1

# La capa de Transporte

Transporte orientado a conexión: TCP

2

## Tema 3: La capa de Transporte

3.1 La capa de Transporte y sus servicios
3.2 Multiplexacción y demultiplexación
3.3 Transporte sin conexión: UDP
3.4 Principios de transferencia de datos fiable
3.5 Transporte orientado a conexión: TCP
- 3.5.1 La conexión TCP
- 3.5.2 Estructura del segmento TCP
- 3.5.3 Estimación del tiempo de ida y vuelta y fin de temporización
- 3.5.4 Transferencia de datos fiable
- 3.5.5 Control de flujo
- 3.5.6 Gestión de la conexión TCP

3.6 Principios de control de congestión
3.7 Mecanismo de control de congestión de TCP

## TCP: Transmission Control Protocol

3

- Especificado en varias RFC:
  - RFC 793, 1122, 1323, 2018, y 2581.
- Protocolo de la capa de Transporte.
  - Orientado a conexión, dado todos los servicios que provee.
  - Provee control de flujo y control de congestión.
  - Fiable, en base a los principios básicos estudiados en el apartado 3.4.
    - Detección de errores.
    - Retransmisiones.
    - Reconocimientos acumulativos.
    - Números de secuencia.
    - Temporizadores.



## TCP: La conexión TCP

4

- Orientado a conexión.
  - Los procesos de aplicación, en sendos sistemas finales, primero deben establecer-acordar la comunicación.
    - Intercambio de segmentos especiales para establecer los parámetros para la transferencia de datos.
      - Las partes (cliente y servidor) inician diferentes variables de estado de TCP.
  - Los sistemas intermedios (*routers*, etc.) son TRANSPARENTES a la "conexión" TCP.
    - No mantienen estados de la conexión TCP.

## TCP: La conexión TCP

5

- Conexión TCP (conexión lógica).
  - Full-duplex (bidireccional simultánea).
  - Flujo de octetos no estructurados, no flujo de mensajes.
  - Punto a punto.
  - No permite multidifusión ni difusión.
  - Acuerdo en tres fases.
    - Intercambio de **tres segmentos especiales** para el "acuerdo lógico".
      - Primeros dos segmentos no portan datos de capa de aplicación.
      - Tercer segmento puede portar datos de capa de aplicación.
    - Concluye con el establecimiento de las variables de estado y capacidades en *buffers.*
      - En ambos procesos TCP (cliente y servidor) del par de sistemas finales.
    - Conlleva un retardo intrínseco previo a la transferencia de datos.



## TCP: La conexión TCP

6

- Conexión TCP establecida.
  - Proceso de aplicación cliente pasa datos a la capa TCP.
  - TCP transfiere segmentos que contienen PDUs (datos) de la capa de aplicación.
    - "Según su propia conveniencia". La especificación TCP es vaga en este sentido.
  - *Buffers* TCP → Introducen cierto retardo.
    - Ver *applet* que muestra animación del funcionamiento de los *buffers* TCP emisor y receptor.
    - http://media.pearsoncmg.com/aw/aw_kurose_network_4/applets/flow/FlowControl.htm

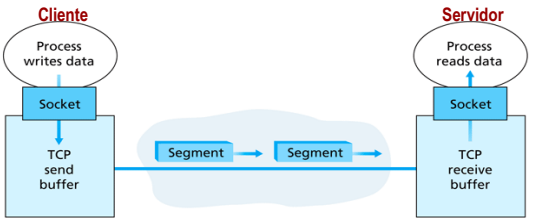



**Figure 3.28** ♦ TCP send and receive buffers

**Toda conexión TCP requiere en Emisor y Receptor**:
- *Buffers*
- Variables
- *Sockets* de conexión a un proceso

## TCP: La conexión TCP

7

- MSS: Maximum Segment Size.
  - Cantidad máxima de datos que puede portar el segmento TCP.
  - Depende del tamaño de la MTU, según:
    - Trama más grande que puede ensamblar el emisor, y
    - Trama más grande que puede enviarse entre emisor y receptor a través de todos los enlaces entre origen y destino.
      - Hay métodos para determinar la MTU del trayecto y establecer el MSS.
    - Garantiza que el segmento TCP se "ajuste" a una única trama ($PDU_2$).
      - Evita tener que segmentar/fragmentar.
  - Se acuerda o se impone en la fase de establecimiento de la conexión.
    - Valor mínimo propuesto por las partes, sino valor por defecto = 536 octetos.
    - Posibilita que la $PDU_{TCP}$ no necesite un campo de control que indique su longitud.



