Capítulo 3: Capa Transporte - III

ELO322: Redes de Computadores

Agustín J. González

Este material está basado en:

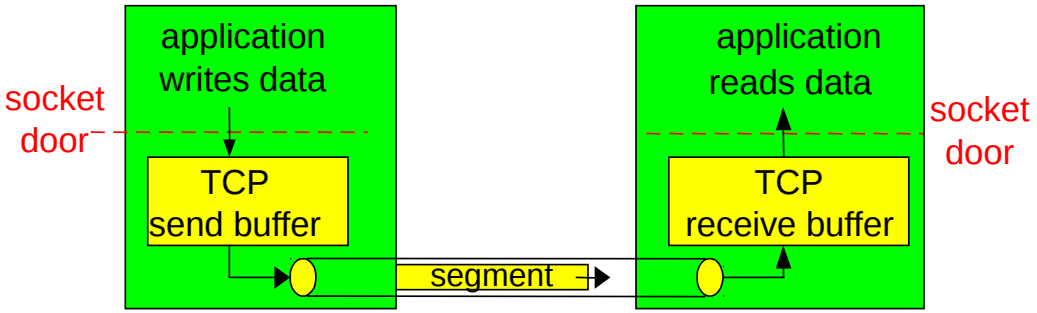
- Material de apoyo al texto *Computer Networking: A Top Down Approach Featuring the Internet.* Jim Kurose, Keith Ross.

Capítulo 3: Continuación

- 3.1 Servicios de la capa transporte
- 3.2 Multiplexing y demultiplexing
- 3.3 Transporte sin conexión: UDP
- 3.4 Principios de transferencia confiable de datos
- 3.5 Transporte orientado a la conexión: TCP
  - Estructura de un segmento
  - Transferencia confiable de datos
  - Control de flujo
  - Gestión de la conexión
- 3.6 Principios del control de congestión
- 3.7 Control de congestión en TCP

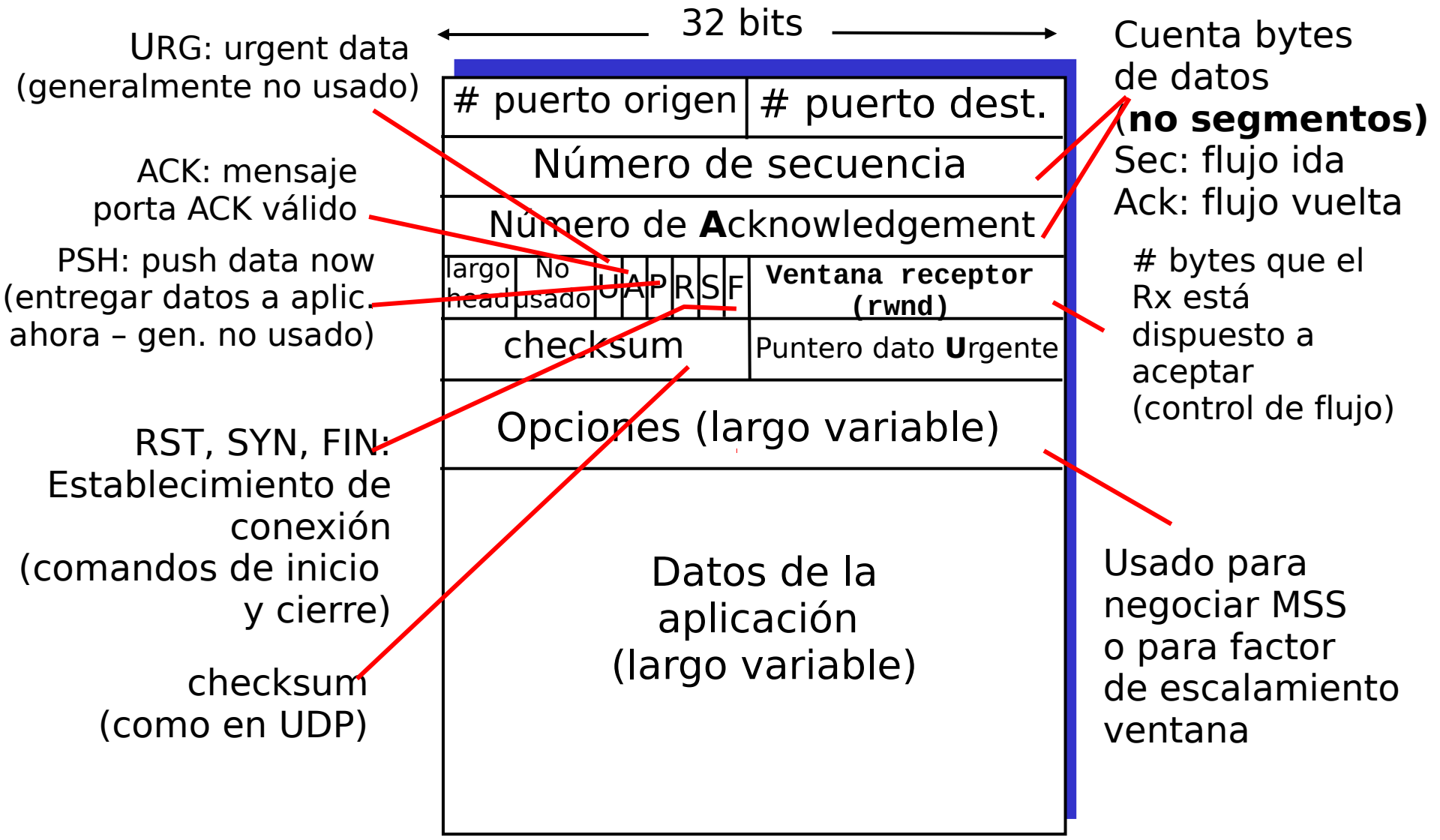
TCP: Generalidades RFCs: 793, 1122, 1323, 2018, 2581

- Es una comunicación Punto-a punto:
  - Un Tx y un Rx
- *flujo de bytes* confiable y en orden*:*
  - No hay “límites del inicio/término de mensaje”
- Usa pipeline:
  - El tamaño de la ventana TCP es definido por el control de congestión y control de flujo
- *Usa buffer en Tx & Rx*



- Transferencia full duplex (dos sentidos):
  - Flujo de datos bidireccionales en la misma conexión
  - Maximum segment size (MSS), depende del Maximum Transmission Unit de la capa enlace
- Orientado a la conexión:
  - Handshaking (intercambio de mensajes de control) inicializa al Tx y Rx antes del intercambio de datos
- Tiene control de flujo:
  - Tx no sobrecargará al Rx
- También tiene control de congestión
  - No sobrecargar la ruta

Estructura de un segmento TCP



¿Cómo se determina el tamaño de un segmento TCP?

- TCP no incluye campo Length (largo), como UDP.
- Para determinar el tamaño de un segmento, TCP requiere información del encabezado IP, el cual incluye el campo “Total Length” de su datagrama.
- TCP determina el tamaño sustrayendo el tamaño del encabezado IP del largo total del datagrama IP.

Número de Sec. y ACKs en TCP

Número de Sec.:

