

# Redes de Computadores

Grado en Ingeniería Informática



# Contenido de la asignatura

Tema 1: Redes de Computadores e Internet

Tema 2: Capa de Aplicación

**Tema 3: Capa de Transporte**

Tema 4: Capa de Red

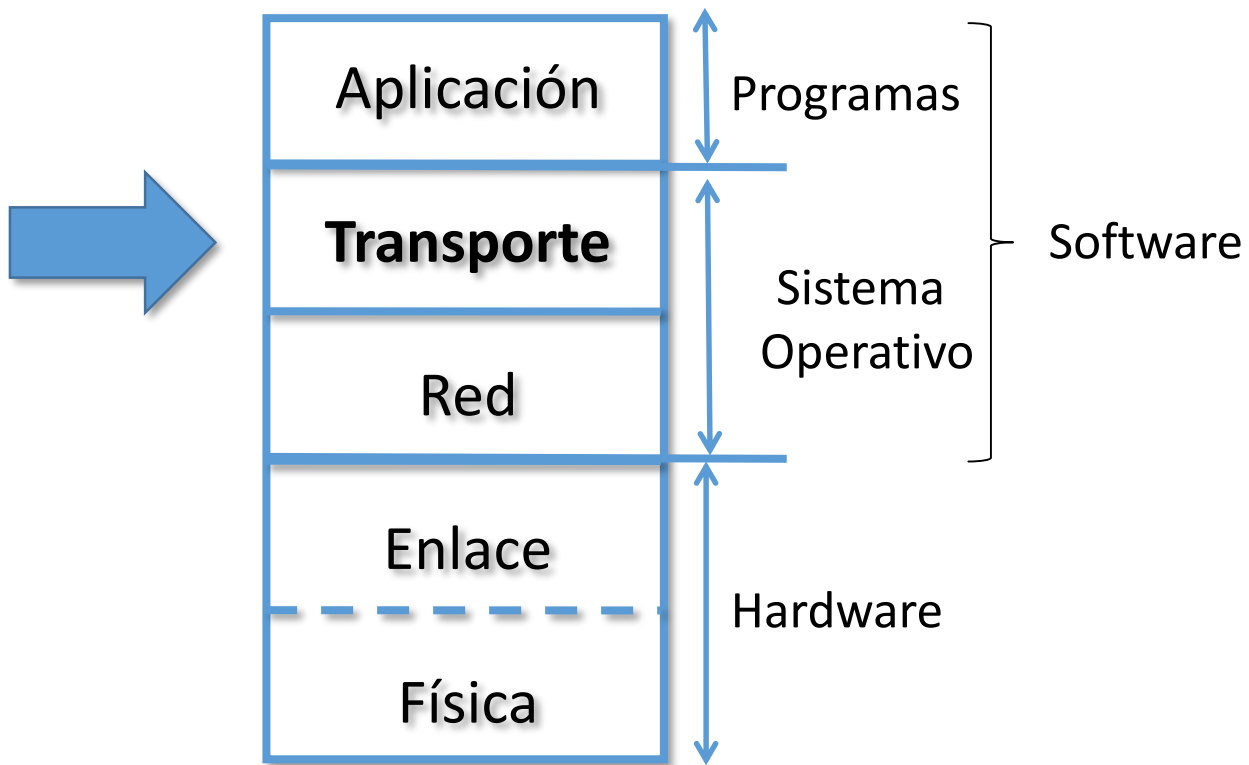
Tema 5: Capa de Enlace de Datos

# Redes de Computadores

## Tema 3

### La Capa de Transporte





# Tema 3: La Capa de Transporte

## Objetivos

- Entender los principios que hay detrás de los servicios del nivel de transporte:
  - Multiplexión/demultiplexión
  - Transferencia de datos confiable
  - Control de flujo
- Conocer los protocolos de transporte usados en Internet:
  - UDP: transporte no orientado a la conexión
  - TCP: transporte orientado a la conexión

## Contenido

1. Servicios del nivel de transporte
2. Multiplexión y demultiplexión
3. Transporte sin conexión: UDP
4. Principios de la transferencia fiable
5. Transporte orientado a la conexión: TCP
  - Estructura del segmento TCP
  - Transferencia de datos fiable
  - Control de flujo
  - Gestión de la conexión

# Tema 3: La Capa de Transporte

## Objetivos

- Entender los principios que hay detrás de los servicios del nivel de transporte:
  - Multiplexión/demultiplexión
  - Transferencia de datos confiable
  - Control de flujo
- Conocer los protocolos de transporte usados en Internet:
  - UDP: transporte no orientado a la conexión
  - TCP: transporte orientado a la conexión

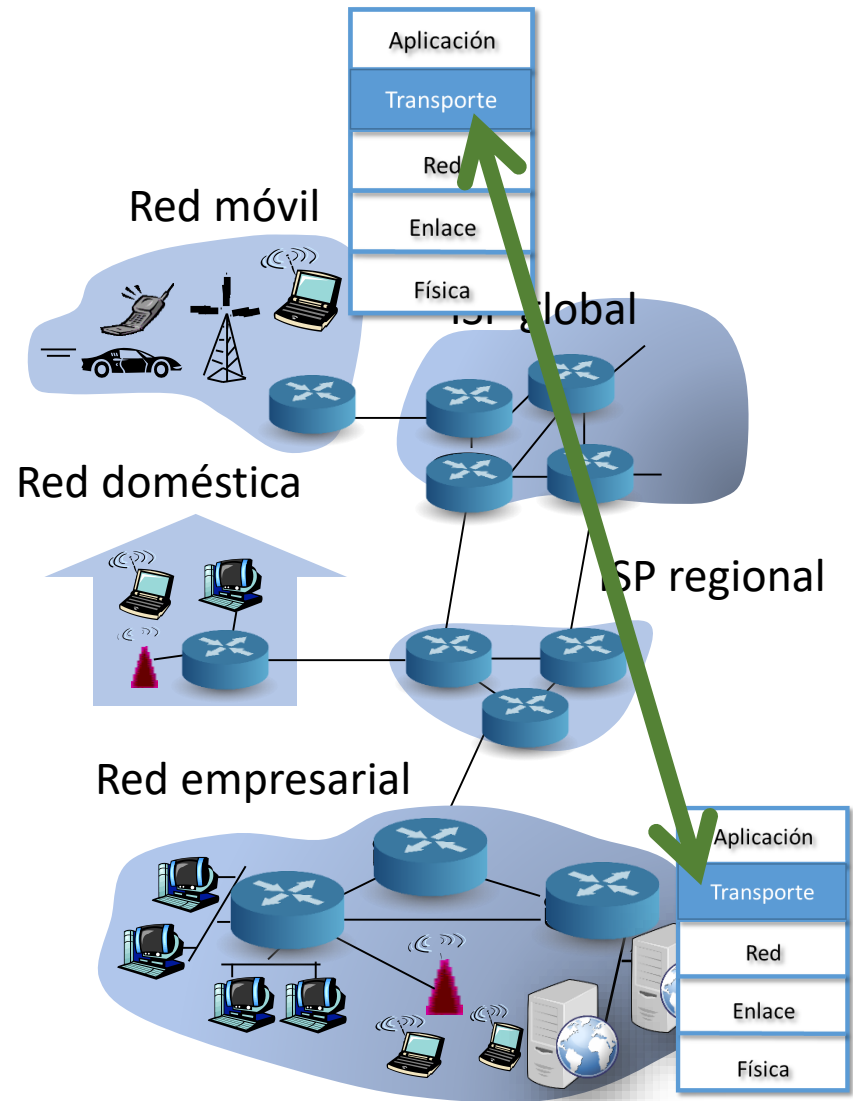
## Contenido

1. **Servicios del nivel de transporte**
2. Multiplexión y demultiplexión
3. Transporte sin conexión: UDP
4. Principios de la transferencia fiable
5. Transporte orientado a la conexión: TCP
  - Estructura del segmento TCP
  - Transferencia de datos fiable
  - Control de flujo
  - Gestión de la conexión

# Servicios del nivel de transporte

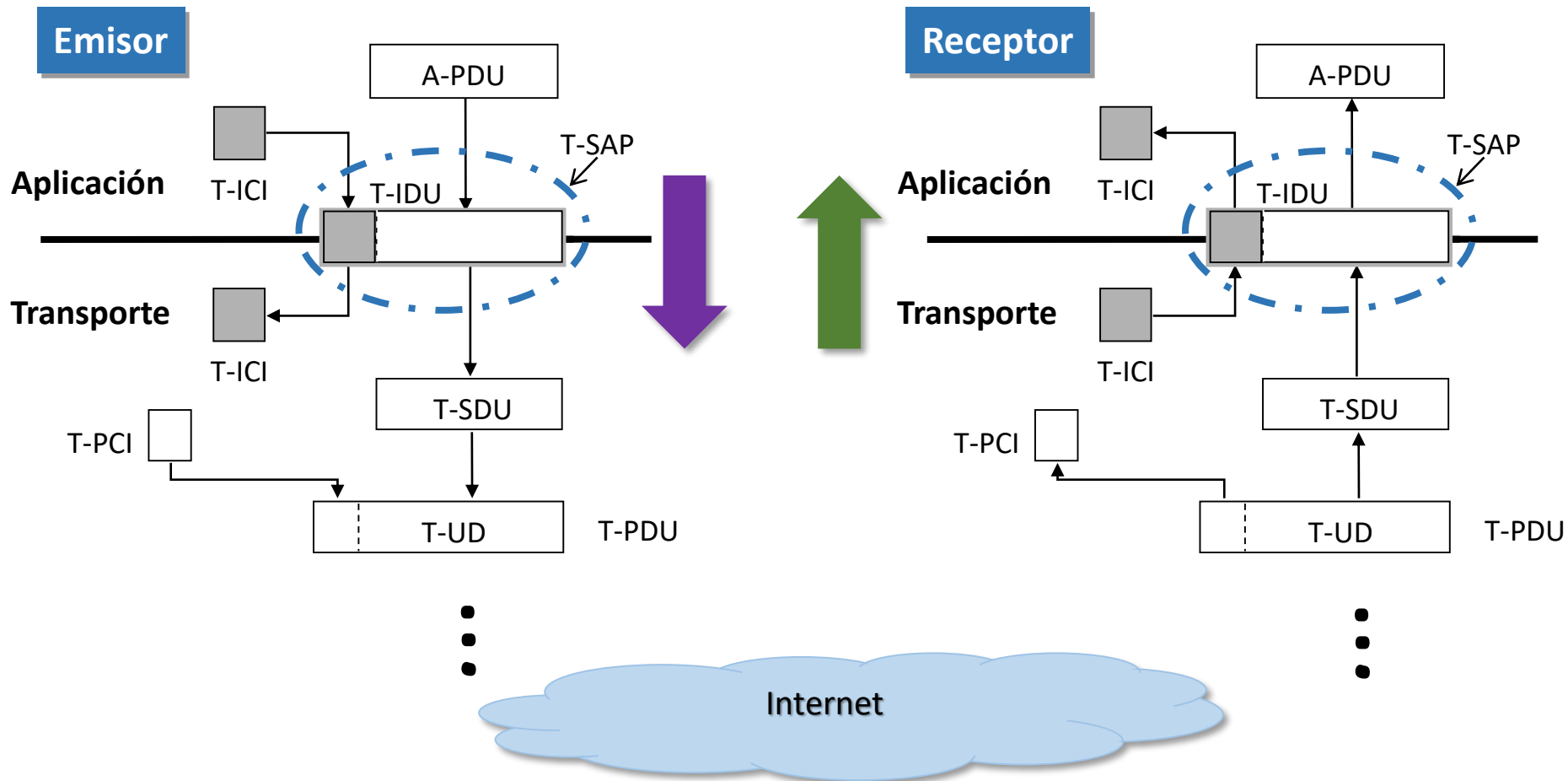
## Comunicación lógica

- Proporciona **comunicación lógica** entre aplicaciones que se ejecutan en hosts diferentes, ocultándoles la complejidad de la red que une a ambos hosts.
- Los protocolos de transporte se ejecutan en los sistemas terminales (hosts), no en los routers.
- Las aplicaciones pueden escoger el protocolo de transporte que quieren usar, en función del tipo de servicio que necesiten
  - En Internet: TCP y UDP



# Servicios de nivel de Transporte

## Sockets (o T-SAP del Tema 2)

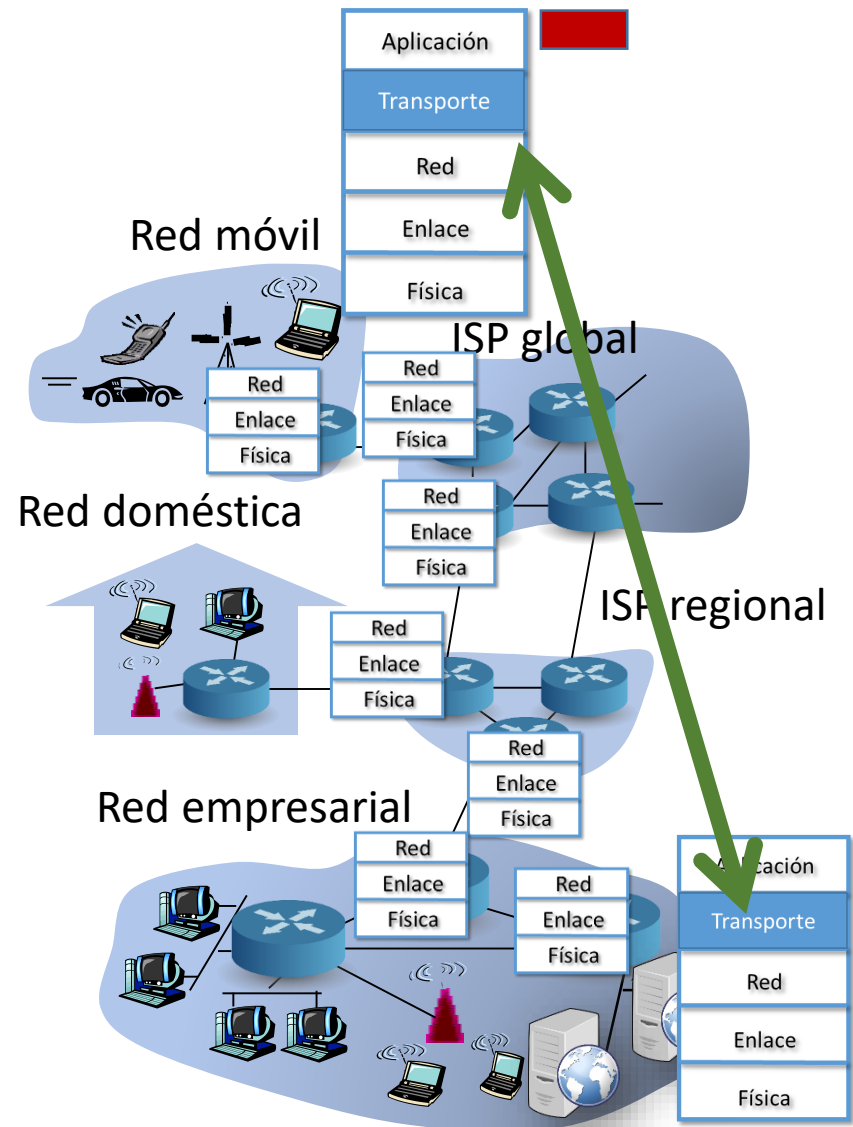




# Servicios de nivel de Transporte

## Protocolos de Transporte en Internet

- **TCP**: Servicio de entrega de datos fiable y en orden
  - Control de flujo
  - Control de la congestión
  - Establecimiento de la conexión
- **UDP**: Servicio de entrega de datos no fiable y sin garantía de orden.
  - Protocolo simple “sin florituras”
- Servicios **no** disponibles:
  - Retardo garantizado
  - Ancho de banda garantizado
- Tanto TCP como UDP usan los servicios de IP
  - Protocolo que suministra un servicio de mejor esfuerzo (“best-effort”).



# Servicios de nivel de Transporte

## Transporte vs. Red

- **Capa de red:** proporciona un servicio de comunicación lógica entre equipos terminales (hosts)
  - Permite (gracias a las direcciones IP) identificar un sistema final en Internet.
  - Es un servicio similar al servicio de correo postal que permite enviar una carta a una casa (domicilio).
- **Capa de transporte:** extiende el servicio de la capa de red para proporcionar un servicio de comunicación lógica entre los procesos de aplicación.
  - Permite que procesos distintos en un mismo sistema final usen el mismo nivel de red gracias a los puertos (esto se conoce como multiplexión y demultiplexión de la capa de transporte).
  - Se consigue usando el servicio prestado por la capa de red, para, a partir de él, construir un servicio “mejorado”.