## TCP: Estructura del segmento TCP

8

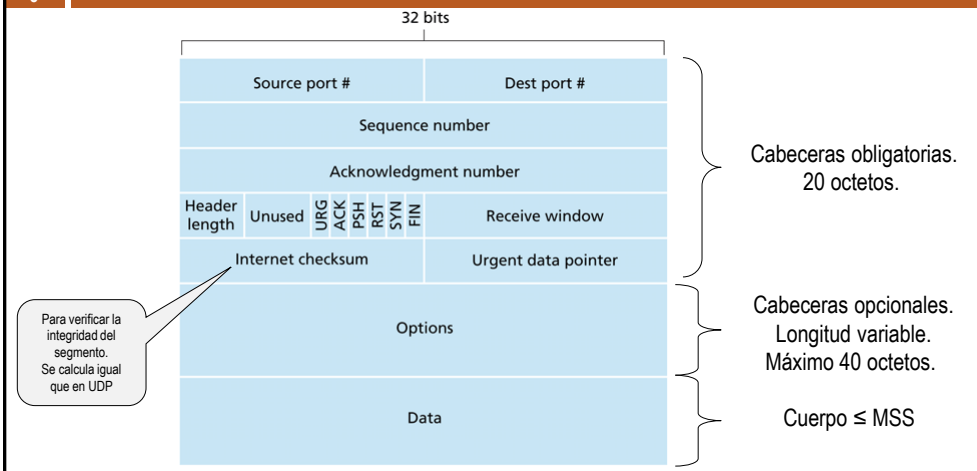



**Figure 3.29** ♦ TCP segment structure

30 32 33 34 35 36 37 40

## TCP: Estructura del segmento TCP. Cabeceras

9

- Puertos origen y destino (16 bits c/u)
- Nº de secuencia (32 bits).
- Nº de reconocimiento (32 bits).

  } Para transferencia fiable y control de flujo.

- Longitud de cabecera (4 bits).
  - En palabras de 32 bits.
  - Si está vacío → La cabecera tiene cinco palabras de 32 bits cada una (20 octetos).
- Indicadores (flags) (1 bit c/u).
  - URG, ACK, PSH, RST, SYN, FIN
- Ventana de recepción (16 bits) ← Para control de flujo
- Suma de comprobación (16 bits) ← Del segmento completo + pseudo cabecera
- Puntero de datos urgente (16 bits) ← Para indicar ubicación (inicio) de datos urgentes
- Opciones (opcional y de longitud variable) ← Normalmente NO presente.



## TCP: Estructura del segmento TCP. Cabeceras

10

### Indicadores (flags)

- ACK.
  - Para indicar que el valor del campo "nº de reconocimiento" es válido.
- SYN, FIN.
  - Para el establecimiento y cierre de conexiones respectivamente.
- PSH.
  - Para indicar al receptor* que pase de inmediato los datos a la capa superior
- URG.
  - Para indicar que el segmento porta datos marcados como urgentes por la capa superior → Campo "puntero de datos urgente" válido.

## TCP: Estructura del segmento TCP. Cabeceras

11

### Indicadores (flags) (II).

- RST.
  - Se utiliza en el segmento especial de reinicio o rechazo de conexión, para indicar cosas como:
    - Rechazo de solicitud de conexión cuando:
      - No existe *socket* activo para la dirección IP o puerto origen de la solicitud, o
      - Servidor NO está "escuchando" por el puerto al que el cliente envía el segmento SYN.
    - Reinicio de conexión cuando:
      - Se han producido errores o caída de la conexión.
- Todo flag (SYN, ACK, FIN, PSH, URG, RST) con valor lógico:
  - "1" → Indica que está activado.
  - "0" → Indica que está desactivado



