FUNDAMENTOS DE LA CAPA 4

TCP y UDP – Multiplexación, 3-way-Handshake, FIN

…

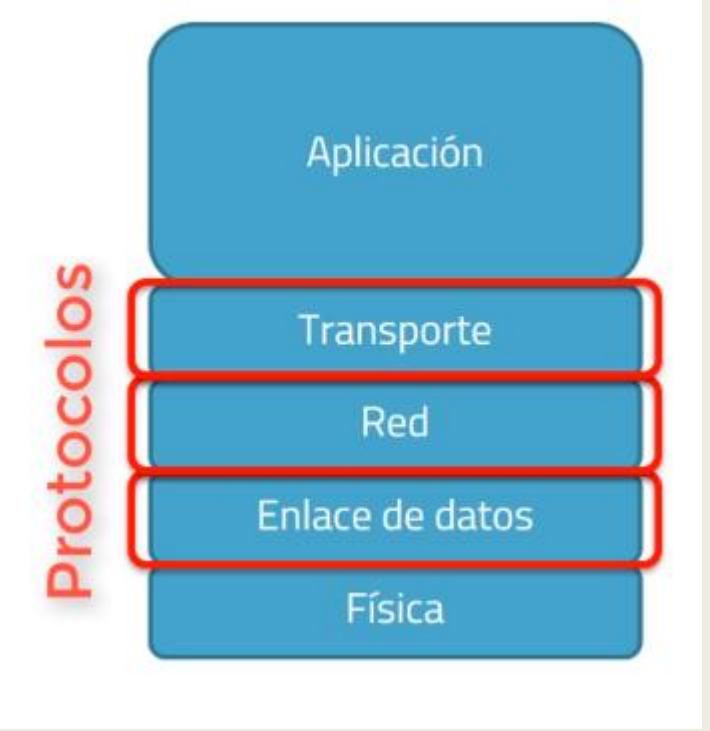
Índice

- Funciones de los protocolos TCP y UDP - Multiplexación
- TCP: Establecimiento de la conexión. 3-Way-Handshake
- TCP: Análisis del 3-Way-Handshake con Wireshark
- TCP: Finalizado de la conexión y envío de datos + Wireshark
- TCP: Control de flujo, recuperación de errores, etc.
- TCP y UDP: Checksum

FUNCIONES DE LOS PROTOCOLOS TCP Y UDP - MULTIPLEXACIÓN

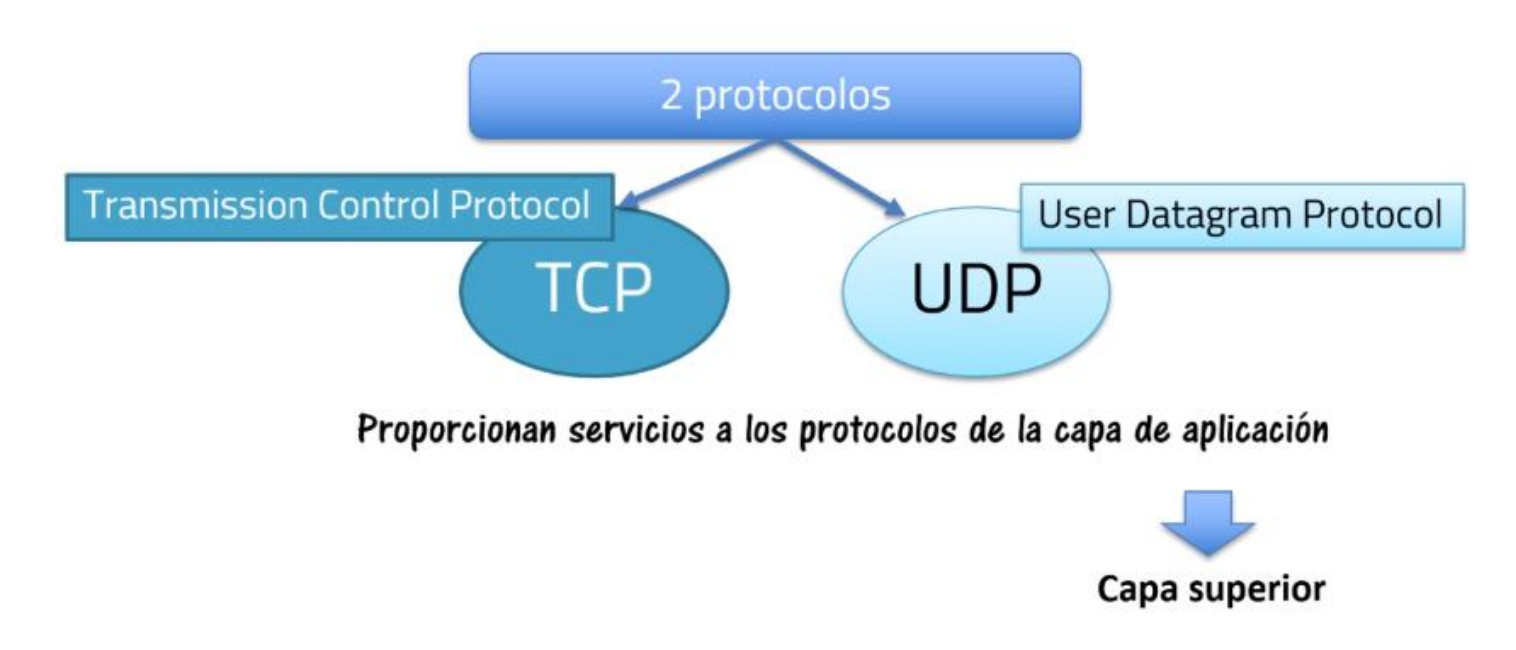
Introducción

- La capa de transporte (4), al igual que la capa de red (3) y la capa de enlace (2), está formada por PROTOCOLOS.
- En la capa 2 el protocolo más importante es Ethernet
- En la capa 3 el protocolo más importante es IP
- ¿Cuál o cuáles serán los protocolos más importantes de la capa 4?



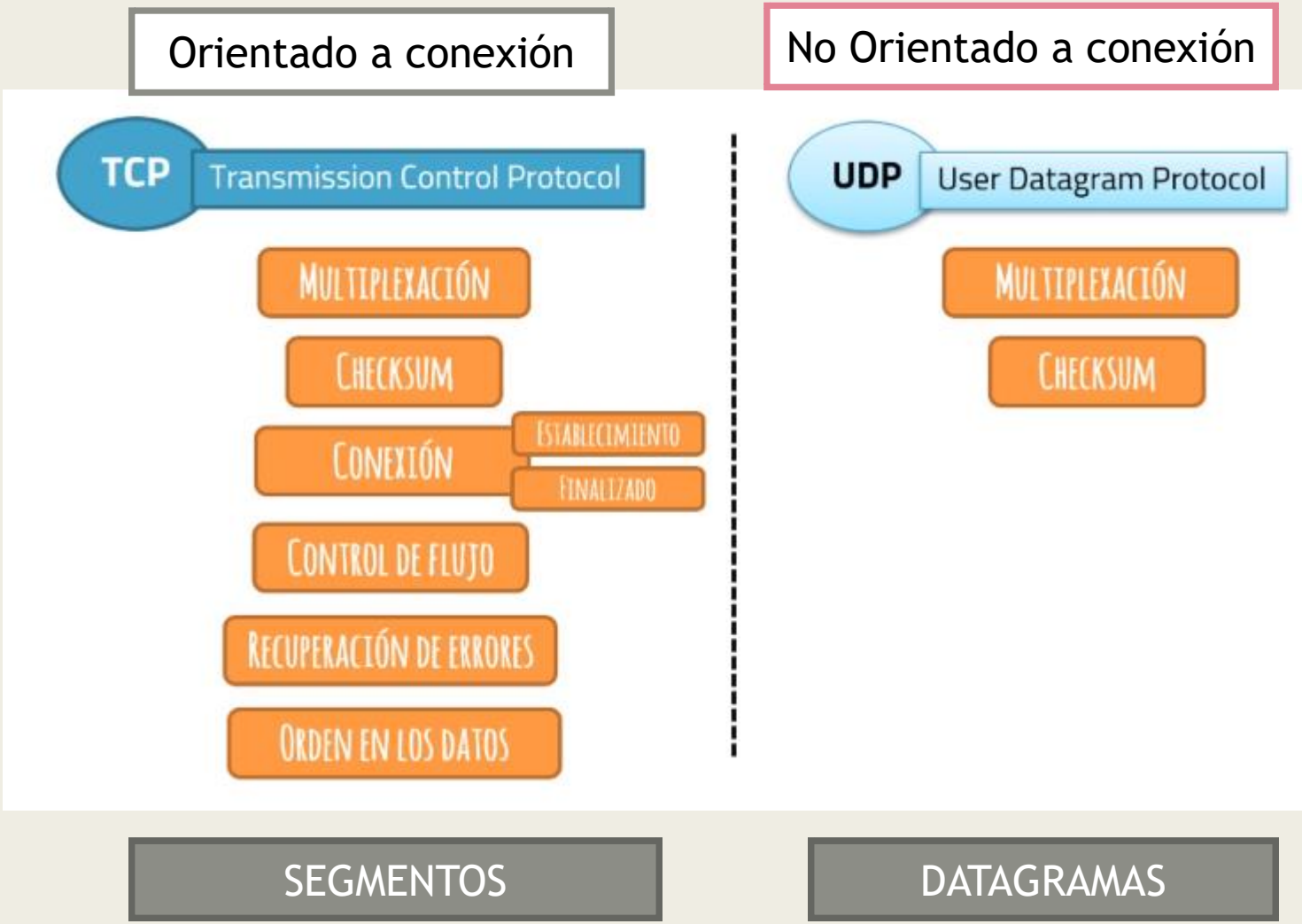
Protocolos más importantes de la capa de transporte

- La capa de transporte cuenta con dos protocolos principales: **TCP y UDP.**
- Al igual que sucede con el resto de capas, a través de la capa de transporte se proporcionan servicios a la capa superior, en este caso, la capa de aplicación.



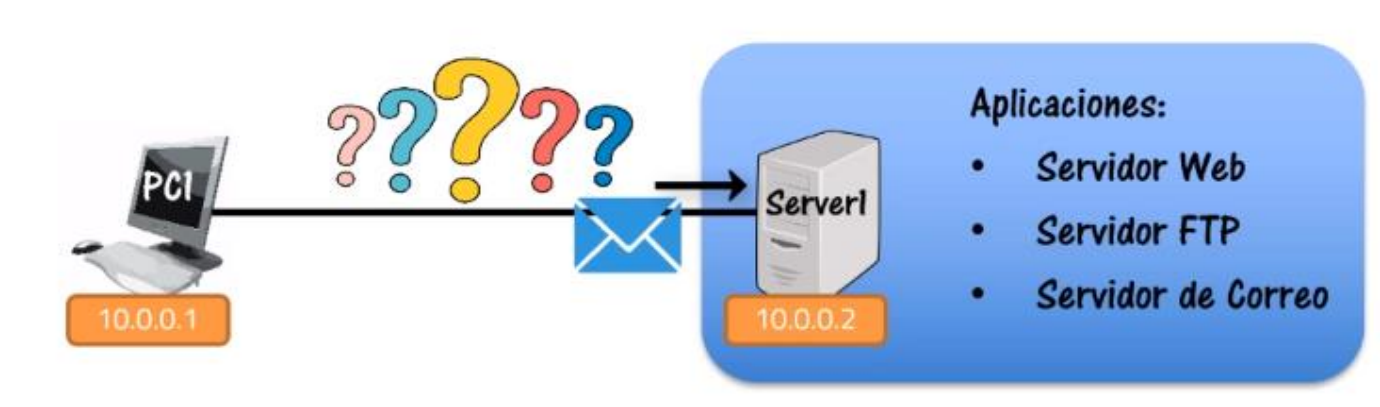
Características de TCP y UDP

- Entre las características principales de ambos protocolos, destaca que **TCP tiene muchas más funciones que UDP**.
- Además, cuando hablamos de TCP, hablamos de un protocolo ORIENTADO A CONEXIÓN, y con UDP hablamos de un protocolo NO ORIENTADO A CONEXIÓN.



Característica: MULTIPLEXACIÓN 1/8

- La multiplexación permite varias comunicaciones de forma simultánea.

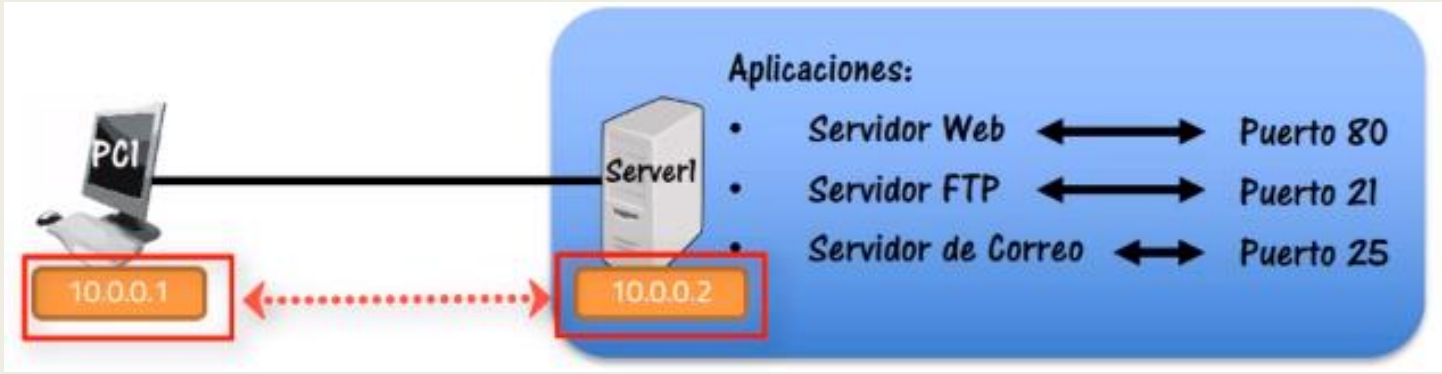


- Utilizando solo las IPs, ¿cómo podemos establecer varias comunicaciones simultáneas entre PC1 y Server1? (sin usar varias IPs para el servidor)
  - *Es decir, que el PC1 utilice por ejemplo el servidor Web y el de correo, o haga varias peticiones al servidor web, y todas usen la misma IP.*
- AQUÍ ES DONDE ENTRARÍA EN JUEGO LA MULTIPLIEXACIÓN, que permite al servidor diferenciar en las diferentes peticiones, a qué aplicaciones van dirigidas-

Característica: MULTIPLEXACIÓN 2/8

- Para entender mejor la multiplexación vamos a usar el símil del correo postal.
- Podemos imaginar el servidor como una CASA.
- Y en esa CASA hay viviendo varias PERSONAS (varias aplicaciones).
- Cuando llega la información a la casa, esa información va a ser para una de esas personas.
- ¿Pero cómo sabemos a cuál de esas personas va? En la dirección postal viene la dirección de la casa, pero cuando llega ahí, hay que asignarlo a una de las personas.
- En el correo postal se añade un campo Destinatario que indica a qué persona va destinada la carta.
- En el caso del servidor, usaremos un identificador en la cabecera de la capa de transporte.