# Tema 3: La Capa de Transporte

## Objetivos

- Entender los principios que hay detrás de los servicios del nivel de transporte:
  - Multiplexión/demultiplexión
  - Transferencia de datos confiable
  - Control de flujo
- Conocer los protocolos de transporte usados en Internet:
  - UDP: transporte no orientado a la conexión
  - TCP: transporte orientado a la conexión

## Contenido

1. Servicios del nivel de transporte
- 2. Multiplexión y demultiplexión**
3. Transporte sin conexión: UDP
4. Principios de la transferencia fiable
5. Transporte orientado a la conexión: TCP
  - Estructura del segmento TCP
  - Transferencia de datos fiable
  - Control de flujo
  - Gestión de la conexión


# Multiplexión y Demultiplexión

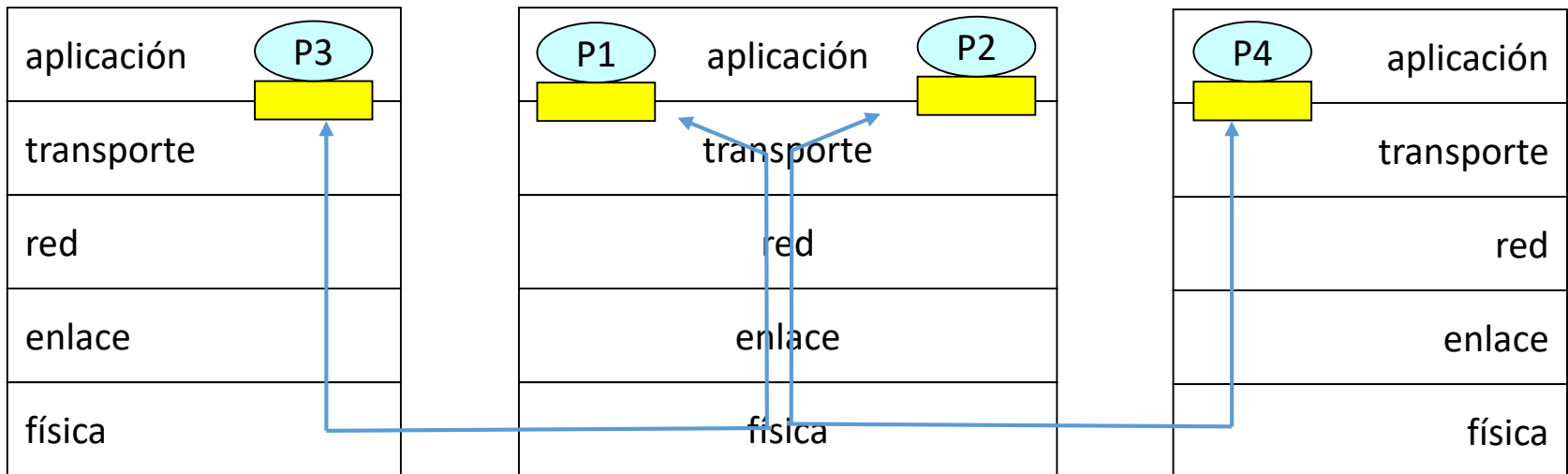
## Multiplexión al enviar

Recolectar datos de múltiples sockets, crear los segmentos (T\_PDU) añadiendo información de cabecera (T\_PCI) que se usará luego al demultiplexar.

## Demultiplexión al recibir

Entregar en el socket correcto el contenido (T\_UD) de los segmentos (T\_PDU) recibidos, gracias a la información de la cabecera (T\_PCI).

 = socket       = proceso



Sistema final 1

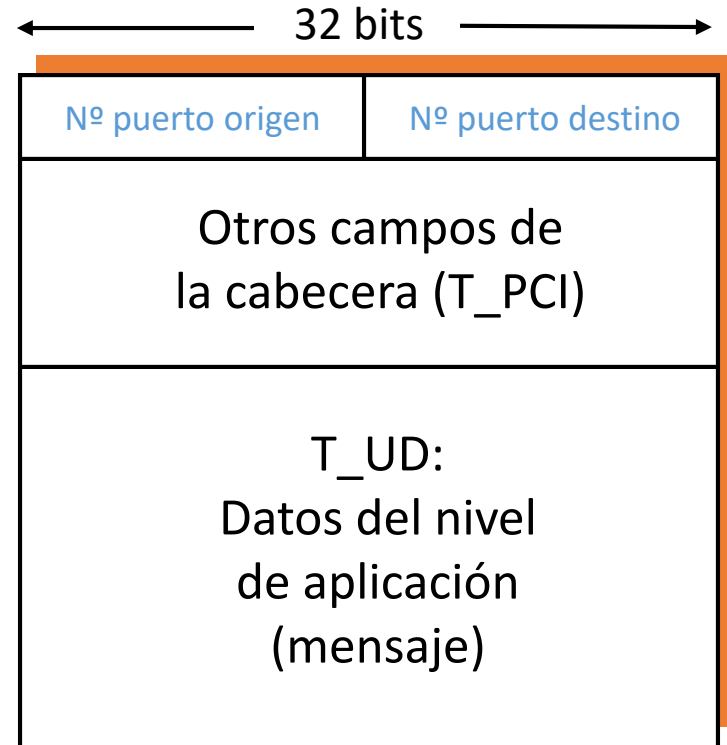
Sistema final 2

Sistema final 3

# Multiplexión y Demultiplexión

## Funcionamiento

- El nivel de red recibe datagramas IP (R\_PDU)
  - Cada R\_PDU tiene una **dirección IP origen** y una **dirección IP destino** en su cabecera (R\_PCI).
  - Cada R\_PDU encapsula en su interior un segmento (T\_PDU)<sup>1</sup>.
- El nivel de transporte recibe segmentos (T\_PDU)
  - Cada T\_PDU tiene en su T\_PCI un **nº de puerto origen** y un **nº de puerto destino**.
  - Cada T\_PDU encapsula datos del usuario, el nivel de aplicación (T\_UD).
- **En el host destino las direcciones IP y los números de puerto sirven para entregar la T\_UD de la T\_PDU al socket adecuado.**

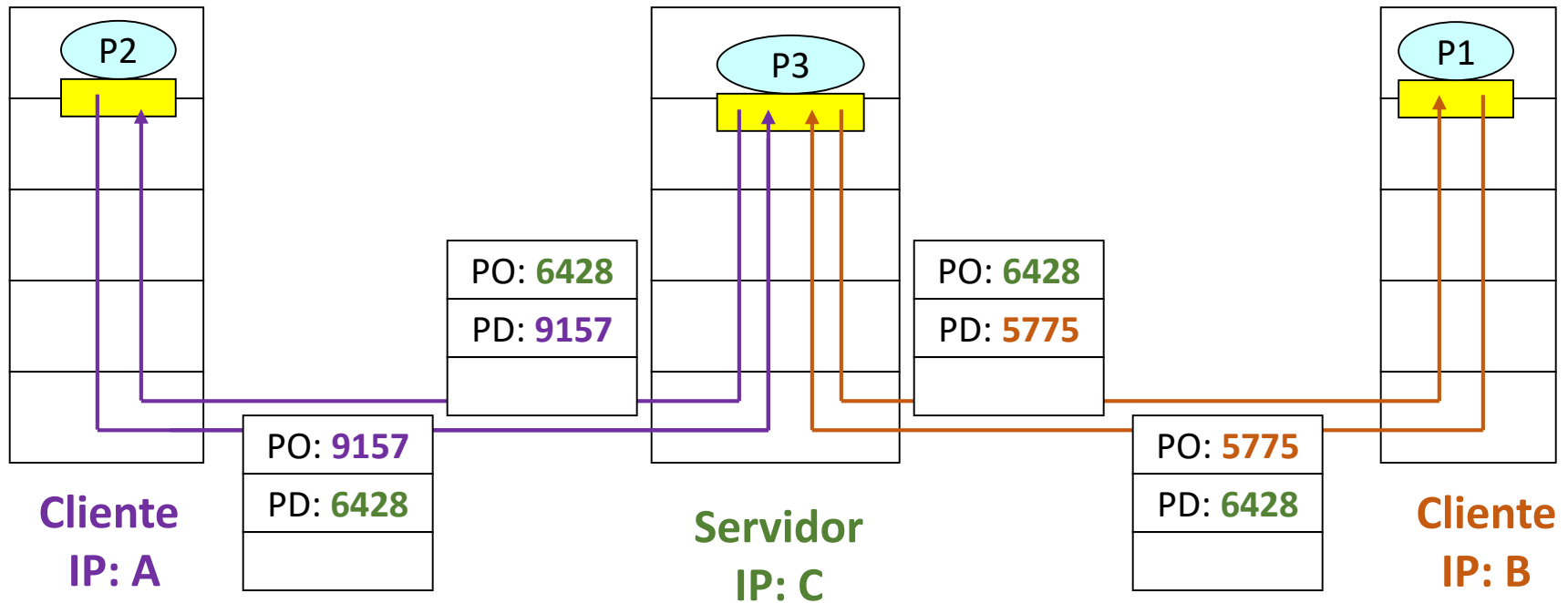


Formato de la T\_PDU  
(común a TCP y UDP)

<sup>1</sup> Ciertamente si no hay fragmentación

# Multiplexión y Demultiplexión

## Sin conexión (UDP)



La IP Origen y el Puerto Origen permitirán al proceso P3 identificar al proceso origen (P1 o P2) y devolverle un mensaje.

PO = Nº de Puerto Origen  
PD = Nº de Puerto Destino

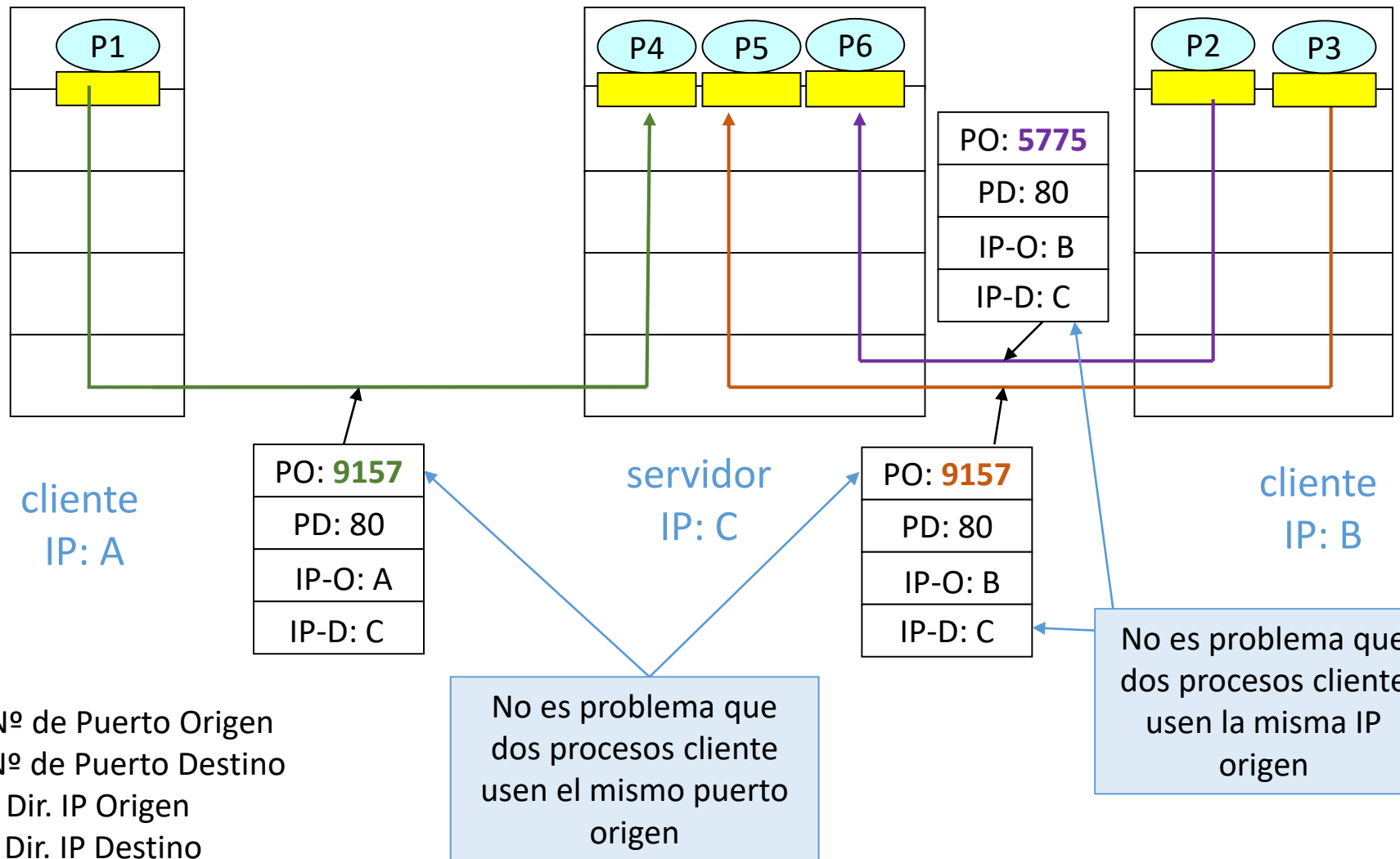
# Multiplexión y Demultiplexión

## Orientada a la conexión (TCP)

- Una conexión TCP se identifica por una 4-tupla:
  - Dir. IP local
  - N° Puerto local
  - Dir. IP remota
  - N° Puerto remoto
- En el host destino se usan los 4 valores, (presentes en la R\_PCI y T\_PCI) para hacer llegar los datos de usuario de la T\_PDU al socket TCP adecuado.
- Una aplicación servidora puede tener varias conexiones TCP funcionando simultáneamente.
  - Cada conexión se identifica por su propia 4-tupla
- Un servidor web tiene una conexión TCP distinta para cada cliente que se conecta.
  - Con HTTP no persistente, cada petición del mismo cliente irá por una conexión TCP diferente.

# Multiplexión y Demultiplexión

## Servidor Web con varios procesos





# Tema 3: La Capa de Transporte

## Objetivos

- Entender los principios que hay detrás de los servicios del nivel de transporte:
  - Multiplexión/demultiplexión
  - Transferencia de datos confiable
  - Control de flujo
- Conocer los protocolos de transporte usados en Internet:
  - UDP: transporte no orientado a la conexión
  - TCP: transporte orientado a la conexión

## Contenido

1. Servicios del nivel de transporte
2. Multiplexión y demultiplexión
- 3. Transporte sin conexión: UDP**
4. Principios de la transferencia fiable
5. Transporte orientado a la conexión: TCP
  - Estructura del segmento TCP
  - Transferencia de datos fiable
  - Control de flujo
  - Gestión de la conexión

# Transporte sin conexión: UDP

## UDP = User Datagram Protocol [RFC 768]

- Es un protocolo de transporte de Internet muy simple, sin “florituras”.
- Ofrece un **servicio de mejor esfuerzo**, “best effort”, por lo que:
  - Las T\_PDU pueden “perderse” y no llegar a su destino.
  - Si las T\_PDU llegan desordenadas, los datos de usuario contenidos en ellas se entregarían desordenados al Nivel de Aplicación.
- **Sin conexión:**
  - No hay una fase acuerdo previa entre el emisor y el receptor UDP.
  - Cada T\_PDU se trata de forma independiente a las demás.

