Tema 4: Capa de Transporte

Redes de Computadores

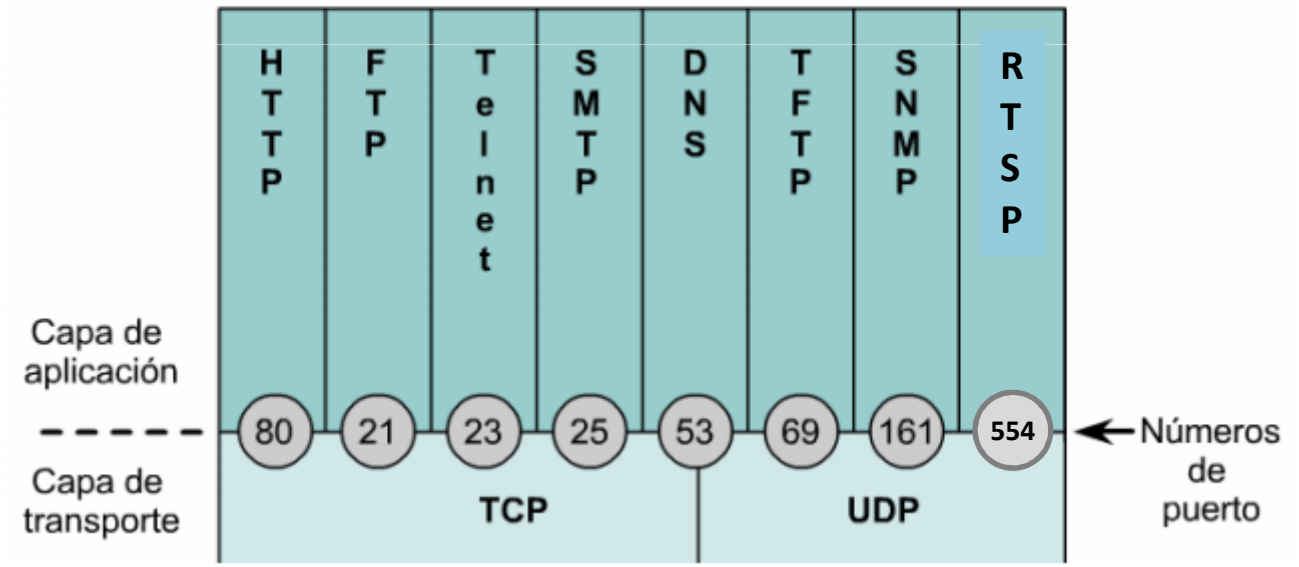
Grado en Ingeniería Informática

Mercedes Rodríguez García

Índice

1. Protocolos de la capa de transporte

- **TCP** (Protocolo de Control de Transmisión): es un protocolo orientado a conexión.
- **UDP** (Protocolo de Datagrama de Usuario): es un protocolo NO orientado a conexión.



2. Protocolo TCP

El protocolo IP de la capa de red no garantiza la entrega de paquetes (IP envía paquetes sin saber si han sido recibidos por el equipo destino), es un protocolo no confiable.

En su defecto, esta garantía la puede proporcionar la capa de transporte mediante el protocolo TCP.

TCP garantiza la entrega

2. Protocolo TCP

En una conexión TCP existen 3 etapas:

- Establecimiento de la conexión.
- Transferencia de datos.
- Fin de la conexión.

Los campos más importantes del encabezado de un segmento TCP:

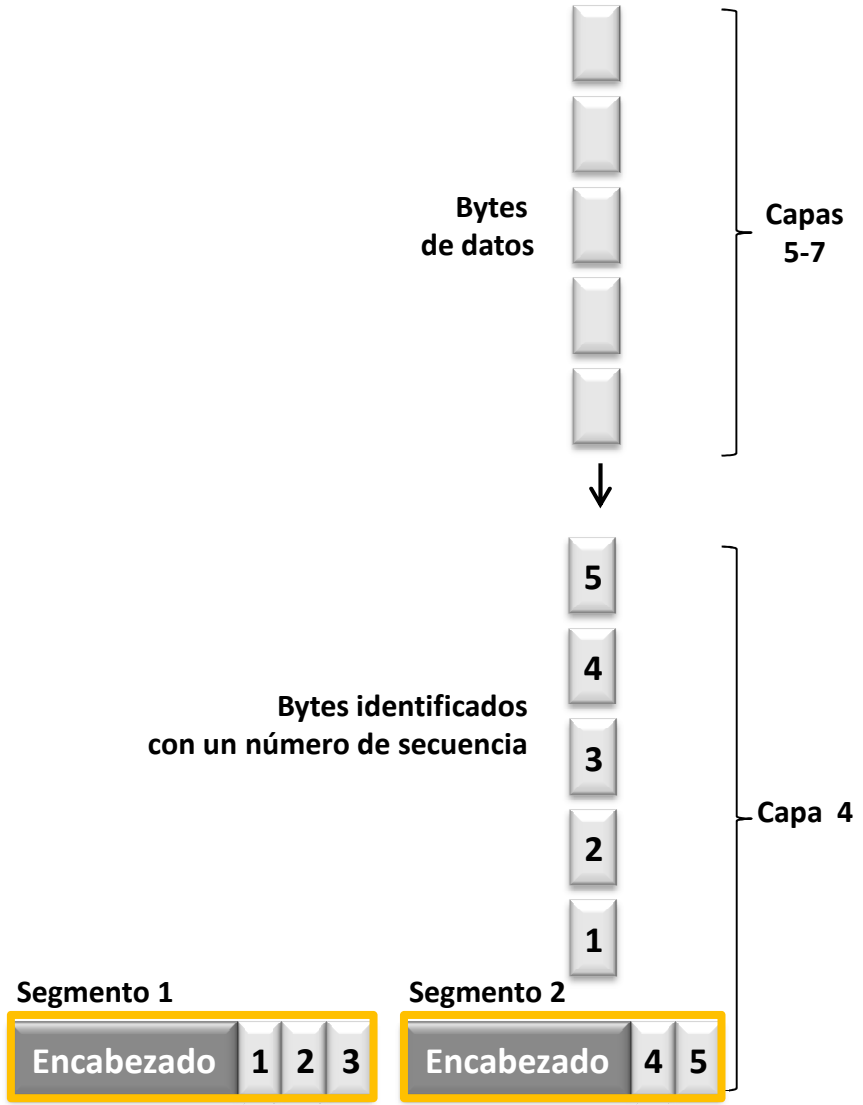
- Puerto origen.
- Puerto destino.
- Tipo de segmento.
- Número de secuencia.
- Acuse de recibo (ACK).
- Tamaño de la ventana.

