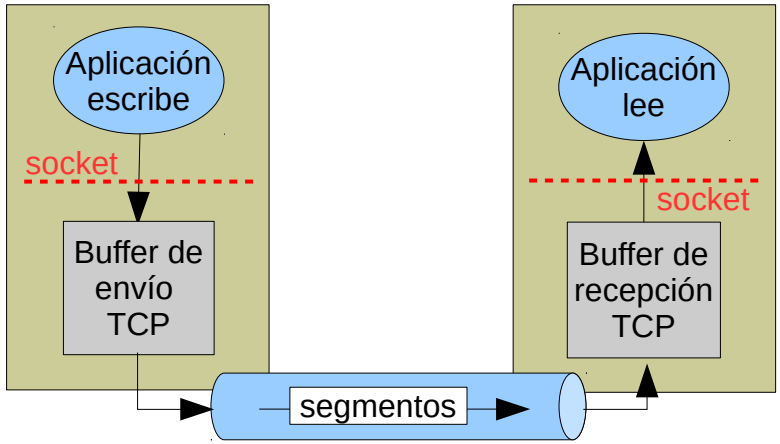
Capítulo 3: Continuación

- 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

- Punto a punto, un Tx y un Rx
- Confiable
  - Flujo de bytes ordenado
  - No hay límites de inicio o fin de mensaje
- Usa pipeline
  - Tamaño de la ventana TCP es definido por el control de congestión y control de flujo



- Usa buffering en Tx & Rx
- Transferencia full duplex (en ambos sentidos)
  - Flujo bidireccional de datos en la misma conexión
  - MSS: maximum segment size
- Orientado a la conexión
  - Handshaking (intercambio de mensajes de control) inicializa al Tx y Rx antes del intercambio de datos
- Usa control de flujo
  - Tx no sobrecargará al Rx

Estructura de un segmento TCP

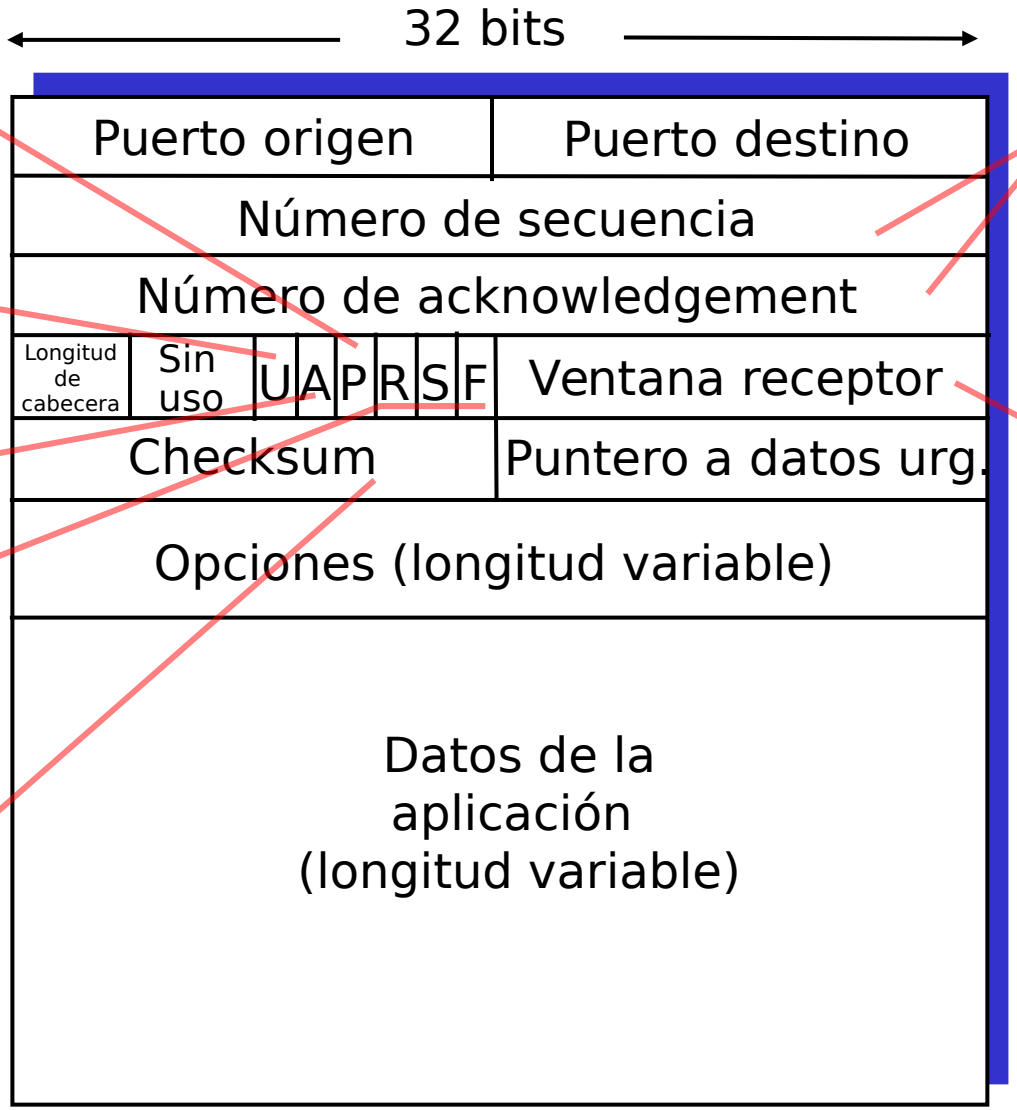
Push data now (Entregar datos inmediatamente)

