Tema 03
curso 2018/19

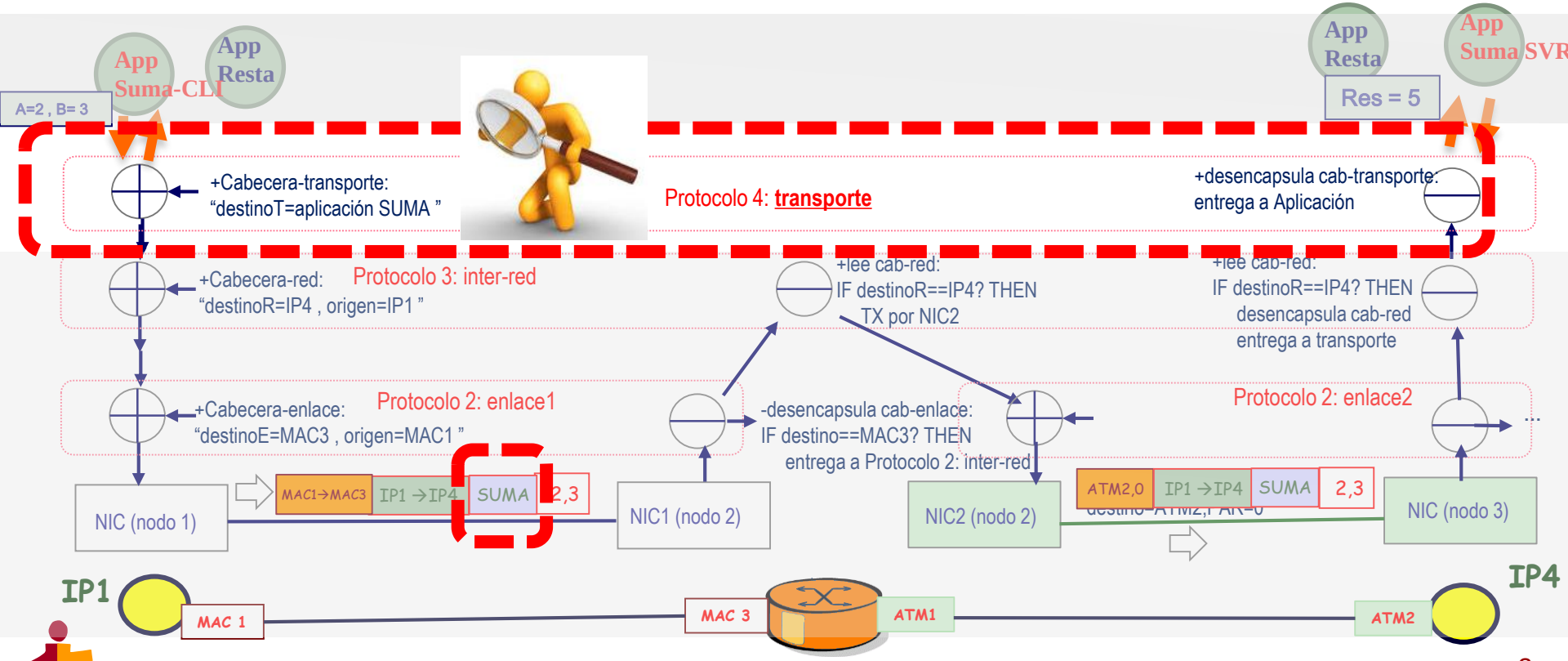
La Capa de Transporte

Antonio J. Estepa Alonso

Departamento de
Ingeniería Telemática

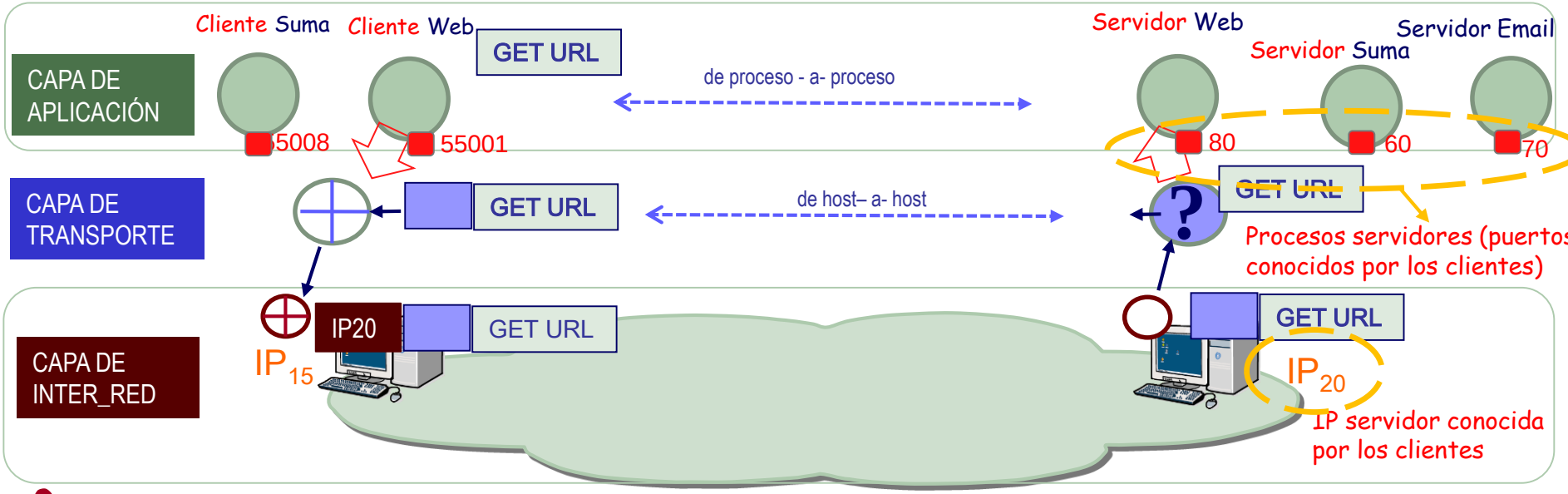
Índice del Tema 03

3.1 Introducción a la capa de transporte: servicios y protocolos
3.2 Servicio básico de multiplexión en la capa de transporte. El protocolo UDP
3.3 El protocolo TCP
3.4 Implementación del Servicio de Transferencia fiable en TCP
3.5 Control de Flujo. Implementación del servicio en TCP
3.6 Control de la Congestión. Implementación del servicio en TCP.



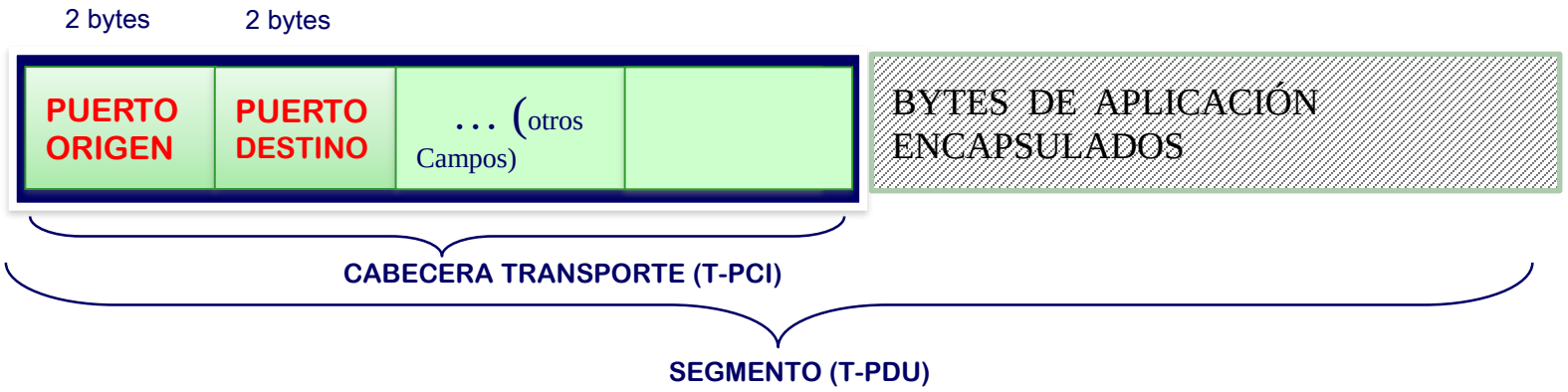
Introducción a la capa de transporte

- Ofrece comunicación lógica **entre procesos** de aplicación
  - Desde un proceso en un equipo hasta otro proceso en otro equipo
- Funcionamiento básico del protocolo de transporte
  - Extremo emisor → **recibe mensajes** de aplicación **y genera segmentos** (T-PDU)
    - En la cabecera de cada segmento se identifican los procesos de aplicación destino y origen
  - Extremo receptor → recibe segmentos, lee T-PCI y avisa al proceso de app destinatario
- La capa de transporte se apoya en el servicio de la capa de red (inter-red)



Información básica de los Protocolos de Transporte

- La **cabecera de cada segmento** (T-PCI) siempre incluirá los campos
  - Puerto origen, puerto destino
  - Tanto el protocol TCP como UDP los incluyen (servicio básico transporte)
- Puerto: entero de 16 bits (0-65.535)
  - La mayoría registrados en IANA (http://www.iana.org)
  - El rango de posibles valores
    - < 1.024 (well-known ports): reservados para aplicaciones generales registradas
      - http://www.iana.org/assignments/port-numbers
    - 1.024-49.151 (puertos registrados): aplicaciones no tan generales (registradas)
    - > 49.151 (puertos dinámicos o efímeros, normalmente usados por los clientes)

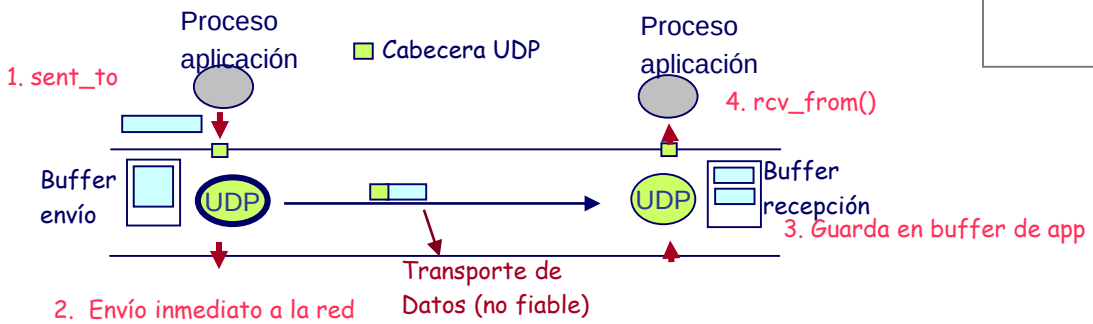


UDP (User Datagram Protocol)

- Servicio: Tranferencia de mensajes NO FIABLE entre los procesos origen y destino
  - Sin conexión establecimiento de conexión, control de flujo, control de congestión.

Ej. Primitivas de acceso al servicio UDP…

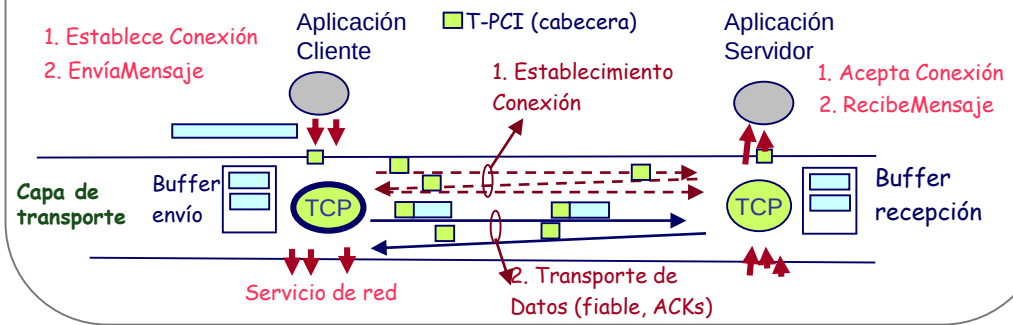
```
(cuando quiero enviar algo)
fd=socket_udp();
send_to(fd,200.1.1.3:53,"hola")
send_to(fd,193.4.2.5:53,"adios")

(cuando quiero recibir)

rcv_from(fd,&mensaje,&remitente);
```



Protocolos de transporte en Internet

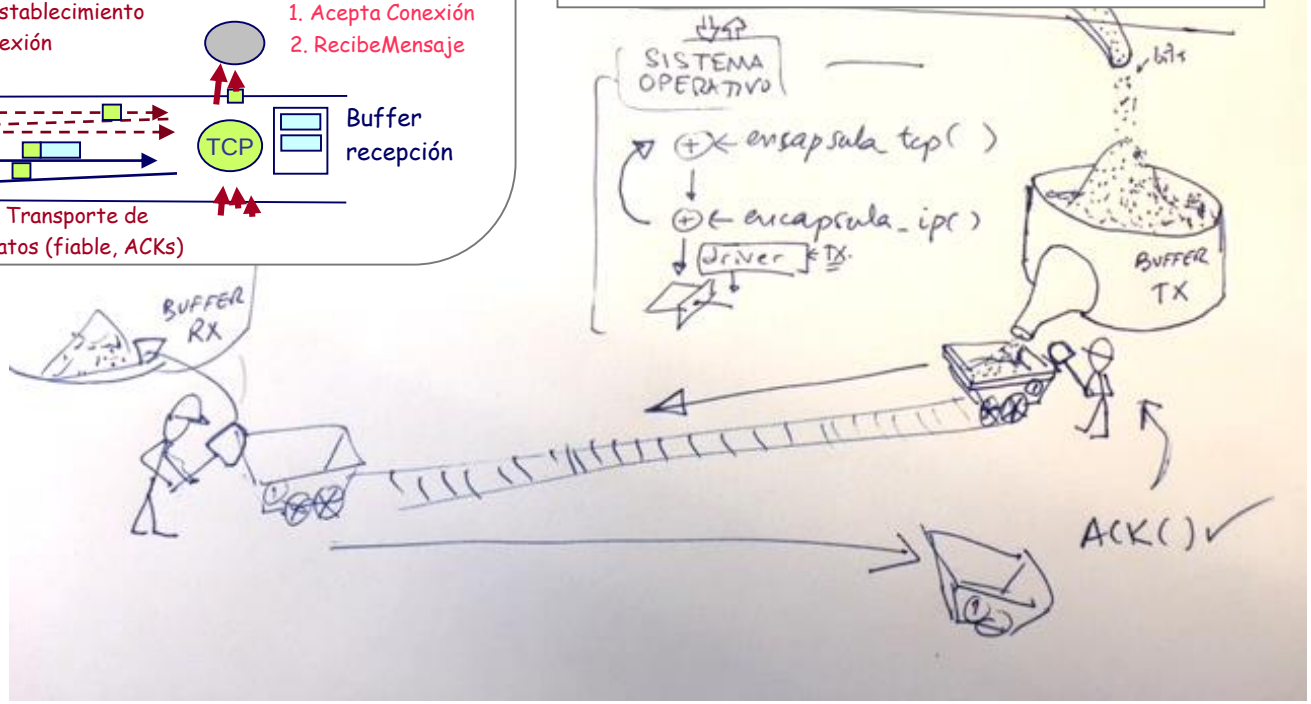
TCP (Transport Control Protocol):

- *Orientado-a-conexión*
- *Transporte Fiable* : la entidad par envía asentimientos cuando recibe datos
- *Control de Flujo* : *no desborda al receptor*
- *Control de Congestión: no desborda a la red*



Ej. Primitivas de acceso al servicio TCP…

```
(cuando quiero enviar algo)
fd=socket_tcp();
connect(fd, 200.1.1.3:53)

while (1):
        send(fd,"hola")

(cuando quiero recibir)
rcv(fd,&mensaje);
```

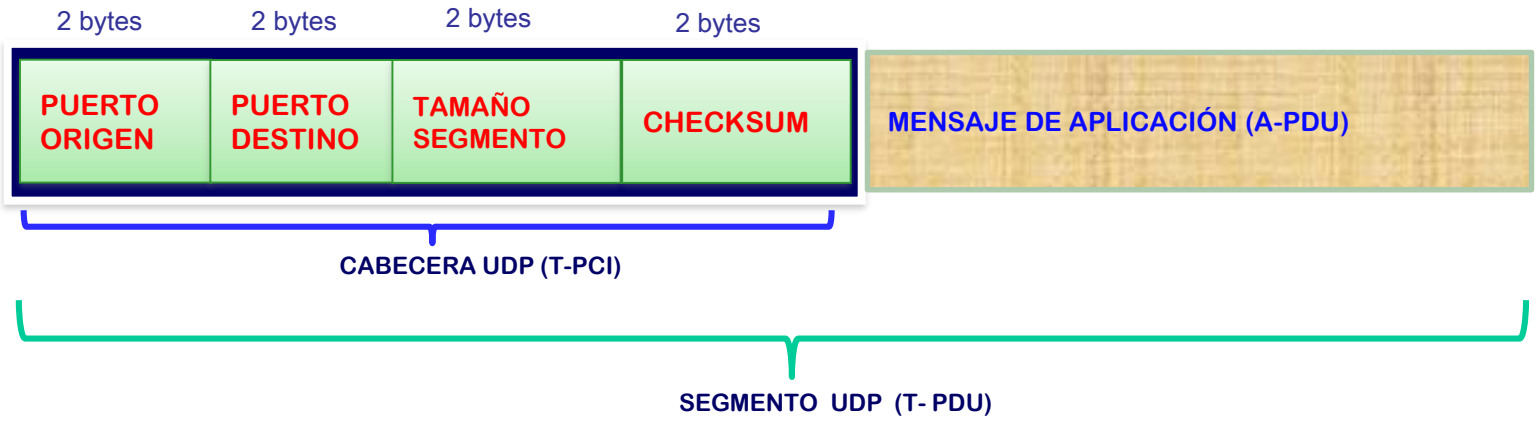


Índice del Tema 03

3.1 Introducción a la capa de transporte: servicios y protocolos
3.2 Servicio básico de multiplexión en la capa de transporte. El protocolo UDP
3.3 El protocolo TCP
3.4 Implementación del Servicio de Transferencia fiable en TCP
3.5 Control de Flujo. Implementación del servicio en TCP
3.6 Control de la Congestión. Implementación del servicio en TCP.

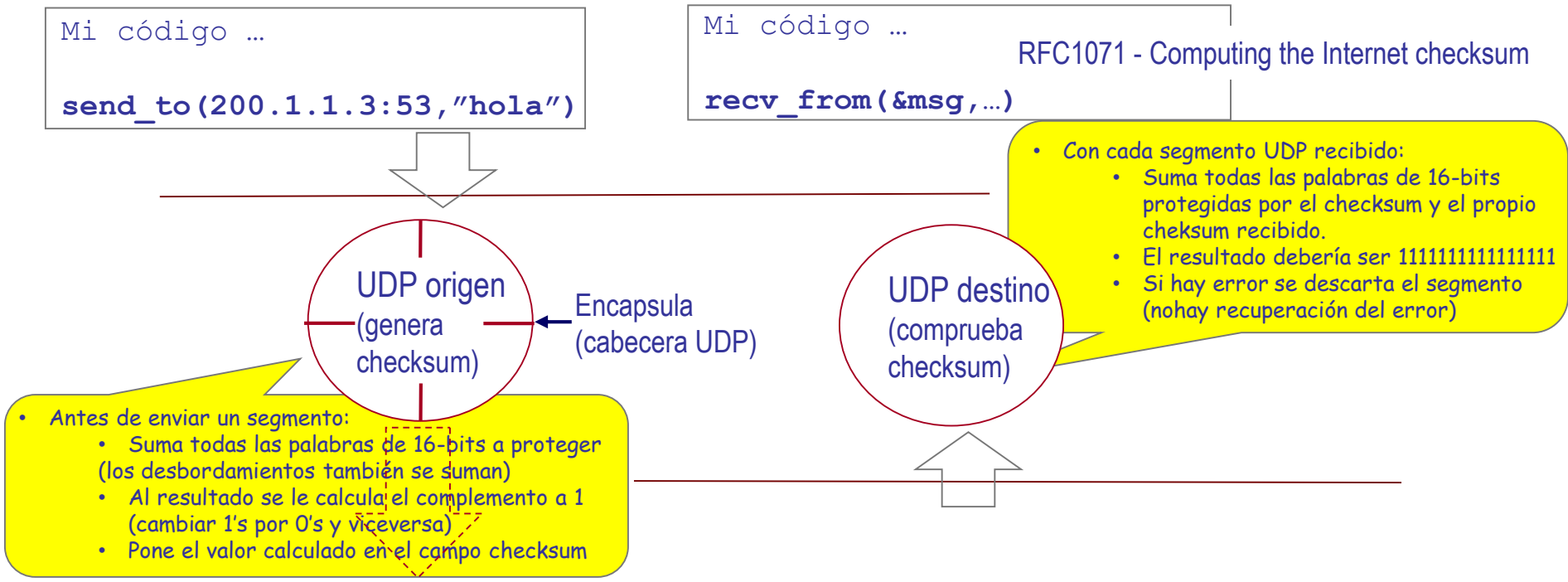
El protocolo UDP: User Datagram Protocol [RFC 768]

- Estructura del segmento UDP (cabecera)
  - 4 CAMPOS de 2 bytes cada uno (8 bytes)
    - Puerto Origen.– identifica al socket del proceso emisor
    - Puerto Destino.– identifica al socket del proceso receptor
    - Tamaño del Segmento.– número de bytes del segmento (T-PDU)
    - Suma de Comprobación (checksum). (16 bits) – comprobación de errores de la cabecera o los datos del segmento.
      - De uso obligatorio si el segmento se encapsula en IPv6 (opcional en IPv4)



El checksum de Internet (detección de errores)

- El origen añade información redundante en el campo checksum que deberá ser comprobada en el destino.



- Uso opcional decidido por el origen (en IPv6 será obligatorio)
  - Si el origen envía 0 como checksum → el destino no lo comprueba

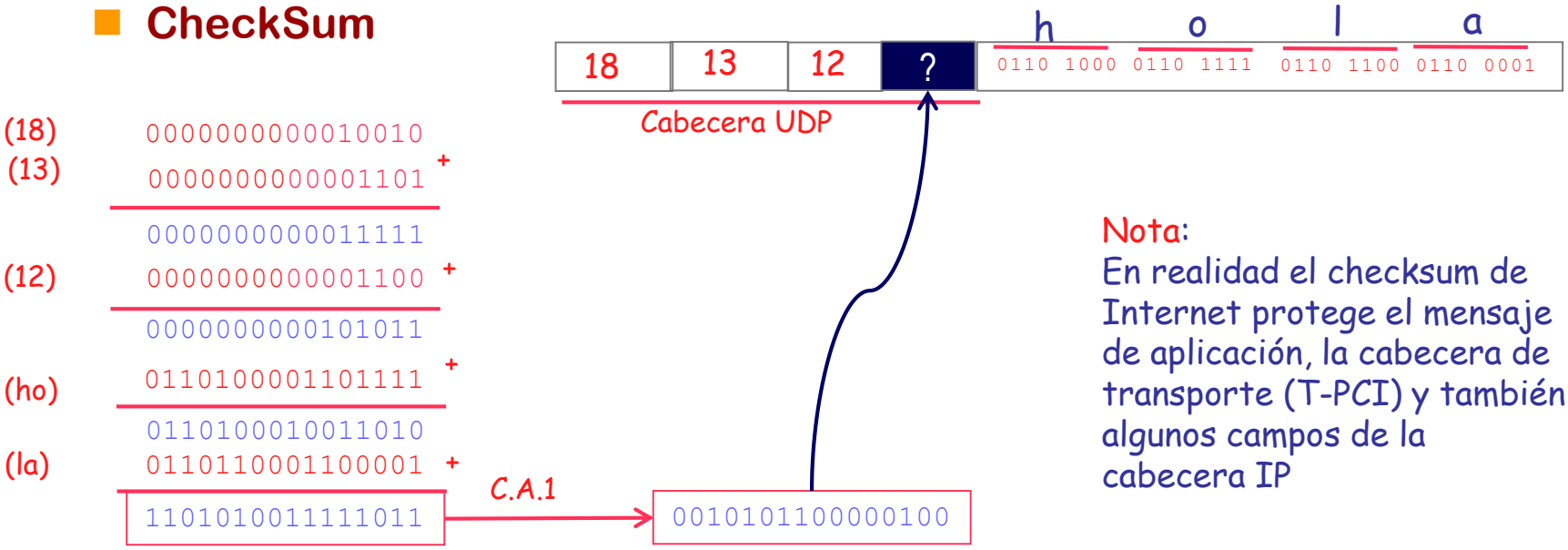
Ejercicio

- Una aplicación (puerto local 18) envía a través de udp el mensaje “hola” a otra aplicación (puerto remoto 12)
  - h (0x68) o (0x6F) l (0x6C) a (0x61)

- ¿cabecera UDP generada?

