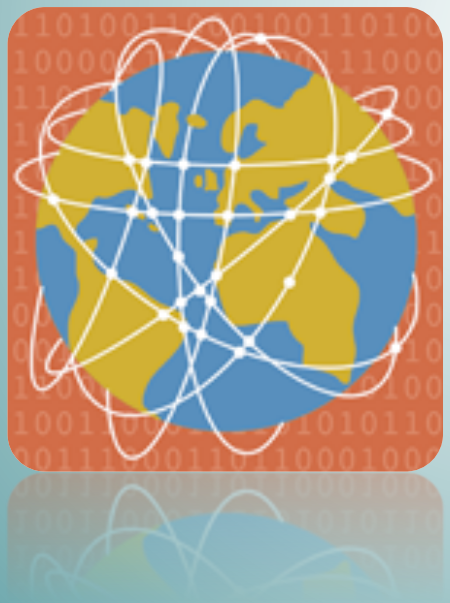
Protocolos de Interconexión de Redes

Tema 03. Protocolos de Capa de Transporte



Luis Sánchez González

DPTO. DE INGENIERÍA DE COMUNICACIONES

Contenido

- Capa de Transporte
  - Introducción
  - UDP
  - TCP

Contenido

- Introducción
  - Funcionalidades
  - Direccionamiento a nivel de transporte
- UDP
- TCP

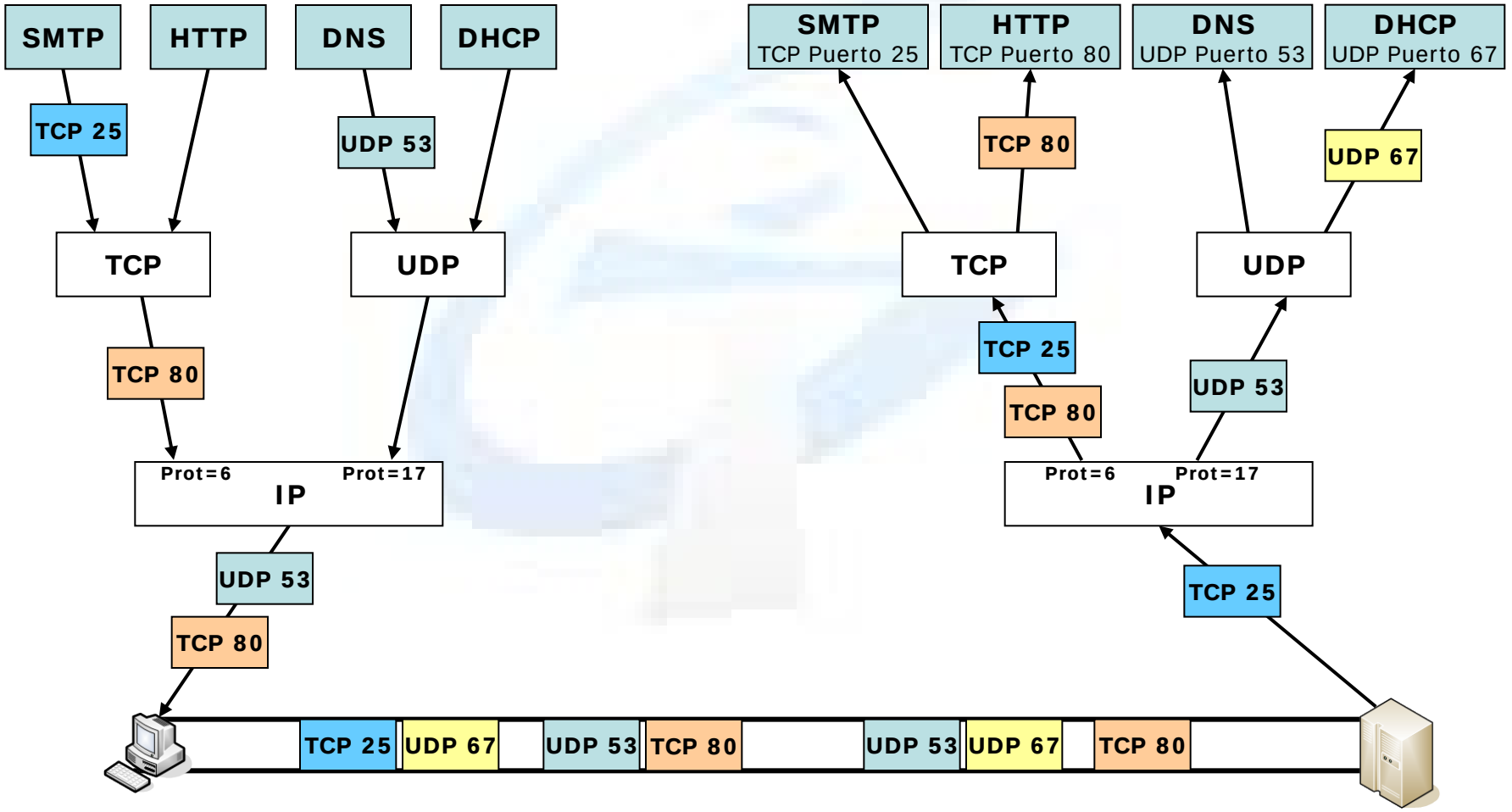
La capa de transporte

- Servicio de transporte de bits a las aplicaciones que se comunican
- Solo existe en las entidades extremas ⇒ nivel *HOST-HOST (E2E)*
- Servicio orientado a conexión: garantía de entrega, sin error, pérdida ni duplicación
  - TCP
  - Unidad básica de información : Segmento
- Servicio no orientado a conexión: sin confirmación, de forma independiente
  - UDP
  - Unidad básica de información : Mensaje

Direccionamiento a nivel de transporte

- Se necesita poder tener varias aplicaciones accediendo a la vez a la red
- Todos los flujos de datos procedentes de las aplicaciones se multiplexan sobre la misma dirección IP
  - Para el tráfico de salida en principio no habría mayor problema
  - **Problema: ¿Cómo se a que aplicación va destinado un paquete si sólo tengo la dirección IP?**
- Aparecen los conceptos de *puerto* y *socket.* Son las "direcciones" de nivel de transporte

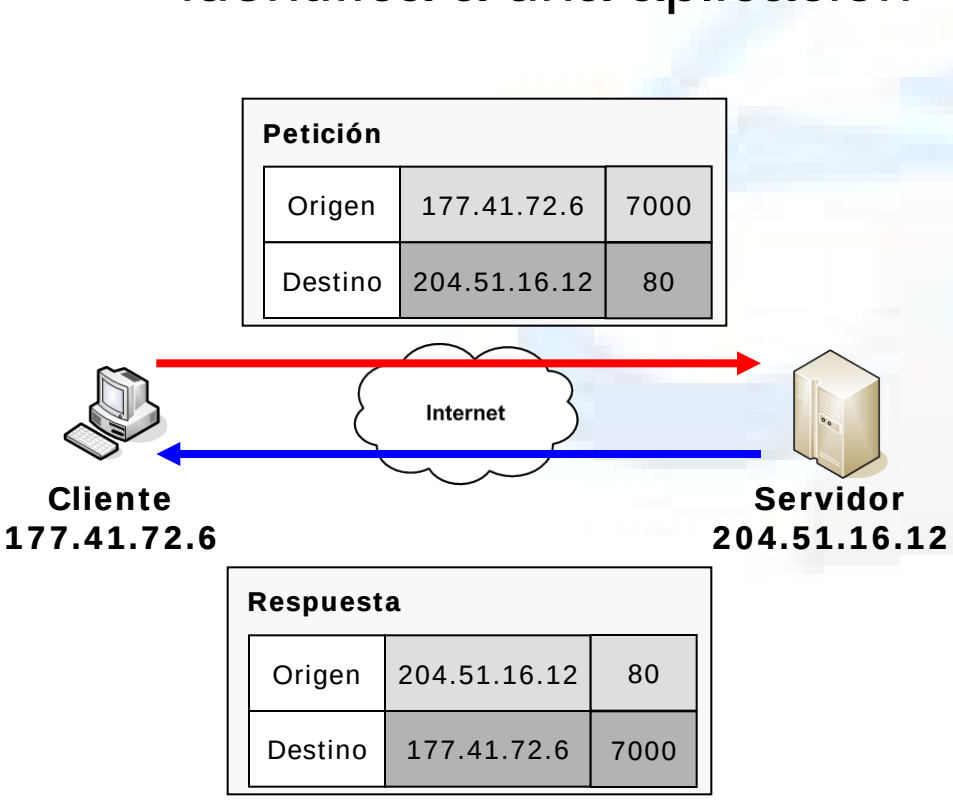
Direccionamiento a nivel de transporte

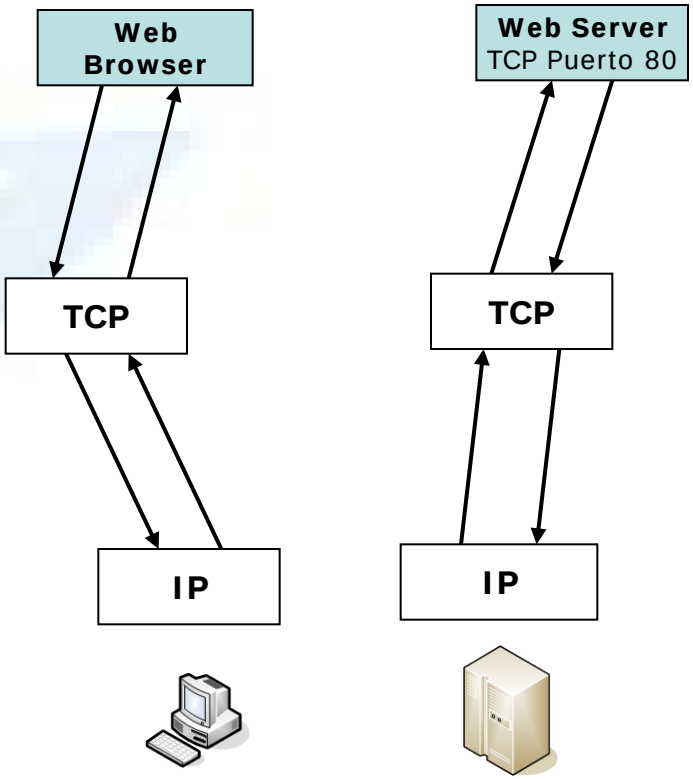


GRUPO DE INGENIERÍA TELEMÁTICA
Grupo de Ingeniería Telemática (G.I.T)
DICOM / Universidad de Cantabria
Luis Sánchez

Direccionamiento a nivel de transporte

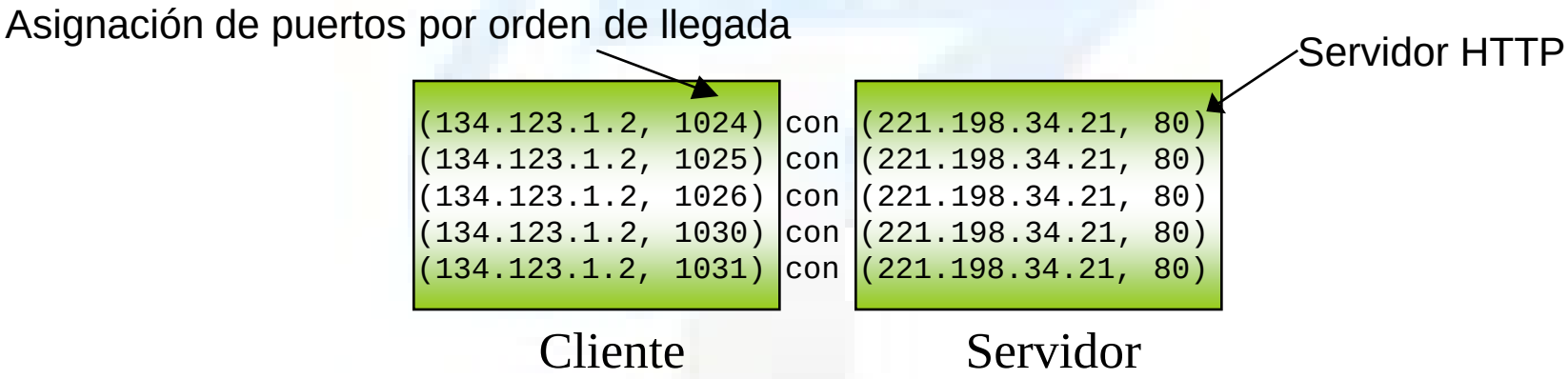
- **Socket:** la combinación de dirección IP y puerto identifica a una aplicación





Dirección IP + Puerto = Socket

Ejm.: 5 ventanas de navegador desde el equipo 134.123.1.2 con página de inicio el site del equipo 221.198.34.21:



Contenido

- Introducción
- UDP
  - Formato del mensaje UDP
  - Puertos y aplicaciones UDP
- TCP

UDP: Protocolo de Datagrama de Usuario

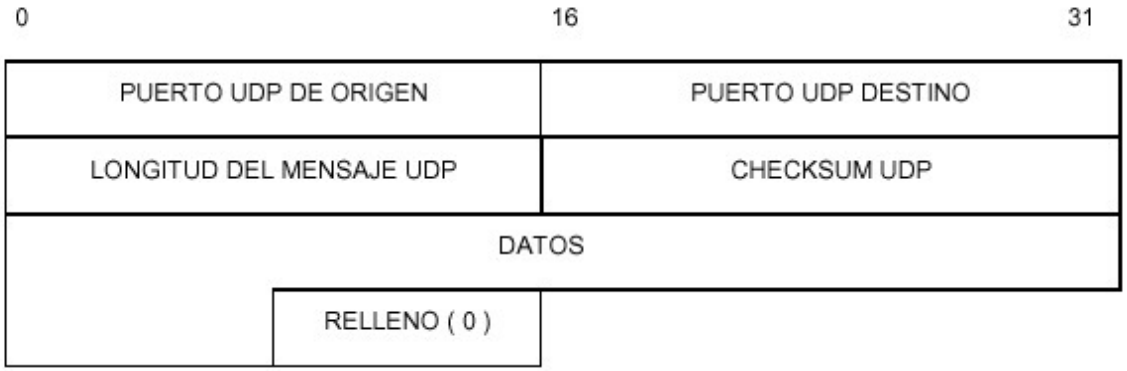
Se basa en el mismo principio no fiable y no orientado a la conexión de IP:

- No utiliza reconocimientos
- No proporciona control de flujo
- No reordena los mensajes recibidos

Los mensajes pueden perderse, duplicarse, llegar fuera de orden o más deprisa que el receptor procesarlos

La aplicación que utiliza UDP acepta la responsabilidad de gestionar el problema de la fiabilidad

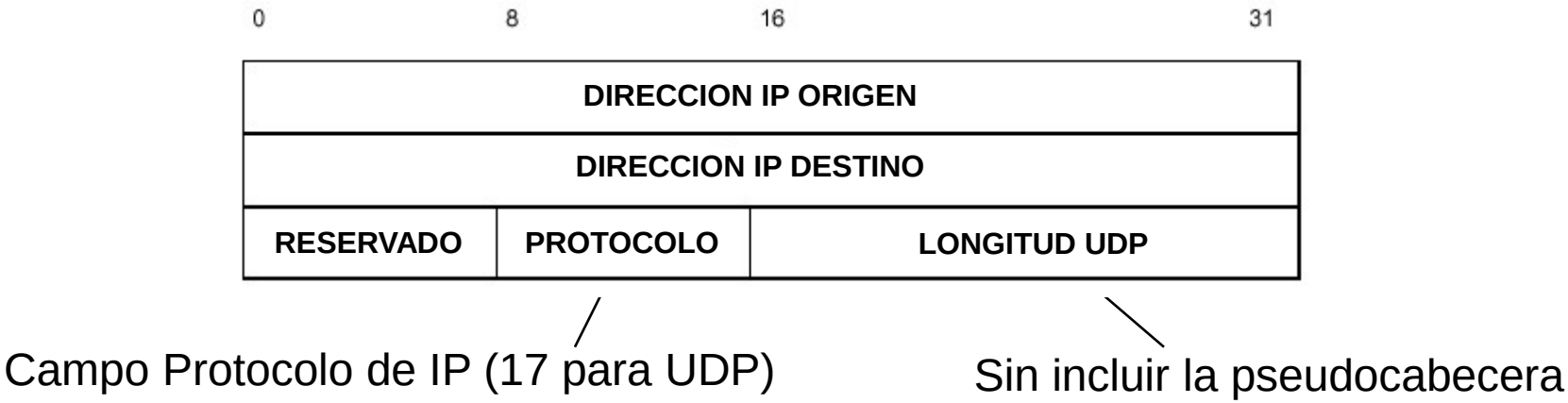
UDP: Formato de los mensajes



- **Puerto de origen y Puerto de destino**: identifican al proceso emisor y receptor
  ...Puerto de origen es opcional (0 si no se utiliza)
- **Longitud**: nº de octetos de cabecera + datos. Valor mínimo = 8
- **Checksum**: (opcional) valor 0 si no se calcula (IP no chequea los datos)
  - *Divide los datos en campos de 16 bytes*
  - *Complemento a uno de su suma en complemento a uno*

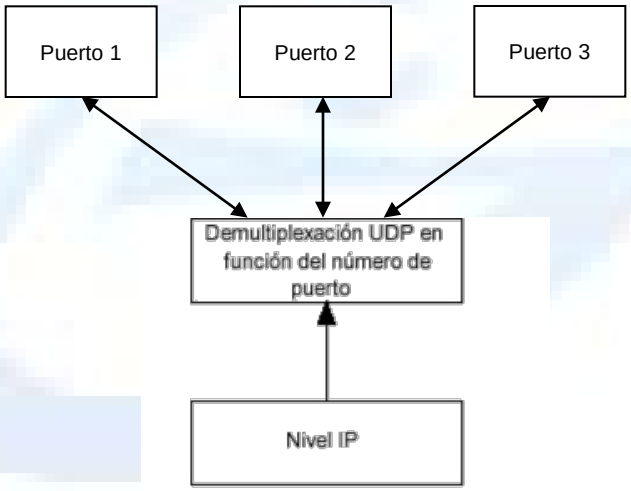
UDP: Pseudocabecera UDP

- Es añadida al mensaje antes de efectuar el cálculo del Checksum
- El relleno a ceros asegura múltiplos de 16 bits
- Para calcular el checksum:
  - Rellena el campo a ceros
  - Calcula la suma en complemento a uno de 16 bits
  - (incluida la pseudocabecera, cabecera UDP y datos)
- Verifica que el destino es correcto
- La comprobación es realizada mediante la dir IP de la cabecera IP