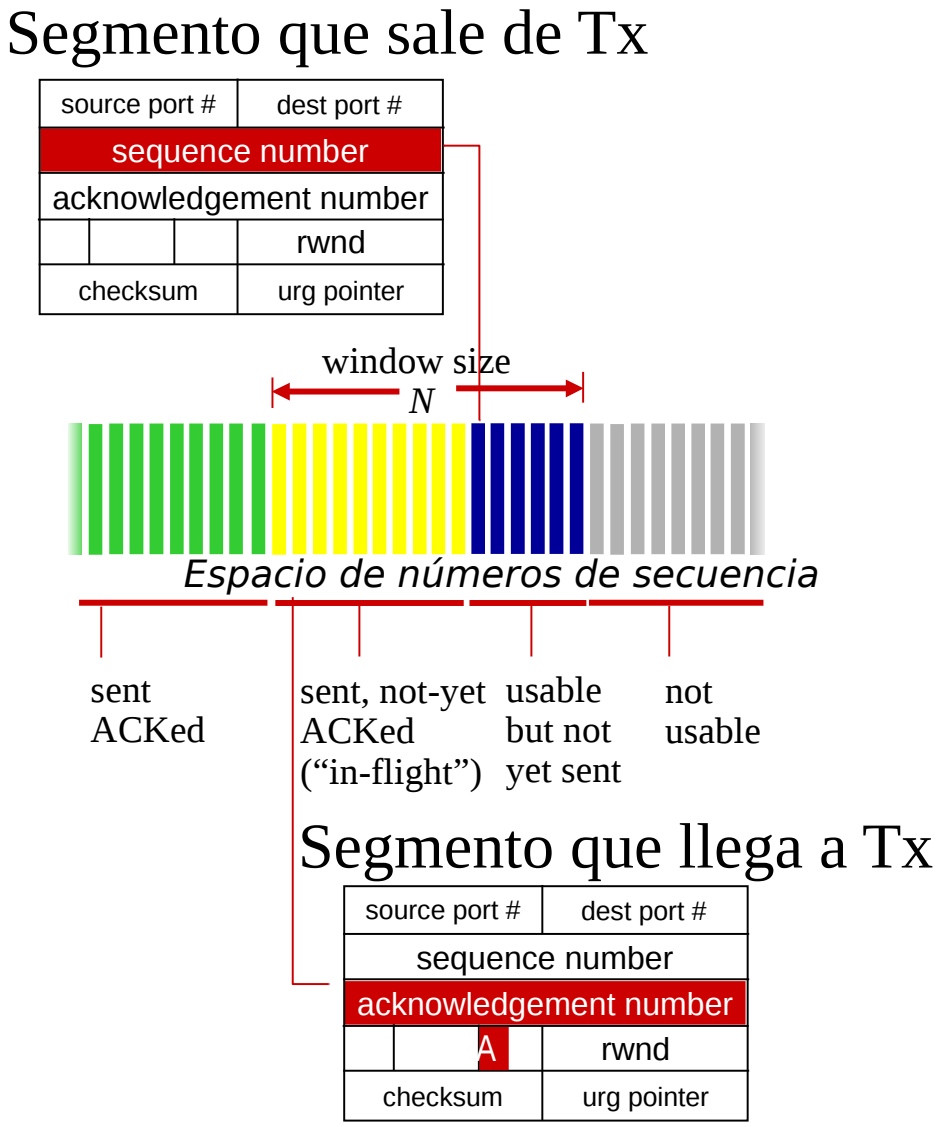
- "número" del byte dentro del flujo correspondiente al primer byte del segmento de datos

ACKs:

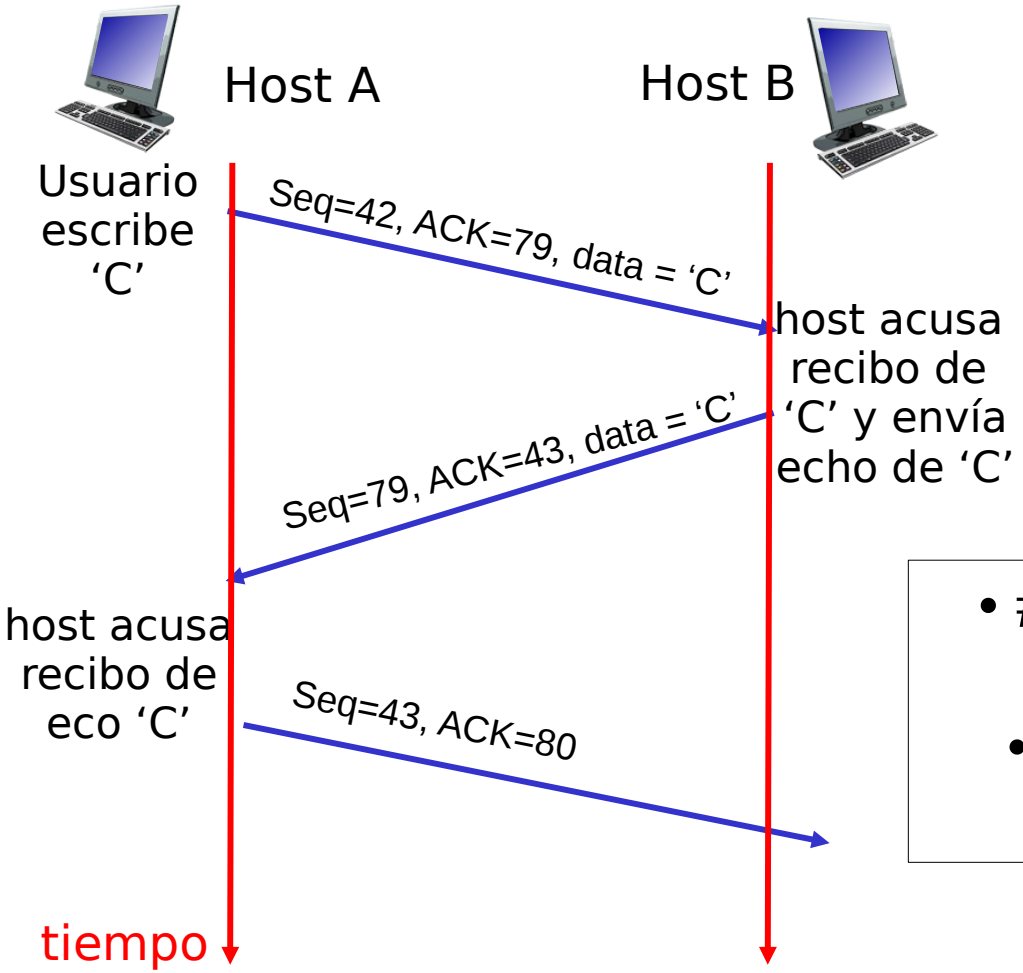
- # sec. del próximo byte esperado desde el otro lado
- ACK es acumulativo

Q: ¿Cómo el receptor maneja segmentos fuera de orden?

- la especificación de TCP lo deja a criterio del implementador



Ejemplo: Uso de # de sec. y ACKs en Telnet (Aplicación sobre TCP)



- # Sec: número para 1° byte del segmento
- ACK: próximo byte esperado en Rx

Escenario telnet simple
Con conexión ya establecida

Round-Trip Time y Timeout en TCP

Q: ¿Cómo fijar valor de timeout en TCP?

- Mayor que RTT
  - pero RTT varía
- Muy corto: timeout prematuro
  - Retransmisiones innecesarias
- Muy largo: lenta reacción a pérdidas de segmentos

Q: ¿Cómo estimar el RTT?

- **SampleRTT**: mide tiempo desde tx del segmento hasta recibo de ACK
  - Ignora retransmisiones
- **SampleRTT** varía, hay que suavizar el RTT estimado
  - Promediar varias medidas recientes, no considerar sólo el último **SampleRTT**
- **Estimación de RTT no considera tamaño de paquetes.**