## TCP: Estructura del segmento TCP. Cabeceras

12

- Número de secuencia.
  - Indica lo que se envía.
  - Indica el nº del primer octeto que porta el segmento respecto al flujo total de octetos.
- Número de reconocimiento (asentimiento).
  - Indica lo que se espera recibir.
  - Indica el nº de secuencia del siguiente octeto que se espera recibir.
- Ambos SIEMPRE presentes en TODO segmento TCP.
  - Hacen referencia al flujo de octetos, no a los segmentos.
  - TCP percibe los datos como un flujo de octetos ordenados pero no estructurados.
- Elección de los nº de secuencia.
  - De manera aleatoria por ambas partes, para minimizar posibilidad de coincidencia con segmentos antiguos que estén en la red.
  - Ver caso de estudio mediante aplicación Telnet: 5ª e, pág. 234-236; 7ª e, pág. 196-197.

## TCP: asentimientos y nºs de secuencia.

13

- Reconocimientos acumulativos.
  - TCP sólo confirma (reconoce) hasta el primer octeto que falta en el flujo.
- ¿Qué hace un host cuando NO recibe los segmentos en orden?
  - Las especificaciones TCP no establecen nada al respecto, queda a decisión de los implementadores.
  - Opciones básicas:
    - Receptor descarta segmentos que no lleguen en secuencia.
      - Simplifica el diseño del receptor.
    - Receptor acepta segmentos fuera de secuencia, los almacena en el *buffer* en espera de los segmentos faltantes.
      - Solución más eficiente respecto a uso del ancho de banda.
      - Solución normalmente utilizada en la práctica.



## TCP: Estimación del RTT y fin de temporización

14

- ¿Cómo TCP se recupera ante la pérdida de segmentos?
  - Con temporizadores y procedimientos ARQ.
- Gestión de temporizadores en TCP (RFC 2988).
  - En base al valor del RTT: *timeout* > RTT
    - RTT varia para cada segmento, se calculan RTTs estimados.
  - *Timeout* = IntervaloFinTemporización = RTTestimado + 4RTTdesv
    - RTTestimado = (1 –α)RTTestimado + αRTTmuestra
      - Se recomienda α = 0,125 = 1/8
      - RTTestimado = 0,875RTTestimado + 0,125RTTmuestra
    - RTTdesv: estimación de cuánto se desvía RTTmuestra del RTTestimado.
      - RTTdesv = (1- β)RTTDesv + β|RTTmuestra – RTTestimado|
      - Se recomienda β = 0,25 = 1/4
      - RTTdesv = 0,75RTTDesv + 0,25|RTTmuestra – RTTestimado|

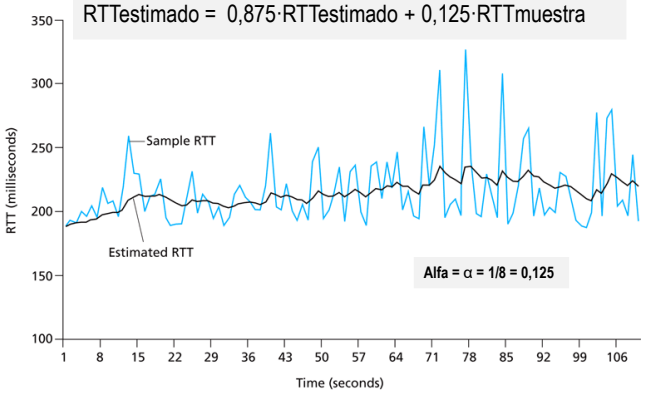
**Media móvil exponencial ponderada**, a largo plazo da mayor peso a las muestras recientes respecto a las muestras antiguas.

## TCP: Estimación del RTT y fin de temporización

15

- RTTmuestra y RTTestimado para una conexión TCP entre EE.UU. y Francia

RTTestimado = 0,875·RTTestimado + 0,125·RTTmuestra




Alfa = α = 1/8 = 0,125

**RTTmuestra**: tiempo que transcurre desde que se pasa el segmento a la capa IP (se envía) hasta que se recibe su reconocimiento (ACK).
**RTTestimado**: valor promedio de los valores RTTmuestra.