Solución

- Una aplicación (puerto local 18) envía a través de udp el mensaje "hola" a otra aplicación (puerto remoto 13)
  - h (0x68) o (0x6F) l (0x6C) a (0x61)
- ¿cabecera UDP generada?
  - Puerto Origen → 18 → 00000000.00010010
  - Puerto Destino → 13 → 00000000.00001101
  - Tamaño Segmento → 12B → 00000000.00001100
  - CheckSum



Nota:
En realidad el checksum de Internet protege el mensaje de aplicación, la cabecera de transporte (T-PCI) y también algunos campos de la cabecera IP

Ventajas y Desventajas de UDP

Ventajas ☺

- Control a nivel de aplicación sobre los datos enviados
  - El usuario decide cuándo se envia el segmento a la red
  - Importante para aplicaciones que requireren una tasa mínima
    - Aplicaciones de tiempo real
- Sin establecimiento de la conexión
  - Sin espera a la transferencia
- Sin estado de la conexión
  - Consume pocos recursos
- Menor tamaño de cabecera que TCP
  - 8 bytes frente a 20 bytes.

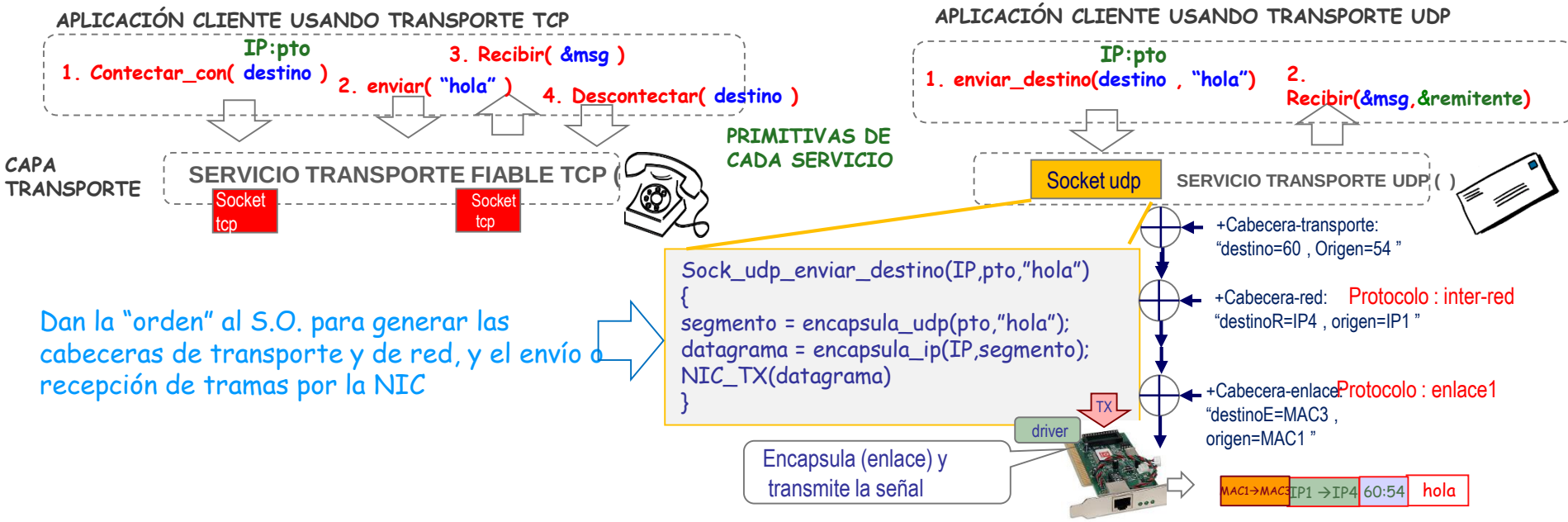
Desventajas ☹

- No fiable … depende del servicio del protocolo de red
  - Posibilidad de …
    - Entrega de paquetes duplicados, entrega de paquetes desordenados respecto al origen… No entrega de paquete …
- No tiene control de flujo entre aplicaciones …
  - Posible desbordamiento del buffer si el emisor envía datos a la red a mayor velocidad que son leídos por el proceso de aplicación receptor
- No tiene control de congestión
  - …es malo en general para la red

EJEMPLO TÍPICO DE USO: MULTIMEDIA

Recordatorio: sockets

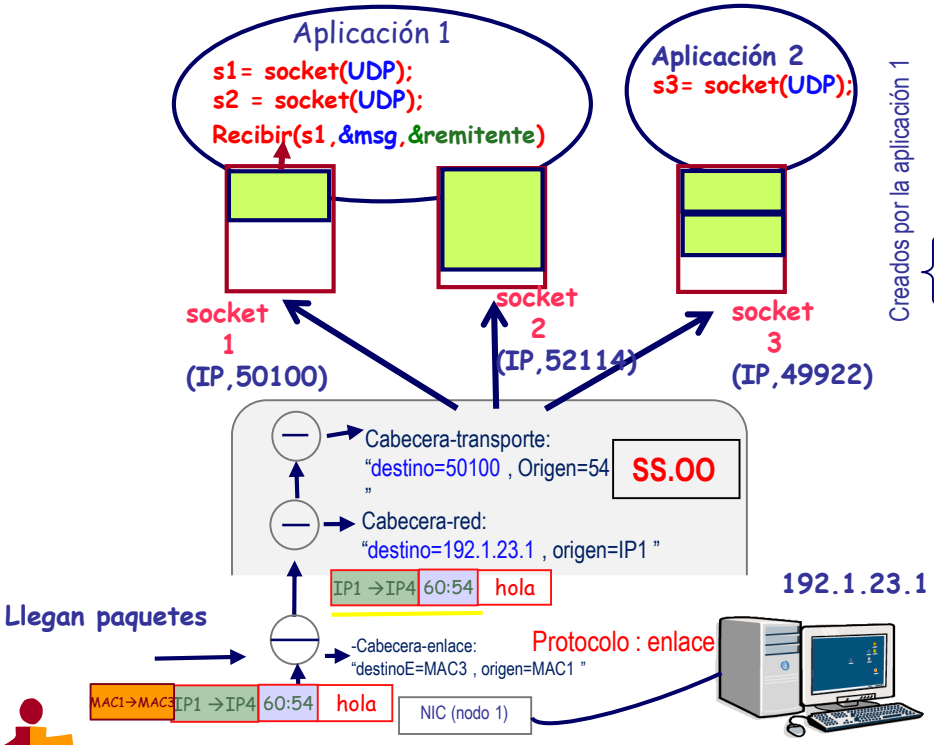
- Punto de acceso a los servicios de transporte (… y a los de red)
  - Son creados por las aplicaciones



- Un socket de tipo UDP vale para enviar a cualquier destino (hay que indicar el destino en cada envío) y para recibir de cualquier origen
- Un socket de tipo TCP, una vez conectado, sólo puede enviar y recibir del proceso al otro lado de la conexión.

Desmultiplexión con sockets UDP

- El S.O. de cada equipo mantiene una lista con los sockets que las aplicaciones han creado durante su ejecución
  - El S.O. identifica a cada socket **udp** creado a través del par de valores: **(dirección IP local, Puerto local)**
  - Al llegar un paquete el S.O. examina las cabeceras de red y transporte (IP y puerto destino) y selecciona el socket adecuado



Creados por la aplicación 1

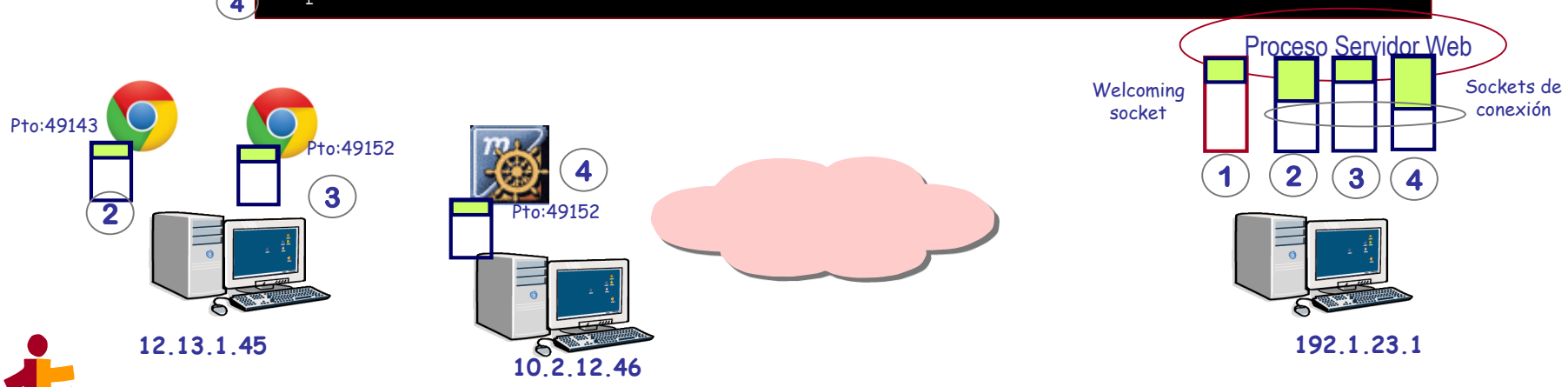
```
%>netstat -p udp
Protocol   Local addr   Local port
1 udp      192.1.23.1     50100
2 udp      192.1.23.1     52114
3 udp      192.1.23.1     49922
```

(en Windows, con permiso de administrador, netstat –ano), después mirar con tasklist

Desmultiplexión en TCP

- Servicio orientado a conexión: **se identifica a la conexión**
  - El servidor escucha peticiones de conexión y las acepta, **creando un nuevo socket tcp con cada conexión aceptada** simultáneamente
- El S.O. identifica al socket tcp de conexión por 4 valores:
  - **(IP local, IP remota, puerto local, puerto remoto)**
  - En los sockets de tipo escucha (welcoming) la dirección del extremo remoto es desconocida (*) – existe 1 en cada servidor

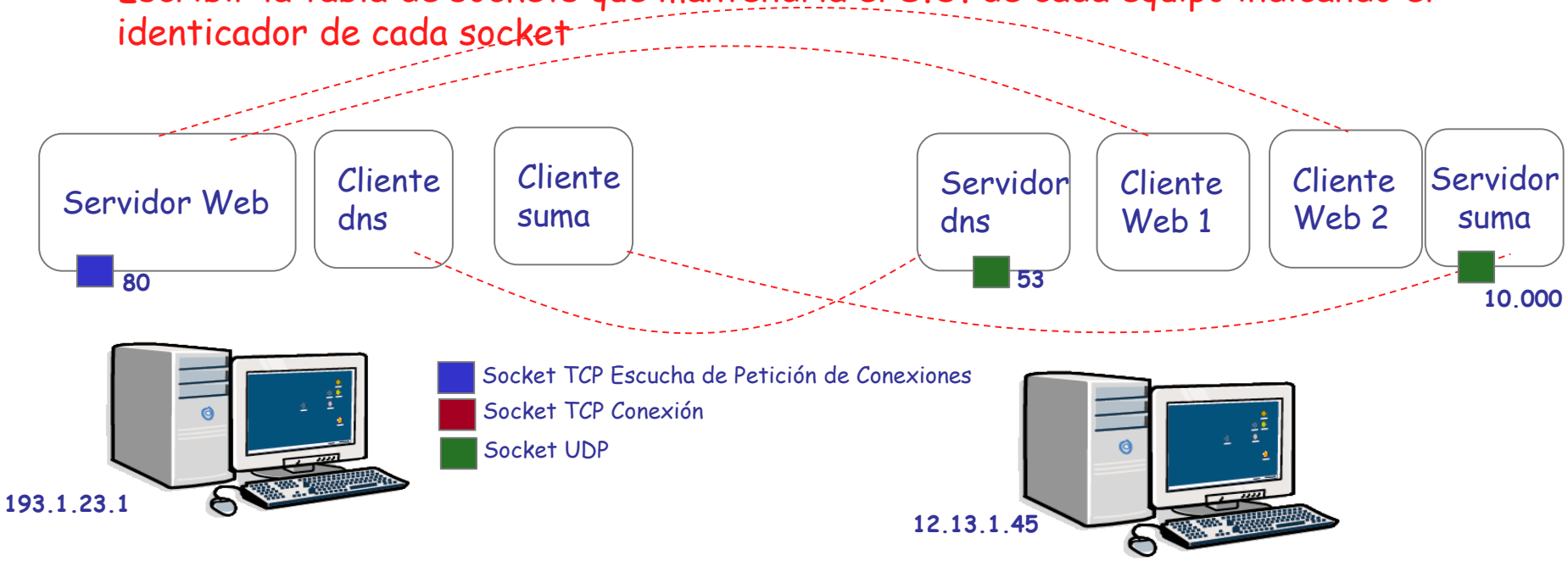
```
%>netstat -p tcp
Protocol   Local addr   Local port   Remote addr   Remote port
tcp        192.1.23.1     80           *              *
tcp        192.1.23.1     80           12.13.1.45     49143
tcp        192.1.23.1     80           12.13.1.45     49152
tcp        192.1.23.1     80           10.2.12.46     49152
```



Ejercicio: sockets utilizados en las comunicaciones

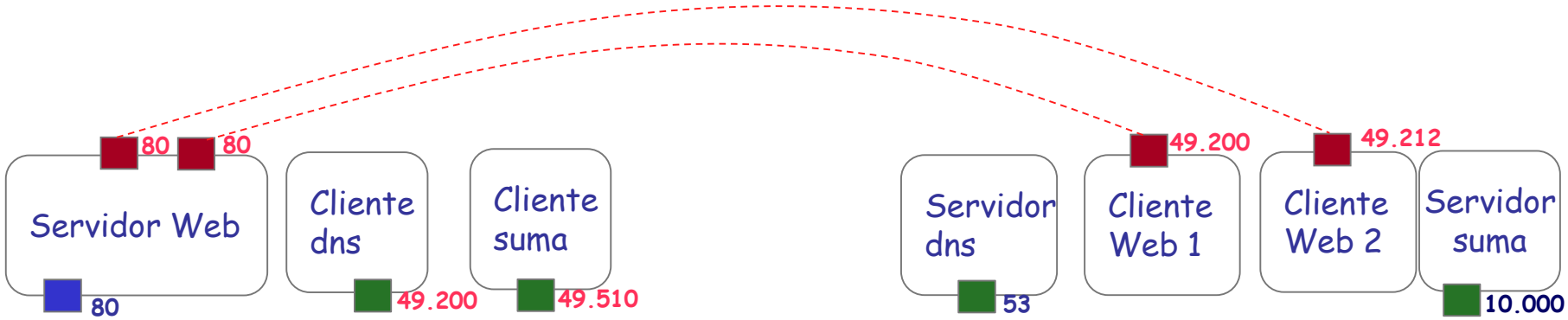
- Cada proceso de tipo cliente se comunica con el proceso de tipo servidor correspondiente
- Las aplicaciones dns y suma usan udp, la web tcp
  - Los puertos usados por los servidores están indicados.

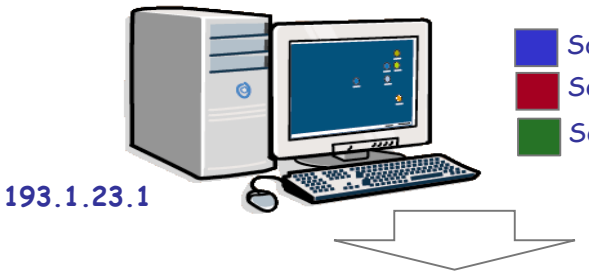
- Dibujar los sockets utilizados por cada proceso y su identificador de puerto local
- Escribir la tabla de sockets que mantendría el S.O. de cada equipo indicando el identicador de cada socket



Solución

- Cada proceso de tipo cliente se comunica con el proceso de tipo servidor correspondiente





Socket TCP Escucha de Petición de Conexiones
Socket TCP Conexión
Socket UDP


```
%>netstat -p tcp
Protocol  Local addr   Local port   Remote addr   Remote port
tcp       193.1.23.1     80           *              *
tcp       193.1.23.1     80           12.13.1.45     49200
tcp       193.1.23.1     80           12.13.1.45     49212
udp       193.1.23.1    49200
udp       193.1.23.1    49510
```

```
%>netstat -p tcp
Protocol  Local addr   Local port   Remote addr   Remote port
udp       12.13.1.45     53
tcp       12.13.1.45     49200       193.1.23.1     80
tcp       12.13.1.45     49212       193.1.23.1     80
udp       12.13.1.45     10000
```

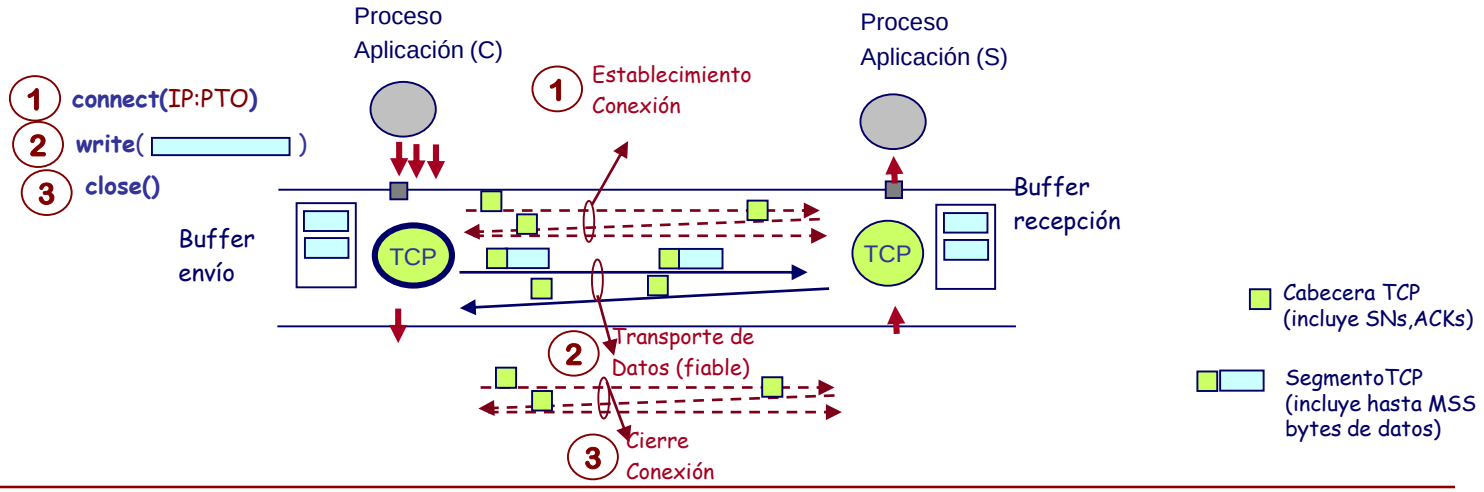
Índice del Tema 03

3.1 Introducción a la capa de transporte: servicios y protocolos
3.2 Servicio básico de multiplexión en la capa de transporte. El protocolo UDP
3.3 El protocolo TCP
3.4 Implementación del Servicio de Transferencia fiable en TCP
3.5 Control de Flujo. Implementación del servicio en TCP
3.6 Control de la Congestión. Implementación del servicio en TCP.

Transport Control Protocol (TCP) RFCs: 793, 1122, 1323, 2018, 2581

Características Principales

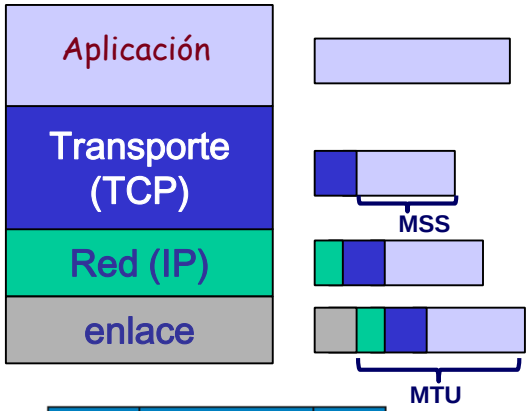
- *Orientado-a-conexión:* se requiere el establecimiento de una "conexión" entre los procesos cliente y servidor antes del intercambio de mensajes
- *Punto-a-punto:* un emisor, un receptor
- *Flujo de bytes, fiable y ordenado:* no hay "límites del mensaje"
  - *MSS (maximum segment size)*
- *Full-duplex:* flujo de datos bi-direccional en la misma conexión.
- *Control de Flujo :* conoce el estado del buffer de recepción
- *Control de Congestión:* TCP decide cuándo y cómo usar el servicio de red
- *(pipelined) Envío Contínuo:* los algoritmos de control de flujo y congestión determinarán dinámicamente el número máximo de bytes que el emisor puede tener pendientes de asentimiento.



Maximum Segment Size (MSS)

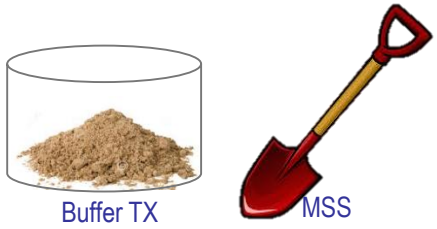
- La máxima cantidad de datos encapsulados en un segmento TCP:
  - Calculado para generar paquetes que no excedan la MTU del nivel de enlace
    - Ejm: MTU 1.500 octetos (ethernet) →
      - MSS = MTU – 20(TCP) – 20(IP) = 1.460

¿cuántos segmentos TCP son necesarios para Transmitir un mensaje de aplicación de 2000Bytes a través de una NIC de tipo WiFi (MTU=1.500B)?



