

# Tema 3: El nivel de transporte

Bibliografía: [Kurose17], Capítulo 3 (excepto 3.4 y 3.6)

- Justificar la existencia del nivel de transporte
- Comprender el funcionamiento del servicio de transporte sin conexión (UDP)
- Aprender el funcionamiento básico del servicio de transporte orientado a la conexión (TCP)
  - En particular los mecanismos de TCP para:
    - Control de flujo y de errores
    - Control de la congestión

- 
- 1. Servicios del nivel de transporte**
  - 2. Transporte sin conexión: UDP**
  - 3. Fundamentos de la transferencia fiable de datos**
  - 4. Transporte orientado a la conexión: TCP**

## 1. Servicios del nivel de transporte

2. Transporte sin conexión: UDP

3. Fundamentos de la transferencia fiable de datos

4. Transporte orientado a la conexión: TCP

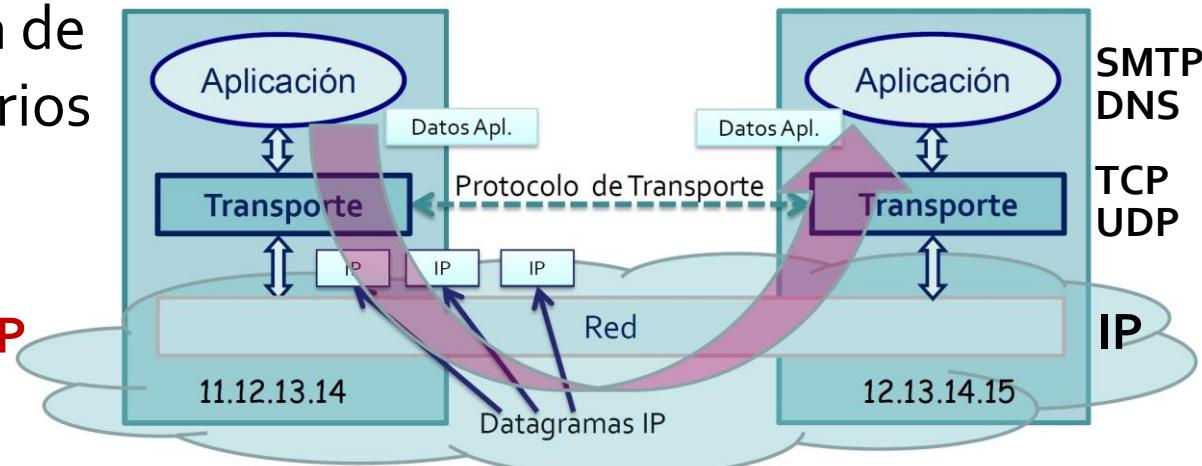
## 1. Servicios del nivel de transporte

### Conceptos:

- El nivel de transporte
  - Concepto de protocolo de extremo a extremo
- Direccionamiento de transporte
  - Direccionamiento de red y necesidad de identificar los procesos
  - Demultiplexación en el nivel de transporte

## Servicios del nivel de transporte

- Proporciona **comunicación lógica** entre procesos de aplicación
- Los protocolos de transporte se implementan en los hosts
  - Extremo emisor: **fragmenta** los mensajes de aplicación en **segmentos** y los pasa al nivel de red
  - Extremo receptor: **ensambla** los **segmentos** en mensajes de aplicación y los pasa al nivel superior
- En una arquitectura de red puede haber varios protocolos de transporte:
  - Internet: **TCP** y **UDP**



## Nivel de transporte vs Nivel de red

- **Nivel de red**
  - Protocolo IP (Internet Protocol)
  - Comunicación lógica entre hosts (o incluso dentro del mismo host)
  - Servicio de entrega de “mejor esfuerzo” (best effort)
- **Nivel de transporte**
  - Comunicación lógica entre procesos (en diferentes hosts o en el mismo host)
    - Permite la implementación de múltiples servicios de aplicación en el mismo host
  - Servicio de entrega (fiable o no fiable) dependiendo del protocolo usado (TCP o UDP)

## Protocolos del nivel de transporte

### ■ TCP (Transmission Control Protocol)

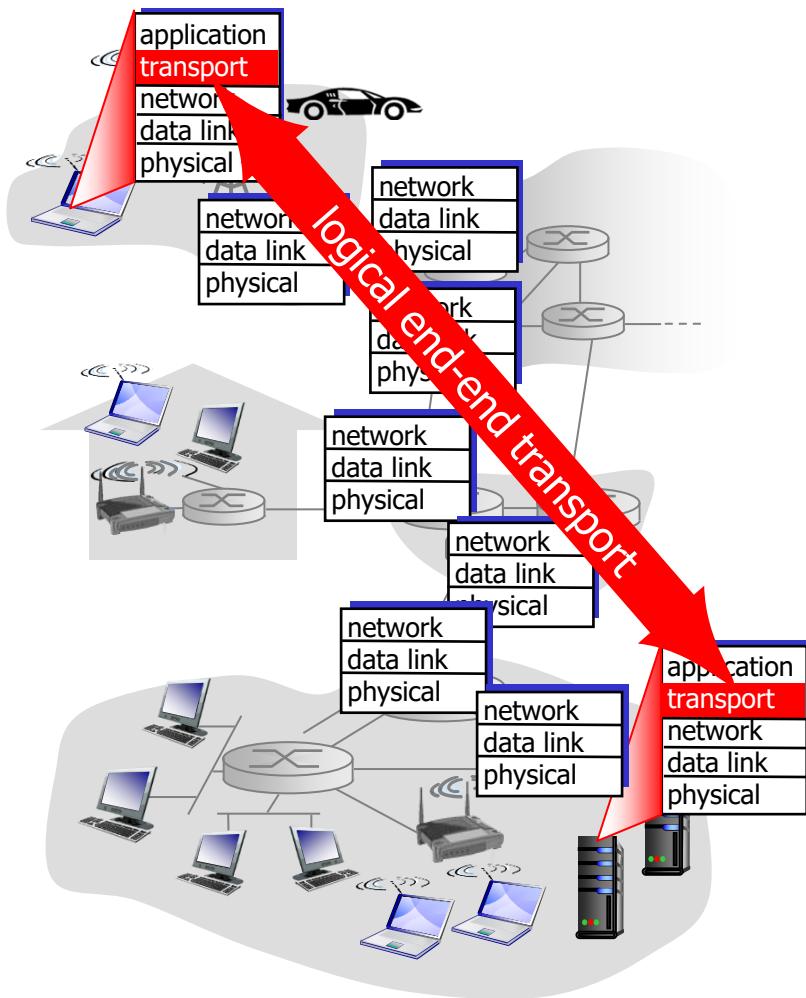
- Fiable:
  - Sin pérdidas
  - Entrega en orden
- Establecimiento de conexión
- Control de flujo
- Control de congestión

### ■ UDP (User Datagram Protocol)

- No fiable:
  - Posibilidad de pérdidas
  - Posibilidad de entrega fuera de orden
- Sencillo, extensión del servicio IP “mejor esfuerzo”

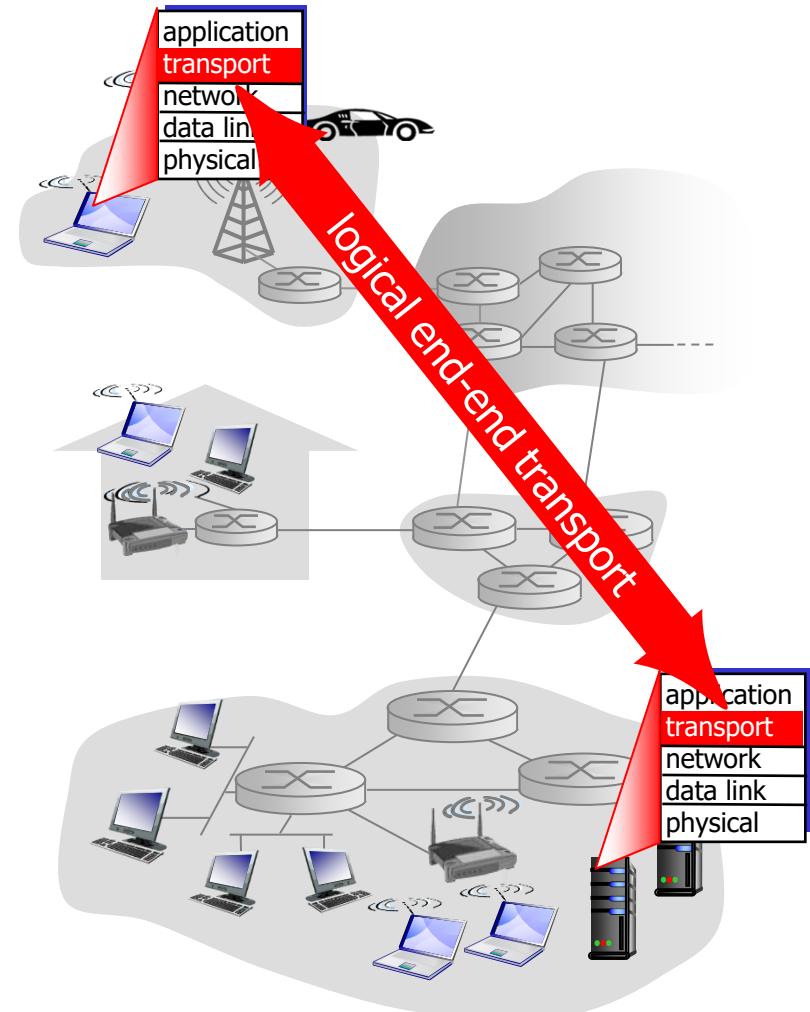
### ■ Servicios no disponibles:

- Garantía de retardos
- Garantía de anchos de banda



## El nivel de transporte abstracte

- El nivel de transporte oculta los detalles de los niveles que tiene por debajo de él



## Direccionamiento de transporte

### ■ Direccionamiento de red

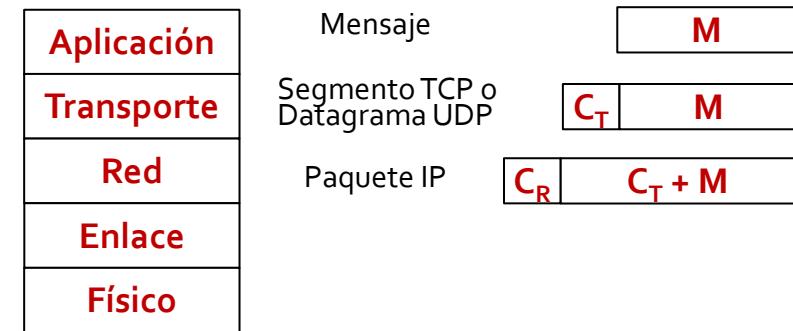
- Comunicación lógica entre hosts
- Identificación de hosts: **dirección IP**

### ■ Direccionamiento de transporte

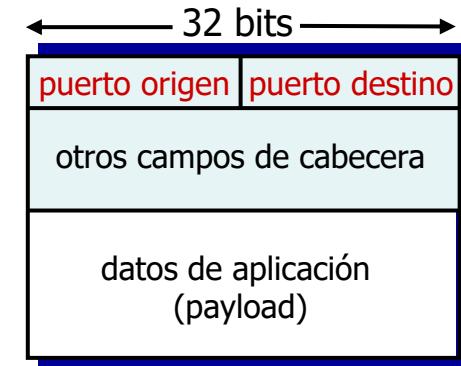
- Comunicación lógica entre procesos
- Amplía el servicio de entrega de IP entre dos hosts a un servicio de entrega entre dos procesos
  - **Multiplexación y demultiplexación** del nivel de transporte
- Se apoya en los servicios del nivel de red y puede mejorarlo
- Identificación de procesos
  - De forma simplista se puede pensar que el identificador es el puerto
  - Pero ... el identificador debe incluir la **IP** y el **puerto**

## Demultiplexación en el nivel de transporte

- El host recibe un paquete IP
  - Cada paquete IP tiene una **dirección IP fuente** y una **dirección IP destino**
  - Cada paquete IP contiene un segmento (o datagrama) del nivel de transporte
  - Cada segmento (o datagrama) contiene un **puerto origen** y un **puerto destino**
- El nivel de transporte usa las **direcciones IP** y los **números de puerto** para dirigir el segmento (o datagrama) al socket apropiado

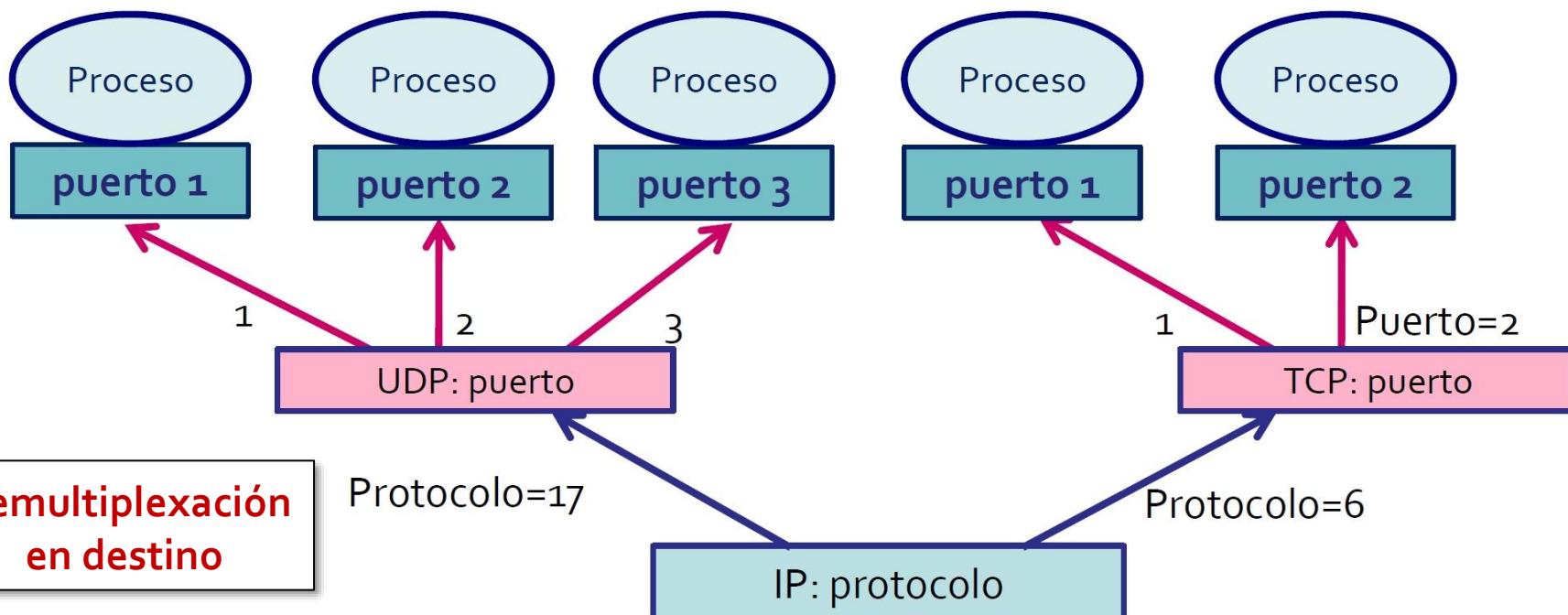


**Formato segmento TCP  
o datagrama UDP**



## Demultiplexación en el nivel de transporte

- Receptor: la información de las cabeceras permite entregar los datos al proceso correcto
  - **Socket**: Punto de acceso a los servicios de transporte
  - Los atributos que diferencian los distintos sockets en Internet son la **dirección IP** y el **número de puerto**



## Demultiplexación en TCP

En el establecimiento de conexión se crea un socket (**s**)  
 (IP local, puerto local, IP remota, puerto remoto)

```
import java.net.*;
import java.io.*;
import java.util.Scanner;

public class ClienteTCP {
    public static void main(String args[])
        throws UnknownHostException, IOException {
        Socket s = new Socket("zoltar.redes.upv.es", 7777);
        PrintWriter salida = new PrintWriter(s.getOutputStream());
        salida.print("Hola, esto es un mensaje \r\n");
        salida.flush();
        Scanner entrada =new Scanner(s.getInputStream());
        System.out.println(entrada.nextLine());
        s.close();
        System.out.println("Desconectado");
    }
}
```

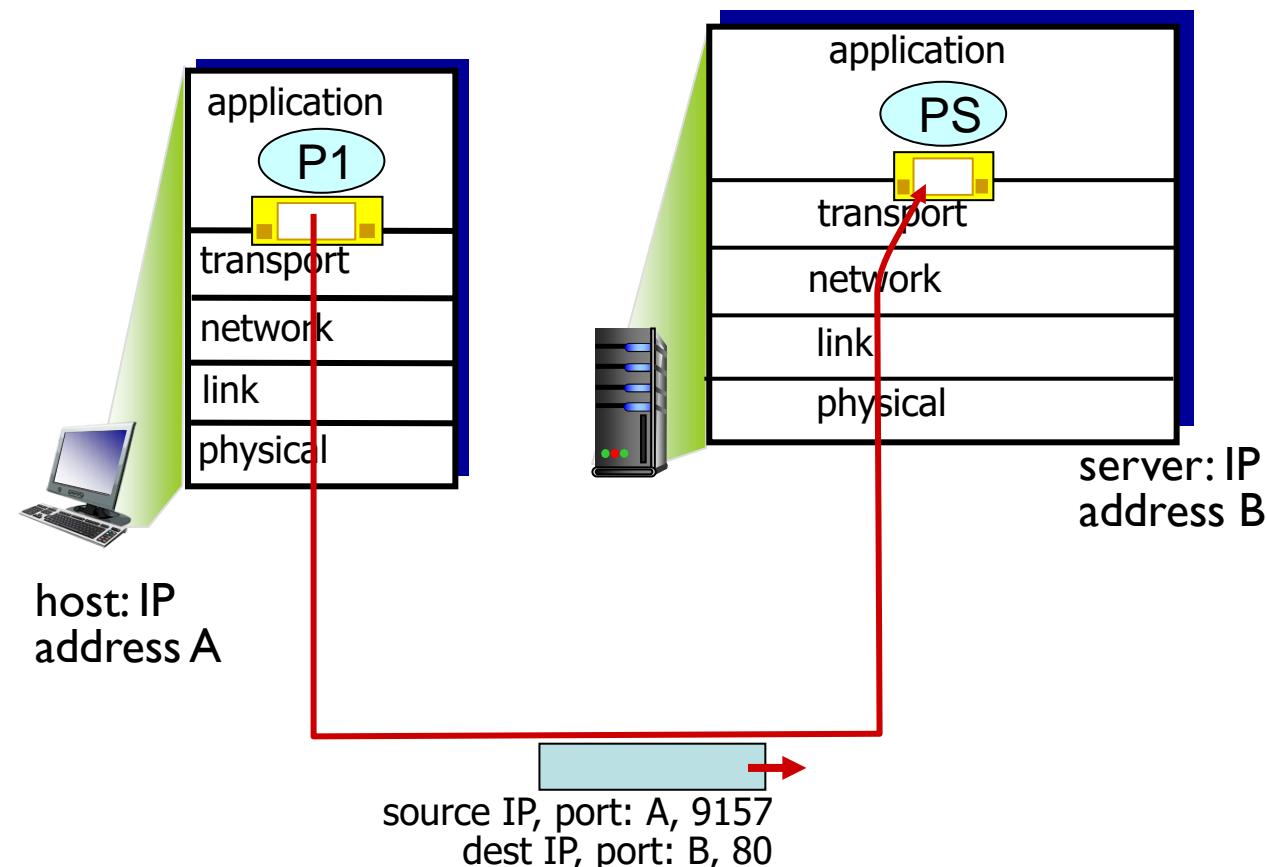
```
import java.net.*;
import java.io.*;

public class ServidorTCP {
    public static void main(String args[])
        throws UnknownHostException, IOException {
        ServerSocket ss = new ServerSocket(7777);
        while(true){
            Socket s1 = ss.accept(),
            Scanner entrada = new Scanner(s1.getInputStream());
            PrintWriter salida = new PrintWriter(s1.getOutputStream());
            salida.println(entrada.nextLine());
            salida.flush();
            s1.close();
        }
    }
}
```

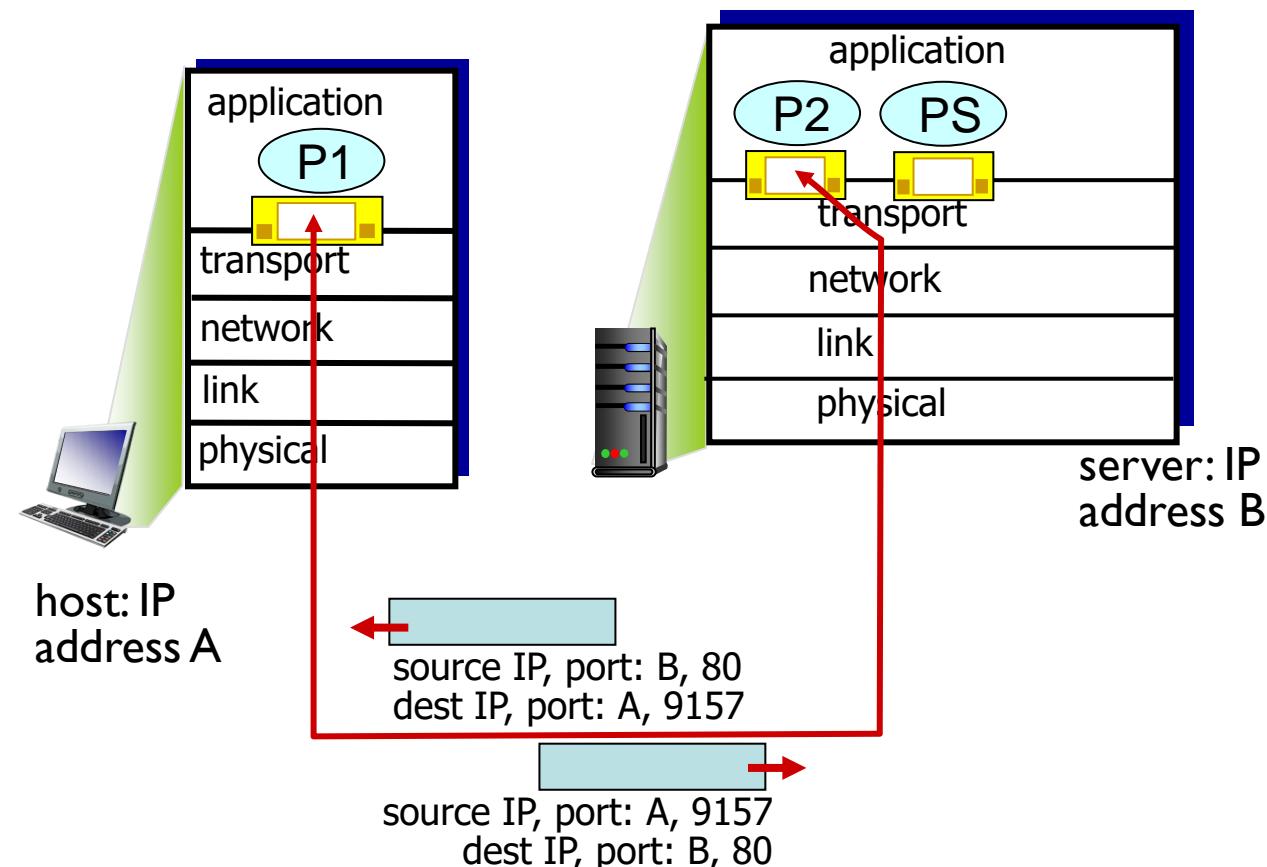
El socket servidor TCP tiene un puerto local