UDP: Multiplexación/Demultiplexación de Puertos

- Mux/Demux a través de Puertos
- UDP demultiplexa las UDP de IP basándose en los puertos



- Cada puerto tiene asociada una cola
- Caso de no encontrar el puerto asociado al datagrama se genera un comando ICMP de puerto inalcanzable y descarta el datagrama
- Caso de encontrar el puerto, el datagrama es encolado
- Caso de encontrar el buffer lleno, el datagrama es descartado

UDP: Puertos reservados y disponibles

Dos métodos de asignar los puertos:

- Asignación universal: Puertos bien conocidos
- Asignación dinámica: el SO asigna los puertos y los extremos se informan de las asignaciones

- Nº puerto: 0..65535
- Well-known ports (0..1023)
- RFC 1700 (Assigned Internet Numbers). Obsoleto desde 2002. (www.iana.org/assignments/port-numbers)

UDP: Aplicaciones y puertos bien conocidos

	Protocolo	Clave	Descripción
0	-	-	RESERVADO
7	ECHO	echo	ECHO
13	DAYTIME	daytime	Daytime
37	TIME	time	Time
53	**DNS**	nameserver	Domain Name Server
67	BOOTPC	bootps	Bootstrap Protocol Server
68	BOOTPC	bootpc	Bootstrap Protocol Client
69	TFTP	tftp	Trivial File Transfer
123	NTP	ntp	Network Time Protocol
161	SNMP	snmp	SMNP net monitor
162	-	snmp-trap	SNMP traps

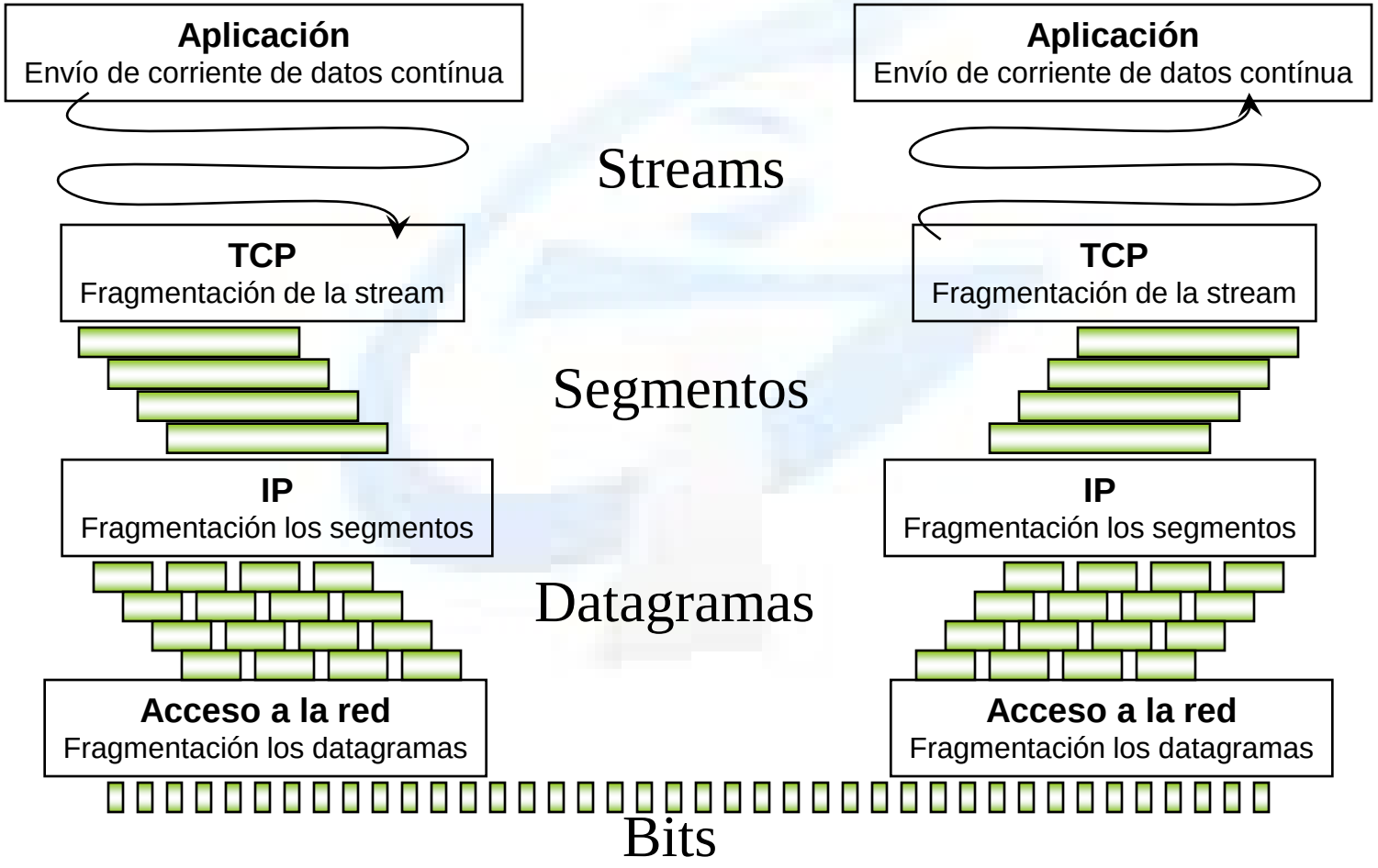
Contenido

- Introducción
- UDP
- TCP
  - Funcionalidades de TCP
  - Formato del segmento TCP
  - Estados de la conexión TCP
  - Manejo de la conexión TCP
  - Control de flujo
  - Control de errores
  - Servicios a la capa de aplicación
  - API Socket

TCP: Protocolo de Control de Transmisión

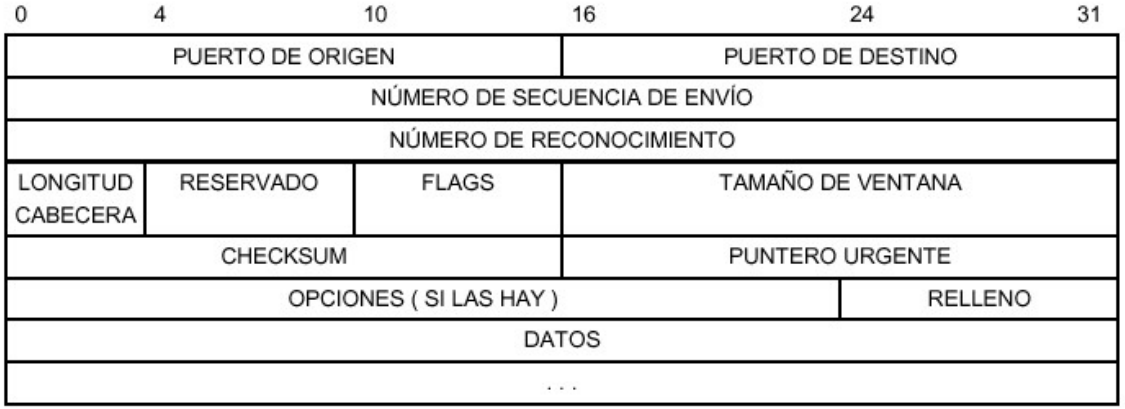
- Servicio de transmisión orientado a la conexión
- Flujo continuo de bytes durante la conexión
- Unidad de información adecuada a IP : SEGMENTO
- Reconocimientos temporizados
- Retransmisión tras timeout
- La verificación del Checksum permite los descartes (sin reconocimiento)
- Asegura la reordenación de los datos, indep. de los datagramas IP
- Control de flujo en función del espacio local de almacenamiento

TCP: Manejo de la información



TCP: Formato del segmento

20 octetos (sin opciones)



- **Nº secuencia:** posición del 1er octeto del campo de datos de cada segmento en la secuencia de la conexión
- **Nº reconocimiento:** Nº de secuencia del siguiente octeto de datos que se espera recibir
- **Tamaño de ventana:** Informa del tamaño de la ***Ventana de Envío*** del emisor del segmento
- **Longitud de cabecera:** va desde 20 hasta 60 octetos

TCP: Flags de cabecera

- **URG (Urgent)** indica Datos urgentes

(El **puntero urgent**e contiene la dirección donde terminan estos)

- **ACK (Acknowledgement)** Se ha hecho uso del campo **número de reconocimiento.**

(En la práctica el bit ACK está a 1 siempre, excepto en el primer segmento enviado por el host que inicia la conexión)

- **PSH (Push)** El receptor debe pasar estos datos a la aplicación tan pronto como le sea posible, sin esperar a acumular varios segmentos.

- **RST (Reset)** Debe reinicializarse la conexión.

(Indica que se ha detectado un error de cualquier tipo, por ejemplo, una terminación unilateral de una conexión o que se ha recibido un segmento con un valor inadecuado del número de secuencia o número de ACK.)

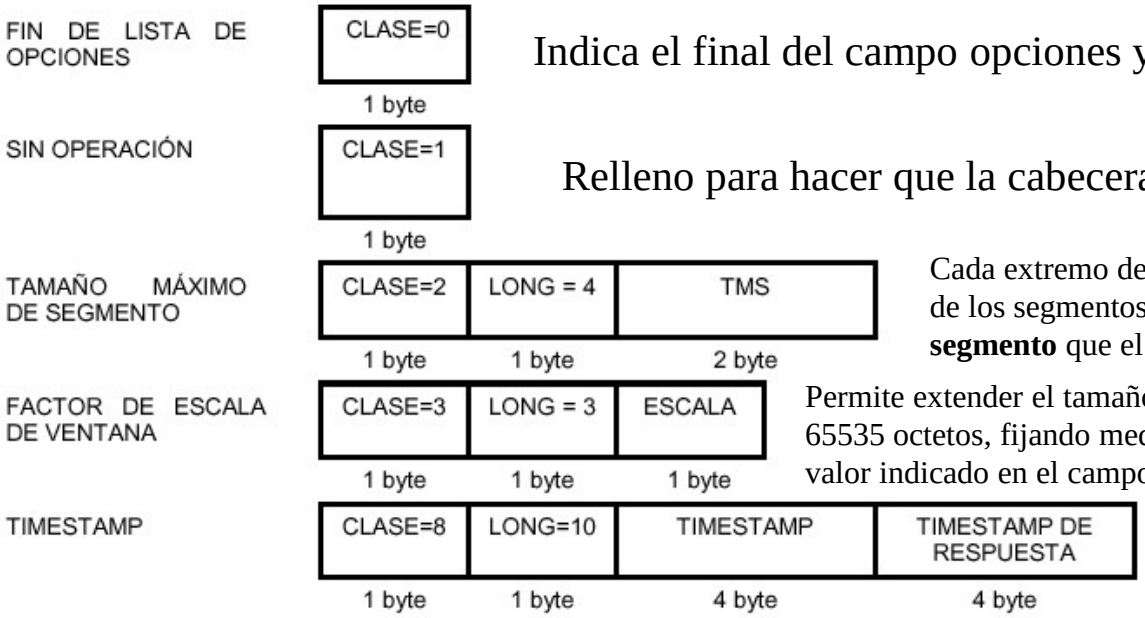
- **SYN (Synchronize)** Petición de sincronismo de números de secuencia para iniciar la conexión.

(El campo **número de envío** contiene el **número inicial de secuencia.** Este flag solo está puesto en el primer mensaje enviado por cada lado en el inicio de la conexión)

- **FIN (Finish)** El emisor ha terminado de enviar datos.

(Para que una conexión se cierre de manera normal cada host ha de enviar un segmento con el bit FIN puesto)

TCP: Opciones de cabecera



Indica el final del campo opciones y el inicio del campo de datos.

Relleno para hacer que la cabecera sea múltiplo de cuatro octetos

Cada extremo de una conexión especifica en el primero de los segmentos intercambiados el **tamaño máximo del segmento** que el desea recibir

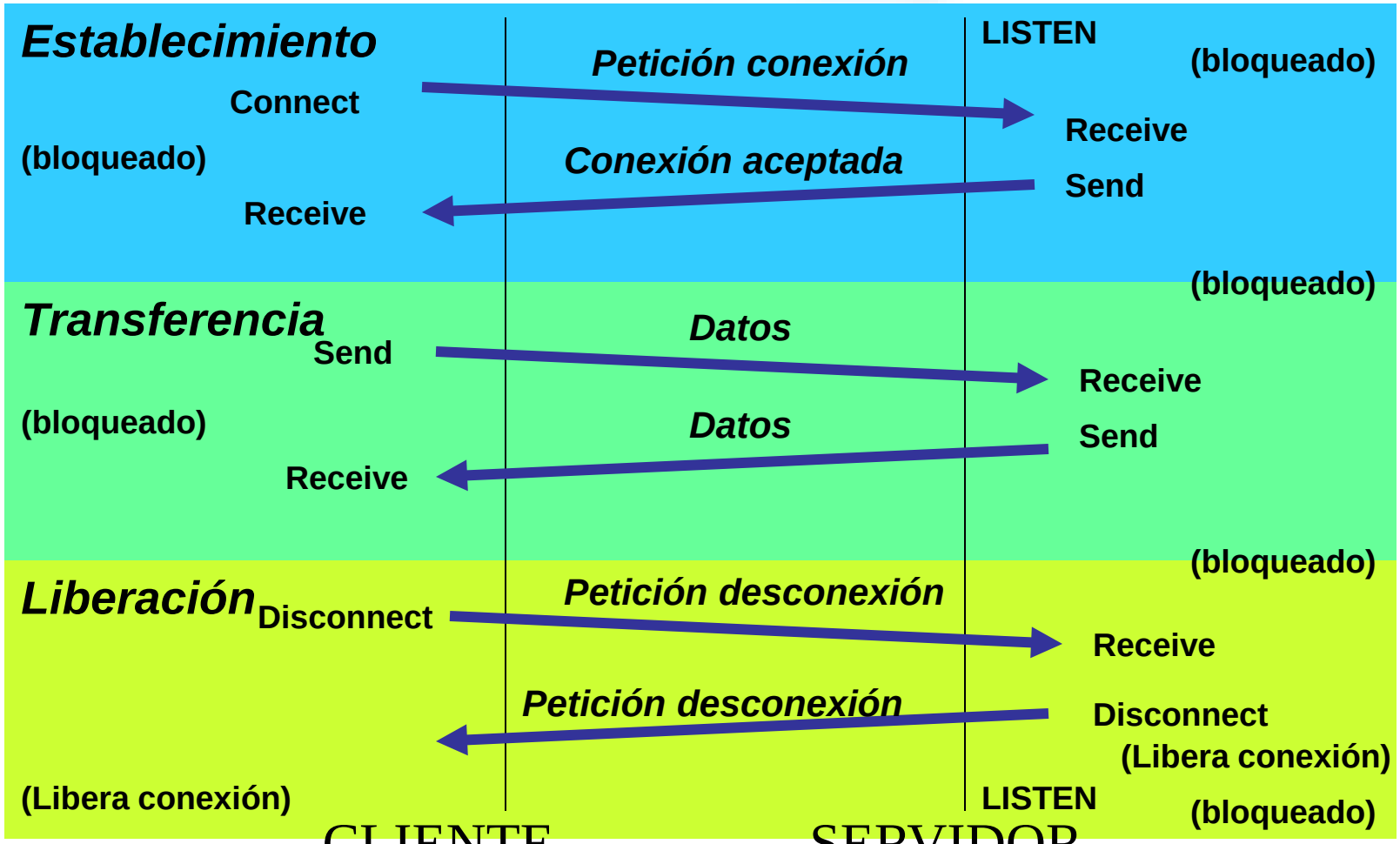
Permite extender el tamaño de la ventana más allá del límite de 65535 octetos, fijando mediante esta opción un factor de escala al valor indicado en el campo tamaño de ventana

Intercambia entre ambos extremos los valores del reloj del sistema (no todas las implementaciones hacen uso de esta opción)

TCP: Servicio orientado a la conexión

- Los protocolos orientados a conexión requieren un procedimiento explícito de establecimiento y terminación de la comunicación.
- El modelo más habitual de los servicios orientados a conexión se basa en dos protagonistas:
  - Cliente: el que inicia la conexión
  - Servidor: el que es invitado a conectar
- La conexión puede terminarse tanto por iniciativa del cliente como del servidor.