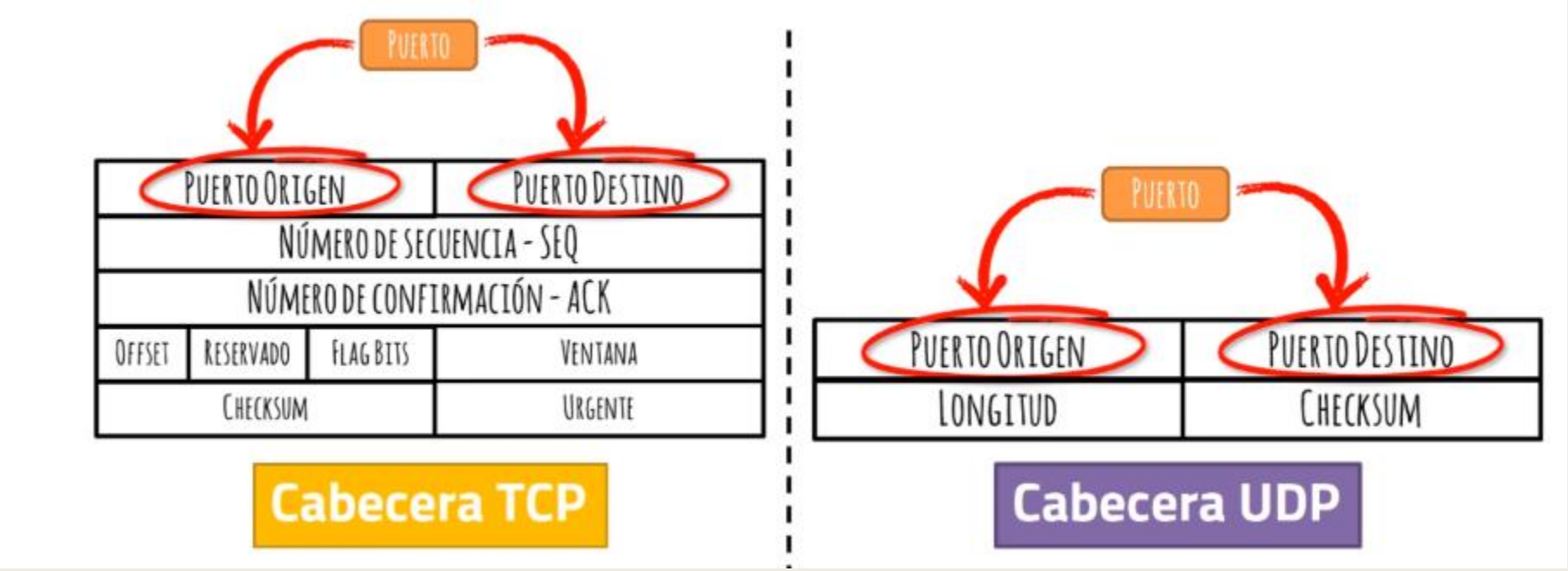
Característica: MULTIPLEXACIÓN 3/8



- PUERTO: Identificador en la cabecera de la capa de transporte
  - *Cada aplicación del servidor tendrá asignado un puerto.*
  - *Este puerto es ligeramente diferente a los puertos físicos para conectar cables.*
- CANAL: Enlace de comunicación entre dos IPs.
  - *Podemos emitir varios canales a la vez, pudiendo hacer varias comunicaciones simultáneas con el mismo o diferentes puertos.*
- Ejemplo: Cuando el servidor recibe un segmento de información identificado con el puerto 80, sabrá que va destinado al Servidor Web.

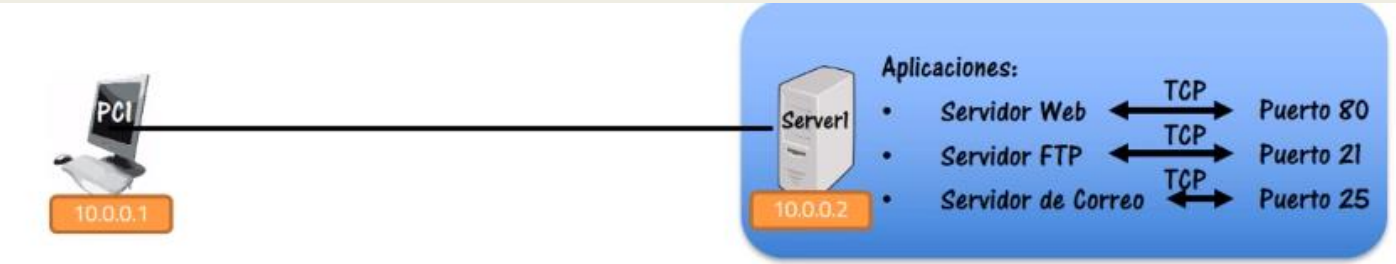
Característica: MULTIPLEXACIÓN 4/8

- En las cabeceras de TCP y UDP los campos más importantes son el PUERTO ORIGEN Y EL PUERTO DESTINO.
- TCP tiene más campos en su cabecera porque realiza más funciones que UDP.

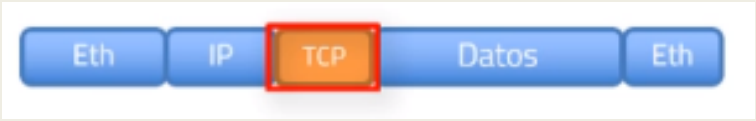


Característica: MULTIPLEXACIÓN 5/8

- Imaginemos que el PC1 quiere acceder a una página web que se encuentra en Server1



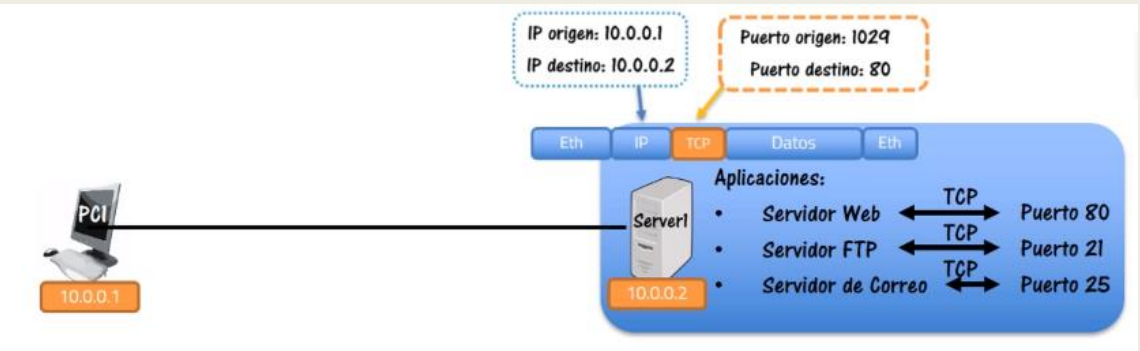
- El PC1 generará la trama ethernet con el conjunto de datos necesarios, incluyendo en los datos la cabecera TCP (que hasta ahora no le habíamos prestado atención).

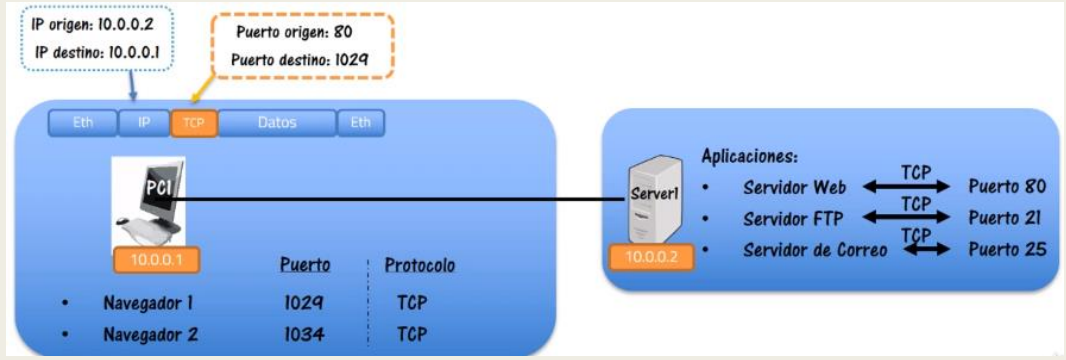


En la cabecera Ethernet irá:	En la cabecera IP irá:	En la TCP irá:	Puerto origen es un puerto aleatorio que decide el PC1
MAC Origen: MAC-PC1 MAC Destino: MAC-Server1	IP origen: 10.0.0.1 IP destino: 10.0.0.2	Puerto origen: 1029 Puerto destino: 80	

Característica: MULTIPLEXACIÓN 6/8

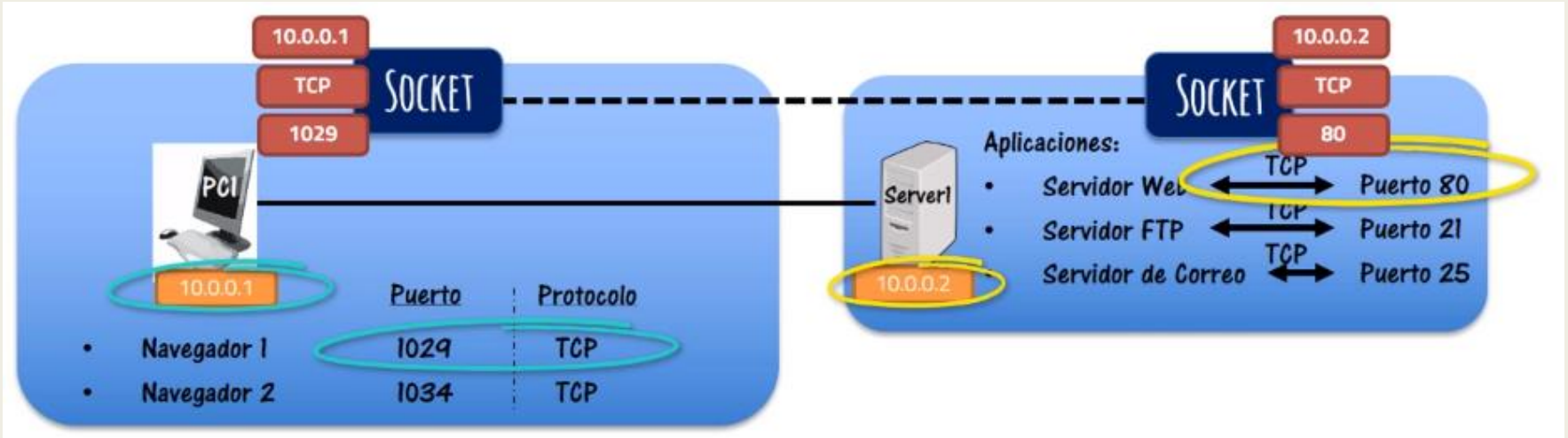
- Comunicación PC1 – Servidor:
  - *Cuando la trama de comunicación llega al Server1, se van descartando cabeceras hasta llegar a TCP, donde se comprobará el campo de puerto destino para decidir qué aplicación se quiere utilizar de server1.*
- Comunicación Servidor – PC1:
  - *Cuando la trama sea devuelta al PC1, se intercambian puertos de origen y destino en la cabecera TCP.*
  - *Es importante hacer una respuesta específica al puerto que envió la información, porque el PC1 podría tener varios navegadores abiertos (o aplicaciones) y en ese caso no se podría determinar cuál de ellos debe recibir la respuesta.*





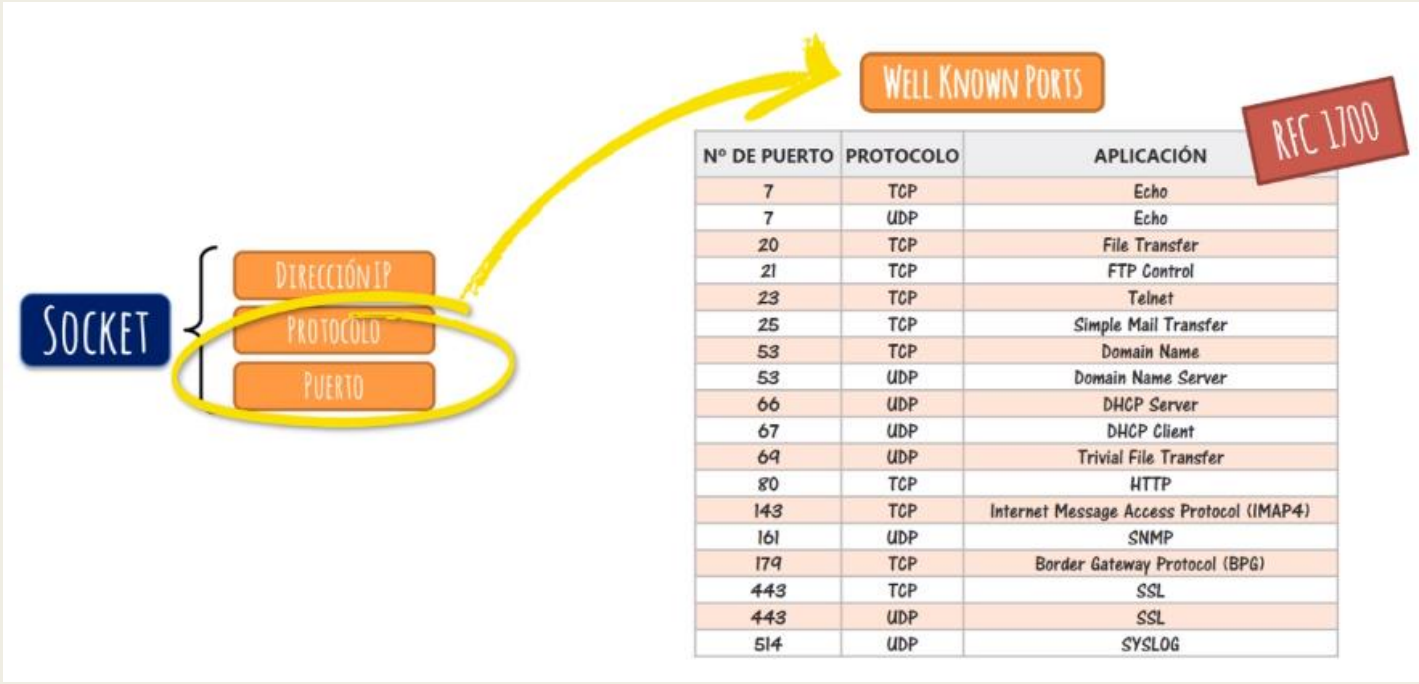
Característica: MULTIPLEXACIÓN 7/8

- SOCKET: Asociación entre una dirección IP, un protocolo de capa 4, y un puerto.
- La comunicación entre dos equipos realmente se produce de socket a socket.