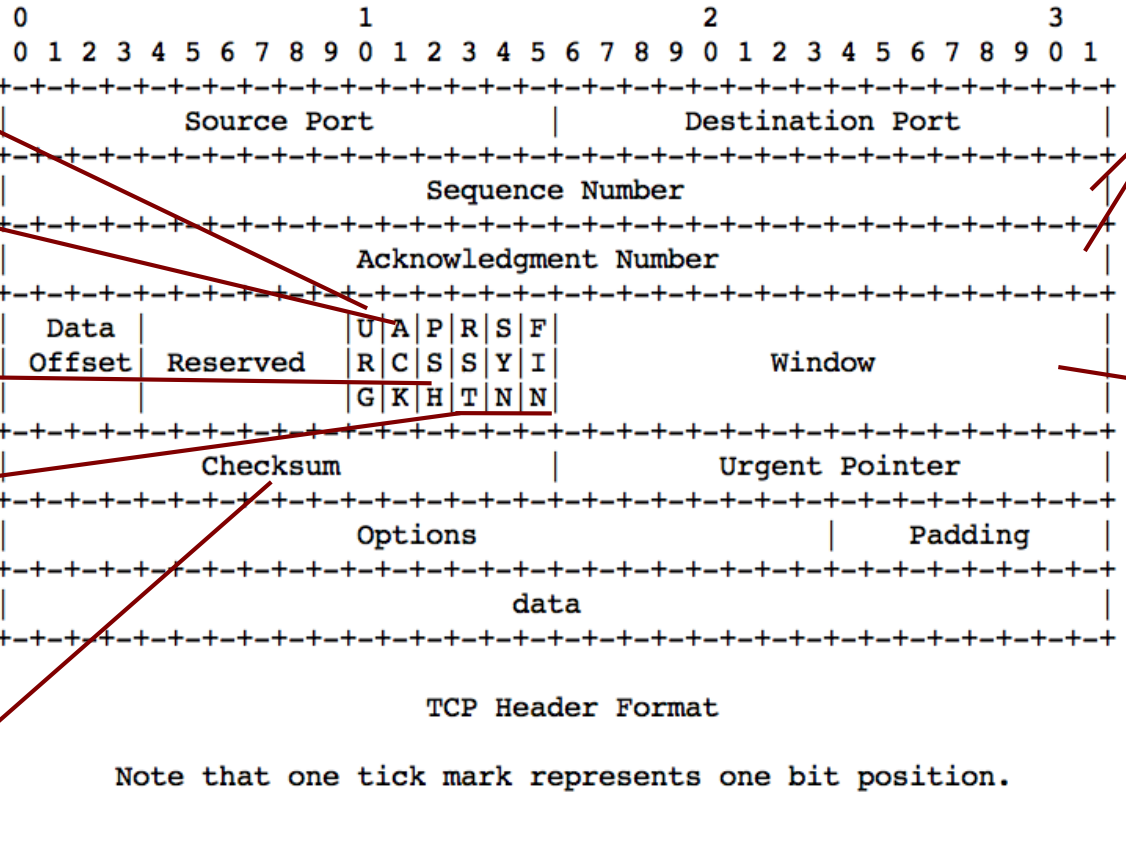
MTU	Protocol	RFC
576	Default	879
1500	PPP default	1134
296	PPP (low relay)	1144
1500	Ethernet	895
1006	SLIP	1055
1492	PPPOE	2516

  - Procedimiento path discovery [RFC 1191] → averigua la menor MTU de un camino



Cabecera TCP

- Descrita en RFC 793
  - Tamaño: 20 octetos (sin opciones)



URG: urgent data (generalm. no usado)

ACK: # ACK valido

PSH: push data now (generalm. no usado)

RST,SYN,FIN: Conexión (reset, establecimiento, cierre)

Checksum Internet (como en UDP)

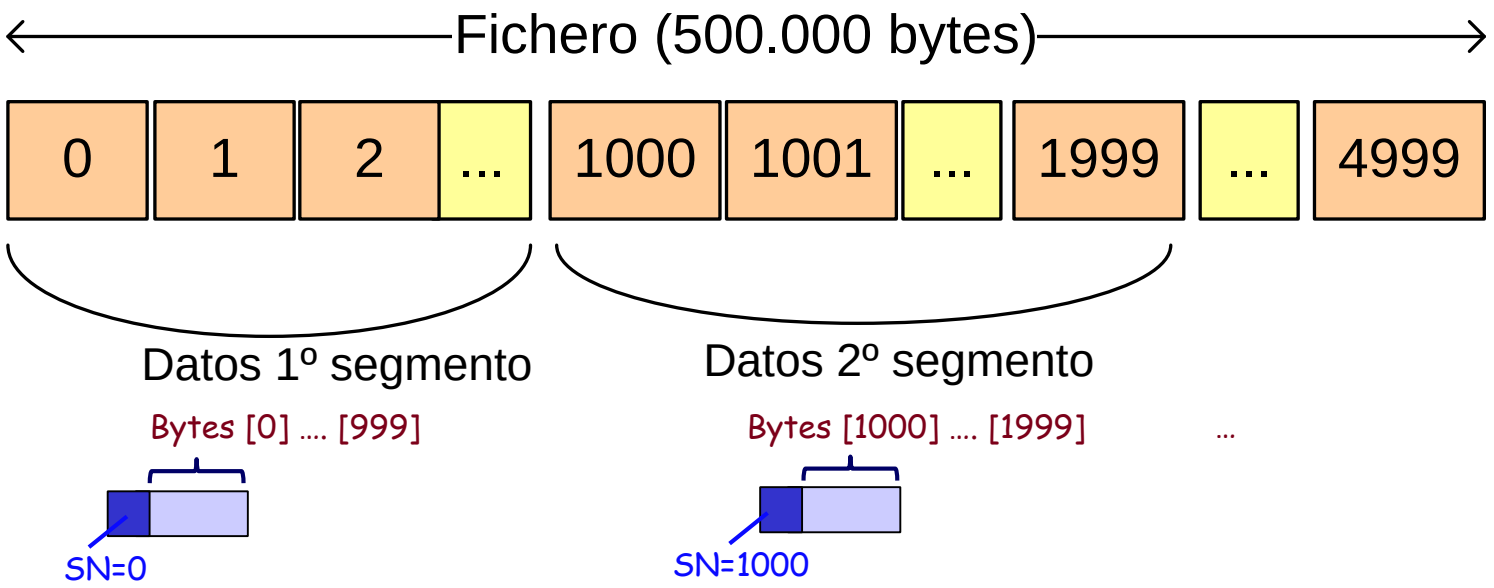
Cuentan bytes de datos (no segmentos)

bytes libres En buffer de RX (#bytes que el emisor del segmento aceptaría

Números de secuencia y asentimiento

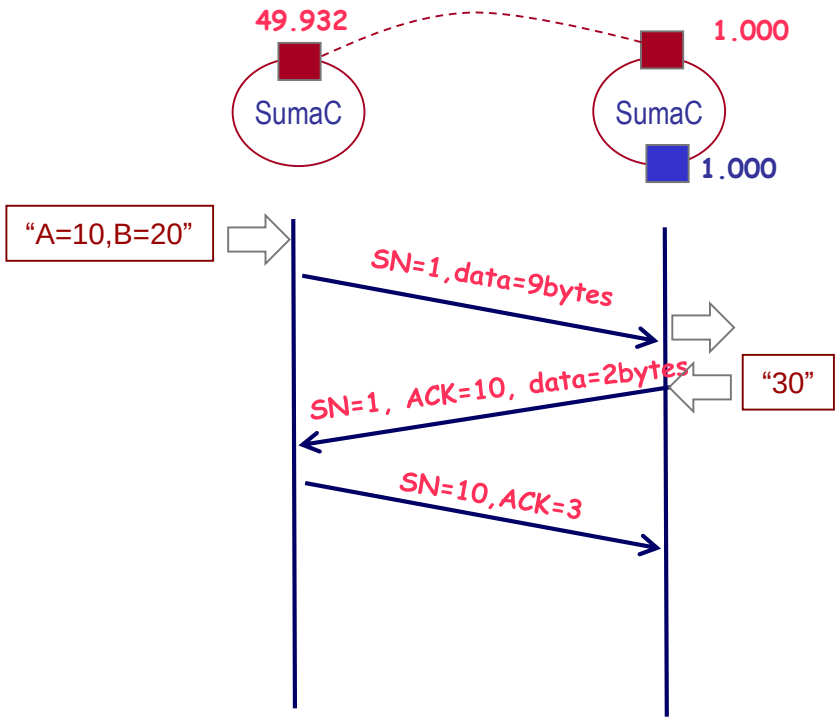
- TCP ve los datos de aplicación como un flujo ordenado de bytes
- El número de secuencia (SN) de un segmento es el número del primer byte dentro el flujo que viaja en el segmento
- El número de asentimiento (ACK) de un segmento indica el siguiente byte que se espera recibir del otro extremo (ACKs acumulativos)

Ejemplo: supongamos que el proceso A quiere enviar al proceso B un bloque de datos de 500.000 bytes y que MSS es 1000 bytes.
Si suponemos que el primer byte del flujo comienza por cero, entonces el flujo ordenado será algo como:



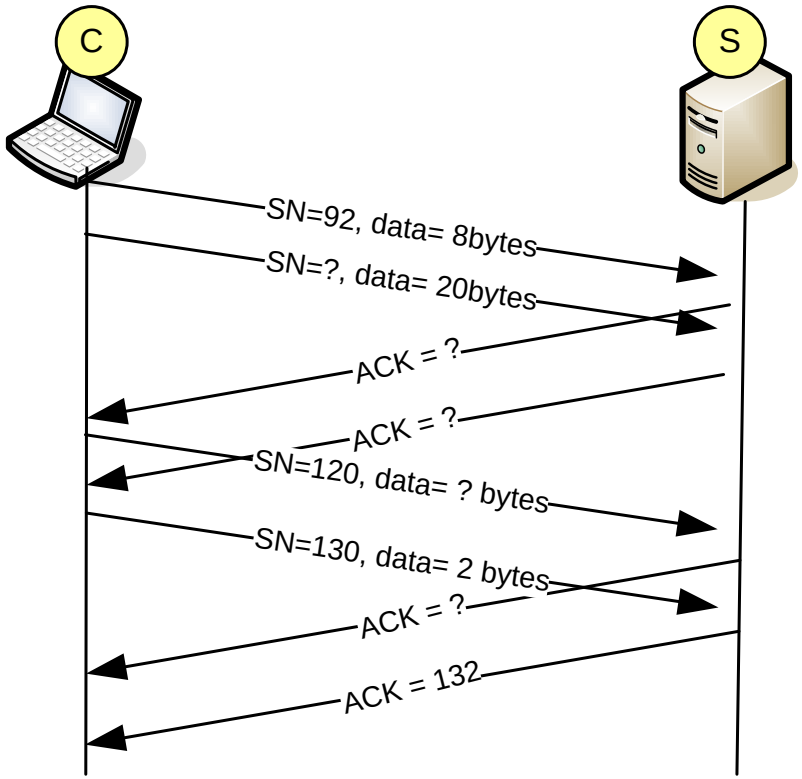
Números de secuencia y asentimiento

- Ejemplo Aplicación Suma
  - El cliente envía al servidor el mensaje “A=10,B=20” (número de secuencia inicial 1)
  - El servidor envía de vuelta al cliente “30” (número de secuencia inicial 1), y a la vez tcp asiente la recepción del mensaje del cliente
  - El protcolo tcp del cliente asiente la llegada de la respuesta del servidor.



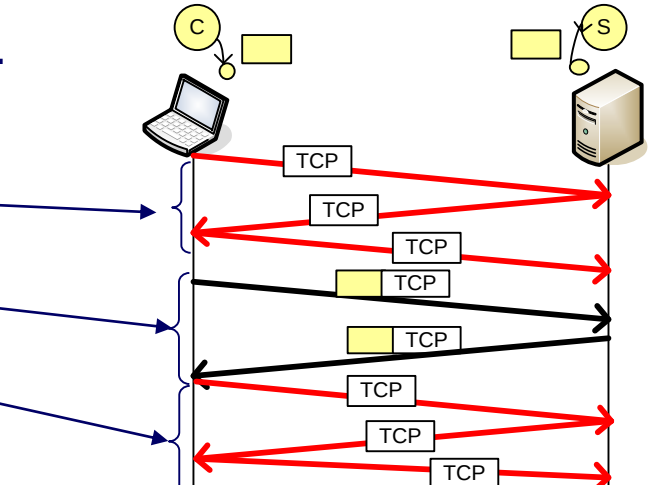
Ejercicio

- Complete los números de secuencia o asentimiento de la siguiente figura



Control de la conexión en TCP

- TCP: protocolo fiable **orientado a conexión**.
- Fases:
  - Apertura de la conexión
  - Transferencia
  - Cierre de la conexión



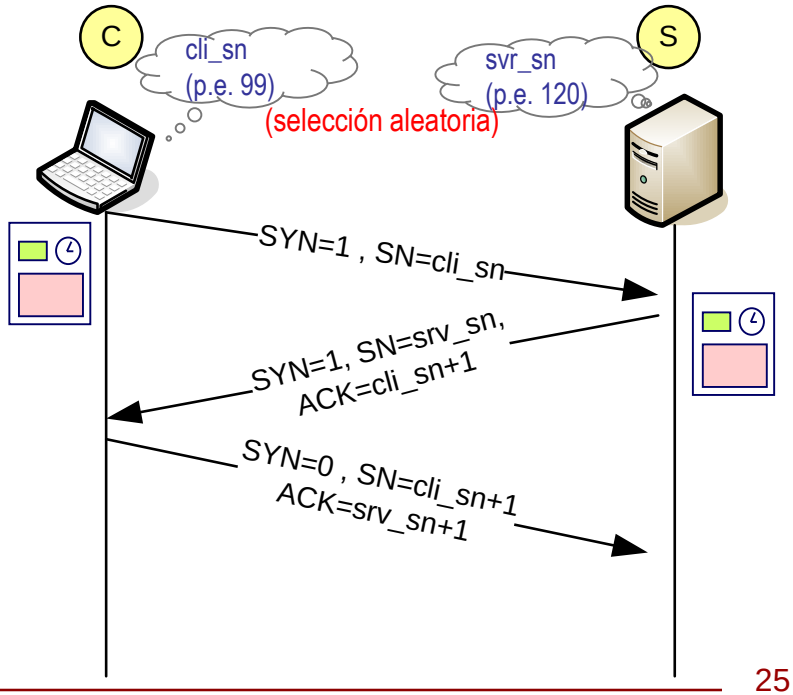
- Establecimiento de la conexión

Three way handshake:

Paso 1: host cliente envía segmento TCP SYN al servidor
- **Especifica el # seq inicial del cliente**
- **Sin datos**

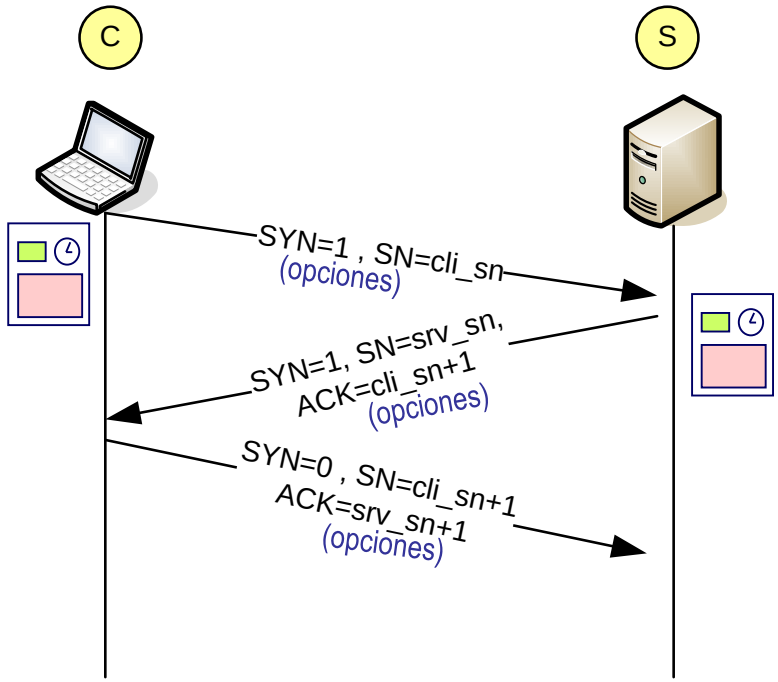
Paso 2: host servidor recibe SYN, responde con segmento SYNACK
- **Servidor reserva buffers**
- **Especifica el #seq inicial del servidor**

Paso 3: cliente recibe SYNACK, responde con segmento ACK, podría contener datos



Establecimiento de la conexión (cont.)

- También se pueden negociar parámetros opcionales como
  - **Maximum-receive-segment-size**
    - Valor MSS máximo admitido
  - **Window-scale option** (RFC 1323)
    - unidades usadas en el campo window
  - **SACK-permitted option and SACK option** (RFC 2018)
    - Variaciones en el prot. transf. fiable
  - Parámetros que se ajustan para intentar las mejores prestaciones posibles (thoughput) en la futura conexión.
    - El mayor tamaño de paquete IP posible
    - Una ventana de tamaño adecuado al producto retardo-ancho de banda
    - Una recuperación rápida ante errores



Cierre de la conexión

- Envío de bit FIN cuando un extremo ya no tiene nada mas que transmitir.(p.e. el cliente cierra el socket con `close()`)
  - Se supone que tampoco espera nada mas del otro extremo (shutdown)
- Cada extremo de la conexión termina de manera independiente
- Ejemplo de cierre

Paso 1: host cliente envía segmento TCP FIN al servidor
- El cliente no tiene nada más que enviar al servidor

Paso 2: host servidor recibe FIN, responde con segmento ACK. Cierra conexión y envía segmento FIN al cliente.
- Suponemos que el servidor también quería cerrar la conexión

Paso 3: cliente recibe FIN, responde con segmento ACK
- Entra en estado "time&wait", responderá con ACK a los FINs recibidos

Paso 4: servidor recibe ACK, conexión cerrada.

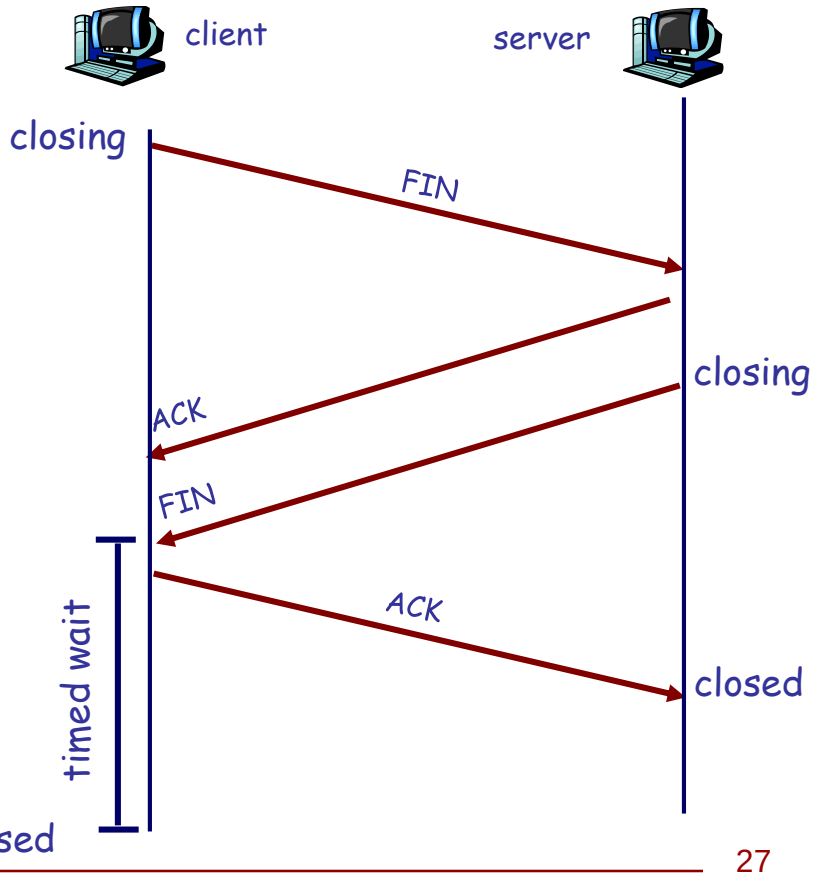
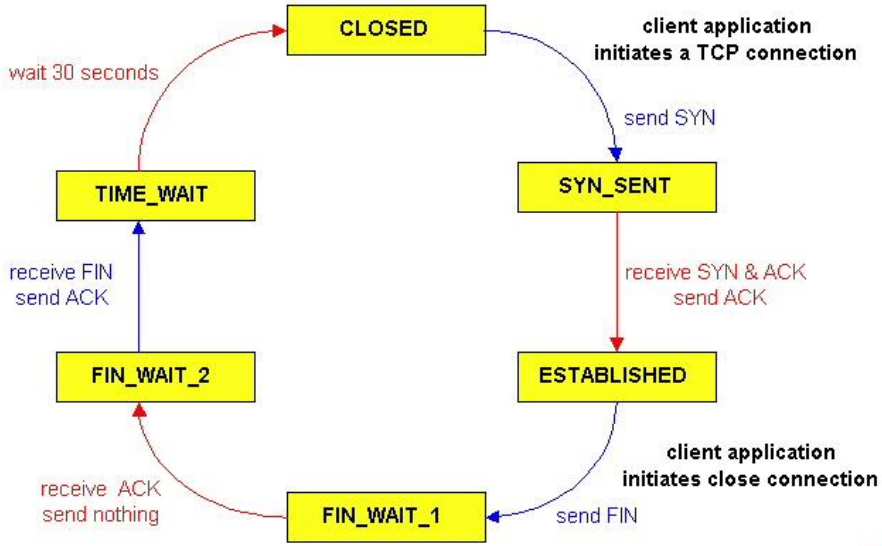


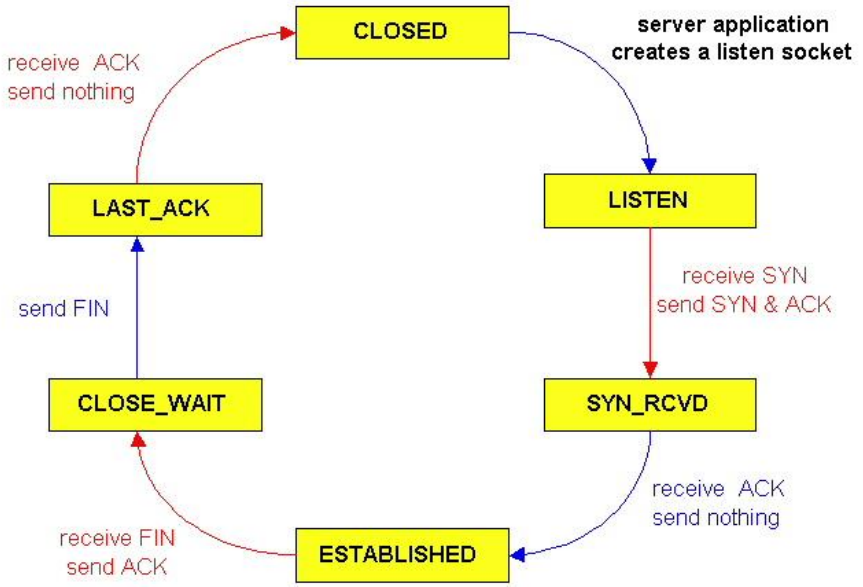
Diagrama de estados de TCP

- TCP utiliza un estado sincronizado entre los dos extremos.



