

# TCP y UDP

Álvaro González Sotillo

19 de febrero de 2018

## Índice

1. Introducción	1
2. UDP	1
3. TCP	2
4. <i>Sequence number y Acknowledgment number</i>	3
5. Mensajes TCP	9
6. Puertos	11
7. TCP vs UDP	13
8. Referencias	14

## 1. Introducción

- TCP y UDP son de la capa de transporte
- Es la primera que une dos entidades (procesos), en vez de dos hosts
- Suele ser la primera capa visible para los programadores de aplicaciones
  - IP se puede considerar la *frontera* entre los *administradores* y los *programadores*

## 2. UDP

- *User Datagram Protocol*
- Funciones:
  - Entregar un datagrama entre el emisor y receptor (procesos)
  - Detección de errores

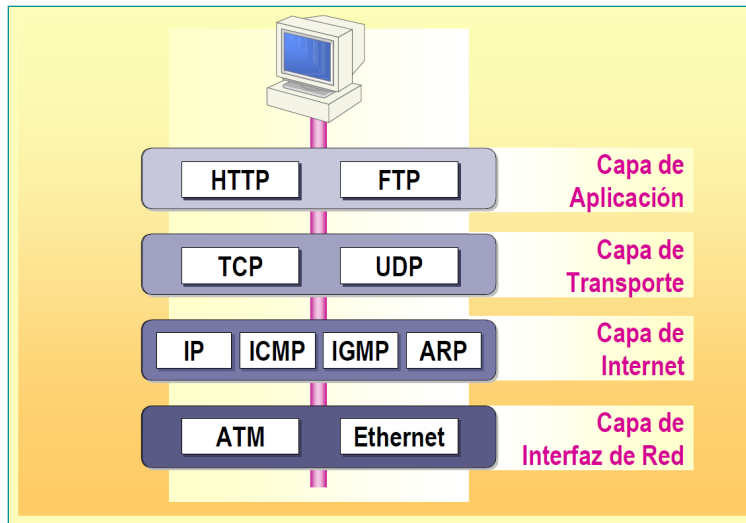


Figura 1: Esquema de niveles de red

### 2.1. Formato de trama UDP

	Bits 0 - 15	16 - 31
0	Source Port	Destination Port
32	Length	Checksum
64	Data	

### 2.2. Características de UDP

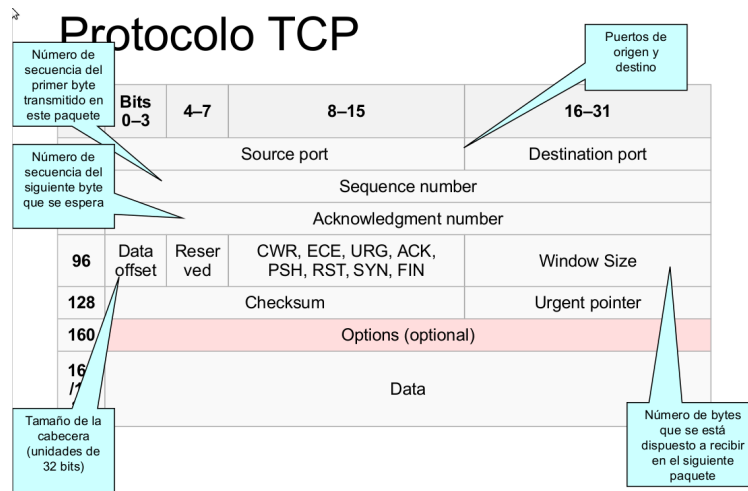
- Los datagramas pueden llegar en un orden diferente al enviado (si IP elige rutas distintas para ellos)
- El emisor no tiene la seguridad de que los datagramas llegan al receptor
- Por tanto, no se utilizan **conexiones** ni es **confiable**. Cada datagrama se envía de forma independiente.

- 
- ¿Cuántos puertos hay?
  - ¿Cuál es el tamaño máximo de un datagrama UDP?

