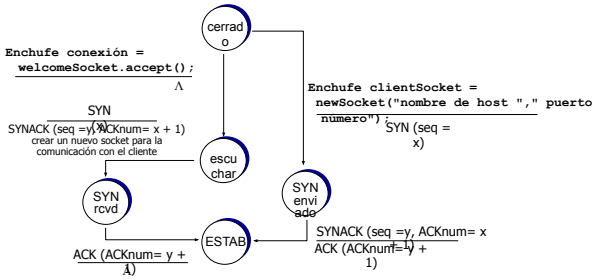
paquete fuera de servicio:

- descartar (don 't búfer): *sin búfer del receptor!*
- re-ACK pkt con el número de secuencia en orden más alto

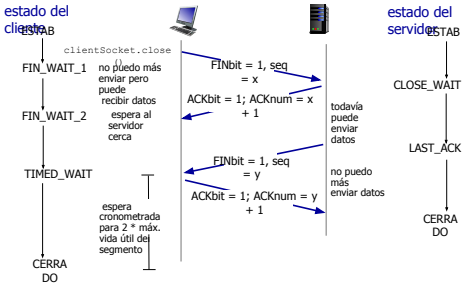
Remitente TCP (simplificado)



FSM de protocolo de enlace de 3 vías TCP

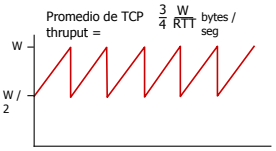


Cerrar una conexión TCP



Rendimiento de TCP

- prom. TCPthruput en función del tamaño de la ventana, RTT?
  - ignore el inicio lento, suponga que siempre hay datos para enviar
- W: tamaño de la ventana (medido en bytes) donde ocurre la perdida
  - prom. el tamaño de la ventana (# bytes en vuelo) es ¾ W
  - prom. thruput es 3 / 4W por RTT



TCP sobre "tubos largos y gruesos "

- ejemplo: segmentos de 1500 bytes, RTT de 100 ms, desea un rendimiento de 10 Gbps
- requiere W = 83,333 segmentos en vuelo
- rendimiento en términos de probabilidad de pérdida de segmento, L [Mathis 1997]:

$$\text{Rendimiento de TCP} = \frac{1{,}22 \cdot MSS}{RTT\sqrt{L}}$$

➜ para lograr un rendimiento de 10 Gbps, necesita una tasa de pérdida de $L = 2 \cdot 10^{-10}$ - *¡una tasa de pérdida muy pequeña!*

- versiones de TCP para escenarios largos y de alta velocidad