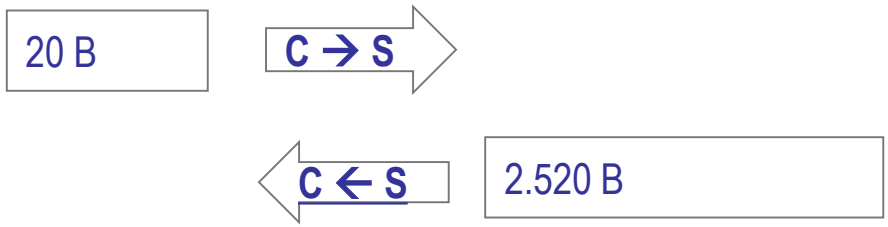
Ciclo de vida
Cliente TCP

Ciclo de vida
Servidor TCP

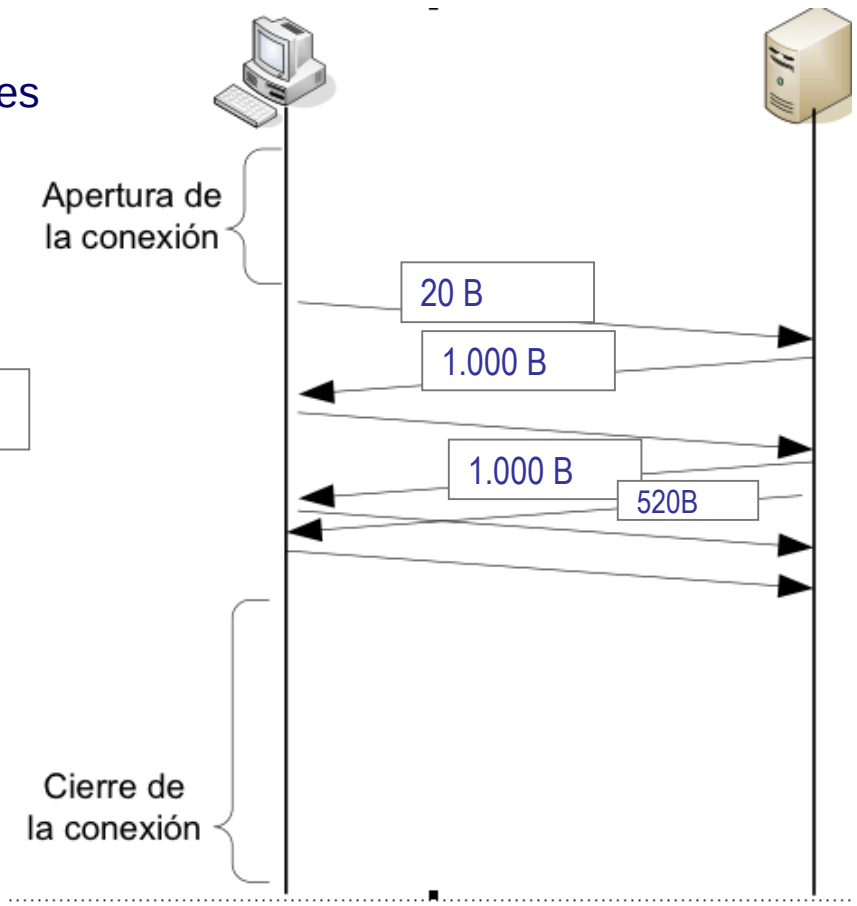


EJERCICIO

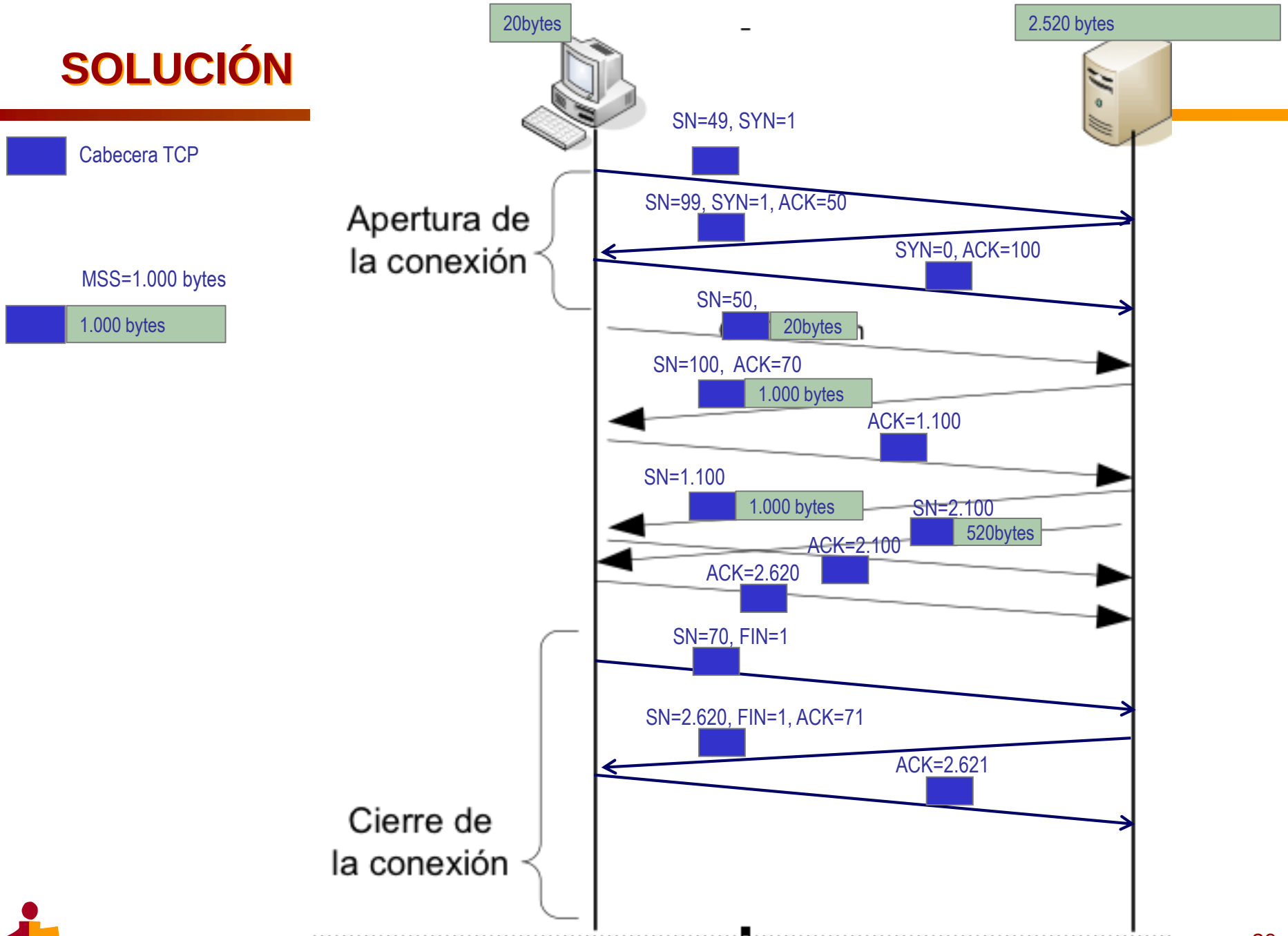
- Dibuje los segmentos intercambiados en los procesos de apertura y cierre de la conexión (ambos iniciados por el cliente) junto con los valores de los campos de la cabecera relevantes para estos segmentos.



- Escriba los números de secuencia y asentimiento para cada uno de los segmentos intercambiados durante la conexión. Puede escribir estos campos en la propia figura. El servidor genera los segmentos indicados en la figura, y el el tamaño de MSS es de 1KB.



SOLUCIÓN


Cabecera TCP
MSS=1.000 bytes
1.000 bytes
20bytes
2.520 bytes
SN=49, SYN=1
SN=99, SYN=1, ACK=50
SYN=0, ACK=100
Apertura de la conexión
SN=50,
20bytes
SN=100, ACK=70
1.000 bytes
ACK=1.100
SN=1.100
1.000 bytes
SN=2.100
520bytes
ACK=2.100
ACK=2.620
SN=70, FIN=1
SN=2.620, FIN=1, ACK=71
ACK=2.621
Cierre de la conexión

Índice del Tema 03

3.1 Introducción a la capa de transporte: servicios y protocolos
3.2 Servicio básico de multiplexión en la capa de transporte. El protocolo UDP
3.3 El protocolo TCP
3.4 Implementación del Servicio de Transferencia fiable en TCP
3.5 Control de Flujo. Implementación del servicio en TCP
3.6 Control de la Congestión. Implementación del servicio en TCP.

Servicio de Transferencia Fiable (Reliable Data Transfer)

- Errores implícitos en los sistemas de transmisión
  - Durante la transmisión, propagación o mientras esta almacenado en los buffers de transmisión de los nodos de interconexión
- Transferencia fiable:
  - Asegura que los datos son repartidos desde el proceso emisor al proceso receptor de manera correcta y secuencial.
    - Elimina la posibilidad de …
      - Entrega de paquetes duplicados
      - Entrega de paquetes desordenados respecto al origen
      - No entrega de paquete …
- El problema de la falta de fiabilidad puede resolverse a diferentes niveles (top-10 de la lista de tópicos de redes)
  - Nivel de Aplicación y/o Transporte y/o Enlace

Tareas básicas en la transferencia fiable

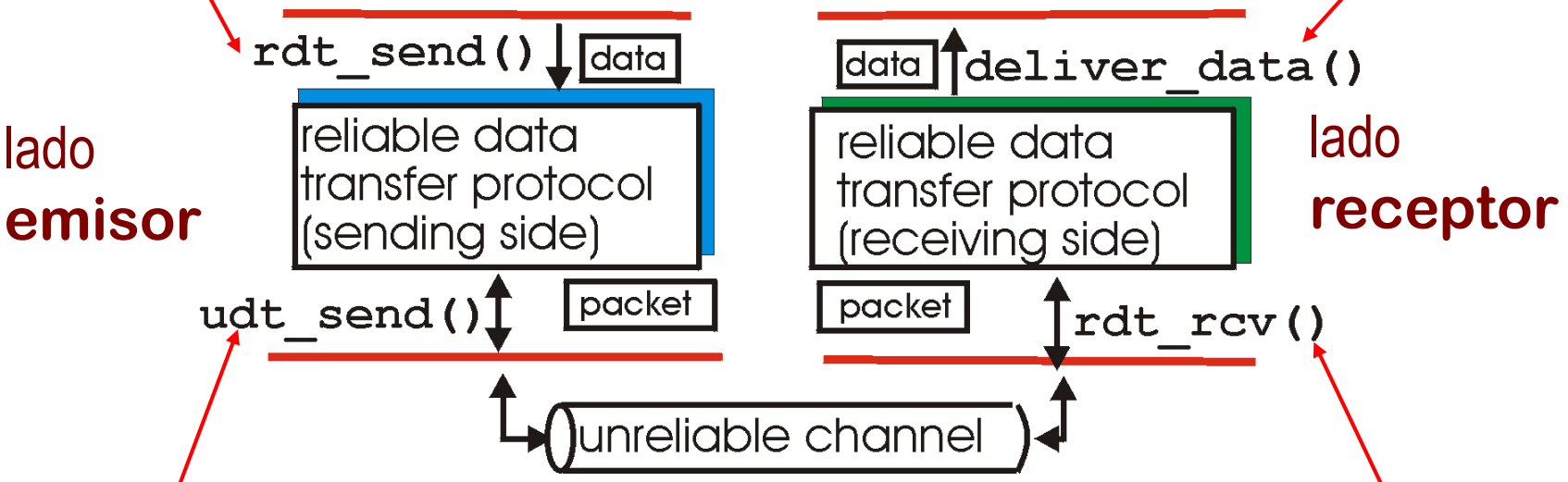
- En una conversación entre dos personas
  - ¿mecanismos para detección y corrección de errores?

- Tareas básicas para una transferencia fiable (suelen requerir cooperación de los extremos que se comunican)
  - Detección del error (en Receptor y/o en Emisor)
    - P.e. uso de información redundante, temporizador en el lado emisor
  - Corrección del error (en Receptor y/o en Emisor)
    - Si existe canal de retorno: técnicas ARQ (automatic Repeat reQuest))
    - Si no canal de retorno: técnicas FEC (Forward Error Correction)

Protocolo de Transferencia Fiable (Reliable data transfer): primitivas del servicio

`rdt_send()`: llamada al servicio,. Se pasan los datos para repartir a la capa superior en el receptor

`deliver_data()`: llamada por `rdt` para repartir los datos a la capa superior



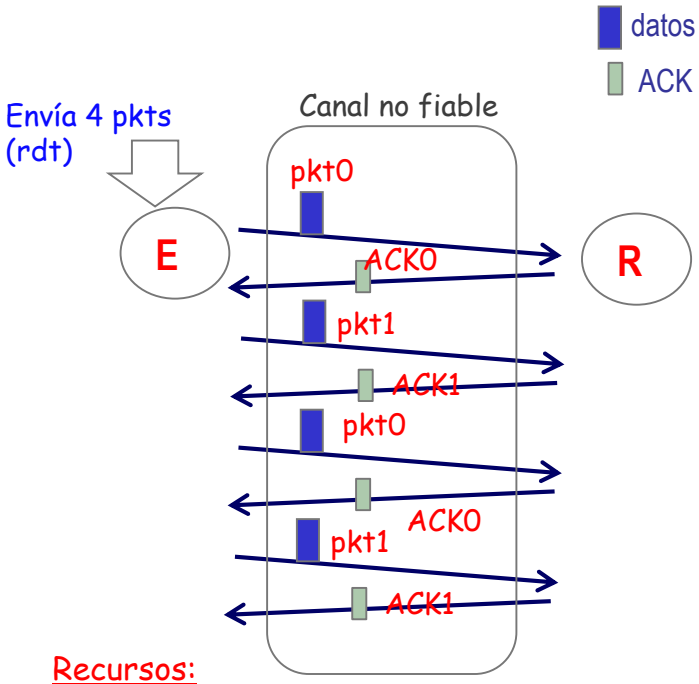
`udt_send()`: llamada por rdt, para transferir paquetes al receptor sobre un canal no fiable

`rdt_rcv()`: llamada cuando un paquete llega por el canal

- **Las características del canal no fiable serán determinantes en la complejidad del protocolo de transferencia fiable (rdt, reliable data transfer)**

Algunos Protocolos Standard

Parada y Espera



Recursos:

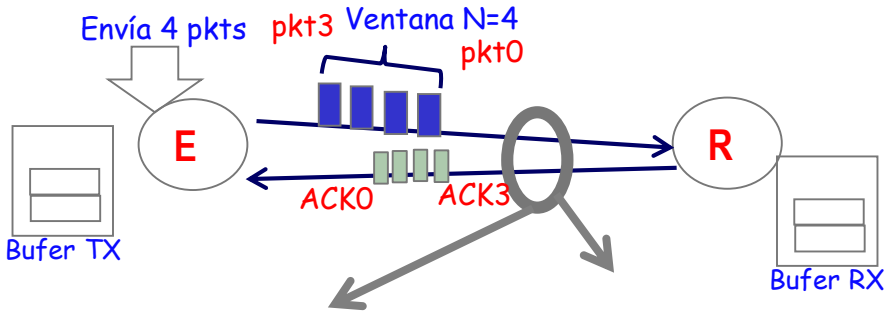
Timer: si en un t no llega el ACK, se retransmite

Números de Secuencia y ACK, identifican a unos paquetes frente a los demás. Permite detectar paquetes duplicados

¿Cuál es mejor para valores altos del producto retardo x throughput extremo a extremo?

Envío Contínuo

- Hasta N ptks pendientes de ACK (ventana)



GoBackN:

Receptor: Asiente el último ptk que le llega en orden. (si un ptk no llega en orden se tira)

Emisor:, Si expira el timer retransmite todos los pkts que tuviese pendientes de recibir ACK (todo el buffer TX)

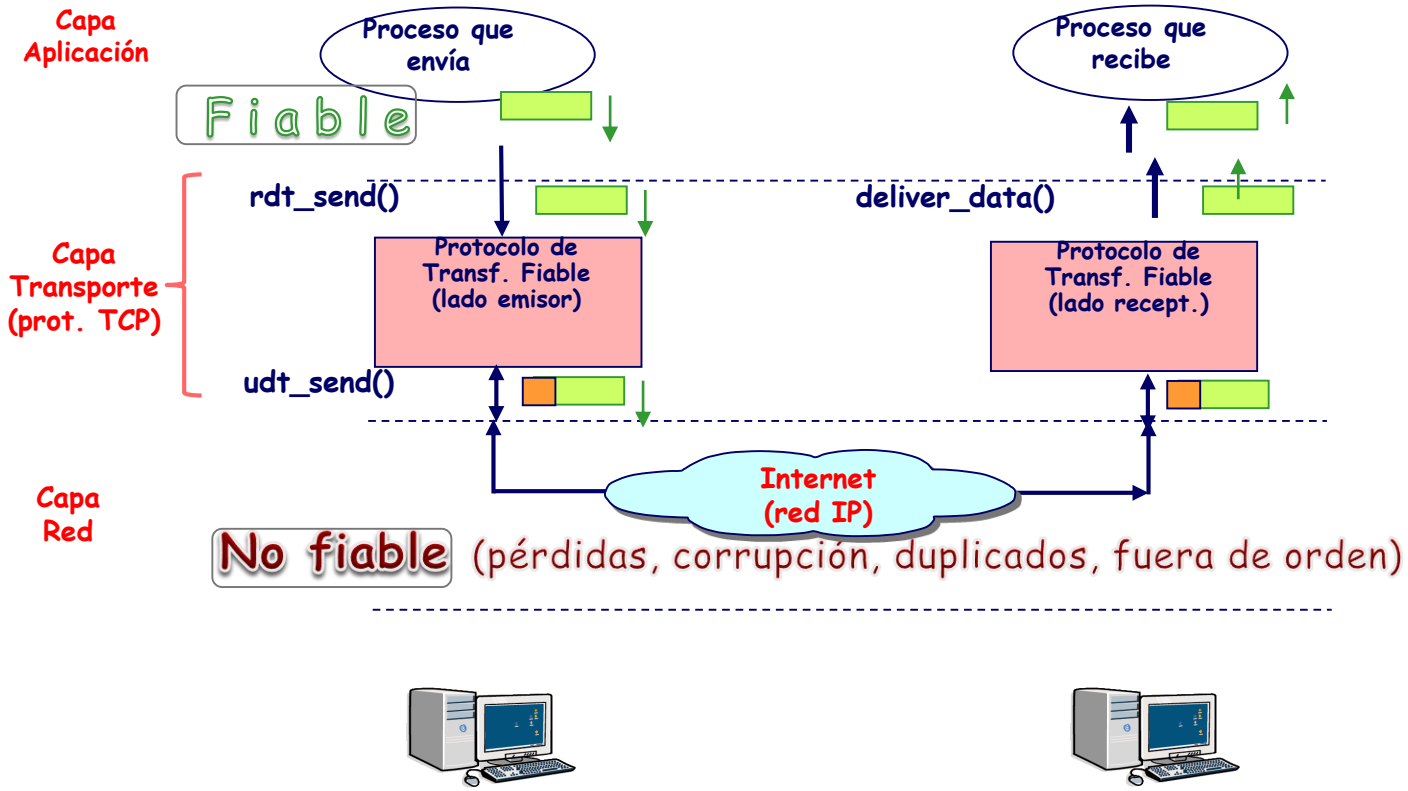
SelectiveRepeat:

Receptor: Asiente cada ptk que le llega.(llegue o no en orden)

Emisor:, Si expira el timer de un pkt retransmite sólo ele pkt (un reloj por cada paquete enviado)

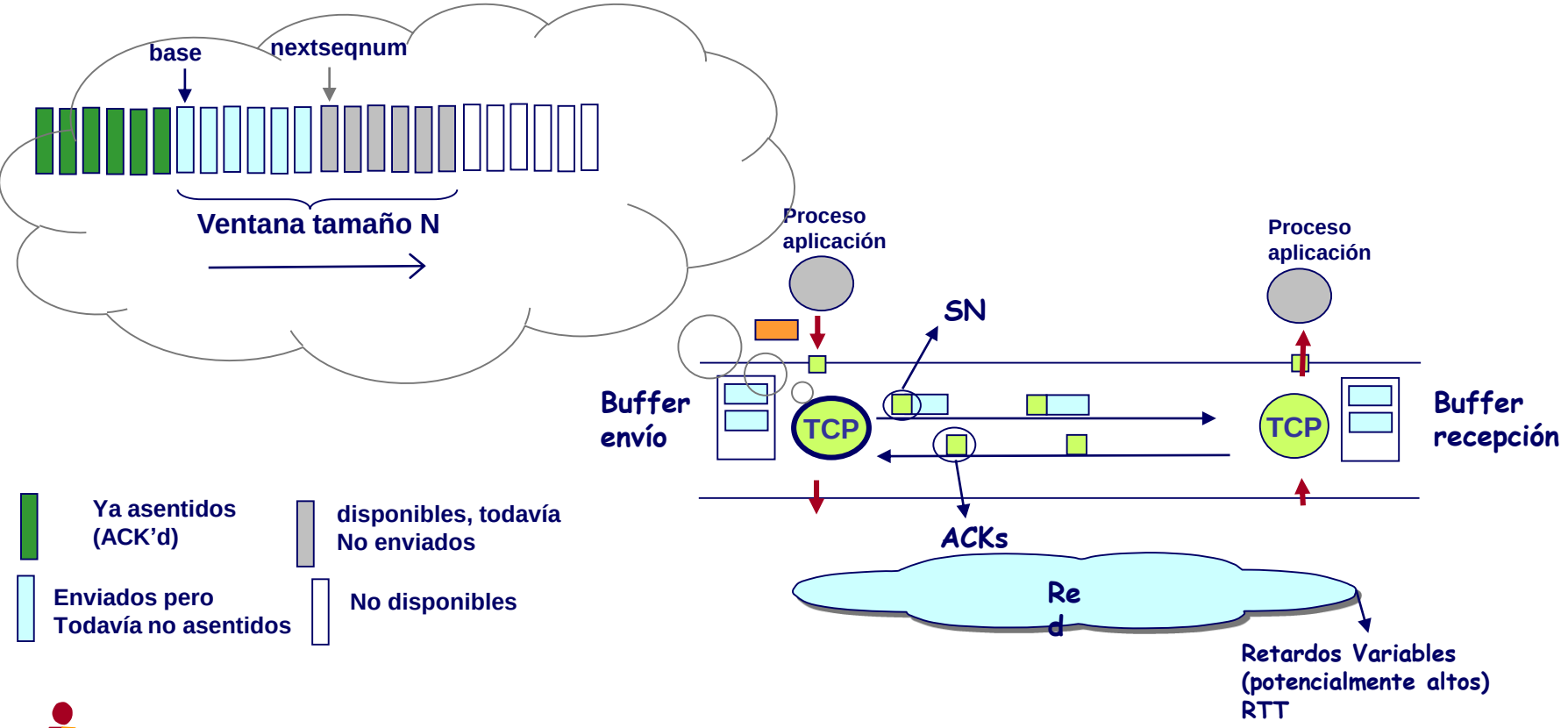
TCP ofrece un servicio de transferencia fiable

- TCP se comporta como un protocolo **propio** de transferencia fiable de **envío continuo**
  - Sirve a los dos procesos de aplicación que usan la conexión.