TCP Header

Offsets Octet	0								1								2								3							
Octet	Bit	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
0	0	Source port																Destination port															
4	32	Sequence number																															
8	64	Acknowledgment number (if ACK set)																															
12	96	Data offset				Reserved 0 0 0			NS	CWR	ECE	URG	ACK	PSH	RST	SYN	FIN	Window Size															
16	128	Checksum																Urgent pointer (if URG set)															

La capa 4 recibe de las capas 5-7 los **bytes** que se quieren transmitir. La capa 4 **ASOCIA un número de secuencia** a cada byte para identificarlo. El número de secuencia inicial es aleatorio.

La capa 4 **agrupa estos bytes en segmentos.** El número de secuencia que aparece en la encabezado del segmento es el **número de secuencia del primer byte** del segmento.



La cantidad máxima de bytes de datos que puede contener un segmento se denomina **MSS** (Maximum Segment Size).

Normalmente el MSS se establece para que un segmento se ajuste a una única trama de la capa de enlace. Como ya sabemos, el tamaño de la trama queda determinado por la MTU (Maximum Transmission Unit)

Cuando TCP envía un mensaje grande divide el mensaje en fragmentos de tamaño MSS, excepto el último fragmento que probablemente será más pequeño que MSS.

1 **Objetivo de un ACK**

Confirmar al emisor los bytes de datos que se han recibido correctamente. Así conseguimos que TCP sea un protocolo de transporte confiable.

2 **¿Qué es un ACK?**

Es el número de secuencia del siguiente byte que se espera recibir, quedando confirmados los bytes anteriores.

Por ejemplo, si ACK es igual a 100 significa que se han recibido correctamente los 99 primeros bytes y se está esperando recibir del byte número 100 en adelante.

1 **EMISOR: ¿Puedo enviar el siguiente segmento antes de recibir el ACK del envío anterior?**

Sí, el emisor puede seguir enviando segmentos.
En caso contrario, se desperdiciaría buena parte del ancho de banda de la red.

2 **RECEPTOR: ¿Es necesario enviar un ACK nada mas recibir un segmento?**

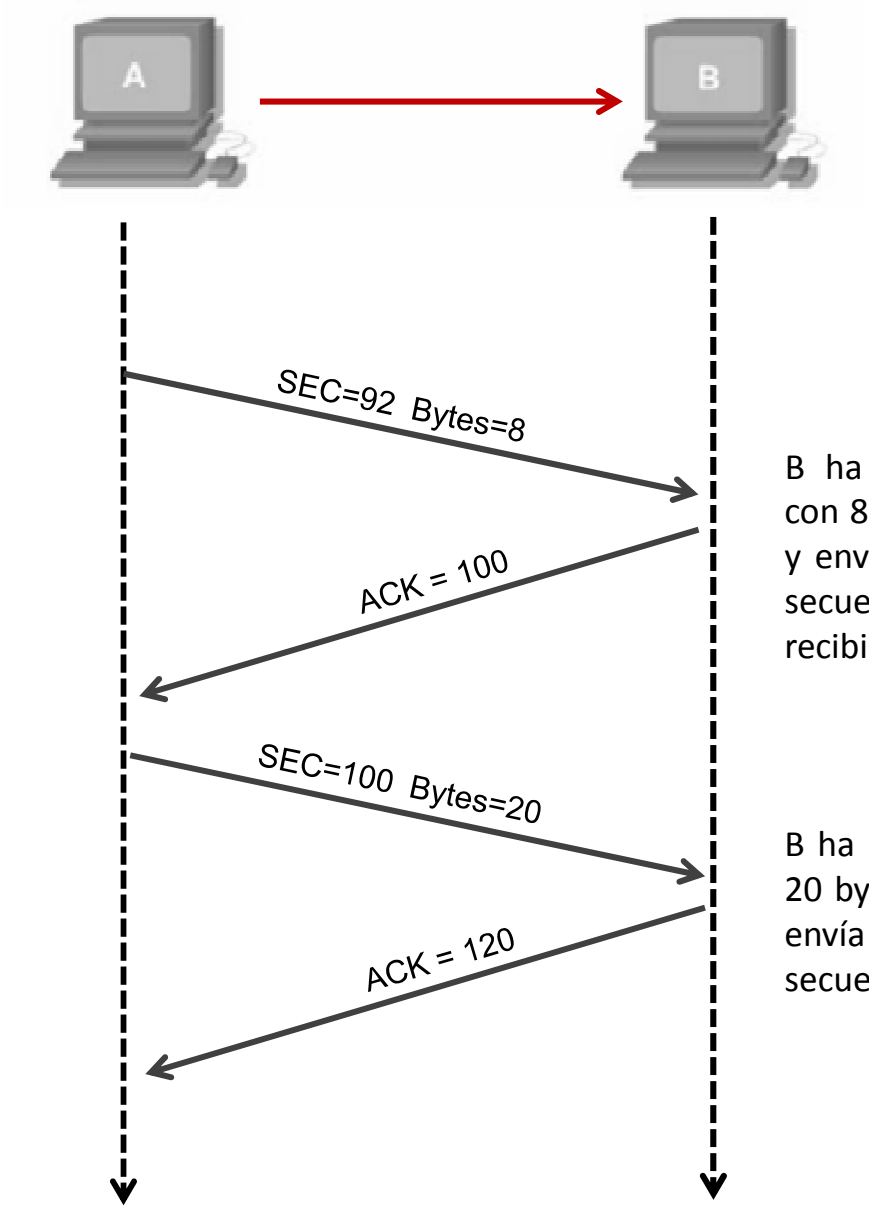
No, el receptor puede esperar y emitir un ACK para un conjunto de segmentos.