En el establecimiento de conexión se crea un socket (**s1**)  
 (IP local, pto local, IP remota, pto remoto)

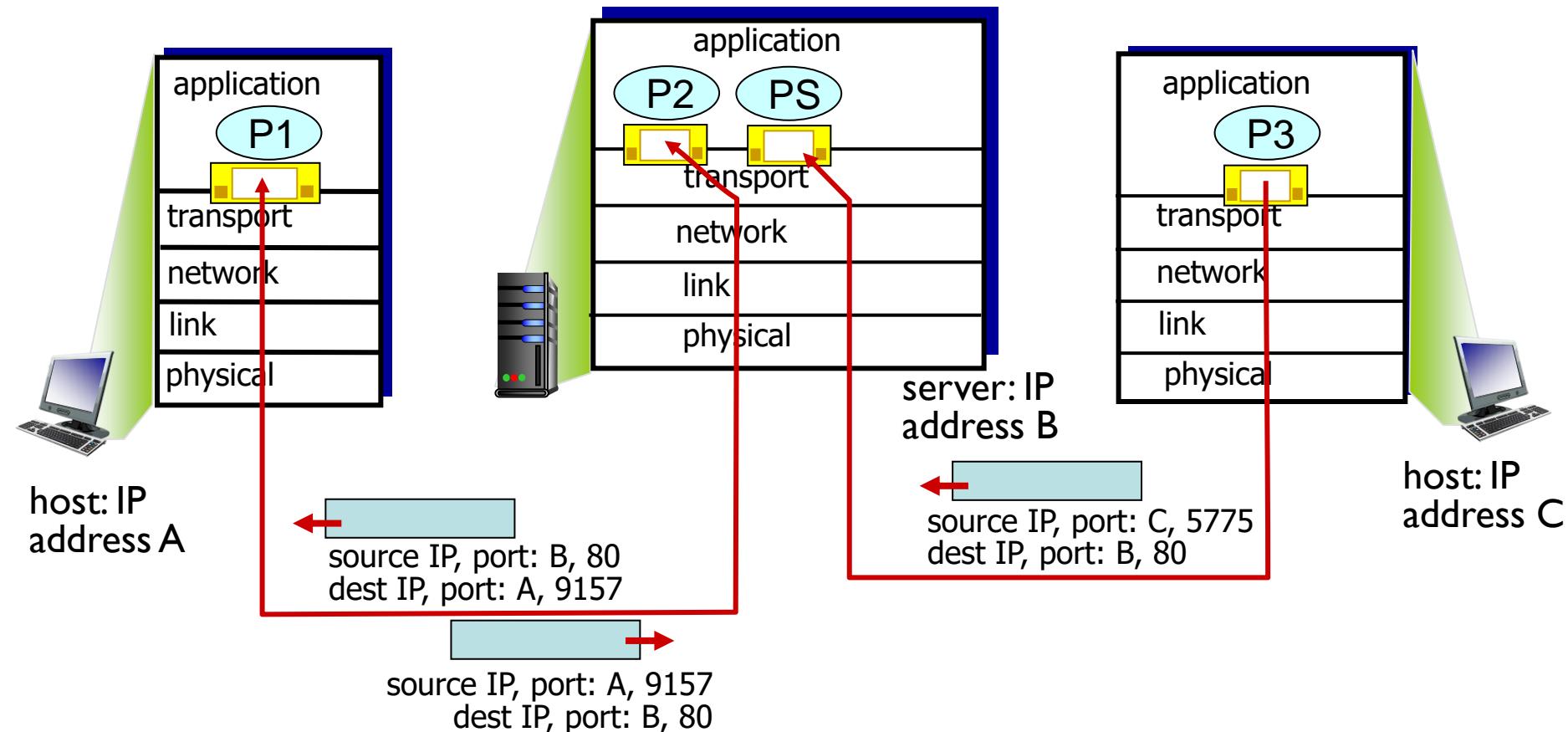
## Demultiplexación en TCP



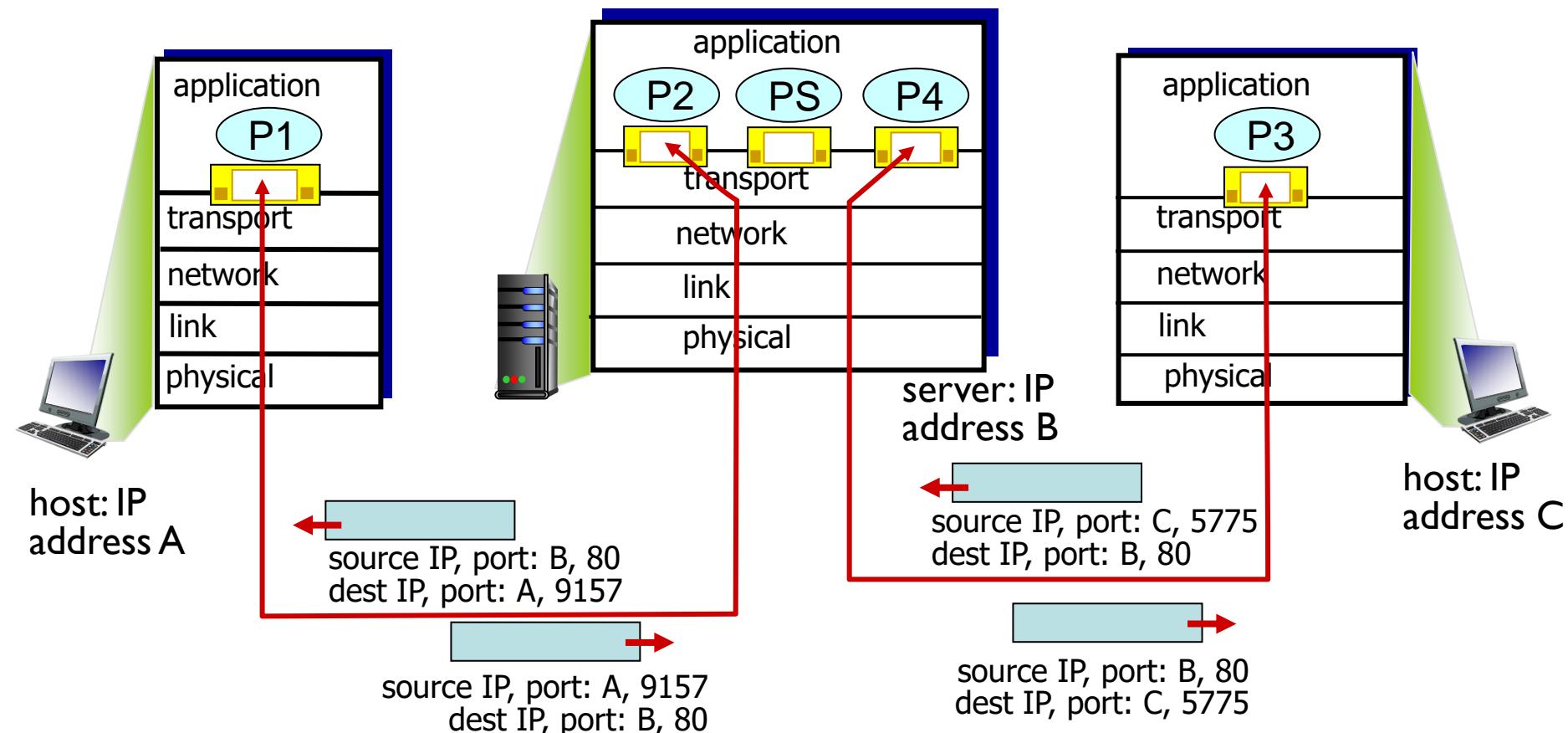
## Demultiplexación en TCP



## Demultiplexación en TCP

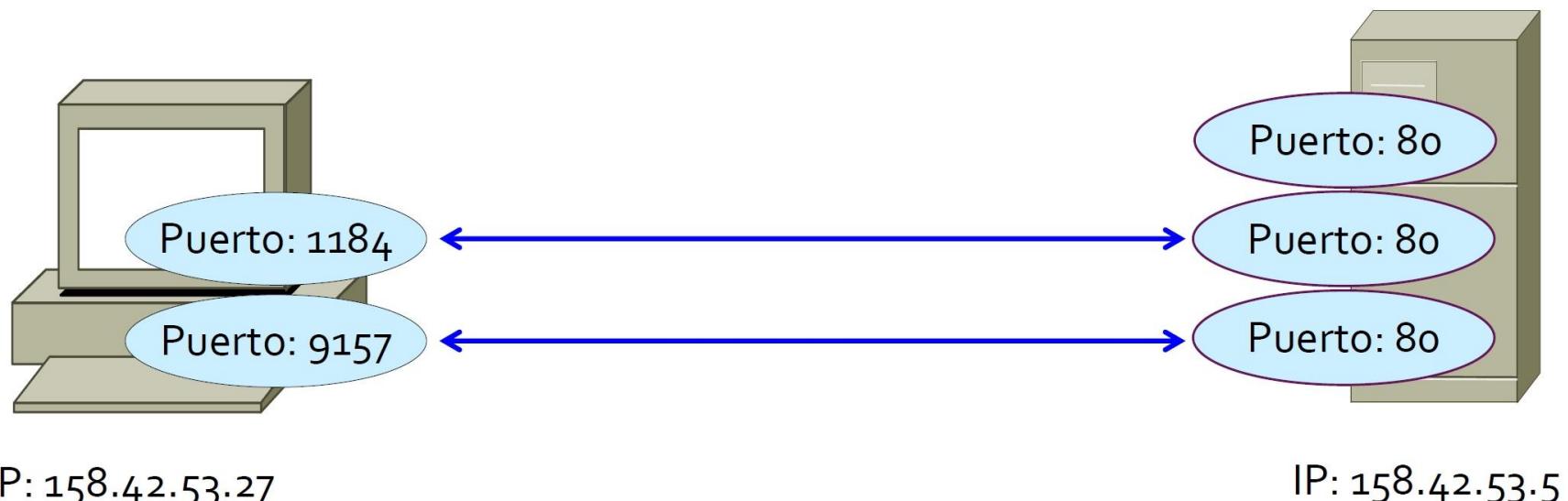


## Demultiplexación en TCP



## Demultiplexación en TCP

- Un mismo número de puerto TCP puede emplearse simultáneamente en distintas conexiones
  - Ej: (158.42.53.27, **1184**, **158.42.53.5, 80**) y (158.42.53.27, **9157**, **158.42.53.5, 80**)



## Demultiplexación en TCP: resumen

- Un **socket TCP sin conectar** se identifica por dos valores:
  - dirección IP local y puerto local
- Un **socket TCP conectado** se identifica por una 4-tupla:
  - dir IP fuente, puerto fuente, dir IP destino, puerto destino
- El receptor usa los cuatro valores para dirigir el segmento al socket apropiado
- El host servidor puede soportar varios sockets TCP simultáneos en el mismo número de puerto:
  - Cada socket identificado por su 4-tupla

## Demultiplexación en UDP

El sistema operativo asigna automáticamente un número de puerto al socket

El datagrama a enviar por el socket UDP especifica IP y puerto destino

```
import java.net.*;
import java.io.*;

public class ClienteUDP{
    public static void main(String[] args) throws IOException {
        DatagramSocket s = new DatagramSocket();
        InetAddress dir = InetAddress.getByName("zoltar.redes.upv.es");
        String msg = "Hola, esto es un mensaje \r\n";
        byte[] buf = new byte[256];
        buf = msg.getBytes();
        DatagramPacket p = new DatagramPacket(buf, buf.length, dir, 7777);
        s.send(p);
        s.receive(p); // se bloquea hasta que recibe un datagrama
        String ans = new String(p.getData(), 0, p.getLength());
        System.out.println(ans);
        s.close();
    }
}
```

```
import java.net.*;
import java.io.*;

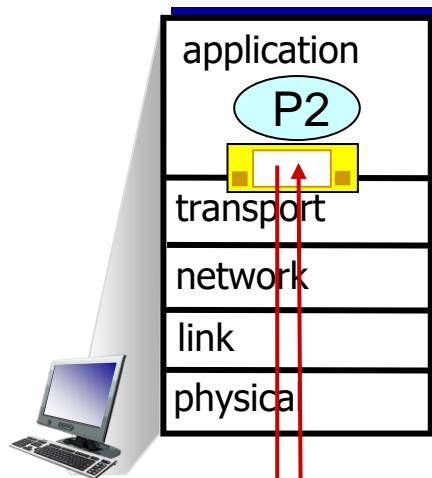
public class ServidorUDP{
    public static void main(String[] args) throws IOException {
        DatagramSocket s = new DatagramSocket(7777);
        DatagramPacket p = new DatagramPacket(new byte[512], 512);
        while(true){
            s.receive(p); // se bloquea hasta que recibe un datagrama
            s.send(p);
        }
    }
}
```

El socket UDP tiene un puerto local

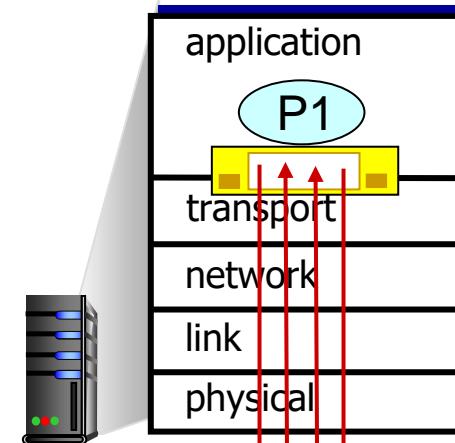
El datagrama recibido (p) contiene la IP y el puerto del emisor

## Demultiplexación en UDP

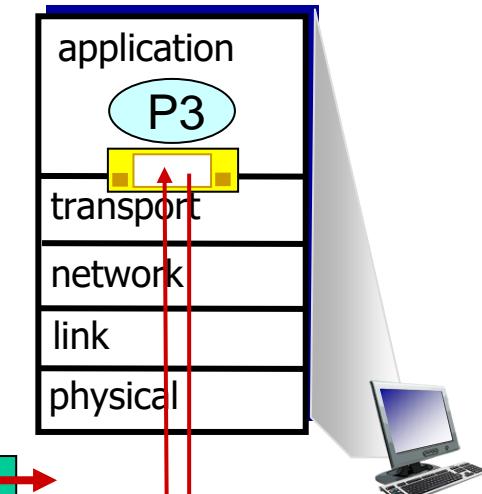
```
DatagramSocket  
mySocket2 = new  
DatagramSocket(9157);
```



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



```
DatagramSocket  
mySocket3 = new  
DatagramSocket(5775);
```



source port: 9157  
dest port: 6428

source port: 6428  
dest port: 5775

source port: 5775  
dest port: 6428

## Demultiplexación en UDP

- Un socket UDP se identifica por dos valores:  
**Dirección IP destino, número de puerto destino**
- Dos datagramas UDP con diferentes direcciones IP y/o número de puerto de origen, **pero la misma dirección IP destino y el mismo número de puerto destino**, se enviarán al mismo proceso de destino a través del mismo socket de destino

1. Servicios del nivel de transporte
- 2. Transporte sin conexión: UDP**
3. Fundamentos de la transferencia fiable de datos
4. Transporte orientado a la conexión: TCP

# 2. Transporte sin conexión: UDP

### Conceptos:

- Funcionamiento
- Necesidad
- Estructura de los segmentos UDP
- Suma de comprobación

## UDP (User Datagram Protocol, RFC 768)

- Servicio de **transferencia NO fiable**
  - Sin conexión: sólo transferencia de datos
    - Cada datagrama se trata de forma independiente
  - Sin garantía de entrega y sin garantía de entrega en orden
  - Sobrecarga mínima
- Bloques de hasta 64 KBytes
- Permite difusiones
- Unidad de transferencia de información: **datagrama**
- Aplicaciones UDP
  - Flujos multimedia (tolerantes a pérdidas, sensibles a la tasa de transferencia)
  - DNS, RIP, etc
- Si se requiere transferencia fiable sobre UDP, debe implementarla la aplicación

## ¿Por qué existe UDP?

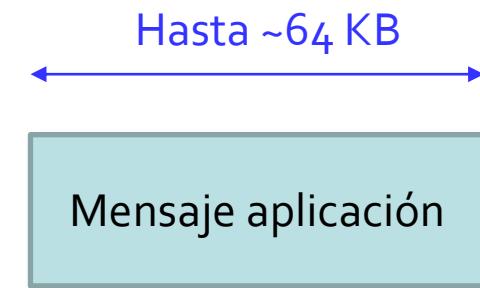
- No hay establecimiento de conexión (que puede añadir un retardo)
- Sencillo: no hay estado de la conexión ni en el emisor ni en el receptor
- Cabecera del datagrama pequeña (8 bytes en UDP frente a 20 bytes en TCP)
- No hay control de congestión: UDP sale del emisor tan rápido como se desee

## Encapsulado de datagramas UDP

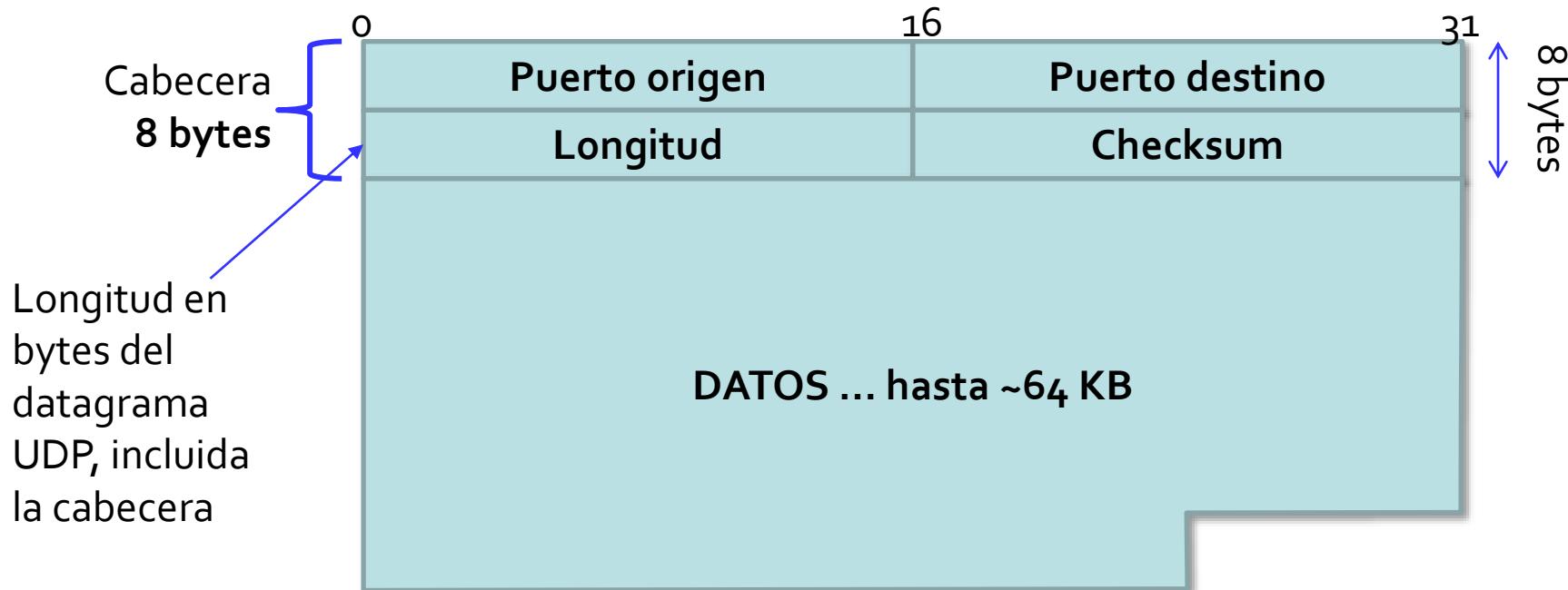
**APLICACIÓN:** Mensaje

**UDP:** Datagrama UDP

**IP:** Datagrama IP o paquete IP



## Formato de un datagrama UDP



## Cálculo del checksum UDP

- **Objetivo:** detectar errores en los datagramas recibidos
- **Emisor:** calcula el checksum:
  - Considera el datagrama (cabecera+datos) como una secuencia de enteros de 16 bits
  - Realiza la suma de todas las palabras de 16 bits empleando aritmética en complemento a uno:
    - Los acarreos se propagan, sumándose al bit menos significativo
  - Calcula el complemento a uno del resultado y lo emplea como checksum
    - Si el checksum computado es cero se transmitirá como todo unos
    - Un checksum todo a ceros indica que el emisor no calculó el checksum
- **Receptor:** Realiza nuevamente la suma (incluido el checksum)
  - El resultado debe ser 0 en complemento a 1, representado por  
**1111111111111111**
  - En caso contrario, el mensaje se considera erróneo