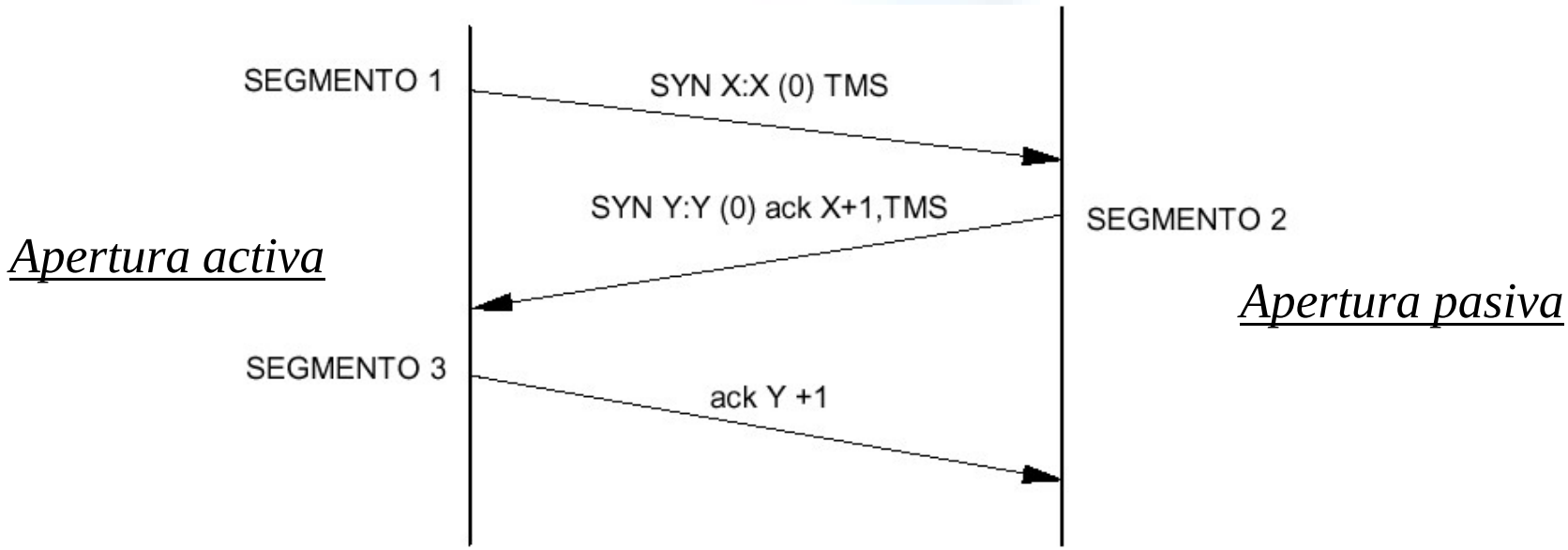
Transporte: Fases de una conexión



TCP: Saludo a tres vías (“*Three way handshake*”)

- En el nivel de transporte pueden llegar TPDUs duplicadas (ej. un emisor que no recibió el correspondiente ACK)
- Con un procedimiento de conexión simple una sesión entera podría verse duplicada.
- El ‘saludo a tres vías’ es un procedimiento de conexión que evita los problemas debidos a duplicados
- Para ello se identifica cada intento de conexión mediante un número diferente. El cliente elige un número para la comunicación en sentido de ida y el servidor otro para el sentido de vuelta.
- Estos dos números actúan como PINs que identifican la conexión frente a paquetes retrasados que pudieran aparecer por la red de conexiones anteriores.

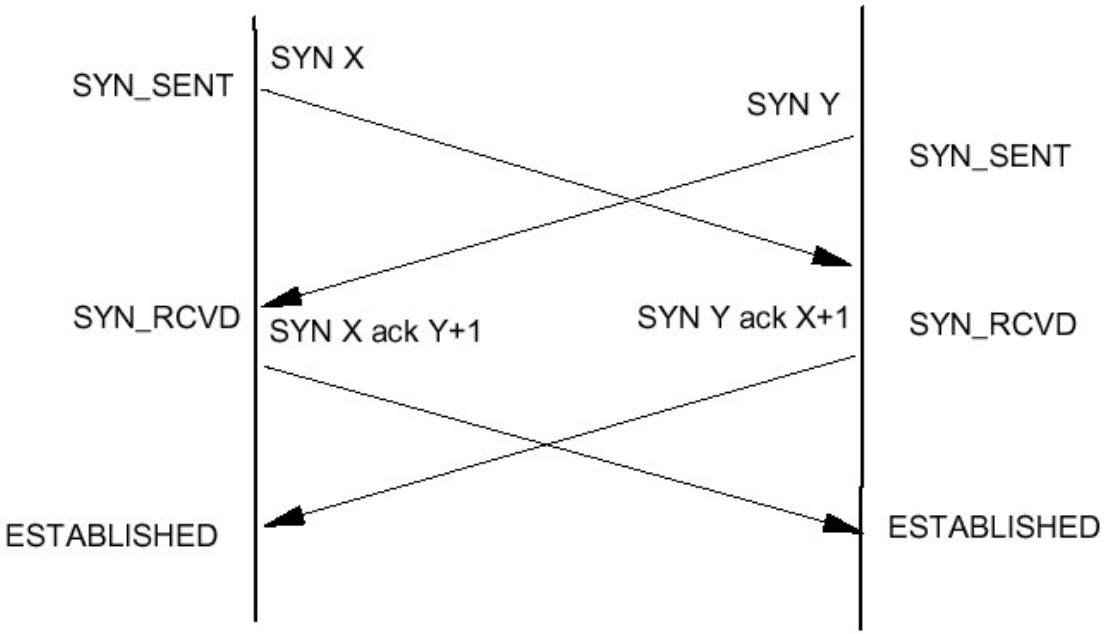
TCP: Establecimiento de la conexión



- NSI ≈ contador de 32 bits, con incremento de 1 cada 4 μs (4h46')
- Tras 6 segundos sin contestación a un SYN(1), se repite el SYN(2)
- Tras 24 segundos sin contestación al SYN(2), se repite el SYN(3)
- Si no hay contestación al SYN(3) se abandona el establecimiento
- Ante caídas del TCP, el arranque del software de TCP debe demorarse al menos 2 minutos ≈ TTL máximo de un paquete

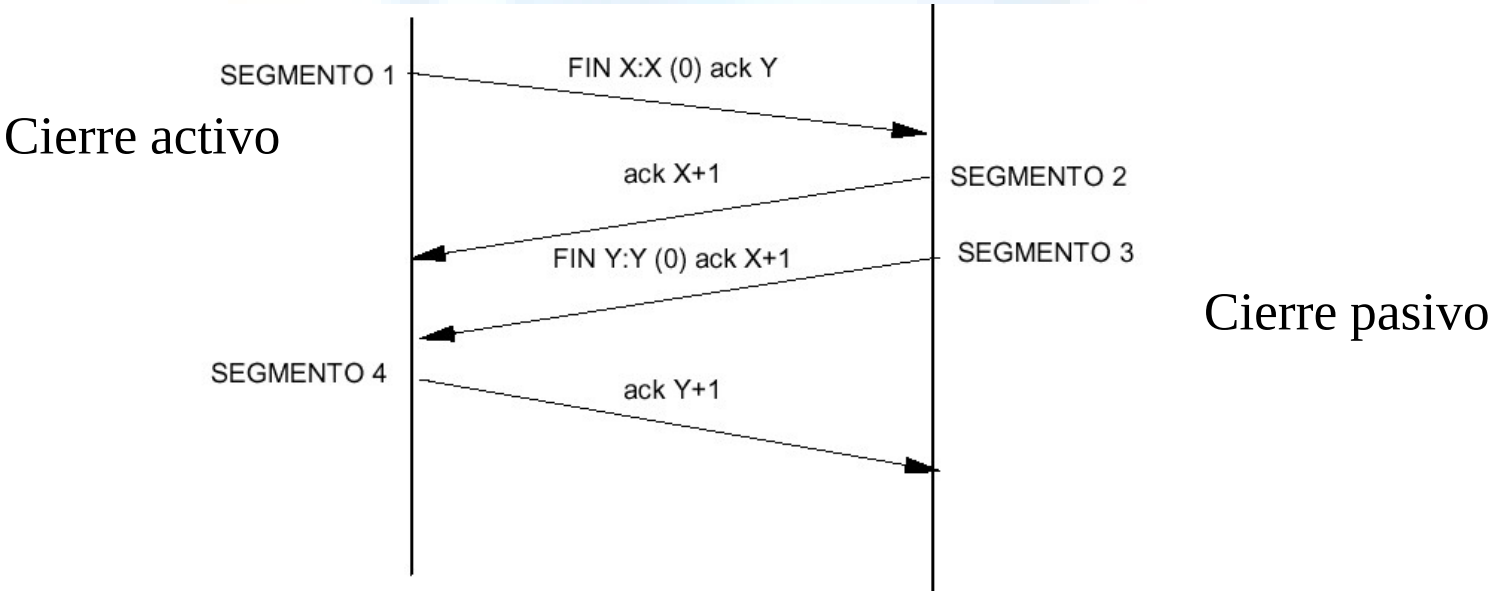
TCP: Apertura simultánea

- Dos aplicaciones efectúan una apertura activa a la vez
- Cada extremo tiene un puerto local conocido por el otro extremo
- TCP sólo establece una conexión



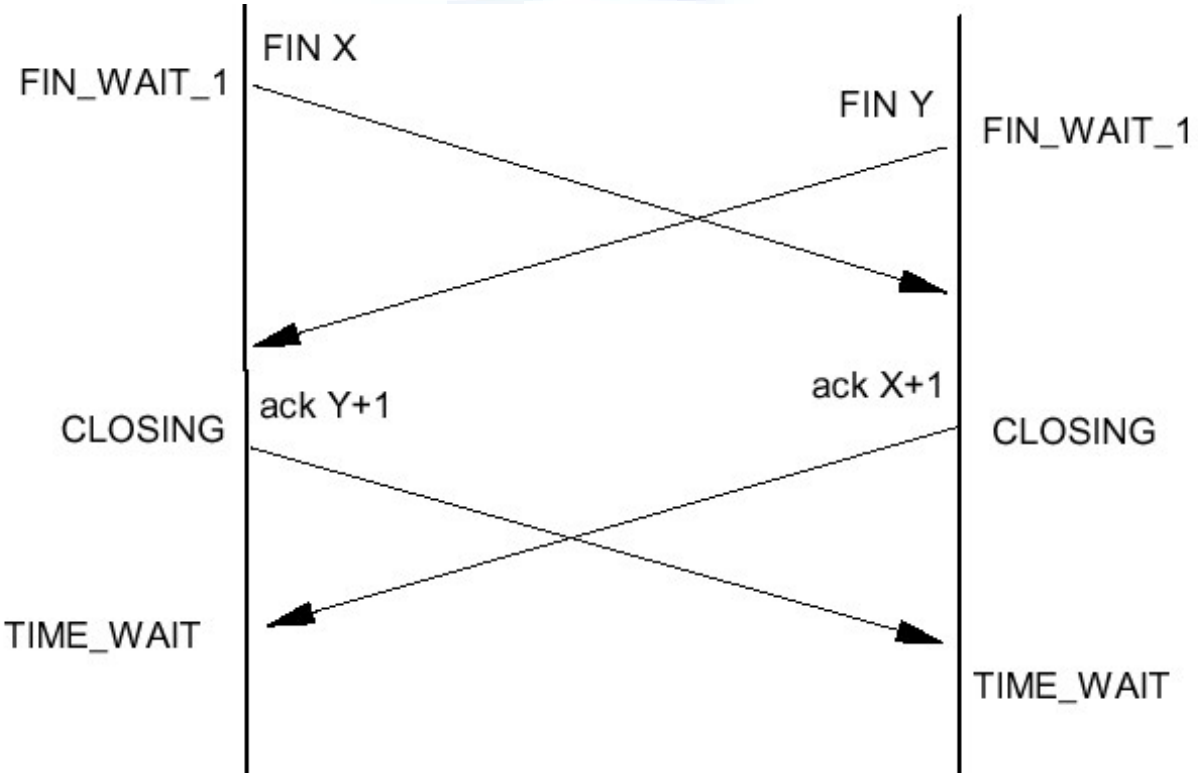
TCP: Liberación de la conexión

- Las conexiones son full-duplex ⇒ cada sentido es cerrado de forma independiente
- Son necesarios dos segmentos por cada sentido de la conexión