## 3. TCP

- *Transmission Control Protocol*

- Asegura que la transmisión se realiza por un medio fiable
- Garantiza la recepción de los mensajes en orden correcto
- Garantiza al emisor que los mensajes llegan correctamente al receptor
- Por tanto, es orientado a **conexión** y **confiable**.



### 3.1. Ventana y corrección de errores

- TCP necesita confirmación de cada mensaje enviado
  - Para garantizar la confiabilidad
- Opciones:
  - **Parada y espera**: Cada mensaje necesita confirmación
  - **Piggybacking**: La confirmación puede retrasarse algunos mensajes (**ventana**)
- La parada y espera es más simple, pero desaprovecha ancho de banda

## 4. *Sequence number y Acknowledgment number*

- **TCP** intenta que la comunicación se asemeje a un flujo de bytes
  - Todos los bytes que entran por un extremo
  - ... deben salir por el otro lado
- En **cada paquete** se envía
  - el número de secuencia del primer byte transmitido en el paquete
  - el número de secuencia del siguiente byte que se espera

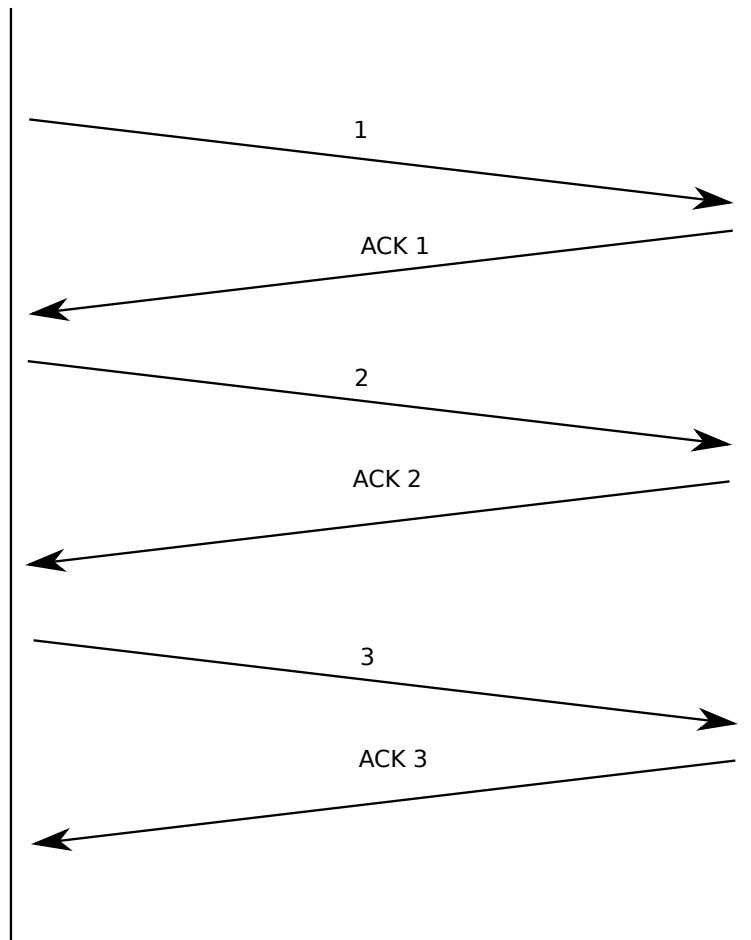


Figura 2: Ejemplo de parada y espera

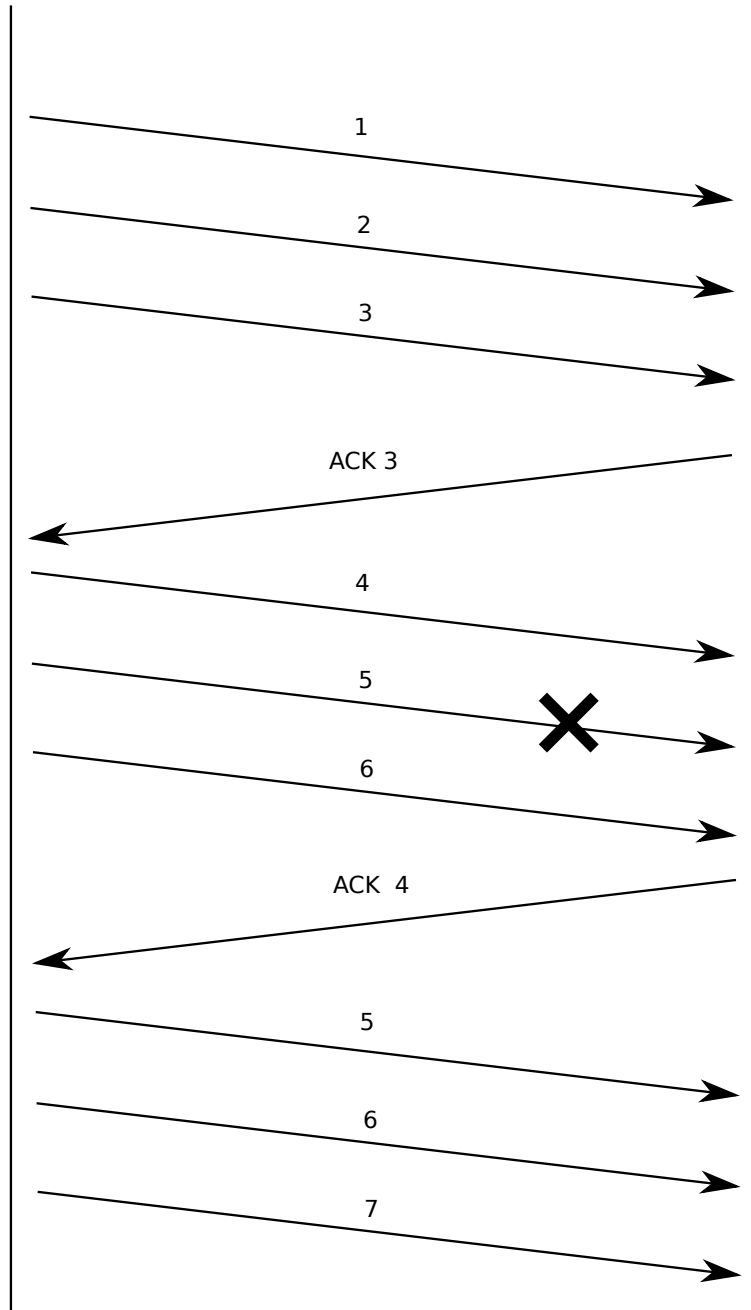


Figura 3: Ejemplo de piggybacking con ventana 3

## 4.1. Estado de la conexión

- Cada extremo de la comunicación debe saber:
  - Cuál es su siguiente byte a enviar (*sequence number*)
  - Cuál es su siguiente byte a recibir (*acknowledgment number*)
- Además, también lleva la cuenta de su opinión acerca del *sequence number* y del *acknowledgment number* del otro extremo