## Ejemplo de cálculo de checksum

- Ejemplo con dos sumandos de 16 bits

	1	1	0	1	1	0	0	1	1	0	0	1	0	0	1	0
	+	1	1	1	0	1	0	0	0	1	0	0	0	1	0	0
<hr/>																
Acarreo	1	1	1	0	0	0	0	1	0	0	0	0	1	1	0	1
																
<hr/>																
Suma	1	1	0	0	0	0	1	0	0	0	0	1	1	1	0	0
Checksum	0	0	1	1	1	1	0	1	1	1	1	0	0	0	1	1

1. Servicios del nivel de transporte
2. Transporte sin conexión: UDP
- 3. Fundamentos de la transferencia fiable de datos**
4. Transporte orientado a la conexión: TCP

## 3. Fundamentos de la transferencia fiable de datos

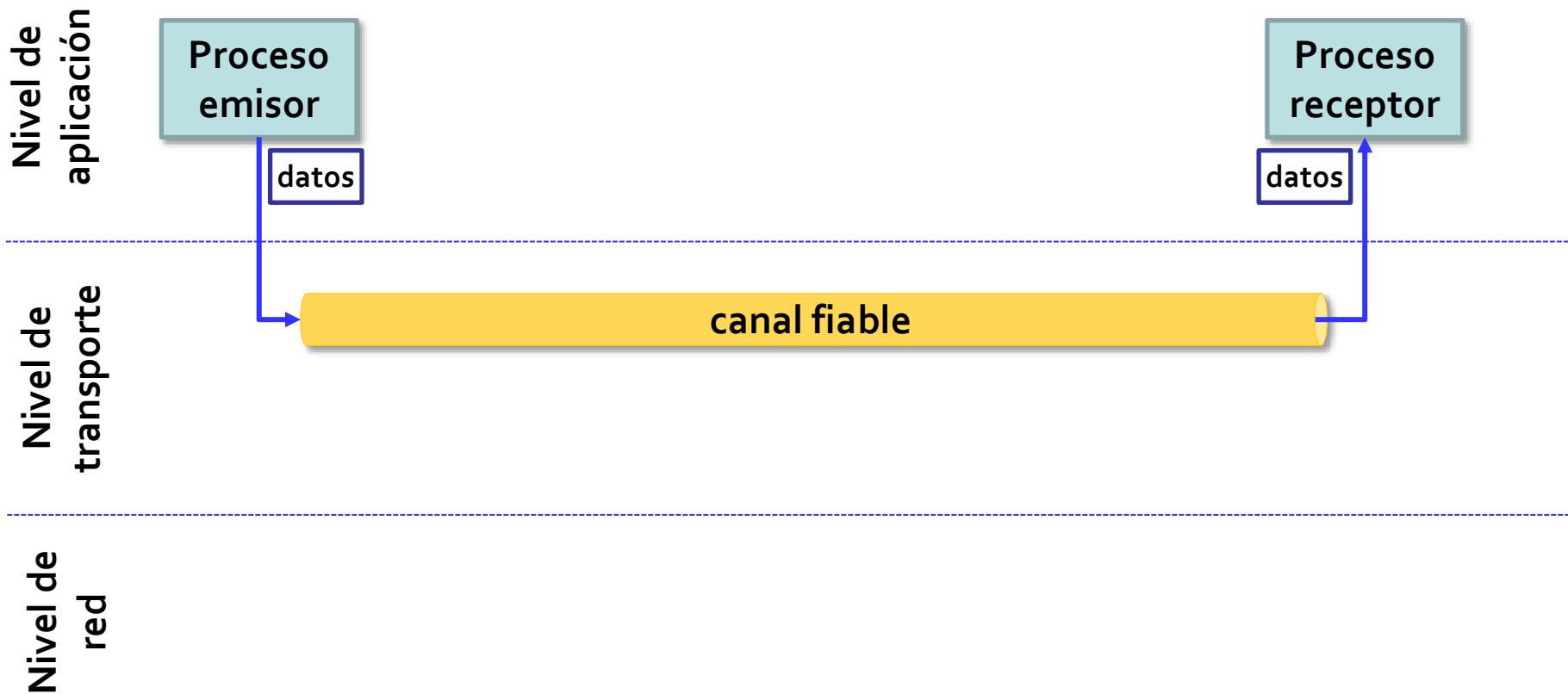
### Conceptos:

- Protocolo ARQ (Automatic Repeat reQuest)
  - Elementos necesarios
    - Detección de errores en el receptor
    - Reconocimientos (ACK)
    - Retransmisión
  - Detección de errores por el emisor
    - Vencimiento del temporizador de retransmisión
    - Generación de duplicados
    - Concepto de reconocimiento acumulativo
  - Protocolo de “parada y espera”
  - Protocolo de “procesamiento en cadena” (pipeling o sliding window)
  - Comparación de rendimiento
  - Control de errores en el protocolo de “procesamiento en cadena”
    - Vuelta atrás
    - Retransmisión selectiva
  - Transmisión bidireccional: Piggybacking

## Fundamentos de la transferencia fiable

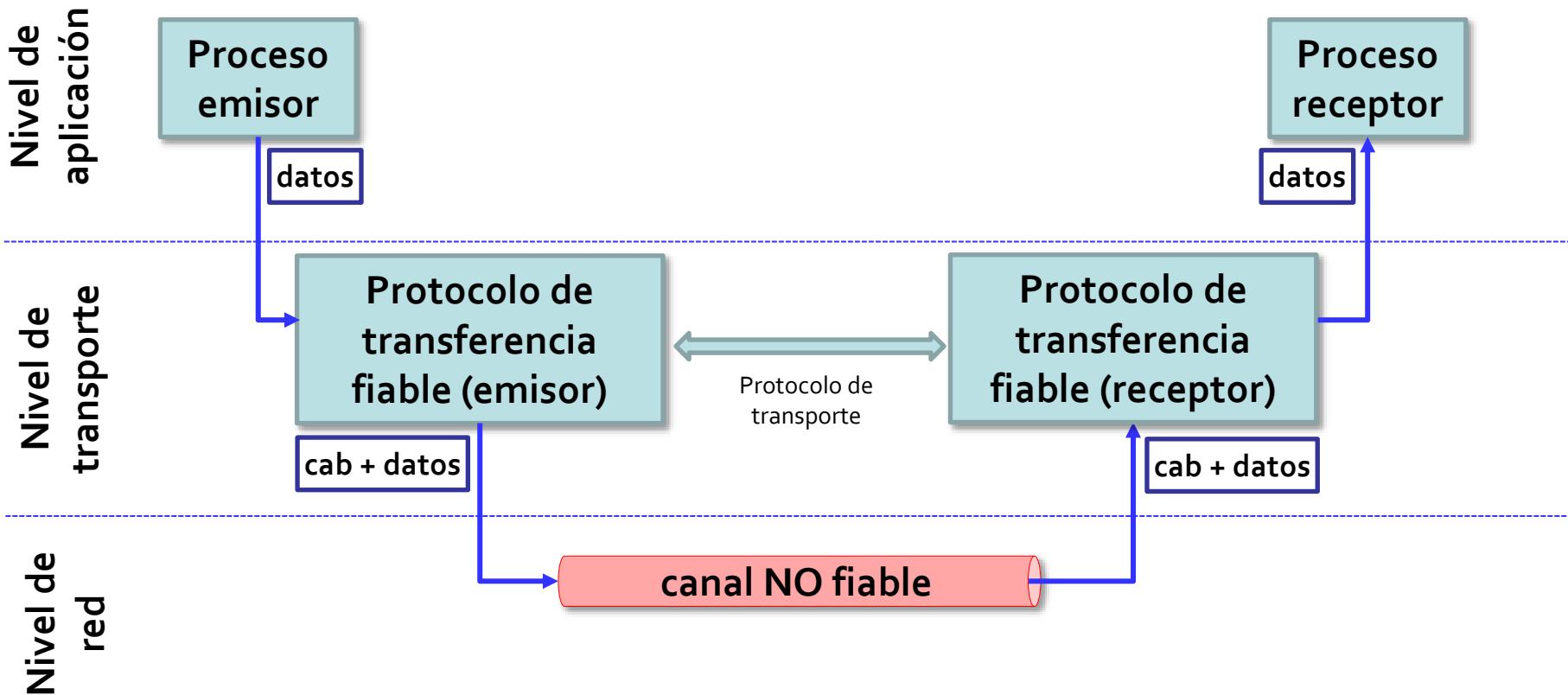
- La transferencia fiable es un problema que se puede plantear en distintos niveles de la arquitectura
- Problema:
  - ¿Cómo conseguir transferencia fiable sobre una red que puede perder segmentos?
- Solución:
  - Proporcionar mecanismos que garanticen la **detección** y **retransmisión** de los datos perdidos o dañados

## Fundamentos de la transferencia fiable



- **Canal fiable:** entrega de datos correctos y en orden

## Fundamentos de la transferencia fiable



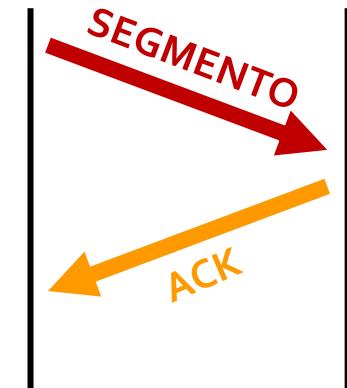
- **Canal fiable:** entrega de datos correctos y en orden
- **Canal no fiable:** errores de transmisión, congestión, encaminamiento, entrega fuera de orden, etc

## Protocolos para transferencia fiable

- Cuestiones básicas a resolver:
  - **Receptor**
    - ¿El dato (segmento) recibido es correcto?
      - Evaluación de un código redundante (checksum o similar)
    - ¿Qué hacer en caso de que no sea correcto?
  - **Emisor**
    - ¿Se ha recibido correctamente el segmento enviado?
- Protocolo ARQ (**A**utomatic **R**epeat **re****Q**uest)
  - Solicitud automática de repetición (retransmisión)

## ARQ (Automatic Repeat reQuest)

- Gestión de los errores de transmisión
  - **Detección del error:** uso de códigos detectores de errores
  - **Recuperación del error** en base a dos mecanismos que actúan de forma simultánea:
    - **Reconocimientos:** (Acknowledgements, ACKs) deben ser devueltos dentro de un plazo de tiempo
      - El problema puede presentarse también en el ACK (la transmisión de los ACKs tampoco es fiable)
    - **Retransmisión:** si el reconocimiento no llega en un tiempo acotado (*timeout, RTO retransmission timeout*) se reenvía el segmento

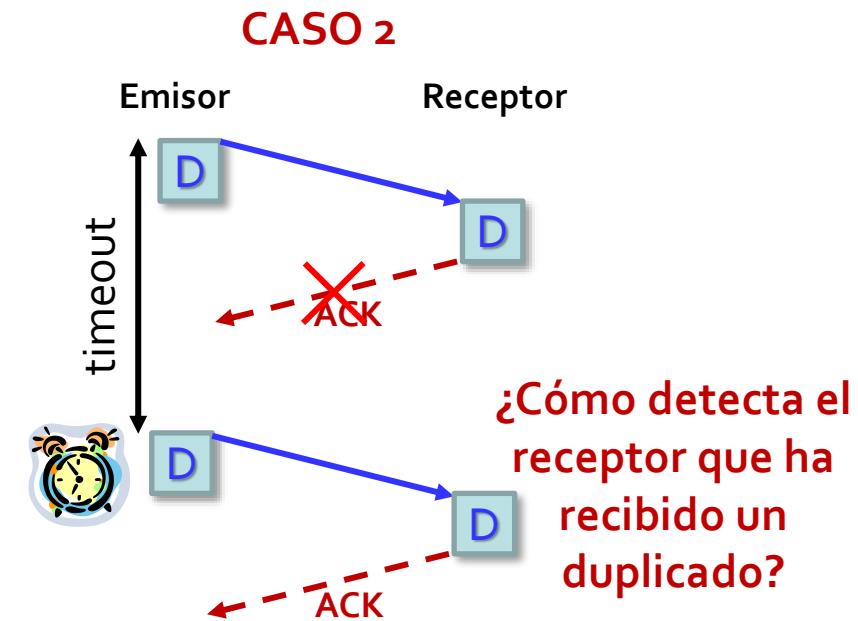
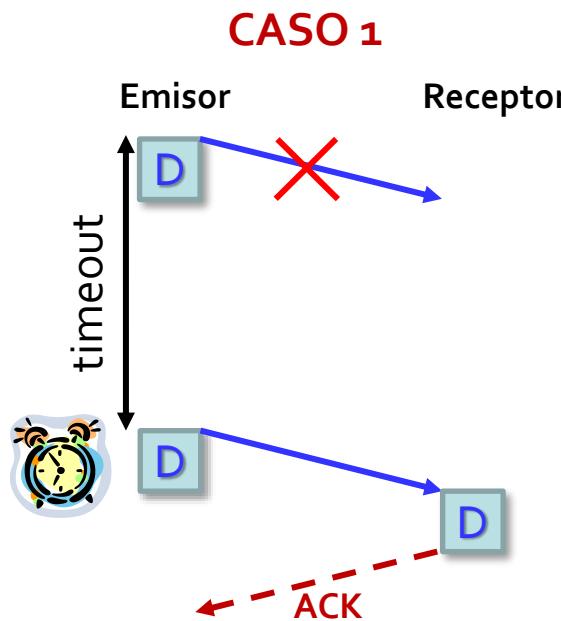


¿Por qué no llega el ACK?

- Se pierde el segmento
- Se pierde el ACK
- El segmento llega con errores

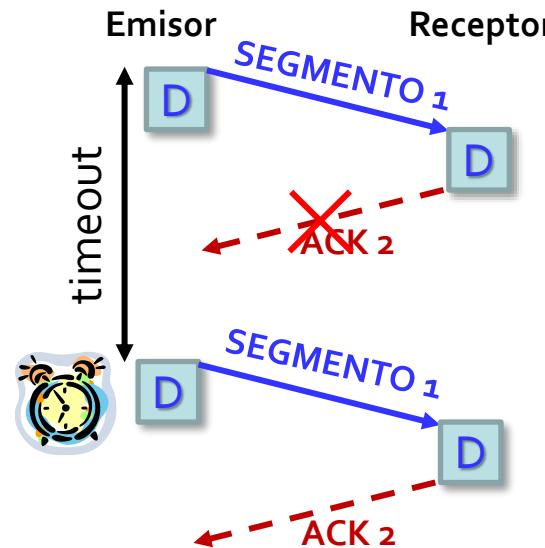
## Detección de pérdida de segmento

- El emisor sigue el siguiente protocolo
  - Espera un tiempo
  - Reenvía el segmento si no llega el reconocimiento



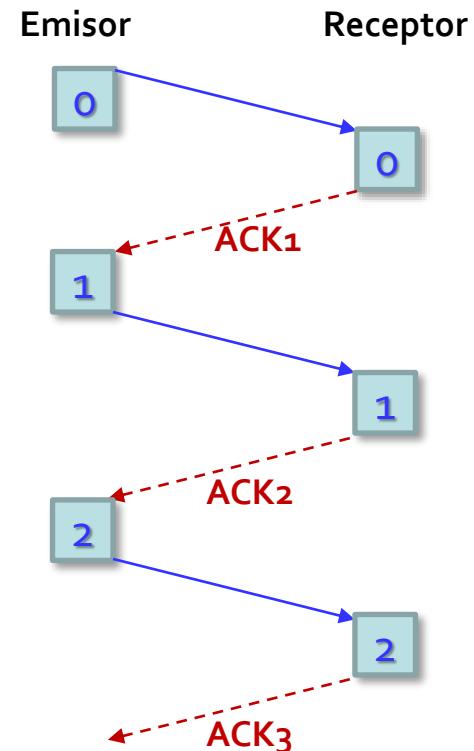
## ¿Cómo detecta el receptor que ha recibido un duplicado?

- Se numeran los segmentos
- Los ACKs también se numeran
- El número del ACK indica el segmento que se espera, no el recibido



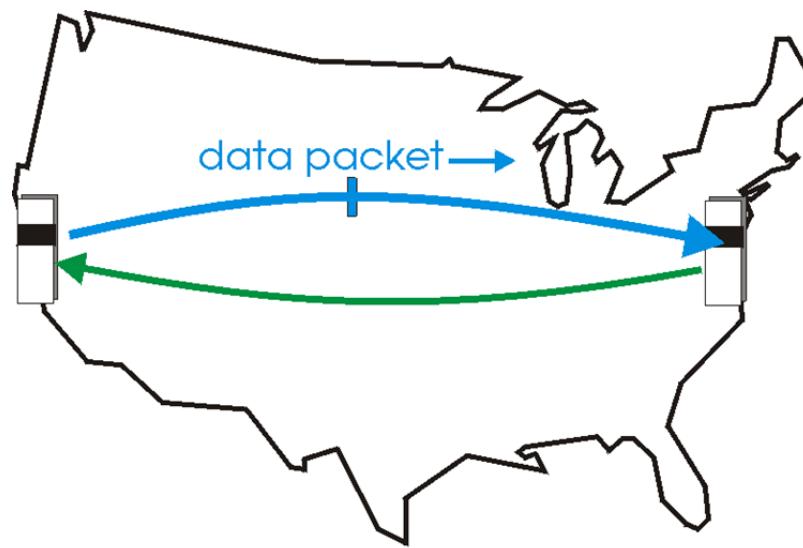
## Stop and wait (parada y espera)

- Se identifican los segmentos
- El emisor sólo puede tener un segmento pendiente de reconocimiento
- Mecanismo sencillo pero ineficiente
  - Se pierde mucho ancho de banda

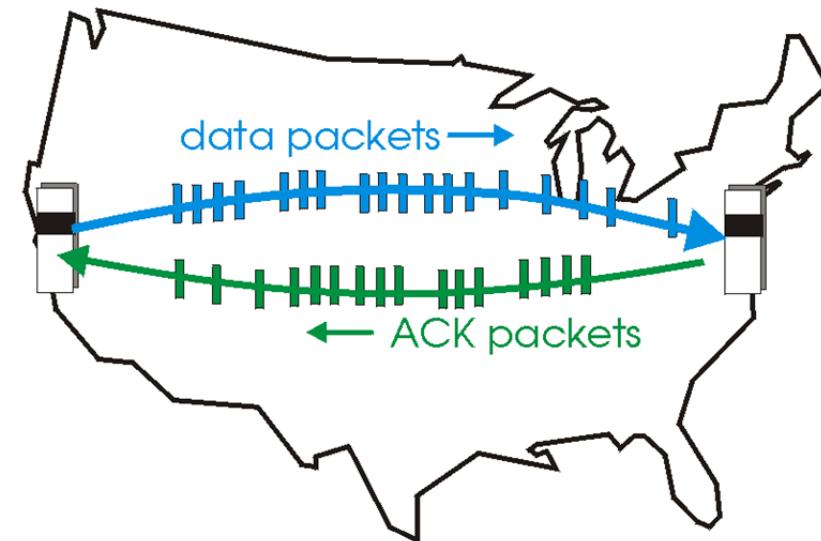


## Mejora de prestaciones

- Varios segmentos por RTT



Parada y espera

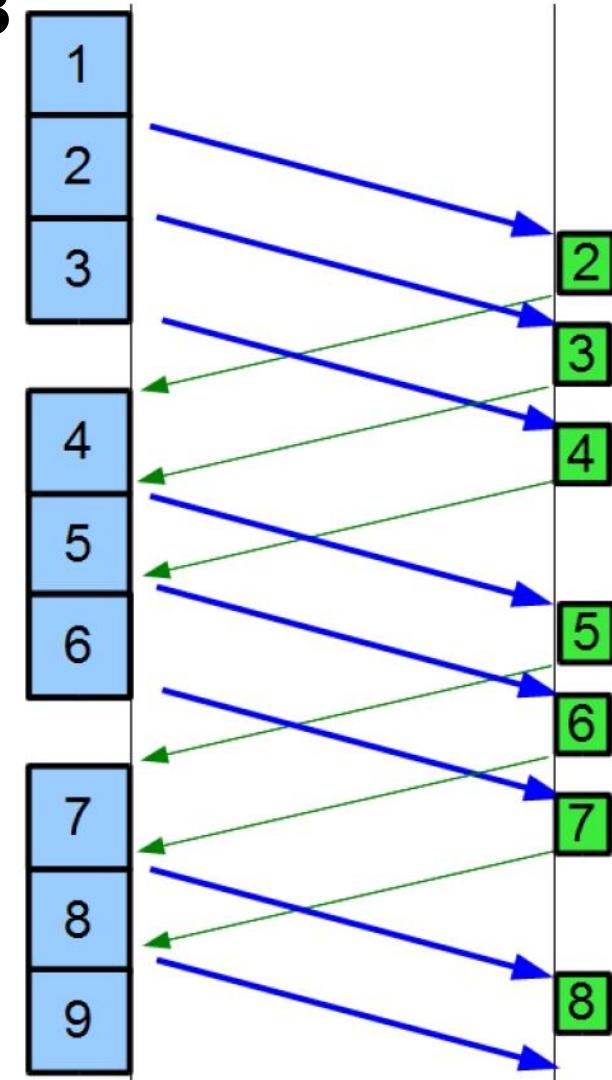


Ventana deslizante

## Ventana deslizante

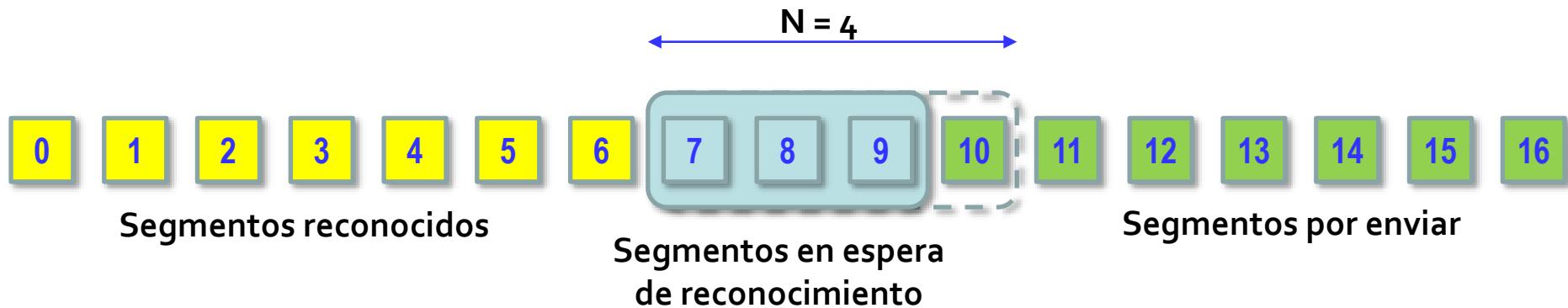
- Mejora de la eficiencia: consiste en enviar varios segmentos por RTT (antes de la llegada del primer reconocimiento). Se llama **pipelining o procesamiento en cadena**
- El emisor debe almacenar los segmentos enviados pendientes de reconocimiento
  - Los almacena en la **ventana de transmisión**:
    - Tamaño variable limitado a N, donde N es el número máximo de segmentos enviados que están pendientes de reconocimiento