Característica: MULTIPLEXACIÓN 8/8

- Los puertos a utilizar por cada aplicación están definidos en el RFC 1700.
  - *Lo que se conoce como Well Kknown Ports.*
- Están definidos para que cualquier equipo y país utilice los mismos puertos y se facilite la programación de las aplicaciones y su uso.



Nº DE PUERTO	PROTOCOLO	APLICACIÓN
7	TCP	Echo
7	UDP	Echo
20	TCP	File Transfer
21	TCP	FTP Control
23	TCP	Telnet
25	TCP	Simple Mail Transfer
53	TCP	Domain Name
53	UDP	Domain Name Server
66	UDP	DHCP Server
67	UDP	DHCP Client
69	UDP	Trivial File Transfer
80	TCP	HTTP
143	TCP	Internet Message Access Protocol (IMAP4)
161	UDP	SNMP
179	TCP	Border Gateway Protocol (BPG)
443	TCP	SSL
443	UDP	SSL
514	UDP	SYSLOG

En esta tabla hay solo algunos de ellos, pero el RFC 1700 define más.

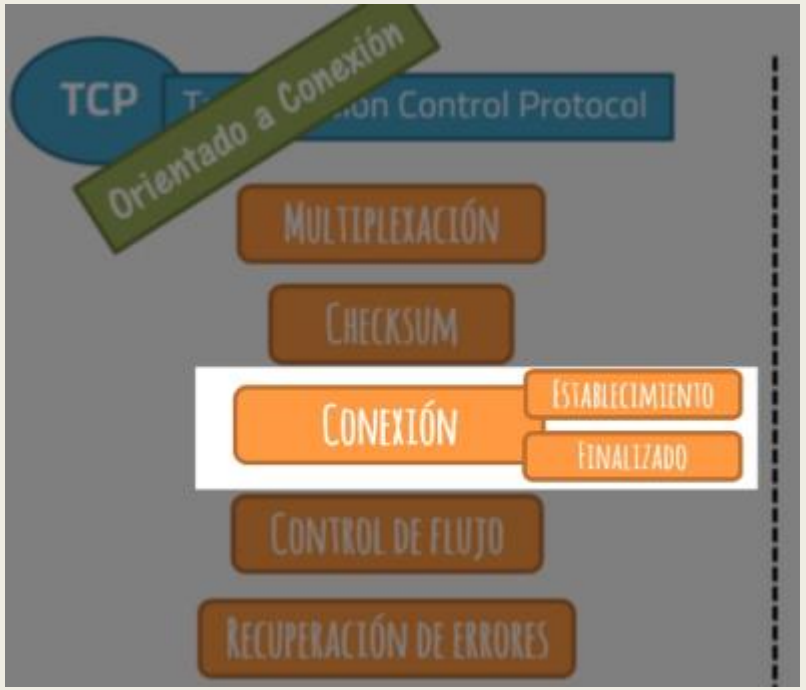
Breve inciso: Interacción entre capas

- La cabecera de cada capa contiene información sobre el protocolo que encontrará en la capa superior.
  - *La capa 2 tiene un campo TIPO que indica el protocolo de la capa de red.*
    - 0x0800 = IPv4
  - *La capa 3 tiene un campo PROTOCOLO*
    - TCP: 0x06
    - UDP: 0x11
  - *La capa 4 utiliza el campo de PUERTO DESTINO*
    - Si recibe por ejemplo el Puerto 80, sabrá que es HTTP y por tanto el servidor web

TCP: ESTABLECIMIENTO DE LA CONEXIÓN. 3-WAY-HANDSHAKE

Introducción

- En esta sección vamos a tratar la característica de conexión del protocolo TCP.
- Es la característica más importante de las que tiene únicamente TCP.
  - *Es la que permite crear las conexiones, y también las establece, las finaliza, y las gestiona.*



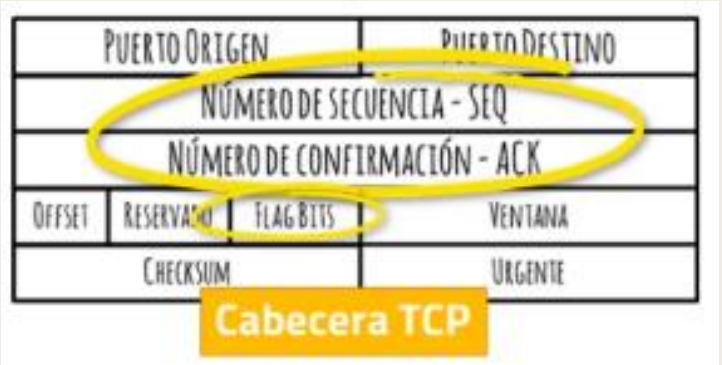
Etapas de la Conexión

- Una comunicación TCP la podemos dividir en 3 etapas.
  - *1º El establecimiento de la conexión*
  - *2º El envío de datos*
  - *3º El finalizado de la conexión*

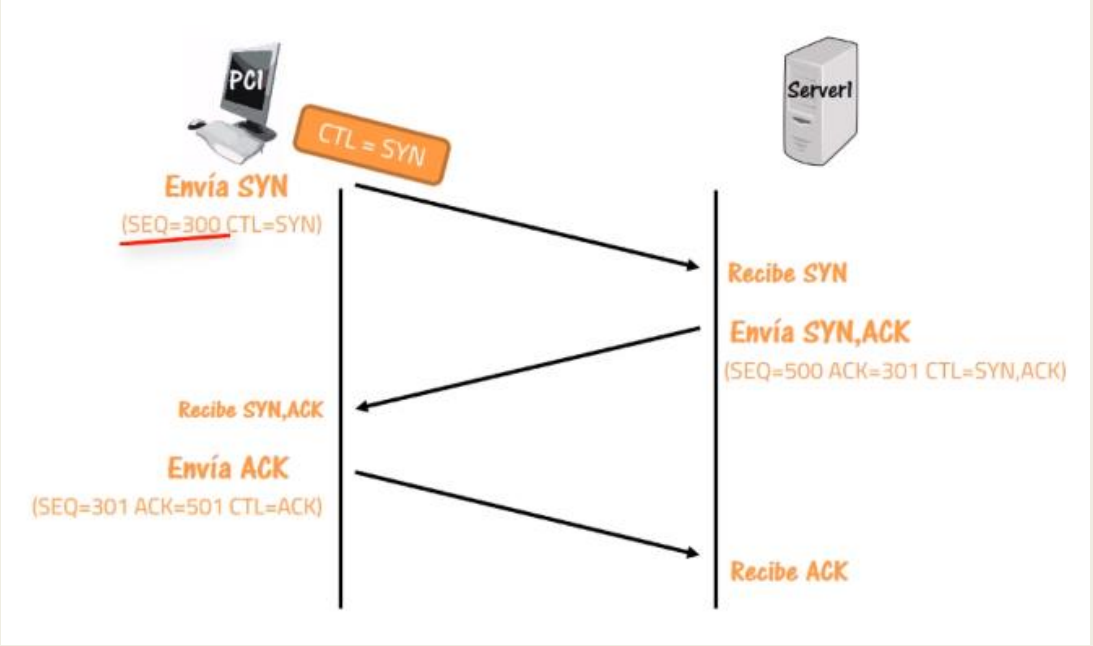
1º Establecimiento de la conexión

- La creación de la conexión se realiza mediante un procedimiento llamado **3-way-handshake**
  - *(en español: "establecimiento de conexión TCP de tres vías" o "negociación en 3 pasos").*
- El procedimiento 3-way-handshake:
  - *Es necesario antes de empezar cualquier envío de datos TCP.*
  - *Permite la sincronización entre ambos miembros.*
  - *Crea la conexión entre SOCKETS, es decir, la apertura de los puertos (que irán asociados a un protocolo y a una dirección IP)*
  - *Utiliza números de SEQ (secuencia) y de ACK*
  - *Utiliza flags de control (llamados bits de control o flag bits)*

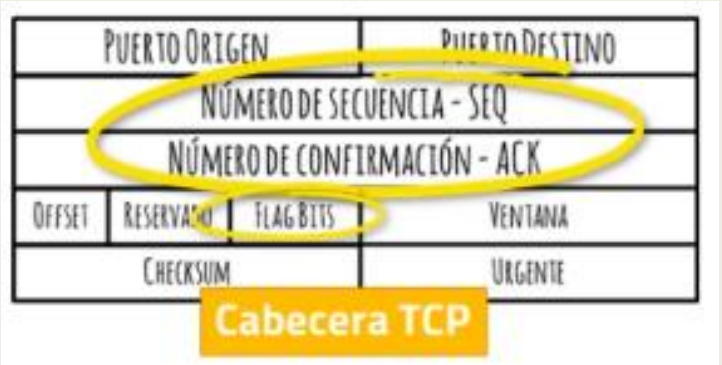
3-Way-Handshake – Paso 1



- El cliente (PC1) envía un segmento al Server1.
  - *El segmento tendrá los campos:*
    - SEQ o número de secuencia
      - *Número aleatorio entre 0 y 4.294.967.295*
    - Flag Bits o bits de control (CTL = ConTroL).
      - *En este primer mensaje tiene el valor SYN (de sincronización)*
        - Es el objetivo de la comunicación, sincronizar la comunicación entre cliente y servidor.
- El servidor recibe el segmento del cliente.

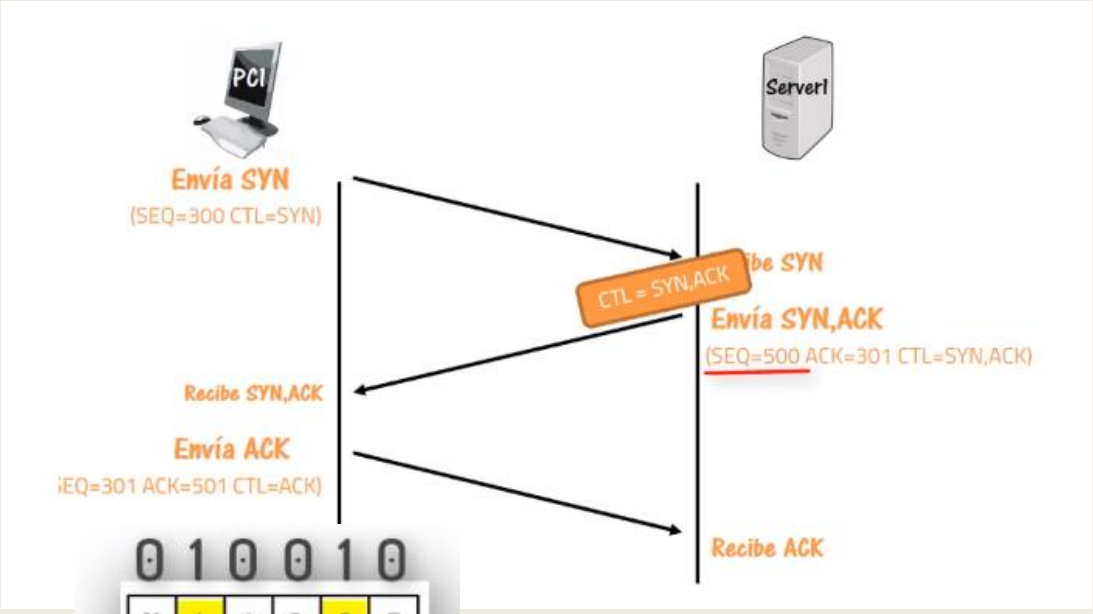


3-Way-Handshake – Paso 2



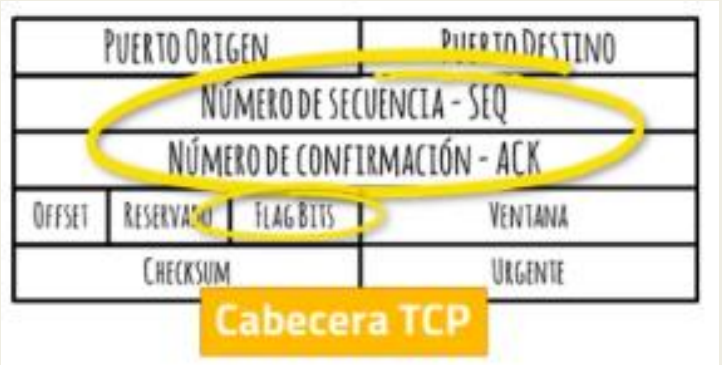
- El servidor contesta al PC1 con otro segmento.
  - *El segmento tendrá los campos:*
    - SEQ o número de secuencia
      - *Número aleatorio entre 0 y 4.294.967.295*
      - *Ojo: Es diferente al SEQ del paso 1*
    - ACK (número de confirmación)
      - *Es el SEQ que nos había enviado el cliente + el número de bytes del segmento recibido.*
      - *El primer segmento del 3-way-handshake siempre pesa 1 byte, por lo que ACK = 301*
      - *Sirve para indicarle al emisor que hemos recibido correctamente toda su información.*
    - Flag Bits.
      - *Envía SYN, ACK (quiere decir algo así como "te confirmo la sincronización").*
      - *Ojo: El campo ACK es diferente al flag bit ACK*

N° ACK = N°SEQ + Tamaño de Bytes recibidos

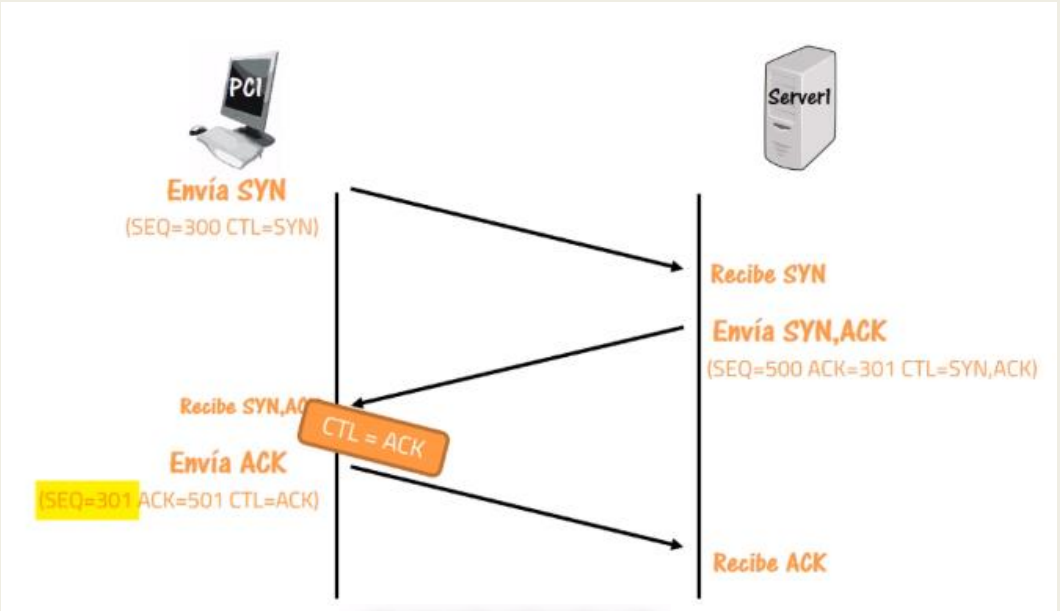


El campo de bits de control puede tener uno o varios bits a 1

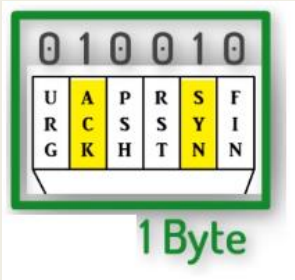
3-Way-Handshake – Paso 3



- El PC1 recibe el segmento respuesta del Server1 y envía el último al servidor.
  - *El segmento tendrá los campos:*
    - SEQ o número de secuencia
      - *Tendrá el valor 301 (porque el valor inicial del PC1 era 300, y sigue ahora el 301, luego será el 302, etc.)*
    - ACK (siguiendo el mismo procedimiento que en el paso 2)
    - Flag Bits
      - *En este caso solo tendrá a 1 el bit de ACK*
- El servidor recibe el segmento del cliente y de esta forma se habrá establecido la conexión entre el cliente y el servidor.