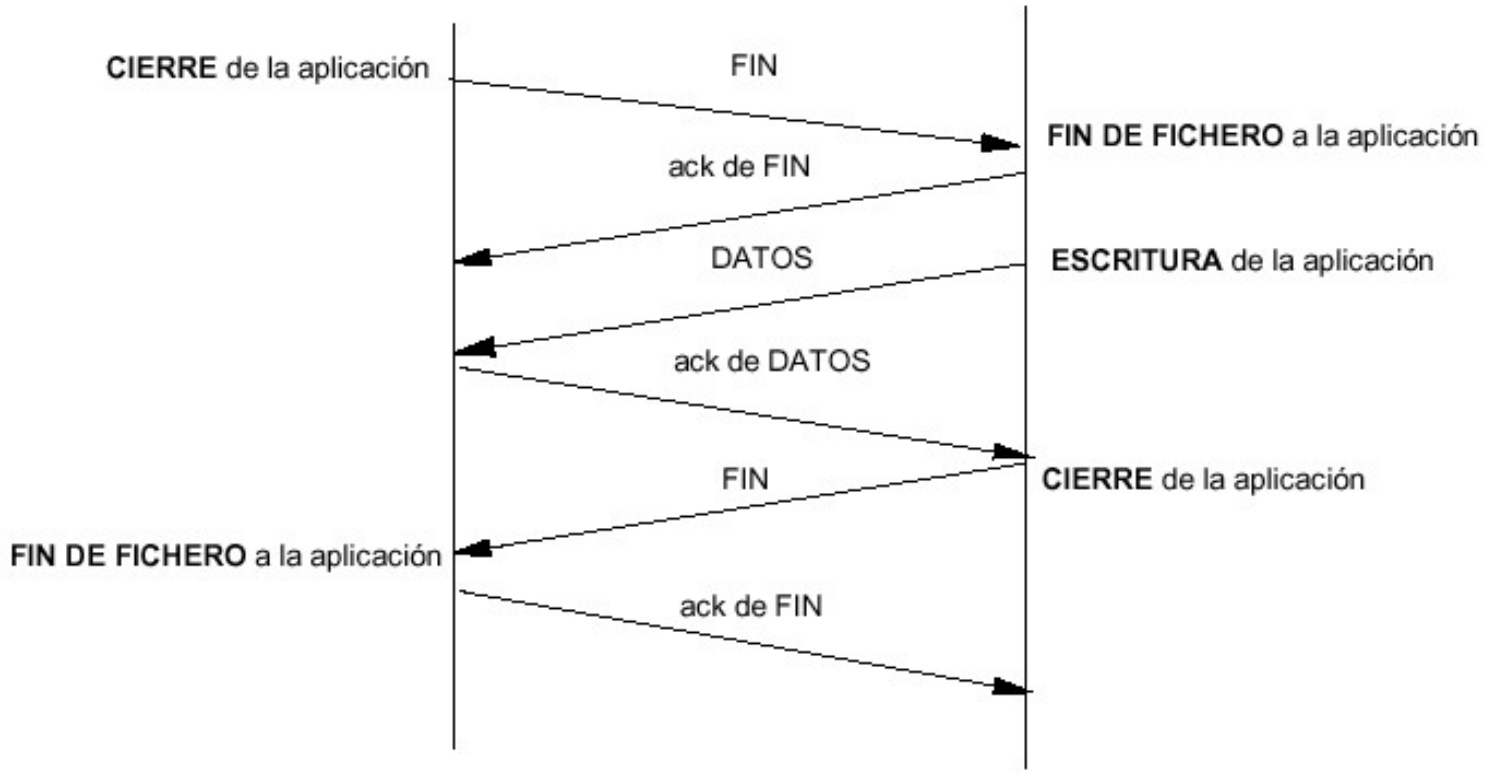


TCP: Cierre simultáneo

- Dos aplicaciones efectúan un cierre activo a la vez



TCP: Semicierre de la conexión



TCP: Problemas en el cierre

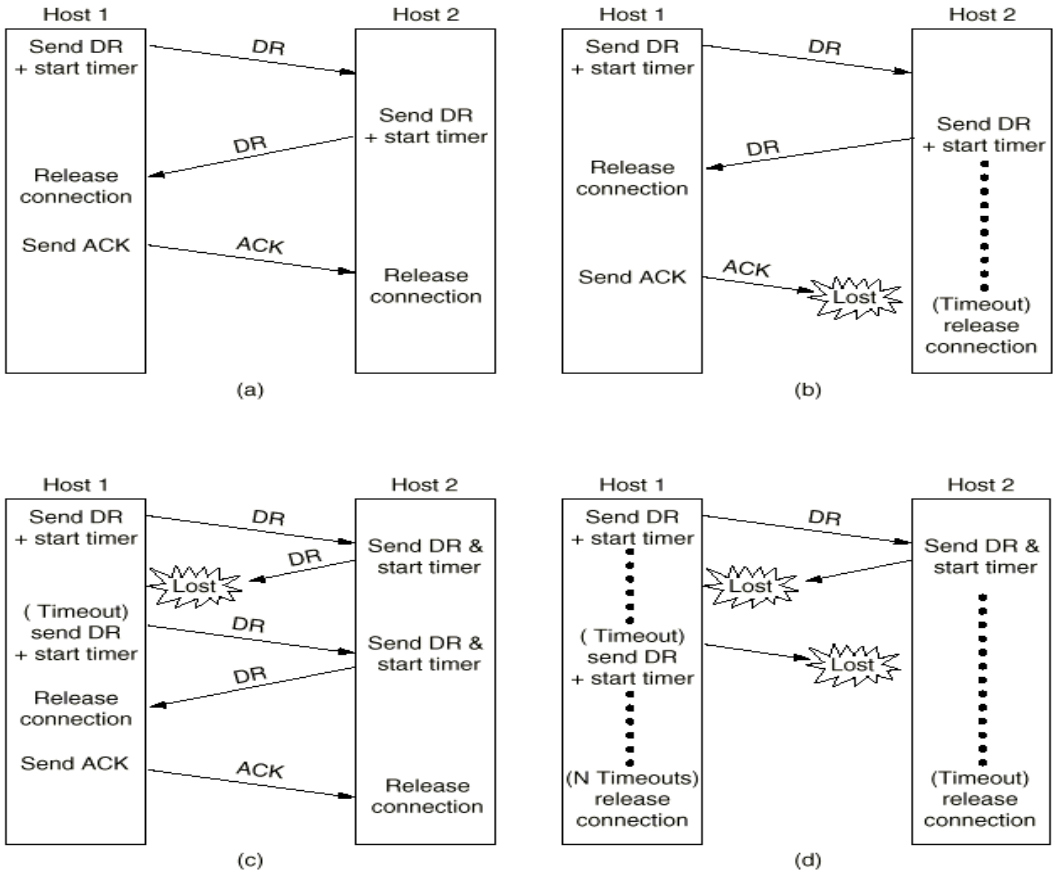
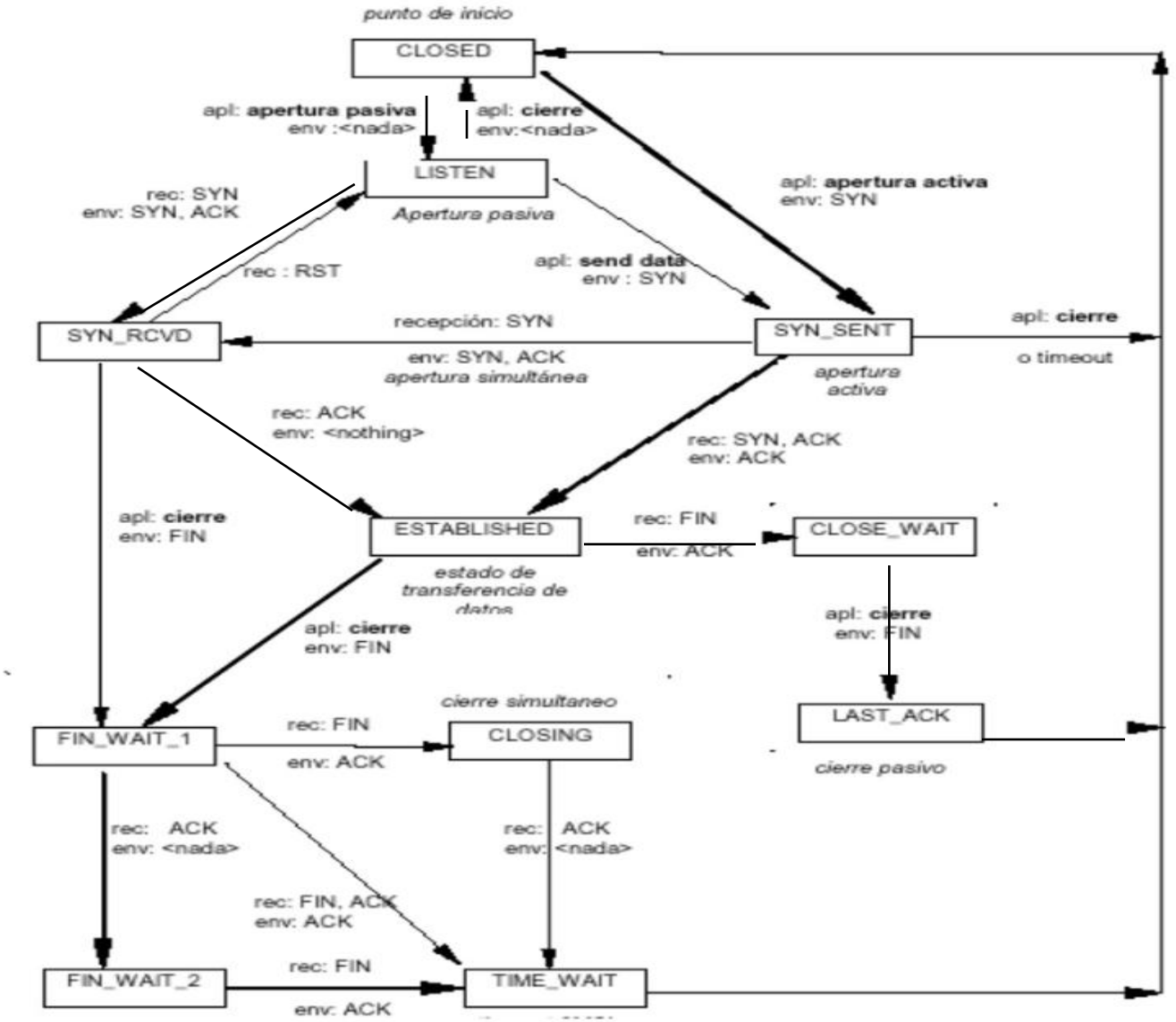


Fig. 6-14. Four protocol scenarios for releasing a connection. (a) Normal case of three-way handshake. (b) Final ACK lost. (c) Response lost. (d) Response lost and subsequent DRs lost.

TCP: Diagrama de Estados



Nota: La transición de SYN_RCVD a LISTEN solo es válida si se accedió a partir de LISTEN

TCP: Identificación de conexiones

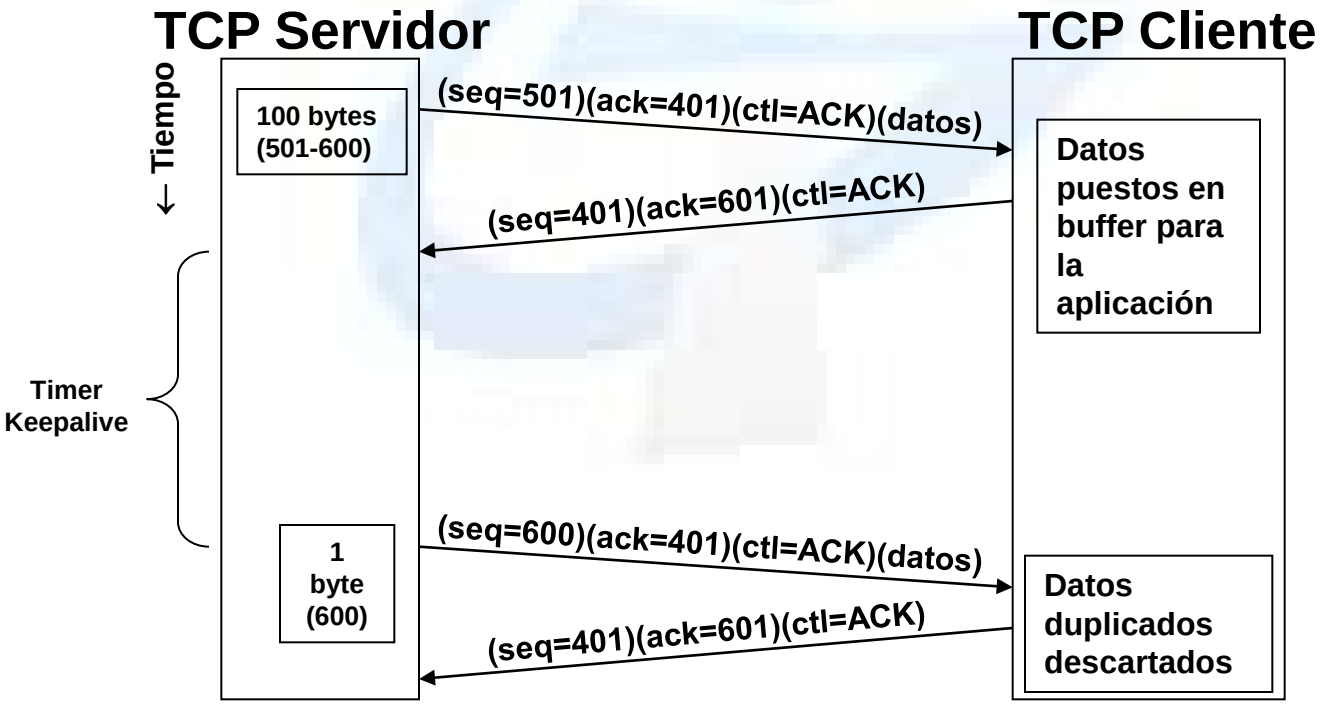
- Una vez establecida la conexión, ésta queda identificada por la dupla de sockets origen y destino
  - (193.144.201.22:3054, 61.12.199.5:80)
  - (193.144.201.23:3054, 61.12.199.5:80)
  - (193.144.201.22:3055, 61.12.199.5:80)
  - (193.144.201.22:3056, 61.12.199.5:80)
- Las conexiones son full-duplex por lo que la aplicación (origen o destino) simplemente con enviar el flujo de datos a través de su socket local, se asegura que llegará al destino correctamente.

TCP: Tamaño máximo del segmento

- TMS: Cantidad máxima de datos a enviar por TCP al otro extremo
- El valor no es negociado, cada extremo lo notifica (Por defecto 536 bits)
- Son preferibles valores elevados (evitar la fragmentación)
- Al establecer la conexión el segmento SYN incluye el TMS
- TMS_{max} = MTU – (Header IP + Header TCP)

TCP: Mensajes de keepalive

- Evitan que se queden conexiones 'medio abiertas'
- Se implementan reenviando el último byte transmitido en un segmento; el receptor descarta el dato pero devuelve un ACK
- Si se envían varios keepalive sin respuesta se considera que se trata de una conexión medio abierta y se cierra.
- Para declarar una conexión medio abierta se espera a veces hasta 2 horas.
- El tiempo de envío de los mensajes se regula con el timer de keepalive. Suele ser del orden de 2 minutos.
- El keepalive no requiere modificaciones en el TCP receptor



TCP: Segmentos de reinicio

- TCP envía un reinicio siempre que llegan segmentos incorrectos dentro del marco de la conexión referenciada

Ejm.: Petición de conexión a un puerto no asignado

- UDP envía un mensaje de puerto no alcanzable
- TCP manda un segmento RST

Liberación
- Ordenada : mediante segmento FIN
- Desordenada : mediante segmento RST

Conexión Abortada ≈ descarte de datos encolados, RST inmediato, el receptor sabe que no es un cierre

Conexión Semiabierta ≈ uno de los extremos ha cerrado o abortado sin conocimiento del otro extremo

Fallo en uno de los nodos... No es detectable !!!

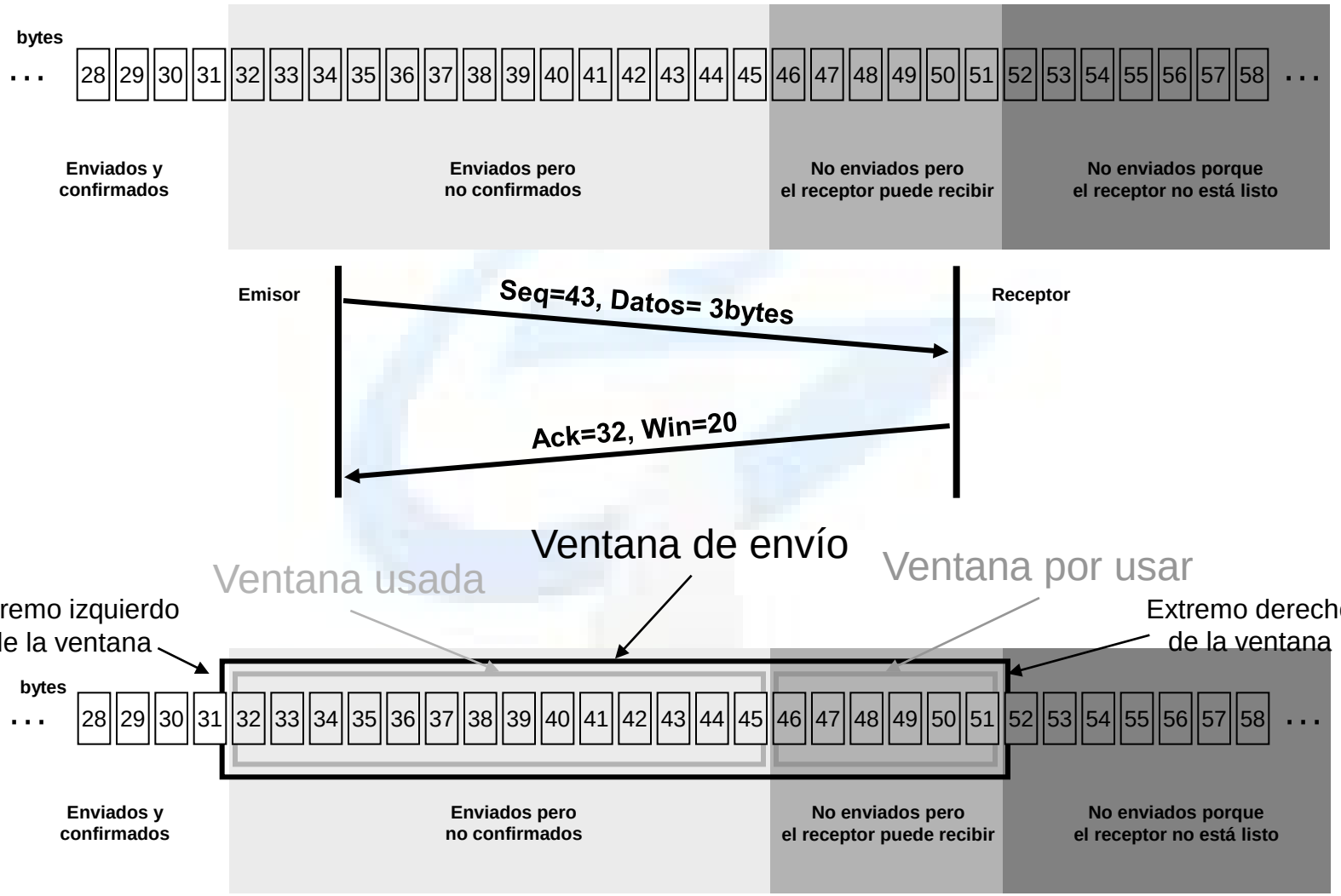
Ejm.: Desconectar los clientes en vez de terminar la aplicación y apagar el nodo: si no se estaba transfiriendo datos cuando el cliente fue desconectado el servidor no se enterará de la desconexión

Si el fallo es en el servidor, al recibir datos de los clientes obliga a RST

Comparación entre TCP y UDP

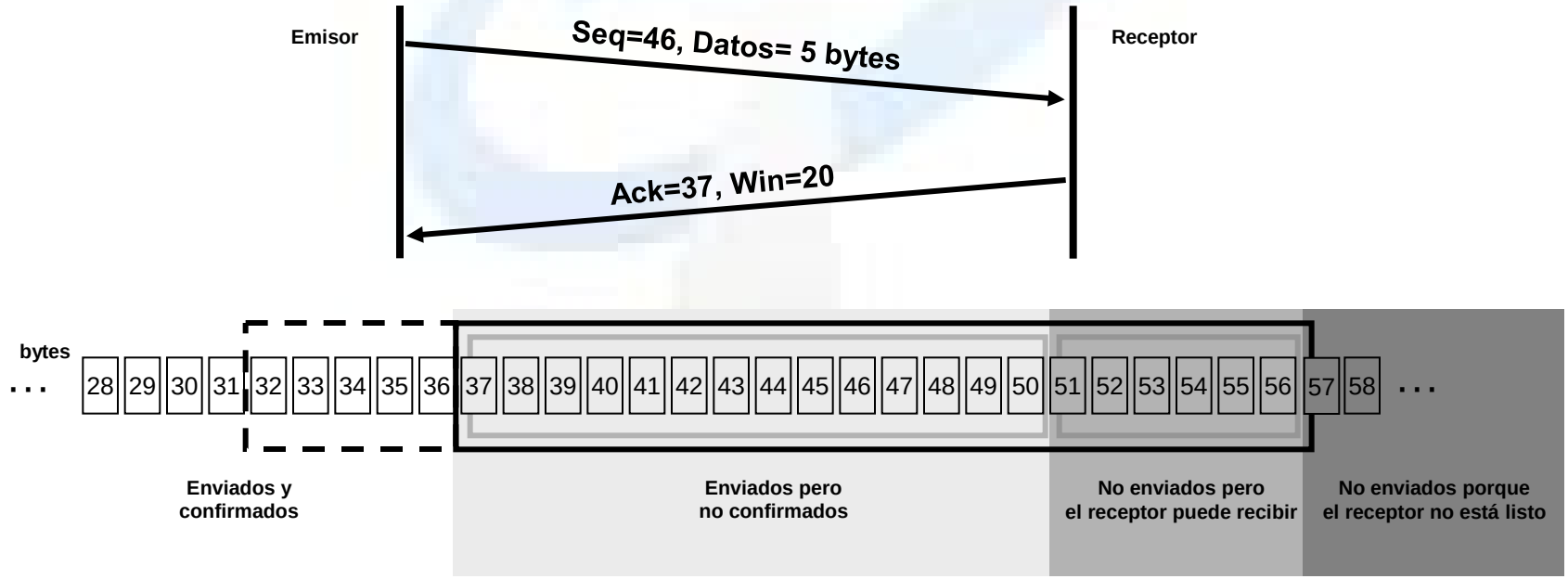
Característica	UDP	TCP
Descripción General	Simple	Complejo
Conexión	Sin conexión	Con conexión
Interfaz con la aplicación	Basado en mensajes	Basado en flujo
Fiable	Envío tipo best-effort	Asegura que todo llega
Retransmisiones	No. La aplicación se tiene que ocupar de ellos	Si
Control de flujo	No	Ventana deslizante y manejo de la congestión
Sobrecarga	Muy baja	Media
Velocidad de transmisión	Muy alta	Menor que UDP
Tipo de aplicaciones	El envío a tiempo es lo más importante	La información es lo más importante
Ejemplos de aplicaciones	Aplicaciones multimedia	FTP, HTTP, Telnet, IMAP, POP, SMTP