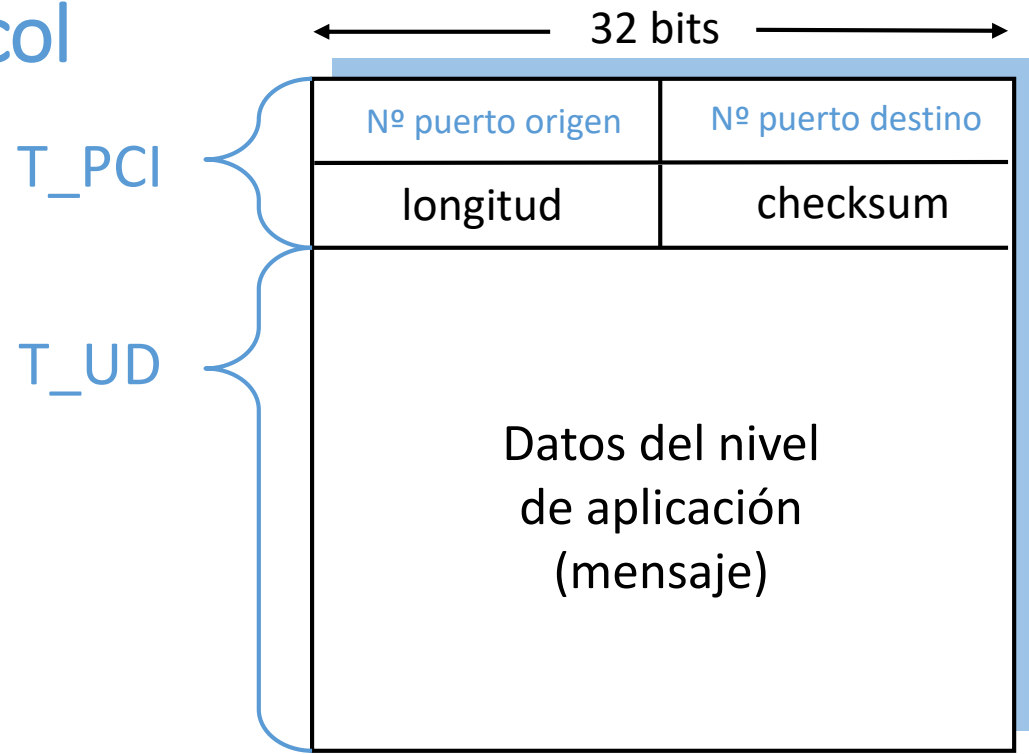
### ¿Por qué existe UDP?

- No hay un establecimiento de la conexión (que añadiría un retardo).
- Es simple de implementar: no hay que mantener el estado de la conexión ni en el transmisor ni en el receptor.
- La cabecera (T\_PCI) es más simple y pequeña.

# Transporte sin conexión: UDP

## User Datagram Protocol

- A menudo usado por aplicaciones de streaming multimedia
  - Tolerantes a pérdidas
  - Sensibles al ancho de banda
- Otros usos de UDP
  - DNS
  - SNMP
- ¿Transferencia fiable sobre UDP? Es posible si añadimos fiabilidad a la capa de aplicación
  - ¡Recuperación de errores específica de la aplicación!



### Formato de la T\_PDU de UDP

La cabecera (T\_PCI) solo tiene 4 campos. La **longitud** es en bytes y es la de la T\_PDU completa, con cabecera.

# Transporte sin conexión: UDP

## Checksum (suma de comprobación)

**Objetivo:** detectar “errores” (e.g., bits “cambiados”) en una T\_PDU transmitida.

### El emisor:

- Trata la T\_PDU como una secuencia de enteros de 16 bits.
- Simplificando un poco, podemos decir que suma todos los enteros de 16 bits que componen la T\_PDU y luego le calcula el complemento a 1.
- Coloca el valor calculado en el campo checksum de la cabecera (T\_PCI).

### El receptor:

- Calcula el checksum, otra vez, de la misma forma que lo hizo el emisor, sobre la T\_PDU recibida.
- Comprueba si el checksum calculado **es idéntico** al valor del campo checksum de la T\_PDU recibida.
  - NO → ¡Error detectado!
  - Sí → No se detecta error alguno, pero... ¿Puede que haya un error?

# Tema 3: La Capa de Transporte

## Objetivos

- Entender los principios que hay detrás de los servicios del nivel de transporte:
  - Multiplexión/demultiplexión
  - Transferencia de datos confiable
  - Control de flujo
- Conocer los protocolos de transporte usados en Internet:
  - UDP: transporte no orientado a la conexión
  - TCP: transporte orientado a la conexión

## Contenido

1. Servicios del nivel de transporte
2. Multiplexión y demultiplexión
3. Transporte sin conexión: UDP
4. **Principios de la transferencia fiable**
5. Transporte orientado a la conexión: TCP
  - Estructura del segmento TCP
  - Transferencia de datos fiable
  - Control de flujo
  - Gestión de la conexión

# Principios de la transferencia fiable

La T.F. es importante en capas de aplicación, transporte y enlace de datos.  
¡Está en la lista de los 10 temas más importantes sobre redes!

## ¿Por qué es necesaria la T.F.?

- **PDU's erróneas**

En las transmisiones sobre los enlaces se producen interferencias que alteran los bits transmitidos.

- **PDU's perdidas**

Las colas de paquetes, cuando están saturadas, empiezan a descartar los paquetes entrantes.

- **PDU's duplicadas**

Determinados problemas en la comunicación provocan que se retransmitan PDU's que ya han sido recibidas.

## ¿Cómo se detectan estos problemas?

- **PDU's erróneas**

Usando mecanismos de comprobación de errores (incluidos en la PCI).

- Algoritmos similares al del checksum.
- Los algoritmos más complejos y fiables se utilizan en el **nivel de enlace**.

- **PDU's perdidas y duplicadas**

Añadiendo "algo" a la cabecera (PCI) de cada PDU que permita distinguirla del resto de PDU's enviadas.

## ¿Cómo se solucionan estos problemas?

### Retransmisiones

El transmisor vuelve a transmitir una copia exacta de la PDU que tuvo problemas.

# Principios de la transferencia fiable

## Comunicación entre entidades pares

### Caso general de comunicación entre entidades pares de un mismo nivel:

- Al comunicarse con su entidad par, una entidad le envía a la otra una PDU formada por cabecera (PCI) y, en general, datos de usuario (UD).
- **Las cabeceras (PCI) contienen información de control del protocolo.**
- En general, ambas entidades transmiten y reciben datos de usuario.
- **Transferencia bidireccional de datos de usuario entre entidades pares.**

### Simplificación del caso general de comunicación entre entidades pares:

- Facilita la explicación de los principios de transferencia fiable.
- **Supondremos una transferencia unidireccional de datos de usuario.**
- A una de las entidades pares del nivel la llamaremos **transmisor (Tx)**.
- A la otra entidad par la llamaremos **receptor (Rx)**.
- El Tx transmite PDUs con PCI y UD (los UD vienen de su nivel superior).
- El Rx recibe PDUs con PCI y UD (los UD los pasará a su nivel superior).
- El Rx transmite PDUs que solo tendrán PCI (sin UD, solo info. de control).
- El Tx recibe PDUs que solo tendrán PCI.
- **Transferencia bidireccional de información de control del protocolo.**

# Principios de la transferencia fiable

## Tipos de PDUs

- **PDU de datos**

- Solo las envía el Tx
- Contiene datos de usuario (UD)
- Contiene información de control del protocolo (en la PCI)

- **PDU de control**

- Solo las envía el Rx
- **No** contiene datos de usuario (UD)
- Solo contiene información de control del protocolo (en la PCI)

**Nota**

Es un error pensar que el Tx no envía información de control porque solo envía PDUs de datos.  
Las PDUs de datos también llevan información de control.



# Principios de la transferencia fiable

## ¿Qué contiene la cabecera (PCI)?

- La **PCI** de una **PDU de datos** contiene información que permite:
  - Que el Rx detecte si esa PDU de datos tiene errores.
  - Identificar esa PDU de datos y distinguirla de otras PDU enviadas por el Tx.
- La **PCI** de una **PDU de control** contiene información que permite:
  - Que el Tx detecte si esa PDU de control tiene errores.
  - Que el Rx identifique una determinada PDU de datos, e informar que dicha PDU de datos
    - ha sido recibida correctamente por el Rx.
      - **ACK, acknowledgement, acuse de recibo.**
    - **no** ha sido recibida correctamente por el Rx.
      - **NAK, NACK, negative acknowledgement, acuse de recibo negativo.**

### Nota

La información de la PCI que sirve para identificar una determinada PDU de datos se llama **número de secuencia**.

# Principios de la transferencia fiable

## Funcionamiento básico

### Transmisor (Tx):

- La entidad Tx de un determinado nivel, construye una PDU de datos con UD y PCI y la transmite al Rx.
- Ahora, el Tx espera durante un tiempo, conocido como `time_out`, a recibir una PDU de control del Rx que le indique
  - con un ACK que la PDU de datos llegó bien al Rx.
    - Tx no hace nada más.
  - con un NACK, que la PDU de datos llegó con errores al Rx.
    - Tx retransmite la PDU.
  - Si expira el `time_out` antes de que llegue la PDU de control del Rx, entonces
    - Tx retransmite la PDU.

### Receptor (Rx):

- Al recibir una PDU de datos la entidad Rx de un determinado nivel:
  - Si la PDU de datos llegó correctamente, es **obligatorio** que el Rx le mande al TX una PDU de control de tipo **ACK**, avisándole.
  - Si la PDU de datos llegó con errores, es **opcional** que el Rx le mande al TX una PDU de control de tipo **NACK**, avisándole.

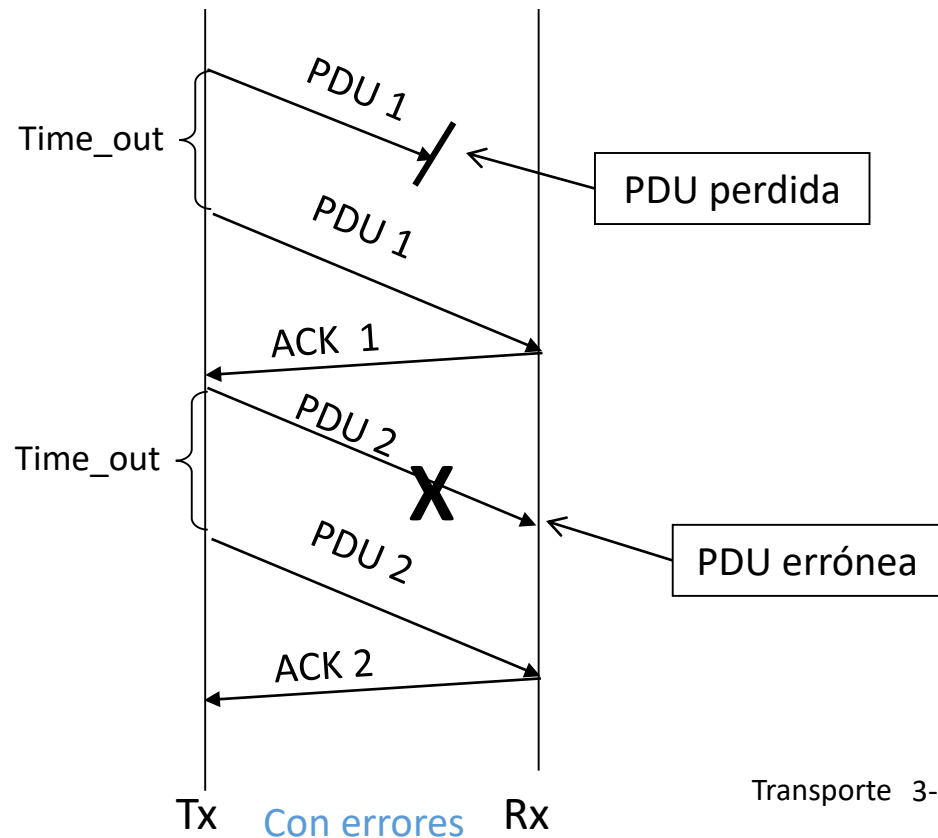
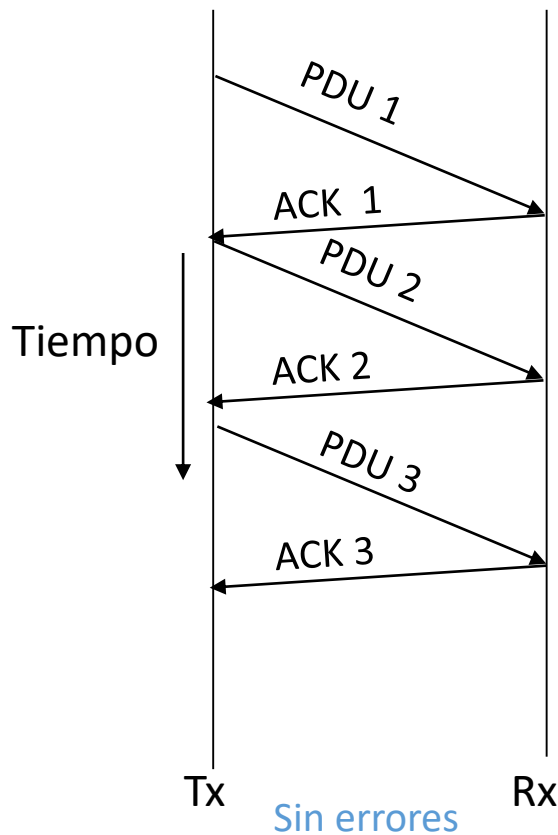
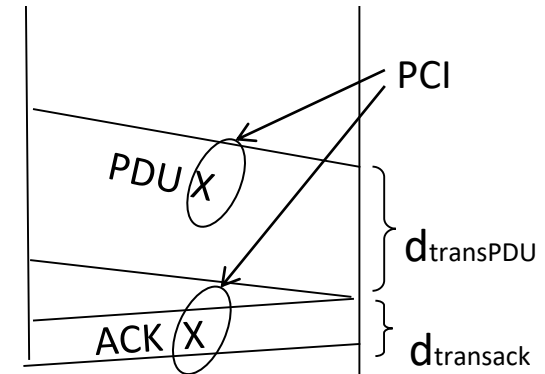
**P:** ¿Cuánto debe valer, como mínimo, el `time_out`?

**P:** ¿Entrega siempre el Rx al nivel superior los UD de una PDU de datos que llega correctamente?

# Principios de la transferencia fiable

## Ejemplo 1: pérdida y error

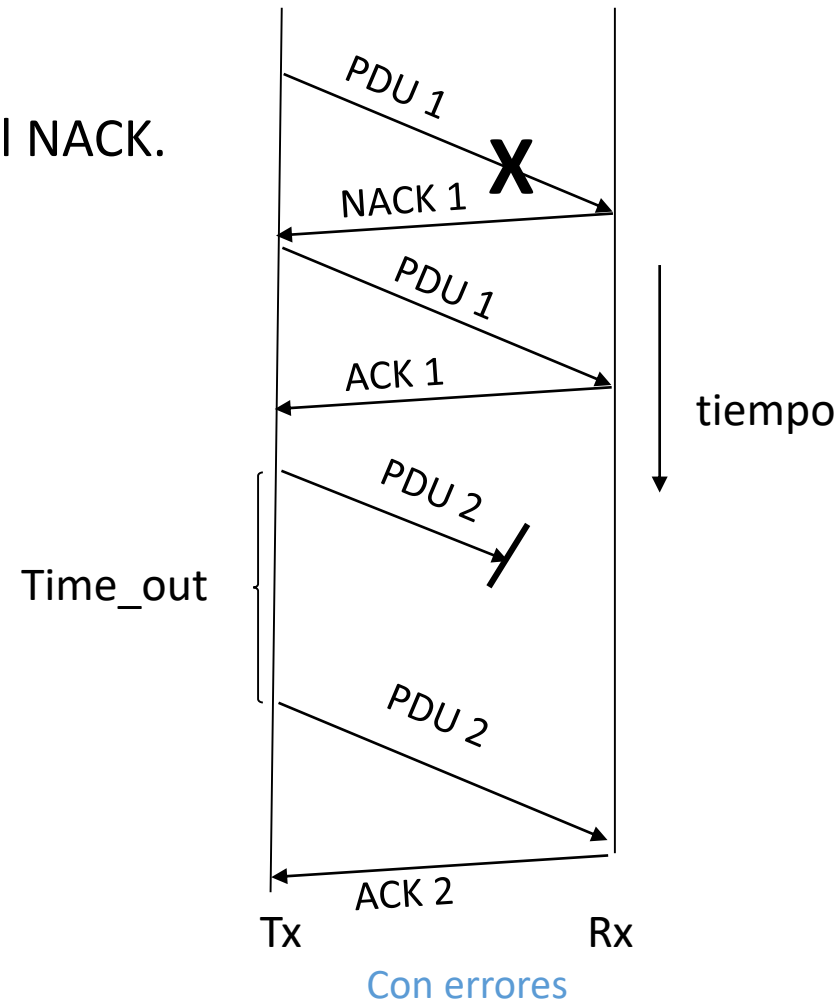
- Transmisor envía sólo una PDU con UD y no envía la siguiente hasta tener éxito en la transmisión.
- Receptor sólo envía PDU de control ACK.