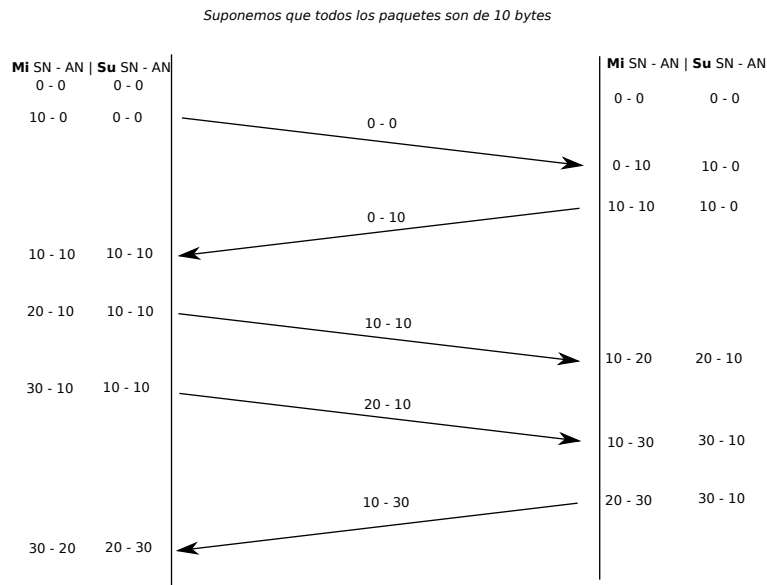


Figura 4: *Sequence number y acknowledgment number*

## 4.2. Corrección de errores

- Si se recibe un *sequence number* posterior a nuestro *acknowledgment number*
  - Es un paquete **posterior** al que esperamos
    - Se puede guardar en la capa **TCP** hasta que lleguen los anteriores
    - O se puede ignorar, y **reclamar los paquetes perdidos** enviando un *acknowledgment number* menor que el que espera el otro
  - El otro lado reenviará los paquetes necesarios
- Si se recibe un *sequence number* anterior a nuestro *acknowledgment number*
  - Es un paquete ya recibido (se habrá duplicado)
  - Por tanto, se ignora
- Si me llega un *acknowledgment number* menor que los bytes que yo he enviado