Recapitulando y datos importantes

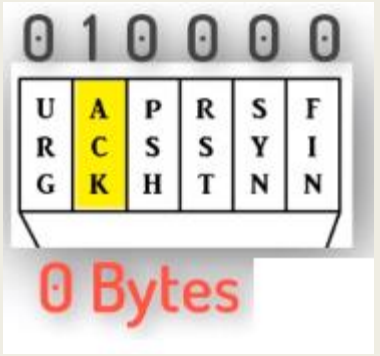


- En los tres pasos, los flag bits han variado de la siguiente forma:
  - *Paso 1 – SYN*
  - *Paso 2 – SYN, ACK*
  - *Paso 3 – ACK*
  - *A la hora de establecer la conexión, los bits de control SIEMPRE tendrán esos valores.*
- Para calcular el Nº de ACK usamos la fórmula:

  N° ACK = N°SEQ + Tamaño de Bytes recibidos

  - *Detalle importante: En el PASO 1, PASO 2, y PASO 3, SE ENVÍAN 0 BYTES!!*
  - *Pero los estándares de TCP/IP decidieron que era necesario confirmar la sincronización, y que si los flag bits de SYN o FIN están a 1, el segmento ocupará 1 BYTE.*
    - A ese byte se le conoce como “PHANTOM BYTE” porque realmente no existe.

¿Y si solo está el bit de ACK en 1?

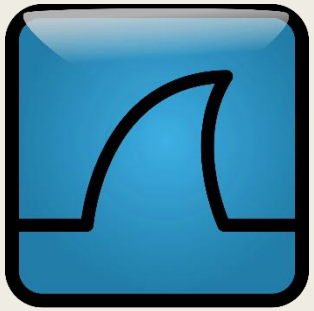


- Si no está activado ni el bit de SYN ni el bit de FIN se considerará que el tamaño es de 0 bytes.
- **¿Y por qué le damos 1 byte al paso 1 y 2 pero no se lo damos al paso 3?**
- Porque tanto el mensaje del paso 1 como el mensaje del paso 2 necesitan de una confirmación posterior, así lo requiere el protocolo y por eso ambos tienen activados el bit de SYN.
- Pero el mensaje del paso 3 que se envía para la confirmación final ya no espera recibir luego otro ACK. Como ya no requiere de una confirmación, no se necesitan calcular ACKs, y por tanto, no es necesario considerar que pesa 1 byte.

Puede ser un poco complejo de entender al principio, pero a medida que lo veamos, puede ir quedando más claro.

TCP: ANÁLISIS DEL 3-WAY-HANDSHAKE CON WIRESHARK

Introducción



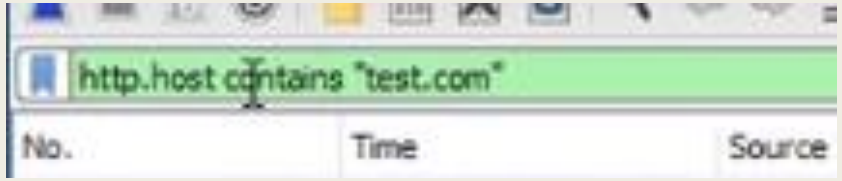
- Vamos a comprobar a través de Wireshark que efectivamente se realiza el 3-way-handshake.
- Podemos hacerlo cada uno en nuestro ordenador para poder identificar el tráfico y lo tengamos más claro.

- Abriremos una página web en nuestro navegador y analizaremos el tráfico.

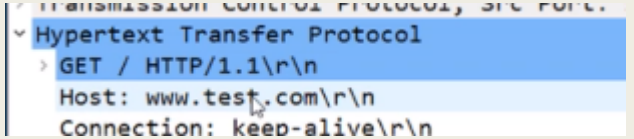
Captura del tráfico

- 1º Abrimos Wireshark
- 2º Elegimos nuestra interfaz para capturar tráfico
  - *Y vemos que empieza a capturar tráfico*
- 3º Abrimos el navegador web
- 4º Escribimos alguna web en el navegador, por ejemplo www.test.com
  - *La abrimos y esperamos a que cargue la página*
- 5º Detenemos la captura de tráfico
  - *Lo que queríamos capturar ya lo tenemos (el 3-way-handshake)*

Filtrar el tráfico



- Veremos que tenemos infinidad de paquetes capturados
- Vamos a usar un filtro de visualización para buscar la información que nos interesa:
  - *http.host contains "test.com"*
    - Vamos a buscar en la cabecera HTTP, en su campo host, aquellos paquetes que contengan "test.com"
- El resultado obtenido es el paquete que viene justo después del 3-way-handshake



- Vamos a hacer lo siguiente con el paquete obtenido:
  - *Clic derecho sobre él → Conversation filter → TCP*
    - Esto nos va a generar un filtro en base a la conversación TCP de ese paquete, es decir, nos mostrará todos los paquetes que han intervenido en ella.

Si no os carga esa, probad con otras páginas como rae.es

Analizar el tráfico

- Vemos que el filtro de visualización ha cambiado.
  - *Ahora tenemos un filtro bastante específico, que nos muestra únicamente la conexión implicada en esta comunicación.*
    - Normalmente se establecen varios intentos de comunicación de manera simultánea, para cargar a la vez imágenes, vídeos, etc. Y así la carga web sea más rápida. Pero en nuestro caso, todo eso nos entorpecería porque solo queremos analizar el tráfico del 3-way-handhsake.
- A continuación vamos a ir analizando los diferentes mensajes que hemos visto en la teoría.

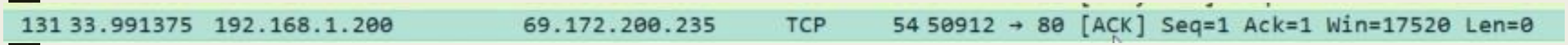
Mensaje de sincronización (Paso 1 3-way-hanshake)

- Lo envía nuestro PC con destino la IP del servidor.
- Vemos que la longitud es 66 (es longitud de la trama, no de los datos).
- El puerto origen es 50912 y el destino el puerto 80
- Como flag bits tenemos solamente SYN
- Como número de SEQ se ha establecido el 0 (mentirijilla de Wireshark)
  - *Wireshark nos muestra el número de SEQ como si fuera 0 en el inicio para que sea más fácil de entender, así que lo dejaremos así.*
  - *Si queréis ver la realidad, vamos a Edit → Preferences → Protocols → TCP → Desmarcamos la casilla Relative sequence numbers* Seq=246510072
- Y vemos que la longitud de los datos es 0 (Len=0)

Respuesta del servidor (Paso 2 de 3-way-handshake)

- Lo envía el servidor con destino la IP de nuestro PC.
- El puerto origen es 80 y el destino el puerto 50912
- Como flag bits tenemos SYN, ACK
- Como número de SEQ se ha establecido el 0
- Como número de ACK se ha establecido el 1
  - *Está confirmando que ha recibido bien el primer segmento*
- Y vemos que la longitud de los datos es 0 (Len=0)

Última respuesta cliente para crear la conexión (Paso 3 de 3-way-handshake)

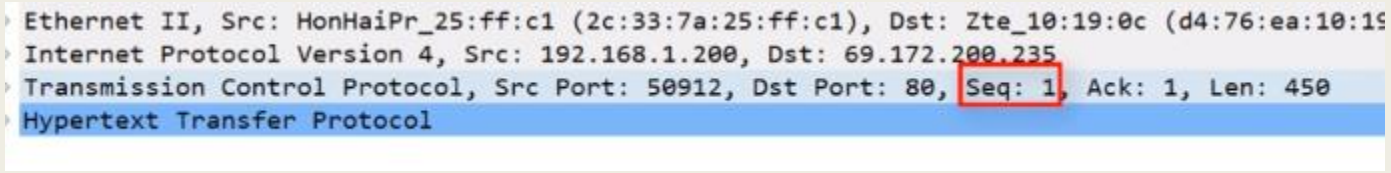


- Lo envía nuestro PC con destino la IP del servidor.
- El puerto origen es 50912 y el destino el puerto 80
- Como flag bits tenemos solamente ACK
- Como número de SEQ se ha establecido el 1
- Como número de ACK se ha establecido el 1
  - *Está confirmando que ha recibido bien el anterior segmento*
- Y vemos que la longitud de los datos es 0 (Len=0)

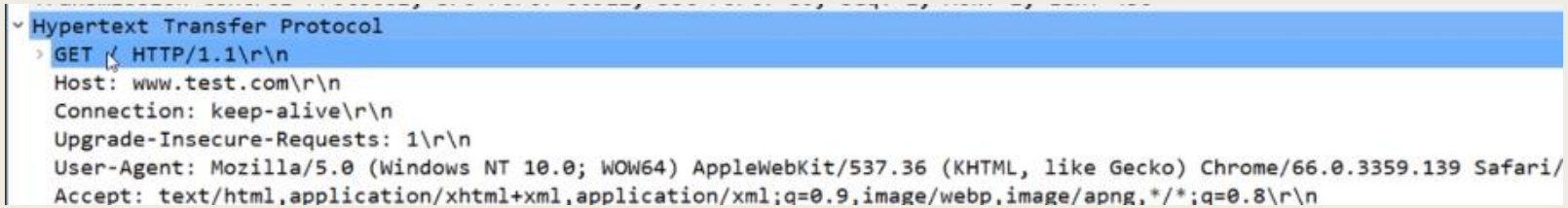
La comunicación

```
133 33.991977  192.168.1.200        69.172.200.235      HTTP    504 GET / HTTP/1.1
```

- Después de establecerse la conexión con el 3-way-handshake viene la comunicación.
- Es una petición HTTP en la cual nuestro PC le pide la página web al servidor



- Si nos fijamos, el número de SEQ vuelve a ser 1, igual que en el mensaje anterior. ¿Por qué?
  - *Porque no estaba activado ni el bit de SYN ni el bit de FIN*
- La longitud en este caso es 450 bytes, y son los datos que contiene esa cabecera de capa 4. Y son los contenidos de la página web de esa petición.



Pregunta

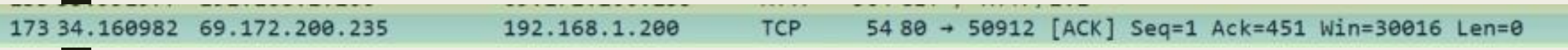
- La respuesta del servidor tiene Nº de ACK, ¿cuál será el número de ACK si la longitud del mensaje era 450 bytes y partíamos de SEQ=1?

Pregunta - Solución

- Para calcular el número de ACK aplicamos la fórmula que ya conocemos:

  Nº ACK = NºSEQ + Tamaño (Bytes)

- ACK = 1 + 450 = 451 bytes



- Esta característica en inglés recibe el nombre de Reliable Transmission, en español, envío confiable.
- Podemos compararlo con correos y el envío de las cartas certificadas, donde tenemos la seguridad de que la carta llegará a su destino y nos lo van a confirmar.

Siguientes paquetes

```
174 34.160982  69.172.200.235   192.168.1.200    HTTP   489 HTTP/1.1 302 Moved Temporarily  (text/html)
182 34.201415  192.168.1.200    69.172.200.235   TCP     54 50912 → 80 [ACK] Seq=451 Ack=436 Win=17085 Len=0
```

- El siguiente paquete es debido a que la página web es de tipo HTTPS
- Es un mensaje de redirección a otra página diferente:

```
Keep-Alive: timeout=20\r\n
Location: https://www.test.com/\r\n
X-DIS-Request-ID: d037baebe808d18ec014d30b
```

- Esto iniciará una conexión a otra página, que aquí ya no aparece porque estamos aplicando un filtro muy específico.
  - *Del mismo modo que no vemos los paquetes de imágenes y otros.*
- Vemos que la longitud de estos datos es de 435 bytes y el número de SEQ es 1

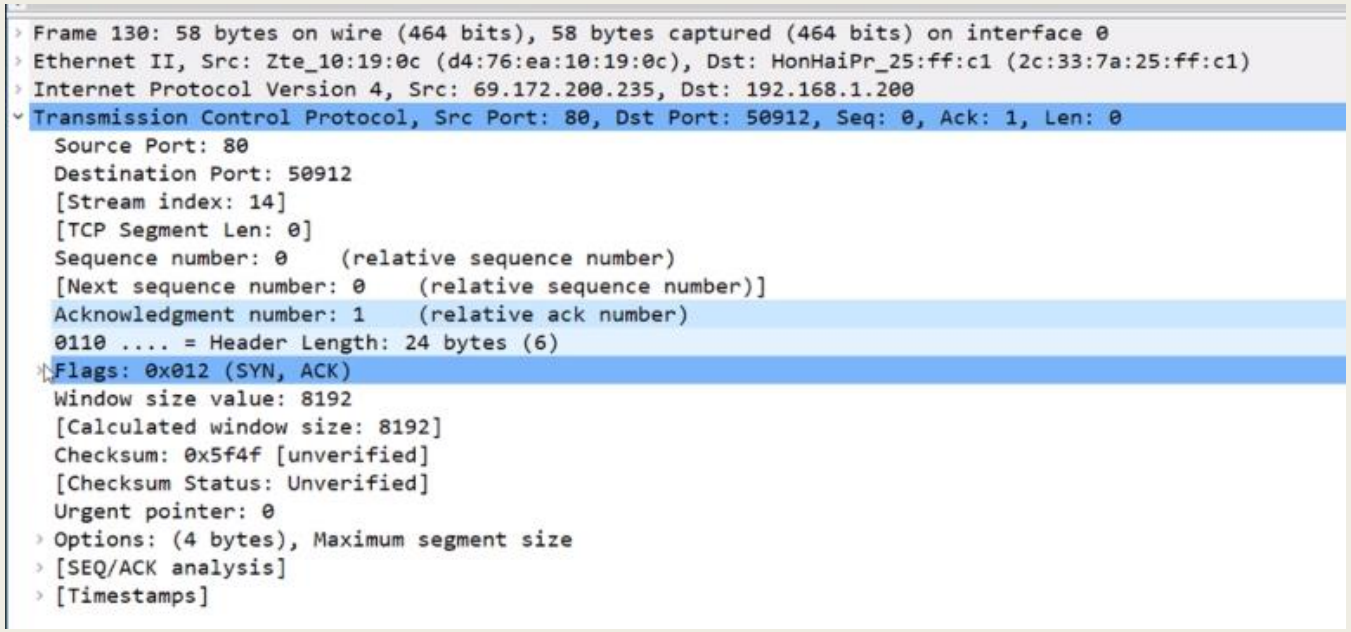
```
Transmission Control Protocol, Src Port: 80, Dst Port: 50912, Seq: 1, Ack: 451, Len: 435
```