El host que inicia la comunicación se etiqueta como **cliente** y el otro como **servidor**.

En los ejemplos consideraremos únicamente casos de **transferencia unidireccional de carga útil**, es decir, la carga útil (bytes de datos) se transmite siempre del host A (cliente) al host B (servidor).

Cuando ambos equipos transmiten carga útil, se dice que la transferencia es bidireccional. La **transferencia bidireccional de carga útil** conceptualmente no es más difícil, pero es más complicada de representar en un ejemplo.

2.4. Acuse de recibo (ACK)

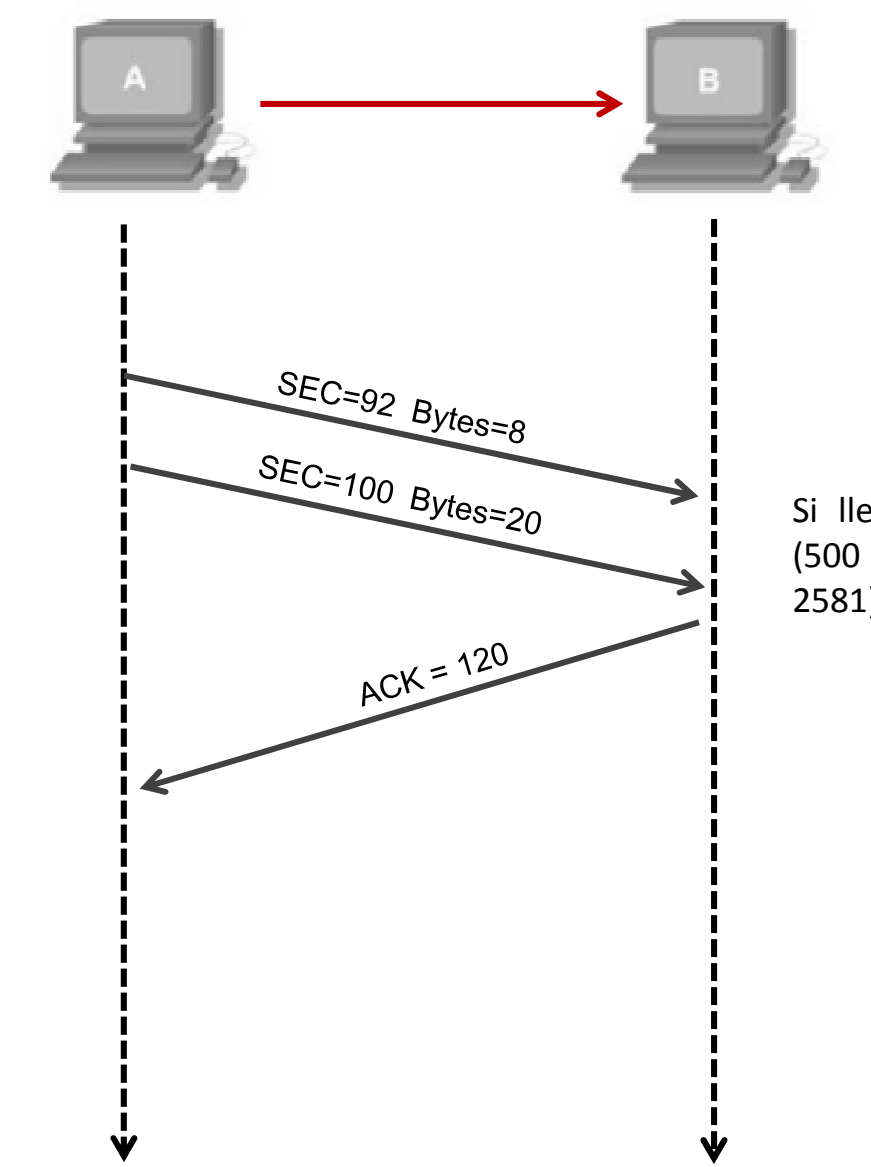


CASO 1: uso del ACK

B ha recibido correctamente un segmento con 8 bytes de datos (Nº secuencias = 92-99) y envía confirmación a A indicando el nº de secuencia del siguiente byte que espera recibir.

B ha recibido correctamente un segmento con 20 bytes de datos (Nº secuencias = 100-119) y envía confirmación a A indicando el nº de secuencia del siguiente byte que espera recibir.