N=3



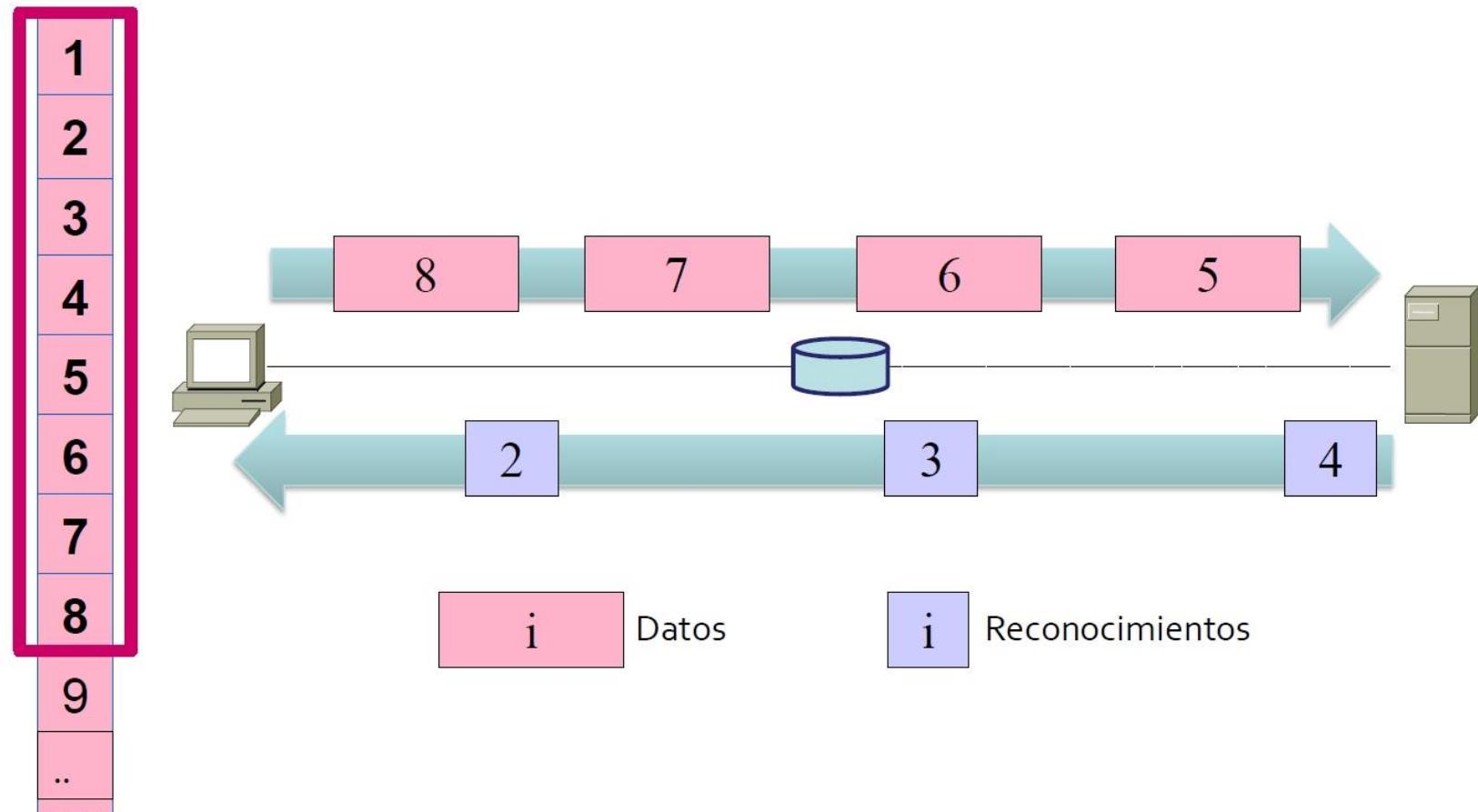
## Ventana de transmisión

- Contiene los segmentos enviados y pendientes de reconocimiento
- De longitud variable, máximo N
- La ventana incrementa su límite inferior al recibir reconocimientos y su límite superior con nuevas transmisiones



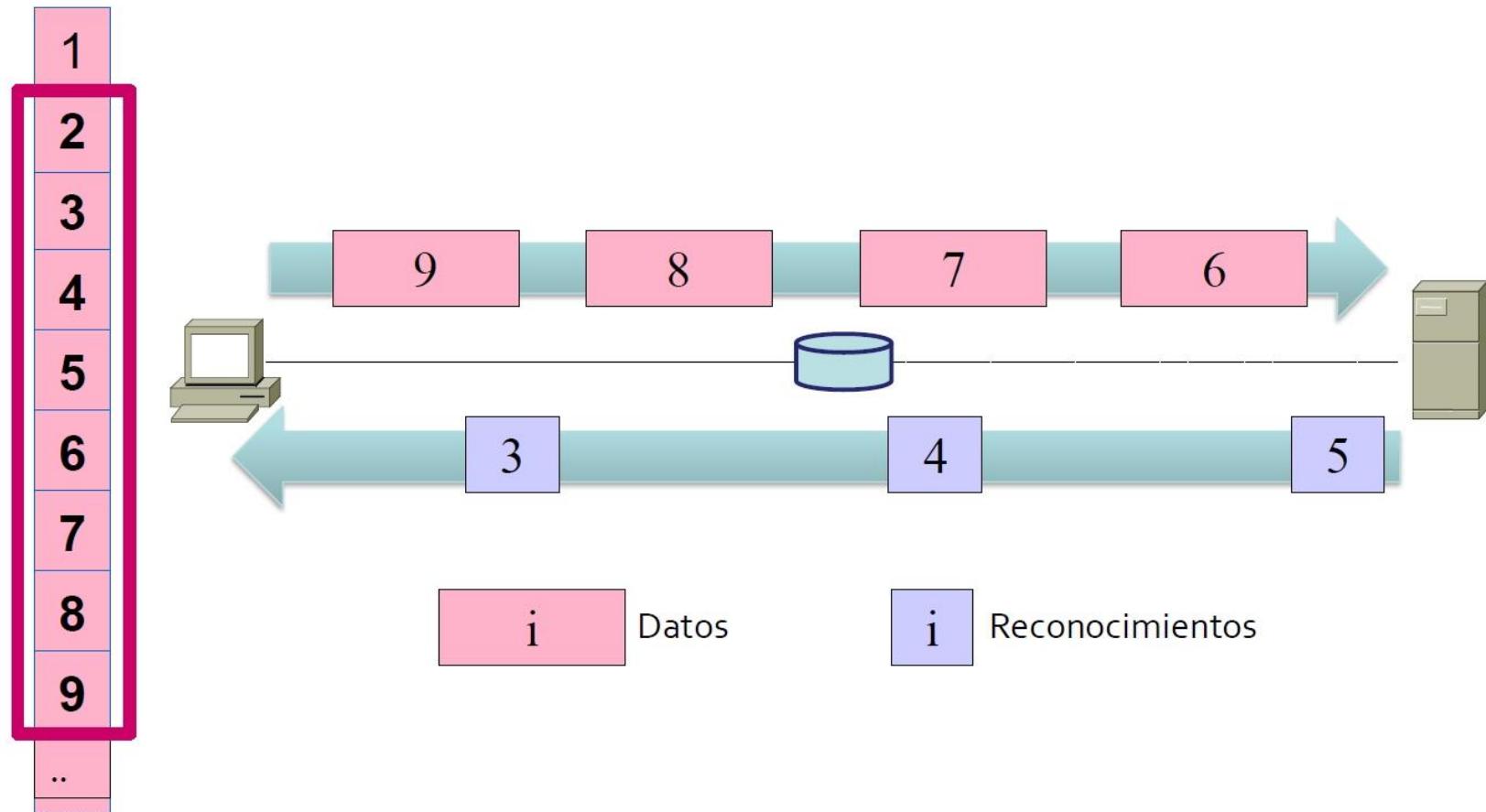
## Ventana de transmisión

Ventana máxima = 8



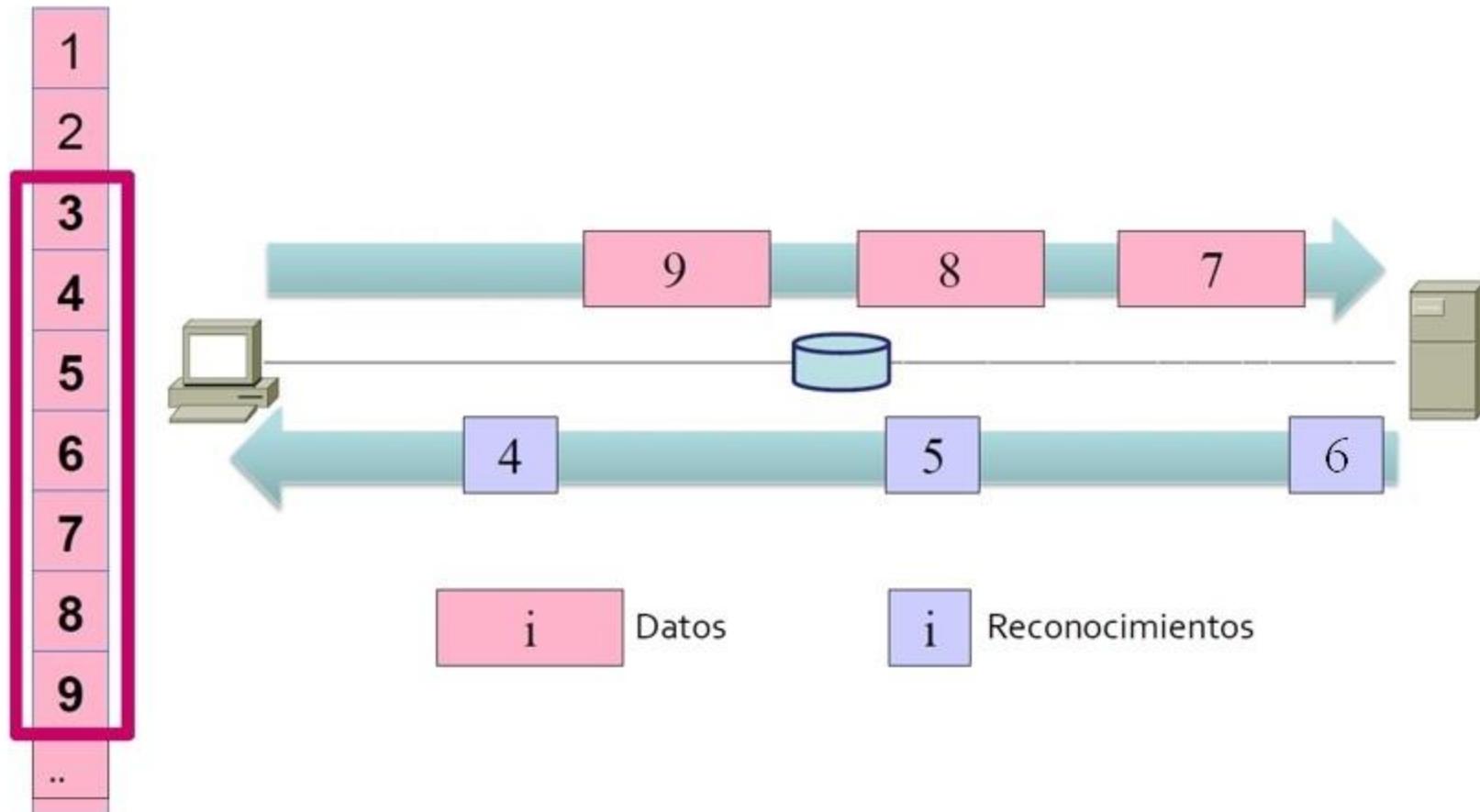
## Ventana de transmisión

Ventana máxima = 8



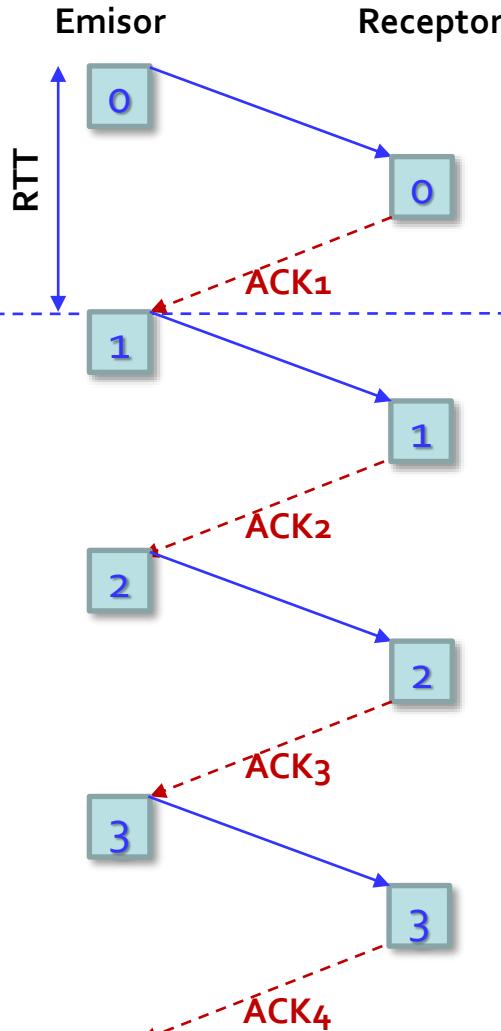
## Ventana de transmisión

Ventana máxima = 8

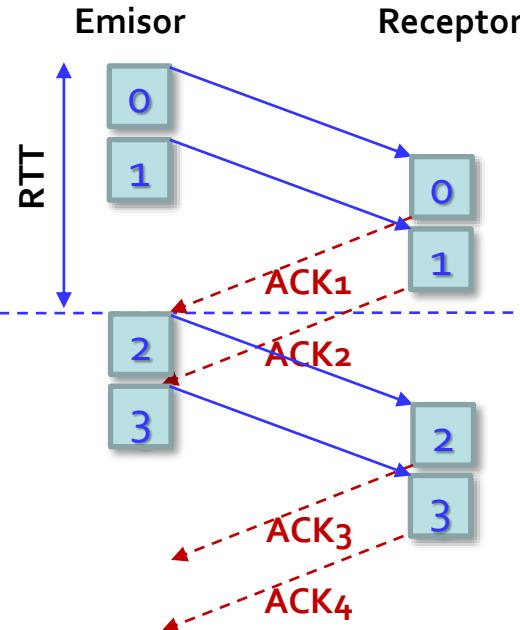


### 3. Transferencia fiable de datos

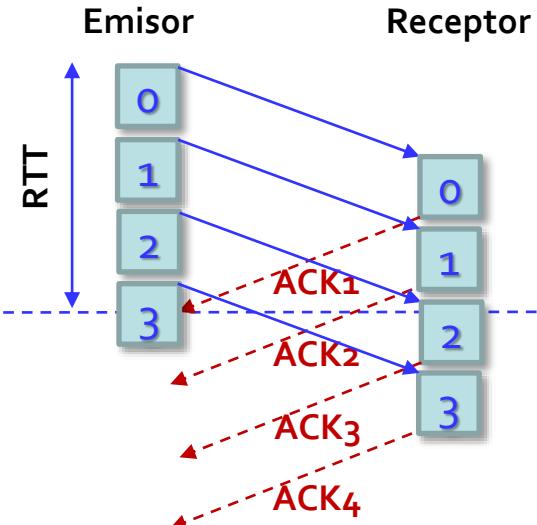
Parada y espera  
(Ventana N=1)



Ventana N=2  
 $N \cdot T_{trans} < RTT$



Ventana N=4  
 $N \cdot T_{trans} \geq RTT$



El envío continuo se produce cuando  
Tamaño de ventana (bits)  $\geq RTT \cdot V_{trans}$

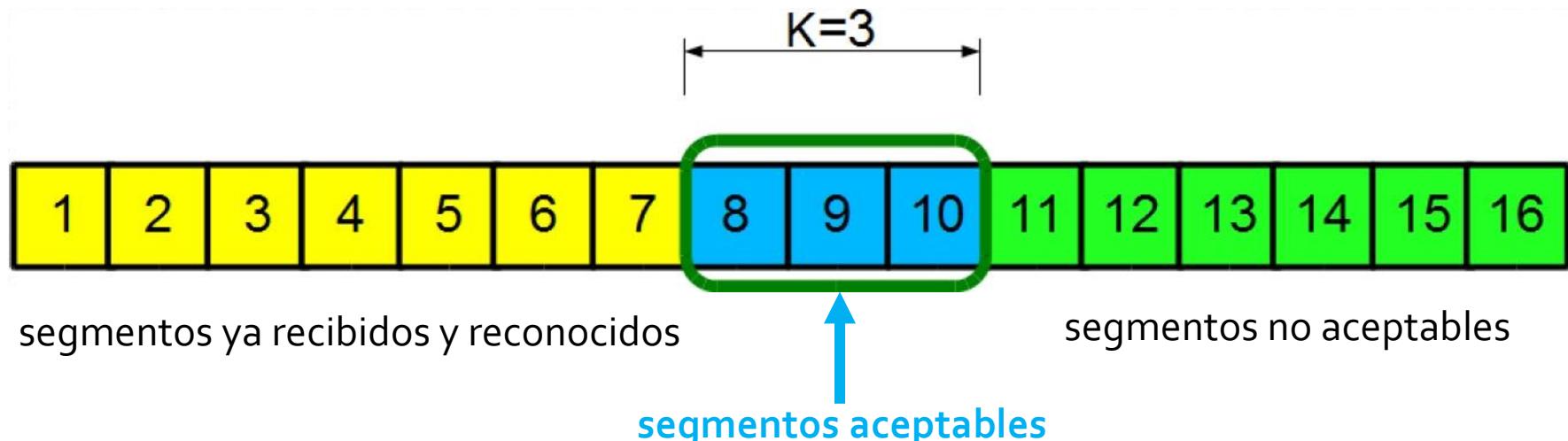
$RTT \cdot V_{trans}$  = bits que caben en un RTT

## Recepción en ventana deslizante

- Los segmentos pueden perderse o llegar desordenados
- ¿Y si llega un segmento fuera de orden?
  - Se rechaza: **vuelta atrás**
  - Se acepta: **retransmisión selectiva**
- Introducimos la **ventana de recepción**
  - Tiene tamaño fijo
  - Incluye los números de secuencia de los segmentos que el receptor puede recibir
- **Mejora:** Los reconocimientos son acumulativos

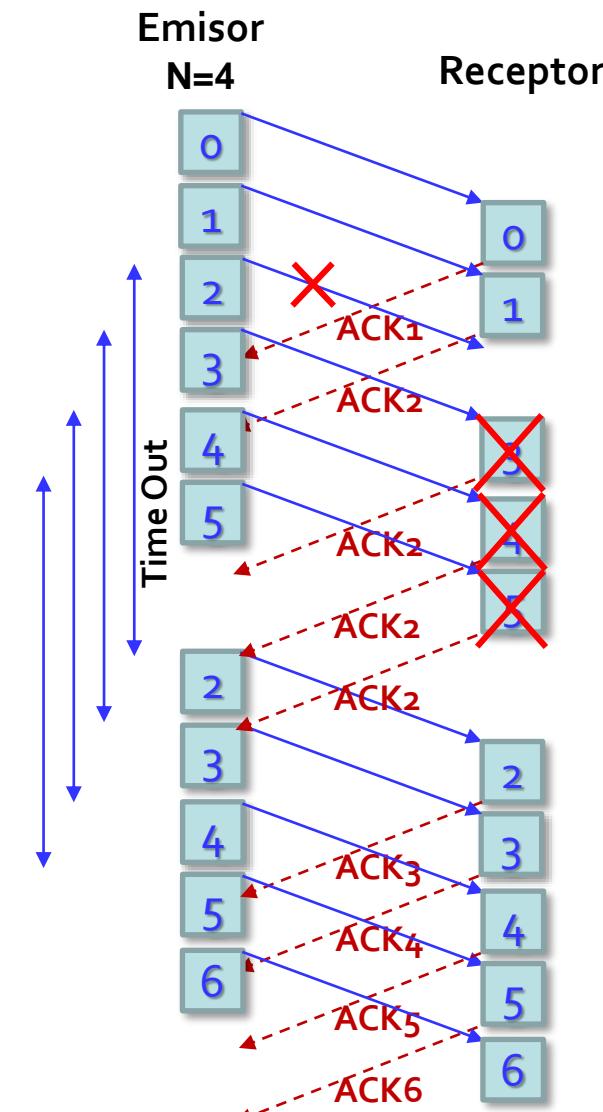
## Ventana de recepción

- Contiene los números de secuencia de los segmentos que el receptor espera recibir
- Tiene una longitud fija de K
  - Vuelta atrás:  $K=1$
  - Retransmisión selectiva:  $1 < K \leq N$