Urgent data (raramente usado)

Número de ACK válido

RST, SYN, FIN Establecimiento de conexión (comandos de inicio y cierre)

Checksum (como en UDP)

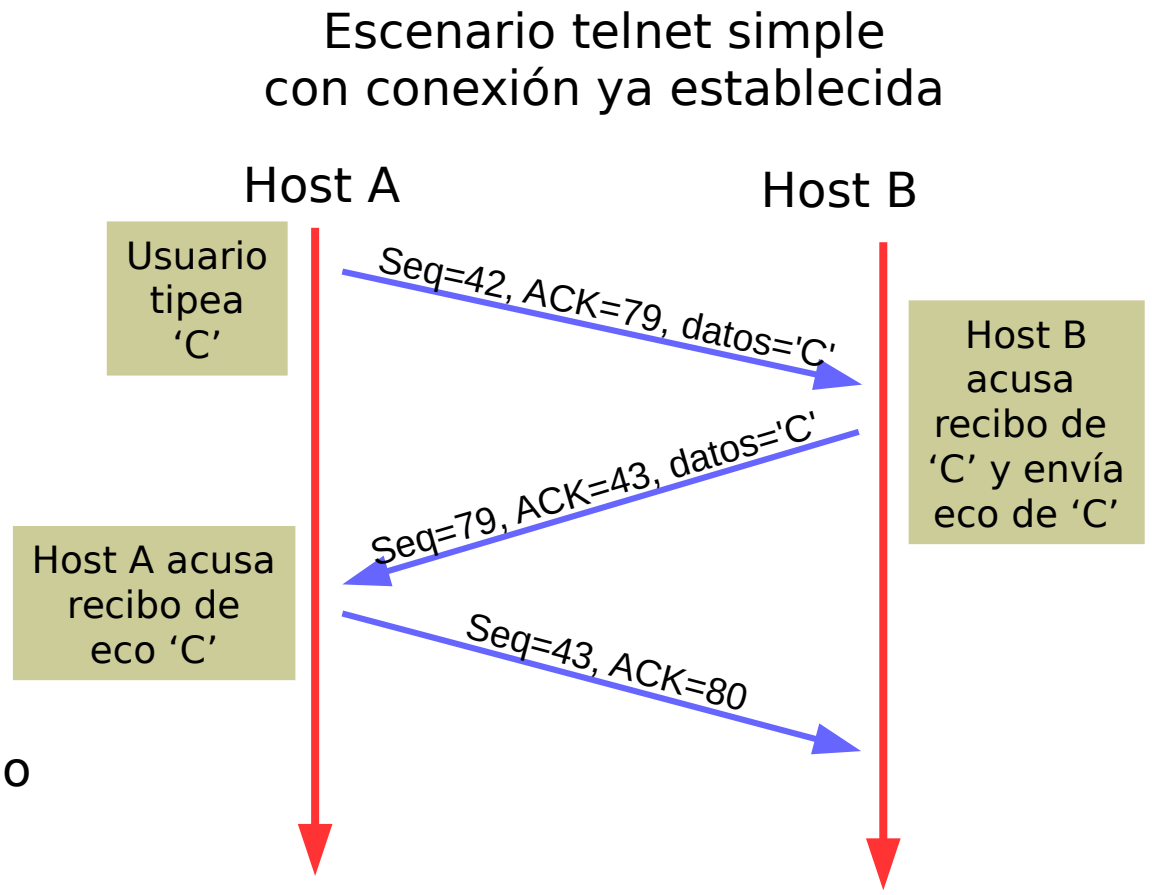


Cuentan bytes de datos (no segmentos)