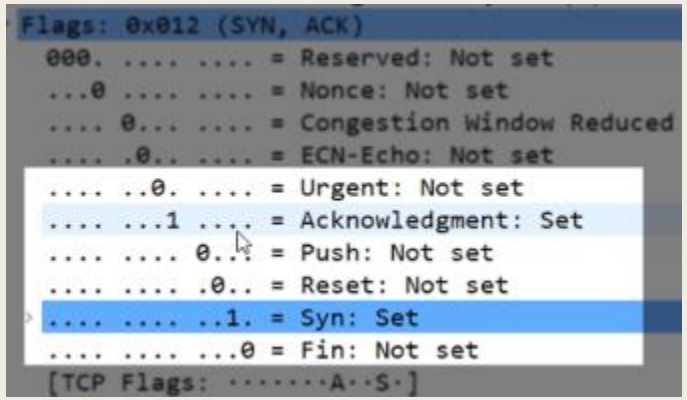
- De modo que en el siguiente segmento, el ACK será de 436

```
[ACK] Seq=451 Ack=436 W
```

Por último: Análisis de las cabeceras

- Si pulsamos sobre el segundo segmento para ver sus cabeceras (hasta ahora estábamos mirando un resumen), apreciamos lo siguiente en TCP:





- Si nos enfocamos en los flags, y los desglosamos, podemos apreciar cómo los bits activados están a 1, y el resto a 0.

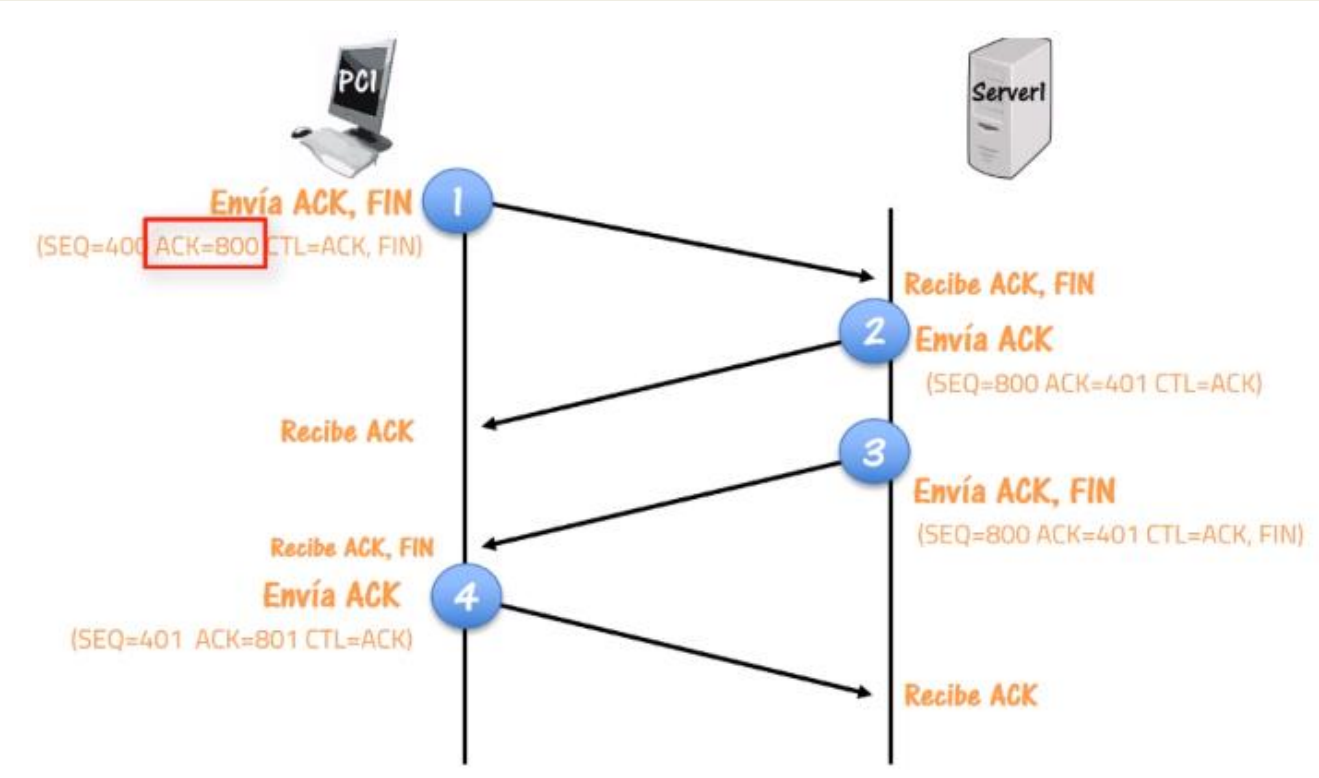
TCP: FINALIZADO DE LA CONEXIÓN Y ENVÍO DE DATOS + WIRESHARK

Introducción

- En esta sección vamos a tratar el finalizado de la conexión.
- Seguimos utilizando la misma cabecera TCP.
- La finalización de la conexión se negocia en **4 PASOS.**
- Se puede terminar desde un lado o del otro de forma independiente (desde el cliente y desde el servidor).
- Permite liberar el socket asociado en cada extremo.

Finalizado de la conexión. Paso 1.

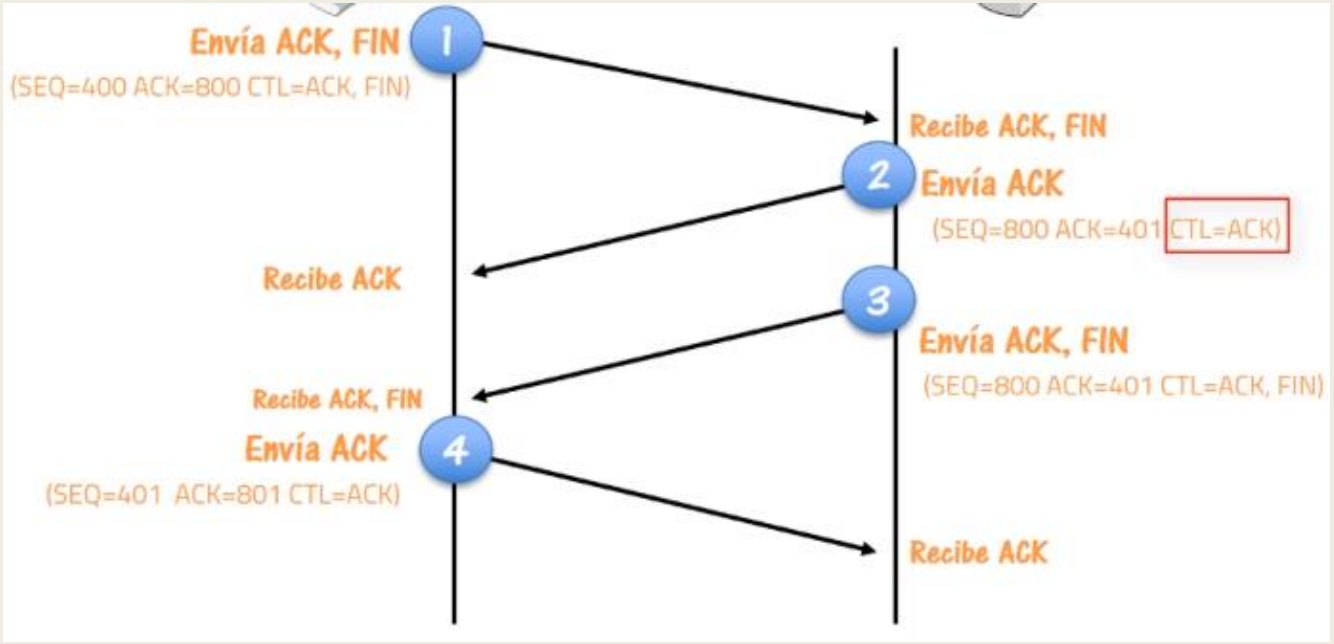
- Hemos comenzado con PC1, pero como hemos indicado, podría haberla finalizado el servidor.
- Como SEQ tendrá el número que le corresponda en ese momento (sabemos que empieza con el 3-way-handshake y luego se envían datos, así que será el que siga después).
- Tenemos ACK=800 (ya sabemos cómo se calcula)
- Como bits de control tenemos a 1 el ACK y el FIN



Hay algunos libros que pueden no incluir el ACK en paso 1 y 3 para simplificar, pero la realidad es esta y sí están

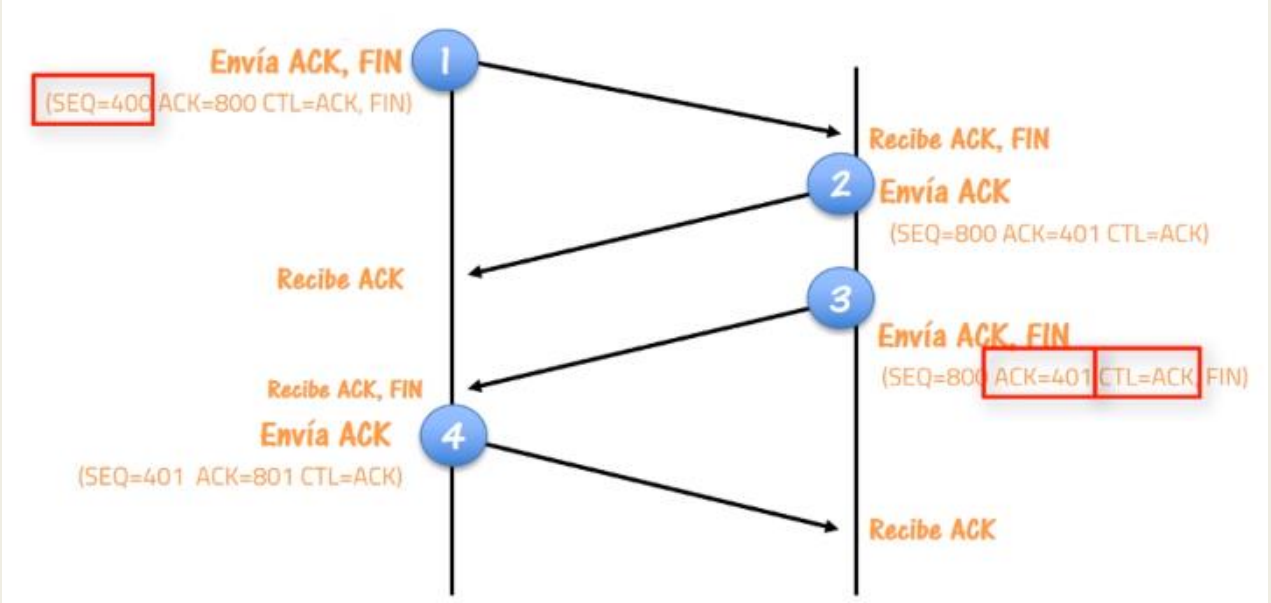
Finalizado de la conexión. Paso 2

- El servidor recibirá el mensaje de finalizado y responderá con un segmento con la siguiente información:
- SEQ=800 (ocurre lo mismo que ocurría con el SEQ del PC1)
- ACK=401 (recordatorio: si el flag bit de FIN está a 1, el segmento se tomará como que pesa 1 byte).
- Como flag bit se marca a 1 el de ACK
- Ese mensaje de respuesta confirmará el cierre para el PC1.