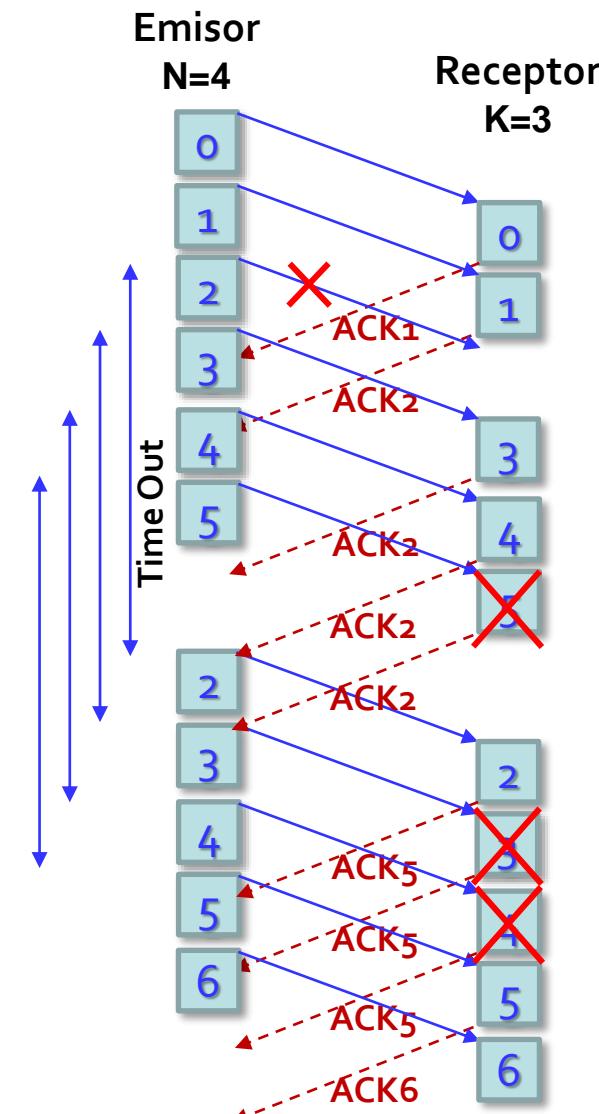
## Vuelta atrás (go-back-n)

- Los paquetes fuera de orden se rechazan



## Retransmisión selectiva

- Los paquetes fuera de orden se aceptan
  - Si están dentro de la ventana de recepción de tamaño K



## Resumen: Tamaño de ventanas

### Ventana de Transmisión

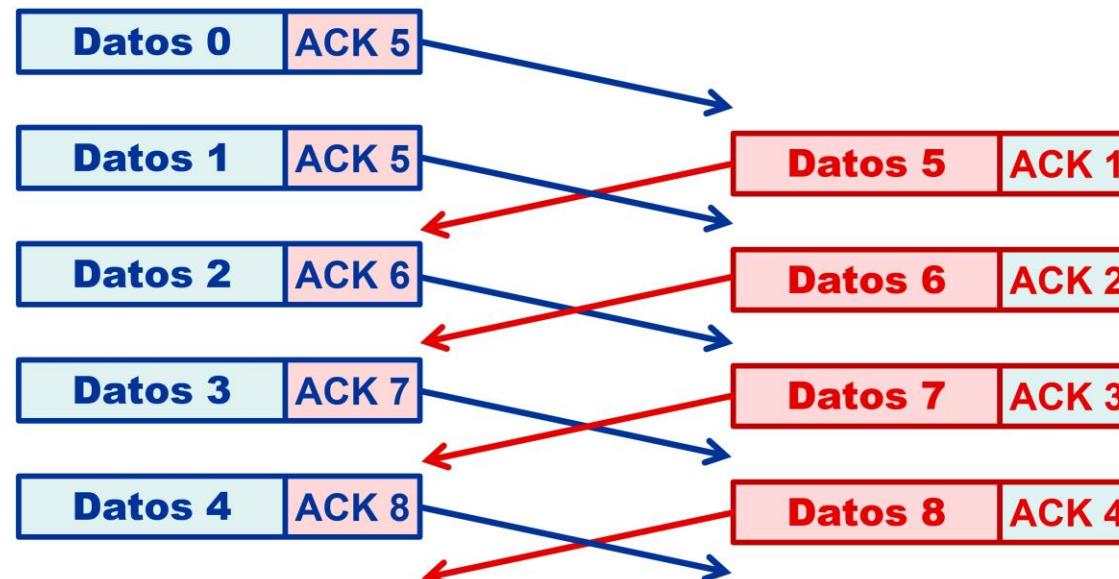
- Tamaño variable
  - Máximo = N
- Segmentos con temporizador de retransmisión asociado
- Tamaño para envío continuo:
  - Nº bits ventana  $\geq RTT * V_{Trans}$

### Ventana de Recepción

- Tamaño fijo
- Vuelta atrás
  - Recepción ordenada
    - Tamaño K = 1
- Retransmisión selectiva
  - Recepción fuera de orden
    - Tamaño  $1 < K \leq N$
- Reconocimientos acumulativos

## Transmisión Bidireccional

- Los protocolos estudiados pueden funcionar en ambos sentidos
- Cada extremo actúa simultáneamente como transmisor y como receptor
- Los reconocimientos se pueden incluir en los paquetes de datos en sentido contrario (**piggybacking**)



1. Servicios del nivel de transporte
2. Transporte sin conexión: UDP
3. Fundamentos de la transferencia fiable de datos
4. **Transporte orientado a la conexión: TCP**
  - 1) Concepto
  - 2) Formato de un segmento
  - 3) Control de flujo
  - 4) Gestión de una conexión TCP
  - 5) Control de error
  - 6) Control de la congestión en TCP
  - 7) Opciones TCP

## 4. Transporte orientado a la conexión: TCP

### Conceptos:

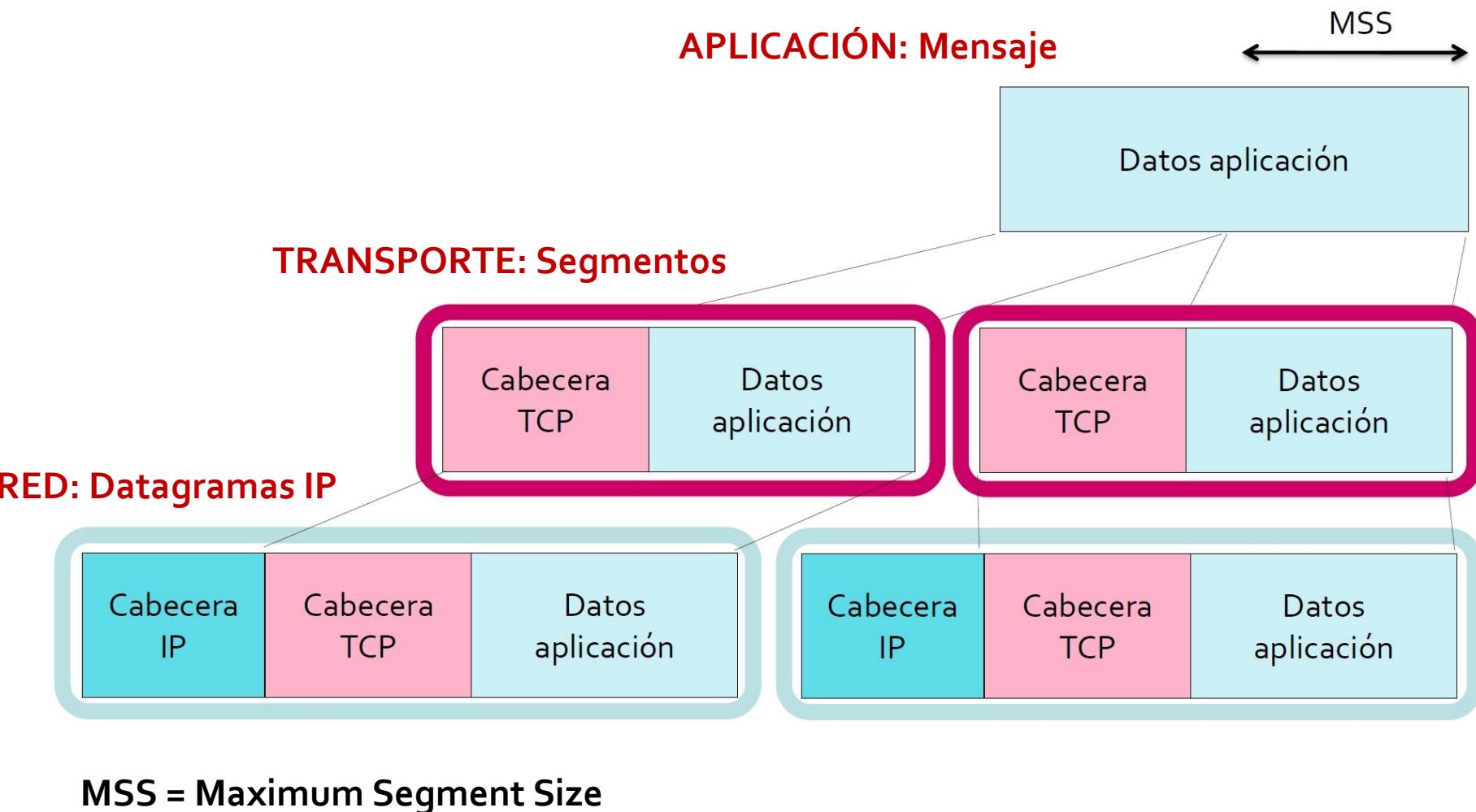
- Transporte orientado a conexión: TCP
  - Concepto
- Estructura de un segmento TCP
- Números de secuencia y reconocimientos
  - Reconocimientos acumulativos
- Estimación del RTT y fin del temporizador
  - Estimación del tiempo de ida y vuelta
  - Duplicación del intervalo de fin de temporización
- Transferencia fiable de datos en TCP
  - Reconocimientos retardados
  - Reconocimientos duplicados
  - Retransmisión rápida
  - Vuelta atrás N o repetición selectiva
- Control de flujo
  - Ventana de recepción
  - Funcionamiento

### TCP (Transmission Control Protocol)

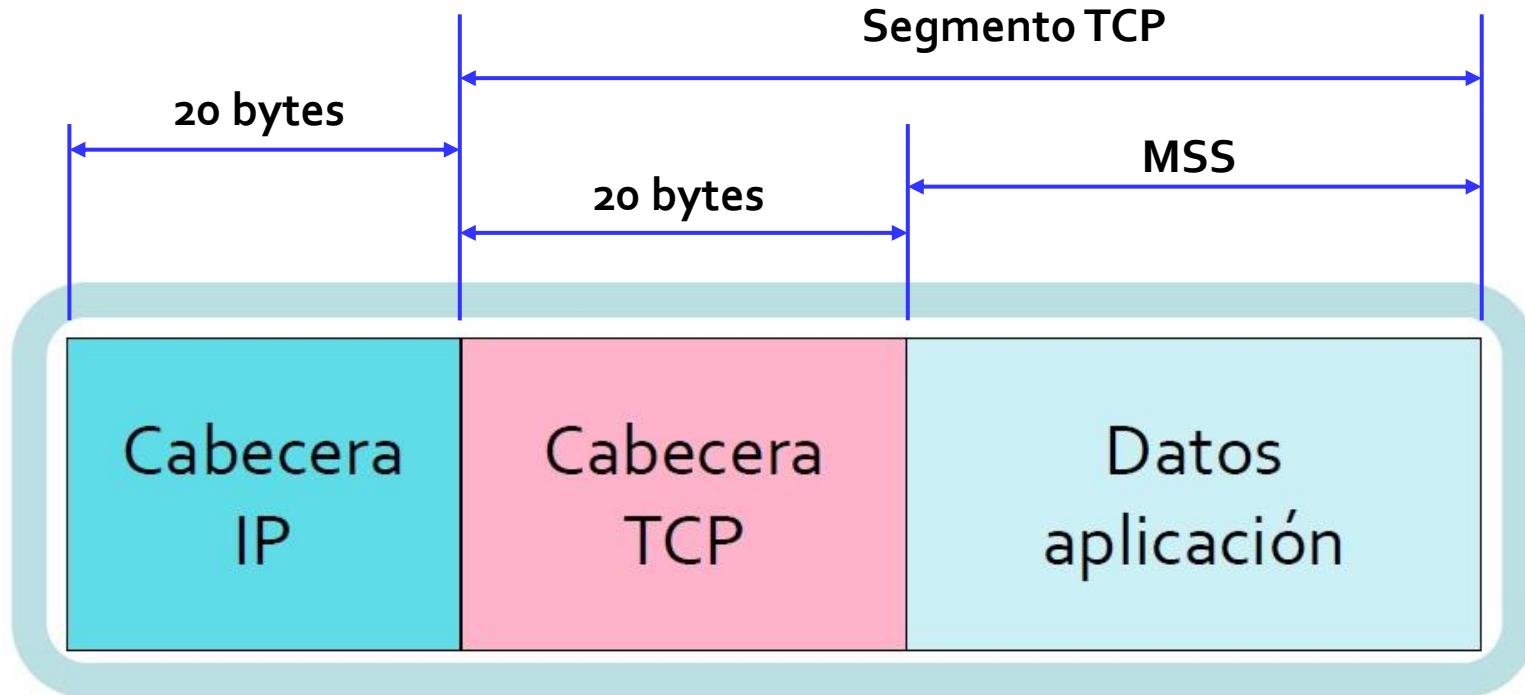
- RFC's 793, 1122, 1323, 2018, 2581
- Servicio orientado a la conexión:
  - Entrega en orden
  - Fiable (retransmisiones)
  - Punto a punto
  - Control de flujo
- Flujo de bytes ordenado entre aplicaciones
  - No tiene en cuenta las fronteras entre mensajes
  - MSS: Maximum Segment Size
- Datos full dúplex
  - Bidireccional sobre la misma conexión
- Control de Congestión
- Mucha información de estado asociada a la conexión

## 4. Transporte orientado a la conexión: TCP

### Encapsulado de TCP



## Maximum Segment Size (MSS)



- Intenta evitar la fragmentación de IP
- Cada extremo anuncia su MSS al establecer la conexión
  - No se le pueden enviar segmentos mayores que MSS
- Por defecto  $MSS = 536$  octetos

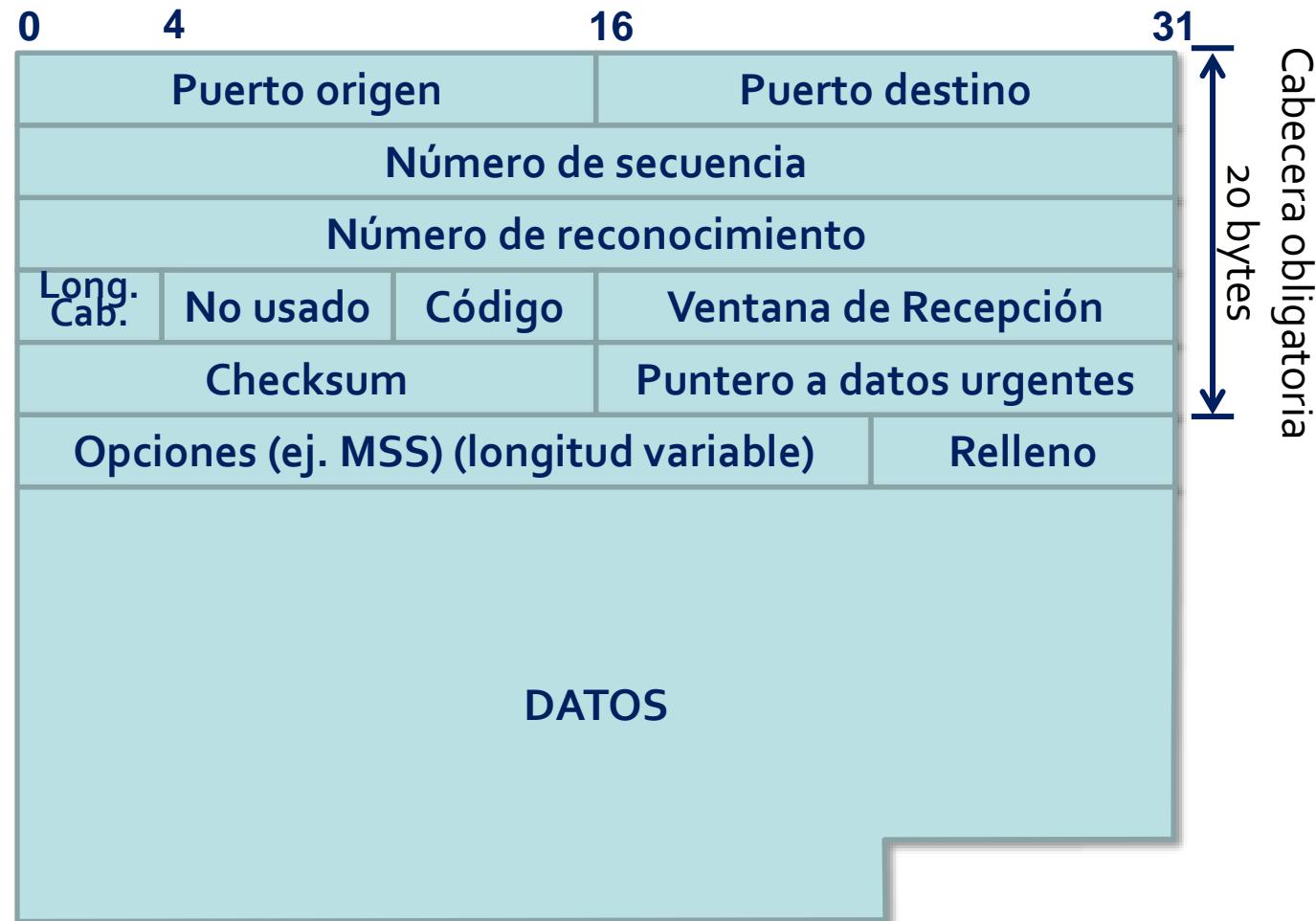
### 4. Transporte orientado a la conexión: TCP

- 1) Concepto
- 2) **Formato de un segmento**
- 3) Control de flujo
- 4) Gestión de una conexión TCP
- 5) Control de error
- 6) Control de la congestión en TCP
- 7) Opciones TCP

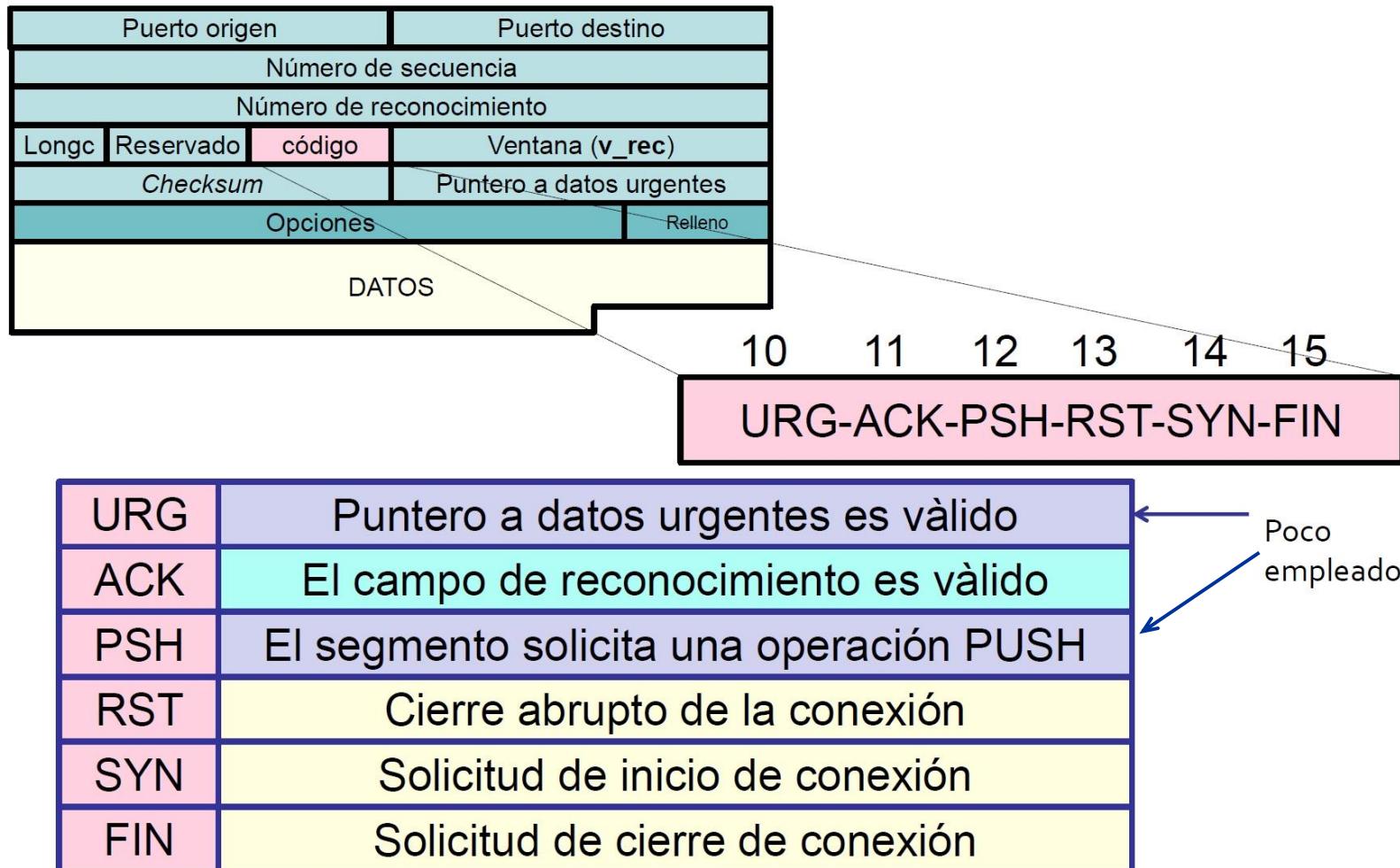
## Formato del segmento TCP

La **cuenta de datos** se lleva **en bytes**:

- Todos los campos que se refieran a los datos se contarán **en bytes**



## Campo de código (flags)

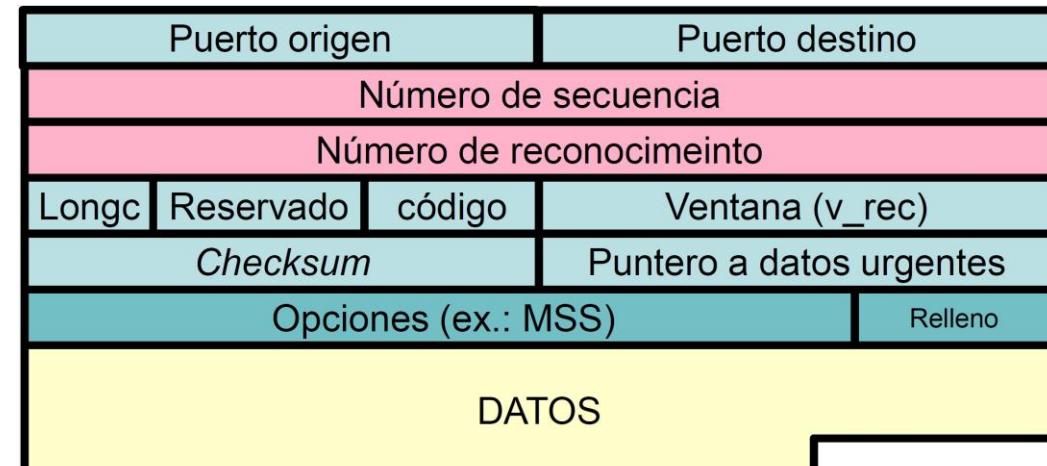


### Checksum en TCP

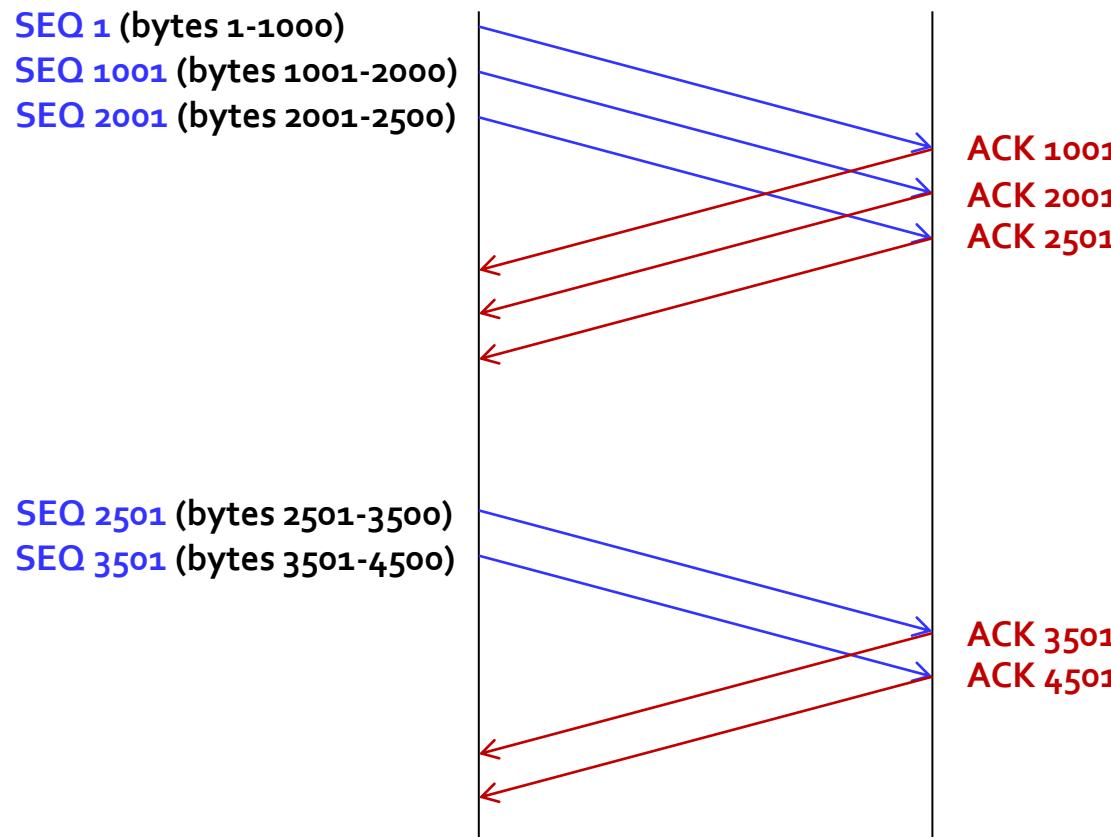
- El checksum asegura que los segmentos no han sido modificados en tránsito
- Cálculo similar al checksum UDP
  - Se aplica a todo el segmento (cabecera+datos)
- Un segmento dañado se descarta
  - Tendrá que ser retransmitido

## Numeración de segmentos

- En TCP se utilizan números de 32 bits
  - Número de secuencia:
    - Indica el número de orden del primer byte de datos que viaja en el segmento
    - El número de secuencia inicial se elige de forma aleatoria
  - Número de reconocimiento:
    - Indica el byte que el receptor espera recibir
    - Es acumulativo
    - Válido si el flag ACK = 1
  - Reconocimientos superpuestos (piggybacking)



## Ejemplo: numeración de segmentos



Como el tránsito de datos es bidireccional\* los reconocimientos pueden viajar **superpuestos** (piggybacking)

\*aunque no se muestre en la figura

### 4. Transporte orientado a la conexión: TCP

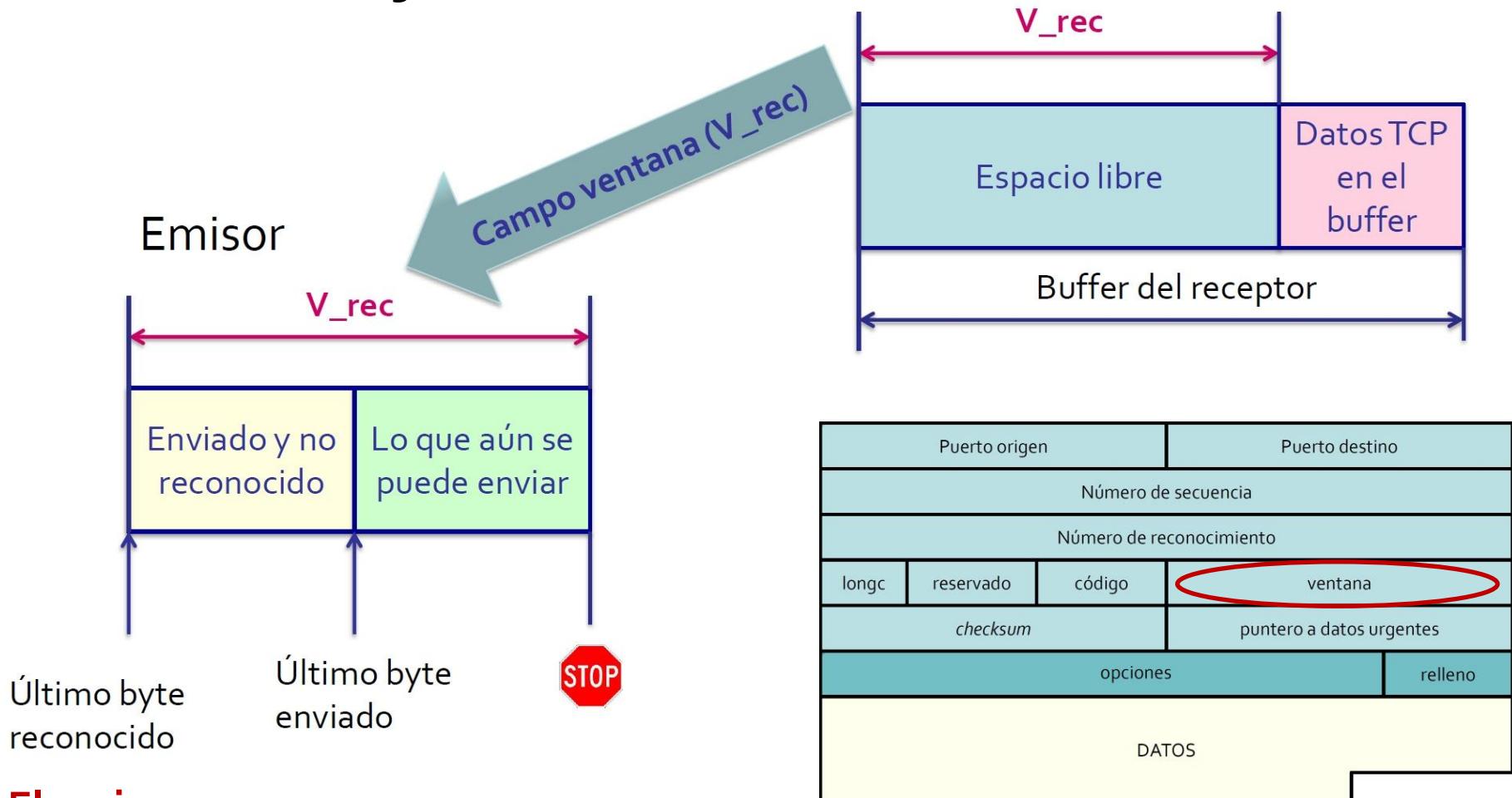
- 1) Concepto
- 2) Formato de un segmento
- 3) **Control de flujo**
- 4) Gestión de una conexión TCP
- 5) Control de error
- 6) Control de la congestión en TCP
- 7) Opciones TCP

## Control de flujo en TCP

- Cada extremo dispone de un buffer de recepción
- Los datos recibidos correctamente y en orden se almacenan en el buffer
- El proceso de aplicación lee los datos “a su ritmo”
  - Si es lento en las lecturas, el emisor podría provocar un desbordamiento del buffer
  - Solución: control de flujo mediante una **ventana de recepción (V\_rec)**
    - La ventana de recepción indica el espacio libre en el receptor
    - Es dinámica y puede tomar el valor cero

## 4. Transporte orientado a la conexión: TCP

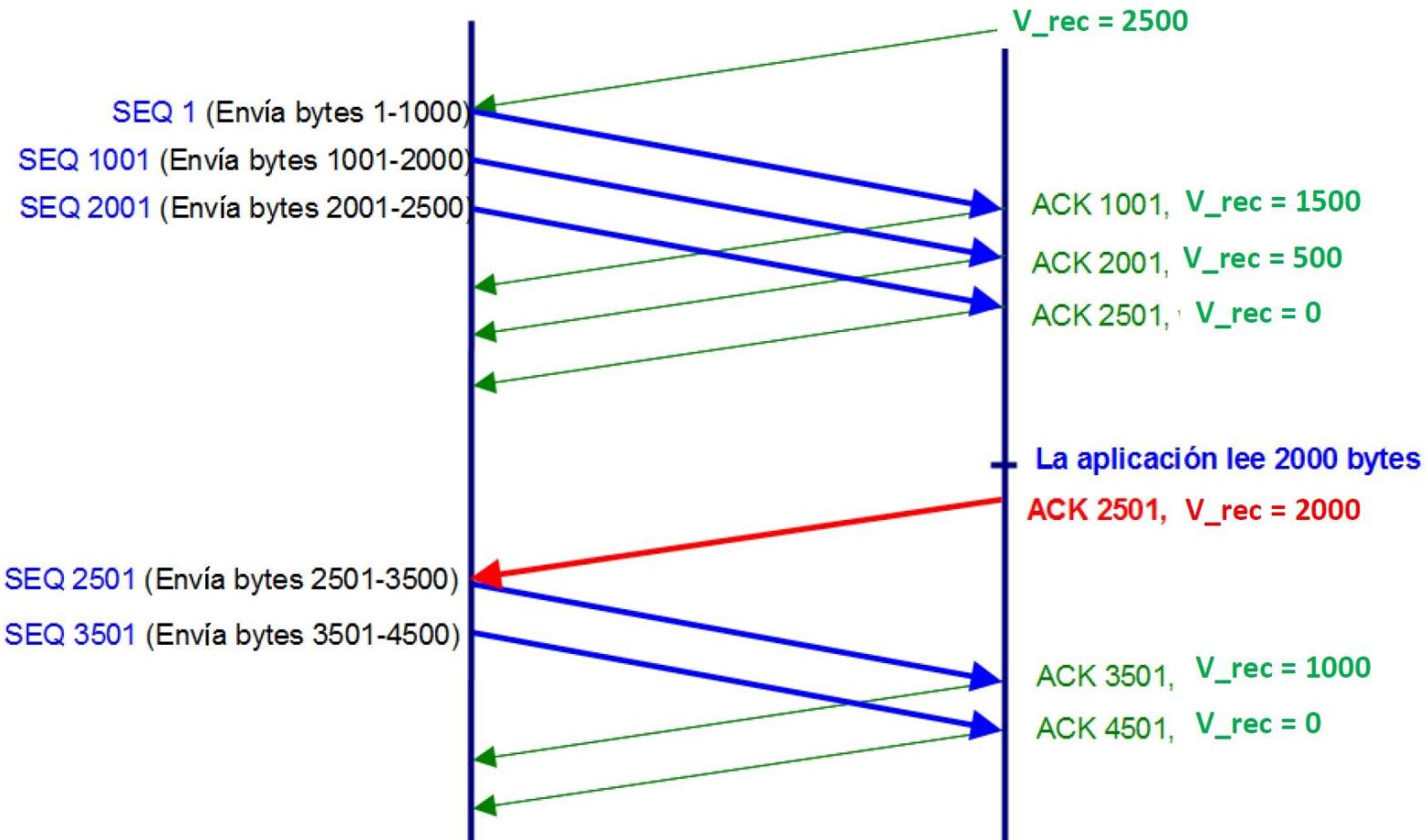
### Control de flujo en TCP



**El emisor asegura que:**

**Último Byte enviado – Último Byte reconocido  $\leq V_{rec}$**

## Control de flujo en TCP



No se envía ningún dato que no “quepa” en el receptor

## 4. Transporte orientado a la conexión: TCP

- 1) Concepto
- 2) Formato de un segmento
- 3) Control de flujo
- 4) Gestión de una conexión TCP
- 5) Control de error
- 6) Control de la congestión en TCP
- 7) Opciones TCP

### Conceptos:

- Establecimiento de la conexión
  - El bit SYN
  - Acuerdo en tres fases
- Cierre de la conexión
  - El bit FIN
  - Acuerdo en tres o cuatro fases
- El bit RST

### Gestión de la conexión TCP

#### 1. Establecimiento de conexión

- `s = new Socket("NombreHost", NumPuerto);`
- intercambio de mensajes

#### 2. Transmisión de datos

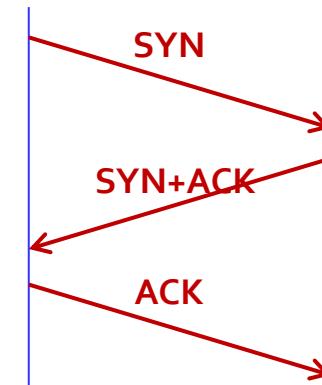
- bidireccional
- `entrada.read(b)` y `salida.write(b)`

#### 3. Cierre de la conexión

- `s.close();`

### Establecimiento de la conexión TCP

- Protocolo en tres fases
  - 1) Petición
  - 2) Aceptación
  - 3) Reconocimiento
    - Cada extremo debe pedir y reconocer la petición del otro



## Establecimiento de la conexión TCP

(Apertura activa)

## Cliente

`s = new Socket(...);`Envía **SYN x**  
número de secuencia  
inicial del clienteRecibe **SYN y, ACK x+1**  
Envía **x+1, ACK y+1**puede llevar datos  
del cliente

(Apertura pasiva)

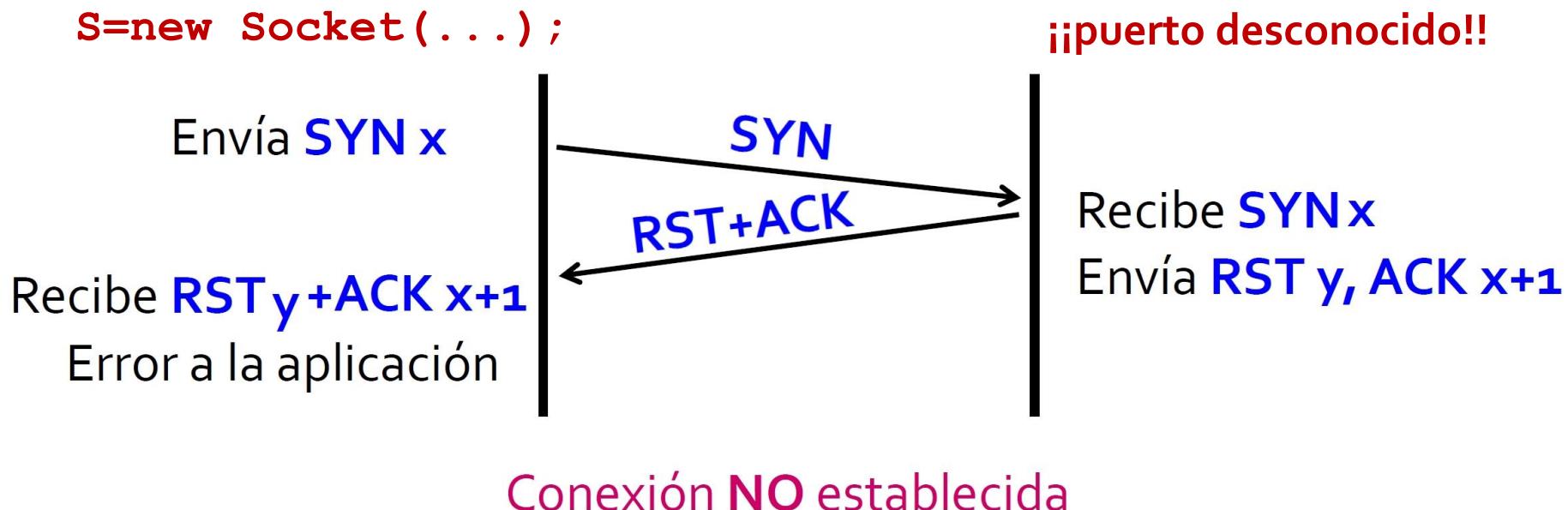
## Servidor

`ss = new ServerSocket(pto);  
ss.accept();`Recibe **SYN x**  
Envía **SYN y, ACK x+1**  
número de secuencia  
inicial del servidorRecibe **ACK y+1**

Conexión establecida

## El bit RST

- RESET: aborta la conexión TCP
- Causas:
  - Número de secuencia imposible
  - El puerto destino no está en uso (no ServerSocket)



## Cierre de la conexión TCP

- Cualquiera de los dos procesos puede finalizar la conexión
- Cierre de los dos flujos de datos de forma **independiente**

`s.close();`

Envía **FIN x**

Recibe **ACK x+1**  
(Conexión semicerrada)

Recibe **FIN y+ACK x+1**

Envía **ACK y+1**  
(Conexión cerrada)

**FIN**

**ACK**

**FIN+ACK**

**ACK**

(Informa a la aplicación)

Envía **ACK x+1**

`s.close();`

Envía **FIN y**

Recibe **ACK y+1**  
(Conexión cerrada)

### 4. Transporte orientado a la conexión: TCP

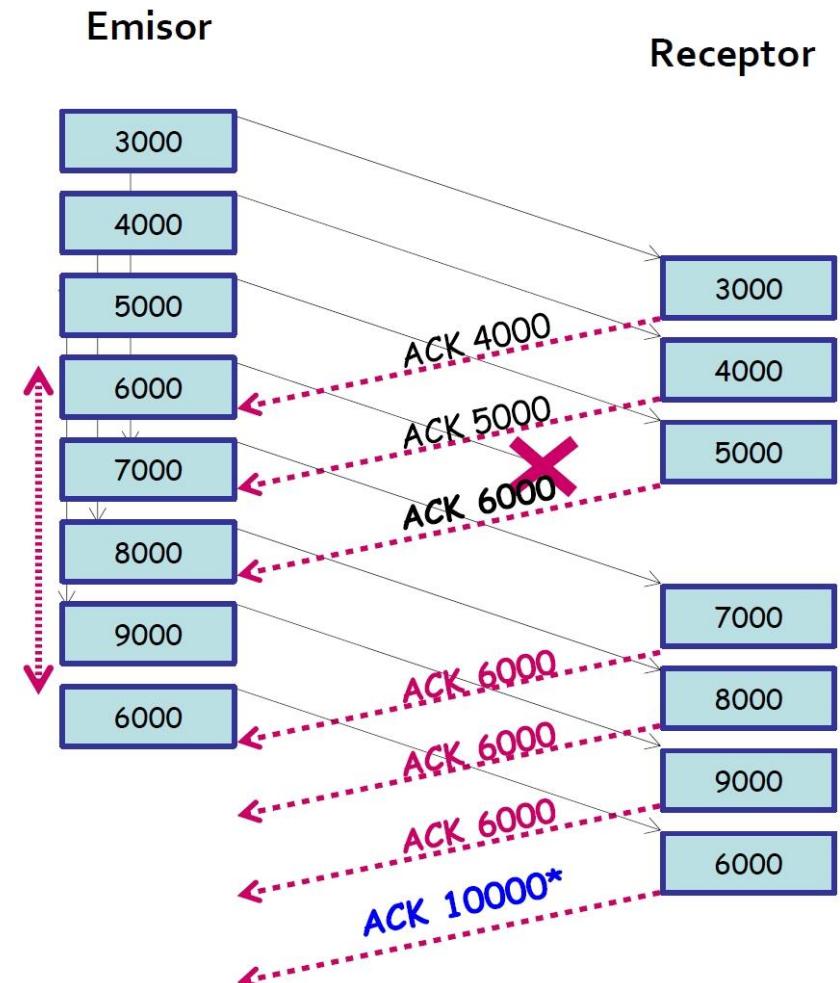
- 1) Concepto
- 2) Formato de un segmento
- 3) Control de flujo
- 4) Gestión de una conexión TCP
- 5) **Control de error**
- 6) Control de la congestión en TCP
- 7) Opciones TCP

### Control de error en TCP

- Utiliza ARQ (Automatic Repeat reQuest):
  - Reconocimientos (ACK) y retransmisiones
- Detección de los segmentos erróneos: checksum
- **Ventana deslizante**
  - Ventana de transmisión de tamaño máximo variable
    - Fijado por el receptor (campo ventana recepción, V\_rec)
  - Si los datos no llegan en orden (no definido en el RFC)
    - Se pueden descartar: **vuelta atrás**
    - Se pueden aceptar: **retransmisión selectiva**