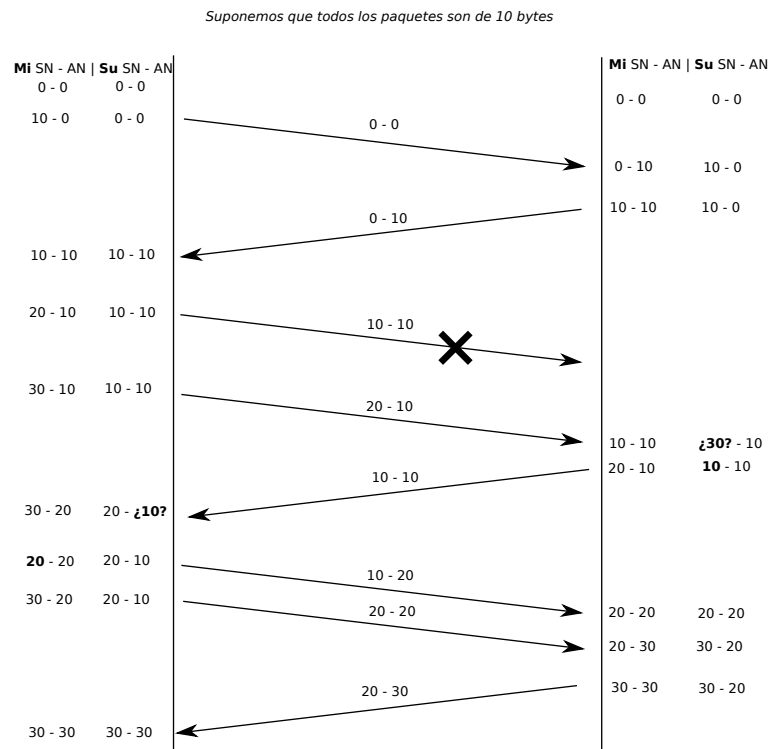


Figura 5: Llegada de un paquete posterior al esperado

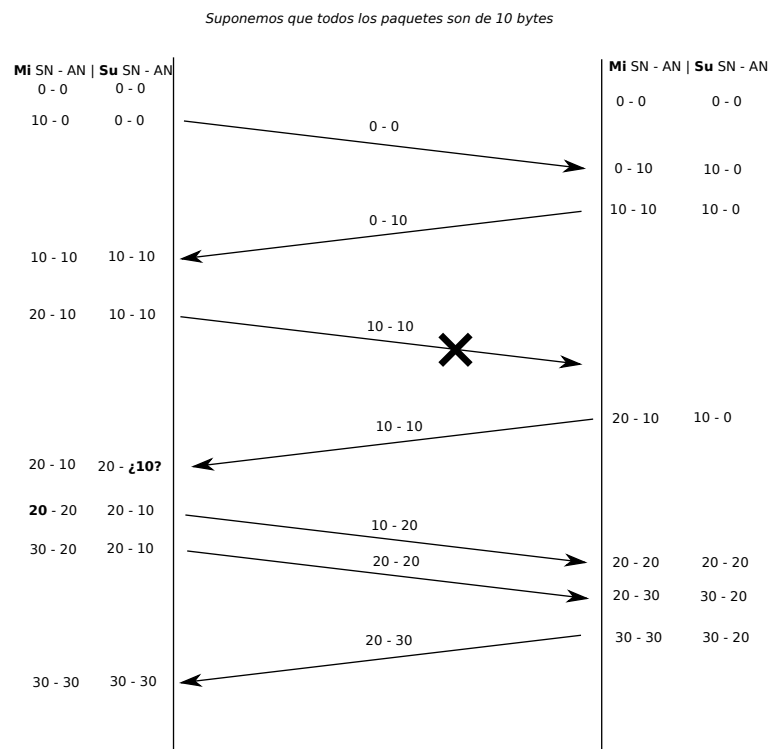


Figura 6: El otro lado informa de la pérdida de algún paquete