Figure 3.32 ♦ RTT samples and RTT estimates

Nuevo valor **RTTmuestra** cada RTT segundos aproximadamente



## TCP: Transferencia de datos fiable

16

- Gestión de temporizadores → Carga de trabajo que puede ser considerable.
- En consecuencia, para TCP la RFC 2988 recomienda:
  - Un único temporizador de retransmisión, independientemente del nº de segmentos pendientes de ser reconocidos.
- Sucesos importantes en el emisor TCP relativos a la transmisión y retransmisión de datos son:
  - Datos recibidos desde la capa de aplicación.
  - Fin de la temporización (*timeout*).
  - Recepción de un ACK.

## TCP: Transferencia de datos fiable

17

Sucesos importantes en el emisor TCP (versión simplificada)

- Datos recibidos desde la capa de aplicación.
  - TCP lo encapsula en un segmento y lo pasa a la capa IP.
  - Si el temporizador NO está iniciado, se inicia el temporizador.
- Fin de temporización (*timeout*) .
  - TCP retransmite segmento asociado a dicha temporización (solo el segmento más antiguo, caso que explica el Kurose) y se reinicia el temporizador.
- Recepción de un segmento con ACK válido.
  - Compara el valor ACK "y" con la variable "BaseEmision".
    - BaseEmision = Nº de secuencia más antiguo pendiente de ACK.
  - Si "y" > BaseEmision → El ACK confirma uno o más segmentos anteriores no reconocidos
    - Se actualiza la variable BaseEmision.
    - Si aún hay segmentos NO reconocidos → Se reinicia el temporizador.

Universidad de Alcalá

## TCP: Transferencia de datos fiable

18

Duplicación del *timeout*.

- Modificación utilizada en la mayoría de implementaciones TCP cuando vence el temporizador.
- Cada vez que TCP retransmite redefine el *timeout*.
  - Nuevo *timeout* = 2 veces *timeout* anterior.
    - En este caso NO se obtiene el *Timeout* a partir de "RTTestimado + 4·RTTdesv".
  - Los *timeouts* crecen exponencialmente después de cada retransmisión.
  - Posibilita una forma limitada de control de congestión.
    - Actuación "consciente" de TCP frente a congestiones en la red, aumentando los intervalos de retransmisión y evitando el empeoramiento de la congestión.
- En los otros dos casos, recepción de "datos de aplicación" y "ACK":
  - Nuevo *timeout* = RTTestimado + 4.RTTdesv

Universidad de Alcalá

Tema 3: La capa de Transporte

## TCP: Transferencia de datos fiable

19

### Política de generación de ACKs por el receptor TCP.

- Recomendación según las RFC 1122 y 2581.
- No descarte de segmentos fuera de secuencia y asentimientos acumulativos.

| | Suceso | Acción del Receptor TCP |
|---|---|---|
| Fila 1 | Se recibe segmento con nº de secuencia esperado, nºs de secuencia anteriores ya reconocidos. No hay otro segmento en orden esperando transmisión de un ACK. | **ACK retardado*.** Se espera 500 mseg la llegada de otro segmento en secuencia. Si no llega, se envía ACK. |
| Fila 2 | Se recibe segmento con nº de secuencia esperado. Hay otro segmento en orden esperando transmisión de ACK. | **ACK único acumulativo**. De inmediato se reconocen ambos segmentos ordenados. |
| Fila 3 | Se recibe segmento fuera de secuencia, con nº mayor que el esperado. Se detecta un "hueco". | **ACK duplicado**. Se envía de inmediato ACK con nº de secuencia del siguiente octeto esperado (límite inferior del "hueco"). |
| Fila 4 | Se recibe segmento que completa parcial o totalmente "hueco" en los datos recibidos. | **ACK inmediato**. Se envía ACK de inmediato si el segmento comienza en el límite inferior del "hueco". De lo contrario, ACK duplicado de inmediato. |

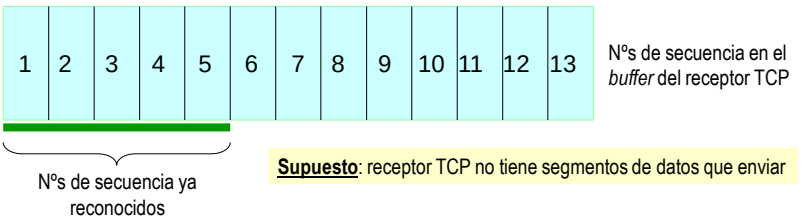
**Supuesto***: El host receptor, su transmisor TCP, no tiene segmentos de datos que enviar, de lo contrario envía los ACK en dichos segmentos sin más, sin regirse por lo indicado en la fila 1 de la tabla.