TCP: Control de flujo – Ventana deslizante



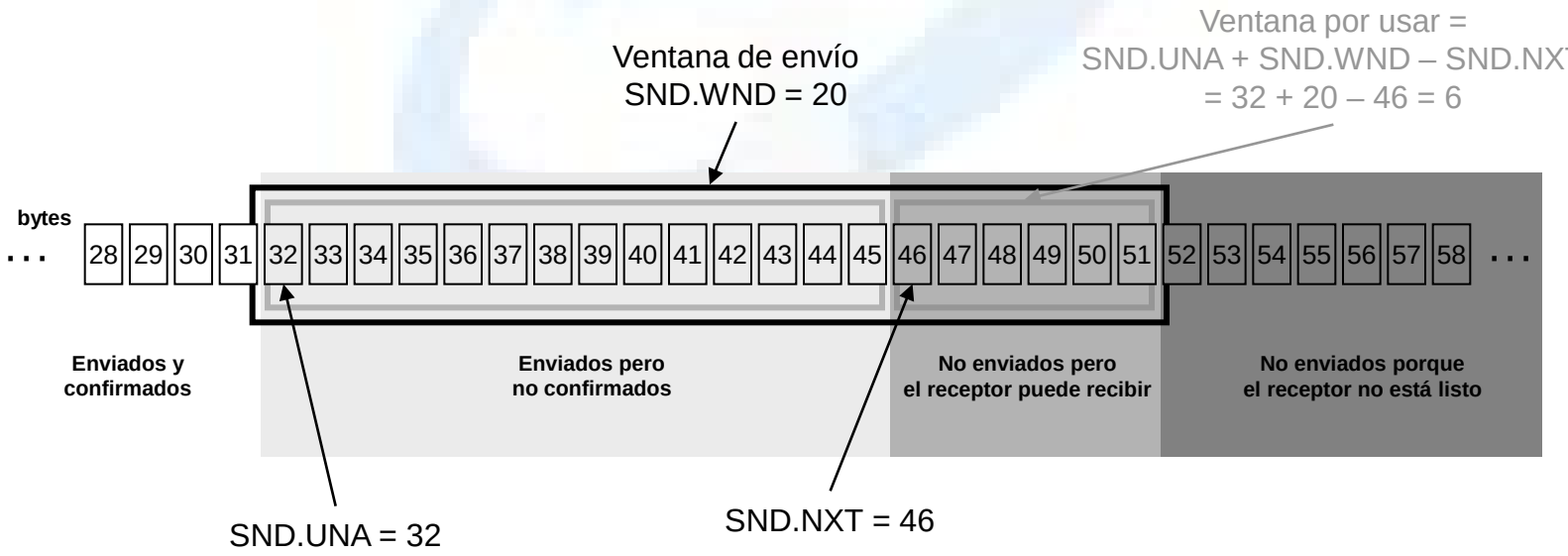
TCP: Control de flujo – Ventana deslizante

- La ventana se desplaza hacia la derecha a medida que el receptor reconoce los datos:
  - Se **cierra** a medida que el extremo izdo. avanza hacia la dcha.
  - Se **abre** a medida que el extremo dcho. avanza hacia la dcha (debido a los ACK).
  - Se **reduce** cuando la distancia entre los extremos disminuye

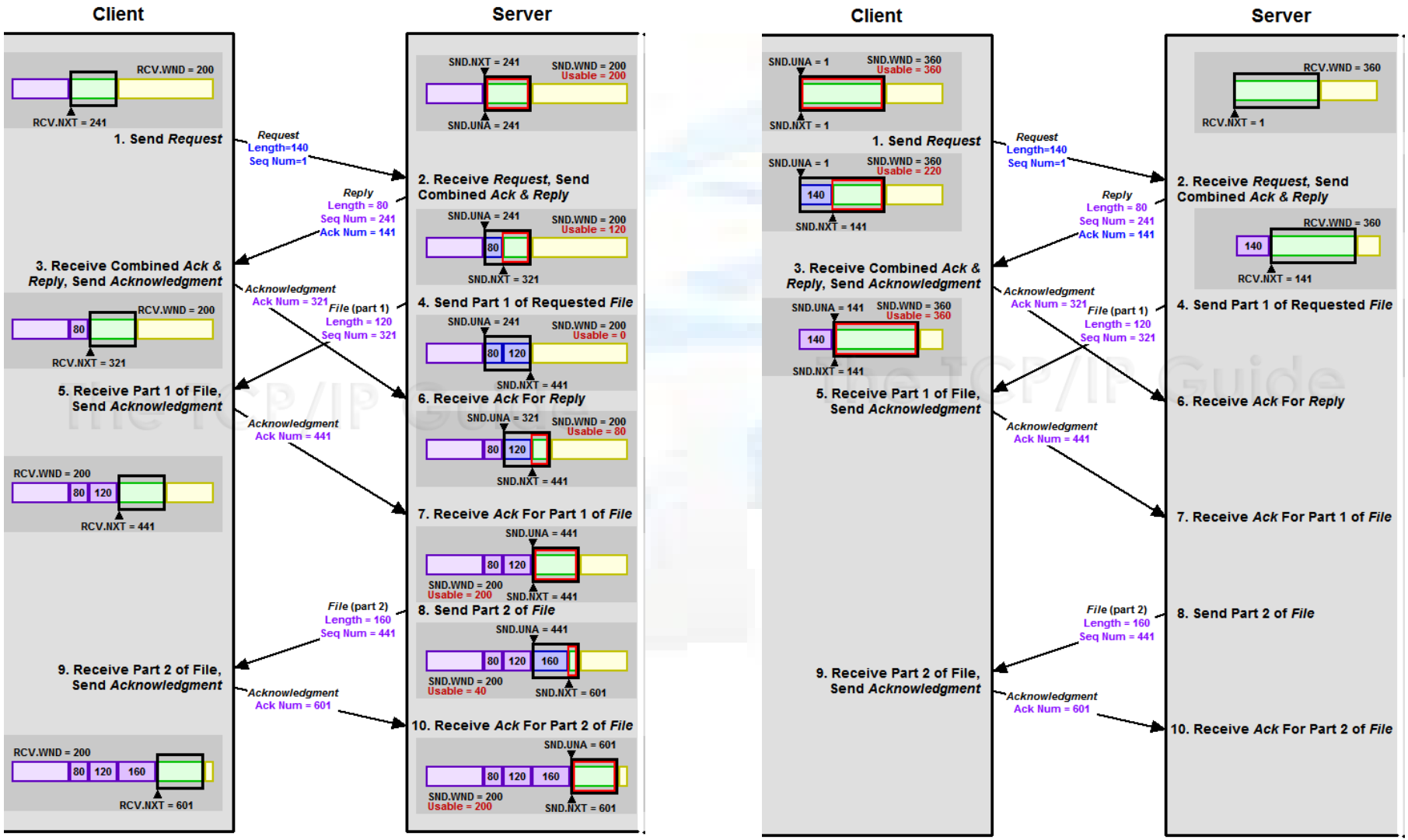


TCP: Control de flujo – Ventana deslizante

- Los procesos mantienen unos punteros tanto para su tráfico saliente
  - SND.UNA: Número de secuencia del 1er byte enviado pero no confirmado
  - SND.NXT: Número de secuencia del siguiente byte a enviar
  - SND.WND: Tamaño de la ventana anunciada por el receptor
- como para el tráfico entrante
  - RCV.NXT: Número de secuencia del siguiente byte que se espera recibir
  - RCV.WND: Tamaño de la ventana que se anuncia al emisor

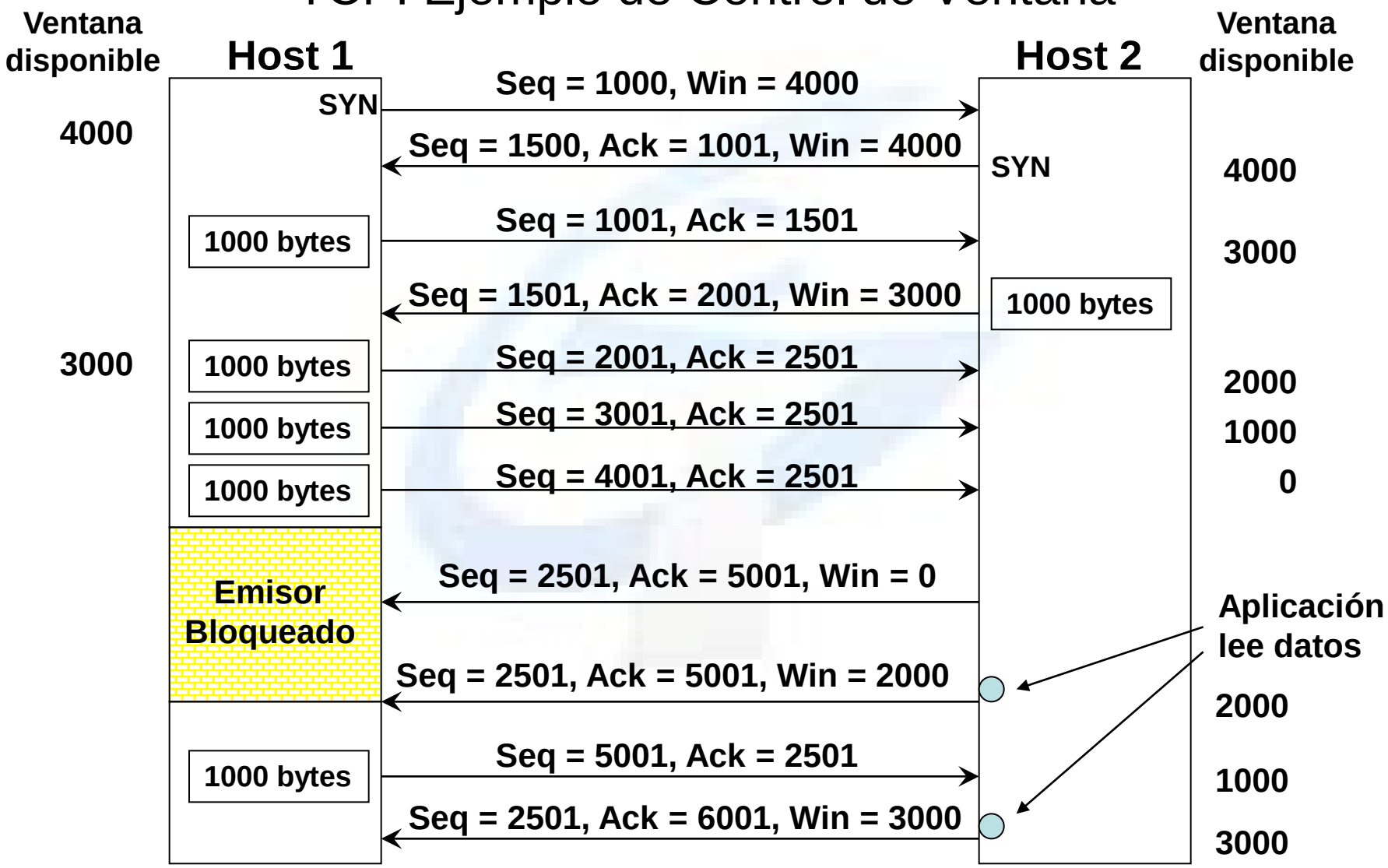


TCP: Ejemplo de ventana deslizante*



* http://www.tcpipguide.com/

TCP: Ejemplo de Control de Ventana



TCP: Sindrome de la ventana tonta

Supongamos una aplicación que genera datos con rapidez y los envía a otra, que los recoge del buffer TCP a razón de 1 byte cada vez:

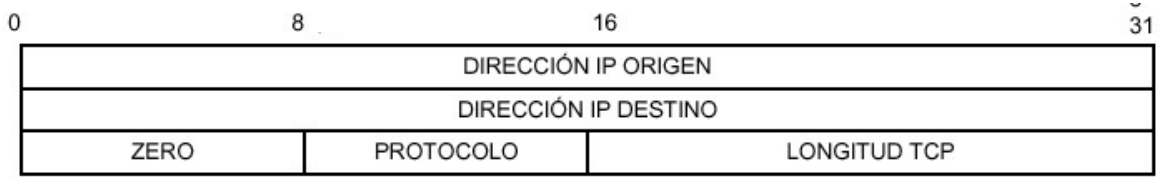
1º El buffer de TCP receptor se llena.
2º El TCP receptor notifica al emisor que su ventana está cerrada (ventana 0).
3º La aplicación receptora lee 1 byte del buffer de TCP.
4º El TCP receptor envía un ACK al emisor para anunciarle que dispone de 1 byte de espacio.
5º El TCP emisor envía un segmento con 1 byte de información útil.
6º Volvemos al punto 1 hasta que se termine la sesión.

Algoritmo de Clark

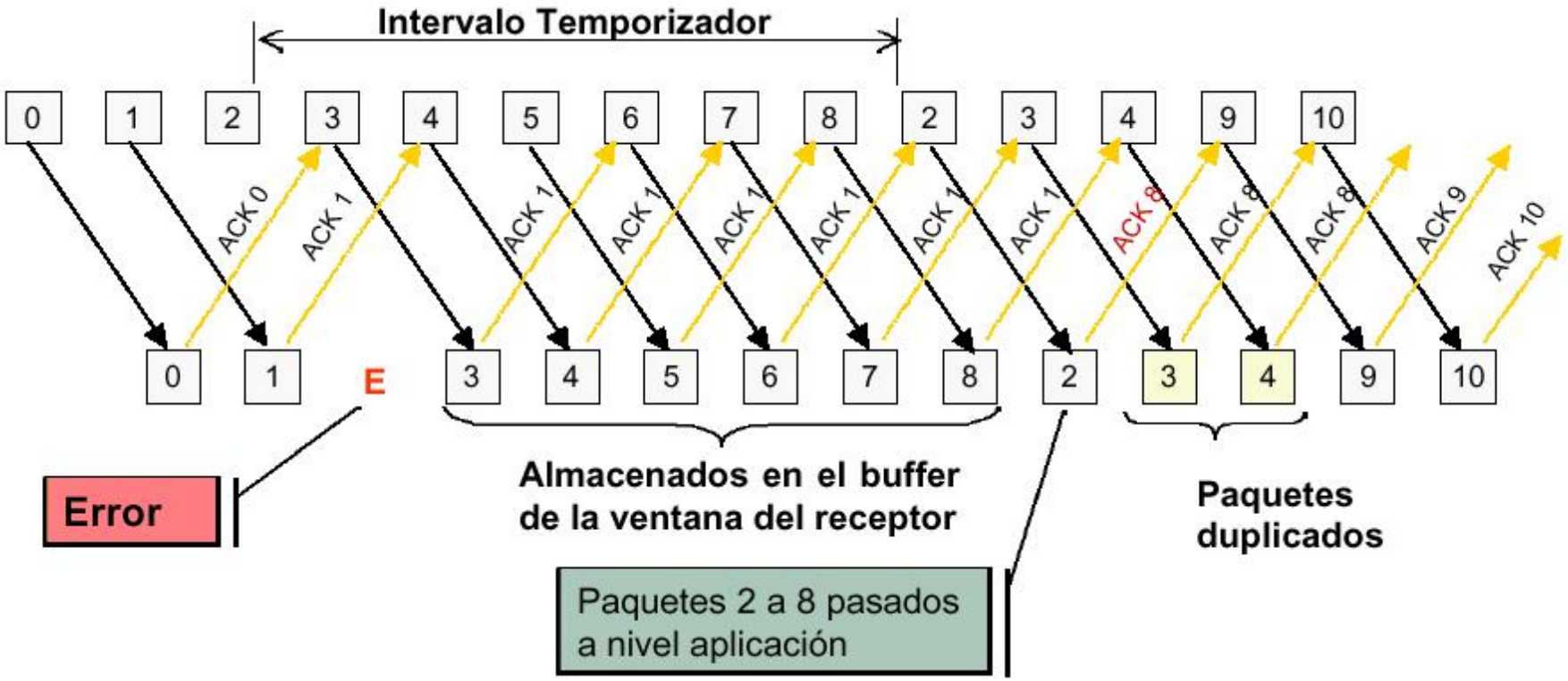
RFC 813: *El TCP receptor no debe notificar el cambio de ventana al emisor mientras no tenga espacio suficiente en su buffer (longitud máxima admitida en la conexión, o la mitad del espacio total del buffer)*

TCP: Control de errores

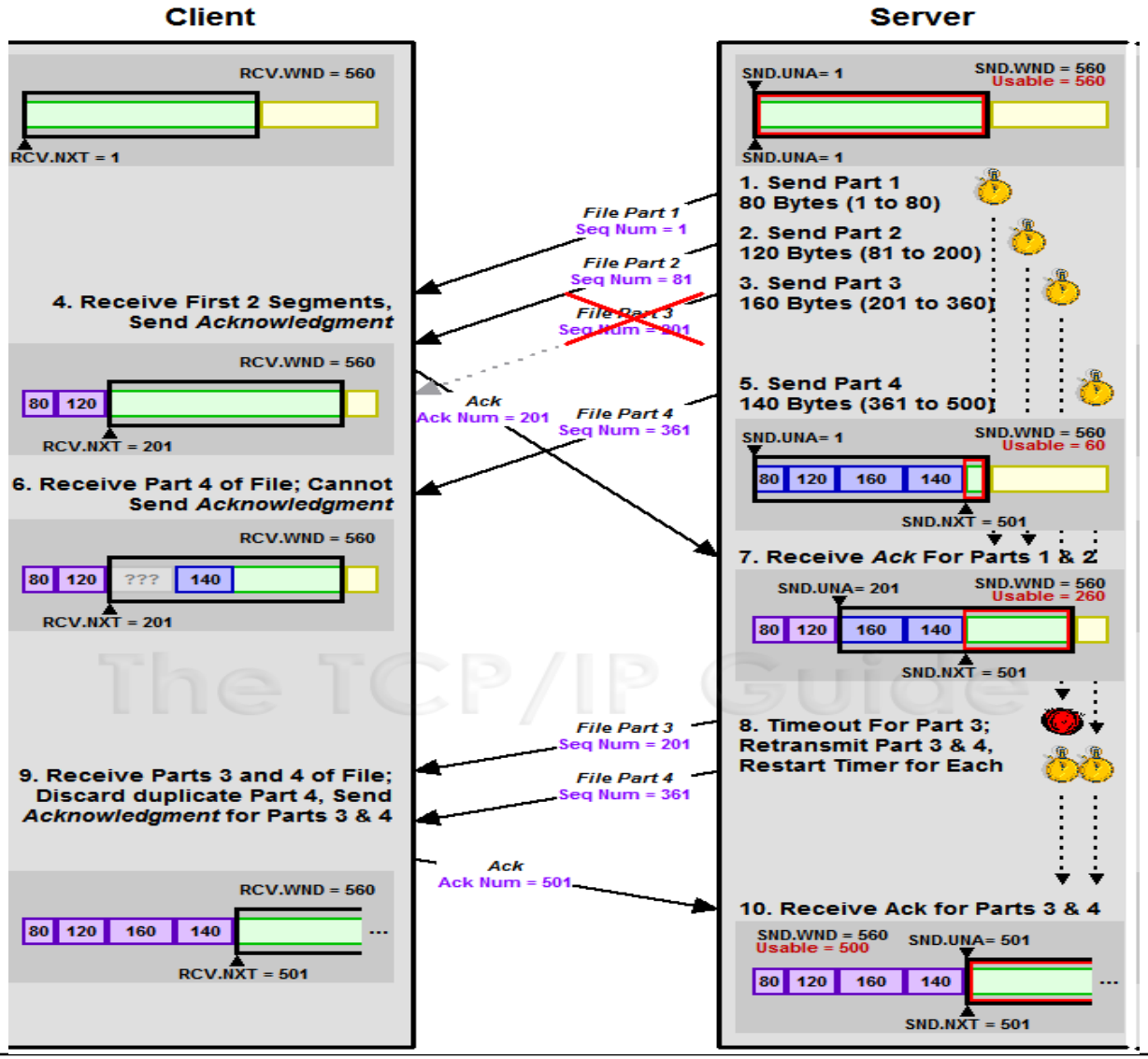
• TCP calcula el checksum tanto de la cabacera como de los datos haciendo uso de una pseudocabecera:



TCP: Reenvío de segmentos

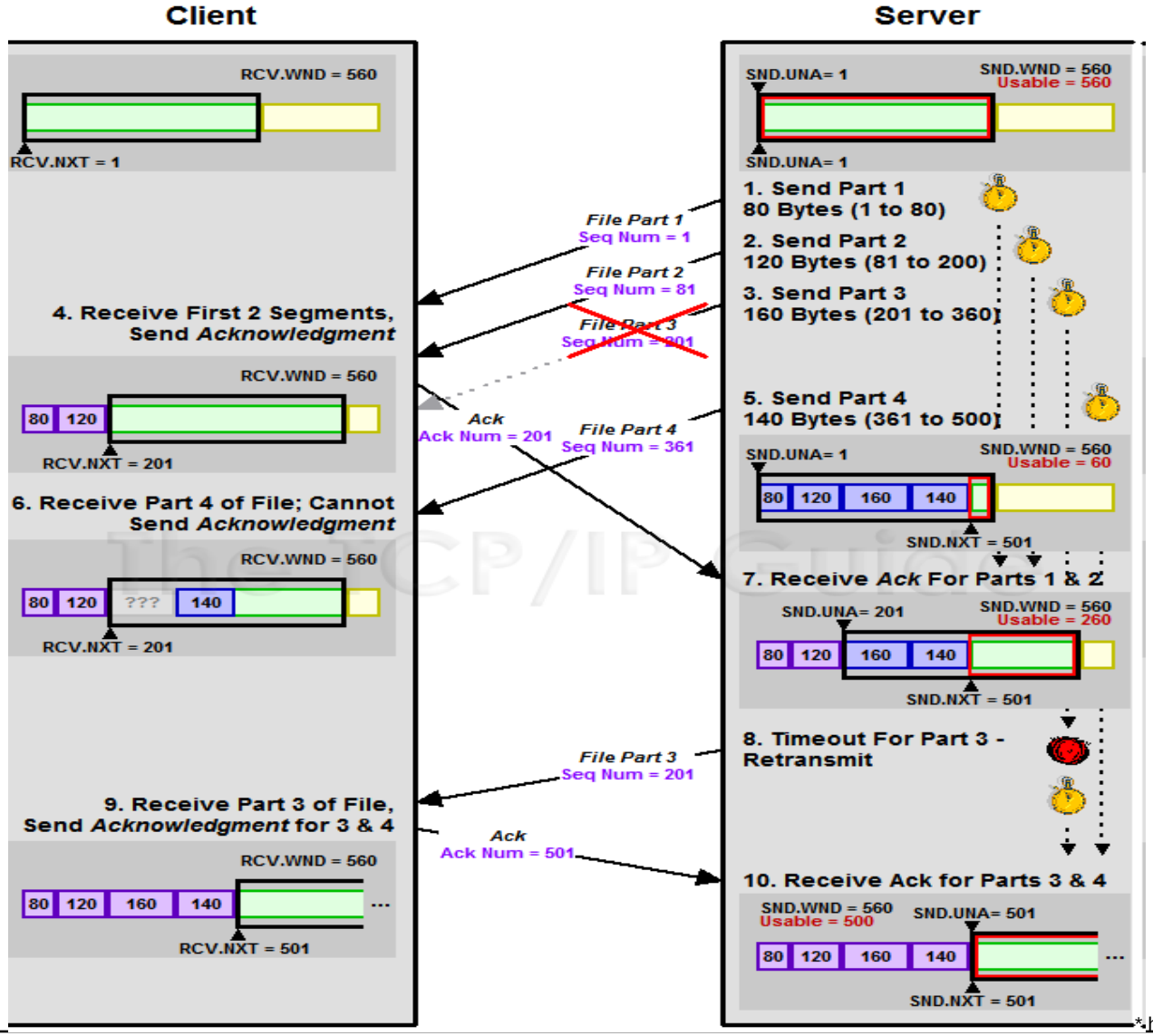


TCP: Retransmisión con retroceso N (Go-Back-N)



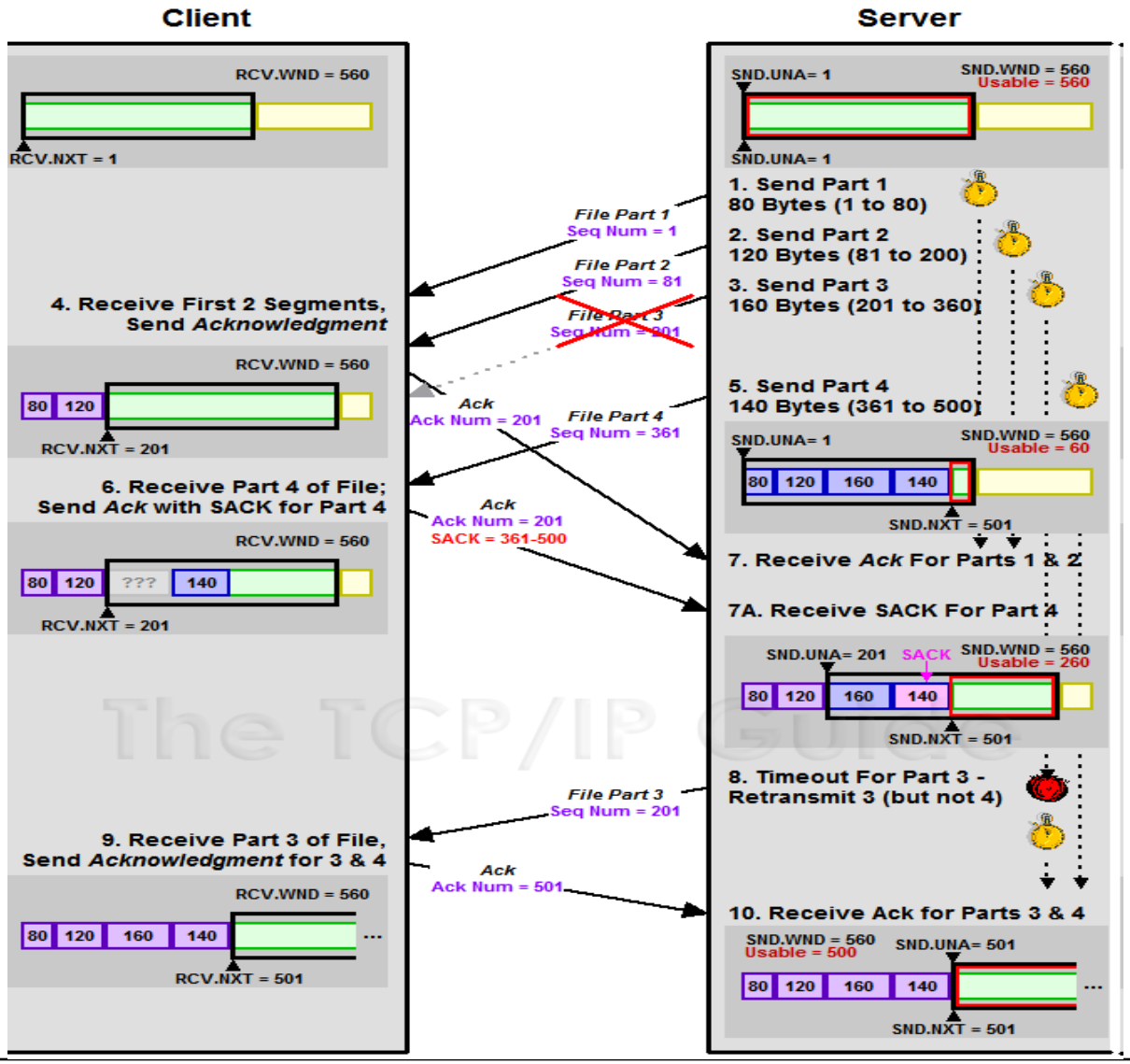
* http://www.tcpipguide.com/

TCP: Retransmisión con repetición selectiva



*: http://www.tcpipguide.com/

TCP: Retransmisión con reconocimiento selectivo (SACK)



* http://www.tcpipguide.com/

TCP: Medición del tiempo de retorno

- Las modificaciones de rutas y de tráfico cambian a lo largo de una conexión
- Es necesario seguir los cambios y modificar los "timeout"
- TCP debe medir el tiempo de retorno entre un byte enviado con un NS dado y la recepción del ACK correspondiente

TCP (RFC 2988) estima el tiempo de retorno mediante un filtro paso bajo:

$$MRTT = \alpha * MRTT + (1-\alpha)RTT$$

Mean round trip time

α Factor de corrección $\approx 0,9$ { 90% del anterior / 10% del nuevo

El RFC793 recomienda un tiempo de retransmisión:

$$RTO = R * \beta$$

β Factor de variancia ≈ 2

Jacobson

Otras estimaciones tienen en cuenta la desviación estándar:

$$D = \delta D_{viejo} + (1-\delta)|MRTT_{viejo} - RTT|$$

$\delta \approx 0,75$

$$RTO = MRTT + 4 * D$$

TCP: Gestión de las retransmisiones

- No se usan los segmentos retransmitidos para el cálculo del RTO
- Para evitar sobrecargar la red con las retransmisiones se aumenta el RTO exponencialmente ($2^{(X-1)} * RTO$)
- Límite máximo es típicamente 2 minutos

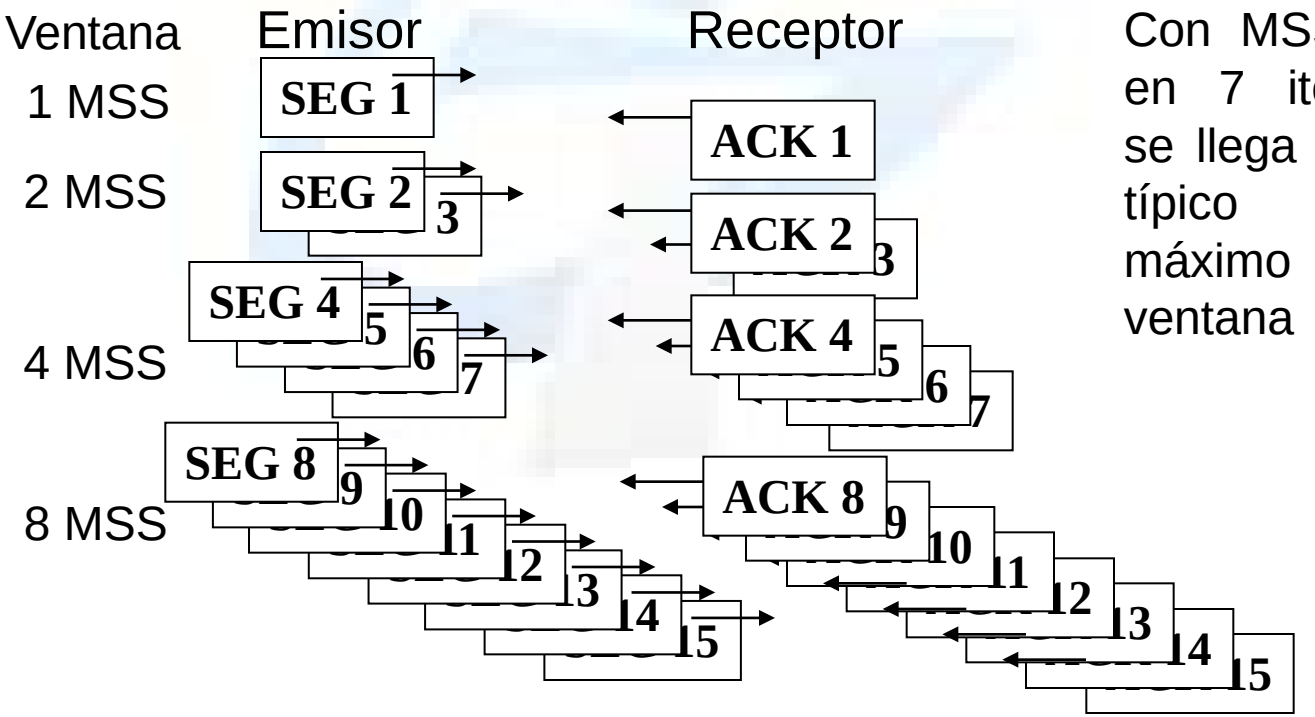
Algoritmo de Karn

TCP: Gestión de la Congestión

- Hasta ahora todos los mecanismos que hemos visto obedecen a requerimientos impuestos por los extremos de la conexión
- Es necesario no olvidar que entre ellos hay una red que puede ser muy compleja
- La congestión se produce en la red cuando tiene demasiados segmentos que gestionar
- Reducir el caudal para aliviar a la red es la solución que implementa TCP
- Pero si no hay congestión, cuanto más rápido mejor

TCP: Slow Start (primera fase)

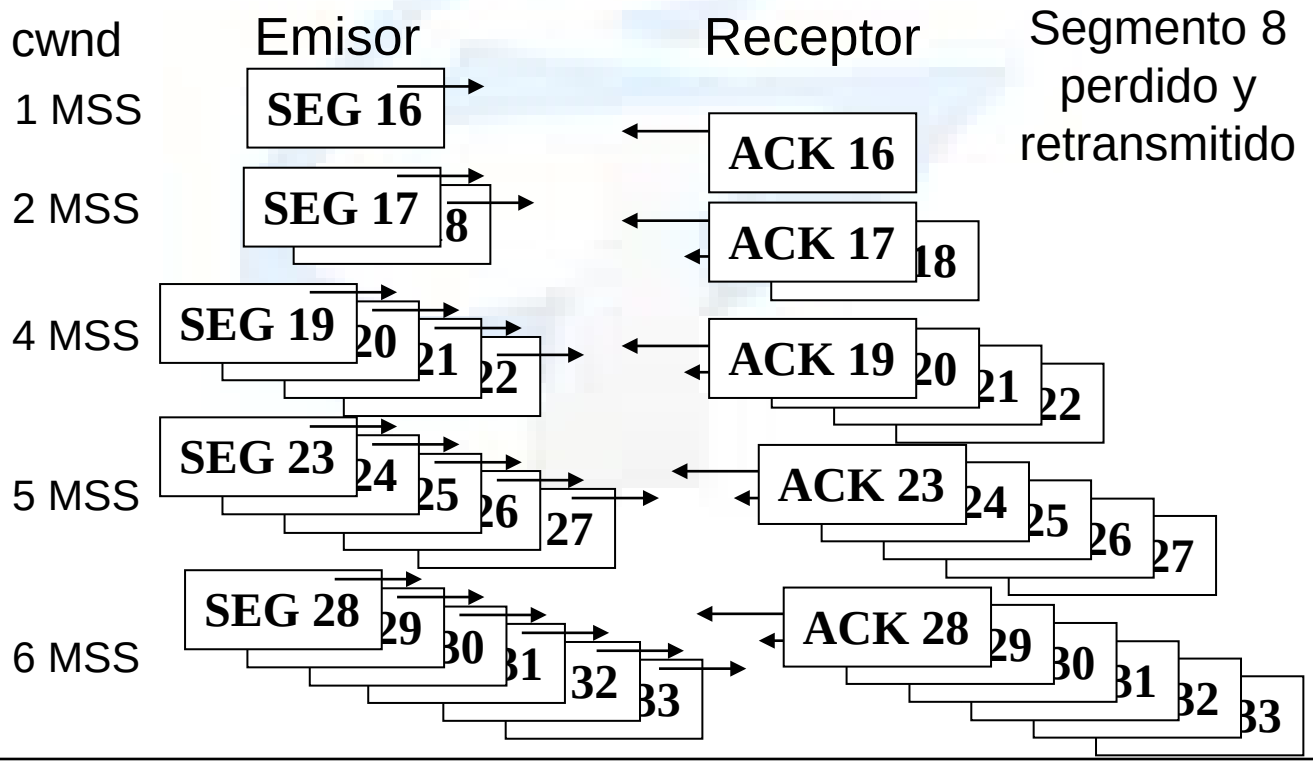
- Inicialmente la Ventana de Congestión (cwnd) tiene el tamaño de un MSS (Maximum Segment Size)
- Por cada segmento enviado con éxito la ventana se amplía en un MSS.
- En la práctica esto supone un crecimiento exponencial en potencias de dos.
- Si la ventana de congestión supera a la de control de flujo se aplica ésta con lo cual aquella deja de crecer