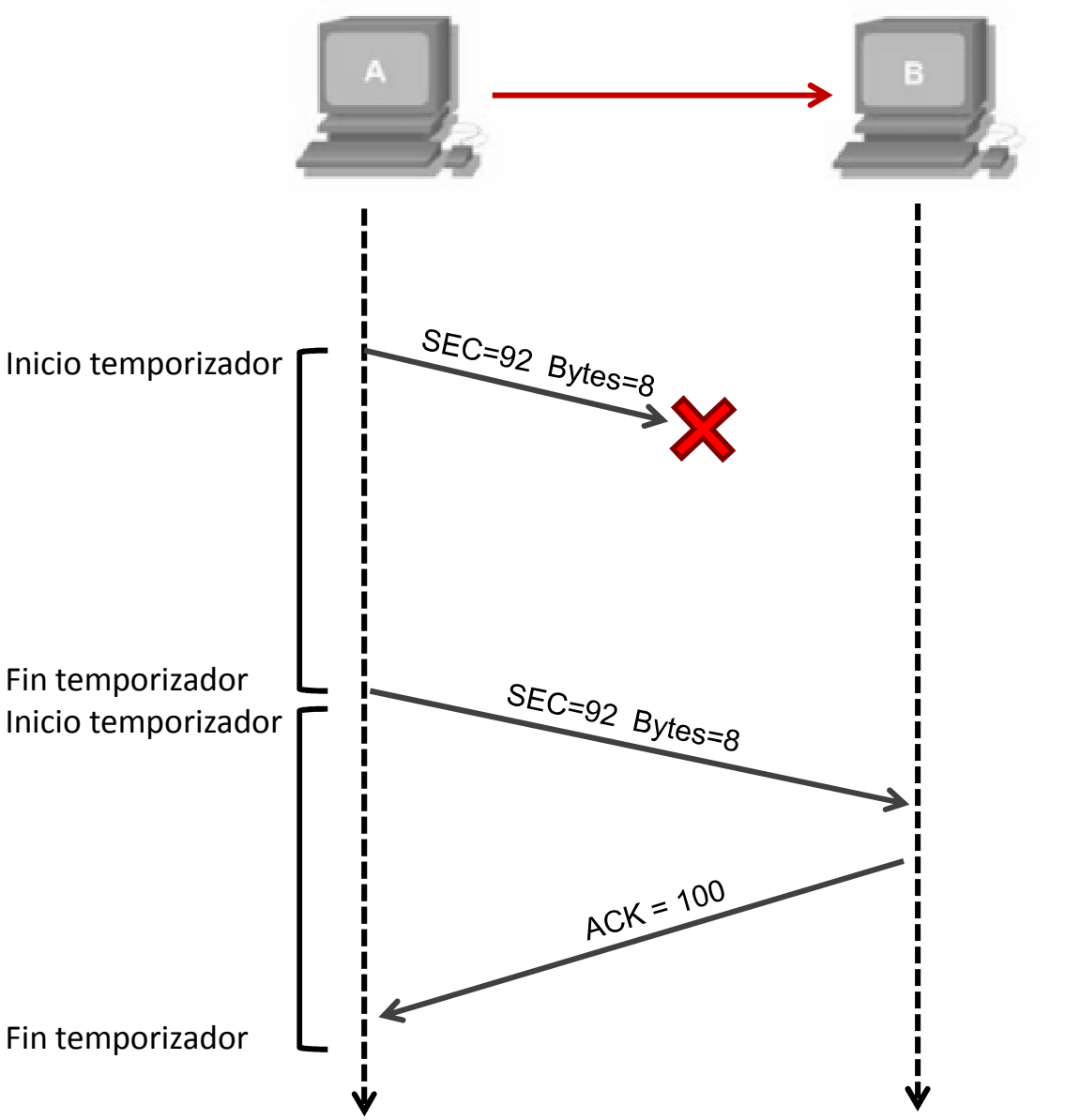
2.4. Acuse de recibo (ACK)



CASO 2: ACK acumulativo

Si llegan dos segmentos muy seguidos (500 ms según recomendación RFC 1122, 2581), se envía un ACK acumulativo.

2.4. Acuse de recibo (ACK)

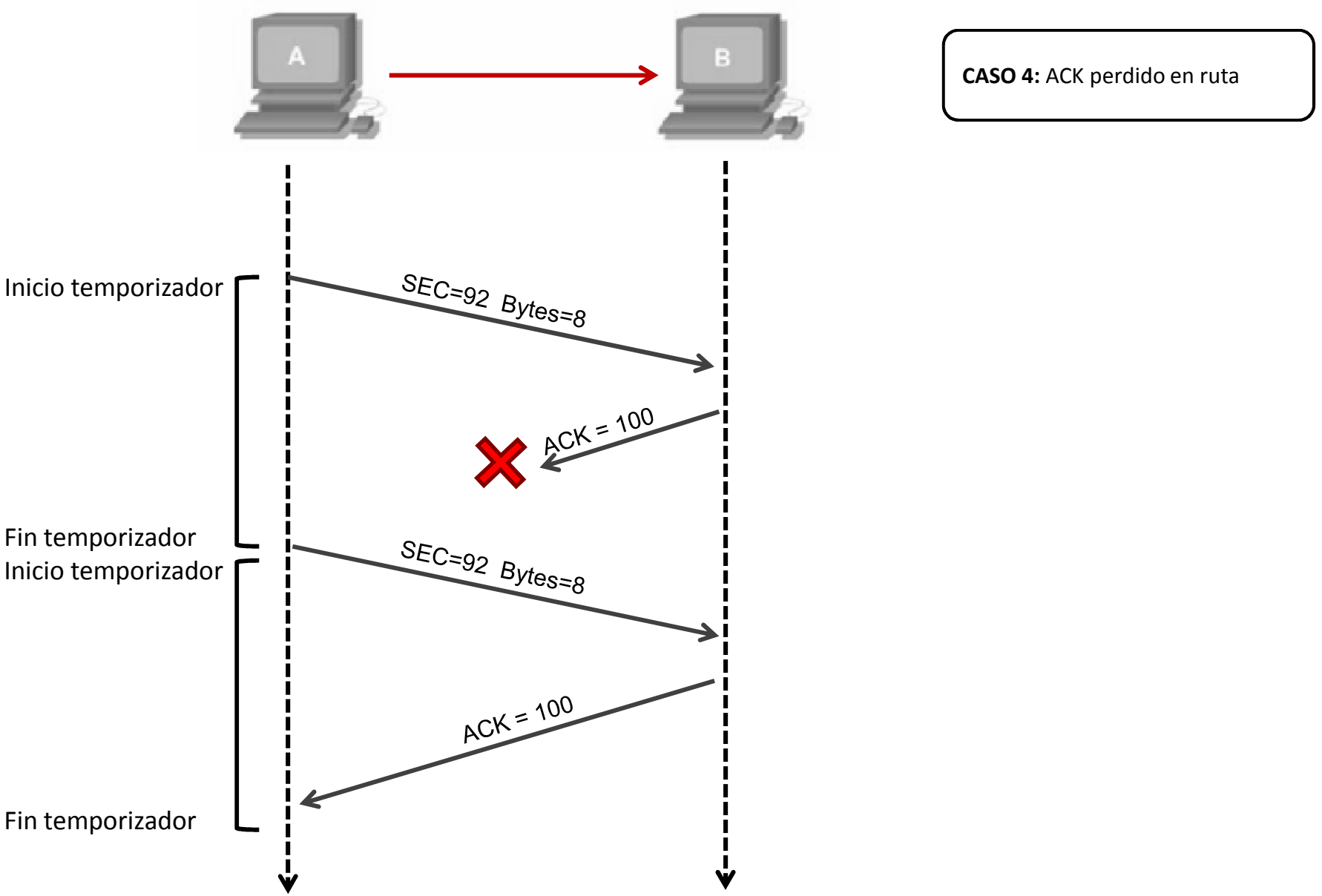


CASO 3: segmento perdido en ruta

Las pérdidas de paquetes en ruta normalmente son debidas al desbordamiento de los buffers de los routers. Típico cuando la red está congestionada.