- Reenviaré a partir de dicho *acknowledgement number*
- Si no tengo confirmación de un paquete enviado tras un *timeout*
  - Reenviaré el paquete

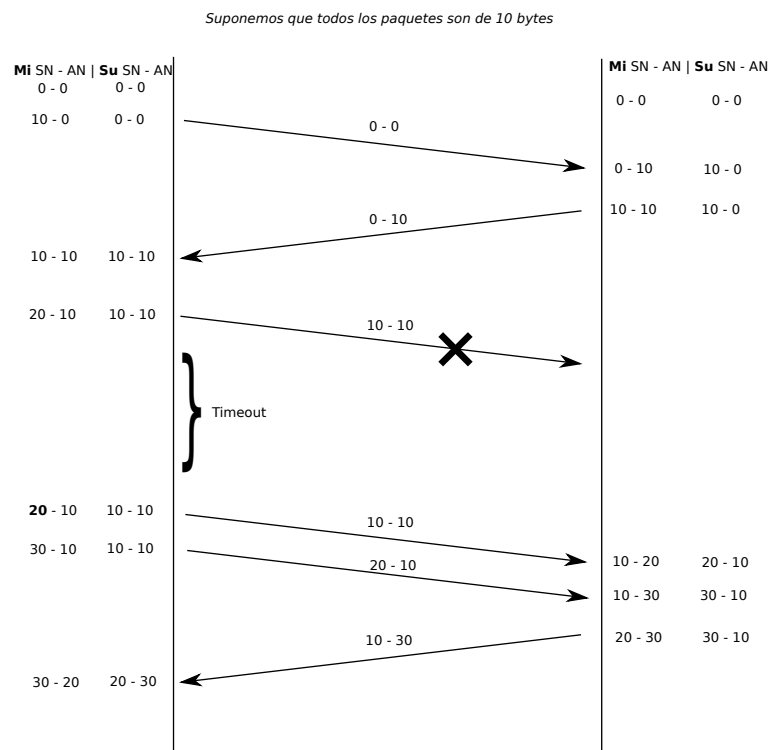
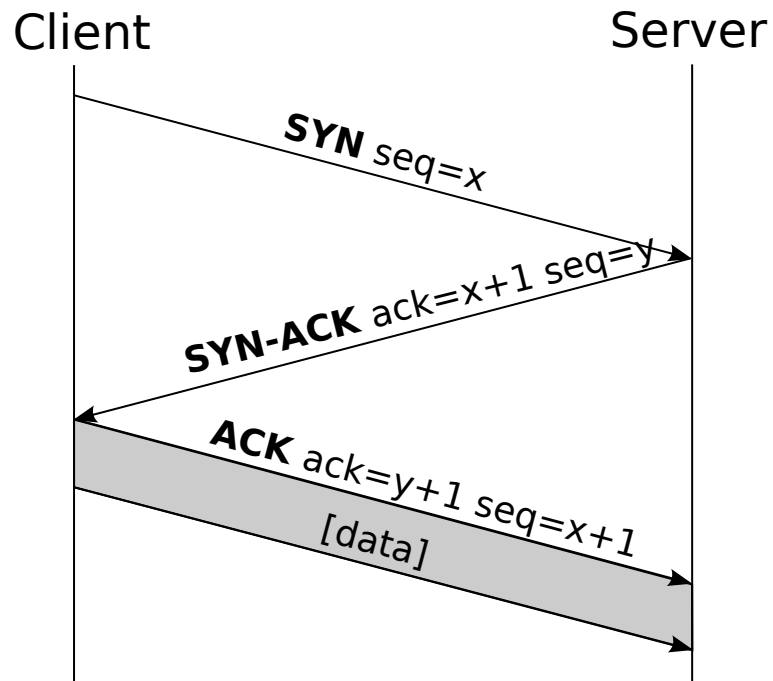