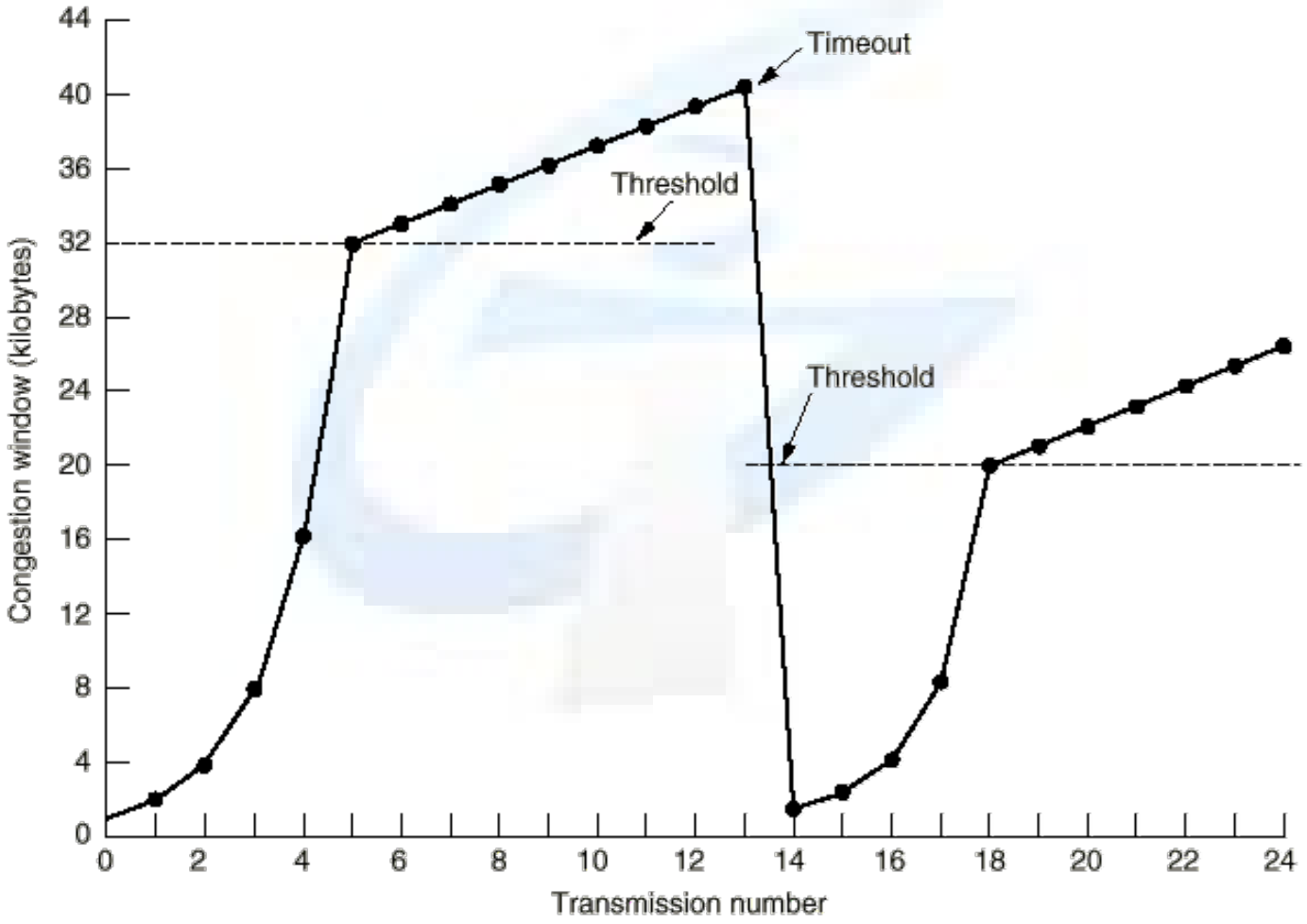
Con MSS = 1KB en 7 iteraciones se llega a 64 KB, típico tamaño máximo de la ventana

TCP: Congestion Avoidance

- Cuando se detecta la congestión:
  - Se fija un 'umbral de peligro' (ssthresh – Slow Start Threshold) en un valor igual a la mitad de la cwnd siempre y cuando este sea mayor que 2. Esto es, ssthresh = max(cwnd/2, 2)
  - Si la congestión se debe a la pérdida de un segmento, la cwnd se fija a 1MSS.
  - La cwnd crece como antes hasta el umbral de peligro; a partir de ahí crece en sólo un segmento cada vez

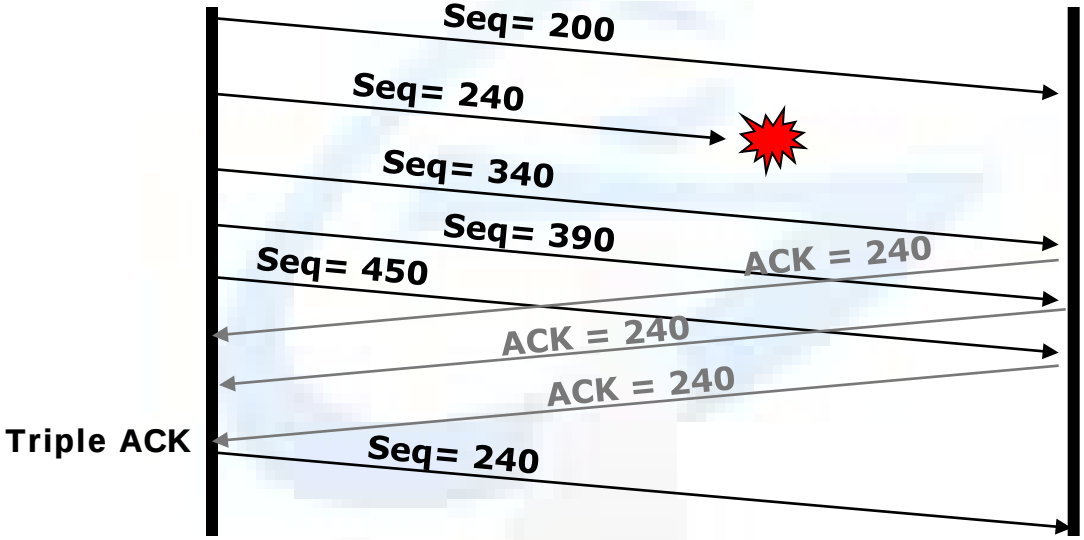


TCP: Congestion Avoidance



TCP: Fast Retransmit

- Se pueden hacer las retransmisiones no sólo por expiración del temporizador sino por la recepción de varios (3) ACKs del mismo segmento



- La recepción de ACKs duplicados también es indicativo de congestión.

TCP: Fast Recovery

- Si se ha lanzado una retransmisión por la recepción de 3 ACKs del mismo segmento (Fast Retransmit), se dispara el procedimiento de Congestion Avoidance.
- No se usa Slow Start puesto que los ACKs recibidos indican que los siguientes segmentos han llegado y por lo tanto la congestión ha desaparecido.
- En el mecanismo de Congestión Avoidance, la cwnd no se fija a 1 MSS como en el caso de Slow Start sino que se fija a:
  - **cwnd = ssthresh + 3 MSS**

- Cuando se reciba el ACK del segmento retransmitido, cwnd pasa a ssthresh y se retoma el crecimiento lineal de la cwnd.

TCP: Congestión vs Errores en el canal

- TCP/IP es una arquitectura que se ideó hace más de 20 años
- TCP ha sido capaz de superar el crecimiento de Internet
- Ante la aparición de las redes inalámbricas las cosas se le han complicado
  - Errores en el canal asumidos como congestión conlleva una infrautilización de los recursos

TCP: Ejercicios (I)

- Un cliente quiere descargarse el fichero “fich.txt” (6000 Bytes) de un servidor para lo cual emplea una aplicación que utiliza TCP como protocolo de transporte.
  - La petición del fichero se hace enviando al servidor un string compuesto por la palabra clave GET seguida del nombre del fichero (“GET fich.txt”). La recepción de esta petición inicia la descarga del fichero en cuestión.
  - Una vez recibido el fichero, el cliente manda otro string “TRANSMISION OK” y cierra la conexión.
- Se pide representar los segmentos TCP (indicando en cada uno de ellos que “Flags” estarán activados; los valores de los campos “Número de Secuencia” y “Número de Reconocimiento” si son representativos; el valor del campo de la “Ventana de envío”; y la cantidad de datos que contiene el segmento) que se intercambian entre ambos extremos teniendo en cuenta lo siguiente:
  - Los dispositivos sólo transmiten segmentos al principio del tic de reloj.
  - Los segmentos tardan en llegar al destino medio tic de reloj.
  - Ambos reconocerán en cuanto puedan los segmentos recibidos.
  - El Tamaño Máximo de Segmento (TMS – MSS) para ambos es 1440 bytes.
  - El Servidor siempre publica una ventana de envío de 4000 Bytes.
  - El Cliente siempre publica una ventana de envío de 2700 Bytes.
  - Ambos extremos implementan mecanismos de Control de la Congestión.

TCP: Ejercicios (I)

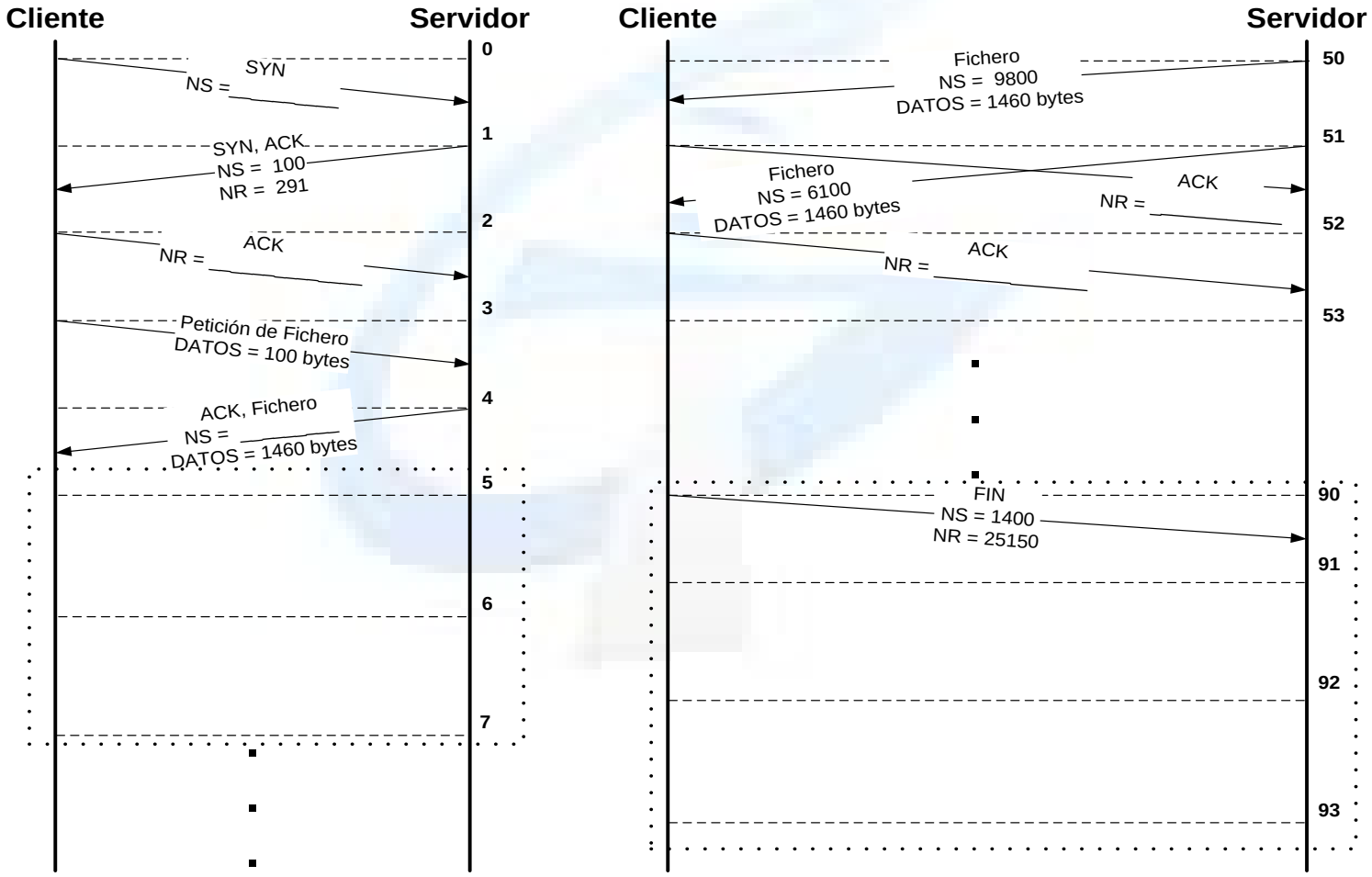
Cliente

Servidor

TCP: Ejercicios (II)

- En la siguiente figura se muestra el proceso de descarga de un fichero en diferentes instantes de tiempo. Se pide:
  - Completar los datos que faltan (NS, NR) allí donde está indicado (Ej. NS = _____)
  - Completar los segmentos TCP que no se han incluido en los periodos marcados por los recuadros punteados (entre los tics de reloj 5 a 7 y 90 a 93). Indicar tanto el tipo de segmento del que se trata (Ej. Datos (27 bytes), ACK+SACK, FIN, etc.) como los campos de NS y NR de dichos segmentos.
- Se debe tener en cuenta lo siguiente:
  - Sólo se transmiten segmentos coincidiendo con el tic de reloj y los segmentos tardan en llegar medio tic de reloj.
  - Las máquinas enviarán datos siempre que puedan y enviarán asentimientos cada vez que reciban un segmento con datos. El TMS es de 1460 bytes.
  - Ambos dispositivos publican una ventana de control de flujo de 65536 bytes.
  - Se emplean los mecanismos de Slow Start y Congestion Avoidance.

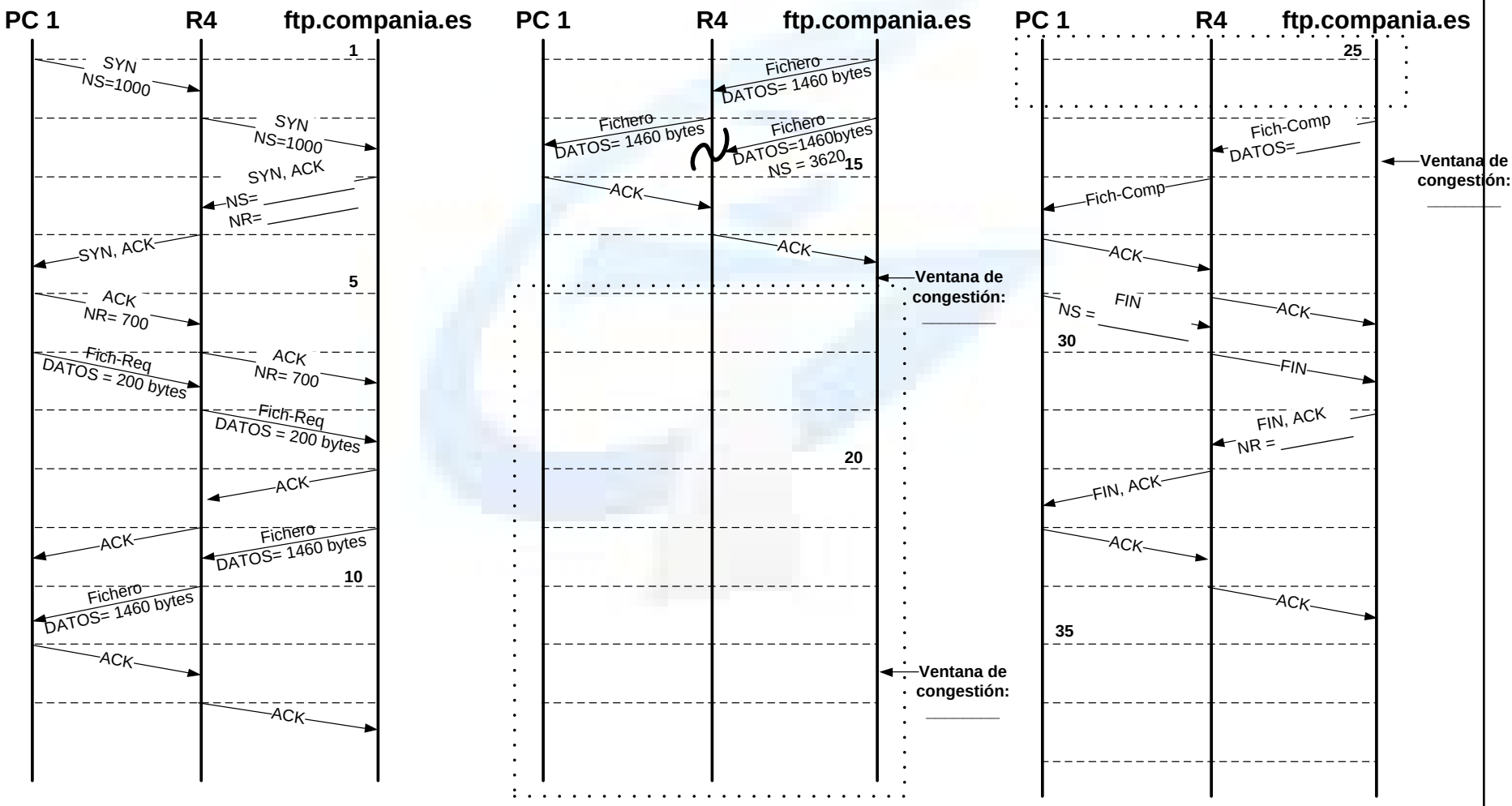
TCP: Ejercicios (II)



TCP: Ejercicios (III)

- En la figura siguiente se muestra el proceso de descarga de un fichero desde el servidor ftp.compania.es. Se pide **completar tanto los datos que faltan (NS, NR, cwnd) allí donde está indicado (Ej. NS = _____), como los segmentos TCP que no se han incluido en el periodo marcado por el recuadro punteado** (entre los tics de reloj 17 a 26). Al completar los segmentos intercambiados, únicamente es necesario indicar el tipo de segmento del que se trata (Ej. Datos (27 bytes), ACK, FIN, etc.) y para los segmentos de datos cuantos bytes de datos contienen.
- Se debe tener en cuenta lo siguiente:
  - El fichero que se quiere descargar ocupa 7000 bytes. Todos los datos en los segmentos de datos enviados por el servidor corresponden con bytes del fichero.
  - Sólo se transmite un segmento (independientemente de que sea de control o de datos) coincidiendo con el tic de reloj y los segmentos tardan en llegar medio tic de reloj.
  - Las máquinas enviarán datos siempre que puedan y enviarán asentimientos cada vez que reciban un segmento con datos. El TMS es de 1460 bytes.
  - El plazo de retransmisión de segmentos no reconocidos inicial (RTO) es de 13 tics de reloj.
  - Ambos dispositivos publican una ventana de control de flujo de 3920 bytes.
  - Se emplean los mecanismos de Slow Start, Congestion Avoidance, Fast Retransmit y Fast Recovery.