Universidad de Alcalá

## TCP: Transferencia de datos fiable

20

- Ejemplos para las cuatro filas de la tabla de la diapositiva nº 24.




**Supuesto**: receptor TCP no tiene segmentos de datos que enviar

**Segmentos nºs 1 – 5 recibidos y ya reconocidos**:
Caso fila 1: se recibe Seg. nº 6 ➔ ACK retardado (supuesto que el receptor no tiene datos que enviar).
Caso fila 2: se recibe Seg. nº 7, nº 6 antes recibido pero no reconocido ➔ ACK acumulativo inmediato.
Caso fila 3: se recibe Seg. nº 7, pero no el nº 6, se detecta un "hueco" en nº 6 ➔ ACK duplicado inmediato.
Caso fila 4: respecto al caso anterior, se recibe Seg. nº 6, se "rellena" el "hueco" ➔ ACK acumulativo inmediato.
Caso fila 4: Seg. nº 8 ya recibido. Se recibe Seg. nº 6, se completa parcialmente el "hueco" (borde inferior) ➔ ACK inmediato.
Caso fila 4: Seg. nº 8 ya recibido. Se recibe Seg. nº 7 (no el nº 6), se completa parcialmente el "hueco" ➔ ACK duplicado inmediato.

Universidad de Alcalá
Tema 3: La capa de Transporte

## TCP: Transferencia de datos fiable

21

**TCP: ¿Retroceder N (GBN) o Repetición Selectiva (SR)?**

- TCP utiliza reconocimientos acumulativos.
- Segmentos bien recibidos pero fuera de secuencia NO se reconocen de manera individual por el receptor (ACK duplicados).
  - Emisor sólo necesita mantener los nºs "BaseEmision" y "SigNumSec" (ver Fig. 3.19)
- Muchas implementaciones TCP almacenan en el *buffer* del receptor segmentos fuera de secuencia.
- TCP provee retransmisión selectiva (versión Reno de TCP).

Procedimiento TCP para recuperación de errores puede ser considerado una solución HÍBRIDA entre GBN y SR



## TCP: Control de flujo

22

- Control de flujo: procedimiento para "frenar"– regular la velocidad de emisión de datos.
- Servicio de adaptación de velocidades entre emisor y receptor.
- Objetivo: no desbordar el *buffer* del receptor TCP.
  - Proceso de aplicación en el receptor NO lee los datos del *buffer* en el instante de llegada, hay cierta demora …mayor o menor…
- Importante.
  - Control de flujo NO ES LO MISMO que control de la congestión.
  - Ambos conllevan a "frenar" al emisor, pero se aplican por razones diferentes.

## TCP: Control de flujo

23

Procedimiento TCP para control de flujo

- Supuesto: descarte de segmentos fuera de secuencia.
- Receptor gestiona un "*buffer* de recepción" (*buffer* del receptor).
- Emisor conoce valor de la variable "ventana de recepción".

- Ventana de recepción: proporciona información del espacio libre en el *buffer* del receptor.
  - Tamaño del *buffer* = BufferRecepcion
  - Variables asociadas al *buffer* son:
    - UltimoByteLeido
    - UltimoByteRecibido



## TCP: Control de flujo

24

Procedimiento TCP para control de flujo (II)

- UltimoByteRecibido – UltimoByteLeido ≤ BufferRecepcion
- **VentanaRecepcion** = BufferRecepcion – [UltimoByteRecibido – UltimoByteLeido]
- **VentanaRecepcion**: variable dinámica, varía en el tiempo.