# Principios de la transferencia fiable

## Ejemplo 2: Acuse de recibo negativo (NACK)

- Idem ejemplo 1.
- Receptor también envía PDU de control NACK.
- Sin errores.
  - Mismo comportamiento anterior



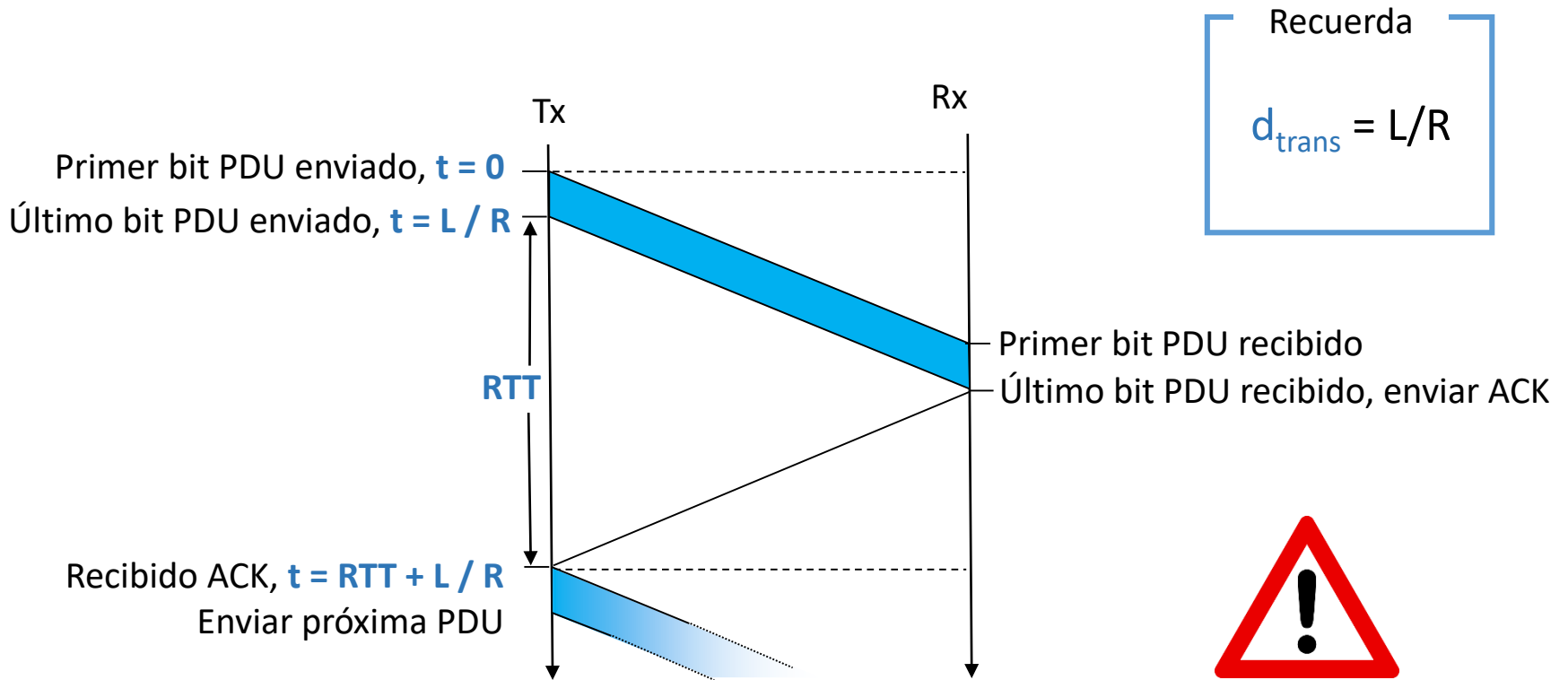
# Principios de la transferencia fiable

## Protocolo de parada y espera

- A este mecanismo de funcionamiento de un nivel para garantizar la transferencia fiable se le conoce como **Protocolo de parada y espera**
- Es poco eficiente
  - El Tx se debe parar y esperar a recibir el ACK antes de poder enviar la siguiente PDU.
- ¿Se puede medir la eficiencia?
  - Para simplificar, vamos a suponer que
    - Tx siempre tiene una PDU lista para transmitir,
    - no se producen errores,
    - cada PDU tiene L bytes de UD y 0 bytes de PCI, y
    - $d_{\text{transack}} = 0$ .
  - Utilización del transmisor (o canal)
    - $U_{\text{transmisor}}$  = fracción de tiempo que el transmisor (o canal) está ocupado transmitiendo bits de la PDU.

# Principios de la transferencia fiable

## Eficiencia del protocolo de parada y espera



Recuerda

$$d_{trans} = L/R$$



**El protocolo limita el uso de los recursos físicos.**

$$U_{transmisor} = \frac{L/R}{RTT + L/R}$$

# Principios de la transferencia fiable

## Eficiencia del protocolo de parada y espera

- El protocolo funciona, pero su eficiencia es mala
- Ejemplo: enlace de 1 Gbps, 30 ms de RTT, PDU 1KByte

$$U_{transmisor} = \frac{L / R}{RTT + L / R} = \frac{0,008}{30,008} = 0,00027$$

$$d_{trans} = \frac{L}{R} = \frac{8kb / pdu}{10^9 b / s} = 8 \mu s$$

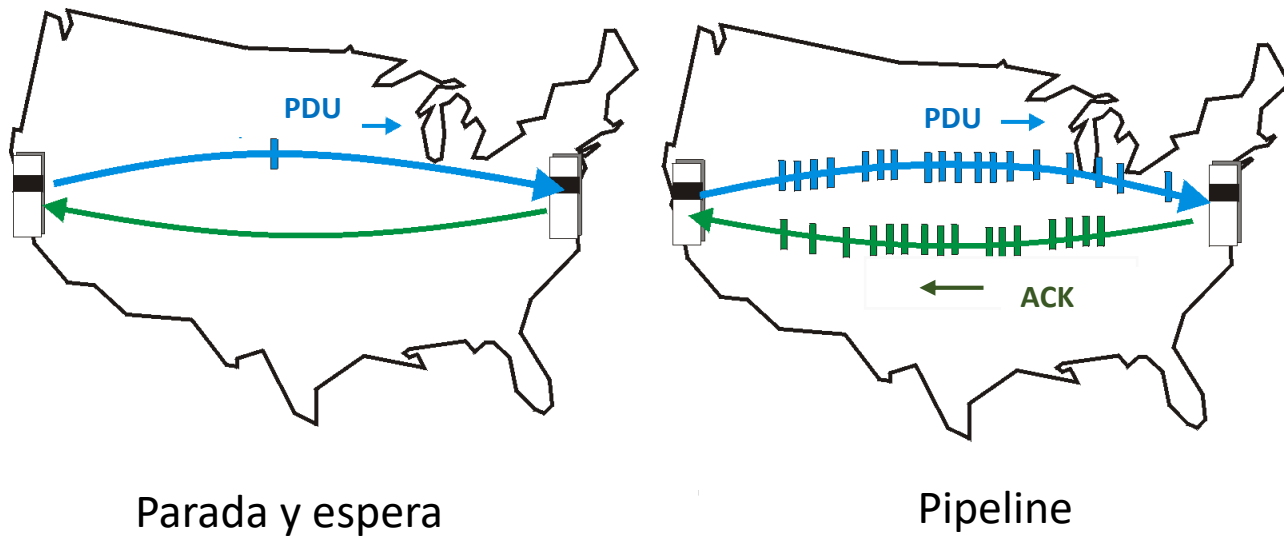
- 1 PDU de 1KByte cada 30,008 ms  $\rightarrow$  33,3kByte/s tasa de transferencia efectiva en un enlace de 1 Gbps
- ¡El Protocolo limita el uso de los recursos físicos!
- ¿Se puede mejorar?

# Principios de la transferencia fiable

## Protocolos con Pipeline - Concepto

Con pipeline, el Tx puede tener múltiples PDUs en tránsito (“en vuelo”) pendientes de confirmación (ACK), mejorando mucho la eficiencia.

- Los números de secuencia utilizados en la PCI deben tener un rango suficiente para que sirva para distinguir todas las PDUs en tránsito.
- Se requieren buffers en el Tx y, a veces, en Rx.

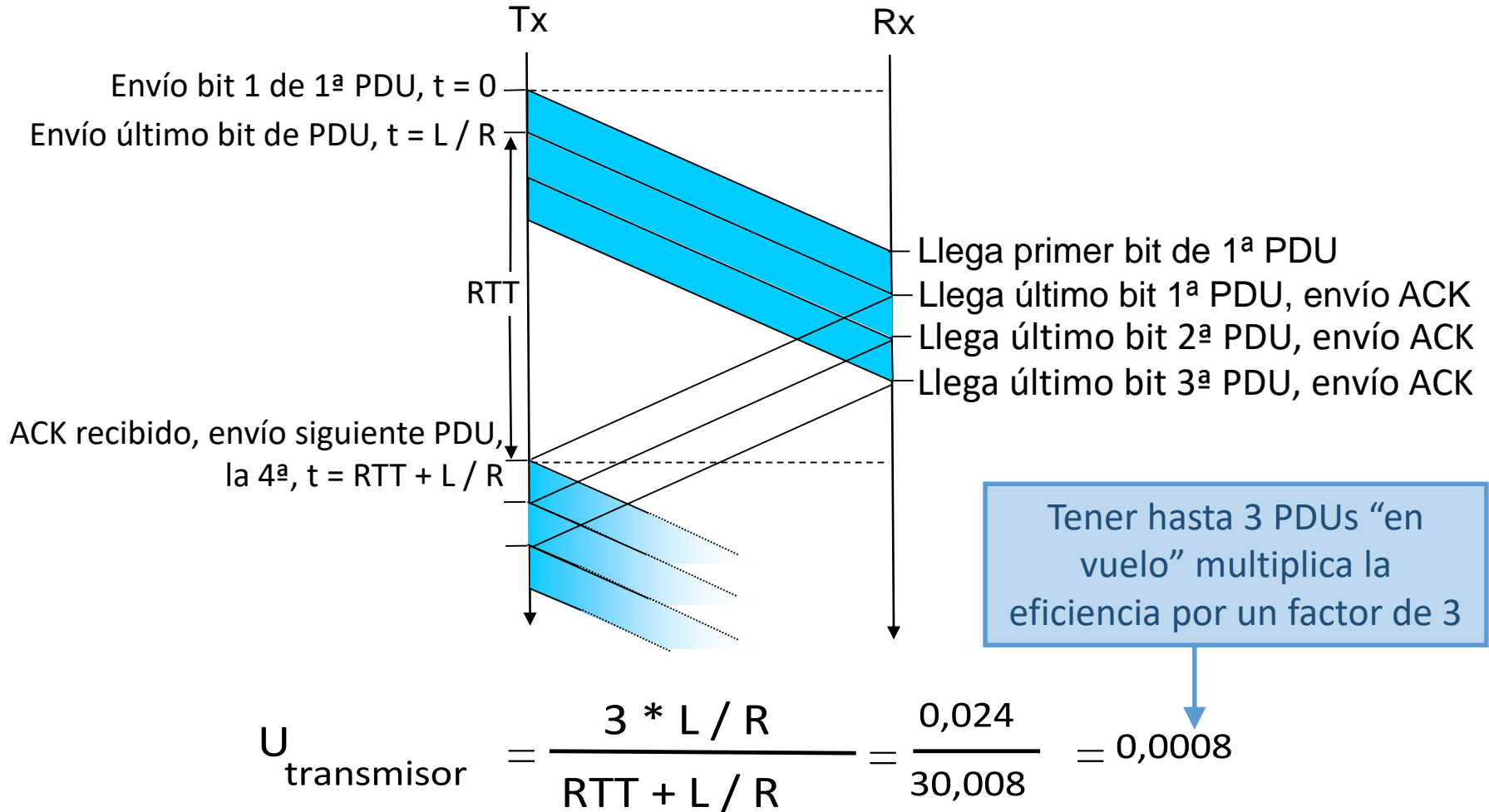


**NOTA:** Profundizaremos en este concepto al abordar TCP



# Principios de la transferencia fiable

## Protocolos con Pipeline - Mejora de la eficiencia



# Tema 3: La Capa de Transporte

## Objetivos

- Entender los principios que hay detrás de los servicios del nivel de transporte:
  - Multiplexión/demultiplexión
  - Transferencia de datos confiable
  - Control de flujo
- Conocer los protocolos de transporte usados en Internet:
  - UDP: transporte no orientado a la conexión
  - TCP: transporte orientado a la conexión

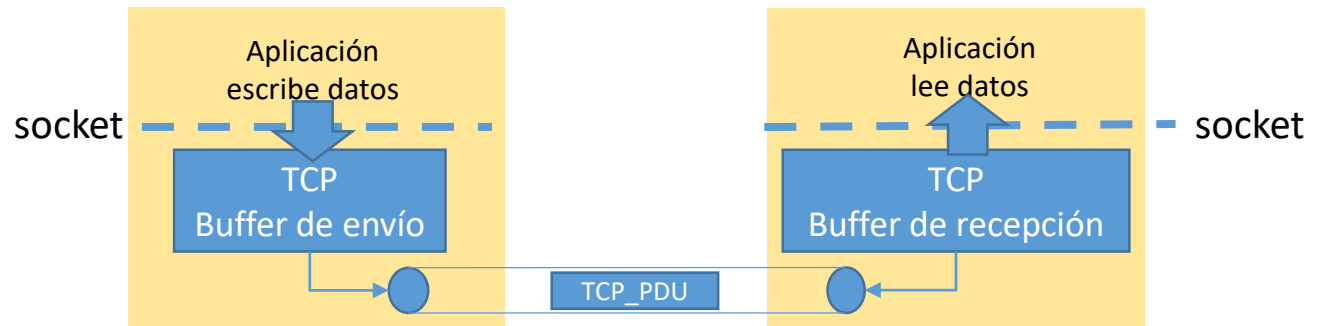
## Contenido

1. Servicios del nivel de transporte
2. Multiplexión y demultiplexión
3. Transporte sin conexión: UDP
4. Principios de la transferencia fiable
5. **Transporte orientado a la conexión: TCP**
  - Estructura del segmento TCP
  - Transferencia de datos fiable
  - Control de flujo
  - Gestión de la conexión