Esquema de la transferencia fiable en TCP

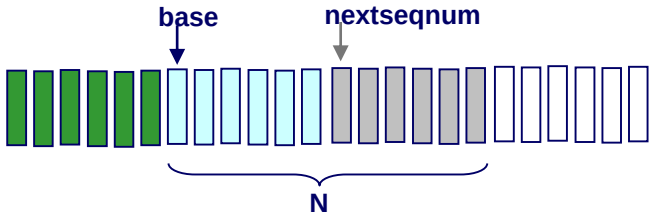
- Carácterísticas principales
  - Full duplex (aunque vemos sólo 1 sentido)
  - Comportamiento híbrido entre GBN y SR
  - Adaptación dinámica (temporizador y ventana)

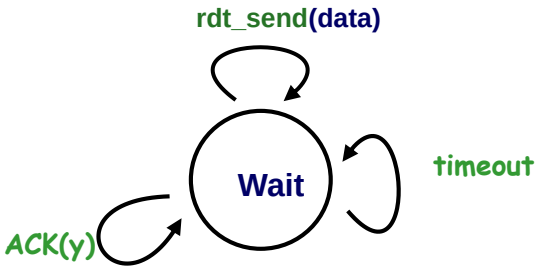


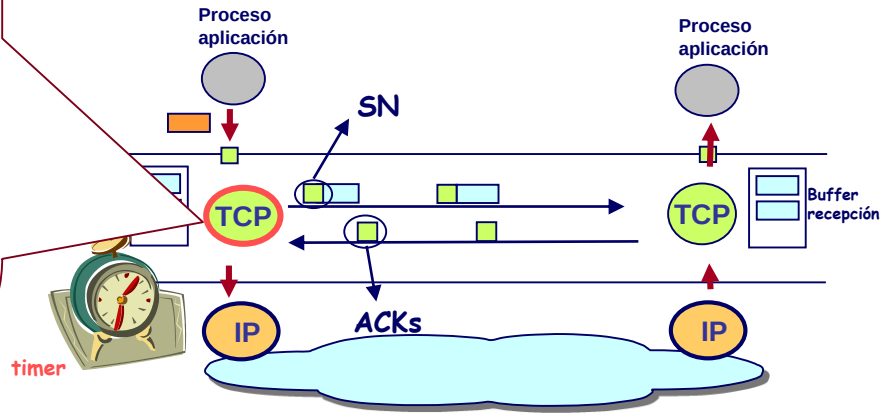
Modelo simplificado: extremo emisor

- Descripción simplificada del emisor TCP
  - Uso de un solo temporizador (timer)
  - Responde ante tres posibles eventos

```
while () {
   switch(evento)

    evento: datos recibidos de aplicación
      crear segmento TCP con SN=NextSeqNum
      if(timer no esta funcionando)
          arrancar timer;
      pasar el segmento a IP;
      NextSeqNum=NextSeqNum + length(data);
      break;

    evento: timeout (expira timer)
      retransmitir el segmento con menor SN
         que no haya sido asentido previamente
      arrancar timer
      break;

    evento: ACK recibido, con valor y
      if(y > SendBase) {
            SendBase = y
            if(existen segmentos pendientes de ACK)
                arranca timer
            }
      break;
}  /* final del bucle infinito*/
```



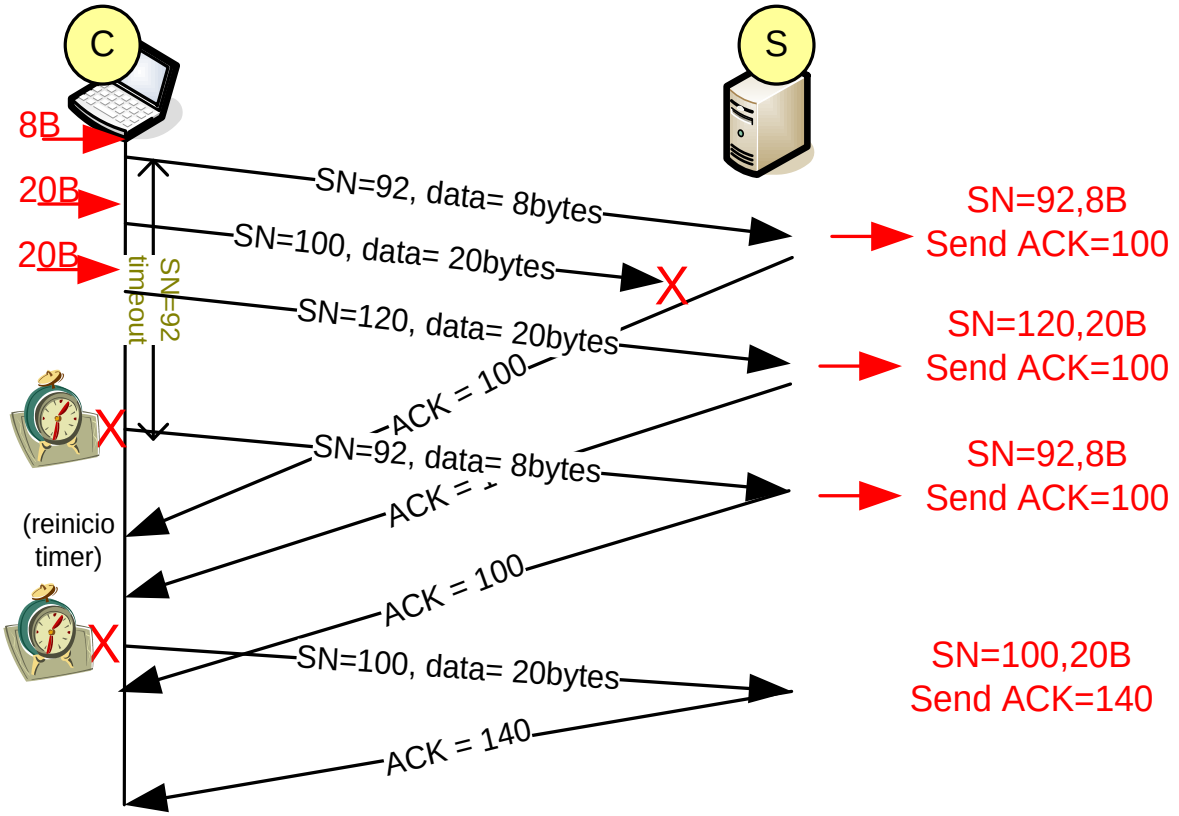




Modelo simplificado: extremo receptor

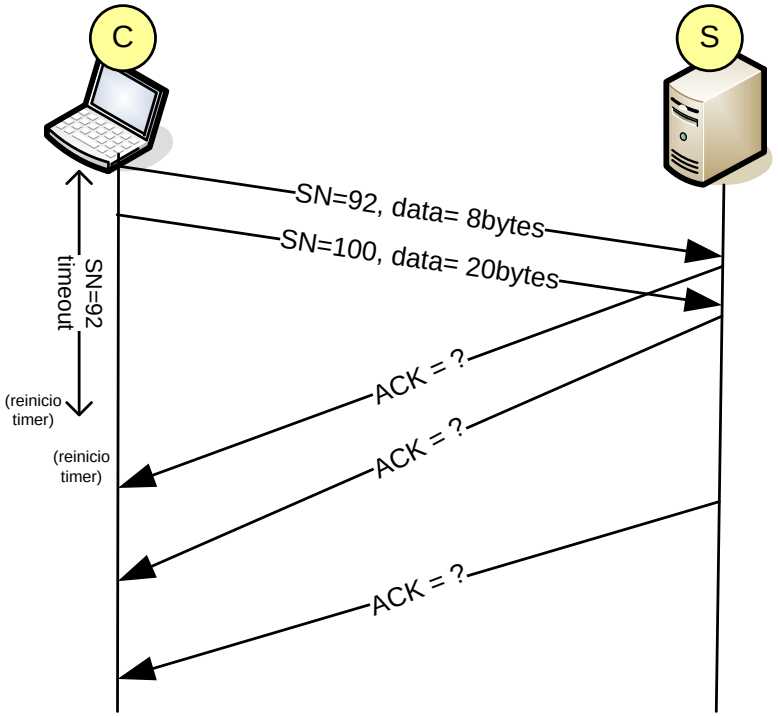
Espera 0.5 seg por si llegan más segmentos en orden (envía ACK acumul.) (RFC 1122, RFC2581)

- Comportamiento **Sólo-ACK**
  - Con cada recepción envía un asentimiento con el último segmento recibido en orden (ACKs acumulativos, similar a GBN)
  - **PERO** guarda los segmentos recibidos fuera de orden (similar a SR)
- Ejemplo:



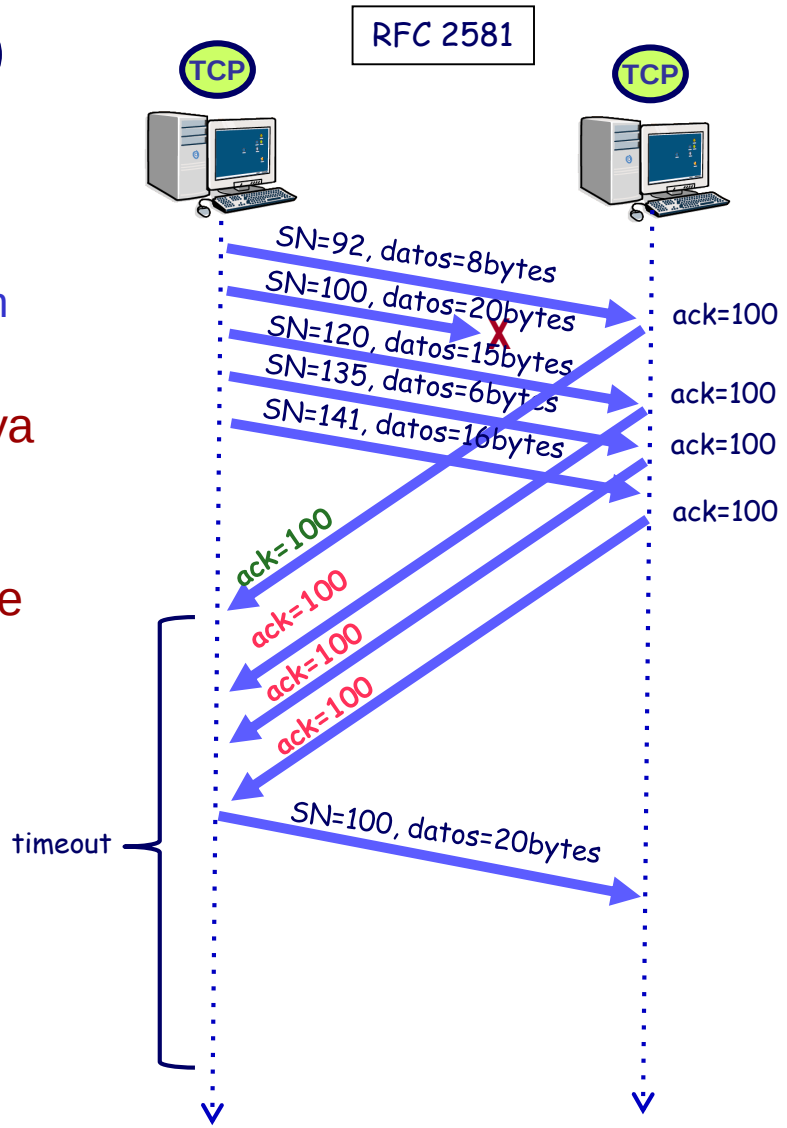
Ejercicio

- Complete (o dibuje) los mensajes de nivel de transporte considerando constante el valor del temporizador
  - ¿cuántas retransmisiones se producen en el lado cliente?

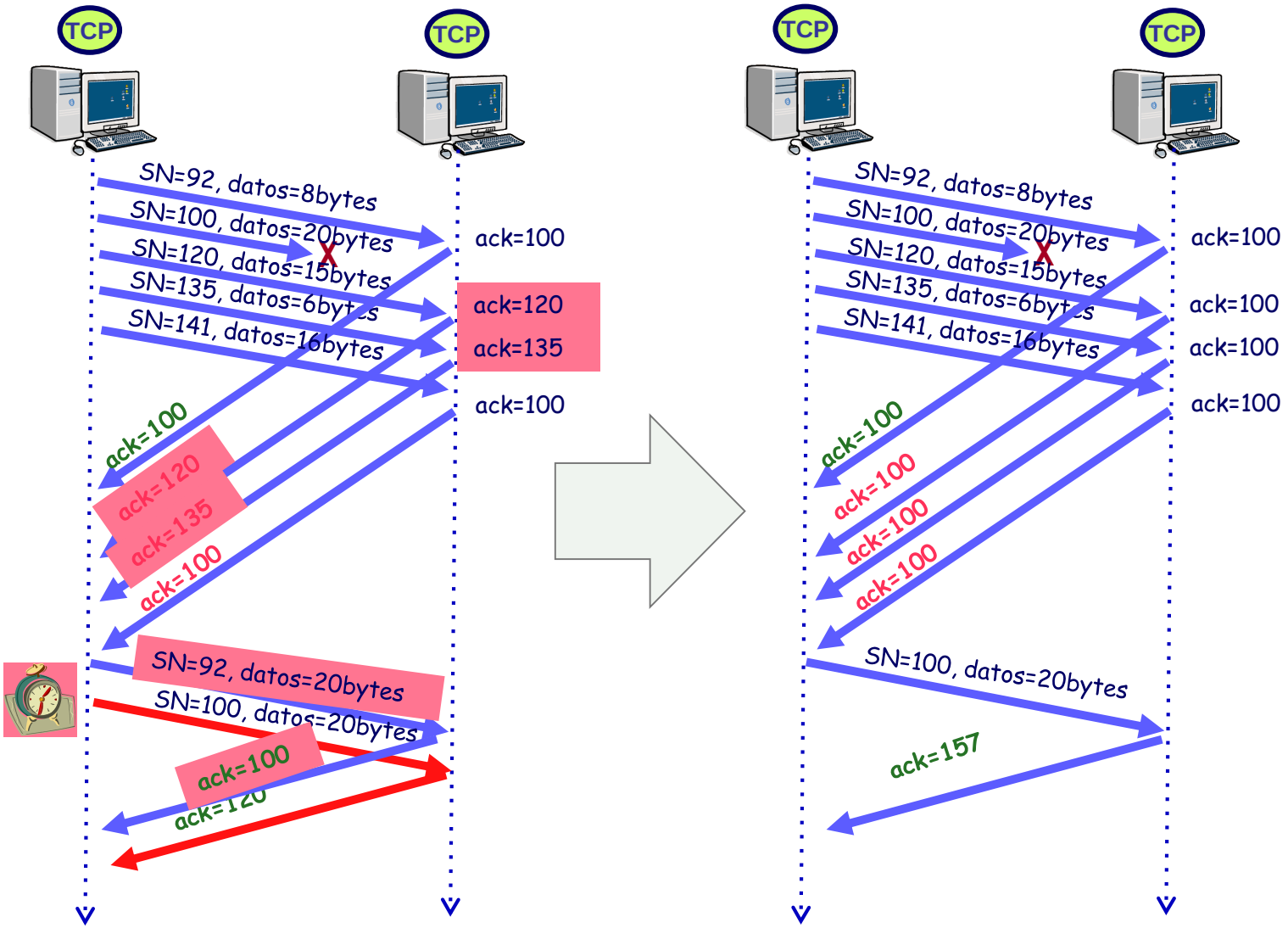


Modificación 1 al modelo simplificado (mejora)

- Retransmisión Rápida (fast-retransmit)
  - Al ocurrir varias retransmisiones seguidas, el temporizador puede aumentar su valor desmesuradamente
    - Se multiplica x2 con cada retransmisión
  - Al recibir 3 ACK duplicados con números de secuencia de segmentos ya enviados asume que el segmento indicado en el ACK se ha perdido y lo vuelve a enviar sin esperar a que expire el temporizador

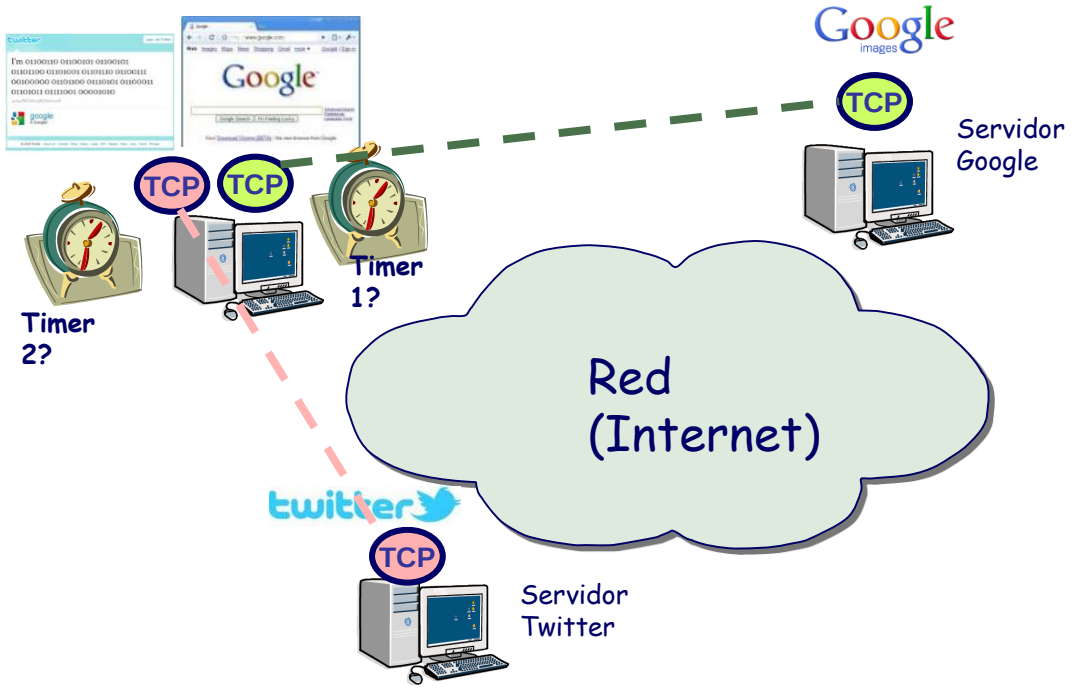


Ejercicio : mostrar los fallos



Estimación del temporizador en TCP

- ¿qué valor ponemos al intervalo del temporizador?
  - En Internet, el retardo en la red es variable
- TCP debe “predecir” continuamente el valor del retardo extremo a extremo y utilizar esta estimación para establecer dinámicamente el valor del temporizador del protocolo

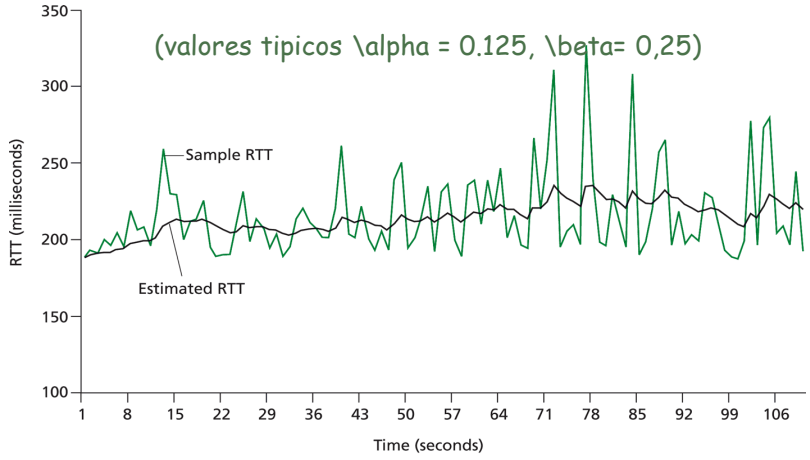


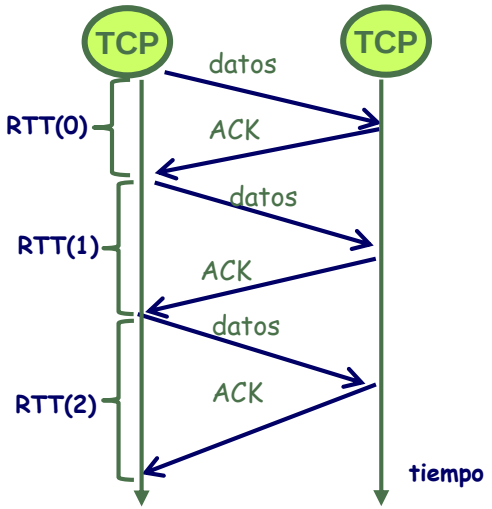
Estimación dinámica del temporizador en TCP

- Supongamos que hay "rondas" de envíos (ciclo datos+ACK). En cada ronda se mide RTT .
- En la ronda (n+1): $t_{out}(n+1) = RTT_{estimado}(n+1) + 4\,Err_{RTT}(n+1)$

  RTO

  (retransmission Timeout)
- Donde
  - **Estimación RTT:** $RTT_{estimado}(n+1) = (1-\alpha)\,RTT_{estimado}(n) + \alpha\,RTT_{medido}(n)$
  - **Estimación error:** $Err_{RTT}(n+1) = (1-\beta)\,Err_{RTT}(n) + \beta\,abs[RTT_{medido}(n) - RTT_{estimado}(n)]$





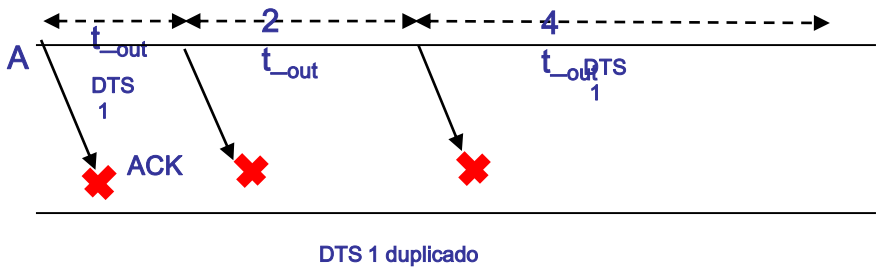
Se cuenta RTT sólo cuando no hay retransmisioness

Opción: TCP Timestamps (RFC1213)
Ambos extremos envían su timestamp en cada segmento

Modificación 2 al modelo simplificado TCP

- Cuando se retransmite, se dobla el valor del temporizador (timeout)
  - Crecimiento exponencial del temporizador ante retransm. consecutivas

$$t_{out}(n+1) = 2\ t_{out}(n)$$



- El temporizador vuelve al valor de la estimación después de la ocurrencia de cualquier otro evento diferente a timeout.

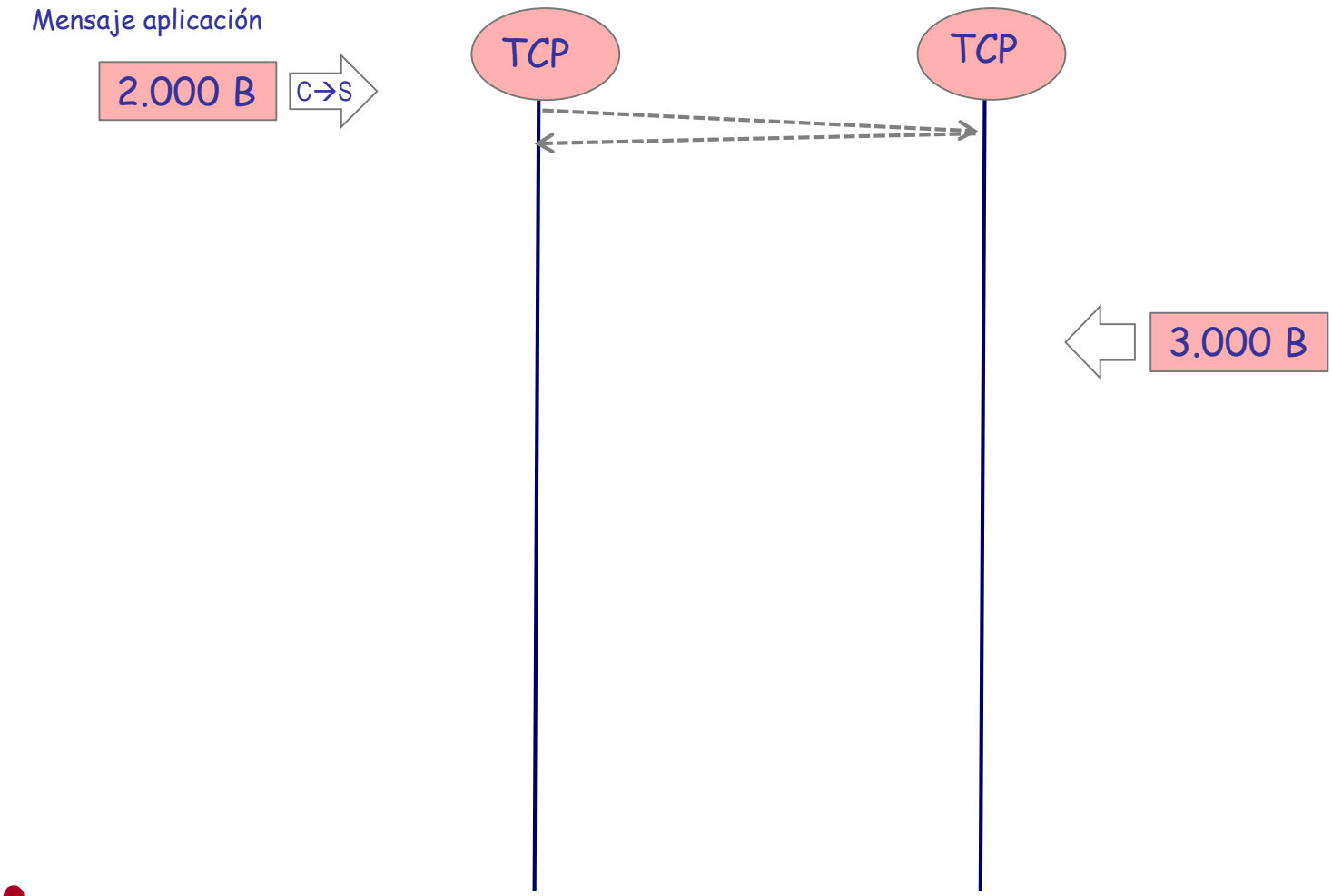
$$t_{out}\ (n+1) = RTT_{estimado}(n+1) + 4\ Err_{RTT}(n+1)$$