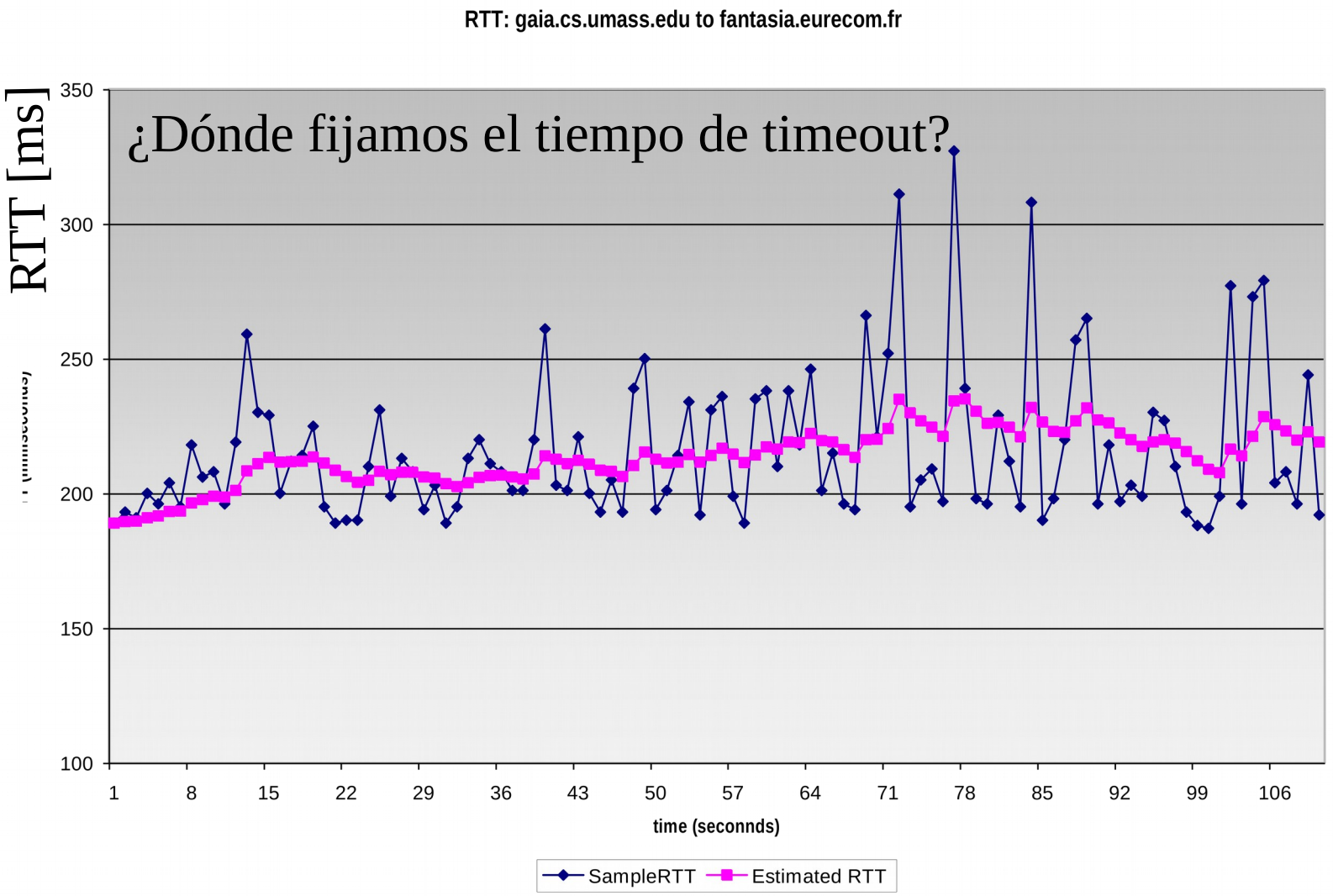
Round-Trip Time y Timeout en TCP

$$\mathbf{EstimatedRTT_i = (1-\alpha)*EstimatedRTT_{i-1} + \alpha*SampleRTT_i}$$

- Promedio móvil ponderado exponencial
- Influencia de las muestras pasadas decrece exponencialmente rápido

  Ejercicio: anote $EstimatedRTT_i$ en función $SampleRTT_1..SampleRTT_i$
- Valor típico: $\alpha = 0.125 = (1/8) = 3$ right shifts. (esto es histórico, hoy un right shift tiene costo similar a una multiplicación)

Ejemplo de estimación de RTT:



Timeout en TCP

- Timeout usa **EstimatedRTT** más un "margen de seguridad"
  - Si hay gran variación en **EstimatedRTT =>** usar gran margen
- Primero estimamos cuánto se desvía el SampleRTT del EstimatedRTT:

$$DevRTT_i = (1-\beta) * DevRTT_{i-1} + \beta * |SampleRTT_i - EstimatedRTT_i|$$

No es desviación estándar, pero es más rápido de calcular.

(típicamente, $\beta = 0.25$)

Entonces TCP fija el timeout como:

$$TimeoutInterval_i = EstimatedRTT_i + 4 * DevRTT_i$$

RTT estimado

Margen de "seguridad"

¿Por qué en la estimación de RTT, TCP omite la medición de SampleRTT de segmentos retransmitidos?

?

- Porque el segmento original podría no haberse perdido y su ACK sólo esté retrasado, luego al enviar la retransmisión, la llegada del ACK podría corresponder al ACK retrasado. La medición de SampleRTT sería errada en este caso; al no distinguir en qué caso estamos, TCP decide no considerar esa medición.

Capítulo 3: Continuación

- 3.1 Servicios de la capa transporte
- 3.2 Multiplexing y demultiplexing
- 3.3 Transporte sin conexión: UDP
- 3.4 Principios de transferencia confiable de datos
- 3.5 Transporte orientado a la conexión: TCP
  - Estructura de un segmento
  - Transferencia confiable de datos
  - Control de flujo
  - Gestión de la conexión
- 3.6 Principios del control de congestión
- 3.7 Control de congestión en TCP

Transferencia confiable de datos en TCP

- TCP crea un servicio de transferencia confiable sobre el servicio no confiable de IP
  - Usa envío de segmentos en pipeline
  - **ACKs acumulativos** como Go-Back-N
  - TCP usa un **timer único** de retransmisión, como Go-Back-N
- Retransmisiones son activadas por:
  - Eventos de timeout
  - ACKs duplicados (distinto a GBN y SR)
- Se retransmite paquete más antiguo sin ACK (sólo 1, como en selective repeat).

Inicialmente consideremos un Tx TCP simplificado:
* Ignora ACKs duplicados
* Ignora control de flujo y control de congestión

TCP eventos en transmisor:

Dato recibido de aplicación:

- crea segmento con #sec.
- #sec. Es # del primer byte de datos del segmento según su # en el flujo
- Iniciar timer si no está ya corriendo
  - Pensar el timer como aquel del segmento más antuguo sin ACK
  - Intervalo de expiración: **`TimeOutInterval`**

timeout:

- retransmitir segmento que causó timeout
- reiniciar timer

ack recibido:

- if ack da acuse de recibo a segmento previo sin ACK
  - Actualizar conocimiento de segmentos con ACK recibido
  - iniciar timer si hay aún segmentos sin ACK

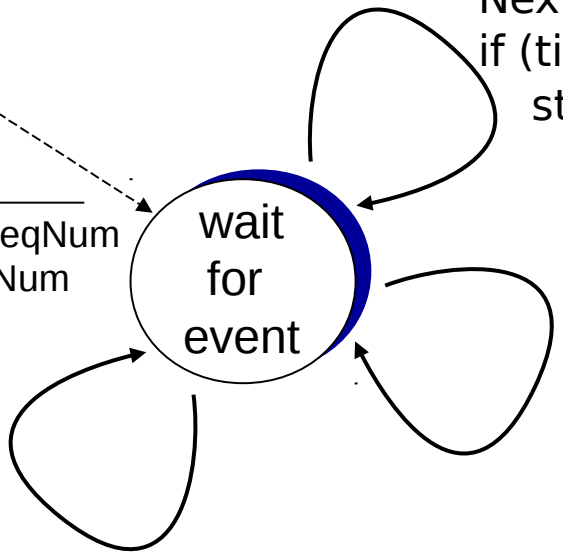
TCP transmisor (simplificado)

data recibida desde aplicación

crea segmento, seq. #: NextSeqNum
pasa segmento a IP (i.e., “send”)
NextSeqNum = NextSeqNum + length(data)
if (timer no está corriendo)
  start timer

Λ

NextSeqNum = InitialSeqNum
SendBase = InitialSeqNum



timeout

retransmitir segmento sin
  acuse de recibo con # sec,
  más pequeño
start timer

ACK recibido, con campo ACK y

```
if (y > SendBase) {
   SendBase = y
   /* SendBase–1: último byte con ACK acumulativo */
   if (hay segmentos sin acuse de recibo)
        start timer
     else stop timer
    }
```

Equivalente a lo previo pero en código

```
NextSeqNum = InitialSeqNum
SendBase = InitialSeqNum

loop (forever) {
  switch(event)

    event: datos recibidos desde la aplicación
          Crear segmento TCP con número de sec. NextSeqNum
          if (timer actualmente no está corriendo)
                iniciar timer
          pasar segmento a IP (capa red)
          NextSeqNum = NextSeqNum + length(data)
          break;

    event: timeout del timer
          retransmitir segmento con menor # de sec. sin acuse
          iniciar timer
          break;

    event: recepción de ACK con campo ACK de valor x
          if (x > SendBase) {
              SendBase = x
              if (hay segmentos sin acuse de recibo aún)
                    iniciar timer
              else detener timer
           }
 }  /* end of loop forever */
```

Tx TCP (simplificado)

Comentarios:

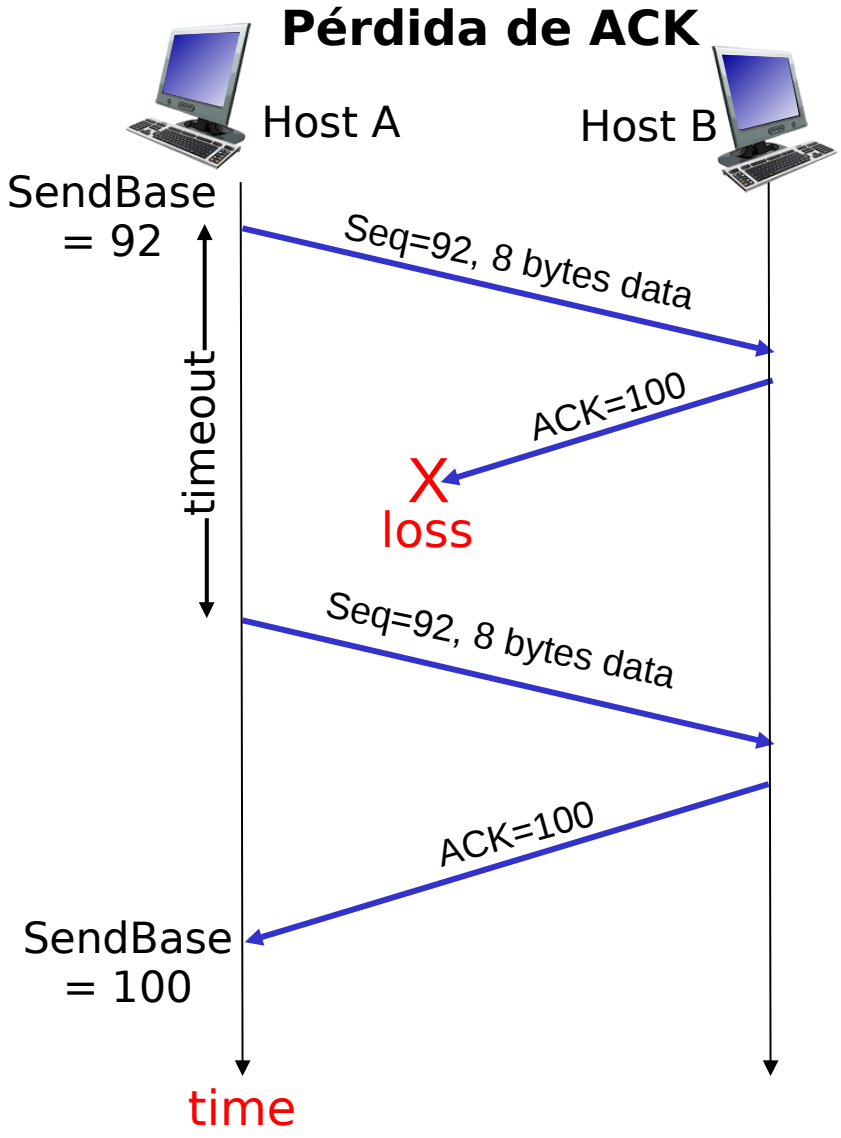
- SendBase: Byte más antiguo sin ACK
- SendBase-1: último Byte con ACK recibido

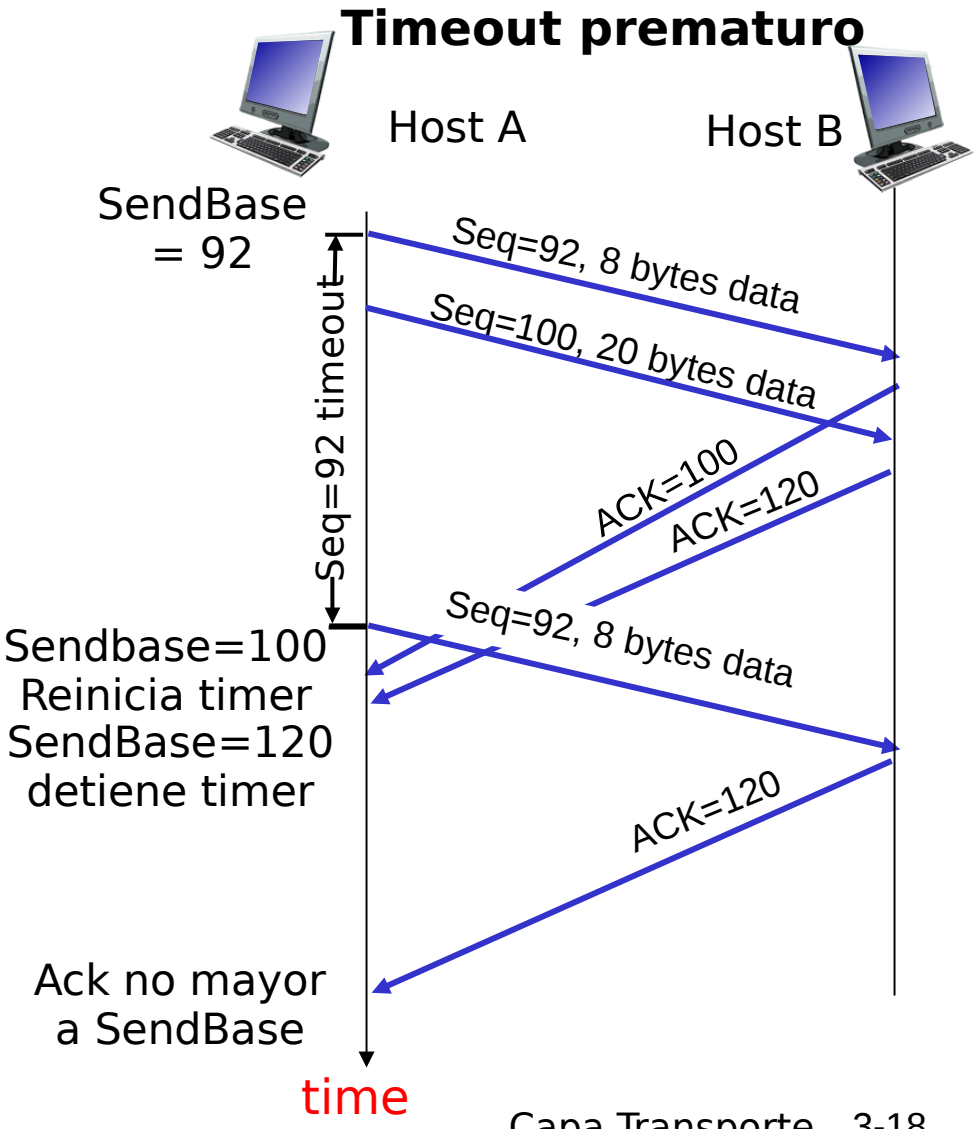
Ejemplo:

- SendBase = 71 y se recibe ACK con x = 72
- El Rx quiere nuevos bytes con seq = 72
- Como x > SendBase, llegó acuse de recibo de dato (seq = 71)

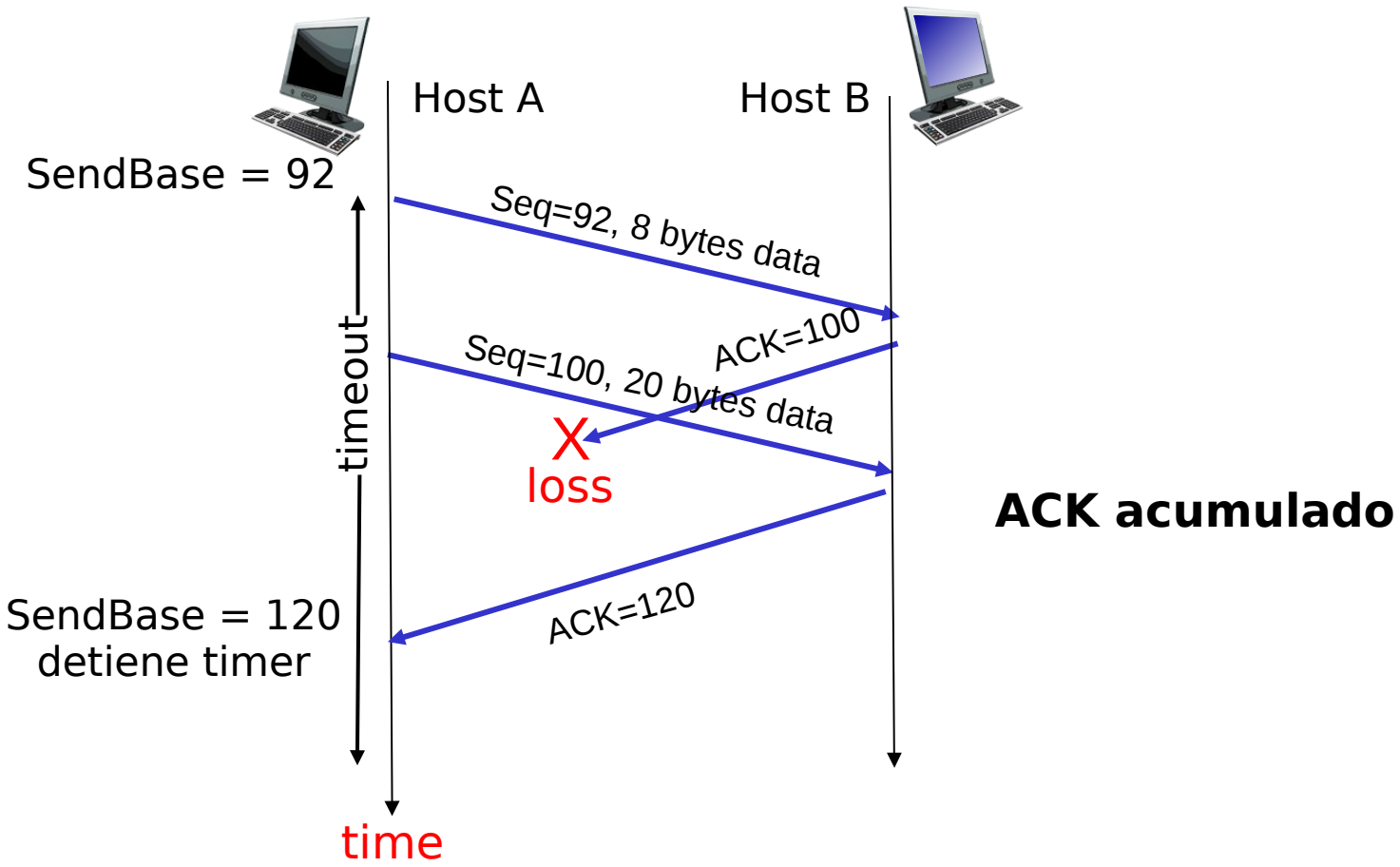
Notar que evento de timeout envía sólo el paquete más antiguo.