Figura 7: Paquete no confirmado tras un *timeout*

## 5. Mensajes TCP

### 5.1. Establecimiento de conexión

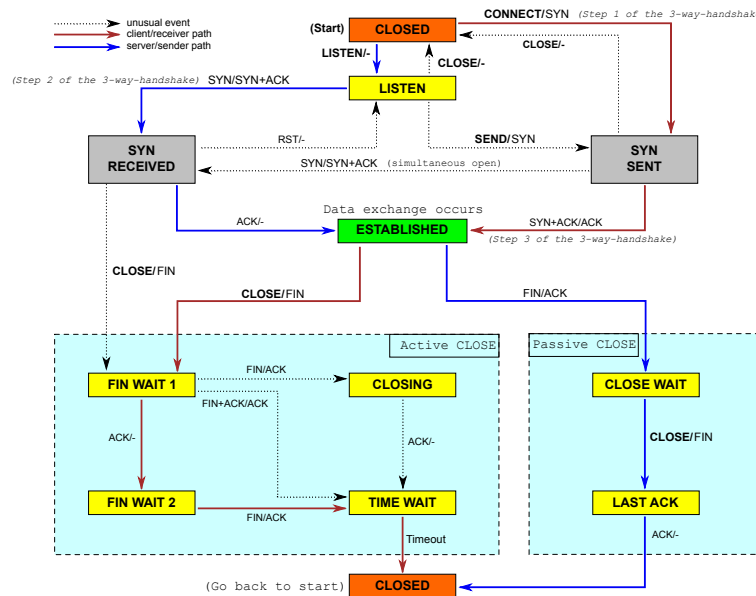
1. Un servidor escucha en un puerto
2. Un cliente envía una solicitud de conexión
3. El servidor responde con una aceptación de la conexión
4. El cliente *acepta* la aceptación



Fuente: By Snubcube

## 5.2. Estados TCP

- Los principales estados son:
  - **Closed:** Ninguna conexión
  - **Listening:** Un servidor está esperando en un puerto a ser conectado
  - **Established:** Un cliente ha conectado con un servidor
  - **Time wait:** Esperando a que la conexión termine
- Todos los estados en <http://www.medianet.kent.edu/techreports/TR2005-07-22-tcp-EFSM.pdf>



Fuente: [By Scil100](#)

## 6. Puertos

- Se llama puerto a la dirección de nivel de transporte en
  - TCP
  - UDP
- La asignación de puertos se realiza según el [RFC 1700](#)