# Transporte orientado a la conexión: TCP

## Estructura del segmento TCP – Visión general

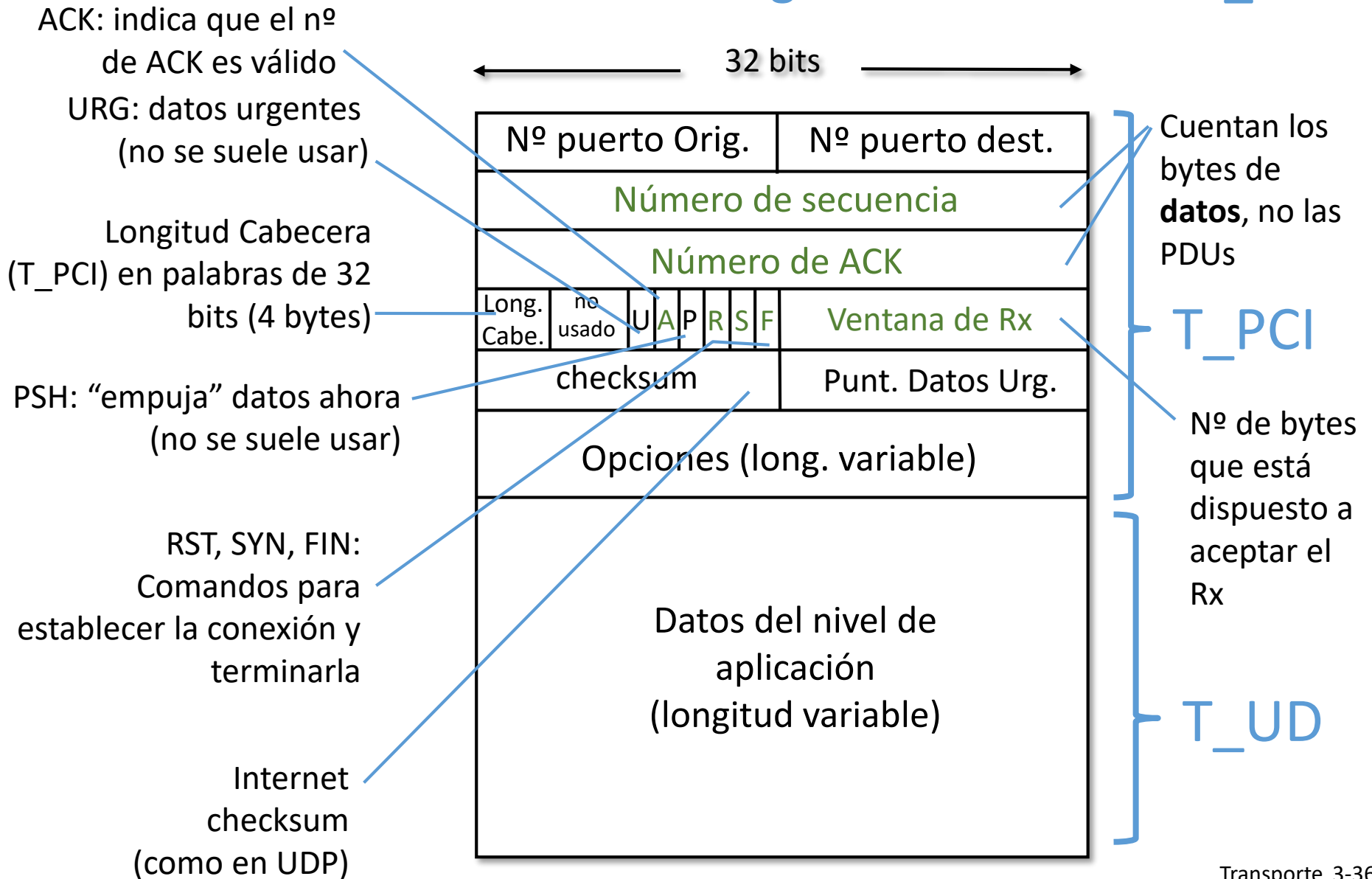
- **RFCs: 793, 1122, 1323, 2018, 2581**
- **Punto a punto:**
  - Un emisor, un receptor (no multidifusión)
- **Flujo de bytes, fiable y ordenado:**
  - Sin “frontera” entre los mensajes (A\_PDUs)
- **Usa “pipeline”:**
  - El control de flujo de TCP fija el tamaño de la ventana (máx. nº datos “en vuelo”)
- **Buffers: Btx y Brx**



- **Servicio Full-Duplex:**
  - Por una conexión fluyen datos bidireccionalmente.
- **MSS:** Tamaño Máximo del Segmento (en realidad, de las T\_UD). Se negocia al iniciar la conexión.
- **Orientado a conexión:**
  - Acuerdo previo al envío de datos. El cliente toma la iniciativa enviando un mensaje de control, que el servidor debe estar esperando.
- **Control de flujo:**
  - El emisor no debe desbordar al receptor.

# Transporte orientado a la conexión: TCP

## Estructura del segmento TCP – La TCP\_PDU



# Transporte orientado a la conexión: TCP

## Estructura del segmento TCP – Nº de secuencia y ACK

### Nº de secuencia:

- Es el número asignado, dentro del flujo de bytes, al **primer byte de los datos del segmento TCP** que se envía a la otra entidad par.
- El valor inicial de este campo lo decide de manera aleatoria cada entidad par al iniciar la conexión.
- Se va incrementado a medida que se envían segmentos que contienen UD.

### Nº de ACK:

- Sirve para indicar el nº de secuencia del byte **que se espera recibir a continuación** por parte de la otra entidad par.
- Todos los bytes anteriores los da por reconocidos (ACK acumulativo).

**P:** ¿Cómo trata el receptor los segmentos desordenados?

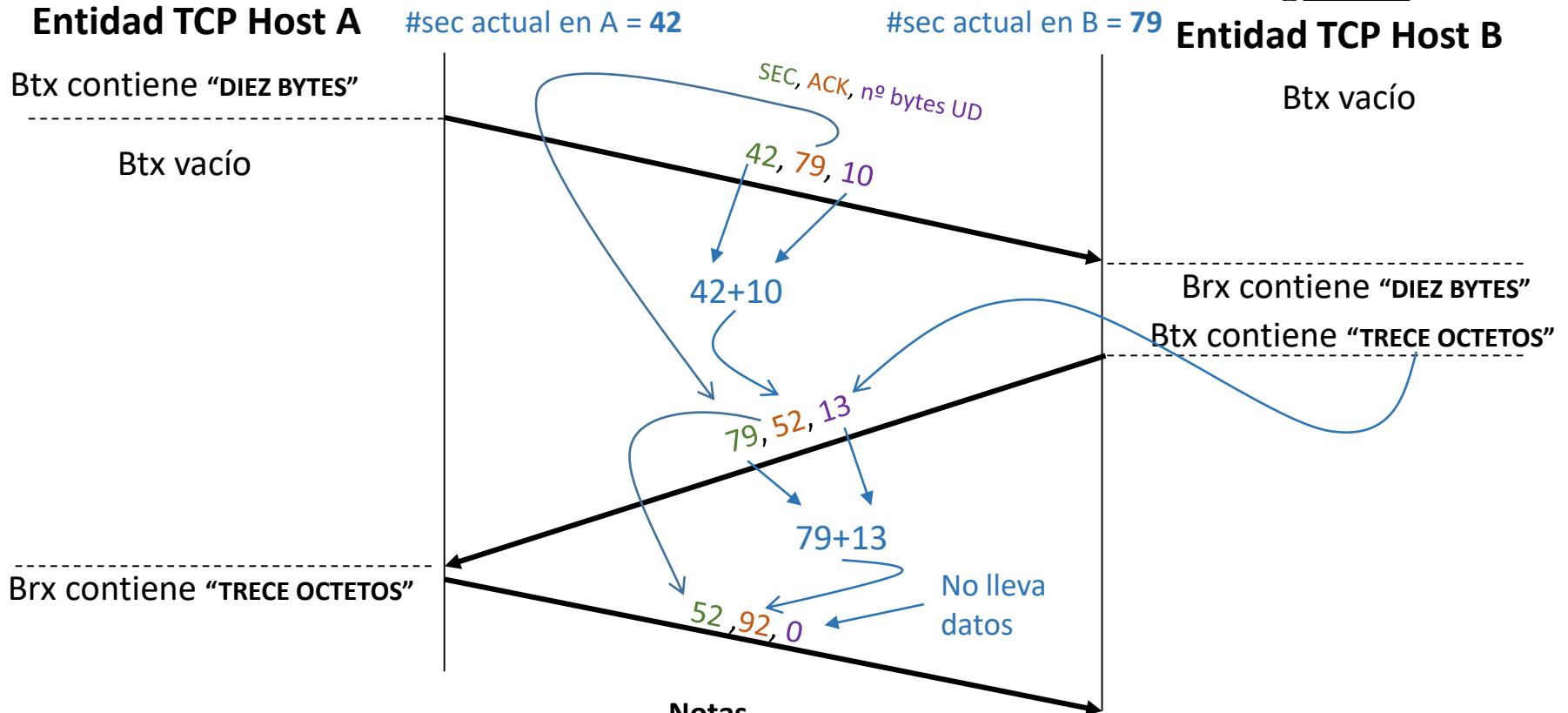
- **R:** La especificación de TCP lo deja a criterio del implementador.

# Transporte orientado a la conexión: TCP

## Estructura del segmento TCP – Nº de secuencia y ACK – Ejemplo



### Escenario simple



#### Notas

- El gráfico muestra intercambio de TCP\_PDU's (segmentos). El buffer de transmisión (Btx) contiene las UD solicitadas por el nivel de aplicación a través del T\_SAP. El buffer de recepción (Brx) se vaciará cuando a través del T\_SAP lo solicite el nivel de aplicación
- TCP envía el ACK "superpuesto" en su PDU de datos ("piggybacking"), como aparece en el segundo segmento que ambos host han enviado.

tiempo

# Tema 3: La Capa de Transporte

## Objetivos

- Entender los principios que hay detrás de los servicios del nivel de transporte:
  - Multiplexión/demultiplexión
  - Transferencia de datos confiable
  - Control de flujo
- Conocer los protocolos de transporte usados en Internet:
  - UDP: transporte no orientado a la conexión
  - TCP: transporte orientado a la conexión

## Contenido

1. Servicios del nivel de transporte
2. Multiplexión y demultiplexión
3. Transporte sin conexión: UDP
4. Principios de la transferencia fiable
5. **Transporte orientado a la conexión: TCP**
  - Estructura del segmento TCP
  - **Transferencia de datos fiable**
  - Control de flujo
  - Gestión de la conexión

# Transporte orientado a la conexión: TCP

## Transferencia de datos fiable en TCP

- TCP crea un servicio de transferencia de datos fiable encima del servicio de transferencia **no** fiable que le proporciona IP.
- TCP usa la técnica del “pipeline”, manteniendo varios segmentos “en vuelo”.
- TCP usa ACKs acumulativos, que reconocen un dato y todos los anteriores.
- TCP usa, por simplicidad, un único temporizador para todas las T\_PDUs de una conexión (lo asocia a los datos más antiguos).
- TCP debe retransmitir los datos cuando:
  - Se produce un `time_out`
  - Llega un ACK duplicado
- Inicialmente consideraremos un emisor TCP simplificado:
  - Ignora ACKs duplicados
  - Sin control de flujo



# Transporte orientado a la conexión: TCP

## Transferencia de datos fiable – Eventos que trata el emisor TCP (simplificado)

### Llegan datos de la aplicación (nivel superior):

- Crea un segmento con un nº de secuencia adecuado y se lo pasa al nivel inferior (IP).
- El nº de secuencia del segmento es el nº de secuencia, dentro del flujo de bytes, del primer byte de datos del segmento.
- Arranca el temporizador, si no estaba ya corriendo (estaría en marcha si había datos anteriores sin reconocer).
- El temporizador se programa para expirar al cabo de `Time_out` segundos.

### Expira el temporizador (`time_out`):

- Retransmite el segmento que ha provocado el `time_out`.
- Rearranca el temporizador.

### Llega un ACK...

- ...correspondiente a datos de los cuales no se había recibido aún un ACK:
  - Actualiza el indicador que apunta a los “datos más antiguos pendientes de ACK” y detiene el temporizador.
  - Arranca el temporizador solo si aún hay “en vuelo” datos pendientes de ACK.

# Transporte orientado a la conexión: TCP

## Transferencia de datos fiable – Emisor TCP (simplificado)

```
SigNumSec = NumeroSecuencialInicial
BaseEmision = NumeroSecuencialInicial
while ( true ) {
    switch( evento )
```

Todos los bytes de datos con nº de secuencia menor a **BaseEmision** están reconocidos acumulativamente. Los de nº mayor o igual están pendientes de ser reconocidos.

```
evento: datos recibidos de la aplicación    /* Llegan datos de la capa superior*/
    crear segmento TCP con datos y número de secuencia SigNumSec
    if ( el temporizador está parado )
        iniciar el temporizador
    pasar el segmento a IP    /* La capa inferior, no fiable, hará llegar el segmento al Rx */
    SigNumSec = SigNumSec + longitud(datos)
```

**SigNumSec** apunta a datos “no enviados”.

```
evento: el temporizador expira                /* Se ha producido un time_out */
    retransmitir el segmento pendiente de ACK con menor número de secuencia
    iniciar el temporizador
```

```
evento: recibido ACK con campo nº de ACK valiendo y
    if ( y > BaseEmision ) { /* Si están reconociendo datos no reconocidos anteriormente */
        BaseEmision = y    /* Actualizar indicador datos reconocidos/pendientes de ACK */
        detener el temporizador
        if ( SigNumSec > BaseEmision ) /* Si hay datos “en vuelo” pendientes de ACK */
            iniciar el temporizador
    }
```

El receptor nos envía acuse de recibo de **todos los datos** con nº de secuencia **menores que y**

```
} /* fin del bucle infinito */
```

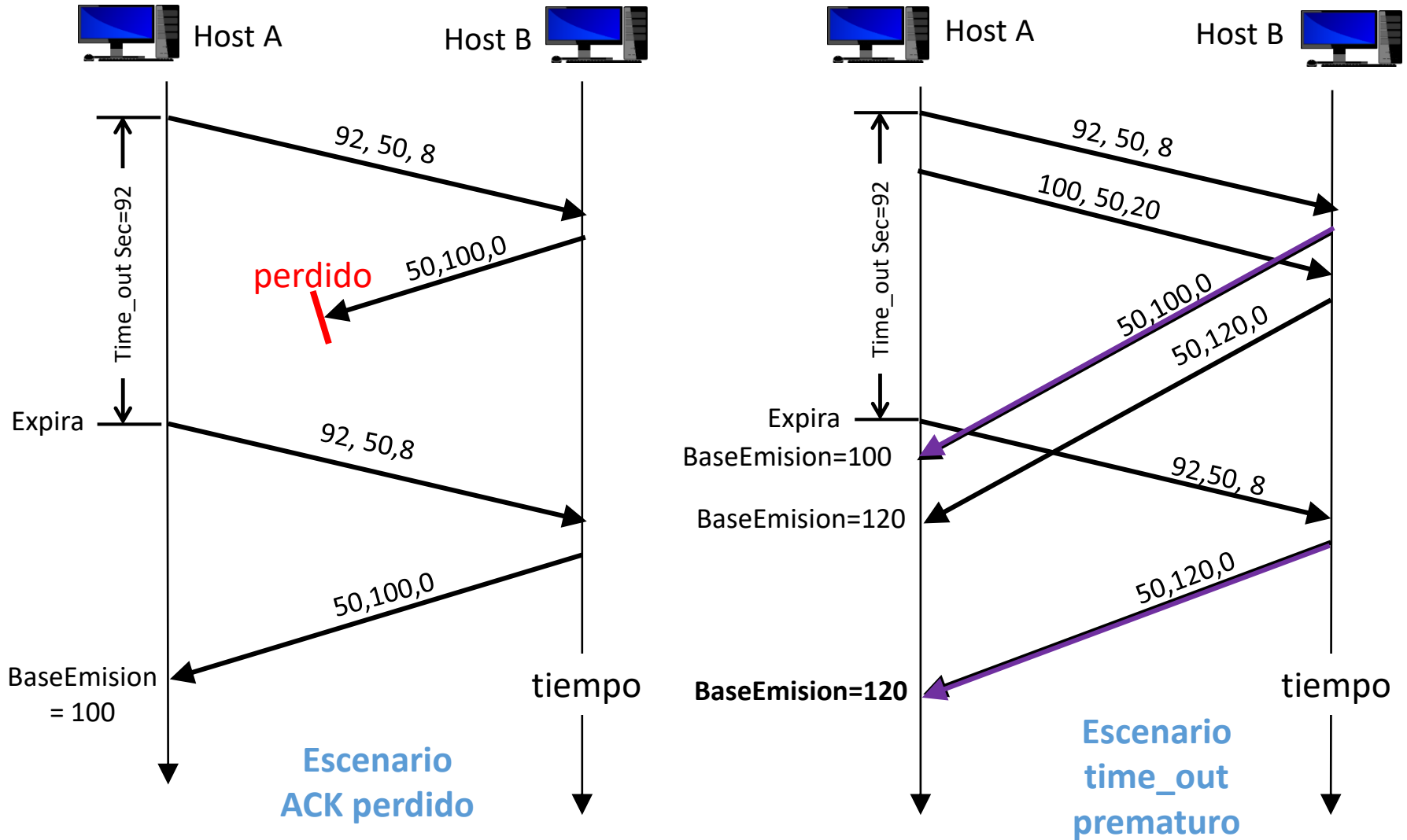
# TCP estima el RTT para saber qué valor usar de `time_out`

¿Qué valor debe usar TCP como `time_out`?