Ejercicio

Asume:
MSS = 1.000B, envío contínuo (ventana = 20MSS)
RTO = 2xRTT

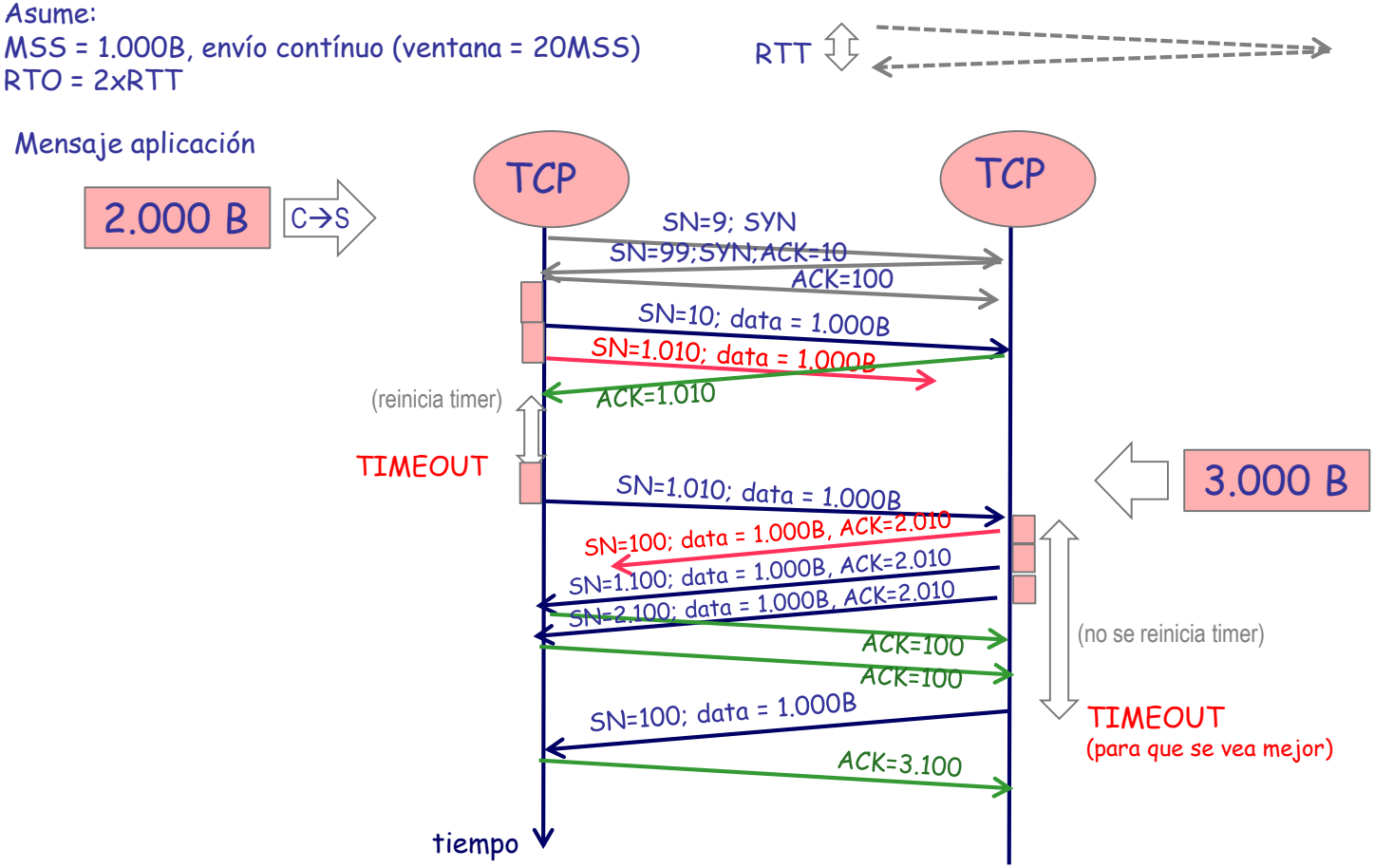
SE PIERDEN:

- El segundo segmento enviado por el cliente (no se incluyen retxn)
- El primer segmento con datos enviado por el servidor



Ejercicio: solución

Asume:
MSS = 1.000B, envío contínuo (ventana = 20MSS)
RTO = 2xRTT

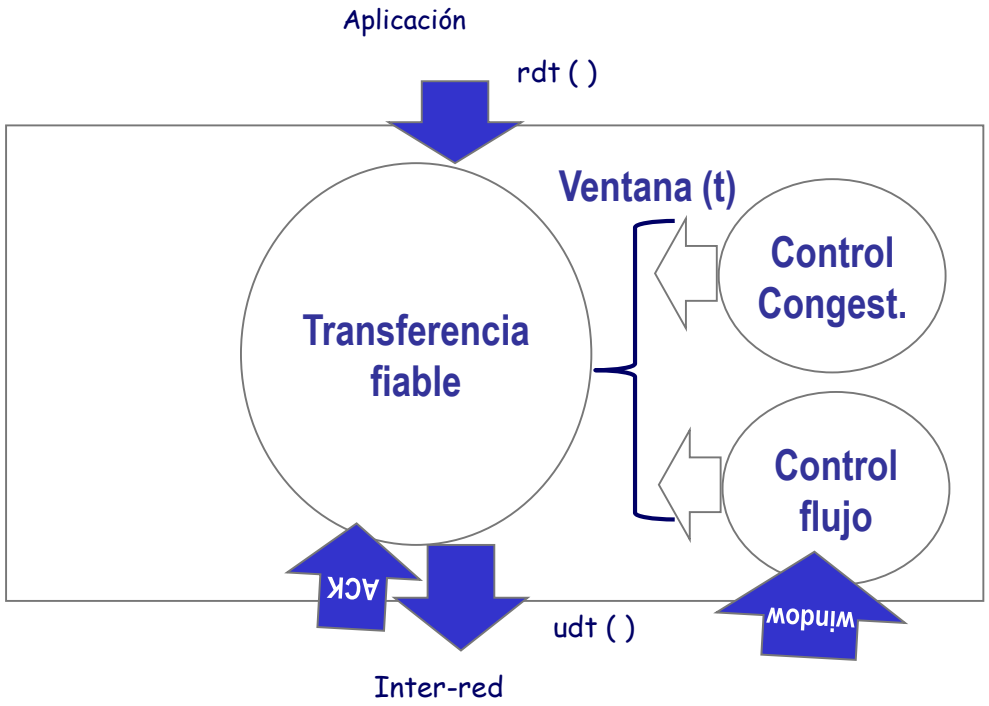


SE PIERDEN:

- El segundo segmento enviado por el cliente (no se incluyen retxn)
- El primer segmento con datos enviado por el servidor

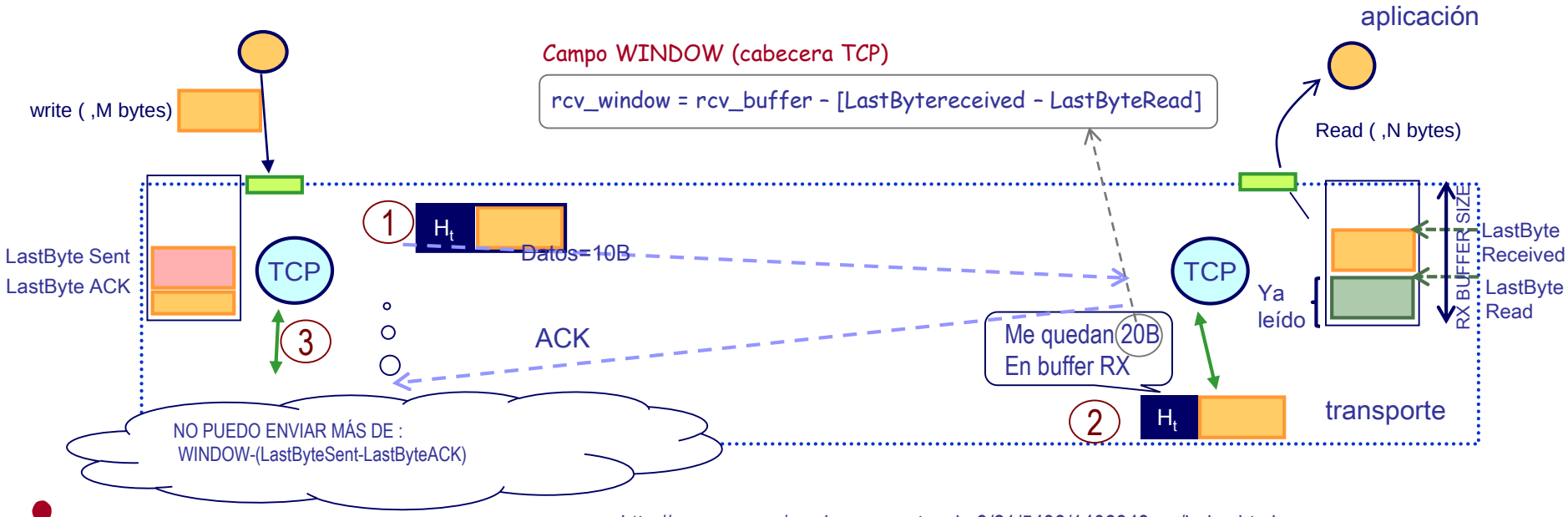
Índice del Tema 03

3.1 Introducción a la capa de transporte: servicios y protocolos
3.2 Servicio básico de multiplexión en la capa de transporte. El protocolo UDP
3.3 El protocolo TCP
3.4 Implementación del Servicio de Transferencia fiable en TCP
3.5 Control de Flujo. Implementación del servicio en TCP
3.6 Control de la Congestión. Implementación del servicio en TCP.



Servicio de control de Flujo en TCP

- "Sincroniza" las velocidades de R/W entre extremos de la conexión
- El lado receptor envía dinámicamente el espacio libre en su buffer
  - Cada vez que genera un segmento (p.e ACK) calcula el valor de rcv_window y lo pone en el campo WINDOW de la cabecera TCP
  - Si se queda sin espacio, envía 0 y el emisor envía 1B
- El lado emisor lee el valor de rcv_window y ajusta su ventana
  - Debe tener en cuenta los bytes pendientes de asentimiento



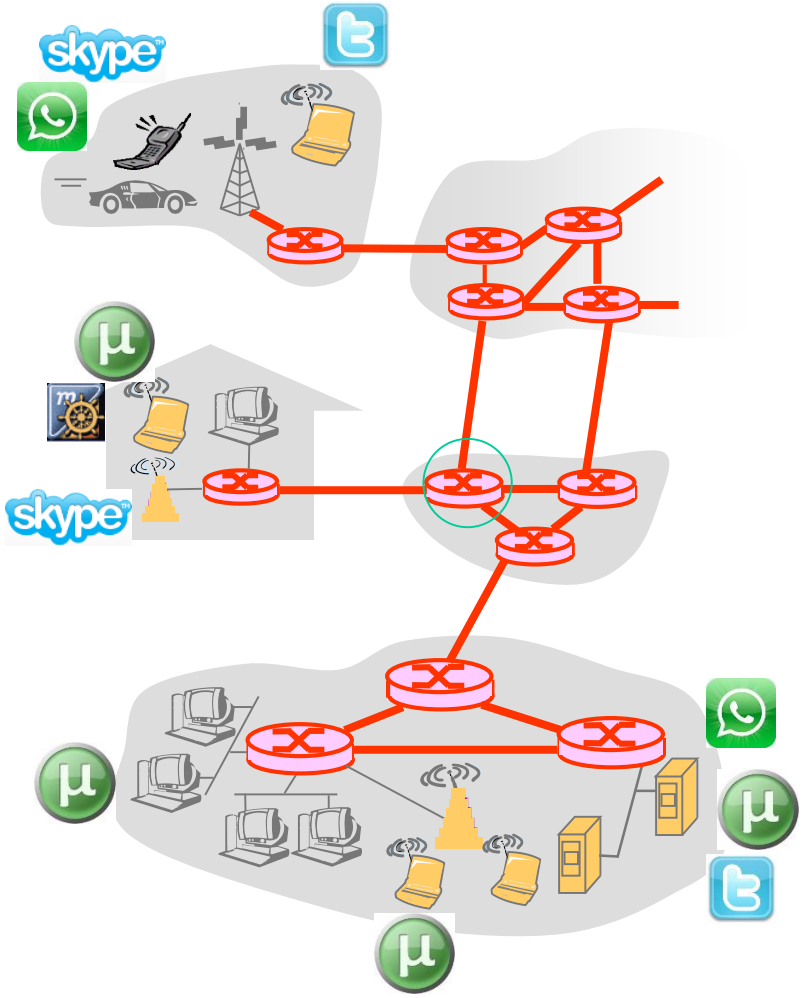
http://wps.aw.com/aw_kurose_network_3/21/5493/1406348.cw/index.html

Índice del Tema 03

3.1 Introducción a la capa de transporte: servicios y protocolos
3.2 Servicio básico de multiplexión en la capa de transporte. El protocolo UDP
3.3 El protocolo TCP
3.4 Implementación del Servicio de Transferencia fiable en TCP
3.5 Control de Flujo. Implementación del servicio en TCP
3.6 Control de la Congestión. Implementación del servicio en TCP.

Costes de la Congestión en una Red de datos

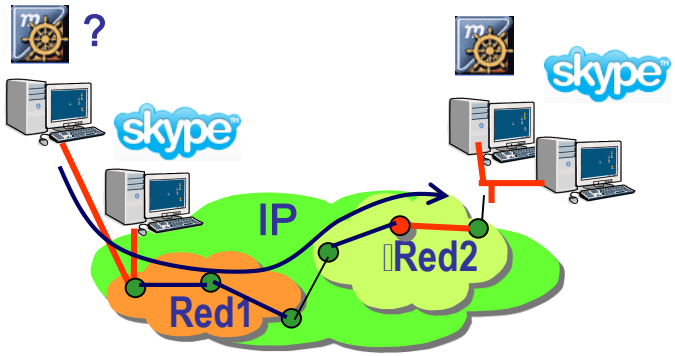
- Causa de la congestión de un sistema: entra más tráfico del que puede salir



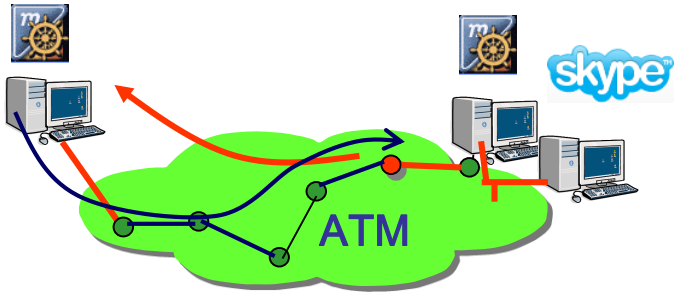
- Costes de la congestión:
  - Retardos,
  - Pérdidas
  - Trabajo en vano

Soluciones al Problema del Control de la congestión

- Extremo a extremo
  - La capa de red no ofrece soporte explícito a la capa de transporte para el control de la congestión
  - La **congestión debe ser inferida por los extremos** observando el comportamiento de la red
  - **TCP sender congestion control" … RFC 2581**

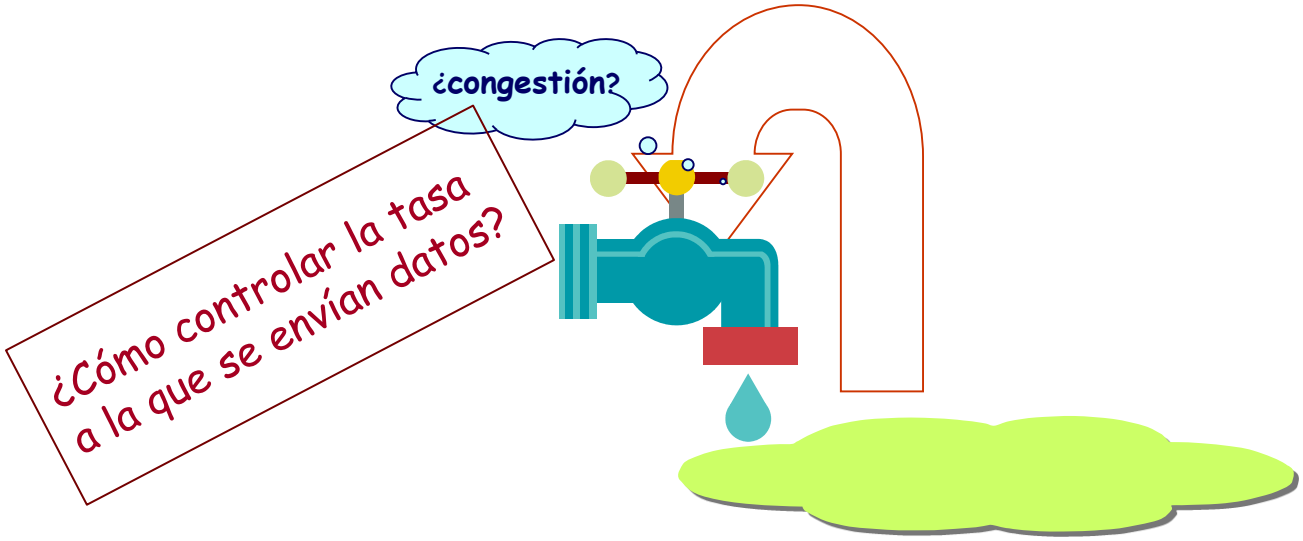


- Asistida por la red
  - Cualquier nodo de la red puede indicar que está en estado de congestión al extremo emisor (p.ej. Redes ATM ABR)
  - existe la opción de ECN RFC 3168 en redes IP



Control de la Congestión en TCP

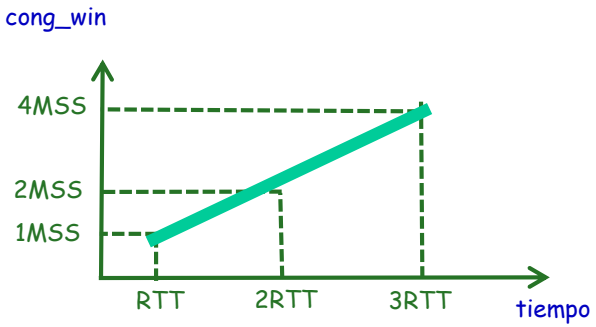
- **Objetivo**: que el emisor TCP envíe lo mas rápido posible sin congestionar la red
- **Descentralizado**: cada extremo emisor TCP ajusta su propia tasa basándose en la siguiente realimentación implícita
  - Recibe ACK → red no congestionada → aumento la tasa de envío
  - Retransmisión → red congestionada → disminuyo la tasa de envío.

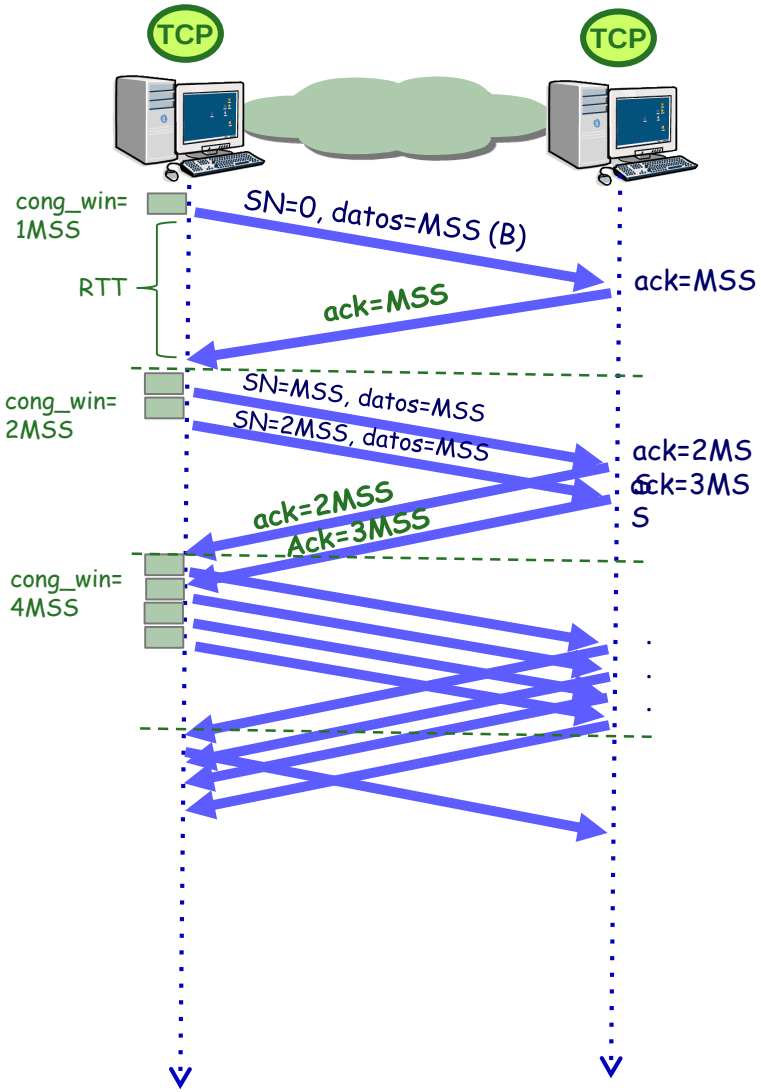


Mecanismo de limitación de tasa de envio

- Ventana de congestión (dinámica)
  - Variable **cong_win**
  - **LastByteSent – LastByteAcked ≤ min{congwin , rcv_window}**
  - Si suponemos que rcv_window no limita
- Tasa media en red *aproximada e ideal* (sin retardo de transmisión ni pérdidas)

$$\text{tasa} = \frac{\text{cong\_win}}{\text{RTT}} \text{ bytes/seg}$$



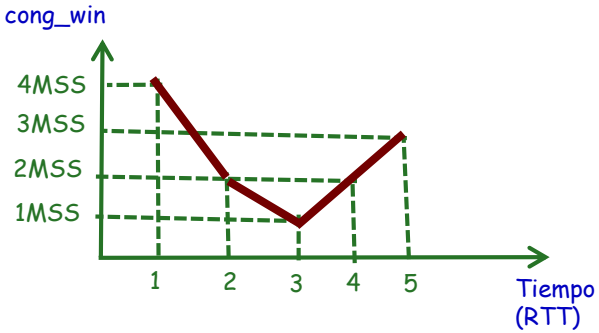


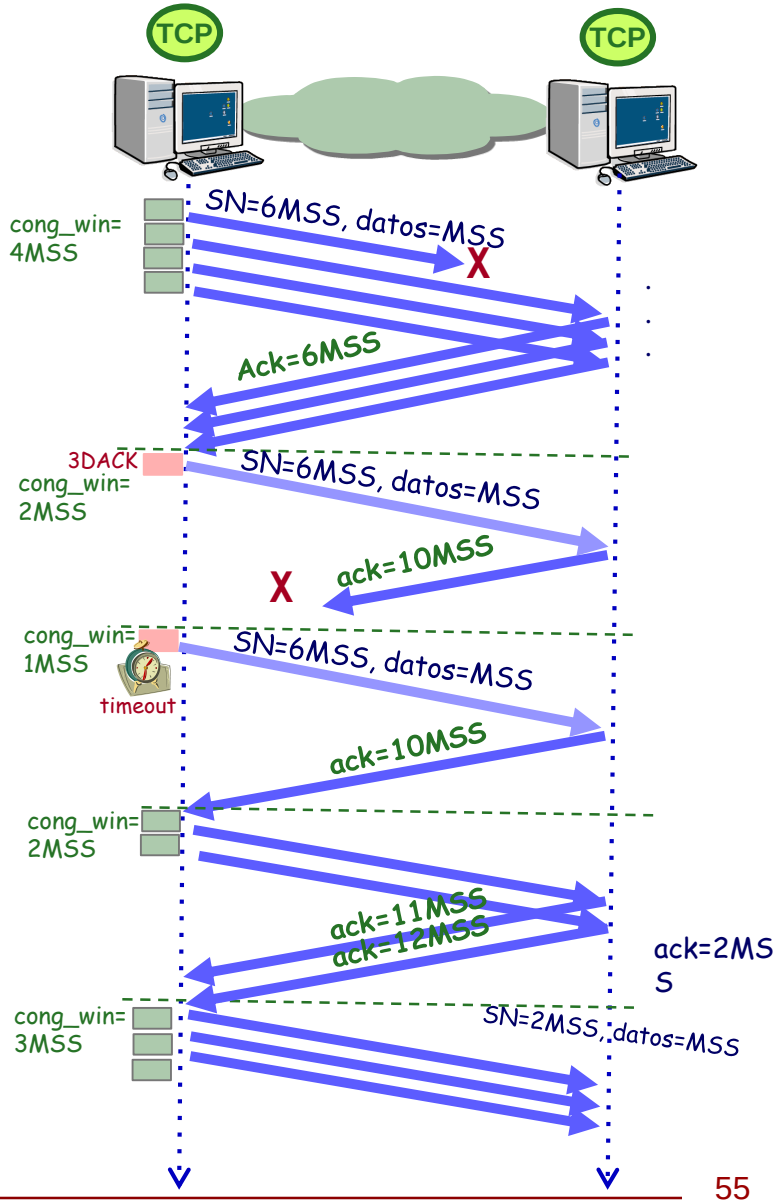
Idea 1: variación de la tasa de envio (si hay problemas, cierra rápido el grifo)