2. Protocolo TCP

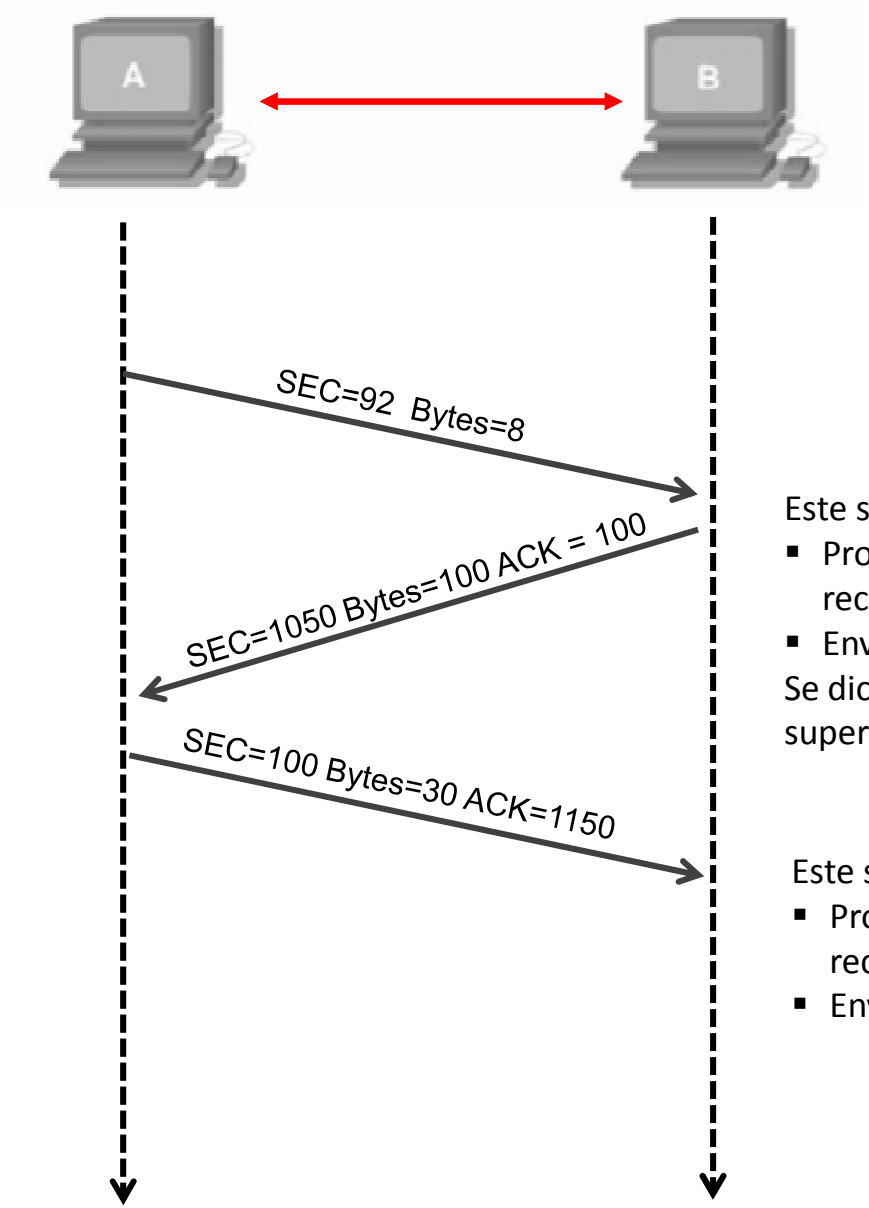
2.4. Acuse de recibo (ACK)



CASO 4: ACK perdido en ruta

2. Protocolo TCP

2.4. Acuse de recibo (ACK)



CASO 5: transferencia bidireccional

Este segmento tiene un doble propósito:

- Proporcionar a A un ACK de los datos recibidos.
- Enviar carga útil (datos) a A.

Se dice que el reconocimiento (ACK) está superpuesto al segmento de datos.

Este segmento tiene un doble propósito:

- Proporcionar a B un ACK de los datos recibidos.
- Enviar carga útil (datos) a B.

1 **¿Para qué se utiliza la ventana de recepción?**

Se utiliza para regular la cantidad de bytes que se pueden enviar consecutivamente al receptor **sin desbordar su buffer de recepción**. Esto se conoce como **control de flujo** y previene que un emisor rápido sature a un receptor lento.

2 **¿Quién establece el tamaño de la ventana de recepción?**

El tamaño de la ventana es establecido por el receptor basándose en la capacidad de su buffer de recepción.

Inicialmente: **Ventana** = BufferRecepción
Durante la comunicación: **Ventana** = espacio libre en BufferRecepción

3 **¿Cómo conoce el emisor el tamaño de la ventana de recepción?**

El receptor comunica al emisor el tamaño de la ventana de recepción mediante el campo ventana de la cabecera del segmento.

4 **¿Por qué se denomina también ventana deslizante?**

Cuando el emisor recibe un ACK, esta ventana se desplaza hacia delante sobre el espacio de números de secuencia.

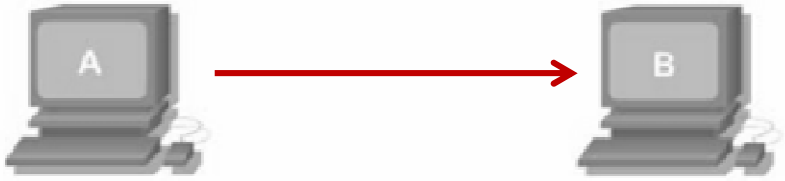
SOBRE LOS EJEMPLOS SIGUIENTES: para simplificar y poder entender el concepto de ventana deslizante, cada transferencia se representa

- SIN indicar los bytes del segmento que se está enviando.
- SIN indicar el número de secuencia del segmento que se está enviando.