- Debe ser algo mayor que el RTT
  - Pero el RTT cambia a lo largo del tiempo...
- Un valor muy pequeño produce un `time_out` prematuro y retransmisiones innecesarias.
- Un valor demasiado grande hace que se reaccione muy tarde ante la pérdida de un segmento.
- TCP sigue un algoritmo para estimar, en tiempo real, el RTT que existe en cada momento y entonces calcula el timeout

# Transporte orientado a la conexión: TCP

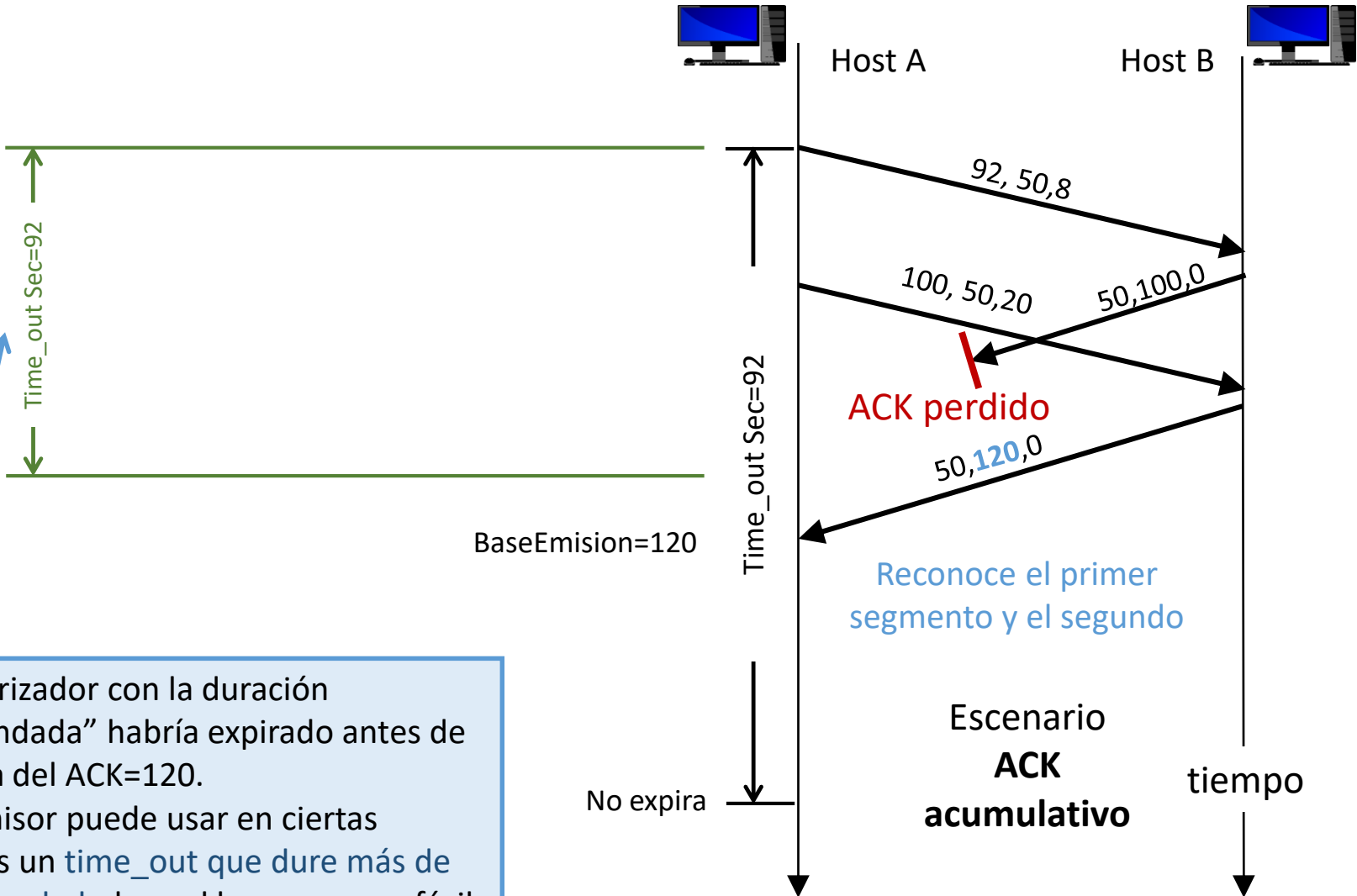
## Transferencia de datos fiable – Escenarios con retransmisiones (I)



Llegan duplicados pero TCP sólo sube uno al nivel de Aplicación.

# Transporte orientado a la conexión: TCP

## Transferencia de datos fiable – Escenarios con retransmisiones (II)

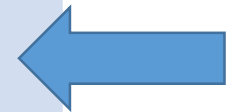


El temporizador con la duración “recomendada” habría expirado antes de la llegada del ACK=120. El transmisor puede usar en ciertas ocasiones un `time_out` que dure más de lo recomendado lo cual hace que sea fácil que se produzcan ACKs acumulativos.

# Transporte orientado a la conexión: TCP

## Transferencia de datos fiable – Generación del ACK [RFC 1122, RFC 2581]

Evento en el receptor TCP	Acción del receptor TCP
Llegada de segmento en orden, con el nº de secuencia esperado. Todos los datos hasta el nº de secuencia esperado ya han sido reconocidos.	ACK retardado. Esperar hasta un máximo de 500ms a que llegue el siguiente segmento en orden. Si no llegase, enviar ACK.
Llegada de segmento en orden, con el nº de secuencia esperado. Hay un segmento anterior pendiente de reconocer.	Enviar inmediatamente un ACK acumulativo que reconozca ambos segmentos ordenados.
Llegada de segmento fuera de orden, con nº de secuencia mayor del esperado. Se detecta un “hueco” en los datos recibidos.	Enviar inmediatamente un <b>ACK duplicado</b> , indicando el nº de secuencia del byte que se espera recibir.
Llegada de un segmento que “rellena” total o parcialmente el “hueco” y que tiene el nº de secuencia esperado (“rellena” el “hueco” por su comienzo o límite inferior).	Enviar inmediatamente un ACK para el segmento recibido o acumulativo dependiendo de si receptor almacena los UD de los segmentos fuera de orden.



# Transporte orientado a la conexión: TCP

## Transferencia de datos fiable – Retransmisión rápida (II)

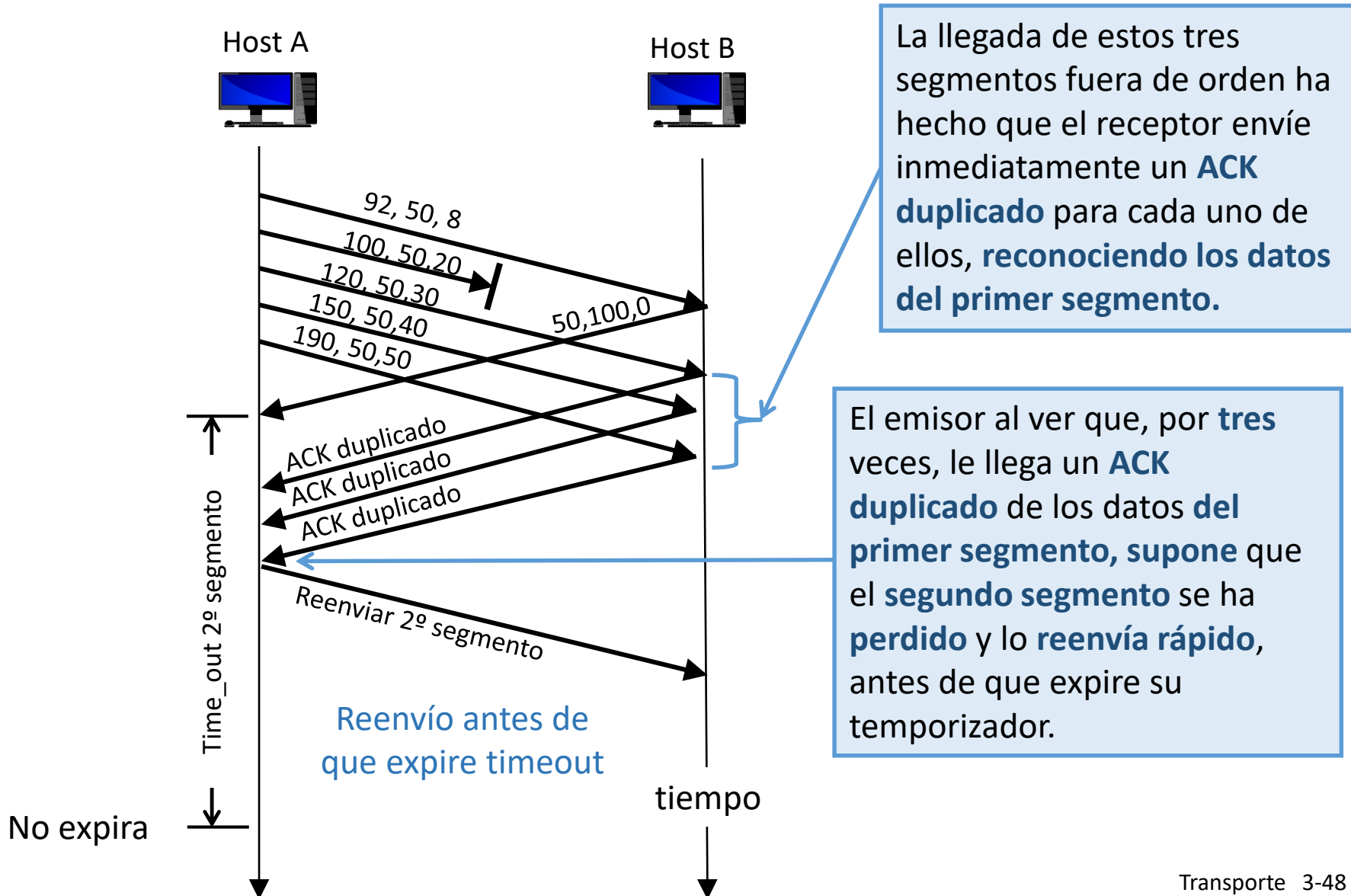
- El `time_out` tiene una duración relativamente larga:
  - Pasa mucho tiempo hasta que se retransmite un segmento perdido.
- Detectar segmentos perdidos gracias a los ACKs duplicados.
  - El emisor a menudo envía muchos segmentos seguidos, muy “pegados”.
  - Si se pierde un segmento, muy probablemente lleguen muchos ACKs duplicados.
- Si el emisor **recibe tres ACKs duplicados** para los mismos datos, supone que se ha perdido el segmento cuyos datos siguen a los datos que están siendo reconocidos:
  - **Retransmisión rápida:** reenviar ese segmento que se supone perdido aunque su temporizador aún no ha expirado.

### Nota

TCP no usa NAKs, por lo que el receptor no puede avisar de que le falta un segmento.

# Transporte orientado a la conexión: TCP

## Transferencia de datos fiable – Retransmisión rápida





# Transporte orientado a la conexión: TCP

## Transferencia de datos fiable – Retransmisión rápida (II)

Este es el evento de recepción de ACK en el emisor TCP simplificado.

**evento:** recibido ACK con campo nº de ACK valiendo **y**

```
if ( y > BaseEmision ) { /* Si están reconociendo datos no reconocidos anteriormente */
    BaseEmision = y /* Actualizar indicador datos reconocidos/pendientes de ACK */
    detener el temporizador
    if ( SigNumSec > BaseEmision ) /* Si hay datos "en vuelo" pendientes de ACK */
        iniciar el temporizador
}
else { /* Se trata de un ACK duplicado de un segmento ya reconocido */
    incrementar el contador de ACKs duplicados de y
    if ( contador de ACKs duplicados de y = 3 ) {
        retransmitir segmento cuyo número de secuencia es y /* retransmisión rápida*/
        iniciar el temporizador
    }
}
```

Le hemos añadido la parte del “else” para que el emisor TCP simplificado no sea tan “simple” y sepa reaccionar ante tres ACKs duplicados, llevando a cabo la retransmisión rápida del segmento no reconocido.

# Tema 3: La Capa de Transporte

## Objetivos

- Entender los principios que hay detrás de los servicios del nivel de transporte:
  - Multiplexión/demultiplexión
  - Transferencia de datos confiable
  - Control de flujo
- Conocer los protocolos de transporte usados en Internet:
  - UDP: transporte no orientado a la conexión
  - TCP: transporte orientado a la conexión

## Contenido

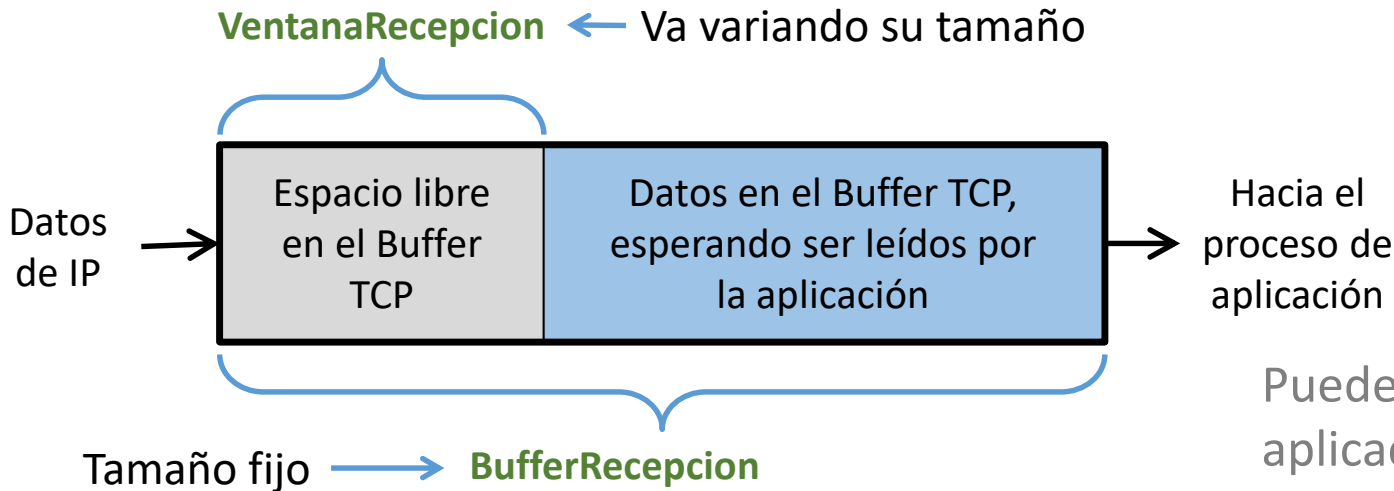
1. Servicios del nivel de transporte
2. Multiplexión y demultiplexión
3. Transporte sin conexión: UDP
4. Principios de la transferencia fiable
5. **Transporte orientado a la conexión: TCP**
  - Estructura del segmento TCP
  - Transferencia de datos fiable
  - **Control de flujo**
  - Gestión de la conexión

# Transporte orientado a la conexión: TCP

## Control de flujo

- El lado receptor de una conexión TCP tiene un **buffer de recepción** donde se acumulan los datos que llegan desde el nivel inferior.

Control de flujo  
Conseguir que el emisor no desborde el buffer del receptor por culpa de transmitir demasiados datos, demasiado rápido.



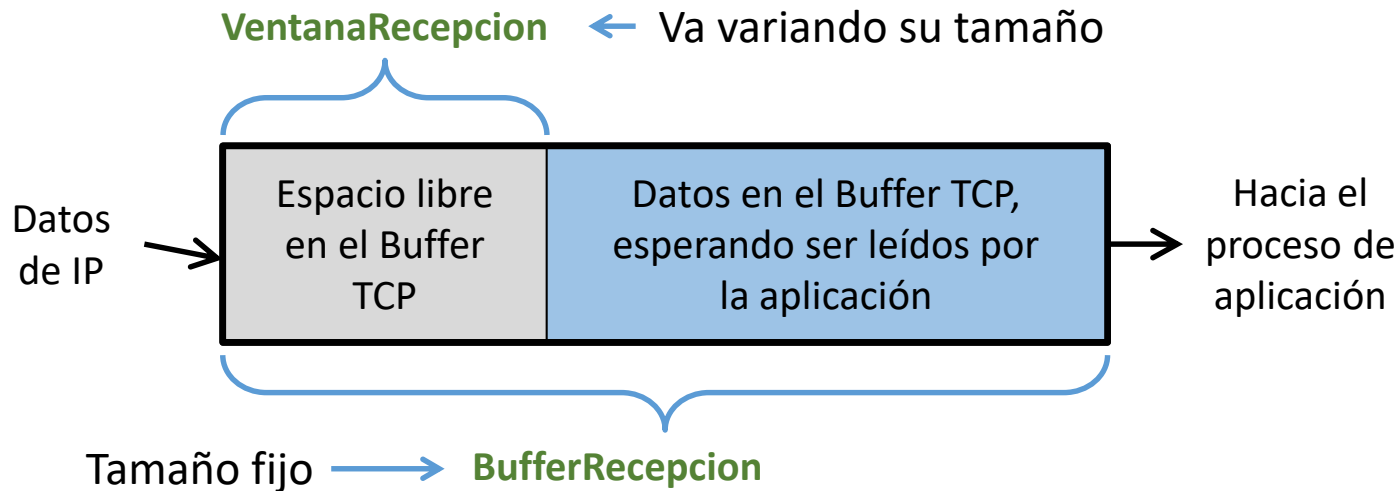
Puede que el proceso de aplicación sea muy lento leyendo del buffer de recepción TCP.