### Reconocimientos en caso de pérdidas

- Reconocimientos acumulativos:
  - Cuando se recibe un segmento dañado o se pierde, los segmentos siguientes NO se pueden reconocer!!!
- Tras la recepción de los segmentos 3000, 4000, 5000, 7000, 8000 y 9000, el reconocimiento será 6000
- Cada recepción puede generar un reconocimiento



\* Suponiendo retransmisión selectiva

### Generación de reconocimientos

- No es necesario un ACK por segmento recibido
- **Reconocimientos retardados (RFCs 1122, 2581):**
  - El ACK se puede retrasar hasta:
    - Recibir otro segmento (se reconocen uno de cada dos)
    - Enviar un segmento de datos en sentido contrario
    - Que venza un temporizador (cada 500 milisegundos)
- El objetivo es reducir el tráfico de reconocimientos

### Envío de datos: temporizador

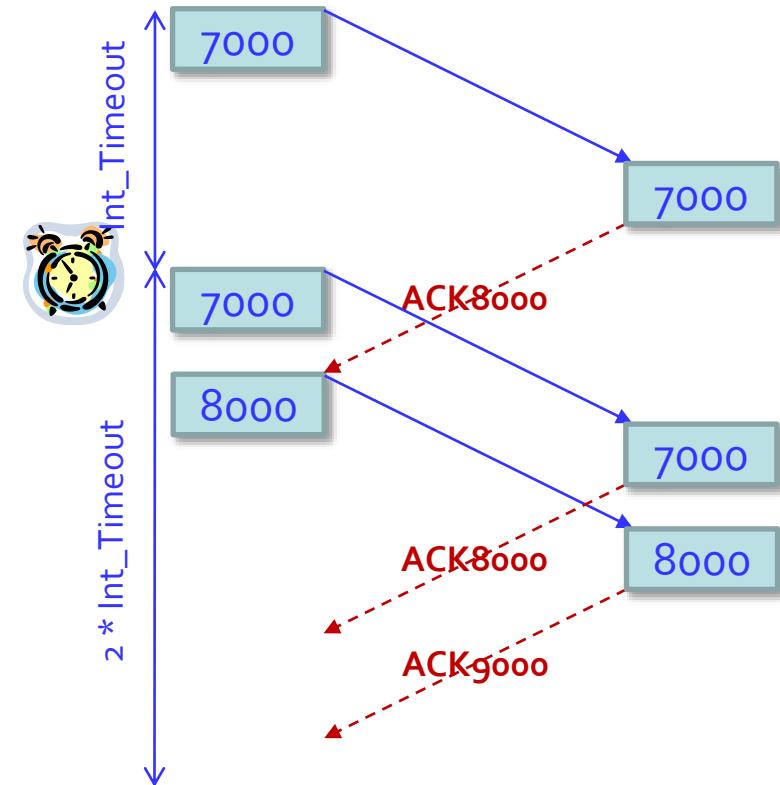
- Al enviar un segmento se pone en marcha un temporizador (si no está ya en marcha)
  - Normalmente se emplea un único temporizador
- El reconocimiento del segmento desactiva su temporizador de retransmisión
  - Si quedan segmentos pendientes de reconocimiento, se reinicia el temporizador
- Si vence el temporizador se produce el reenvío de al menos un segmento (el primero sin reconocer de la ventana) y se reinicia el temporizador

## Temporizador RTO y RTT

- ¿Cuál es el tiempo adecuado de espera para el temporizador de retransmisión **RTO (Retransmission TimeOut)**?
  - Si es demasiado corto: retransmisiones innecesarias
  - Si es demasiado largo: poco eficiente
  - Debería ser un intervalo mayor que el RTT
  - Pero ...
    - el RTT es variable: depende de los retrasos de la red
      - Cantidad de tráfico en cada uno de los routers
      - Velocidades de transmisión y propagación de cada una de las líneas empleadas
    - es imposible calcular el RTT a priori
- TCP utiliza una predicción dinámica basada en la evolución reciente de la conexión:
  - Toma muestras del RTT de cada segmento y su ACK correspondiente

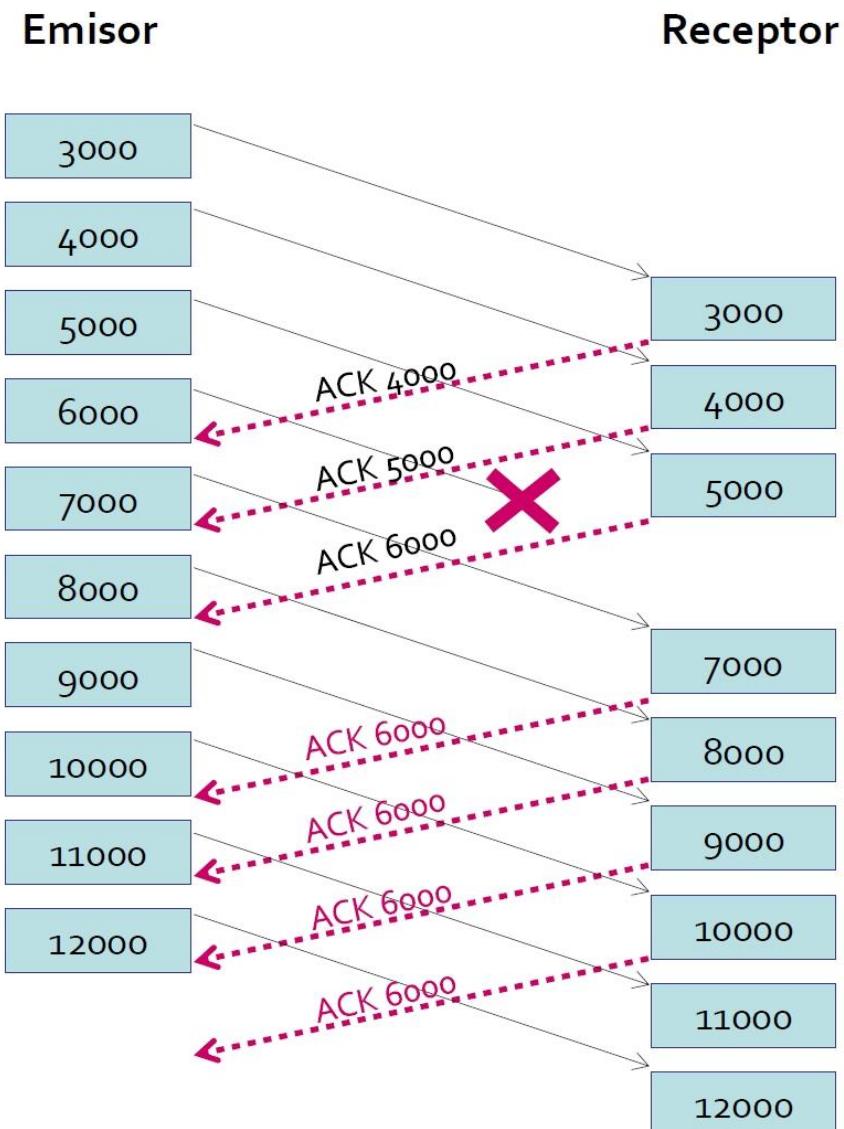
## Duplicación del timeout (RFC 6298)

- Problema:
  - Ambigüedad en el RTT cuando hay retransmisiones
  - Se puede confundir el ACK del segmento original con el de la retransmisión
  
- Solución:
  - No tener en cuenta las muestras del RTT de los segmentos retransmitidos
  - **Al retransmitir doblar el valor del temporizador**
    - Exponential Backoff



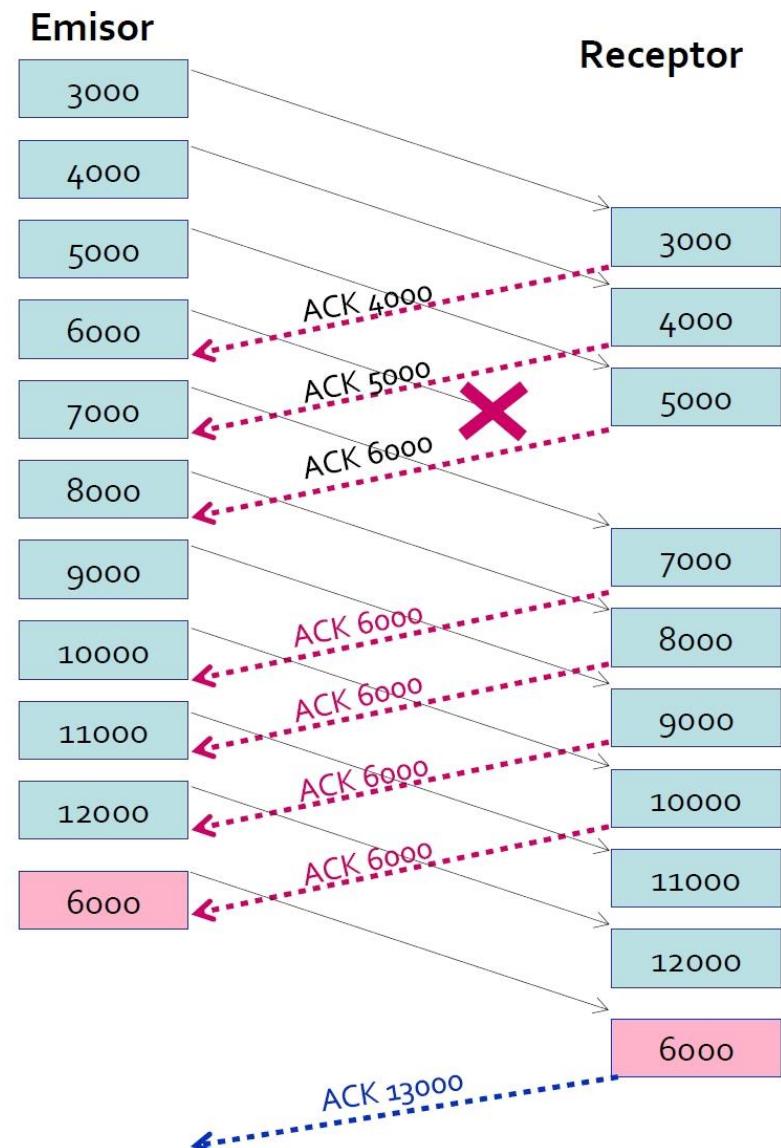
## Reconocimientos duplicados

- Se deben a la llegada de un segmento fuera de orden
  - Debido a entrega desordenada o pérdida
  
- Se envía un ACK por segmento recibido fuera de orden
  - ¡Inmediatamente!!



## Retransmisión rápida

- El intervalo de timeout suele ser grande
  - Se pueden acelerar las retransmisiones si cuando se reciben **tres o más ACKs duplicados...**  
... se retransmite el segmento perdido sin esperar a que venza su temporizador (**retransmisión rápida**, RFC 2581)



### Resumen de la generación de ACK en TCP

Evento en el receptor	Acción del receptor
Llegada de segmento en orden con el nº de secuencia esperado. Todos los datos hasta el nº secuencia ya con ACK	ACK retardado. Esperar 500 ms al siguiente segmento. Si no llega el próximo segmento, mandar ACK
Llegada de segmento en orden con nº de secuencia esperado. Otro segmento tiene el ACK pendiente	Enviar inmediatamente un ACK acumulativo reconociendo los dos segmentos ordenados
Llegada de un segmento desordenado con un nº de secuencia mayor que el esperado. Detección de hueco	Enviar inmediatamente un ACK duplicado, indicando el nº sec. del byte realmente esperado (el límite inferior del hueco)
Llegada de un segmento que completa parcial o totalmente el hueco existente en los datos recibidos (por el límite inferior)	Enviar inmediatamente un ACK para reconocer el mayor byte recibido en orden

## 4. Transporte orientado a la conexión: TCP

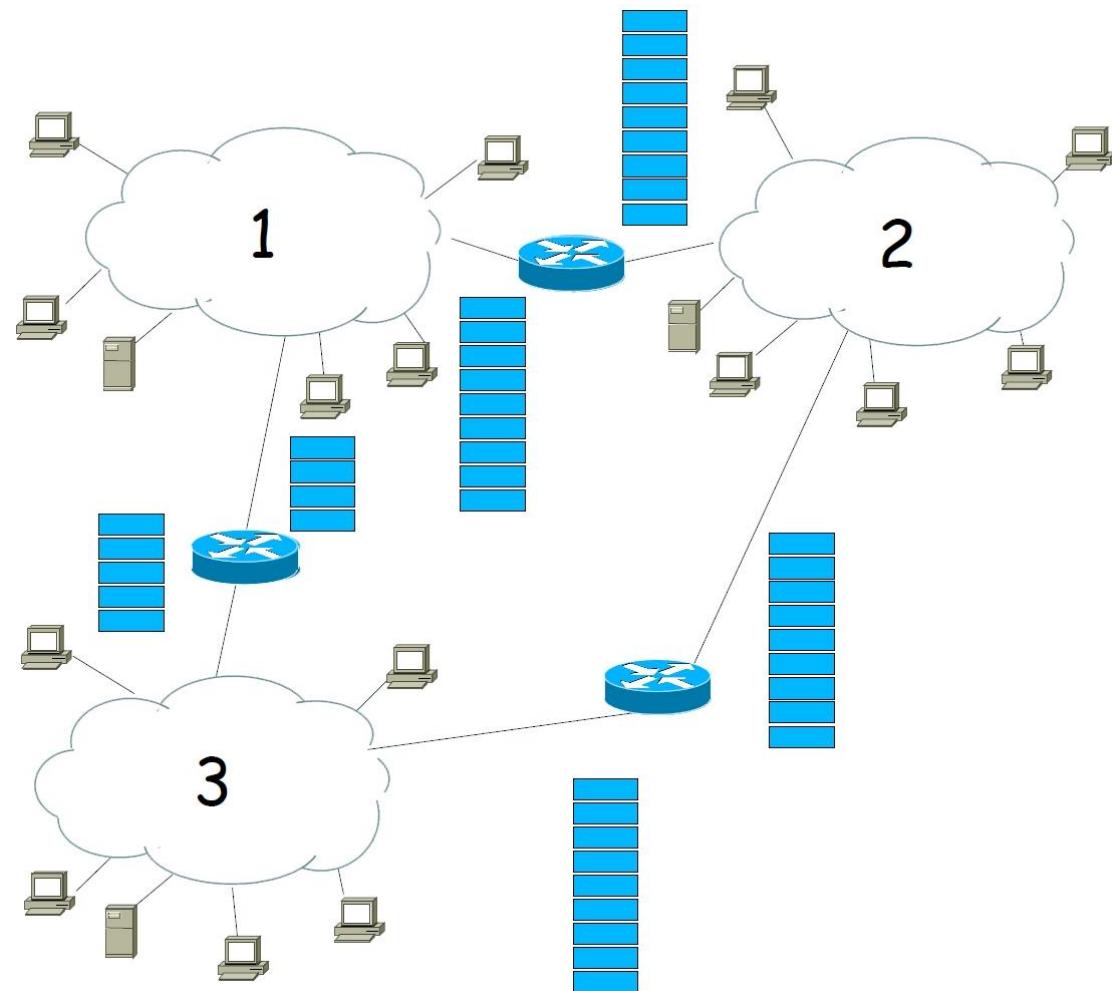
### 6) Control de la congestión en TCP

#### Conceptos:

- El problema de la congestión en la red
- Mecanismo de control de la congestión en TCP
  - Principios de funcionamiento
    - Ventana de congestión
  - Síntomas de congestión
    - Vencimiento del temporizador
    - Recepción de tres ACK duplicados
  - Algoritmo de control de la congestión en TCP
    - Arranque lento
    - Evitación de la congestión
    - Recuperación rápida

## La congestión en la red

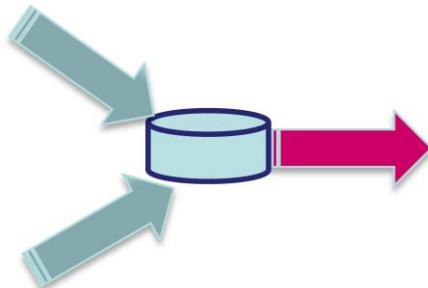
- Los routers tienen capacidad limitada
  - colas (buffers)
- Si las colas se llenan, los routers descartan paquetes



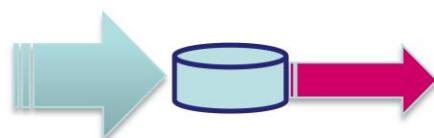
### La congestión en la red

#### Causas básicas

- Enlaces compartidos en los routers



- Tránsito a enlaces con menor ancho de banda que los precedentes



#### Consecuencias

- Grandes retardos en las colas cuando la tasa de llegada se aproxima a la capacidad del enlace
  - El incremento del RTT puede generar retransmisiones innecesarias
- Descarte de paquetes
  - Desperdicio de los recursos ya empleados
- Retransmisiones:
  - Incrementan aún más el tráfico de la red

## Control de la congestión en TCP

- El comportamiento de TCP no sólo garantiza la fiabilidad de la comunicación
- Además pretende:
  - **Reducir la congestión**
    - Sin disponer de información proporcionada por IP
    - Sólo con la información que se detecta en el transmisor
  - Propiciar un **reparto equitativo** de la capacidad disponible entre todos los flujos TCP presentes
- **Método:** limitar el envío de tráfico de cada emisor en función de la congestión de red percibida

### Cuestiones para el emisor

- El emisor deberá resolver tres problemas:
  1. Cómo detectar que existe congestión
  2. Cómo limitar el envío de tráfico
  3. Cómo variar la tasa de transmisión en función de la congestión percibida ¿qué algoritmo emplear?

### 1. Cómo detectar que existe congestión

- TCP supone que la pérdida de un segmento se debe siempre a la congestión
- Dos posibilidades para detectar congestión:
  - Vencimiento de un temporizador (timeout)
  - Recepción de tres ACKs duplicados
  - ¿Debe reaccionarse igual? ¿Tienen la misma gravedad?

### 2. Cómo limitar el envío de tráfico

- Se basa en un límite adicional: **Ventana de congestión (V<sub>cong</sub>)**
  - Corresponde a una estimación de cuánto tráfico puede transmitir la red para cada conexión
  - La ventana de congestión depende de las condiciones de la red
  - La ventana de congestión **limita el tamaño máximo de la ventana de transmisión** (bytes de datos que se pueden enviar)
  - **VentanaTransMax = mínimo(V<sub>rec</sub>, V<sub>cong</sub>)**

### ¿Cómo se mide la V\_cong?

- Tanto **V\_rec** como **V\_cong** se miden en bytes
  - Longitud de la mayoría de segmentos = MSS bytes
    - Por tanto:
      - Vamos a medir la V\_cong en base a MSS
      - La V\_cong va a evolucionar en saltos de MSS bytes
  - La relación es:  
 $\text{número de bytes/MSS} = \text{número de segmentos}$

### 3. Cómo variar la tasa de transmisión en función de la congestión percibida

- Principios de funcionamiento
  - **Indicio negativo:** pérdida de segmento
    - Reducir la tasa de envío
  - **Indicio positivo:** recepción de un ACK que reconoce datos no reconocidos previamente
    - Incrementar la tasa de envío

### Gestión de la ventana de congestión

- Tres algoritmos distintos en el emisor para intentar evitar la congestión de la red:
  - **Arranque lento (Slow Start):**
    - Cómo iniciar el flujo de datos a través de una conexión
  - **Evitación de la congestión (Congestion Avoidance)**
    - Aumento paulatino en caso de ventanas “grandes”
  - **Reducción multiplicativa (Multiplicative Decrease)**
    - Cómo actuar cuando se pierde un paquete
    - Pérdidas por errores de transmisión < 1%
- En la práctica se implementan juntos

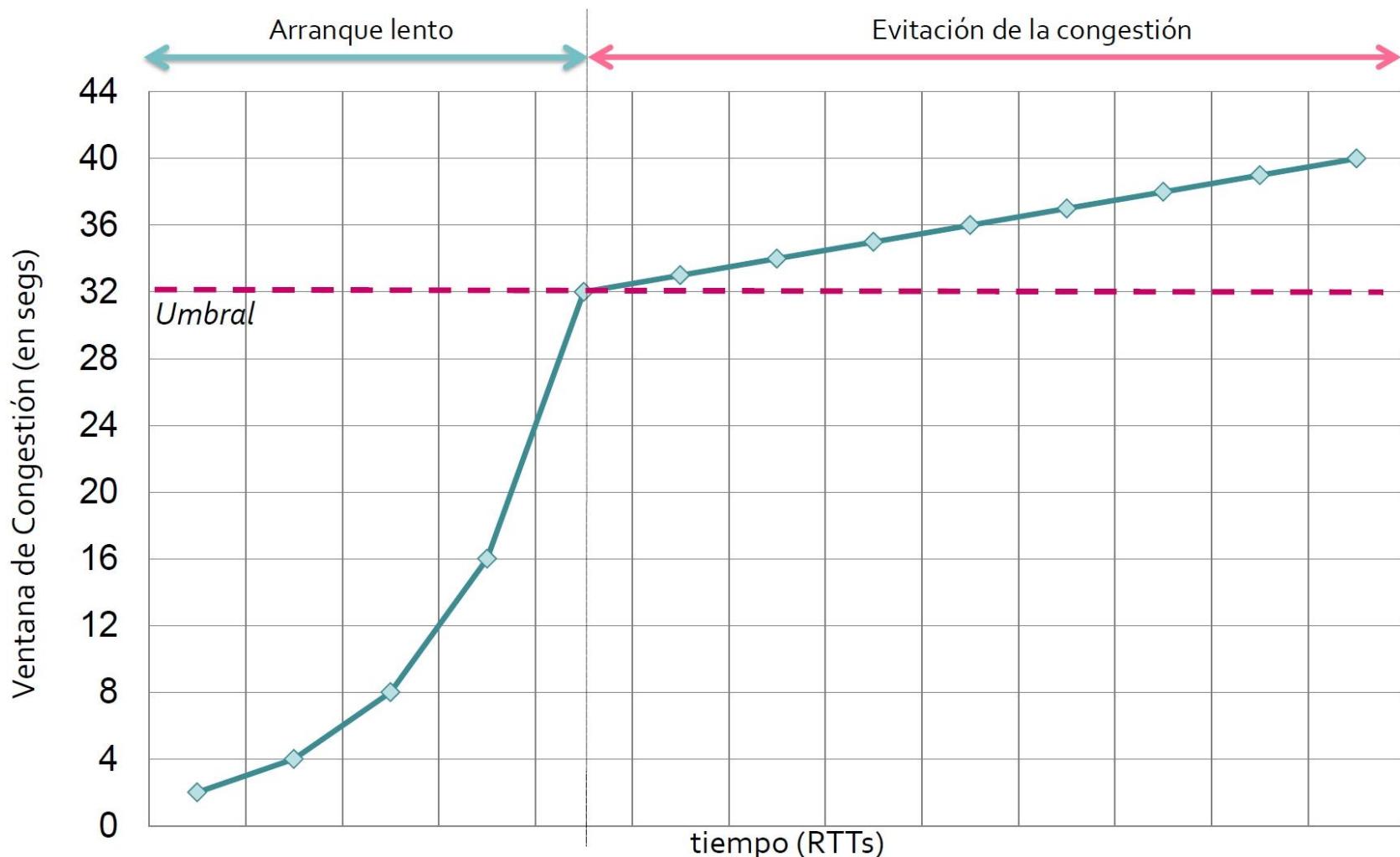
## Arranque lento

- Al comenzar la conexión se podría transmitir todos los segmentos de **V\_rec**
- Sin embargo se comienza “lentamente”:
  - Inicialmente, **V\_cong = 2 MSS**
- Crecimiento exponencial, por cada reconocimiento recibido (que reconozca nuevos datos):
  - **V\_cong = V\_cong + 1 MSS**