TCP: escenarios de retransmisión





TCP escenarios (más)



Estos diagramas no reflejan tiempos de transmisión ni almacenamiento y reenvío en la ruta.

Generación de ACK en TCP [RFC 1122, RFC 2581]

Notar efecto en RTT

Evento en Receptor	TCP acción del receptor
Llegada de segmento en orden con # sec. esperado. Ya se envió el ACK de todo lo previo.	ACK retardado. Espera hasta 500ms por próximo segmento. Si no llega otro segmento, enviar ACK
Llegada de segmento en orden con # sec. esperado. Algún segmento tiene ACK pendiente	Enviar inmediatamente un ACK acumulado se da acuse así a ambos segmentos en orden.
Llegada de segmento fuera de orden con # sec. mayor al esperado. Se detecta un vacío.	Enviar inmediatamente un ACK duplicado, indicando # sec. del próximo byte esperado
Llegada de segmento que llena el vacío parcialmente o completamente	Enviar inmediatamente un ACK si es que el segmento se ubica al inicio del vacío de segmentos recibidos

¿Cuál es el propósito de enviar ACK retardados en TCP?

- TCP envía ACK retardados para reducir el número de ACKs cuando el canal de receptor a transmisor no requiere enviar datos de regreso (recordar que cada conexión TCP es bidireccional). Retardar el envío de ACK permite mejorar la opción de enviar el ACK en un paquete de datos o enviar un ACK acumulado, mejorando el uso de los recursos de la red.

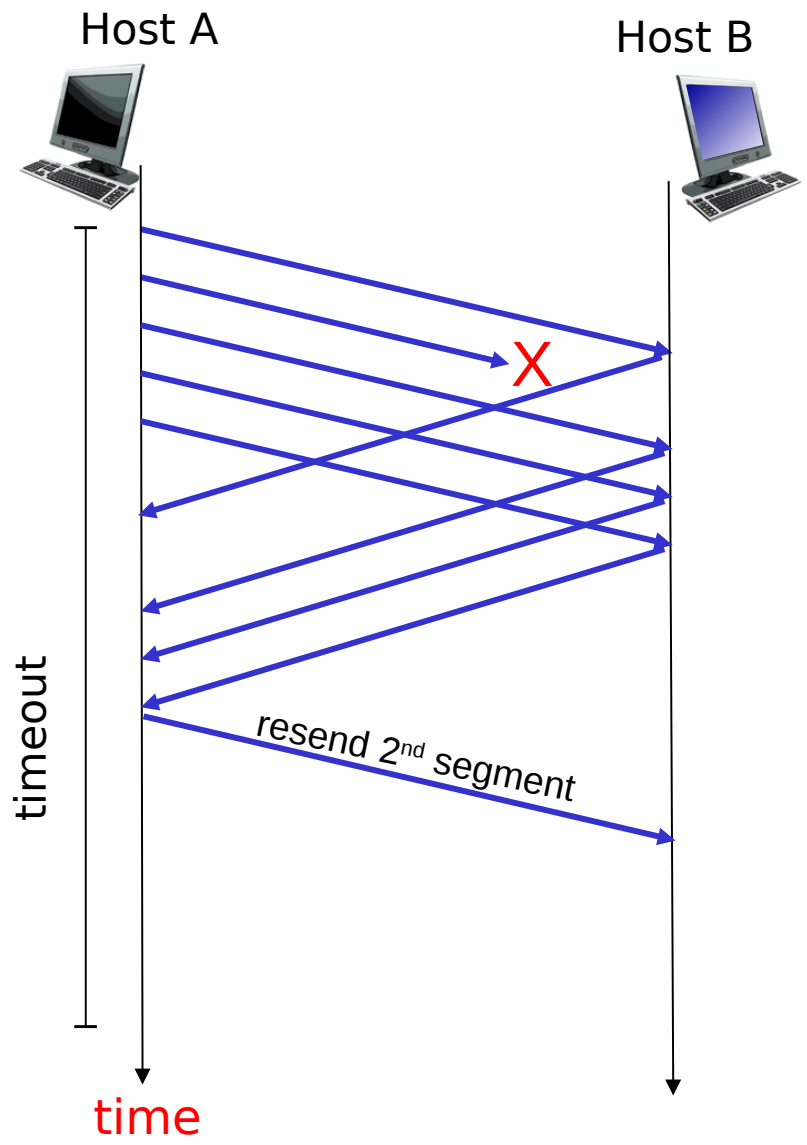
TCP: Retransmisiones Rápidas: no ignoremos ACK duplicados

- Periodo de Time-out es a menudo largo:
  - Retardo largo antes de re-envío de paquetes perdidos
- Se puede detectar paquetes perdidos vía ACKs duplicados.
  - Tx a menudo envía muchos segmentos seguidos
  - Si un segmento se perde, probablemente habrá muchos ACKs duplicados.

Retransmisiones rápidas

- Si Tx recibe 3 ACKs duplicados, éste supone que el segmento después de este dato se perdió:
  - Retransmisiones rápidas: reenviar el segmento antes que el timer expire.

Retransmisiones rápidas



- Los protocolos Stop-and-Wait, Go-Back-N y Selective repeat no obligan a usar un rango de números de secuencia mucho mayor que el tamaño de venta.
- Esto hace que estos protocolos fallen cuando el orden de paquetes cambia en la red.
- TCP usa un rango de números de secuencia (campo de 32 bits) mucho mayor que el tamaño de ventana (campo de 16 bits, que puede tener factor multiplicativos).
- Por lo previo mientras re-envío luego de una ACK duplicado puede conducir a error en los primeros protocolos, no ocurre así en TCP.

TCP: Algoritmo de Retransmisión Rápida

```
event: Llega ACK, con campo ACK de valor x
        if (x > SendBase) {
            SendBase = x
            if (hay segmentos sin acuse de recibo aún)
                iniciar timer
            else detener timer
        }
        else {  // x == SendBase
            incrementar cuenta de ACKs de x duplicados
            if (cuenta de ACKs de x duplicados == 3) {
                re-enviar segmento con # de secuencia x
                iniciar timer
            }
```

ACK duplicado de un segmento con ACK recibido

Retransmisión rápida

¿Cuándo se genera una retransmisión rápida en TCP?

?

- Cuando se recibe un tercer ACK duplicado.