Finalizado de la conexión. Paso 3

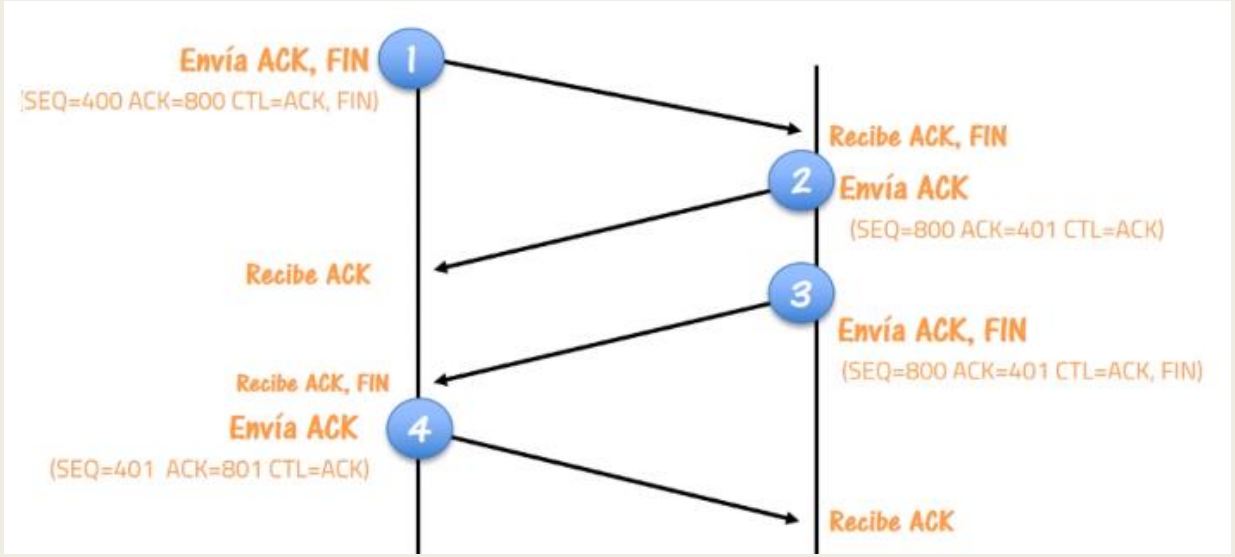
- Tras lo anterior, AÚN NO SE HABRÁ CERRADO LA SESIÓN.
- Lo que sucederá es que el servidor realizará el mismo proceso que el PC1 para cerrar también.
- El servidor enviará un mensaje con bits de control ACK y FIN.
- Número de SEQ el que corresponda (es 800 porque el mensaje anterior no tenía ni el flag de SYN ni el de FIN a 1).
- El número de ACK sigue siendo 401 (por el mismo motivo que el SEQ es 800)



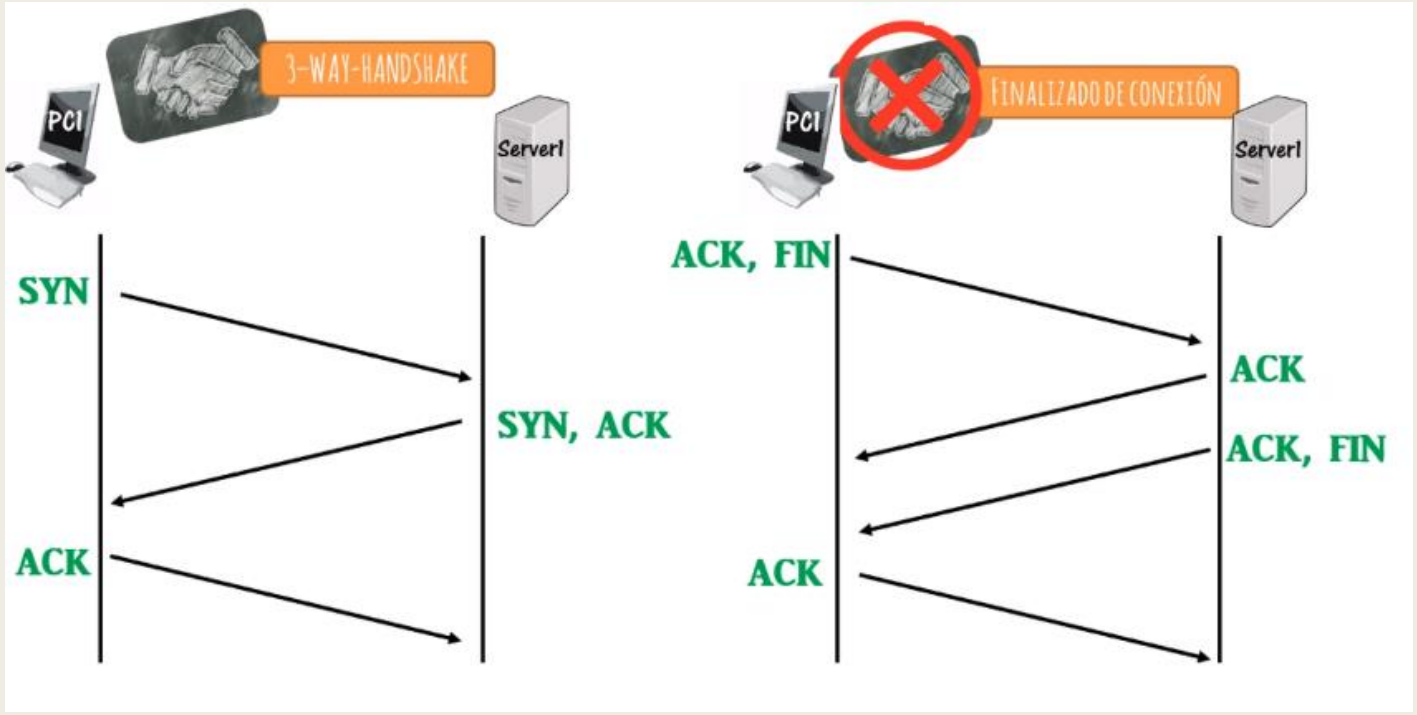
Puede parecer algo redundante volver a confirmar el mismo ACK=401, pero así es como funciona

Finalizado de la conexión. Paso 4

- El cliente recibe el mensaje de FIN del servidor.
- Y confirma con un ACK que lo ha recibido.
- El SEQ será la continuación del anterior, que será 401.
- El ACK será 801 confirmando que ha recibido este byte por parte del servidor.
- Y así, al completarse este proceso, tanto el cliente como el servidor, podrán cerrar la sesión y dar por terminada la sesión.



Resumen de 3-way-handshake y de finalizado de la conexión

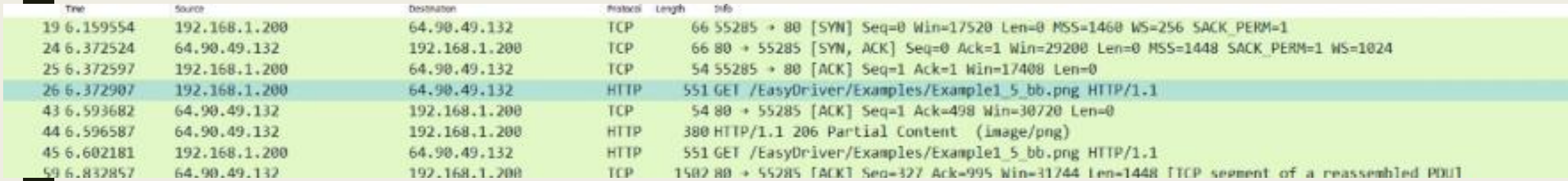


- Al final, lo que permiten los SEQ y ACK es para proporcionar un envío de datos confiable y permitir la recuperación de errores. Si os fijáis, lo único que hacemos es que cada vez que enviamos algo, esperamos que el otro extremo nos confirme que lo ha recibido y la cantidad de lo que ha recibido.

Analizando el finalizado en Wireshark

- En el material de clase tenéis un archivo de tráfico capturado con wireshark
- Se ha capturado tráfico en una página con protocolo HTTP (en lugar de HTTPS como en el anterior ejemplo).
- Vamos a centrarnos en:
  - *Envío / recepción de datos*
  - *Finalizado de la conexión.*
- Vamos al paquete 19
- Clic derecho → Conversation filter → TCP

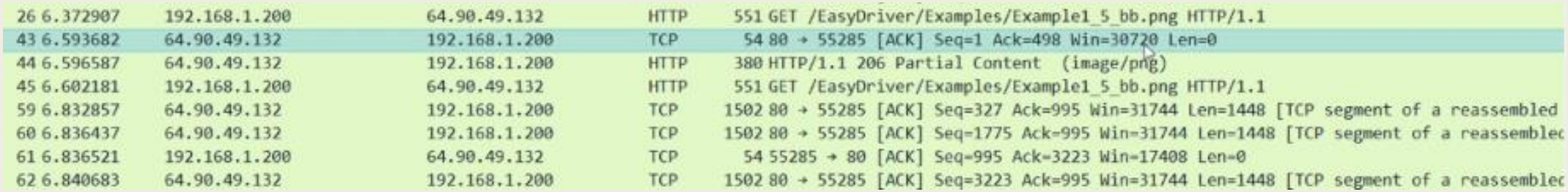
3-way-handshake por encima



	Time	Source	Destination	Protocol	Length	Info
19	6.159554	192.168.1.200	64.90.49.132	TCP	66	55285 → 80 [SYN] Seq=0 Win=17520 Len=0 MSS=1460 WS=256 SACK_PERM=1
24	6.372524	64.90.49.132	192.168.1.200	TCP	66	80 → 55285 [SYN, ACK] Seq=0 Ack=1 Win=29200 Len=0 MSS=1448 SACK_PERM=1 WS=1024
25	6.372597	192.168.1.200	64.90.49.132	TCP	54	55285 → 80 [ACK] Seq=1 Ack=1 Win=17408 Len=0
26	6.372907	192.168.1.200	64.90.49.132	HTTP	551	GET /EasyDriver/Examples/Example1_5_bb.png HTTP/1.1
43	6.593682	64.90.49.132	192.168.1.200	TCP	54	80 → 55285 [ACK] Seq=1 Ack=498 Win=30720 Len=0
44	6.596587	64.90.49.132	192.168.1.200	HTTP	380	HTTP/1.1 206 Partial Content (image/png)
45	6.602181	192.168.1.200	64.90.49.132	HTTP	551	GET /EasyDriver/Examples/Example1_5_bb.png HTTP/1.1
59	6.832857	64.90.49.132	192.168.1.200	TCP	1502	80 → 55285 [ACK] Seq=327 Ack=995 Win=31744 Len=1448 [TCP segment of a reassembled PDU]

- Los primeros 3 segmentos vemos que se corresponden con el 3-way-handshake.
- Comprobamos que los bits de control son tal como en el anterior ejemplo.
- Tras esto, se realiza la petición de contenido web a la página, ya que en este caso ya no nos redirige a otra web con https
- Como detalle a destacar, los envíos tienen todos ACK para confirmar la información.

Envío / recepción de datos



26	6.372907	192.168.1.200	64.90.49.132	HTTP	551 GET /EasyDriver/Examples/Example1_5_bb.png HTTP/1.1
43	6.593682	64.90.49.132	192.168.1.200	TCP	54 80 → 55285 [ACK] Seq=1 Ack=498 Win=30720 Len=0
44	6.596587	64.90.49.132	192.168.1.200	HTTP	380 HTTP/1.1 206 Partial Content (image/png)
45	6.602181	192.168.1.200	64.90.49.132	HTTP	551 GET /EasyDriver/Examples/Example1_5_bb.png HTTP/1.1
59	6.832857	64.90.49.132	192.168.1.200	TCP	1502 80 → 55285 [ACK] Seq=327 Ack=995 Win=31744 Len=1448 [TCP segment of a reassembled
60	6.836437	64.90.49.132	192.168.1.200	TCP	1502 80 → 55285 [ACK] Seq=1775 Ack=995 Win=31744 Len=1448 [TCP segment of a reassemblec
61	6.836521	192.168.1.200	64.90.49.132	TCP	54 55285 → 80 [ACK] Seq=995 Ack=3223 Win=17408 Len=0
62	6.840683	64.90.49.132	192.168.1.200	TCP	1502 80 → 55285 [ACK] Seq=3223 Ack=995 Win=31744 Len=1448 [TCP segment of a reassemblec

- En este ejemplo, lo que vemos es que nuestro PC solicita una imagen (primer paquete).
- Y el servidor le responde confirmando que ha recibido su petición (y aún no envía nada porque len=0).
- En el tercer segmento el servidor indica que va a empezar a enviar la imagen.
- Y el cliente confirma que está listo para recibir la imagen.
- **A partir de aquí, ya comienza el envío de la imagen.**
- El servidor va enviando paquetes, y el cliente, al recibirlos (pueden ser varios a la vez), envía una respuesta con el ACK para indicar al servidor la cantidad recibida y que este sepa si puede seguir o no. Así será el proceso hasta finalizar el envío.

Pregunta

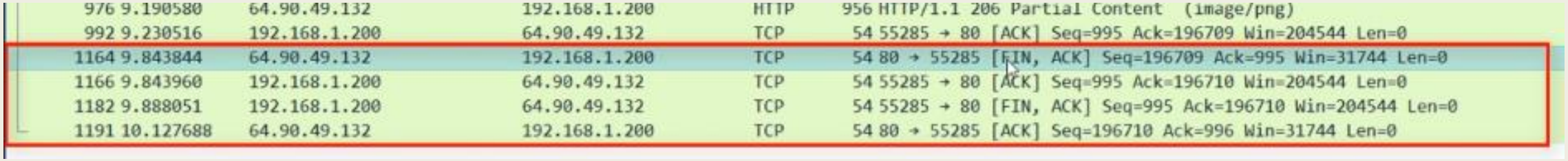
- ¿Qué habría pasado si el cliente hubiera respondido al servidor con un ACK inferior al que corresponde?

Respuesta a la pregunta

- ¿Qué habría pasado si el cliente hubiera respondido al servidor con un ACK inferior al que corresponde?
- El servidor habría comprobado el valor recibido, y volvería a enviar aquellos paquetes necesarios para que la suma del ACK sea la correcta.

Finalizado de la conexión

- Si nos desplazamos con Wireshark hasta los últimos segmentos recibidos, podemos ver los correspondientes al finalizado de la sesión.



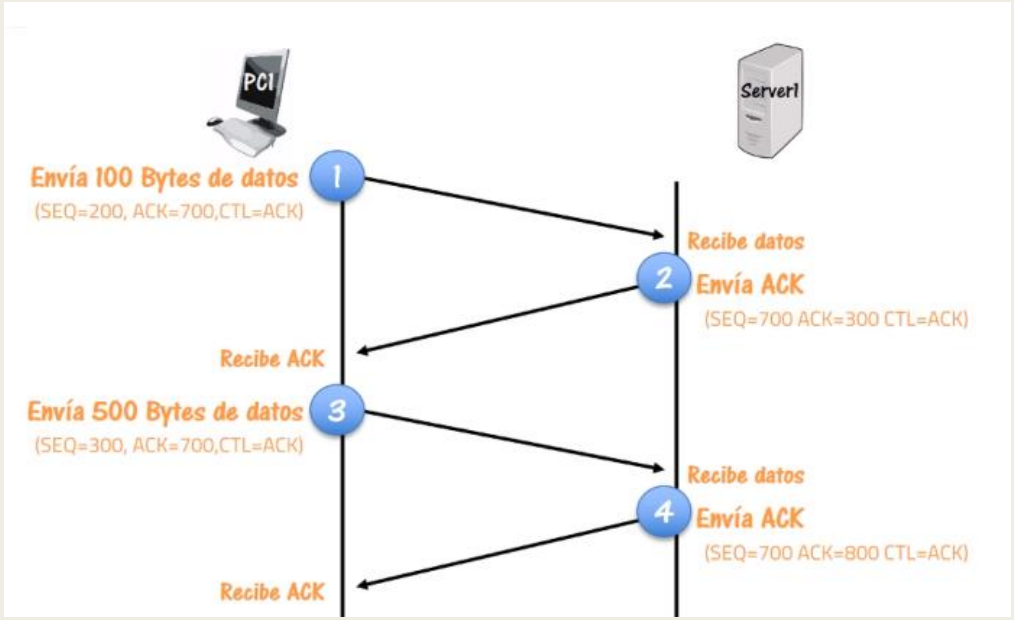
976	9.190580	64.90.49.132	192.168.1.200	HTTP	956 HTTP/1.1 206 Partial Content (image/png)
992	9.230516	192.168.1.200	64.90.49.132	TCP	54 55285 → 80 [ACK] Seq=995 Ack=196709 Win=204544 Len=0
1164	9.843844	64.90.49.132	192.168.1.200	TCP	54 80 → 55285 [FIN, ACK] Seq=196709 Ack=995 Win=31744 Len=0
1166	9.843960	192.168.1.200	64.90.49.132	TCP	54 55285 → 80 [ACK] Seq=995 Ack=196710 Win=204544 Len=0
1182	9.888051	192.168.1.200	64.90.49.132	TCP	54 55285 → 80 [FIN, ACK] Seq=995 Ack=196710 Win=204544 Len=0
1191	10.127688	64.90.49.132	192.168.1.200	TCP	54 80 → 55285 [ACK] Seq=196710 Ack=996 Win=31744 Len=0