Cantidad de bytes que el receptor está dispuesto a aceptar

Números de secuencia y ACKs TCP

- Número de Secuencia
  - Número* del primer byte de datos del segmento dentro del flujo
- ACKs
  - Número de secuencia del próximo byte esperado desde el receptor
- El ACK es acumulativo
- ¿Cómo maneja el receptor los segmentos fuera de orden?
  - La especificación de TCP lo deja a criterio del implementador



Round Trip Time y Timeout en TCP

- ¿Cómo fijar un 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

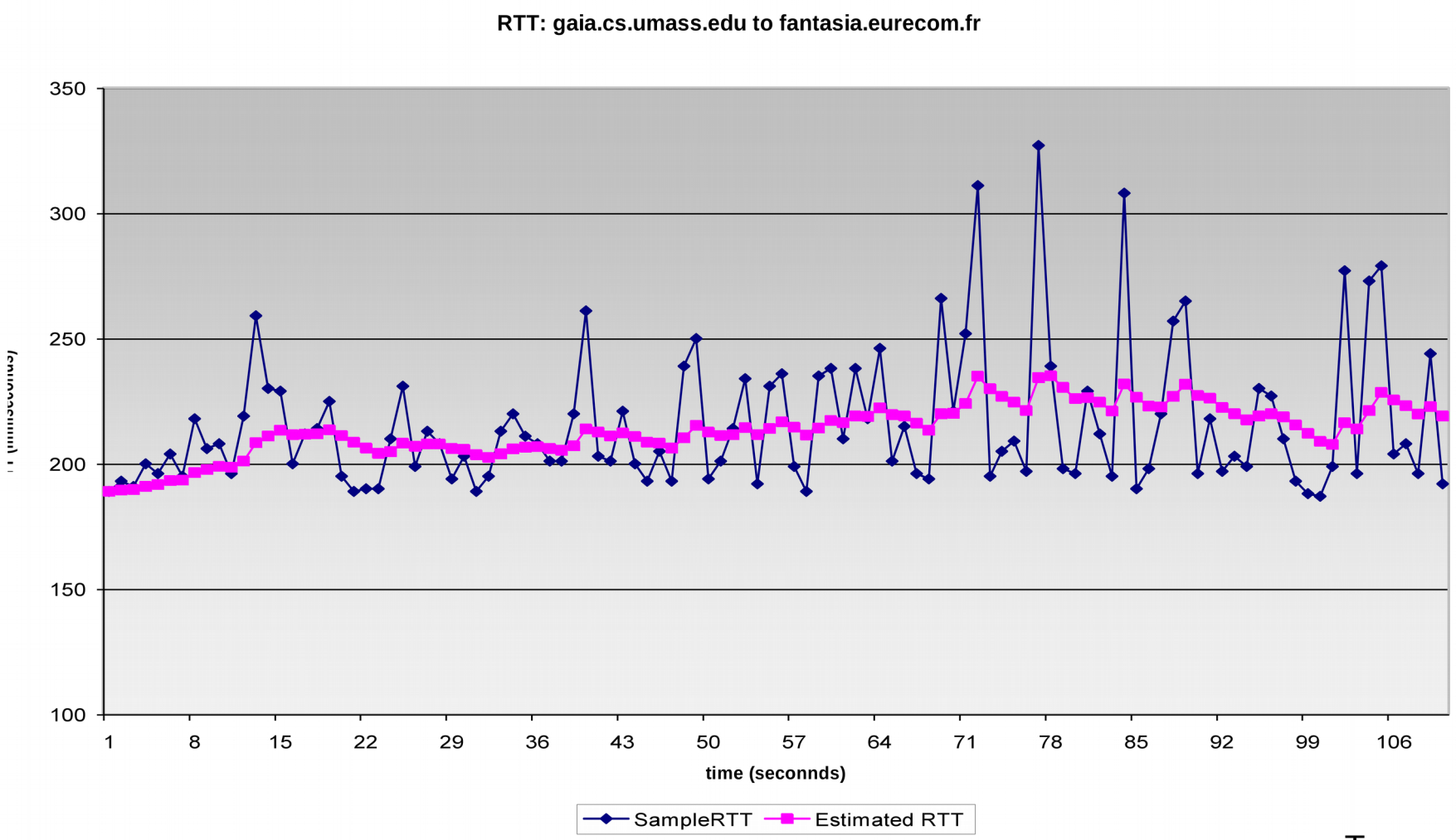
- ¿Cómo estimar el RTT?
- `SampleRTT`: mide tiempo desde transmisión del segmento hasta recibo de ACK
  - Ignora retransmisiones
- `SampleRTT` varía, hay que suavizar el RTT estimado
  - Promediar varias medidas recientes, no considerar sólo el último `SampleRTT`