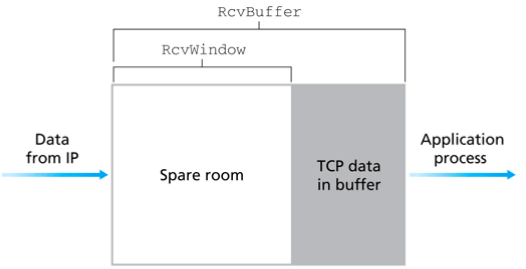



**Figure 3.38** ♦ The receive window (`RcvWindow`) and the receive buffer (`RcvBuffer`)

## TCP: Control de flujo

25

### Procedimiento TCP para control de flujo (III)

- Variable "VentanaRecepcion" y control de flujo.
  - Receptor notifica espacio libre en su *buffer* mediante el campo "ventana de recepción" en cada segmento que transmite. 
  - Receptor inicialmente establece:

    **VentanaRecepcion = BufferRecepcion**
  - Emisor
    - Controla las variables "UltimoByteEnviado" y "UltimoByteReconocido".
      - UltimoByteEnviado – UltimoByteReconocido = Cantidad de datos NO reconocidos.
    - Garantiza, para todo el tiempo de vida de la conexión TCP:
      - UltimoByteEnviado – UltimoByteReconocido ≤ VentanaRecepcion

        Cantidad de datos NO reconocidos



## TCP: Control de flujo

26

### Procedimiento TCP para control de flujo (IV): PROBLEMA.

- Si receptor:
  - *Buffer* lleno → VentanaRecepcion = 0, se notifica al emisor.
  - No tiene nada que enviar → Proceso de aplicación del receptor saca datos del *buffer* pero NO envía segmentos al emisor (ni siquiera ACKs).
    - NO se envían segmentos con nuevos valores de la variable "VentanaRecepcion".
    - Emisor no es informado de la disponibilidad de espacio en el *buffer* del receptor → Emisor bloqueado, no envía datos.
- SOLUCIÓN.
  - Cuando el emisor conoce que el receptor tiene lleno el *buffer* (**VentanaRecepcion = 0**):
    - Debe continuar enviando segmentos con sólo un octeto de datos cada 60 seg.
    - Estos segmentos serán reconocidos por el receptor → Emisor será notificado cuando el receptor disponga de espacio libre en su *buffer* → Emisor no queda bloqueado.
- Ver Applet que ilustra funcionamiento de la "ventana de recepción" en:
  http://media.pearsoncmg.com/aw/aw_kurose_network_4/applets/flow/FlowControl.htm

## TCP: Gestión de la conexión

27

- Gestión de conexión: control de establecimiento y terminación de la conexión TCP.
  - Tras este procedimiento subyacen dos cosas muy importantes:
    - Puede aumentar significativamente el retardo entre sistemas finales.
    - Muchos ataques de red explotan las vulnerabilidades intrínsecas de la gestión de conexión TCP.
  - Establecimiento de la conexión TCP.
    - Tres pasos/fases → Tres segmentos especiales de control, acuerdo en tres fases.
    - Acuerdo/negociación de la conexión en base a los contenidos de las cabeceras TCP obligatorias y opcionales.
  - Terminación de la conexión TCP.
    - Cuatro pasos/fases → Cuatro segmentos especiales de control.*
    - Denominado "cierre normal" de la conexión TCP.



## TCP: Gestión de la conexión


28

Pasos para establecer una conexión TCP

- Paso 1.
  - Cliente.
    - Selecciona de manera aleatoria el nº de secuencia inicial (cliente_nsi).
      - Requiere procedimiento apropiado para evitar ciertos ataques de seguridad.
    - Envía segmento SYN (segmento de solicitud de conexión).
      - Indicador SYN activado → SYN = 1, información de control.
      - Campo nº de secuencia = cliente_nsi.
      - Indicador ACK desactivado → ACK = 0
      - No contiene datos.

## TCP: Gestión de la conexión



29

### Pasos para establecer una conexión TCP

- Paso 2
  - Servidor.
    - Recibe segmento SYN.
    - Si acepta la solicitud asigna *buffer* y variables a la conexión.
      - Vulnerabilidad, ataques por "inundación" de SYN.
      - Ver libro de texto: 5ª e. pág. 254-255; 7ª e. pág.213.
    - Responde, envía segmento "conexión aceptada" (segmento SYN-ACK).
      - Contiene información de control.
        - Campo "Nº de secuencia" = servidor_nsi (seleccionado de manera aleatoria).
        - Campo "Nº de reconocimiento" = cliente_nsi + 1
        - Indicador SYN activado ➔ SYN = 1
        - Indicador ACK activado ➔ ACK = 1
      - No contiene datos.