TCP: Timeout

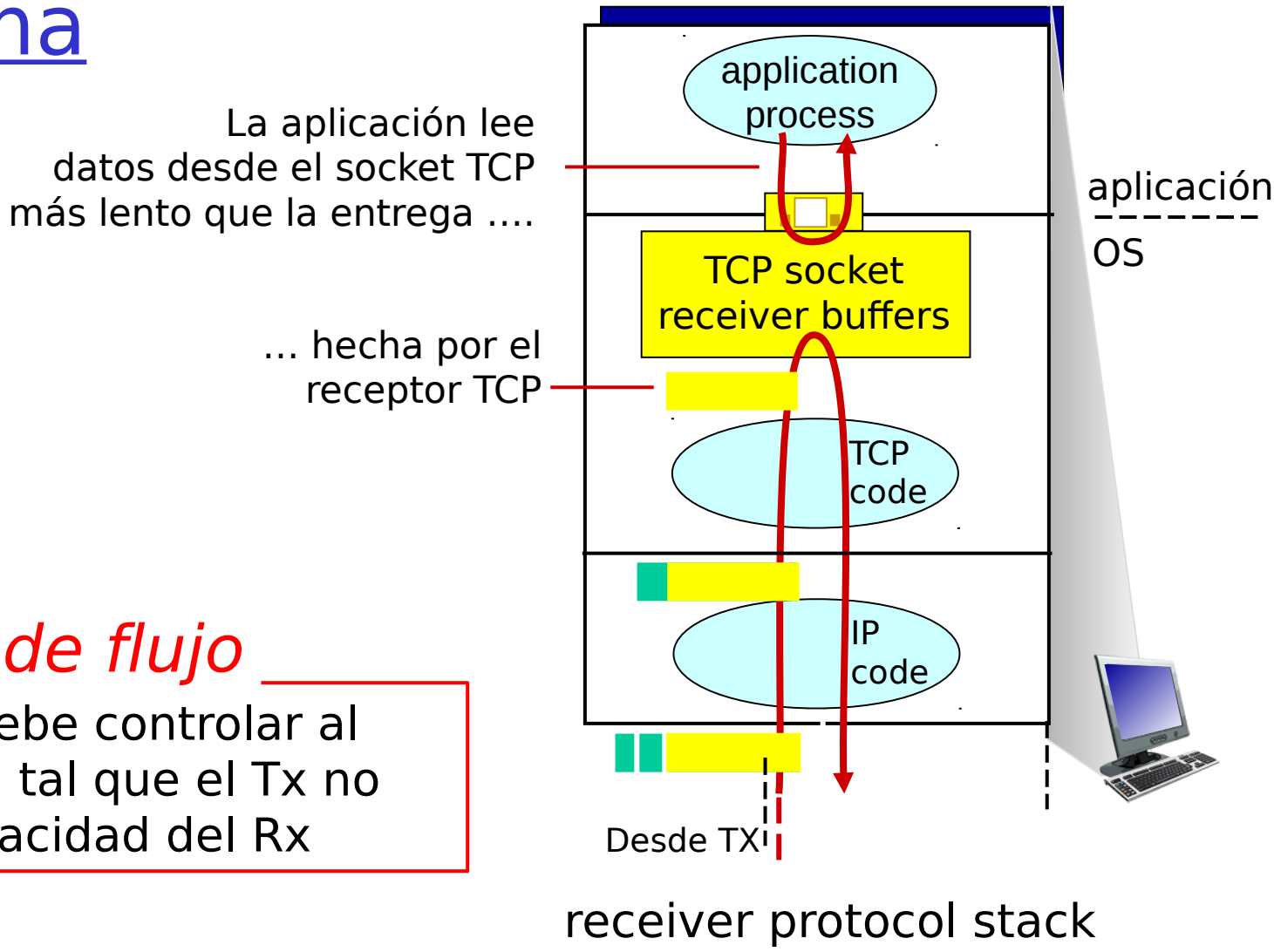
Duplicando el tiempo del timeout

- Algunas modificaciones en muchas implementaciones de TCP:
  - La primera concierne al largo del intervalo de timeout después que el timer expira
    - En esta modificación cuando ocurre un timeout, TCP retransmite el segmento sin ACK con menor número de secuencia pero por cada retransmisión TCP duplica el valor previo de TimeoutInterval
    - De esta forma los intervalos crecen exponencialmente después de cada retransmisión sucesiva.
  - La segunda es si se recibe un ACK no duplicado entonces se recalcula TimeoutInterval usando EstimatedRTT y DevRTT
- Esta es una forma limitada de control de congestión

Capítulo 3: Continuación

- 3.1 Servicios de la capa transporte
- 3.2 Multiplexing y demultiplexing
- 3.3 Transporte sin conexión: UDP
- 3.4 Principios de transferencia confiable de datos
- 3.5 Transporte orientado a la conexión: TCP
  - Estructura de un segmento
  - Transferencia confiable de datos
  - Control de flujo
  - Administración de conexión
- 3.6 Principios del control de congestión
- 3.7 Control de congestión en TCP

TCP control de flujo: Origen del problema

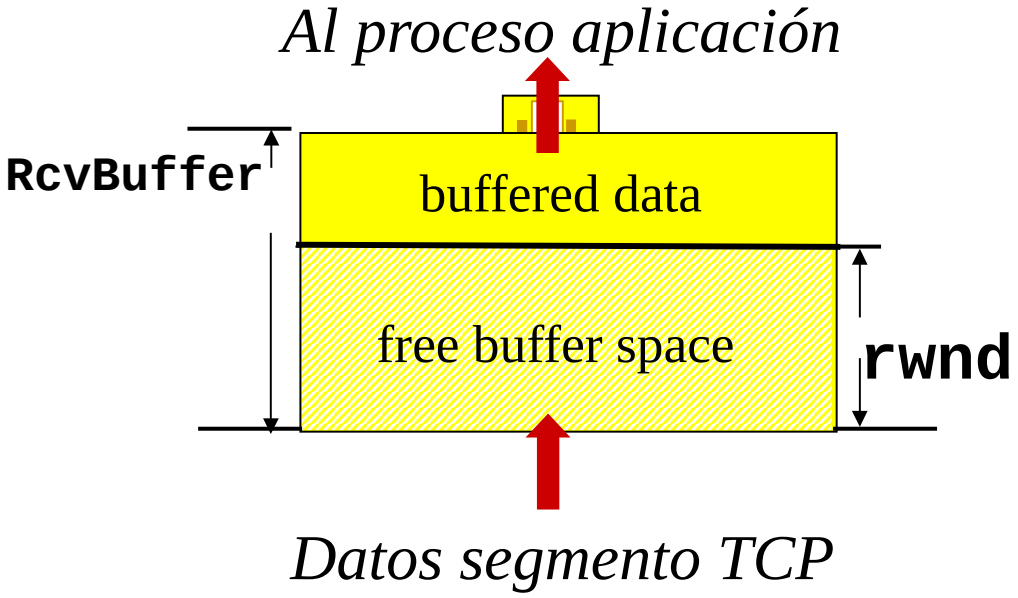


Control de flujo

Receptor debe controlar al transmisor, tal que el Tx no supere capacidad del Rx

Control de flujo en TCP: Cómo trabaja?

- Rx comunica el espacio libre del buffer a través del valor de **rwnd** en los segmentos
- Así el transmisor limita datos en tránsito (sin ACK) a **rwnd**
- Esto garantiza que el buffer del Rx no se rebalse (overflow)



Control de flujo en TCP: Cómo trabaja

- El transmisor debe tomar en cuenta los segmentos en tránsito
- Luego el número de bytes que el Tx puede enviar es en general menor que el anunciado por la RevWindows.
- ¿Cuál es la expresión para el número de Bytes posibles de enviar sin colapsar al receptor?

¿Cuál es la función principal o propósito del control de flujo?

?

Impedir que el transmisor envíe más datos que los almacenables en el buffer del receptor.

Cuando el transmisor de una conexión TCP está a punto de enviar un segmento con número de secuencia 773, recibe un acuse de recibo con numeración 123 y ventana de recepción 1300. ¿Cuántos bytes como máximo puede transportar el segmento que está a punto de enviar? ?

- Los bytes 123 hasta 772 inclusive (650 bytes) están en tránsito para el valor de ventana de recepción 1300. Es así como podemos asegurar que el receptor podrá almacenar 1300-650 =650 bytes, éste es el número máximos de bytes a transportar en el próximo segmento.

Capítulo 3: Continuación

- 3.1 Servicios de la capa transporte
- 3.2 Multiplexing y demultiplexing
- 3.3 Transporte sin conexión: UDP
- 3.4 Principios de transferencia confiable de datos
- 3.5 Transporte orientado a la conexión: TCP
  - Estructura de un segmento
  - Transferencia confiable de datos
  - Control de flujo
  - Administración de conexión
- 3.6 Principios del control de congestión
- 3.7 Control de congestión en TCP

Administración de Conexión en TCP

Recordar: Transmisor y receptor TCP establecen una “conexión” antes de intercambiar segmentos de datos

- TCP inicializa variables:
  - # de secuencia
  - buffers, información de control de flujo (e.g. **`RcvWindow`**)
- *cliente:* Iniciación de conexión **`clientSocket.connect((serverName,serverPort))`**
- *server:* contactado por cliente **`connectionSocket, addr = serverSocket.accept()`**