"Simplemente se dice que se envía un segmento , sin dar más detalles"

2. Protocolo TCP

2.5. Ventana de recepción



CASO 1: funcionamiento de la ventana de recepción.
BufferRecepción = 4

EMISOR

RECEPTOR

SEGMENTO 0 enviado

0 1 2 3 4 5 6 7 8

Segmento = 0

Inicialmente:
V-Recepcion$_B$ = BufferRecepción

SEGMENTO 0 recibido

0 _ _ _

ACK = 1 Ventana = 3

Después:
V-Recepcion$_B$= espacio libre en BufferRecepción

ACK 1 recibido

0 1 2 3 4 5 6 7 8

SEGMENTO 0 leído

_ _ _ _

SEGMENTO 1 enviado

0 1 2 3 4 5 6 7 8

Segmento = 1

SEGMENTO 1 recibido

1 _ _ _

SEGMENTO 2 enviado

0 1 2 3 4 5 6 7 8

Segmento = 2

SEGMENTO 2 recibido

1 2 _ _

V-Recepcion$_B$ define la cantidad máxima de bytes que el host A puede enviar sin recibir un ACK.

V-Recepcion$_B$

BufferRecepcion$_B$

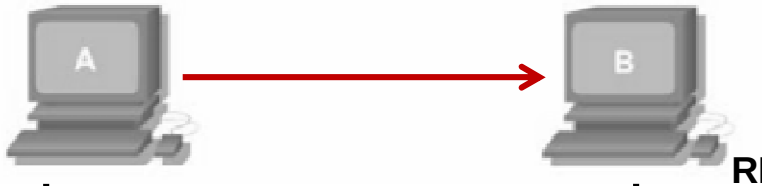
SEGMENTO 1 y 2 leído

_ _ _ _

ACK = 3 Ventana = 4

ACK 3 recibido

0 1 2 3 4 5 6 7 8



CASO 2: La aplicación del receptor tarda en leer los datos recibidos.
BufferRecepción = 4

EMISOR **RECEPTOR**

SEGMENTO 0 enviado

0 1 2 3 4 5 6 7 8

Segmento = 0

SEGMENTO 0 recibido

0 _ _ _

SEGMENTO 1 enviado

0 1 2 3 4 5 6 7 8

Segmento = 1

SEGMENTO 1 recibido

0 1 _ _

ACK = 2 Ventana = 2

SEGMENTO 2 enviado

0 1 2 3 4 5 6 7 8

Segmento = 2

SEGMENTO 2 recibido

0 1 2 _

SEGMENTO 3 enviado

0 1 2 3 4 5 6 7 8

Segmento = 3

SEGMENTO 3 recibido

0 1 2 3

PARALIZADO: ventana llena

ACK = 4 Ventana = 0

0 1 2 3 4 5 6 7 8

PARALIZADO: ventana a cero

Inicialmente:
V-Recepcion$_B$ = BufferRecepción

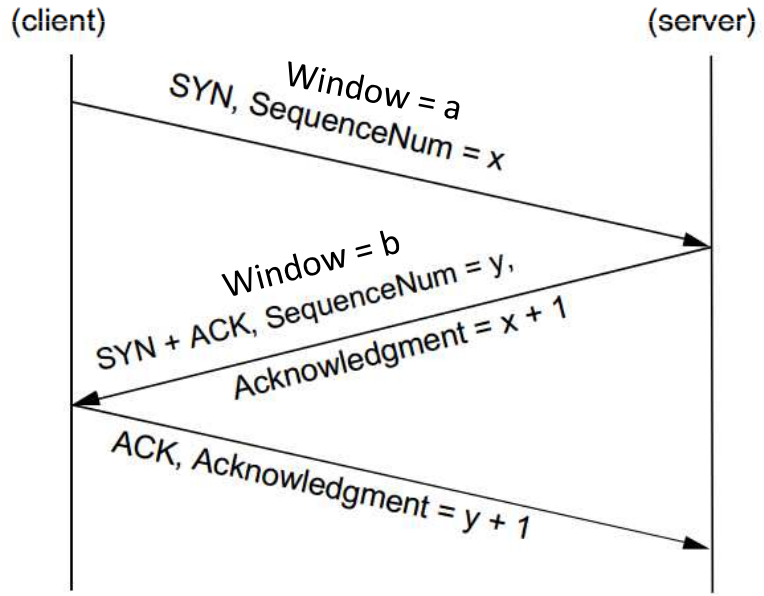
Después:
V-Recepcion$_B$= espacio libre en BufferRecepción

En este caso, la aplicación del receptor tarda en leer los datos. Los segmentos 0, 1, 2, 3 permanecerán en el buffer hasta ser leídos.

Si no existiese control de flujo, el emisor podría desbordar el buffer de recepción.

Antes de transmitir datos, cada host debe poner en conocimiento del otro su número de secuencia inicial y el tamaño inicial de su ventana de recepción.

Esta primera fase también se denomina **saludo de tres vías,** por ser un proceso de tres pasos.



Existe un ataque de denegación de servicio (DoS) denominado **SYN Flooding** (inundación SYN), ¿Puedes explicar el funcionamiento de este ataque?

3. Protocolo UDP

UDP no garantiza la entrega y ni el control de flujo.

4. Protocolo UDP

Ciertas aplicaciones prefieren utilizar el protocolo UDP porque es **más rápido**, aunque éste no garantice la entrega.

En este caso, son los protocolos de la capa de aplicación los que gestionan los errores.

¿Qué aplicaciones utilizan UDP?

- Aplicaciones que requieren **velocidad** y toleran pequeñas pérdidas de datos. En estos casos, resulta más importante transmitir a alta velocidad que garantizar la entrega de paquetes. Ejemplo: VoIP, streaming, juegos on-line.
- Aplicaciones que transmiten **pequeñas cantidades de datos**. Estas pequeñas cantidades de datos no justifican toda la información de control que se debe transmitir durante las fases de establecimiento y finalización de la conexión en el protocolo TCP. Ejemplo: la mayoría de las consultas-respuestas DNS.
- Aplicaciones que desean integrar los mecanismos de transferencia confiable en su código. Es posible que una aplicación proporcione transporte confiable utilizando UDP. Esto se puede conseguir si las **características de fiabilidad se incorporan en la propia aplicación**.