Round Trip Time y Timeout en TCP

```
EstimatedRTT = (1 - α) * EstimatedRTT +
                    α  * SampleRTT
```

- Promedio móvil ponderado exponencial
- La influencia de las muestras pasadas decrece exponencialmente rápido
- Valor típico: $\alpha = 0.125 = (1/8) = 3$ right shifts

Ejemplo de estimación de RTT



Round Trip Time y Timeout en TCP

Definición del timeout

- **EstimatedRTT** más un “margen de seguridad”
  - Si hay gran variación en **EstimatedRTT** → usar gran margen
- Primero estimamos cuánto se desvía el **SampleRTT** del **EstimatedRTT**:

$$\mathbf{DevRTT = (1-\beta)* DevRTT + \beta * |SampleRTT-EstimatedRTT|}$$

(típicamente, $\beta = 0.25$)

- Luego fijamos el timeout como:

$$\mathbf{TimeoutInterval = EstimatedRTT + 4*DevRTT}$$

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 en TCP

- TCP crea un servicio de transferencia confiable sobre el servicio no confiable de IP
- Envía segmentos en pipeline
- Acks acumulativos
- TCP usa un timer de retransmisión único
- Las retransmisiones son activadas por:
  - Eventos de *timeout*
  - Acks duplicados
- Inicialmente consideremos un transmisor TCP simplificado:
  - Ignora acks duplicados
  - Ignora control de flujo y control de congestión

Eventos del transmisor en TCP

Datos recibidos desde aplicación

- Crea segmento con número de secuencia
- Número de secuencia es el número dentro del flujo para el primer byte del segmento
- Inicia timer si no está ya corriendo
  - Asociar el timer al segmento más viejo sin acuse de recibo
- Tiempo de expiración: `TimeOutInterval`

Timeout

- Retransmitir el segmento que causó el timeout
- Re-iniciar el timer

Recepción de Ack

- Si es el ack de un segmento previo sin acuse de recibo
  - Actualizar estado sobre acuses recibidos
  - Iniciar timer si hay segmentos sin acuses de recibo.