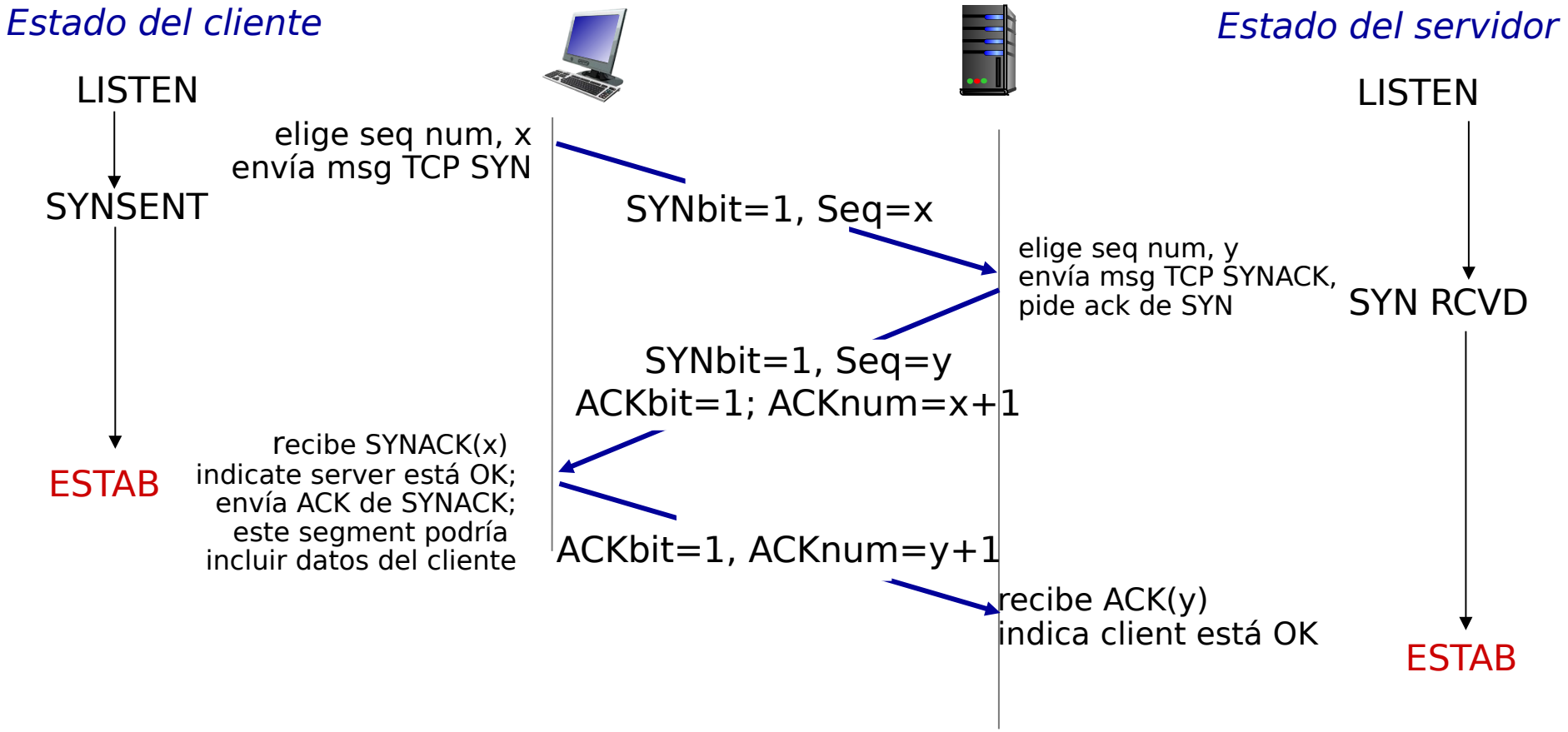
Saludo de manos de tres vías (Three way handshake):

Paso 1: host cliente envía segmento TCP SYN al servidor
- Informa # secuencia inicial
- no data

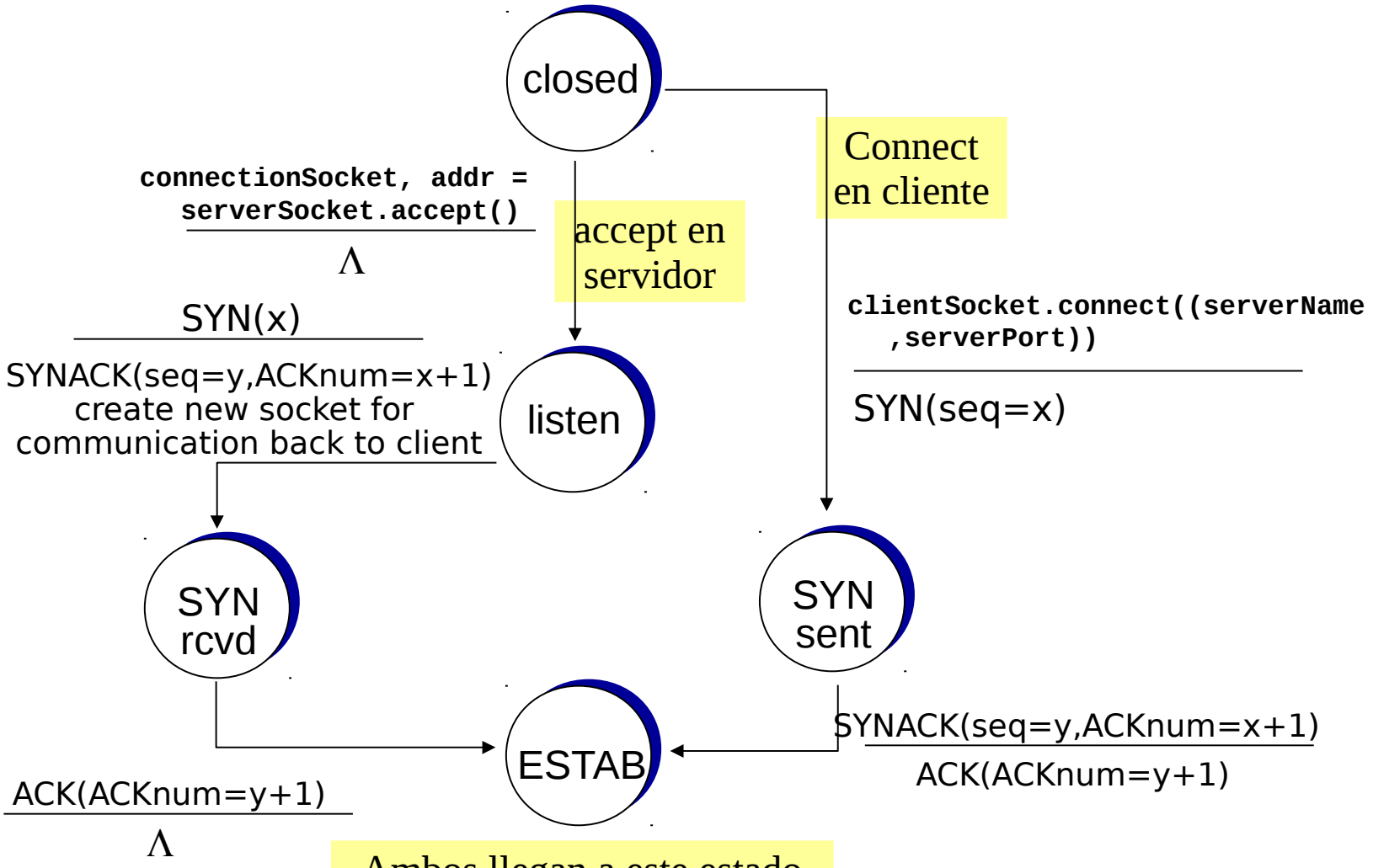
Paso 2: host servidor recibe SYN, responde con segmento SYN & ACK
- Servidor ubica buffers
- Informa su # secuencia inicial
- Informa RcvWindow

Paso 3: cliente recibe SYN & ACK, responde con segmento ACK, el cual podría contener datos.