## TCP: Gestión de la conexión



30

### Pasos para establecer una conexión TCP

- Paso 3
  - Cliente.
    - Recibe segmento SYN-ACK.
    - Asigna *buffer* y variables a la conexión.
    - Envía segmento ACK (reconocimiento de conexión aceptada).
      - Contiene información de control.
        - Campo "Nº de reconocimiento" = servidor_nsi + 1.
        - Indicador SYN desactivado ➔ SYN = 0
        - Indicador ACK activado ➔ ACK = 1
      - Puede o no contener datos.

## TCP: Gestión de la conexión

31

- Pasos para establecer una conexión TCP (ejemplo)
  - Acuerdo en tres fases.
    - Para "sincronizar" a las partes.
  - Parte que inicia la conexión actúa como cliente.
  - Una vez establecida la conexión las partes pueden intercambiar datos mediante "segmentos de datos".
    - Indicador SYN = 0.

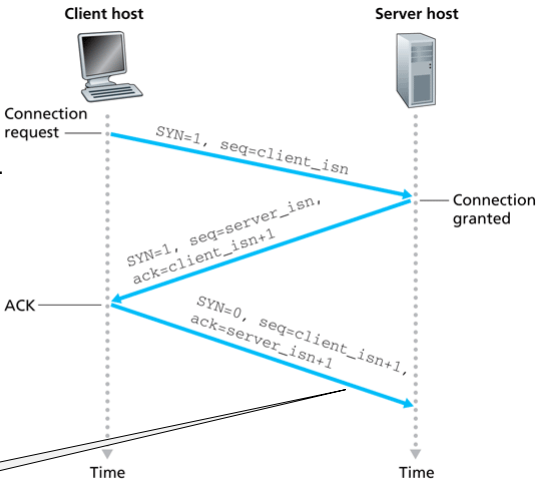



**Figure 3.39** ♦ TCP three-way handshake: segment exchange



## TCP: Gestión de la conexión

32

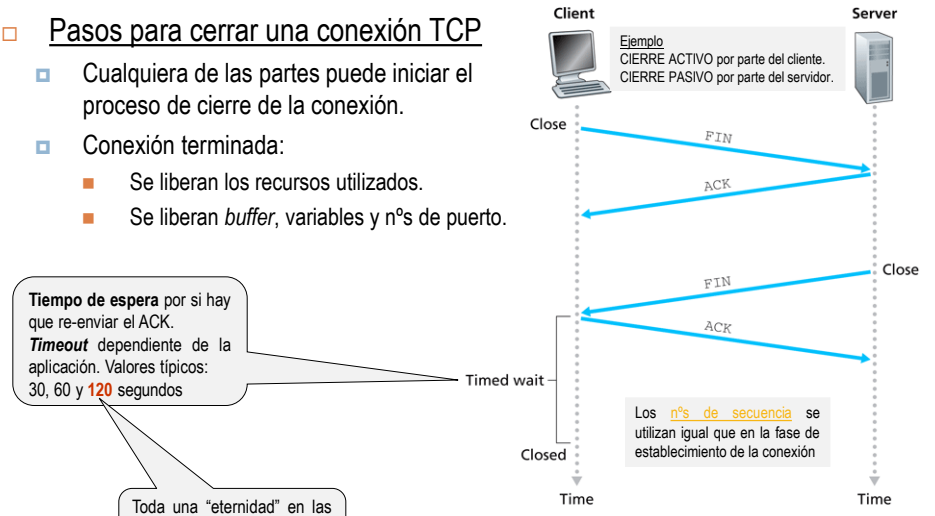
Pasos para cerrar una conexión TCP (cierre normal) Nota

- Paso 1, parte que inicia el proceso de cierre de la conexión:
  - Envía segmento FIN → Indicador FIN activado, FIN = 1
- Paso 2, la otra parte:
  - Envía (devuelve) segmento ACK.
- Paso 3, la otra parte cuando proceda:
  - Envía segmento FIN → Indicador FIN activado, FIN = 1.
- Paso 4, parte que inició el proceso de cierre de la conexión.
  - Envía (devuelve) segmento ACK en respuesta al segmento FIN recibido.