3. Protocolo UDP

¿Por qué es más rápido?

- No hay retardos debidos a fases de establecimiento y finalización de conexión.
- No hay que esperar acuses de recibo (ACK).
- Sobrecarga debida a la cabecera de los segmentos.

¿Por qué un servicio UDP soporta más clientes?

- TCP mantiene información del estado de cada conexión (número de secuencia, número de ACK, ventanas, …), esto consume recursos. UDP no registra esta información.

3. Protocolo UDP

UDP no utiliza:

- Números de secuencia
- Acuses de recibo (ACK)
- Ventanas

Lógicamente, el encabezado de un segmento UDP es diferente al encabezado de un segmento TCP.

Bits 0 - 15	16 - 31
Puerto origen	Puerto destino
Longitud del Mensaje	Suma de verificación

Encabezado de un segmento UDP, los campos sombreados en algunas ocasiones son opcionales.

4. Puertos

Los hosts, ya sean clientes o servidores, pueden ejecutar múltiples aplicaciones de red simultáneamente. Cada una de estas aplicaciones constituye un proceso.

Es necesario **identificar cada proceso de red** con un número. A este número se le llama puerto y tiene 16 bits (por tanto hay 65535 puertos).

Cuando la capa 4 del host destino recibe un segmento, examina el número de puerto destino para saber a qué proceso dirigir los datos.

La combinación de la dirección IP y el número de puerto se denomina **socket**. Un socket identifica un proceso de red de manera única en Internet.

Una **conexión** tiene dos sockets, uno por cada host. Este par de sockets identifica la conexión de manera única en Internet.

4. Puertos

Según IANA, los puertos se clasifican en:

- **Puertos bien conocidos**: 1-1023. Están reservados para protocolos estandarizados por IETF (estándares RFC) como por ejemplo HTTP, POP3/SMTP y Telnet.
- **Puertos registrados**: 1024-49151. Utilizados por aplicaciones no especificadas en RFCs. Cuando se crea un servicio de red, se puede solicitar a IANA reservar uno de estos números de puerto para la aplicación. Evitamos que distintas aplicaciones utilicen el mismo puerto y entren en conflicto.
- **Puertos dinámicos:** 49152-65535. Normalmente se asignan de forma dinámica a las aplicaciones clientes cuando se inicia la conexión.

Sobre la clasificación de puertos: http://tools.ietf.org/html/rfc6335

4. Puertos

Listado completo de números de puerto:
http://www.iana.org/assignments/service-names-port-numbers/service-names-port-numbers.xml

Si investigas la lista, verás que muchas aplicaciones pueden funcionar **tanto con TCP como con UDP**. Por ejemplo, observa la aplicación HTTP, siempre se ha dicho que utiliza TCP, sin embargo, hay dos puertos 80 uno para TCP y otro para UDP. Pues bien, HTTP se asocia con TCP porque es el que utiliza con mayor frecuencia.

Más información sobre HTTP sobre UDP:
http://en.wikipedia.org/wiki/HTTPU
http://en.wikipedia.org/wiki/Universal_Plug_and_Play

Por este motivo, los procesos son identificados con la combinación **protocolo-puerto**.

4. Puertos

Los números de puerto que hay que memorizar son:

Aplicación	Puerto **TCP**	Puerto **UDP**
FTP	21	21
SSH	22	22
Telnet	23	23
SMTP	25	25
DNS	53	53
TFTP	69	69
HTTP	80	80
POP3	110	110
HTTPS	443	443

En la tabla se ha resaltado el protocolo de transporte que generalmente utiliza ese servicio.

4. Puertos

Resultado del comando netstat -n

```
TCP    192.168.1.129:50774    173.194.34.193:80      ESTABLISHED
TCP    192.168.1.129:50788    150.214.80.25:80       ESTABLISHED
TCP    192.168.1.129:50791    63.131.149.244:21      SYN_SENT
```

Una conexión ejemplo sería:
192.168.1.129:50774 - 173.194.34.193:80

Puerto **50774**
Socket: 192.168.1.129:50774

Puerto **51561**
Socket: 192.168.1.129:50788

Puerto **50791**
Socket: 192.168.1.129:50791

Puerto **80**

Google
Socket: 173.194.34.193:80

Puerto **80**

UCA
Socket: 150.214.80.25:80

Puerto **21**

ftp.uvigo.es
Socket: 63.131.149.244:21

4. Puertos

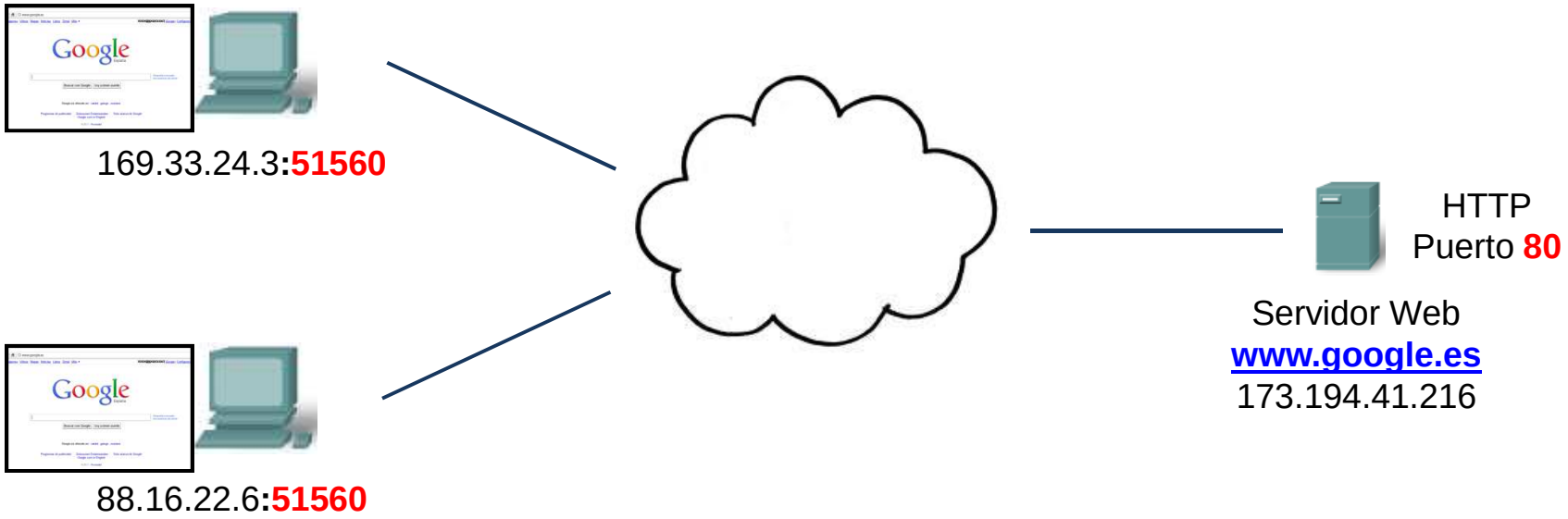
Resultado del comando netstat -n

```
TCP    192.168.1.129:50774    173.194.34.193:80      ESTABLISHED
TCP    192.168.1.129:50788    150.214.80.25:80       ESTABLISHED
TCP    192.168.1.129:50791    63.131.149.244:21      SYN_SENT
```