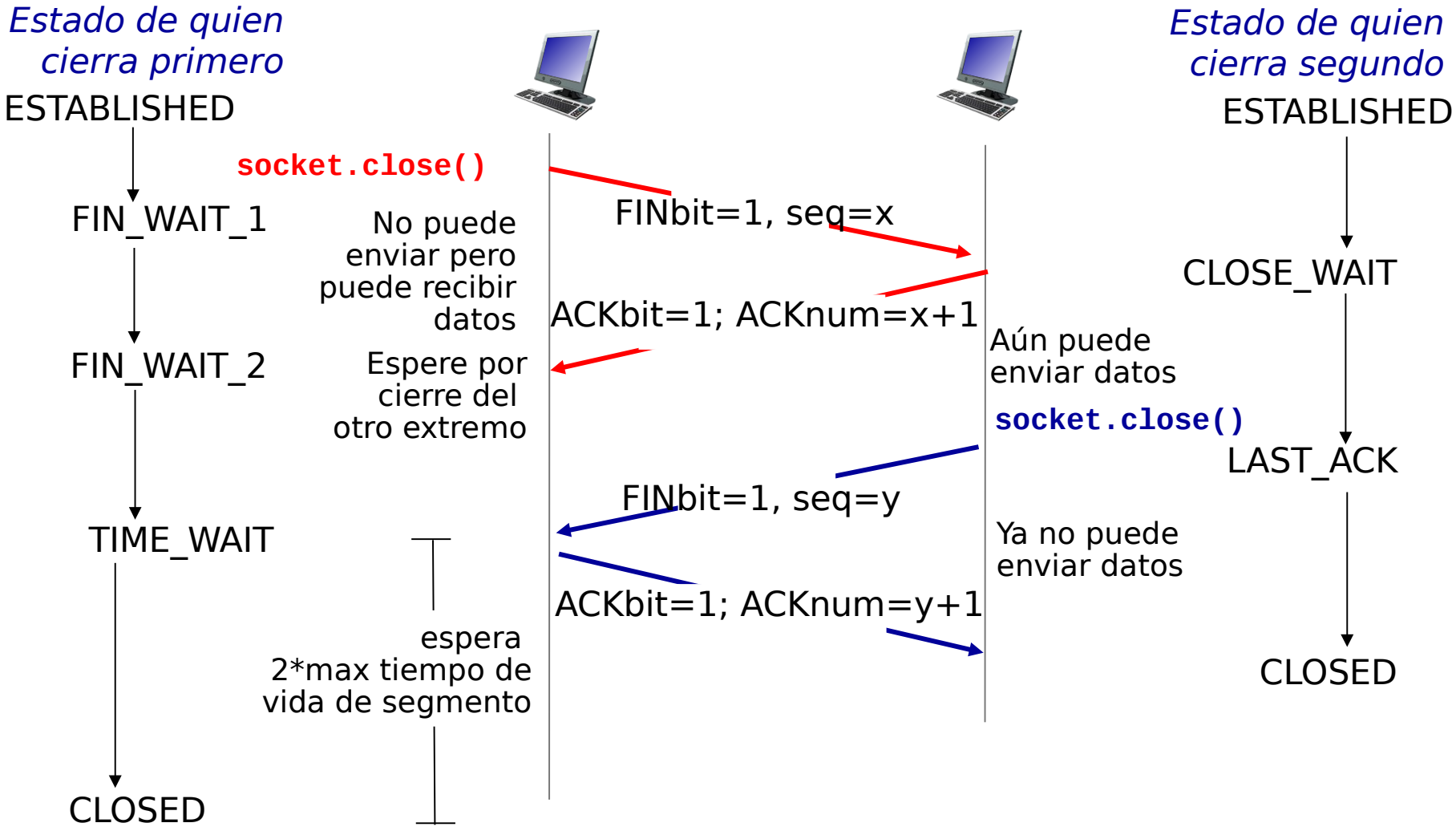
TCP 3-way handshake



TCP 3-way handshake: FSM (simple)

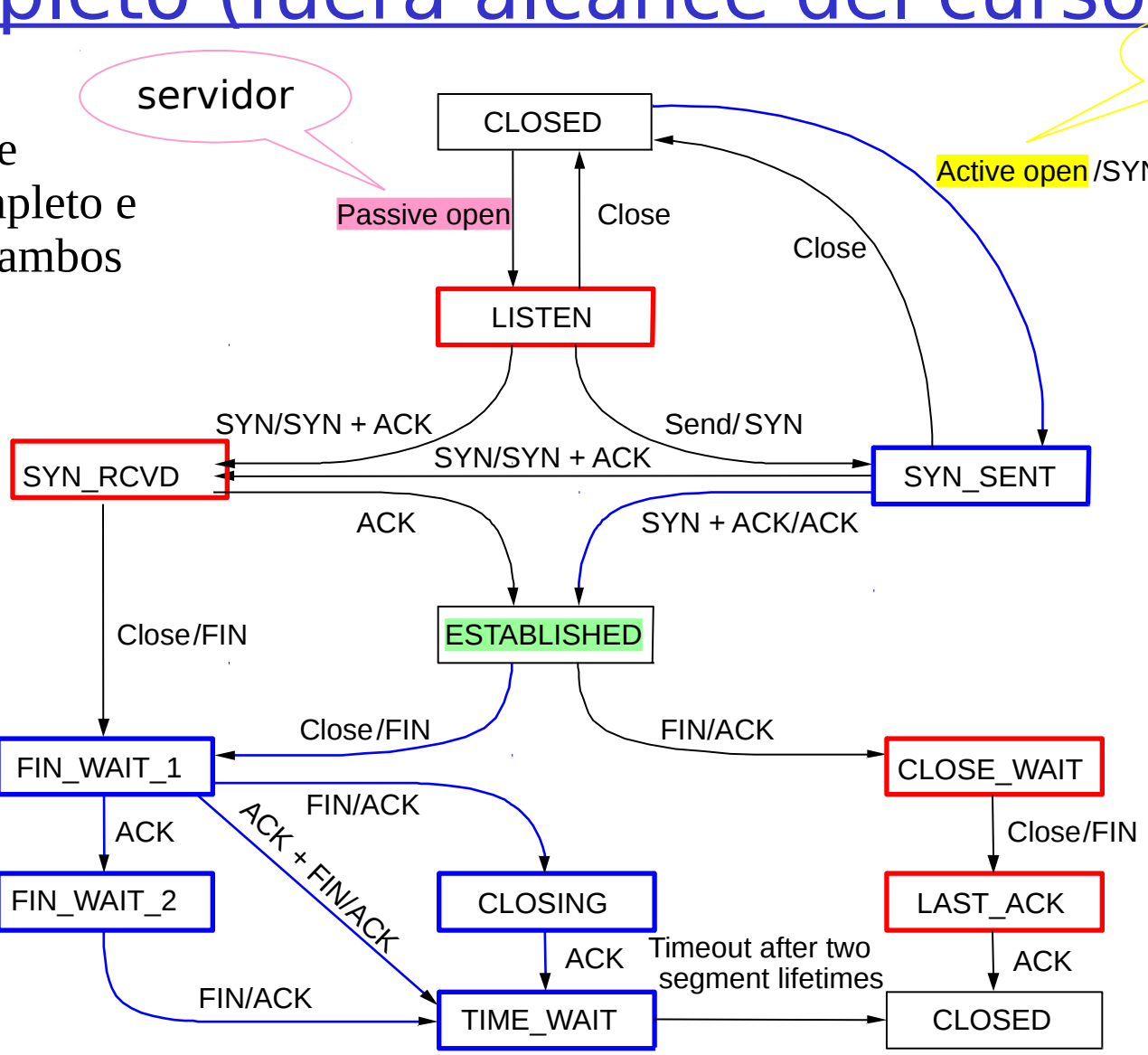


TCP: Cierre de conexión



Administración de la Conexión TCP Completo (fuera alcance del curso)

Diagrama de estados completo e incluyendo ambos casos



Close llega desde capa aplicación

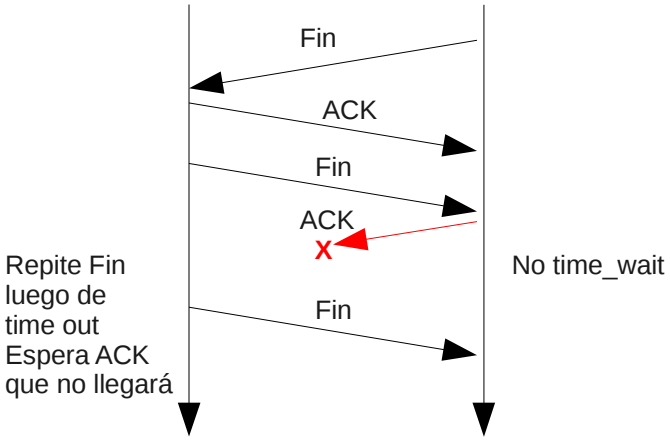
Quien cierra primero

Quien cierra segundo

En una conexión TCP uno de los extremos espera un tiempo (time_wait) luego de enviar su último segmento. Usando un diagrama temporal de intercambio de mensajes muestre y explique el problema que se presentaría si se decidiera no esperar ese time_wait.

?

- Varias situaciones inconvenientes pueden ocurrir, una de ellas se muestra abajo.



Capítulo 3: Continuación

- 3.1 Servicios de la capa transporte
- 3.2 Multiplexing y demultiplexing
- 3.3 Transporte sin conexión: UDP
- 3.4 Principios de transferencia confiable de datos
- 3.5 Transporte orientado a la conexión: TCP
  - Estructura de un segmento
  - Transferencia confiable de datos
  - Control de flujo
  - Administración de conexión
- 3.6 Principios del control de congestión
- 3.7 Control de congestión en TCP