- Servicio de ajuste de velocidades:

Hace que la tasa de transmisión del otro extremo se ajuste al ritmo al que la aplicación receptora “consume” los datos que llegan.

# Transporte orientado a la conexión: TCP

## Control de flujo – Funcionamiento (I)



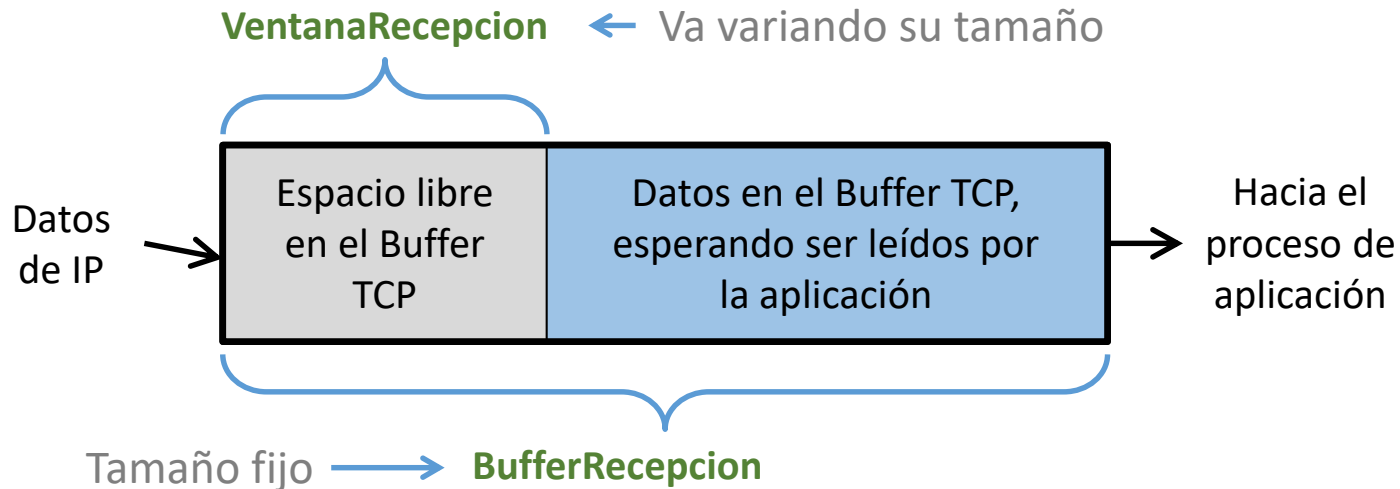
- Supondremos que el receptor TCP descarta los segmentos que recibe desordenados, por lo que esos no ocupan espacio en el buffer de Rx.
- Espacio libre en el buffer de recepción:

$$\text{VentanaRecepcion} = \text{BufferRecepcion} - (\text{UltimoByteRecibido} - \text{UltimoByteLeido})$$

Esta diferencia indica lo que hay “ocupado”

# Transporte orientado a la conexión: TCP

## Control de flujo – Funcionamiento (II)

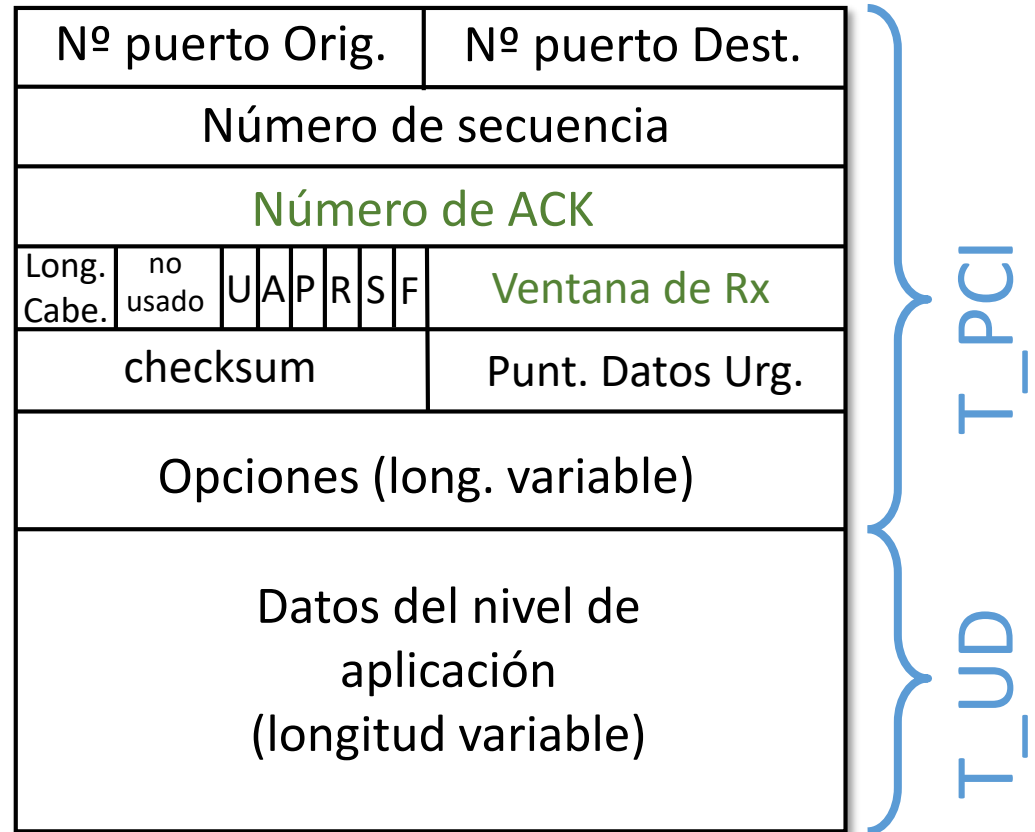


- El receptor informa al emisor del espacio libre en el buffer gracias al campo **VentanaRecepcion** presente en la cabecera (T\_PCI) de los segmentos (T\_PDU) que envía.
- El emisor limita el número de datos “en vuelo” pendientes de ACK de forma que quepan en la **VentanaRecepcion**.
- Esto garantiza que el **BufferRecepcion** nunca se desborda.

# Transporte orientado a la conexión: TCP

## Control de flujo – Funcionamiento (III)

- El valor del campo **VentanaRecepcion** de la T\_PCI está relacionado con el valor del campo **NºdeACK** de la T\_PCI.
- Cuando el emisor recibe un segmento, observa el valor del campo **NºdeACK** y el valor del campo **VentanaRecepcion** para conocer el rango de números de secuencia correspondientes a los datos que está autorizado a tener “en vuelo”, pendientes de confirmación:



**Estructura del segmento (T\_PDU)**

Desde **NºdeACK** hasta (**NºdeACK** + **VentanaRecepcion** – 1)

# Tema 3: La Capa de Transporte

## Objetivos

- Entender los principios que hay detrás de los servicios del nivel de transporte:
  - Multiplexión/demultiplexión
  - Transferencia de datos confiable
  - Control de flujo
- Conocer los protocolos de transporte usados en Internet:
  - UDP: transporte no orientado a la conexión
  - TCP: transporte orientado a la conexión

## Contenido

1. Servicios del nivel de transporte
2. Multiplexión y demultiplexión
3. Transporte sin conexión: UDP
4. Principios de la transferencia fiable
5. **Transporte orientado a la conexión: TCP**
  - Estructura del segmento TCP
  - Transferencia de datos fiable
  - Control de flujo
  - **Gestión de la conexión**

# Transporte orientado a la conexión: TCP

## Gestión de la conexión en TCP: Establecimiento

**Recuerde:** El cliente y el servidor TCP establecen la conexión **antes de intercambiar** segmentos que transporten **datos de usuario**.

Durante esta fase de establecimiento de la conexión, ambos deben inicializar las variables de TCP:

- Números de secuencia a usar.
- Buffers de transmisión y recepción (ambos en cliente y servidor).
- Información de control de flujo (ej: **VentanaRecepcion**).
- etc.

El **cliente** es el que toma la iniciativa de establecer la conexión a través de un socket y el **servidor** está siempre “a la escucha” para recibir el inicio de conexión desde el cliente.



# Transporte orientado a la conexión: TCP

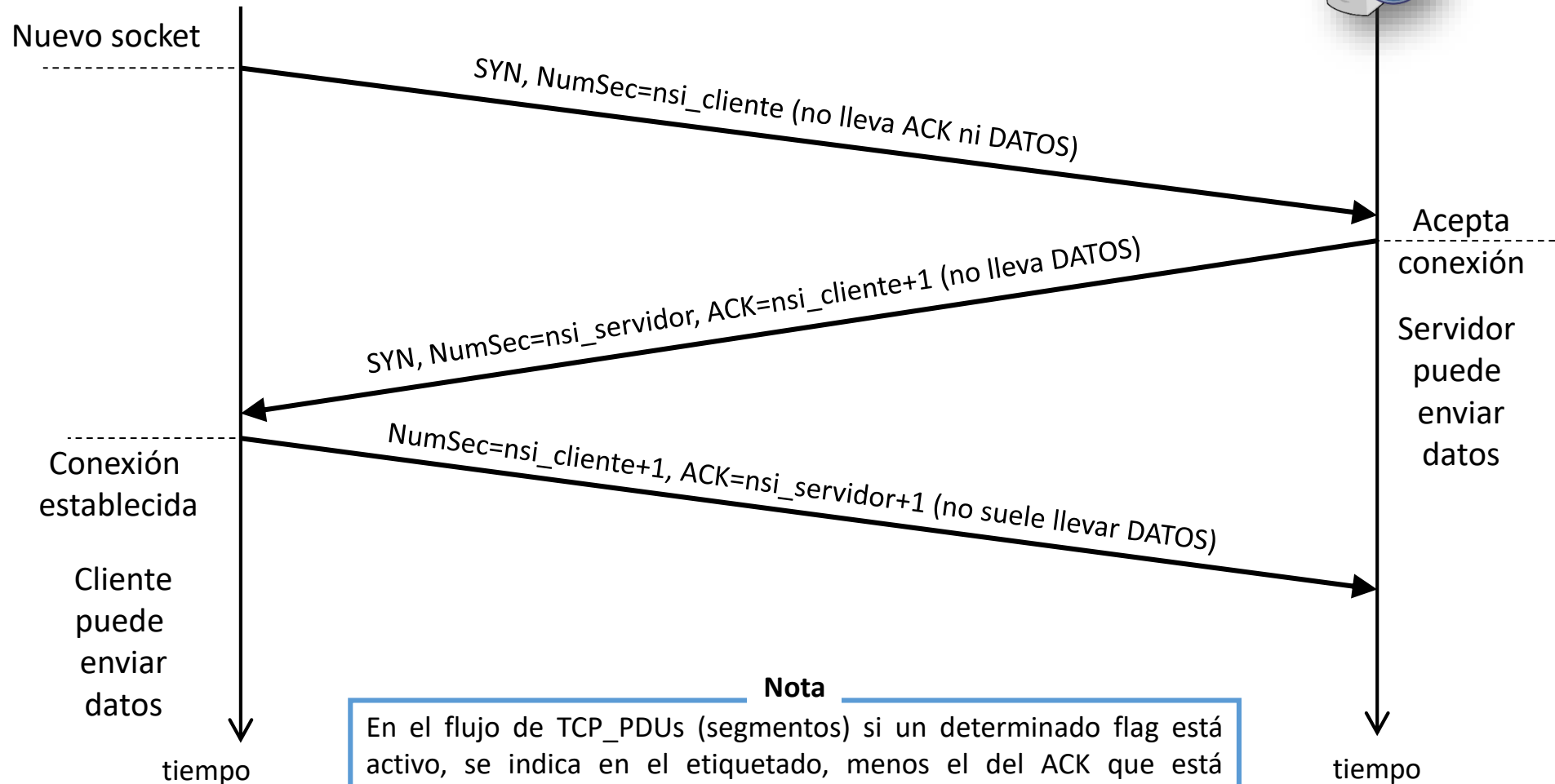
## Gestión de la conexión en TCP: Establecimiento

Cliente



### Proceso de acuerdo en tres fases

Servidor



#### Nota

En el flujo de TCP\_PDU's (segmentos) si un determinado flag está activo, se indica en el etiquetado, menos el del ACK que está implícito cuando el campo ACK tiene valor.

# Transporte orientado a la conexión: TCP

## Gestión de la conexión en TCP: Establecimiento

### Proceso de acuerdo en tres fases:

**Paso 1:** El host cliente inicializa todas las variables de TCP (buffers, número de secuencia inicial, etc.) y envía al servidor un **segmento SYN** que:

- No transporta datos.
- Especifica el Nº de secuencia inicial (**nsi\_cliente**).
- Lleva el bit SYN activado, que a todos los efectos es considerado como el primer byte del flujo de datos de la conexión TCP.

**Paso 2:** El host servidor, al recibir el segmento SYN, inicializa los buffers y variables TCP y responde al cliente enviándole un **segmento SYN-ACK**, que:

- Tiene las mismas características del segmento SYN del cliente pero lo que especifica es el número de secuencia inicial del servidor (**nsi\_servidor**).
- Sirve, además, para reconocer (ACK) que se ha recibido el segmento SYN del cliente.

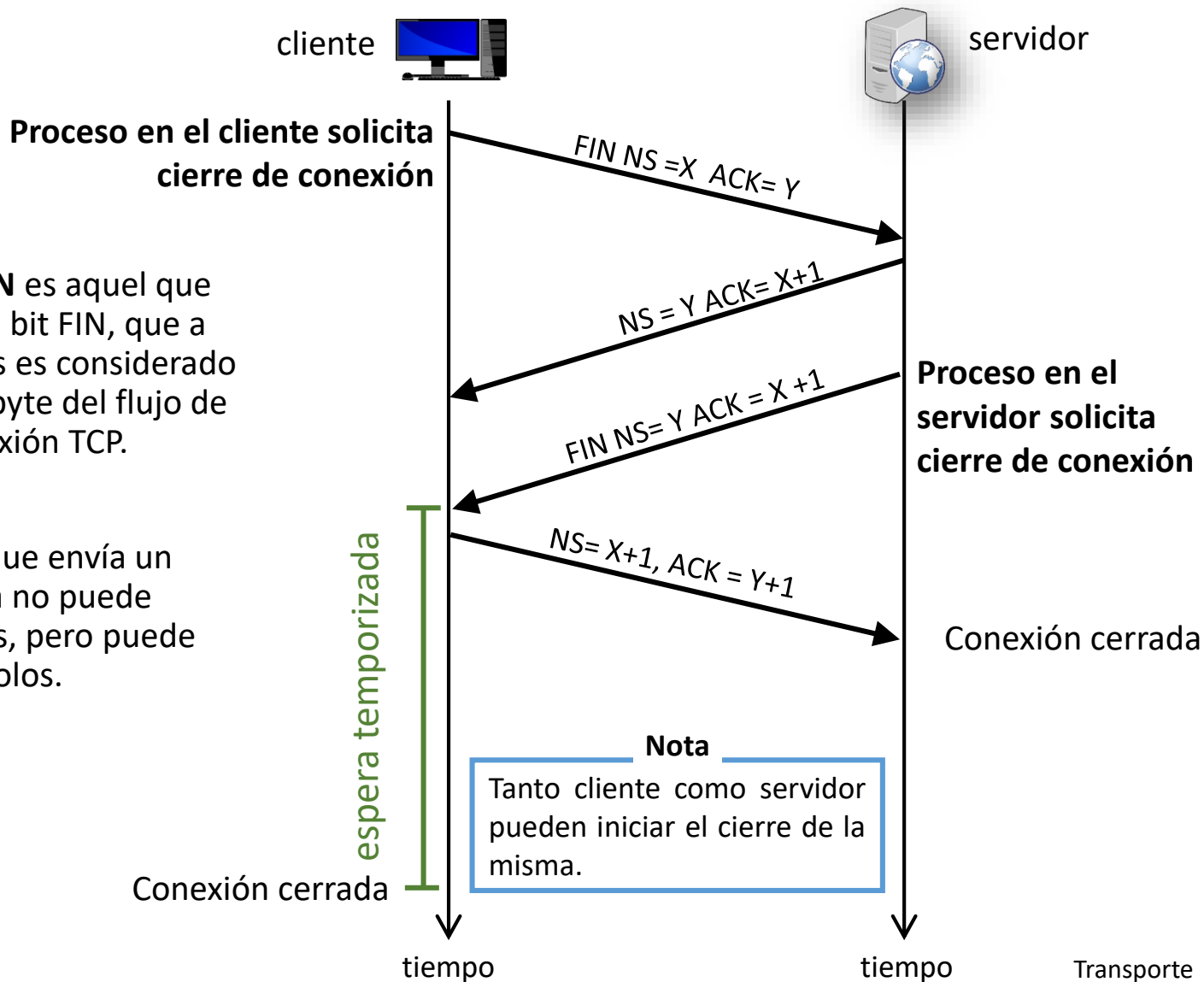
**Paso 3:** El cliente recibe el segmento SYN-ACK del servidor y le responde enviándole un **segmento ACK**, que:

- Sirve para reconocer la llegada del SYN-ACK del servidor.
- Puede contener datos de usuario, aunque no es lo habitual.