<b>Well-known ports</b>	0 - 1023	Solo para el administrador
<b>Registered ports</b>	1024 - 49151	Generalmente, servicios menos críticos
<b>Dynamic ports</b>	49152 to 65535	Asignables a los clientes

### 6.1. Asignación de puertos

- **Servidor:** El proceso escucha en un puerto conocido
  - Ejemplos: 80 para HTTP, 25 para SMTP...
  - Lista de *well known ports* en el fichero `=/etc/services=`
- **Cliente:** El cliente inicia una conexión un servidor.
  - El sistema le asigna un puerto no utilizado cualquiera
  - Generalmente, un puerto dinámico
  - El cliente también puede solicitar un puerto, pero es poco frecuente

---

## 6.2. Comando netstat

- Informa de
  - Las conexiones TCP y UDP activas
  - Los programas escuchando en puertos TCP y UDP

	Linux	Windows
Conexiones TCP	-t	-p tcp
Conexiones UDP	-u	-p udp
Proceso	-p	-o
No traducir direcciones	-n	-n
Escuchando	-l	
Escuchando y establecidas	-a	-a

### 6.2.1. Ejercicio

- Comprueba qué conexiones están establecidas en tu ordenador
- Comprueba a qué direcciones está conectado tu ordenador
- Comprueba qué puertos admiten conexiones
- Comprueba qué procesos admiten conexiones

## 6.3. Comando nc

- **Netcat** permite realizar conexiones TCP/UDP y redirigir su entrada/salida
- **Usos**
  - Simular de forma rápida un cliente para probar un servidor
  - Simular un servidor para probar un cliente
  - Comprobar si el *firewall* permite conexiones
  - Transferir información por red
- Versión Windows:
  - <https://eternallybored.org/misc/netcat/>
- Otras opciones para Windows
  - Máquina virtual con Linux
  - [powercat](#)
  - [Windows Subsystem for Linux](#)

### 6.3.1. Ejercicio

- Pon a **netcat** a escuchar en un puerto
- Haz que un compañero se conecte a ese puerto
- Utiliza **netcat** como chat.

---

### 6.3.2. Youtube, 1990

`nc towel.blinkenlights.nl 23`

## 7. TCP vs UDP

- TCP es un medio de transmisión asegurado
  - Las aplicaciones que usan TCP no envían paquetes, sino bytes.
  - TCP decide cuando enviar un paquete (las aplicaciones pueden *opinar*)
  - Consume más CPU y memoria, por la ventana de emisión y los reenvíos
- UDP es más eficiente
  - No necesita mantener conexión, ni reordenar paquetes, ni retransmitir paquetes
  - Las aplicaciones son *conscientes* de que se envían paquetes, no bytes.
  - En redes con pocos errores, puede ser más adecuado
  - Interesante cuando se necesita mucho ancho de banda pero no importa perder algún paquete (voz, vídeo)

### 7.1. TCP Joke

1. Hello, would you like to hear a TCP joke?
2. Yes, I'd like to hear a TCP joke.
3. OK, I'll tell you a TCP joke.
4. OK, I'll hear a TCP joke.
5. Are you ready to hear a TCP joke?
6. Yes, I am ready to hear a TCP joke.
7. OK, I'm about to send the TCP joke. It will last 10 seconds, it has two characters, it does not have a setting, it ends with a punchline.
8. OK, I'm ready to hear the TCP joke that will last 10 seconds, has two characters, does not have a setting and will end with a punchline.
9. I'm sorry, your connection has timed out. . . . Hello, would you like to hear a TCP joke?

### 7.2. UDP Joke

- Lo anterior era una broma de TCP. No me importa si la pillas o no.

---

## 8. Referencias

- Formatos:
  - [Transparencias](#)
  - [PDF](#)
- Creado con:
  - [Emacs](#)
  - [org-reveal](#)