1. Incremento aditivo, decremento multiplicativo
   - Ante una retransmisión
     - `cong_win (n+1) = cong_win(n)/2`
   - Si hay retransmisiones consecutivas

     16MSS →8MSS →4MSS → 2MSS → MSS

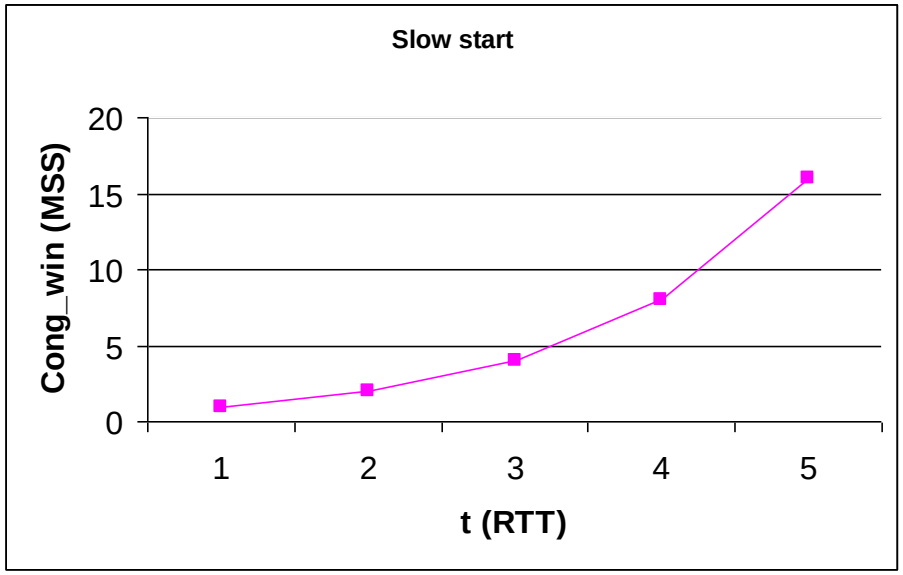
     (decremento multiplicativo)
   - Si ronda correcta, se incrementa en un MSS
     - Ejemplo: MSS → 2MSS → 3MSS → 4MSS …

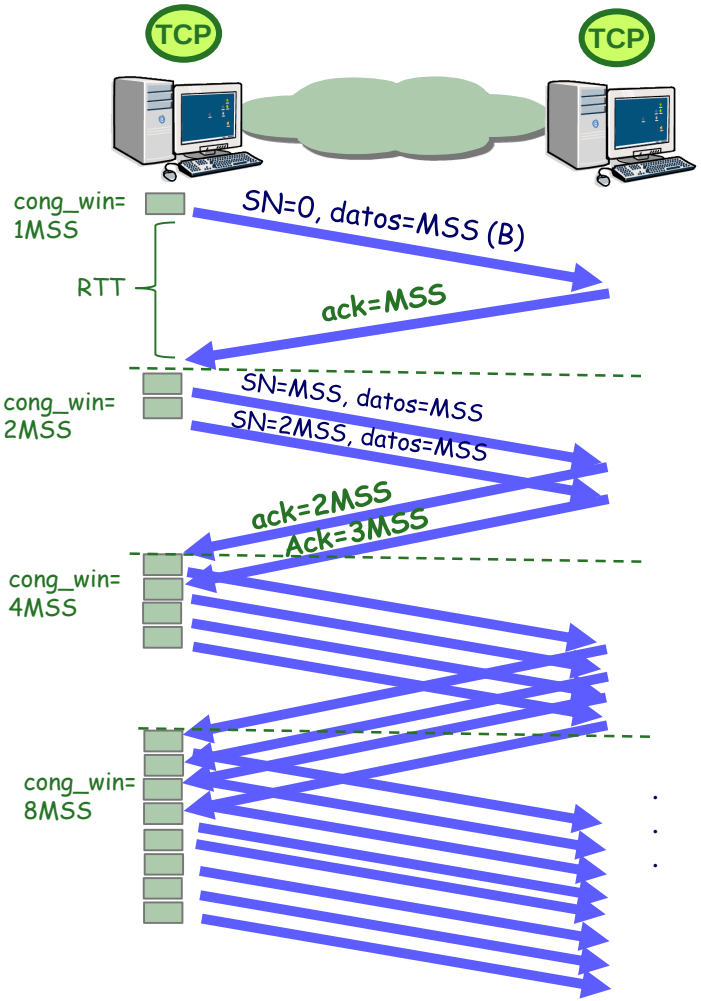




Idea 2: Si no hay problemas, abre el grifo más rápido

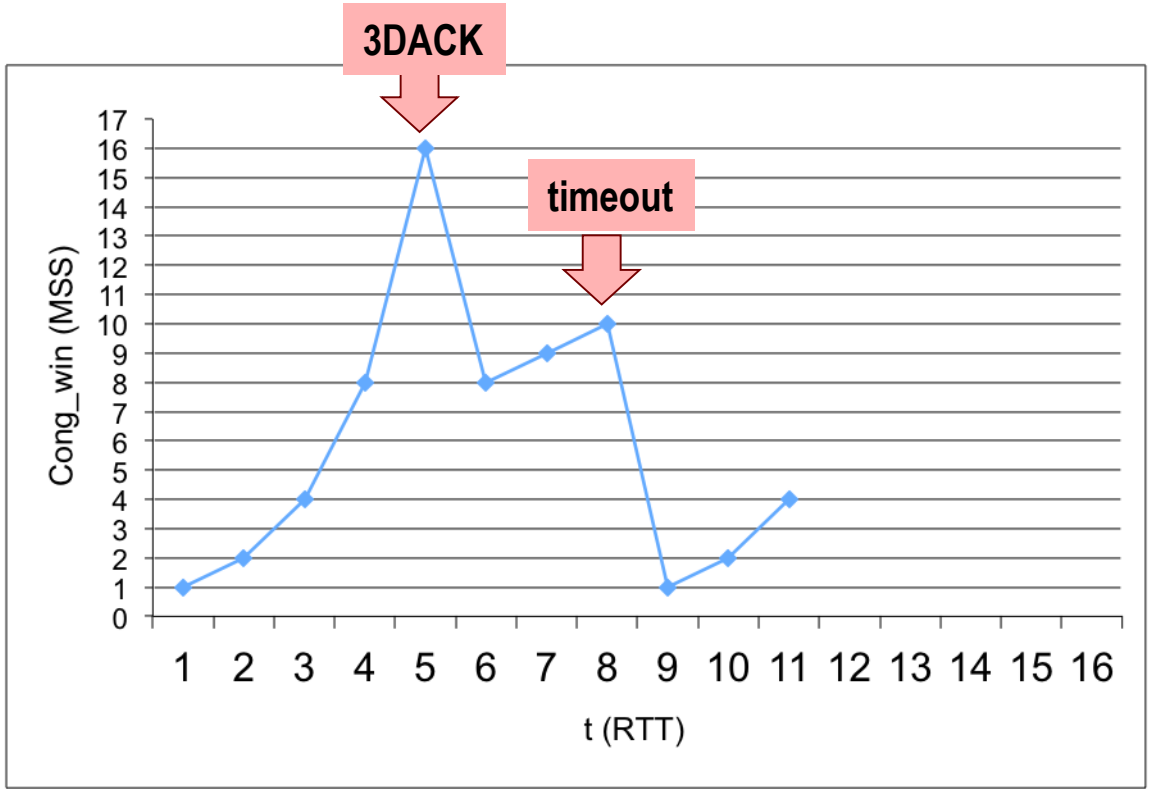
2. SlowStart (arranque lento) [rfc 3390]
   - Al comienzo de cada conexión cong_win = MSS
     - Pobre tasa inicial … quizás no hay congestión
   - Incremento de un MSS con cada ACK recibido
     - Multiplico por 2 en cada RTT
       - cong_win(n+1)= 2 x cong_win(n)
   - Hasta el primer evento de pérdidas
     - Posteriores incrementos serán lineales (aditivos)





Idea 3: ten en cuenta el motivo de la retransmisión

3. Reacción ante eventos de retransmisión según el tipo de evento
   - Triple ACK duplicado → divido por 2 la ventana
   - Timeout → ventana a 1MSS y comienza arranque lento

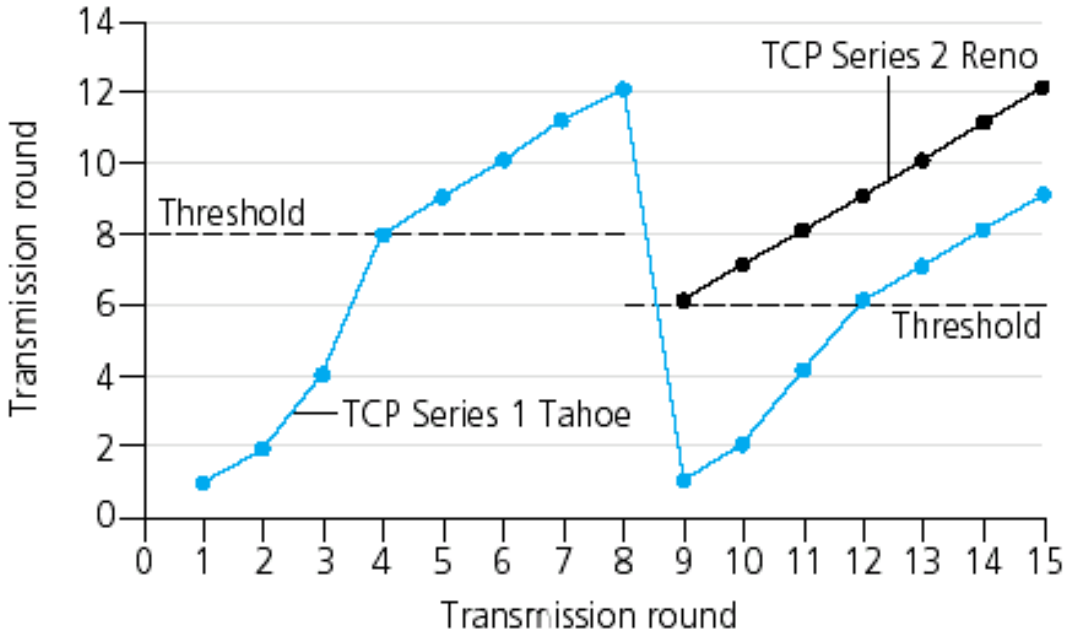


Algoritmo completo (RFC 2581)

- Variable ssthresh: indica cuándo termina la fase de slow start
- Algoritmo de Control de la congestión: versión completa incluyendo el uso del umbral ssthresh [RFC 2581][Jacobson88]
  - Cuando `cong_win < ssthresh`,
    - el emisor en fase slow-start, la ventana crece exponencialmente..
  - Cuando `cong_win >= ssthresh`,
    - el emisor en fase de evitar-congestión, la ventana crece linealmente.
  - Cuando ocurre triple duplicate ACK,
    - `ssthresh` ← `cong_win/2`, `cong_win` ← ~ `ssthresh`
  - Cuando ocurre timeout,
    - `ssthresh` ← `cong_win/2`, `cong_win` ← 1 MSS

Control de Congestión

- Distintas versiones de TCP presentan pequeñas variaciones de comportamiento (cada extremo es independiente)



Modelo simple del Throughput medio de una conexión

$$\frac{1.22 \cdot MSS}{RTT\sqrt{L}}$$

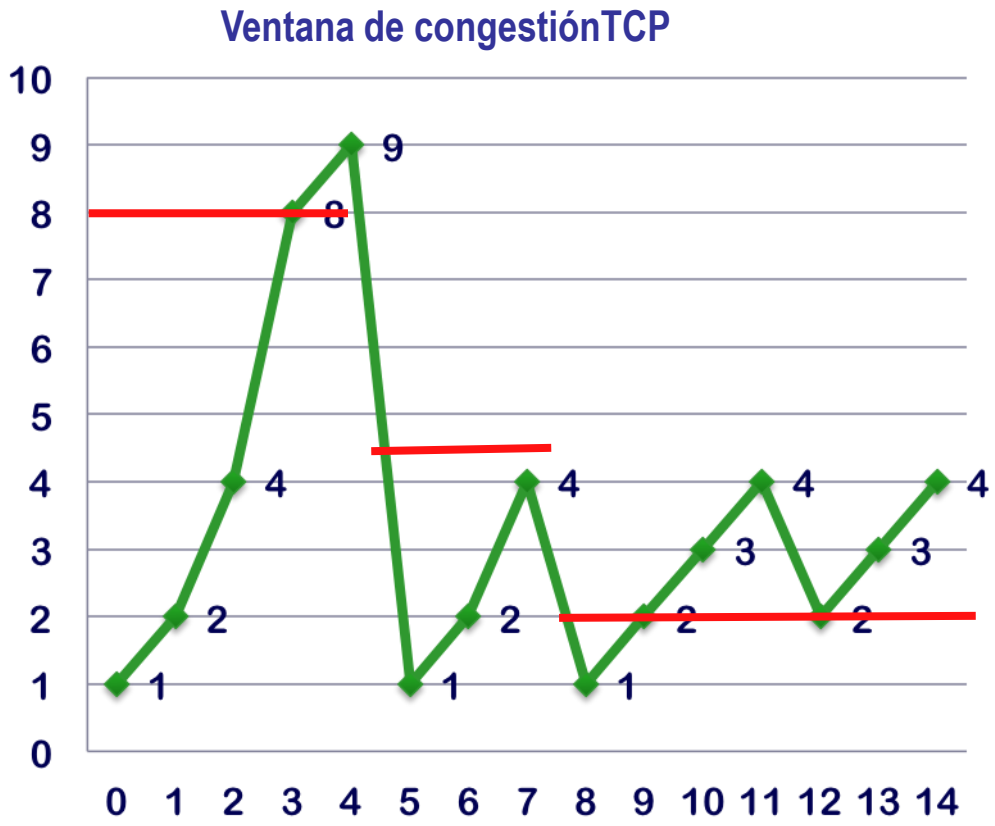
Packet Loss (tasa media)

- Nuevas versiones de TCP para casos de alto BWxretardo
  - P.e. Se necesita $L<2^{-10}$, (MSS 1500B, RTT 100ms) para 10Gbps

[Jin2004][RFC3649][Kelly 2003][Ha2008]

Ejercicio

- Dibuje una figura donde se especifique el tamaño de la ventana de congestión cuando ocurren los siguientes eventos: (en el orden indicado). Suponga ssthresh con un valor inicial de 8 MSS.
  - T=0 inicio con arranque lento
  - T=4RTT , pérdida por temporizador
  - T=7RTT , pérdida por temporizador
  - T=11 RTT, pérdida por triple ACK

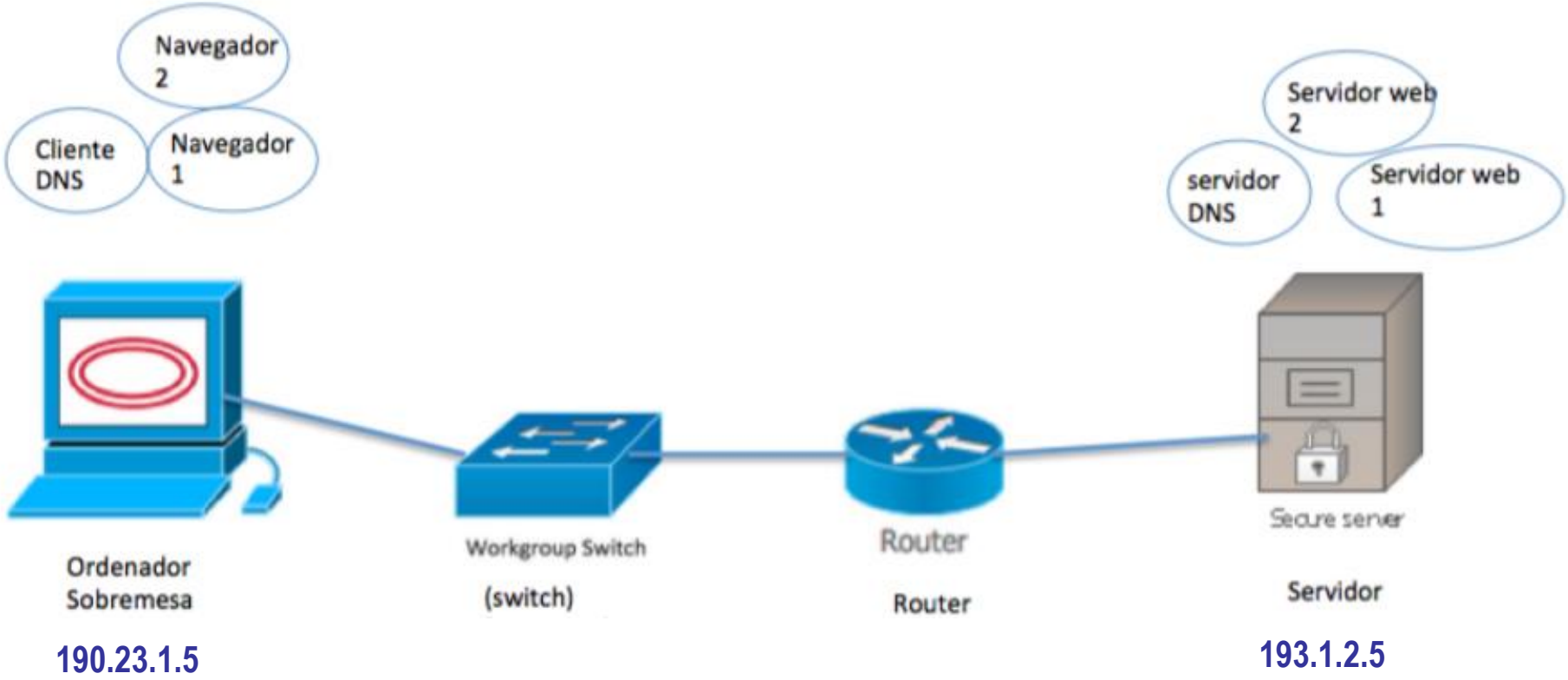


¿cuál es el throughtput de la conexión?

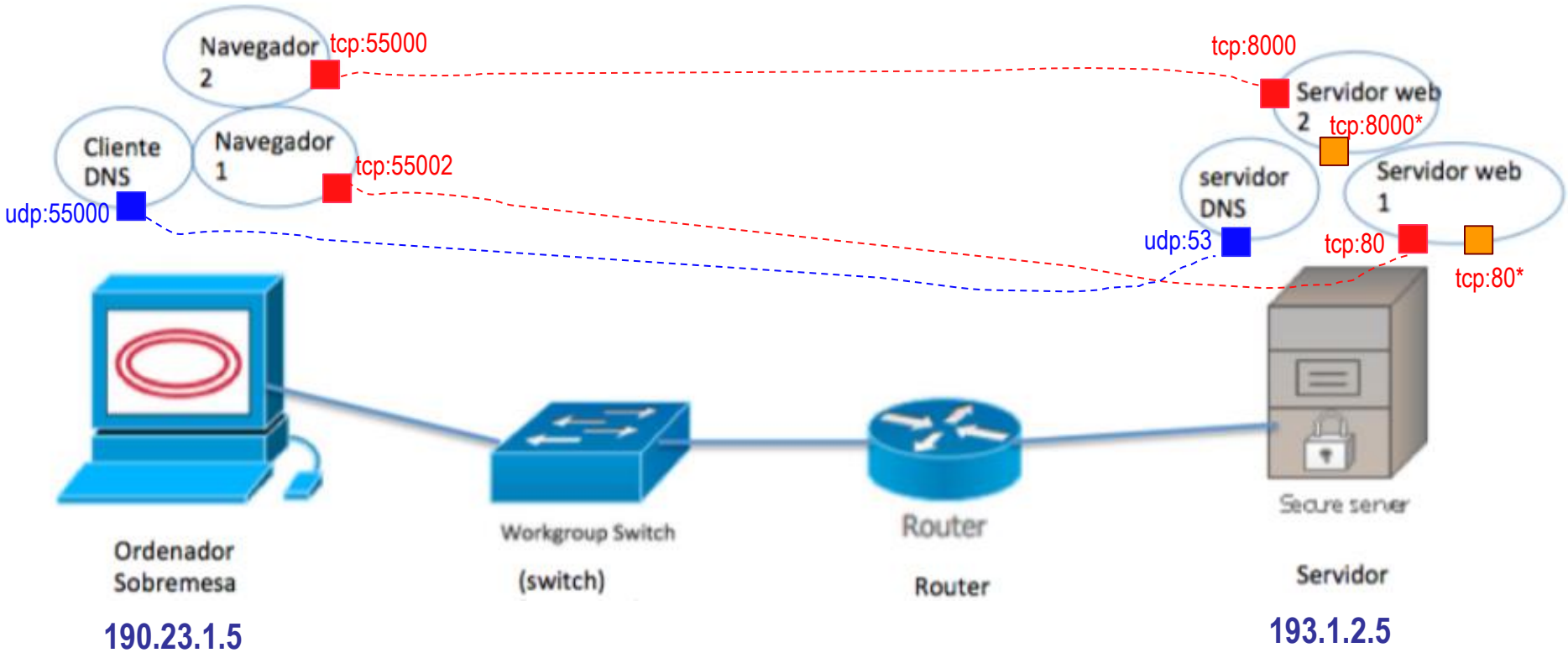
EJERCICIOS DEL TEMA 03

Ejercicio 1.