Estados TCP del lado del cliente:

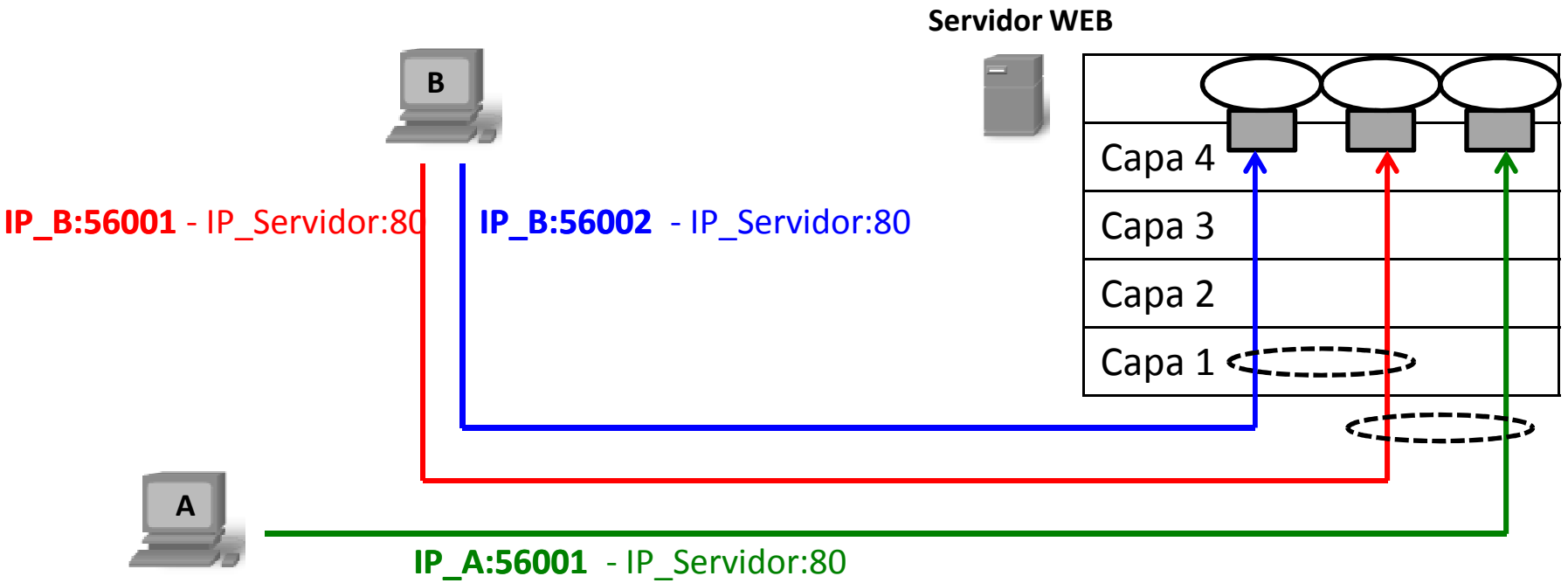
- **SYN_SENT**: el cliente inicia la conexión enviando un segmento SYN al servidor. Después de enviar este segmento, el cliente entra en el estado SYN_SENT.
- **ESTABLISHED**: cuando el cliente recibe respuesta del servidor, le envía el ACK correspondiente y entra en estado ESTABLISHED. Durante el estado ESTABLISHED, cliente y servidor se envían segmentos que contengan datos de carga útil.
- **FIN_WAIT_1**: cuando el cliente decide cerrar la conexión, envía un segmento FIN al servidor. Después de enviar este segmento, el cliente entra en el estado FIN_WAIT_1.
- **FIN_WAIT_2**: cuando el cliente recibe respuesta del servidor, entra en estado FIN_WAIT_2. Durante el estado FIN_WAIT_2, el cliente espera recibir otro segmento del servidor con el bit FIN puesto a 1.
- **TIME_WAIT**: cuando el cliente recibe respuesta del servidor, le envía el ACK correspondiente y entra en estado TIME_WAIT durante 30 seg – 2 min (depende de la aplicación).
- **CLOSED**: cuando expira el tiempo de espera anterior, la conexión se cierra y todos los recursos del cliente empleados en la conexión son liberados.

4. Puertos



Cuando un servidor recibe dos segmentos TCP con el mismo puerto destino (p.e. 80), serán dirigidos a sockets distintos **si difieren en la IP origen o en el puerto origen**.

4. Puertos



En un instante dado:

- El host B inicia dos sesiones HTTP con el servidor web y asigna un número de puerto origen distinto en cada conexión: 56001 y 56002.
- El host A inicia una sesión HTTP con el servidor web. Como A está seleccionando el puerto origen independientemente de B, también puede asignar el número de puerto 56001 a su conexión HTTP.