### Evitación de la congestión

- Un límite adicional, **Umbral**
  - Inicialmente puede ser igual a **V\_rec**
- **A partir de Umbral el crecimiento es lineal**
  - Un MSS cuando se reciben todos los ACKs de la ventana de congestión
  - Muchas implementaciones realizan incrementos parciales, por cada ACK recibido
- El crecimiento se mantiene hasta que se detectan problemas:
  - Vence un temporizador
  - Se reciben 3 ACKs duplicados
- Entonces, se aplica **reducción multiplicativa**

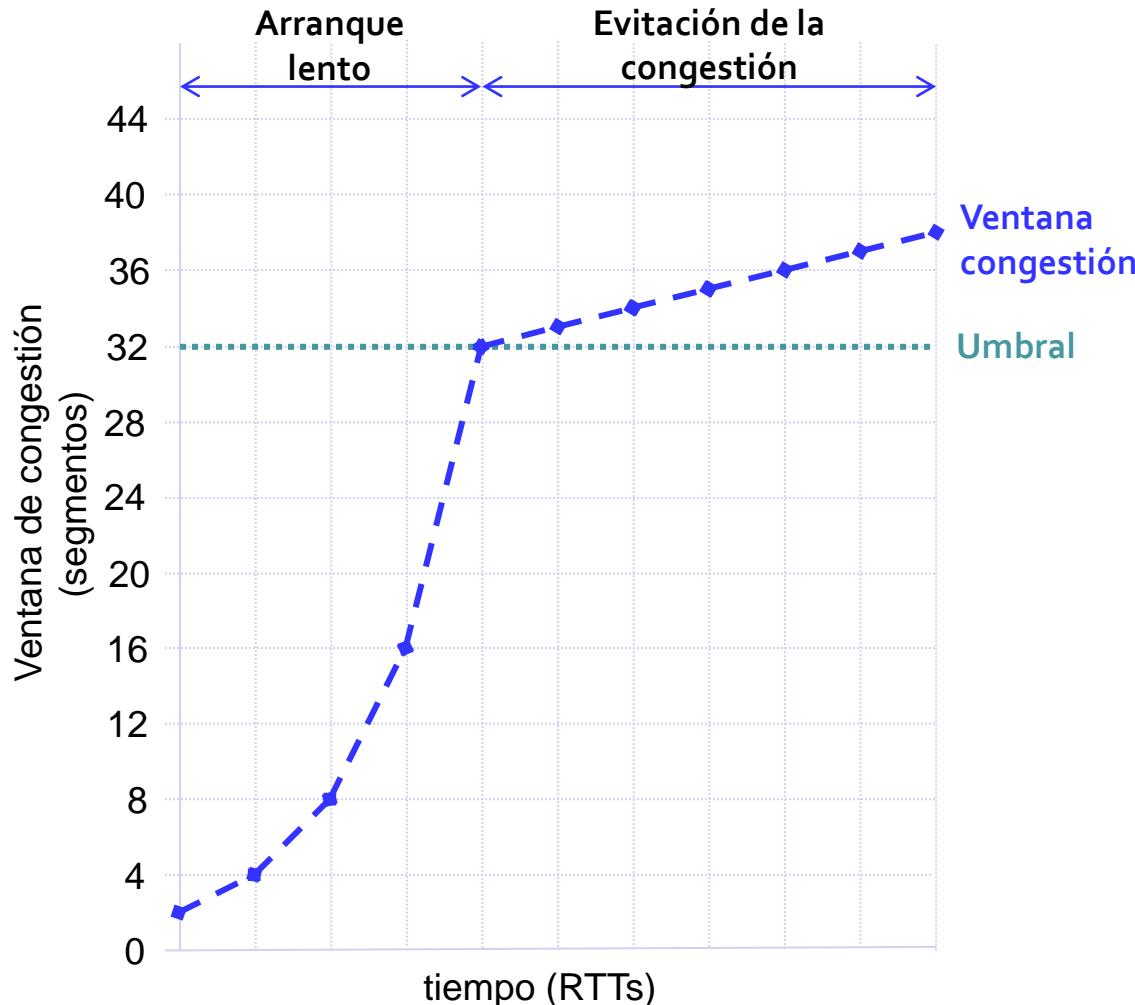
### Evitación de la congestión: ejemplo



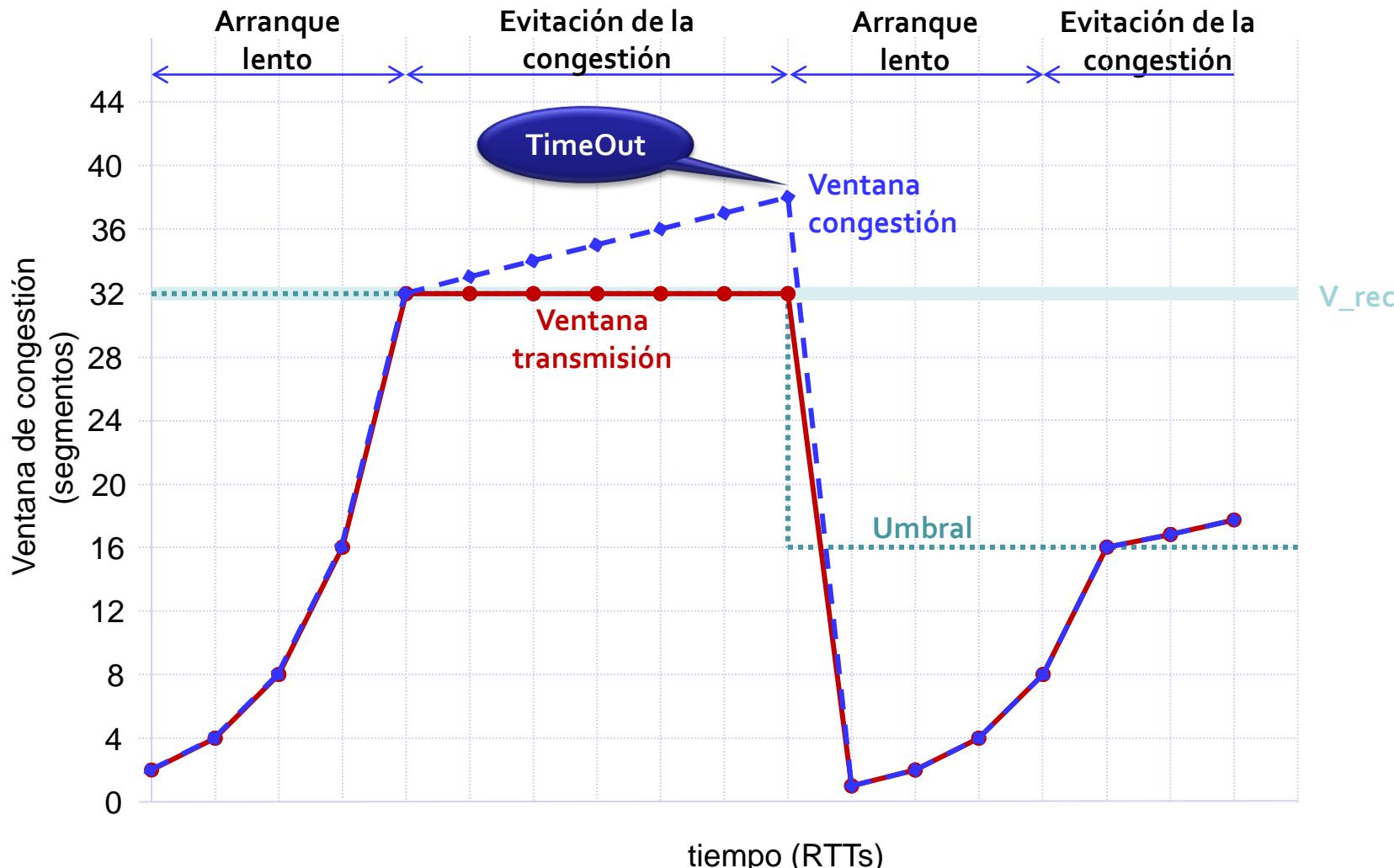
### Reducción multiplicativa

- Si congestión (pérdida ó 3 ACK's duplicados):
  - **Umbral = max(ventana\_tx/2, 2 segmentos)**
  - Si vence temporizador (pérdida)
    - **V\_cong = 1 segmento**
  - Si 3 ACKs duplicados
    - **V\_cong = Umbral**

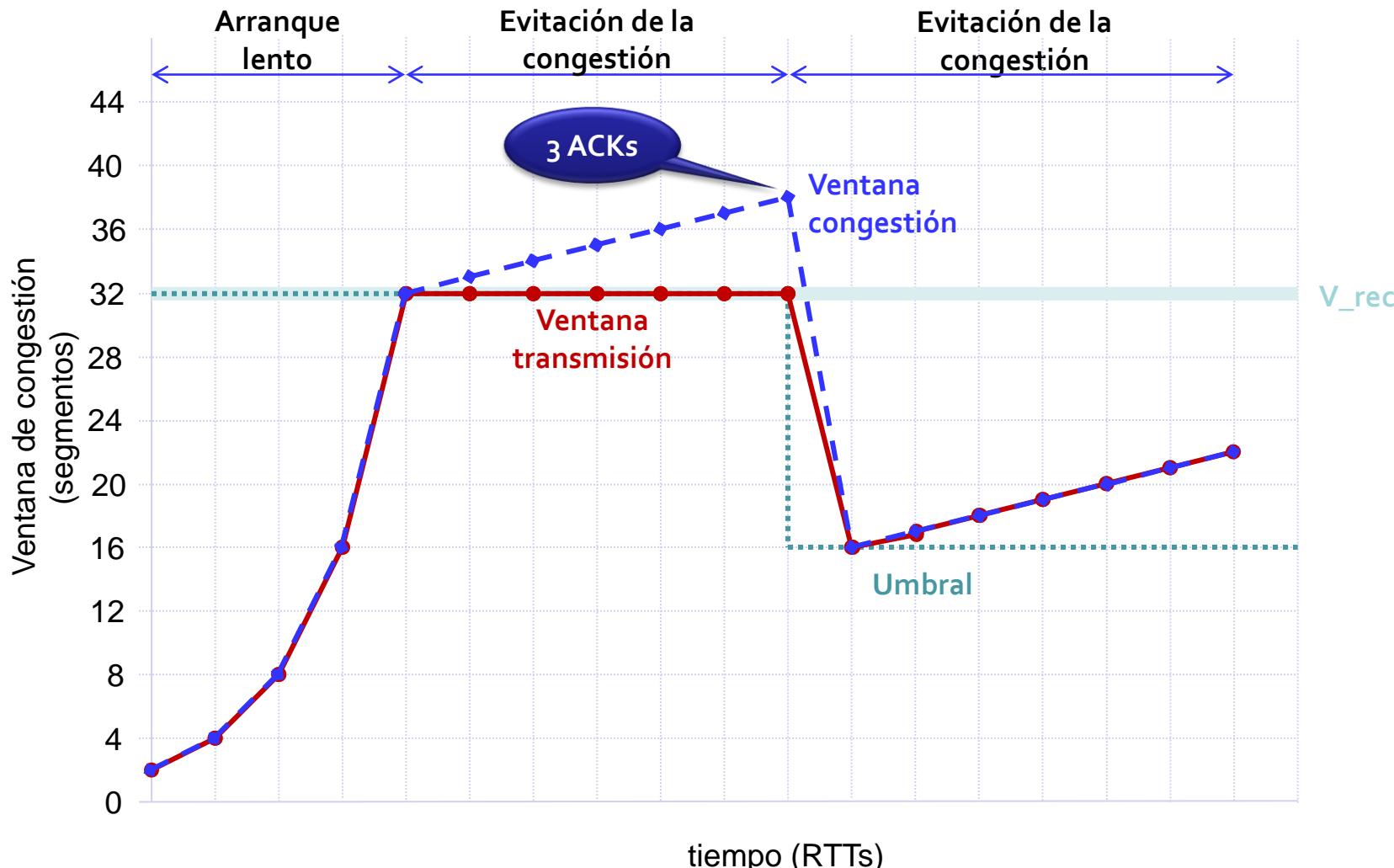
## Evitación de congestión: ejemplo



## Reducción Multiplicativa (TimeOut)



## Reducción Multiplicativa (3 ACKs)

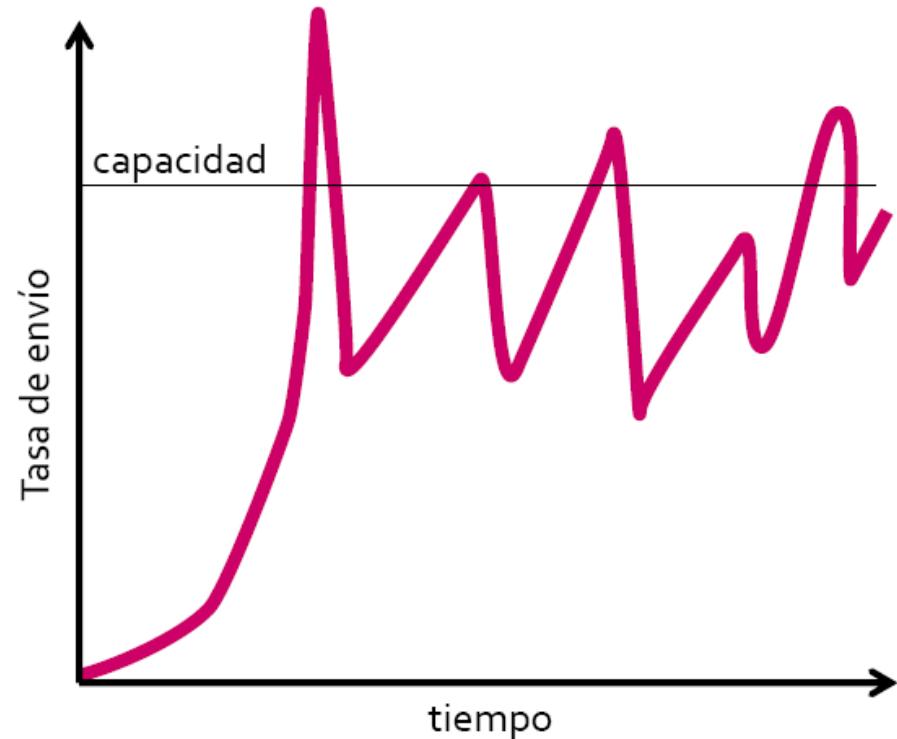


## Control de la congestión: resumen

- Cada conexión está limitada por:
  - El receptor: **V\_rec**
  - La estimación del emisor: **V\_cong**
- La capacidad de una conexión se estima por:
  - Inicio progresivo: arranque lento y evitación de la congestión
  - Reducción al producirse una retransmisión:
    - Si vence el temporizador, se reduce el **Umbral** y la **V\_cong** y se aplica **arranque lento**
    - Si llegan reconocimientos duplicados, se reduce el **Umbral** y la **V\_cong** y se aplica **evitación de la congestión**

### Control de la congestión: conclusión

- La capacidad de una conexión es desconocida
- TCP descubre esa capacidad mediante varios mecanismos:
  - Arranque lento
  - Evitación de la congestión
- TCP intenta conseguir el mejor rendimiento



## 4. Transporte orientado a la conexión: TCP

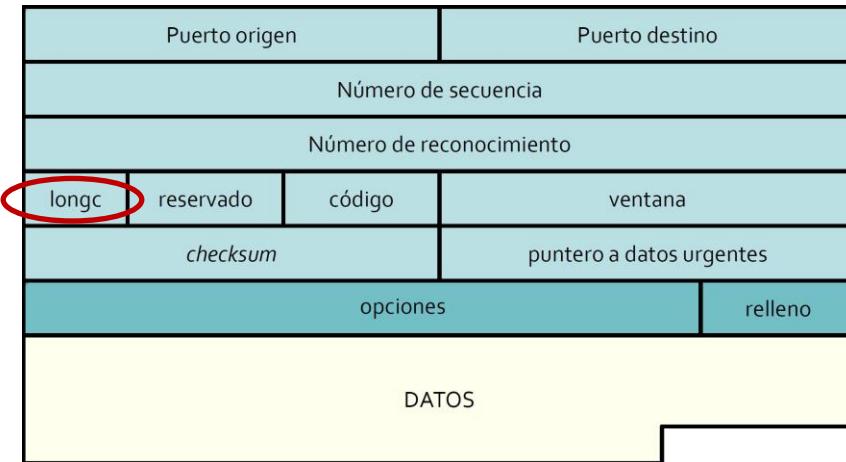
- 1) Concepto
- 2) Formato de un segmento
- 3) Control de flujo
- 4) Gestión de una conexión TCP
- 5) Control de error
- 6) Control de la congestión en TCP
- 7) **Opciones TCP**

### Conceptos:

- Esquema general del campo de opciones
- Tamaño máximo de un segmento
- Reconocimientos selectivos
- Escala de ventana

## Opciones TCP

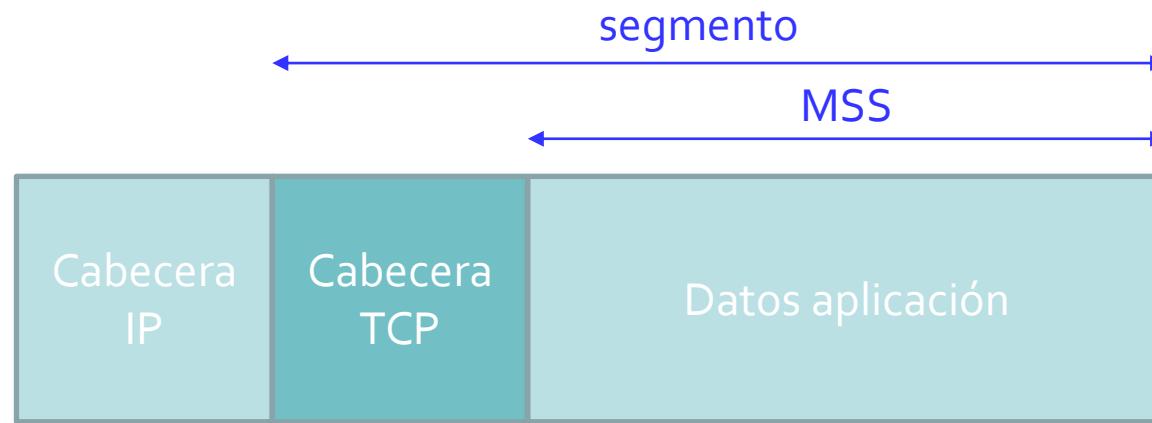
- Principales opciones TCP:
  - Tamaño máximo de segmento (MSS)
  - Escala de ventana
  - Reconocimientos selectivos (SACK)



- Campo de Opciones
  - Variable entre 0 y 320 bits (40 bytes máximo), divisible por 32
  - Longitud determinada por el campo “longitud cabecera”
  - Cada opción tiene hasta tres campos:
    - Código (1 byte)
    - Longitud (1 byte)
    - Datos (variable)

### Opciones TCP: Maximum Segment Size (MSS)

- Intenta evitar la fragmentación de IP
- Cada extremo anuncia su MSS al establecer la conexión
  - No se le pueden enviar segmentos mayores que MSS
- Por defecto MSS = 536 octetos



## Opciones TCP: escala de ventana

- La longitud máxima de ventana = 65.535 bytes
  - $ventana = 2^{16} - 1$
- El envío continuo requiere
  - $ventana(\text{en bits}) \geq RTT * V_{trans}$
- El tamaño máximo de la ventana puede ser una limitación importante en redes de alta velocidad:
  - Si  $RTT = 1 \text{ ms}$  y  $V_{trans} = 1 \text{ Gbps}$ 
    - $RTT * V_{trans} = 10^6 \text{ bits} = 125.000 \text{ bytes}$
- Con esta opción se especifica un  $k \rightarrow ventana = 2^k * ventana$ 
  - $0 \leq k \leq 14 \rightarrow$  Máxima ventana escalada = 1 GB ( $2^{30}$ )
  - Opción escala de ventana solo permitida en segmentos SYN

## Opciones TCP: ACKs selectivos

- Con ACKs acumulativos no se puede confirmar la recepción de segmentos fuera de orden
  - Puede provocar retransmisiones innecesarias
- Los ACKs selectivos (SACK) permiten reconocer bloques de datos no contiguos (RFC 2018)
- Opción **SACK-permitted** en segmentos SYN para activar ACKs selectivos
- Información SACKs:
  - Aparece primero el último bloque recibido (máximo 4 bloques)
  - El límite superior del bloque SACK indica el siguiente octeto de datos que se espera