- **Multiplexión con sockets**
  - Dibuje los sockets utilizados por cada proceso de la figura, suponiendo que cada proceso cliente intercambia mensajes con su respectivo proceso servidor. Invéntese los números de puerto que considere necesarios.
  - Escriba la tabla de sockets activos del Ordenador de Sobremesa



Solución



Protocolo	IP_local:Puerto_local	IP_rem:Puerto_rem
udp	190.23.1.5 : 55000	*:*
tcp	190.23.1.5 : 55000	193.1.2.5 : 8000
tcp	192.23.1.5 : 55002	193.1.2.5 : 80

Protocolo	IP_local:Puerto_local	IP_rem:Puerto_rem
udp	193.1.2.5: 53	*:*
tcp	193.1.2.5 : 8000	*:*
tcp	193.1.2.5 : 80	*:*
tcp	193.1.2.5 : 8000	190.23.1.5 : 55000
tcp	193.1.2.5 : 80	190.23.1.5 : 55002

EJERCICIO 2: cabecera UDP

- **UDP**
  - Una aplicación (IP1:192) quiere enviar el mensaje “hi” a otra aplicación (IP2:256).
  - ¿Es correcto el segmento UDP generado? (checksum sólo protege al segmento)

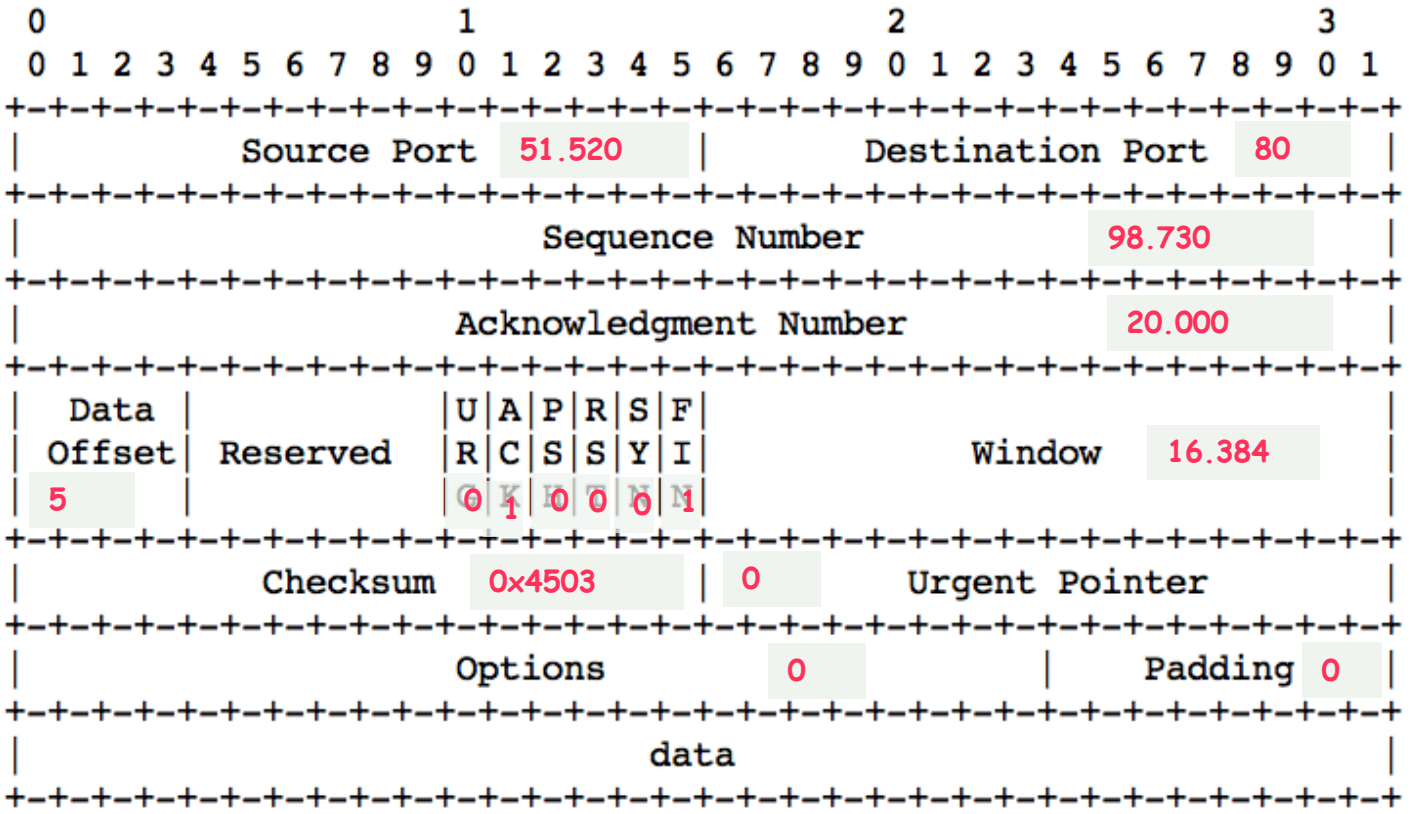
Puerto Org.	Puerto Dest.	Tamaño	Checksum	Datos encapsulados
192	256	10	0x6A33	0110 1000 0110 1001

192 → 0000 0000 1100 0000
256→ 0000 0001 0000 0000
0000 0001 1100 0000
10→ 0000 0000 0000 1010
0000 0001 1100 1010
hi→ 0110 1000 0110 1001
0110 1010 0011 0011 → 0x6A33

(ASCII)
h → 0x68;
i→0x69

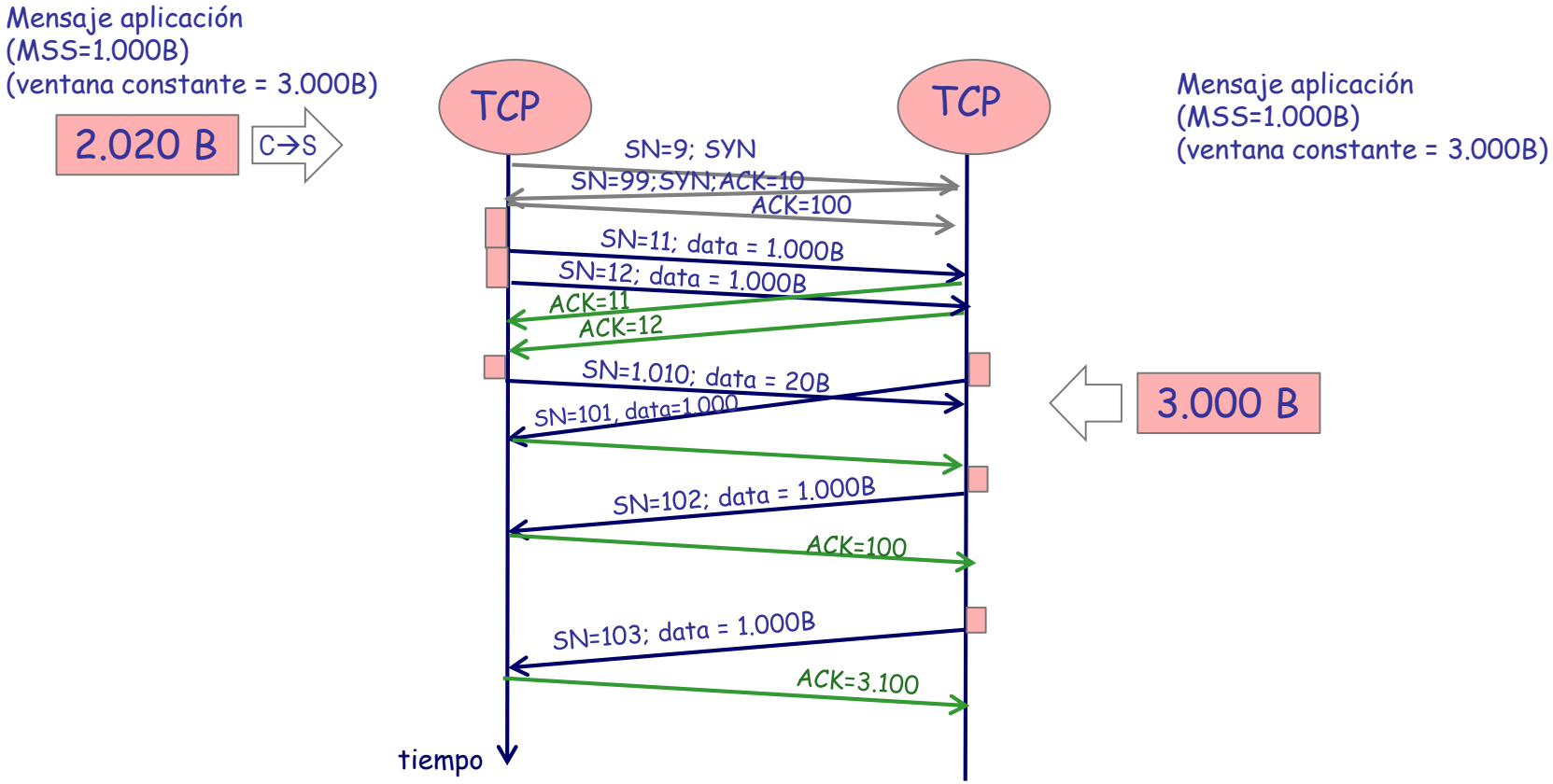
Ejercicio 3: cabecera TCP

- ¿Qué información puedes obtener de esta cabecera TCP?



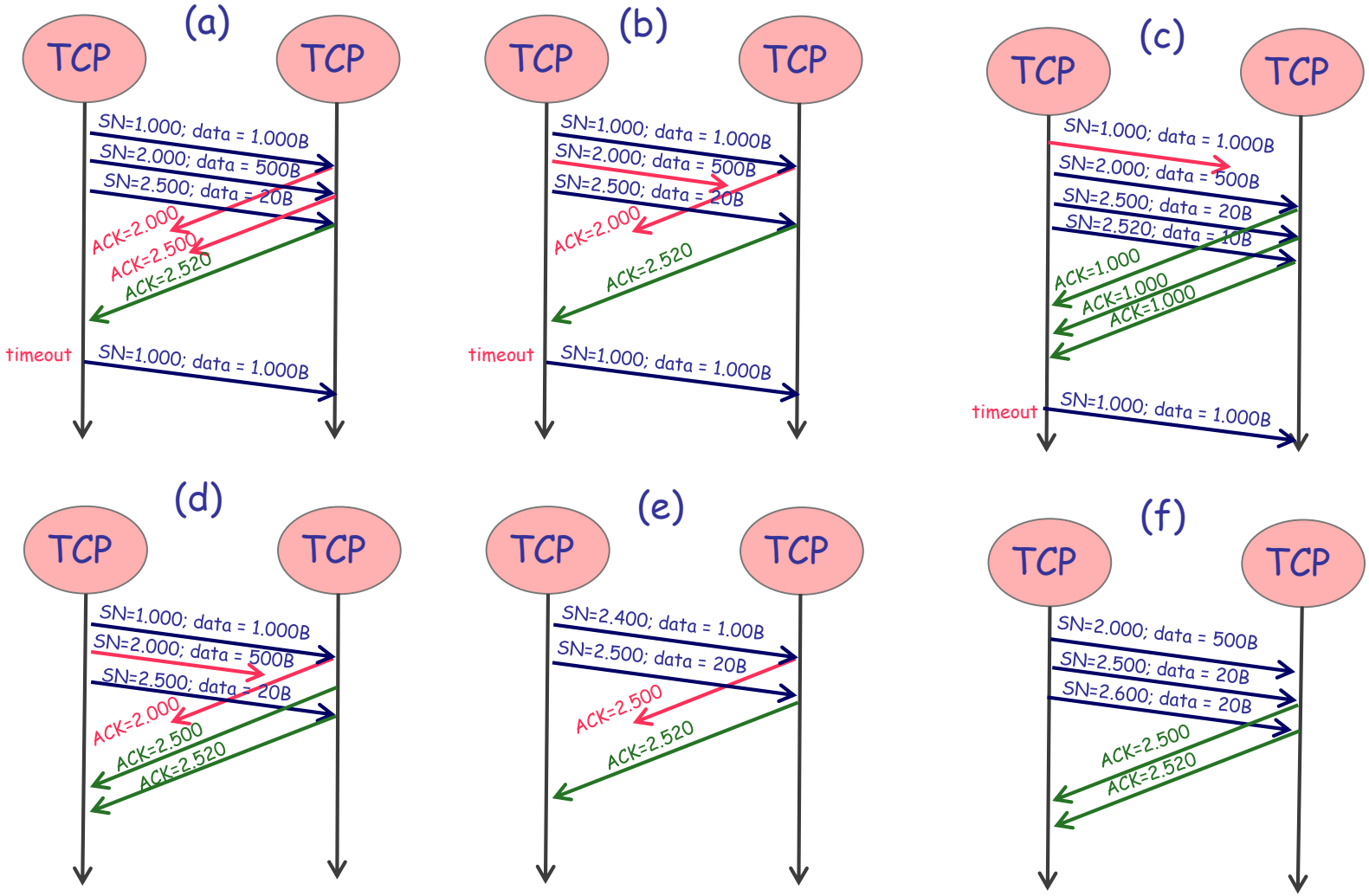
EJERCICIO 4: núm. Sec. y ACKs

- **Encuentre y corrija errores en el siguiente intercambio de mensajes**



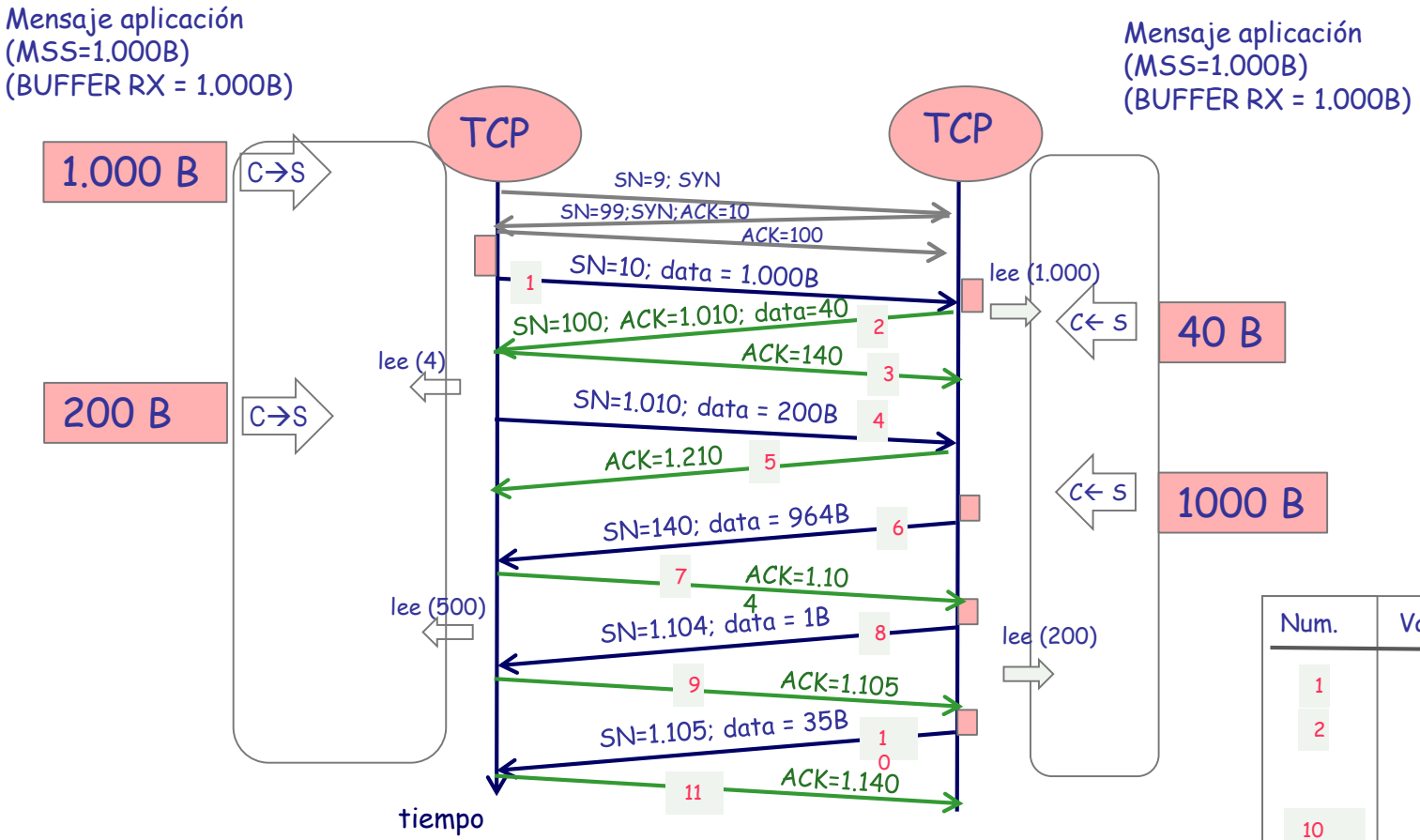
Ejercicio 5: transferencia fiable

- Justifique si son correctos o no los siguientes casos



Ejercicio 6: control flujo

- Indicar los valores de window de cada segmento



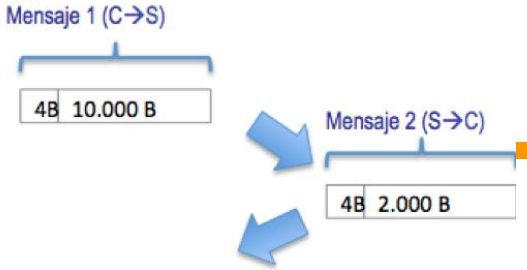
Ejercicio 7: control congestión

- Dibujar en un diagrama la dinámica de la ventana del control de congestión. ¿cuál es el throughput de la conexión hasta el instante 120 ms? (supón que siempre hay datos que TX)
  - Datos:
    - RTT constante = 10ms, comience la figura por 0 ms, MSS = 1.000 B, sshthres = 8MSS

Instante (ms)	evento
3	Retransmisión por timeout
8	Retransmisión por 3 ACKs
10	Retransmisión por timeout

SOL:
(TIEMPO;
CONG_WIN;SSHTH
)
(0,1,8)
(1,2, 8)
(2,4, 8)
(3,8, 8)
(4,1,4)
(5,2,4)
(6,4,4)
(7,5,4)
(8,6,4)
(9,3,3)
(10,4,3)
(11,1,2)
(12,2,2)

Ejercicio 8: todo junto …



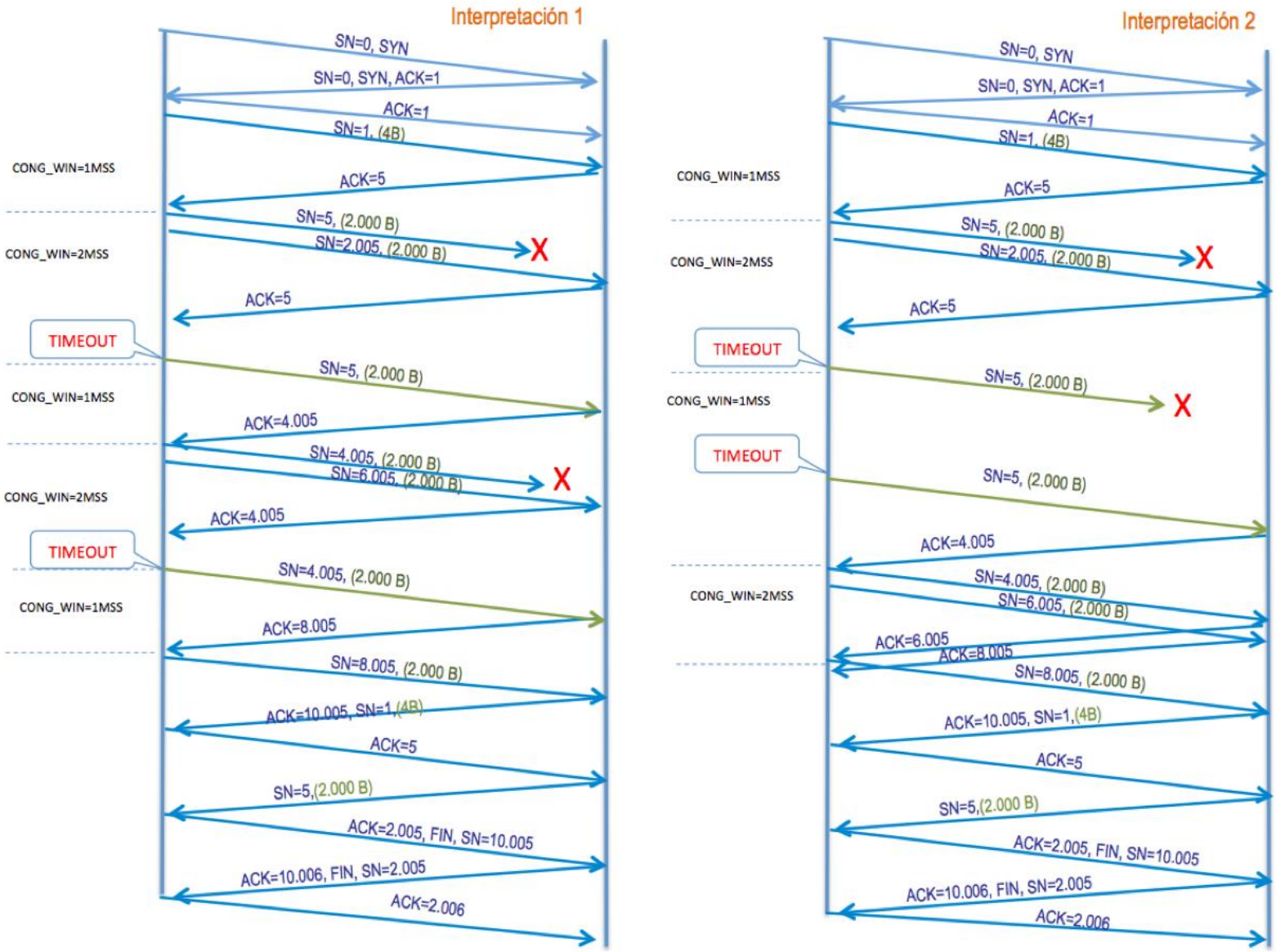
- **Examen curso 2013/14**

Imagínese un intercambio de mensajes entre dos procesos de aplicación a través de TCP. El proceso cliente le envía un fichero de 10.000bytes al servidor y el proceso servidor, después de recibir por completo el fichero, le envía un fichero de 2.000 bytes al cliente). Suponga que los mensajes de aplicación son similares a los utilizados en la práctica 4 (envía_mensaje) donde se envía en primer lugar la longitud del mensaje y a continuación el texto (en nuestro caso el contenido del fichero). El intercambio de mensajes de aplicación sería como el indicado en la figura.

Dibujar un diagrama con el intercambio de segmentos que ocurre entre el cliente y el servidor desde que se inicia la conexión hasta que se cierra la conexión. (procesos de apertura y cierre incluidos con número de secuencia inicial 0 en ambos extremos e indicando el valor de la ventana de congestión en cada ronda).

En su diagrama debe escribir los valores de los campos más significativos de la cabecera de cada segmento y tener en cuenta que:
- MSS vale 2.000 bytes
- El umbral ssth vale 10.000 bytes,
- No se envían datos en el procedimiento de apertura de la conexión,
- La conexión la cierra el cliente
- Durante la fase de transferencia se pierde el segundo y el cuarto segmento con datos enviados por el cliente
- Considere el tiempo de ida y vuelta (RTT) constante. Dibuje la inclinación de las flechas para que le quepa el diagrama completo.

SOL:


Interpretación 1
SN=0, SYN
SN=0, SYN, ACK=1
ACK=1
SN=1, (4B)
CONG_WIN=1MSS
ACK=5
SN=5, (2.000 B)
SN=2.005, (2.000 B)
CONG_WIN=2MSS
X
ACK=5
TIMEOUT
SN=5, (2.000 B)
CONG_WIN=1MSS
ACK=4.005
SN=4.005, (2.000 B)
SN=6.005, (2.000 B)
CONG_WIN=2MSS
X
ACK=4.005
TIMEOUT
SN=4.005, (2.000 B)
CONG_WIN=1MSS
ACK=8.005
SN=8.005, (2.000 B)
ACK=10.005, SN=1,(4B)
ACK=5
SN=5,(2.000 B)
ACK=2.005, FIN, SN=10.005
ACK=10.006, FIN, SN=2.005
ACK=2.006
Interpretación 2
SN=0, SYN
SN=0, SYN, ACK=1
ACK=1
SN=1, (4B)
CONG_WIN=1MSS
ACK=5
SN=5, (2.000 B)
SN=2.005, (2.000 B)
CONG_WIN=2MSS
X
ACK=5
TIMEOUT
SN=5, (2.000 B)
CONG_WIN=1MSS
X
TIMEOUT
SN=5, (2.000 B)
ACK=4.005
SN=4.005, (2.000 B)
SN=6.005, (2.000 B)
CONG_WIN=2MSS
ACK=6.005
ACK=8.005
SN=8.005, (2.000 B)
ACK=10.005, SN=1,(4B)
ACK=5
SN=5,(2.000 B)
ACK=2.005, FIN, SN=10.005
ACK=10.006, FIN, SN=2.005
ACK=2.006