# Transporte orientado a la conexión: TCP

## Gestión de la conexión: Cierre de la conexión TCP

- Un **segmento FIN** es aquel que tiene activado el bit FIN, que a todos los efectos es considerado como el último byte del flujo de datos de la conexión TCP.
- La entidad TCP que envía un segmento FIN ya no puede enviar más datos, pero puede seguir recibiendo los.



# Transporte orientado a la conexión: TCP

## Gestión de la conexión: Cierre de la conexión TCP

**Paso 1:** La aplicación cliente decide cerrar la conexión lo cual provoca que la entidad TCP envíe un segmento FIN al servidor.

**Paso 2:** La entidad TCP del servidor recibe el segmento FIN y responde con un ACK al cliente.

**Paso 3:** La aplicación servidora, cuando lo estime oportuno, cerrará la conexión lo cual provocará que la entidad TCP envíe un segmento FIN al cliente.

**Paso 4:** El cliente recibe el segmento FIN y responde con un ACK.

- El cliente entra durante un tiempo en un estado de espera durante el cual, si le llega un FIN del servidor, responde con un ACK.
- Al acabar el tiempo de espera, el cliente TCP dará por cerrada la conexión (liberando buffers y demás recursos asociados a ella).

**Paso 5:** El servidor TCP recibe el ACK correspondiente a su FIN y da por cerrada la conexión TCP (liberando buffers y demás recursos).

# Transporte orientado a la conexión: TCP

## Gestión de la conexión: Cierre de la conexión TCP

- Los pasos anteriores, con alguna modificación menor, son los mismos que se seguirían para cerrar la conexión si:
  - Es la aplicación servidora la que toma la iniciativa a la hora de cerrar.
  - Ambas aplicaciones, la cliente y la servidora, deciden, simultáneamente, cerrar la conexión.
- Las conexiones se deben cerrar de forma abrupta e inmediata cuando se recibe un **segmento RST** (con el bit RST activado). El envío de un segmento de RESET (“reinicio”) se produce solo en casos especiales y no es la forma normal de provocar el cierre de una conexión.

# Tema 3: La Capa de Transporte

## Objetivos

- Entender los principios que hay detrás de los servicios del nivel de transporte:
  - Multiplexión/demultiplexión
  - Transferencia de datos confiable
  - Control de flujo
- Conocer los protocolos de transporte usados en Internet:
  - UDP: transporte no orientado a la conexión
  - TCP: transporte orientado a la conexión

## Contenido

1. Servicios del nivel de transporte
2. Multiplexión y demultiplexión
3. Transporte sin conexión: UDP
4. Principios de la transferencia fiable
5. Transporte orientado a la conexión: TCP
  - Estructura del segmento TCP
  - Transferencia de datos fiable
  - Control de flujo
  - Gestión de la conexión

# Contenidos

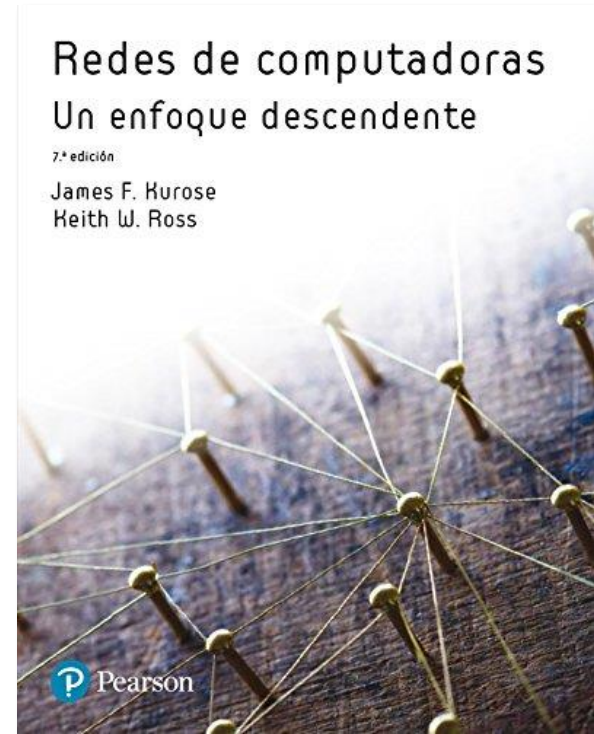
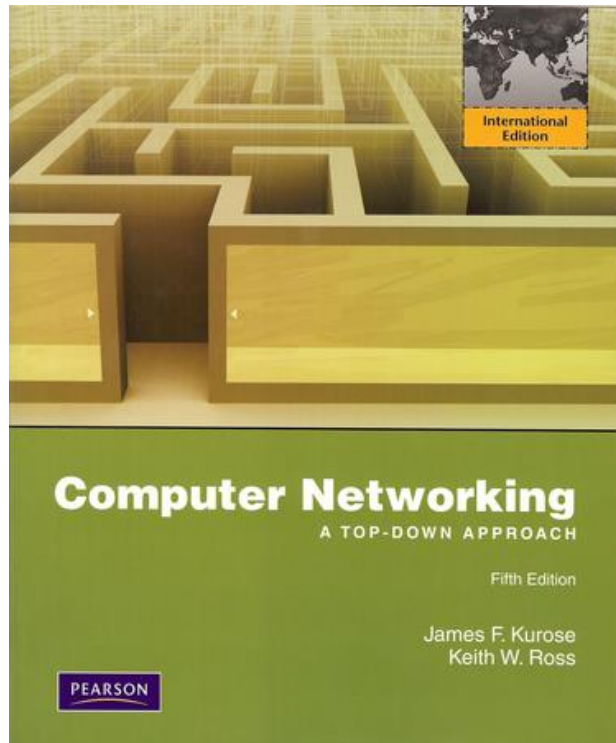
Tema 1: Redes de Computadores e Internet

Tema 2: Capa de Aplicación

Tema 3: Capa de Transporte

**Tema 4: Capa de Red**

Tema 5: Capa de Enlace de Datos



Estas transparencias han sido elaboradas a partir de material con copyright que Pearson pone a disposición del profesorado, a partir del libro:

[Jim Kurose, Keith Ross \(2010\). Computer Networking: A Top Down Approach, 5th edition, Ed. Pearson.](#)

Algunas actualizaciones pertenecen a la última edición:

[Jim Kurose, Keith Ross \(2017\). Redes de Computadoras: Un enfoque descendente, 7ª edición, Ed. Pearson.](#)



# Redes de Computadores

## Tema 3

La Capa de Transporte

**EJERCICIOS**

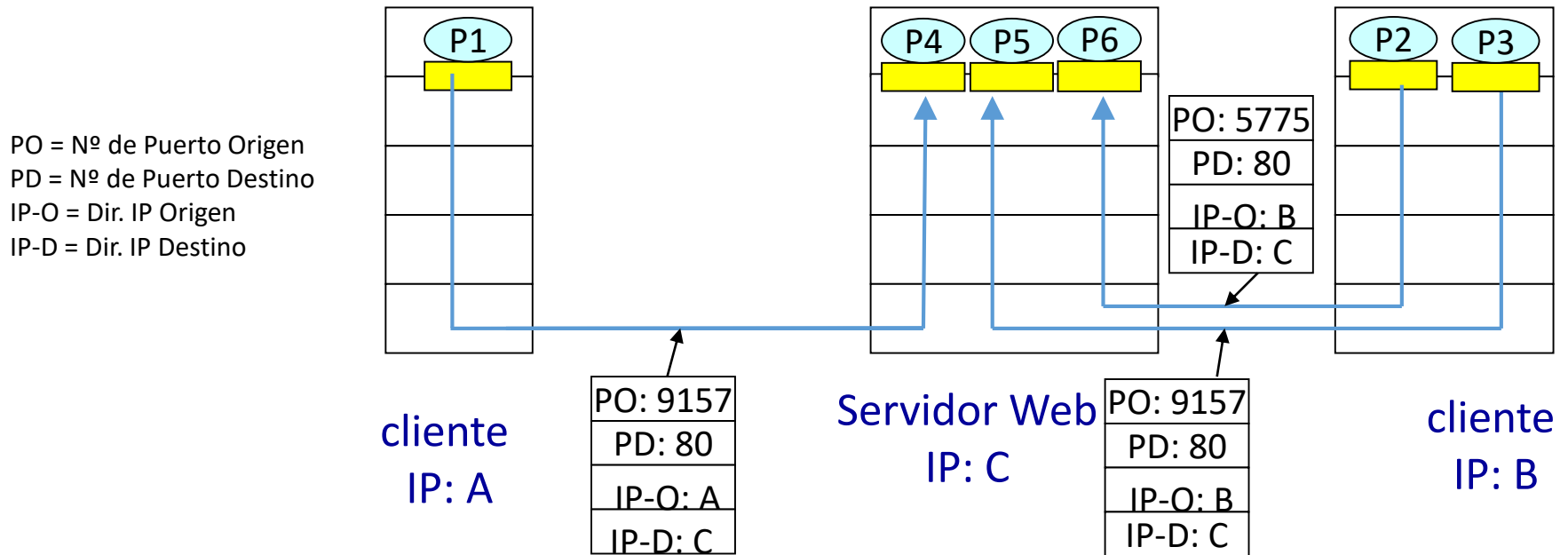


# Problema 1

- Suponga que el cliente A inicia una conexión TCP con un servidor web de nombre S. Más o menos simultáneamente, el cliente B también inicia una conexión TCP con S.
- Indique posibles números de puerto origen y destino para:
  - Los segmentos enviados de A a S
  - Los segmentos enviados de B a S
  - Los segmentos enviados de S a A
  - Los segmentos enviados de S a B
- Si A y B están en host diferentes, ¿podría el número de puerto origen de los segmentos que van de A a S ser el mismo que el de los segmentos que van de B a S?
- ¿Y si los procesos clientes A y B están en el mismo host?

# Problema 2

Observe las conexiones que han iniciado los clientes con el servidor Web y responda a lo siguiente:



- ¿Cuáles son los valores de los puertos de origen y de destino en los segmentos que fluyen desde el servidor de vuelta a los procesos cliente?
- ¿Cuáles son las direcciones IP (origen y destino) de los datagramas de la capa de red (R\_PDU) que transportan a esos segmentos de la capa de transporte?

# Problema 3

Hemos visto que los protocolos con “pipeline” mejoran la eficiencia frente a protocolos de “parada y espera”.

Suponga un Tx y un Rx un enlace de 1Gbps, 30ms de RTT, con PDUs de 1500 bytes y con tamaños de cabecera cero (es decir un tamaño totalmente despreciables frente a los otros tamaños).

¿Cuántas PDUs de datos tiene que tener “en vuelo” el Tx para que la tasa de utilización del canal sea del 95%?

# Problema 4

Una aplicación puede preferir a UDP como protocolo de transporte en lugar de TCP, para así tener un mayor grado de control sobre qué datos se envían en la T\_PDU y en qué instante.

- a) Explique por qué UDP ofrece a la aplicación mayor control sobre qué datos se envían en la T\_PDU.
- b) Explique por qué UDP ofrece a la aplicación mayor control sobre el instante en que se envía una T\_PDU.

# Problema 5

Suponga que un protocolo aplicación cliente desea enviar sólo una PDU de 1000 bytes usando el servicio de transporte fiable de Internet a una aplicación servidora que le responde con 100 bytes.

Realice un diagrama con el flujo de TCP\_PDU's que se intercambiarán etiquetando cada una de ellas con los flags activos, el valor del campo número de secuencia, el valor del campo número de ACK y el nº de bytes de TCP\_UD que transporta. Para cada TCP\_PDU intercambiada debe indicar el tamaño en bytes de la misma. Suponga que TCP no tiene opciones, el número de secuencia inicial (NSI) del cliente es 1000 y el del servidor 3000.

# Problema 6

Se va a transferir de A a B un archivo de gran tamaño ( $L$  bytes) por una conexión TCP en la cual el MSS ha quedado establecido en 536 bytes. Calcule el valor máximo que podrá tener  $L$  si no queremos que los números de secuencia usados en la conexión empiecen a repetirse.

- a) Calcule el tiempo que tardaría en transmitirse un fichero de la longitud  $L$  que ha calculado anteriormente, suponiendo que:
- 1) A y B están conectados por un enlace de 155Mbps
  - 2) Cada segmento TCP se encapsula en un único datagrama IP y este en una única trama, lo cual añade un total de 66 bytes de cabeceras a los datos del nivel de aplicación.
  - 3) Se puede enviar al ritmo máximo, sin peligro de desbordar al receptor, por lo que no tendremos en cuenta el control de flujo.

# Problema 7

Suponga que un protocolo aplicación cliente desea enviar sólo una PDU de 100 bytes usando el servicio de transporte fiable de Internet a una aplicación servidora que le responde con 1000 bytes.

Realice un diagrama con el flujo de TCP\_PDU's que se intercambiarán etiquetando cada una de ellas con los flags activos, el valor del campo número de secuencia, el valor del campo número de ACK y el nº de bytes de TCP\_UD que transporta. Para cada TCP\_PDU intercambiada debe indicar el tamaño en bytes de la misma. Suponga que TCP no tiene opciones, el número de secuencia inicial (NSI) del cliente es 0, el del servidor 0 y el MSS 536.



# Problema 8

Los hosts A y B se están comunicando a través de una conexión TCP. El host B ha recibido de A todos los bytes hasta el byte 126 y A ha recibido sus ACK.

Suponga que A envía ahora a B dos segmentos seguidos, el primero de 70 bytes de datos y el otro de 50.

El N° de secuencia del primer segmento es 127, el N° de puerto origen es el 302 y el N° de puerto destino el 80.

Suponga que B envía un reconocimiento cada vez que le llega un segmento de A.

- a) ¿Qué N° de secuencia, N° de puerto origen y N° de puerto destino tiene el segundo segmento que envió A?
- b) Si el primer segmento llega a B antes que el segundo ¿cuál es el N° de reconocimiento, N° de puerto origen y N° de puerto destino del ACK que enviará B por él?

# Problema 8 (continuación)

- c) Si la capa de red desordena los dos segmentos de A, de forma que el segundo llega en primer lugar a B ¿cuál será el N<sup>o</sup> de reconocimiento del ACK que enviará B nada más recibirlo?
- d) Suponga que los dos segmentos de A a B llegan en orden, y que a los dos ACKs que envía B les ocurre.
- El primero se pierde y no llega a A.
  - El segundo llega después de que en A se haya producido el `time_out` del primer segmento.
- e) Dibuje un diagrama temporal con los dos segmentos que envió A, los dos ACKs de B y añada el resto de segmentos que se van a enviar debido a retransmisiones y nuevos ACKs. Suponga que no se perderán más segmentos. Indique tamaño de los datos, N<sup>o</sup> de secuencia y N<sup>o</sup> de reconocimiento.

# Problema 9

Los hosts A y B se están comunicando a través de una conexión TCP. Ambos están unidos directamente por un enlace de 100Mbps. A está transfiriendo a B un archivo de gran tamaño a través de la conexión TCP.

La capa de aplicación del host A es capaz de enviar datos a su socket TCP a un ritmo de 120Mbps.

La capa de aplicación del host B va sacando los datos del buffer de recepción TCP a un ritmo que no supera los 60Mbps.

Describa el efecto del control de flujo de TCP sobre el ritmo al que la capa de aplicación de A envía datos a su socket TCP.