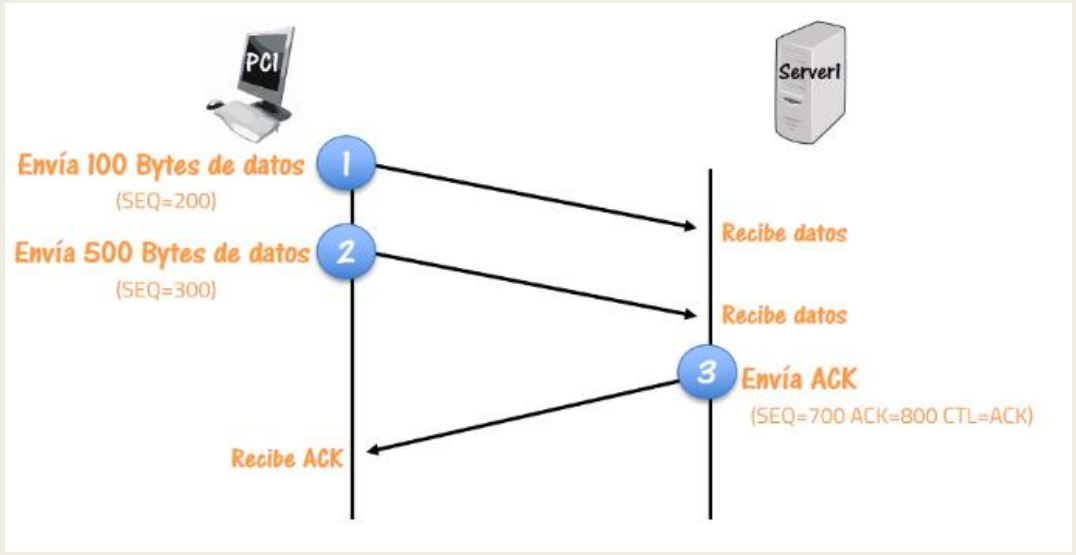
- En este finalizado de la conexión podemos ver los 4 pasos de la teoría.
- Podemos apreciar que los SEQ han aumentado bastante.
- Tenemos las dos partes del finalizado, en la primera el servidor envía el segmento para finalizar la conexión y el cliente responde.
- En la segunda parte, el cliente hace lo mismo que el servidor, enviando un segmento para indicar el finalizado de la conexión y el servidor le responde.

TCP: CONTROL DE FLUJO, RECUPERACIÓN DE ERRORES, ETC.

Recapitulando envío y recepción de datos

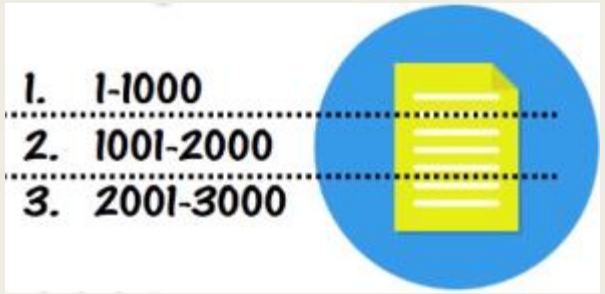
- Hemos visto que el envío de datos se realizaba de la siguiente forma:
  - *Donde cada segmento que se envía, recibe una contestación ACK*
  - *También hemos visto que en la práctica se envían varios segmentos (2, 3, 4…) y después se confirman todos juntos con el ACK*

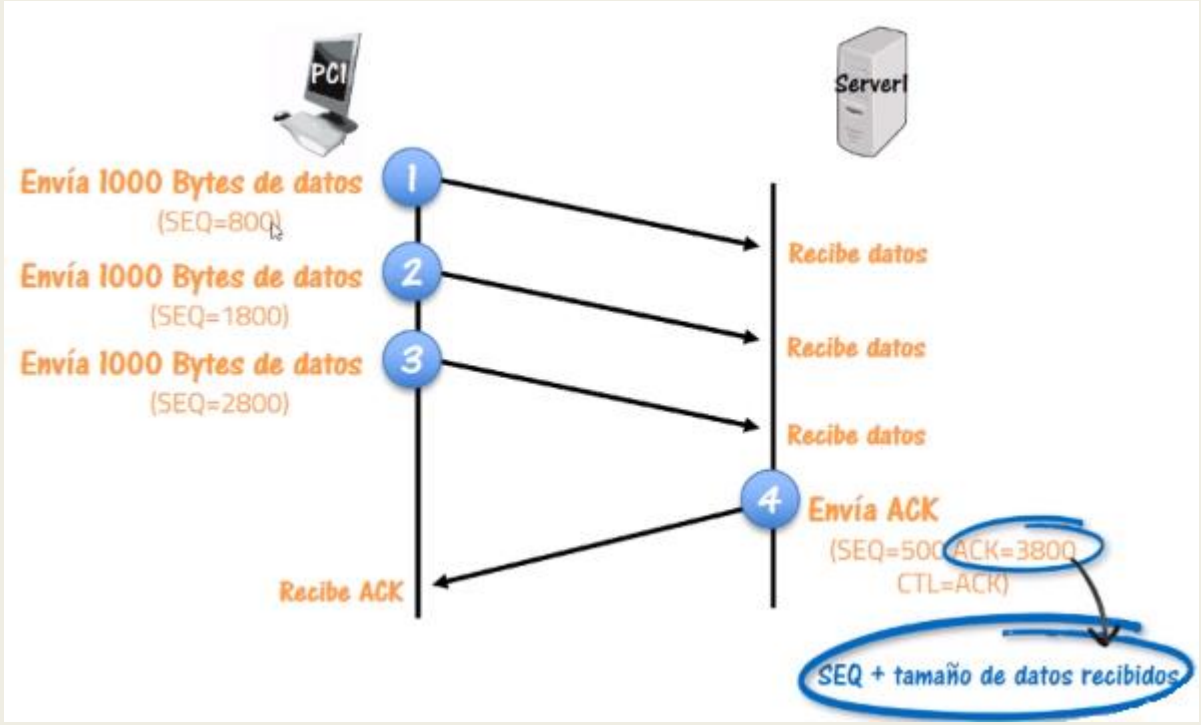




Ejemplo de envío y recepción

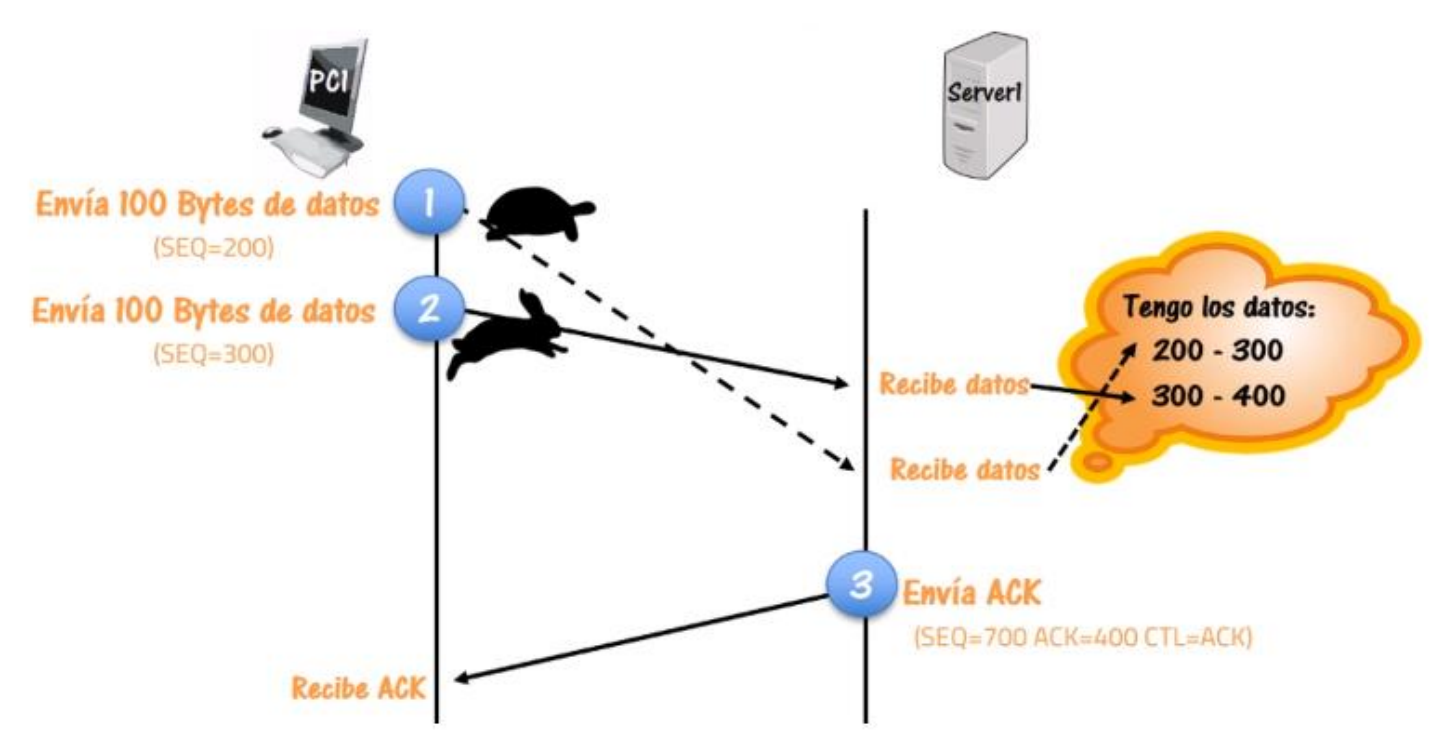
- Si tenemos que enviar un fichero de 3 Kbytes, el equipo puede decidir partirlo por ejemplo en 3 partes de 1 KB





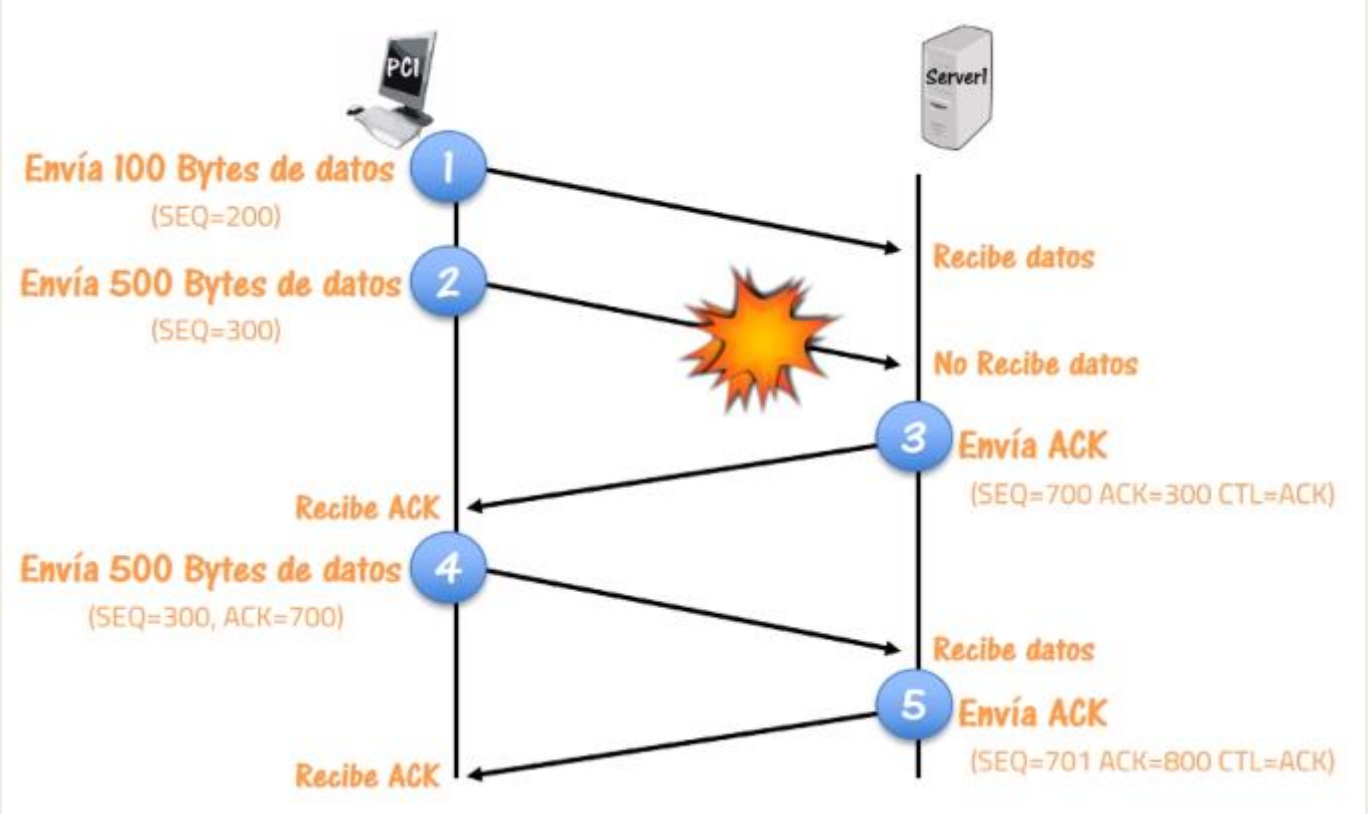
Orden adecuado de los datos

- Para poder determinar el orden de los datos que se envían, se hace automáticamente gracias a los números de secuencia (SEQ)