## TCP: Gestión de la conexión

33

- Pasos para cerrar una conexión TCP
  - Cualquiera de las partes puede iniciar el proceso de cierre de la conexión.
  - Conexión terminada:
    - Se liberan los recursos utilizados.
    - Se liberan *buffer*, variables y nºs de puerto.




**Tiempo de espera** por si hay que re-enviar el ACK.
***Timeout*** dependiente de la aplicación. Valores típicos: 30, 60 y 120 segundos

Toda una "eternidad" en las redes actuales. Establecido en el estándar inicial.

**Figure 3.40** ♦ Closing a TCP connection



## TCP: Gestión de la conexión

34

Estados TCP en el **lado cliente**. Lado cliente inicia el cierre de la conexión

Este diagrama sólo muestra el establecimiento y cierre **normal** de una conexión TCP.
Otros escenarios más complejos se pueden presentar en la práctica. Ver fuente sugerida en el libro de texto (5º e, pág. 252; 7ª e, pág. 212).

Este diagrama corresponde con la **figura 3.40** del libro de texto, donde se muestran explícitamente los cuatro pasos para el cierre de la conexión TCP

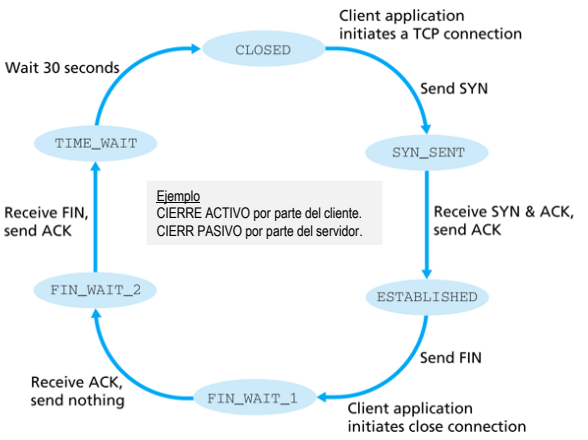



**Figure 3.41** ♦ A typical sequence of TCP states visited by a client TCP

## TCP: Gestión de la conexión



35

Estados TCP en el **lado servidor**. Lado cliente inicia el cierre de la conexión

Este diagrama sólo muestra el establecimiento y cierre **normal** de una conexión TCP.
Otros escenarios más complejos se pueden presentar en la práctica. Ver fuente sugerida en el libro de texto (5º e, pág. 252; 7ª e, pág. 212).

Este diagrama corresponde con la **figura 3.40** del libro de texto, donde se muestran explícitamente los cuatro pasos para el cierre de la conexión TCP

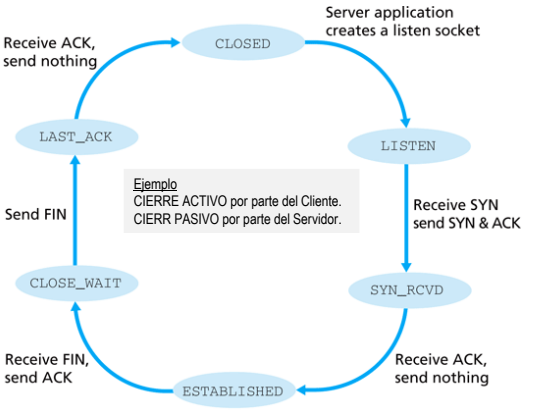



**Figure 3.42** ♦ A typical sequence of TCP states visited by a server-side TCP



## TCP: Transmission Control Protocol

36

- Erratas en el libro de texto (5ª edición)
  - Página 252, primer párrafo, tercer renglón:
    - Dice: " …cliente TCP…"
    - Debe decir: "…servidor TCP…"
- Erratas en el libro de texto (7ª edición)
  - Página 211, 2º párrafo, 7º renglón al final:
    - Dice: " …cliente TCP…"
    - Debe decir: "…servidor TCP…"