TCP: Ejercicios (III)



Servicios a la capa de aplicación

- Las aplicaciones utilizan los servicios de TCP/IP.
  - Servicio con conexión: TCP.
  - Servicio sin conexión: UDP.

    - Red:
      - Transfiere bits
      - **Opera bajo petición de las aplicaciones**
    - Aplicaciones determinan:
      - Qué enviar
      - Cuándo
      - Dónde

Servicios a la capa de aplicación: Modelo cliente-servidor

- Una aplicación:
  - Empieza la ejecución 1º
  - Espera pasivamente en un puerto fijo
- Otra aplicación
  - Empieza la ejecución despues
  - Establece contacto con la 1ª aplicación
  - Esto es la interacción Cliente -Servidor
- Programa pasivo: Servidor
- Programa activo: Cliente
- La información fluye en ambos sentidos, normalmente:
  - el cliente envía petición al servidor y el servidor responde.

Modelo cliente-servidor

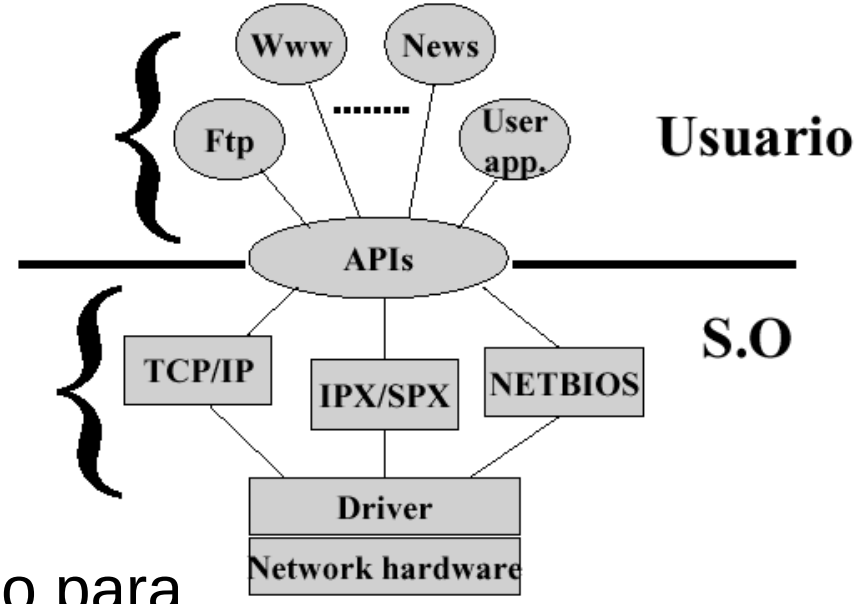
- CLIENTE
  - Programa de aplicación arbitrario
  - Se convierte en cliente de forma temporal
  - Puede realizar otros cálculos
  - Lo invoca directamente el usuario
  - Se ejecuta localmente en el computador usuario
  - Inicia la comunicación con el servidor
  - Se comunica sólo con un servidor cada vez
- SERVIDOR
  - Programa de propósito específico que se ejecuta en modo privilegiado.
  - Dedicado a proporcionar un servicio.
  - Puede manejar múltiples clientes simultáneamente.
  - Inicia la ejecución durante el arranque del sistema.
  - Se ejecuta de forma permanente.
  - Espera pasivamente la llamada del cliente.
  - Acepta peticiones de clientes arbitrarios

Modelo cliente-servidor: API

- Clientes y servidores utilizan protocolos de transporte.

El software de las aplicaciones reside en el espacio de usuario

El software del protocolo forma parte del S.O.



- Necesitamos un mecanismo para unirlos

API *(Application Program Interface)*

Modelo cliente-servidor : API socket

API

- Forma parte del S.O.
- Permite a las aplicaciones utilizar los protocolos.
- Define:
  - Las operaciones permitidas
  - Los argumentos de cada operación

- API Socket
  - Originalmente diseñado:
    - Por BSD *(Berckeley Standart Distribution)*Unix
    - Sigue la filosofía *open-read-write-close* del I/O de Unix para utilizar la pila TCP/IP
  - Estandar de la industria
  - Disponible en muchos S.O.

Modelo cliente-servidor: Socket

- Es una abstracción del S.O. (no HW)
- Creado dinámicamente
- Persiste sólo mientras se ejecute la aplicación.
- Se referencia mediante un descriptor

```
descriptor = socket (int family, int type, int protocol);
```

- **Crea un socket** de un determinado protocolo de una familia de protocolos residente en el host.

```
#include <sys/types.h>
#include <sys/socket.h>

int socket (int family, int type, int protocol);

Family: PF_INET, PF_UNIX, PF_XNS, PF_APPLETALK, etc.
Type: SOCK_STREAM (TCP), SOCK_DGRAM (UDP), SOCK_RAW (ICMP,IP).
Protocol: IP_PROTO_TCP, IP_PROTO_UDP, IP_PROTO_ICMP,
          IP_PROTO_RAW.
```

Es completamente general
• Puede utilizarse:
– Por el cliente
– Por el servidor
– Con TCP
– Con UDP
– Para enviar y/o recibir datos

Ejm.:

```
Int sd;  /* variable que guardará el descriptor del socket */

sd = socket (PF_INET, SOCK_DGRAM, 0);
```

Modelo cliente-servidor: Bind

- **Asignación de dirección local:**
- Especifica el nº de puerto para el socket
- Especifica una dir. IP local para el socket
- La dirección se define como una estructura que incluye la familia de protocolos (PF_INET), la dirección IP y el número de puerto.
- Los clientes suelen dejar la elección de puerto al S.O.

```
bind (num_sock , dirlocal, longdir);
```


```
Int sd, err;
sockaddr_in  myaddr;  /* variable de tipo dirección de socket INET*/

sd = socket (PF_INET, SOCK_DGRAM, 0);
myaddr.sin_family = AF_INET;
myaddr.sin_addr.s_addr = INADDR_ANY;
myaddr.sin_port = 0;  /* BIND = 0 */
err = bind (sd, (struct sockaddr *) &myaddr, sizeof(myaddr));
```

Modelo cliente-servidor: Listen y Accept

```
listen(num_sock ,longcola);
```

- **Listen**
  - Lo utiliza el **servidor**
  - Prepara el socket para aceptar conexiones entrantes
  - Se suele usar después de SOCKET y BIND, y justo antes de ACCEPT (servidores).
  - Sólo para sockets de tipo SOCK_STREAM **(TCP)**.
- **Accept**
  - Utilizado por el servidor TCP
  - Espera la siguiente conexión y devuelve un nuevo socket

```
nuevosock = accept (num_sock ,dir,longdir);
```

Socket maestro: { TCP, IP fuente, Puerto fuente, *, * }
Nuevo socket: { TCP, IP fuente, Puerto fuente, IP dest., Puerto dest. }

Modelo cliente-servidor: Comunicación

- Conexión
  - Utilizado por el **cliente**
  - Uso:
    - Establece una conexión TCP
    - (Opcionalmente) Especifica direcciones de socket destino UDP.

```
connect (num_sock , dirdest, longdir);
```

- Transmisión

```
send (num_sock , buffer, longitud, flags);
```

```
sendto(num_sock , buffer, longitud, flags, dirdest,longdir);
```

- Recepción

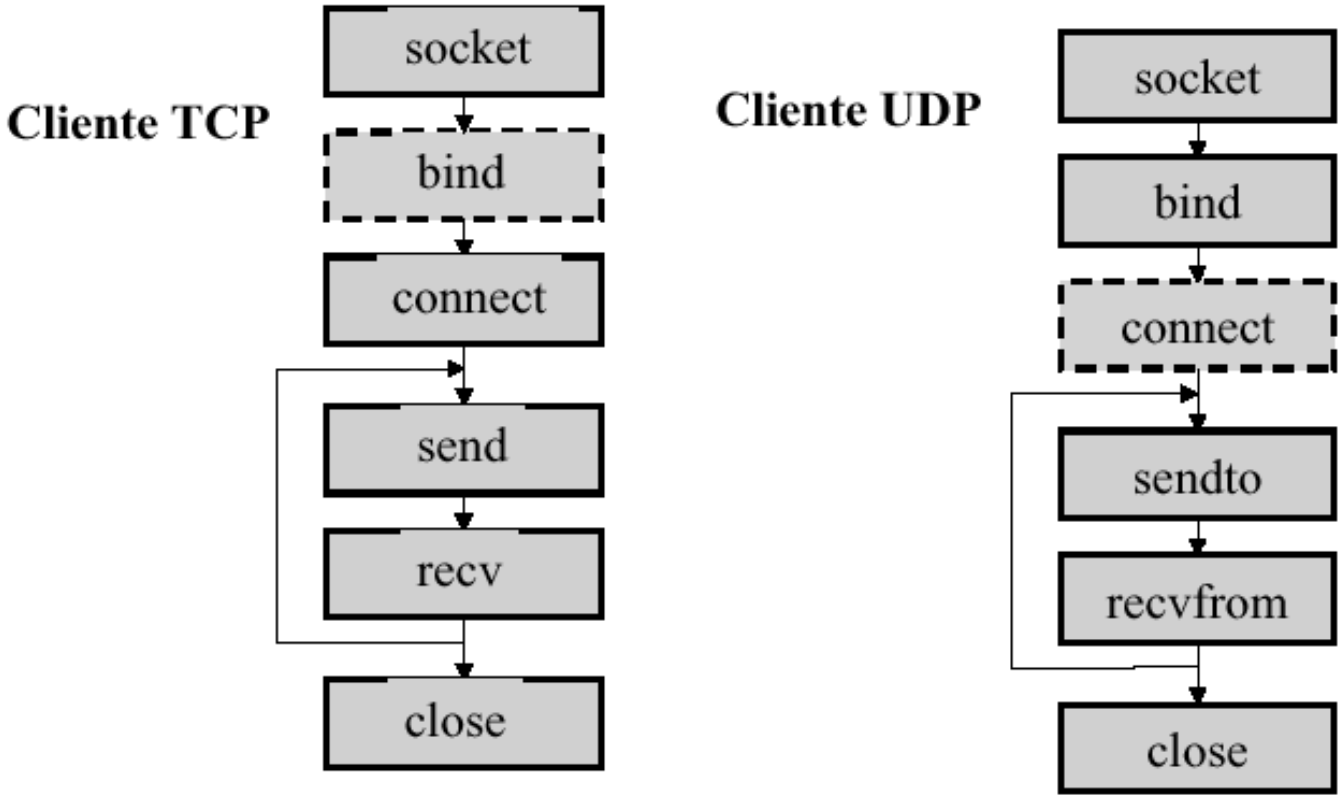
```
recv(num_sock , buffer, longitud, flags);
```

```
recvfrom(num_sock , buffer, longitud, flags, dirorig,longdir);
```

- Desconexión

```
close(num_sock);
```

Modelo cliente-servidor: Clientes TCP / UDP



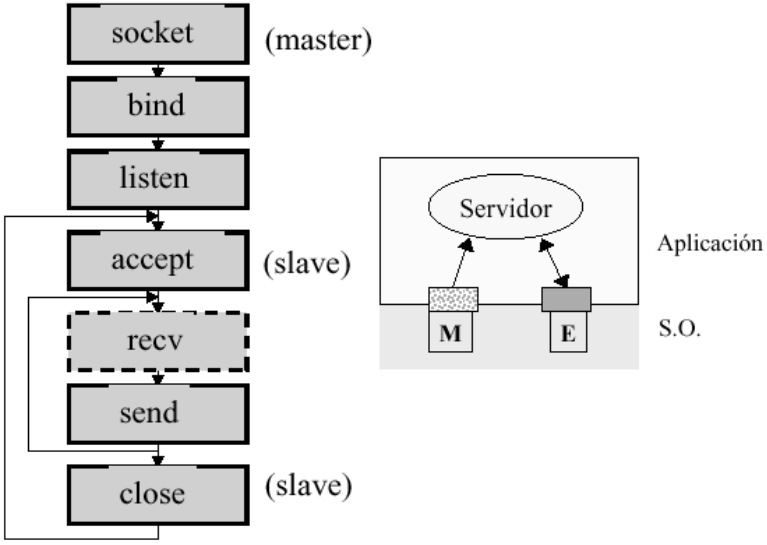
Modelo cliente-servidor: Clases de Servidores

- Servidores secuenciales.
  - Procesan una solicitud cada vez.
  - Las peticiones que lleguen mientras se está atendiendo a una se encolan.
- Servidores Concurrentes.
  - Pueden atender a varios clientes al mismo tiempo.
  - Son más complejos y utilizan más recursos del sistema.

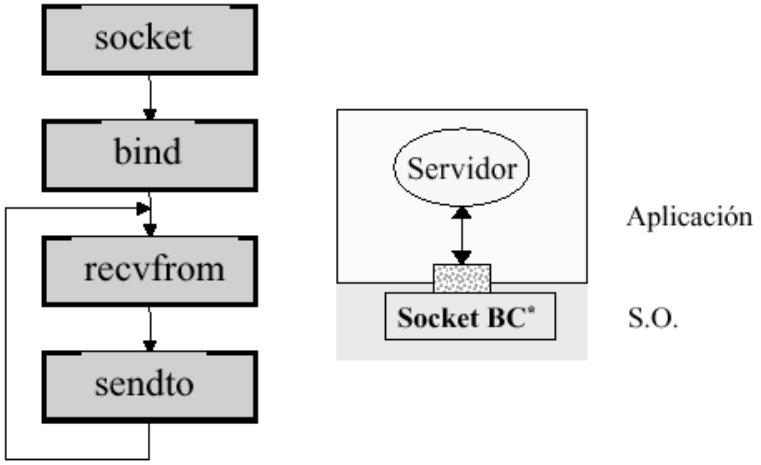
- Secuencial con conexión (TCP): DAYTIME, TIME, ECHO, FINGER, etc.
- Secuencial sin conexión (UDP): DAYTIME, TIME, ECHO, FINGER, etc.
- Concurrente con conexión (TCP): TELNET, WWW, FTP, NNTP, SMTP, etc.
- Concurrente sin conexión (UDP): TFTP.

Modelo cliente-servidor: Servidores Secuenciales

TCP



UDP

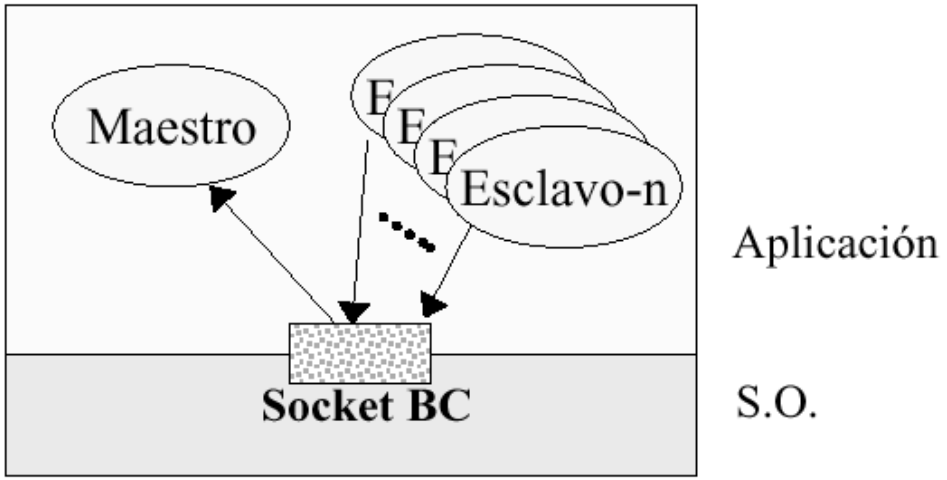


Modelo cliente-servidor: Servidores concurrentes

- **Maestro**
  - Recoge las peticiones de servicio y crea proceso esclavo para atenderlas
  - Nunca habla con el cliente
  - Se ejecuta siempre
- **Esclavo**
  - Responde a las peticiones.
    - ¿Cómo identifica el *cliente* la copia del servidor con la que establece la comunicación?
  - Desaparece al terminar el servicio

Modelo cliente-servidor: Concurrente UDP

- La mayoría de servidores concurrentes suelen ser con conexión (TCP).
- El coste que supone crear un nuevo proceso a menudo no compensa el tiempo de servicio de una petición.
- Una excepción:
  - TFTP (Trivial File Transfer Protocol).



Modelo cliente-servidor: Concurrente TCP

- Pueden atenderse varias conexiones simultáneamente a través de distintos procesos hijos
- El proceso creado (proceso hijo) es una copia del proceso creador (proceso padre)