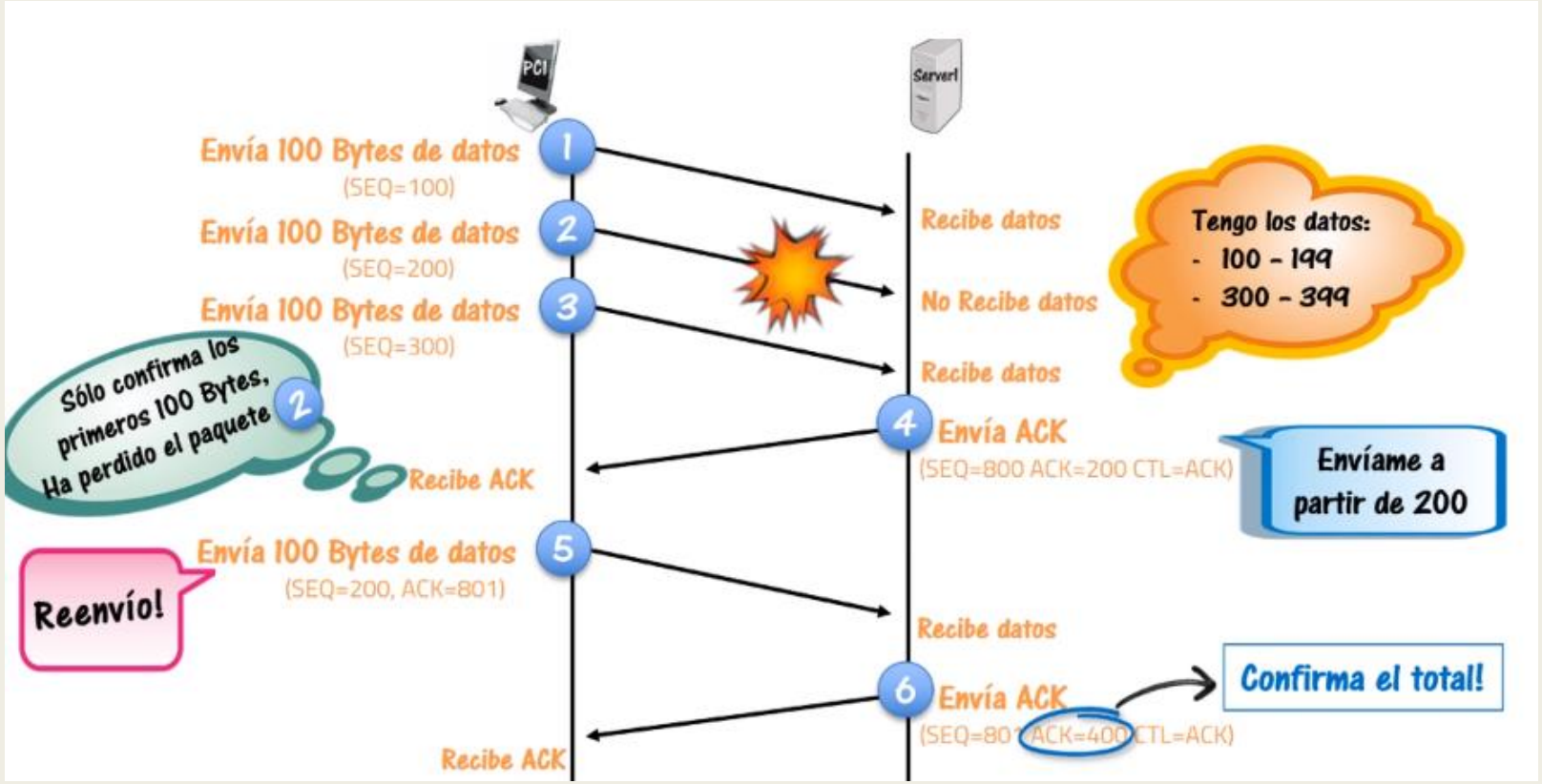
Recuperación de errores

- Ya lo hemos mencionado en la anterior sección.
- Si se envían varios segmentos y uno de ellos se pierde, a través del ACK que envía Server1, el PC1 sabrá que no se están confirmando todos los datos.
- Así que reenviará los datos que faltan a Server1.



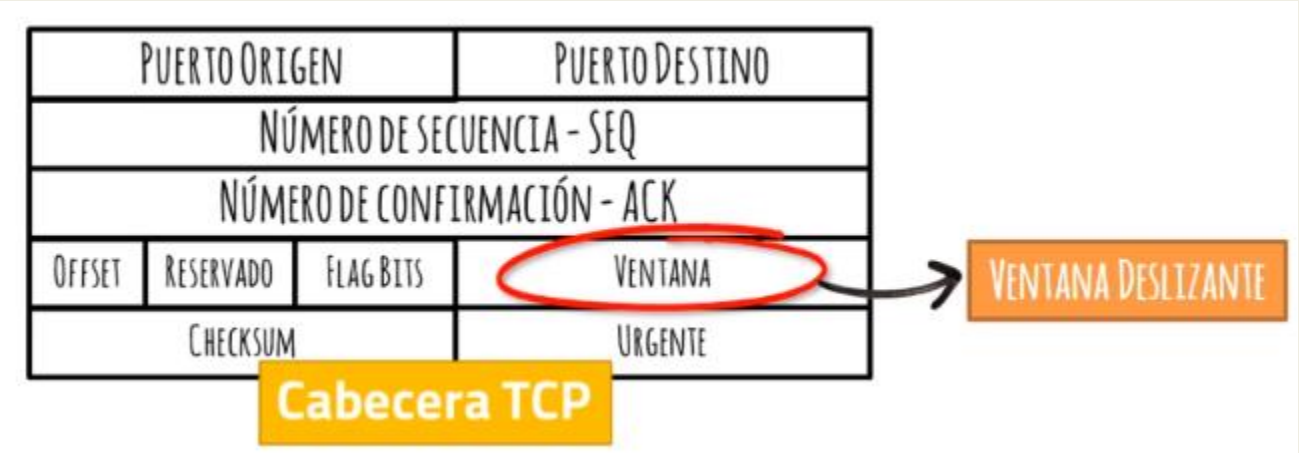
Recuperación de errores - Avanzado

- Podría darse el caso que se envíen varios paquetes y se pierda alguno intermedio.
- En este ejemplo, el server1 indicará en su ACK de confirmación que solo ha recibido hasta 200, y no el resto (porque verá que las SEQ no cuadran).
- Según la implementación y sistema operativo, será necesario reenviar solo el segmento 2, o reenviar tanto el 2 como el 3.



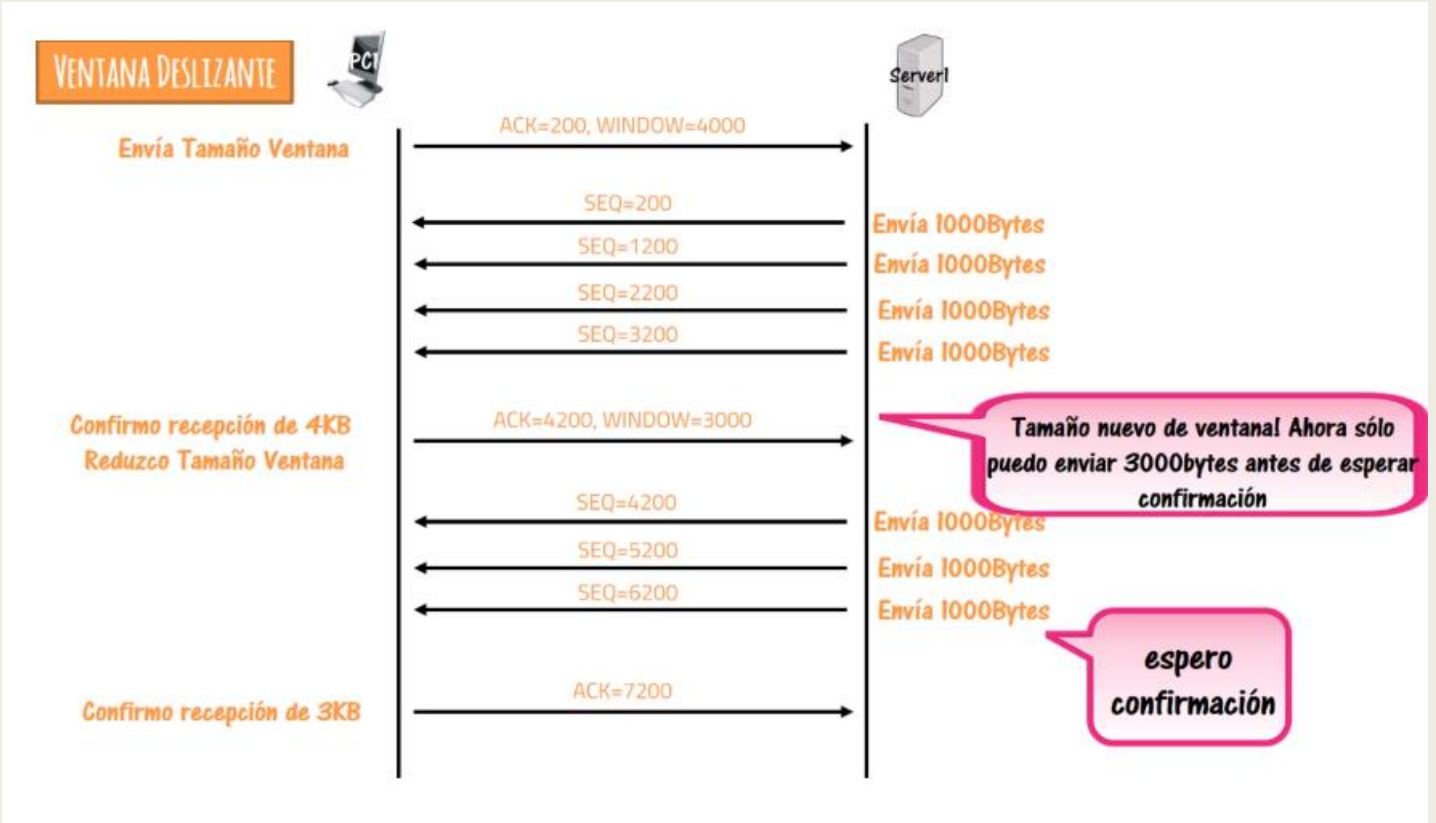
Control de flujo

- El control de flujo hace referencia a poder regular la velocidad del envío de los datos.
- El control de flujo se realiza a través de un mecanismo que se llama Ventana Deslizante (sliding window)
  - *Esta ventana se puede ir ajustando a lo largo de la conversación.*
  - *La pueden ajustar los dos extremos de la conversación.*
- En la cabecera TCP tiene un campo Ventana



Ejemplo de Control de flujo

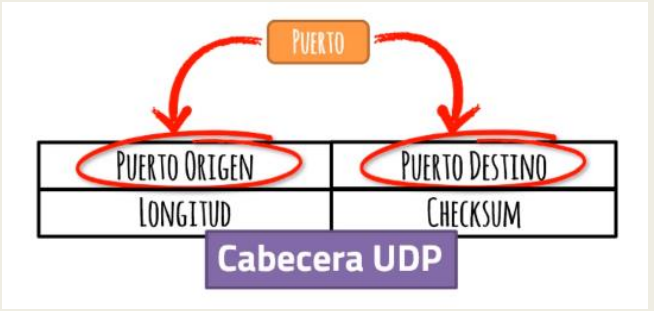
- En el siguiente ejemplo, inicialmente el campo de Ventana está establecido a 4000 bytes.
  - *Es decir, se podrán enviar 4000 bytes seguidos antes de esperar una confirmación.*
- Al acabar el envío de los 4000 bytes, se envía el ACK y se espera una confirmación.
- El otro extremo responderá si ha recibido los datos, y podrá responder con un cambio de la Ventana a otro valor, por ejemplo, 3000 porque se está saturando.



TCP Y UDP: CHECKSUM

UDP

- Ahora vamos a tratar el protocolo UDP.
- En UDP la los datos (PDU) reciben el nombre de DATAGRAMA.
- Este protocolo permite el envío de datagramas sin establecer previamente una conexión.
  - *Ya no necesitamos el 3-way-handshake.*
  - *Aquí se envía directamente la información.*



- UDP envía la información con una cabecera mínima
- No tiene confirmación de envíos, por lo tanto se dice que es no confiable.
- No permite el control de flujo (no tenemos campo ventana)
- No ordena (no hay números de secuencia SEQ)
- Tampoco segmenta la información en partes

¿Entonces qué ventaja tiene UDP?

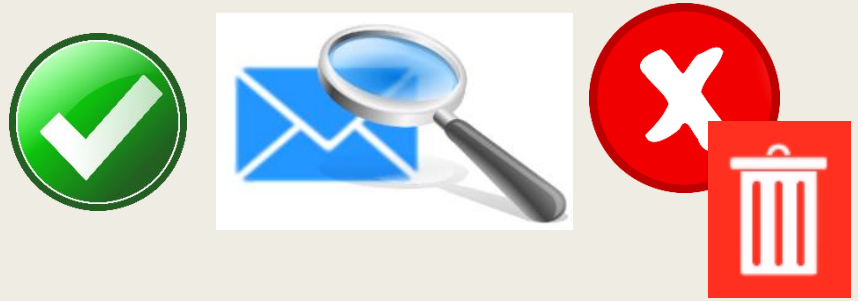
- Es un protocolo ligero y mucho más rápido que TCP.
- Y que UDP no tenga tantas funciones, no supone un problema para según qué tipo de aplicaciones.
  - *Es ideal para aplicaciones en tiempo real.*
  - *Pensad en el streaming, si estamos viendo un partido en directo, no nos interesa que se reenvíe la información, o se reordene si llega fuera de tiempo.*
    - Porque lo estamos viendo en directo, y el tiempo que conllevaría realizar todas las funciones de TCP nos impediría verlo en directo más que perder algún fotograma o sonido puntual.

El campo CHECKSUM

- Ahora vamos a tratar el campo Checksum, que está presente tanto en TCP como en UDP.
- Cuando enviamos información en la red, es posible que esta información se altere, por problemas de la red, o algún acto malintencionado.

- Lo más habitual es el error en la red, dado que se envían cantidades inmensas de 0 y 1, y puede darse el caso que algunos cambien su valor por algún problema en los circuitos eléctricos o señales de radio.
- A nivel de seguridad informática, este proceso para que los datos no se corrompan, recibe el nombre de integridad de los datos.
  - *Y tanto TCP como UDP se encargan de controlar la integridad de los datos a través de su campo **Checksum**.*

Checksum: ¿Qué hace?



- El campo Checksum verifica si el segmento o datagrama ha sufrido algún tipo de alteración.
- Si en esa verificación se concluye que los datos no han sido alterados, se aceptan.
- En caso de detectarse que los datos han sido alterados, se descartan porque están corrompidos.
- En el caso de TCP se reenviarían los datos, y en UDP no, pero esto ya es por las características del protocolo.
- Es decir, la función de CHECKSUM solo se ocupa de validar si los datos son correctos o no. NO LOS CORRIGE.

CHECKSUM: ¿Cómo funciona?

- Checksum = Suma de comprobación
- Cuando un equipo va a enviar los datos, lo que hace es coger todos los datos (DATOS + CABECERA) y realiza una operación aritmética.
  - *Una suma siguiendo unas normas concretas.*
- De esa operación aritmética se obtiene un resultado, un número en binario que se coloca en el campo Checksum.
- Cuando el segmento o datagrama llega al destino tras atravesar la red, este realiza la misma operación aritmética y le dará un resultado.
- Si el resultado de su operación es el mismo que el de la cabecera, se entiende que no se ha alterado la información. Si el resultado es diferente, se decartará la información.

¿Preguntas?

FUNDAMENTOS DE LA CAPA 4

TCP y UDP – Multiplexación, 3-way-Handshake, FIN

…