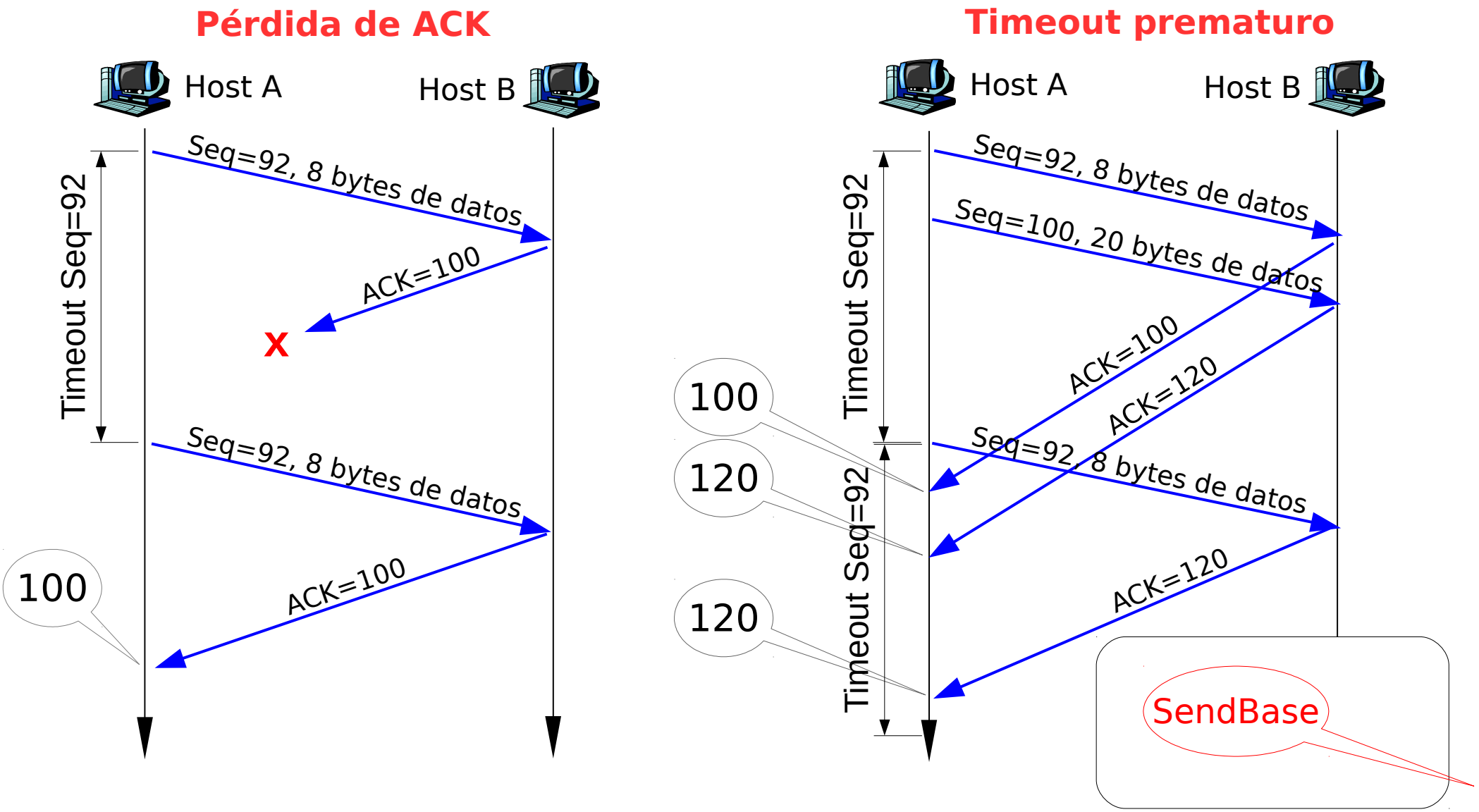
Transmisor TCP
simplificado

```
NextSeqNum = InitialSeqNum
SendBase = InitialSeqNum

loop (forever) {
   switch(event) {

   event: Datos recibidos desde la aplicación
        Crear segmento TCP con número
         de secuencia NextSeqNum
        if (timer actualmente no está corriendo)
                Iniciar timer
        Pasar segmento a IP
        NextSeqNum = NextSeqNum + length(data)
        break;

   event: Timeout del timer
        Retransmitir segmento con menor número
         de secuencia sin acuse
        Iniciar timer
        break;

   event: Recepción de ACK con campo ACK de valor y
        if (y > SendBase) {
                SendBase = y
                if (aún hay segmentos sin acuse)
                        iniciar timer
        }
   }

}  /* end of loop forever */
```

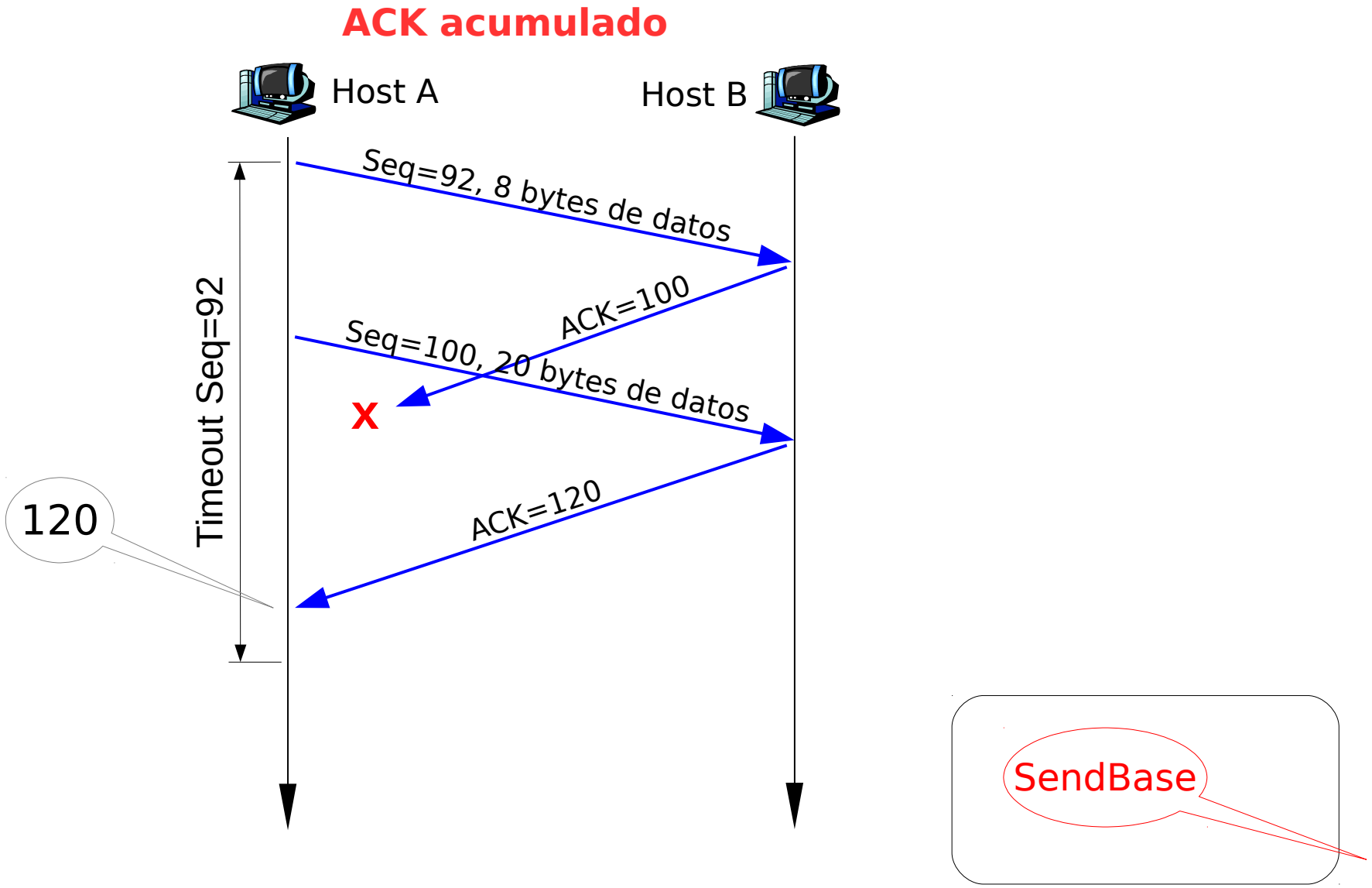
- SendBase-1 = Último byte reconocido con ACK

- Ejemplo
  - SendBase-1=71, y=73
  - El receptor quiere los bytes del 73 en adelante
  - Como y > SendBase, el acuse de recibo que ha llegado corresponde a un nuevo dato

TCP: escenarios de retransmisión



TCP: escenarios de retransmisión



Generación de ACKs en TCP

RFC 1122, RFC 2581

Evento en receptor		Acción del receptor
Llega segmento en orden con número de secuencia esperado	Ya se envió ACK para todo lo anterior	ACK retardado. Espera hasta 500ms por el próximo segmento. Si no llega, enviar otro ACK.
	Hay segmentos con ACK pendiente	Enviar inmediatamente un ACK acumulado, reconociendo ambos segmentos en orden.
Llega segmento fuera de orden	Segmento recibido tiene número de secuencia mayor al esperado → se detecta un vacío	Enviar inmediatamente un ACK duplicado indicando número de secuencia del próximo byte esperado.
	Segmento recibido llena un vacío parcial o completamente	Enviar inmediatamente un ACK si el segmento se ubica al inicio del vacío.

Retransmisiones Rápidas

- Período de timeout a veces es demasiado largo
  - El retardo antes del re-envío de paquetes perdidos es largo
- Los paquetes perdidos se detectan a través de los ACKs duplicados
  - El transmisor a menudo envía muchos segmentos seguidos
  - Si un segmento es perdido, probablemente habrá muchos ACKs duplicados
- Si el transmisor recibe **tres ACKs repetidos de un mismo dato ya reconocido**, supone que el segmento después de este dato se perdió
  - Retransmisión rápida: reenviar el segmento antes que el timer expire

Algoritmo de retransmisión rápida

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

ACK duplicado de un segmento con ACK recibido

Retransmisión rápida

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