Redes de Computadores
I. Informática
Universidad Complutense de Madrid

TEMA 6.
La capa de transporte: protocolos TCP y UDP

PROFESOR: *Rubén Santiago Montero*

Protocolos de transporte en Internet: TCP y UDP

- **El protocolo TCP (Transmission Control Protocol)**
  - Protocolo de transporte "orientado a conexión"
  - Garantiza un servicio extremo a extremo fiable
    - Detectar/retransmitir segmentos de datos perdidos o erróneos
    - Detectar y descartar segmentos duplicados
    - Ordenar los segmentos en el destino y pasarlos de forma ordenada a la capa de aplicación
  - Utilizado en aplicaciones en las que la seguridad en la entrega es más importante que la rapidez
    - FTP, HTTP, Telnet, SMTP, etc.
- **El protocolo UDP (User Datagram Protocol)**
  - Protocolo de transporte "sin conexión"
    - NO garantiza un servicio extremo a extremo fiable
    - No controla la pérdida de paquetes, los errores o la duplicidad
  - Utilizado en aplicaciones en las que la rapidez en la entrega es más importante que la seguridad
    - DNS, SMNP, RIP, RTP, etc.

Protocolos de transporte en Internet: TCP y UDP

Comparación de TCP y UDP

TCP	UDP
Servicio orientado a conexión - Entrega ordenada de paquetes - Retransmisión de paquetes perdidos o erróneos - Detecta duplicados	**Servicio sin conexión (best effort)** - No garantiza la entrega ordenada - No retransmite paquetes perdidos o erróneos - No detecta paquetes duplicados
Unidad de transferencia - Segmento - Tamaño mín. de cabecera = 20 bytes	**Unidad de transferencia** - Datagrama - Tamaño mín. de cabecera = 8 bytes
Fases de la comunicación - Establecimiento de conexión - Transferencia de datos - Cierre de conexión	**Fases de la comunicación** - Transferencia de datos (bloques individuales)
Mecanismos de control de errores/flujo - Método de tipo ventana deslizante - Numeración de segmentos - Confirmaciones del receptor - Retransmisión de segmentos no confirmados	**Mecanismos de control de errores/flujo** - No existen
Ejemplo de aplicaciones -Telnet, FTP, HTTP, E-mail (SMTP, POP3), etc.	**Ejemplo de aplicaciones** - DNS, RIP, SNMP, DHCP, RTP, etc.

Puertos TCP y UDP. Modelo cliente/servidor

El modelo cliente-servidor

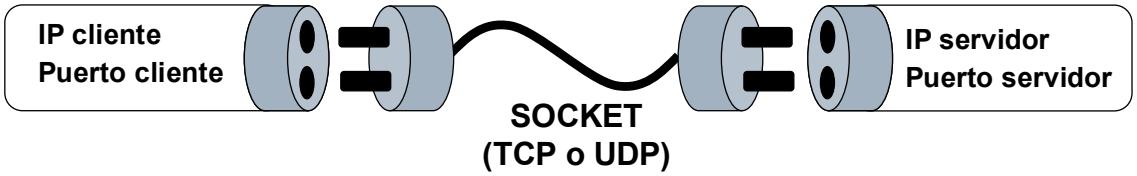
La mayoría de aplicaciones TCP y UDP utilizan e modelo cliente/servidor:

- **El cliente:**
  - Es la aplicación que inicia la conexión o el intercambio de datos con la máquina remota (servidor)
  - La aplicación cliente normalmente la arranca un usuario cuando quiere utilizar un servicio de la red
  - Ejemplos:
    - Un navegador web (TCP)
    - Un cliente de correo electrónico (TCP)
    - Una consulta al servidor DNS (UDP)
- **El servidor:**
  - Es la aplicación que recibe y acepta la solicitud de conexión o intercambio de datos del cliente
  - Esta aplicación normalmente está ejecutándose continuamente en la máquina remota y está a la espera de solicitudes de clientes
  - Ejemplos:
    - Un servidor Web (TCP)
    - Un servidor de correo electrónico (TCP)
    - Un servidor DNS (UDP)

Puertos TCP y UDP. Modelo cliente/servidor

Parámetros en una comunicación cliente/servidor (TCP o UDP)

- El canal de comunicación establecido entre el cliente y el servidor se denomina "**socket**" (enchufe)
  - El socket permite el intercambio de datos bidireccional entre cliente y servidor
  - Los extremos de este canal de comunicación o **socket** vienen fijados por cuatro parámetros
    - **Dir. IP del cliente:**
      - Identifica a la máquina cliente (la que inicia la conexión o intercambio de datos)
    - **Puerto del cliente:**
      - Identifica al proceso cliente dentro de la máquina cliente
    - **Dir. IP del servidor:**
      - Identifica a la máquina servidora (la que acepta la solicitud de conexión o intercambio de datos)
    - **Puerto del servidor:**
      - Identifica al proceso servidor dentro de la máquina servidora



Puertos TCP y UDP. Modelo cliente/servidor

Asignación de números de puerto

El puerto del cliente

- Cuando el usuario arranca el proceso cliente
  - El SO de la máquina cliente le asigna un número de puerto libre
  - Este número de puerto es siempre un número mayor que 1023 (puertos efímeros)

El puerto del servidor

- Cuando un proceso cliente solicita una comunicación con un servidor debe conocer de antemano su número de puerto
- Para ello existen dos soluciones:
  - Que el servidor utilice un "puerto bien conocido" (well-known ports)
    - FTP: 21
    - SSH: 22
    - TELNET: 23
    - SMTP: 25
    - DNS: 53
    - HTTP: 80
    - POP3: 110
    - NNTP: 119
  - Usar el servicio de llamada a procedimiento remoto
    - **RPC** (Remote Procedure Call)

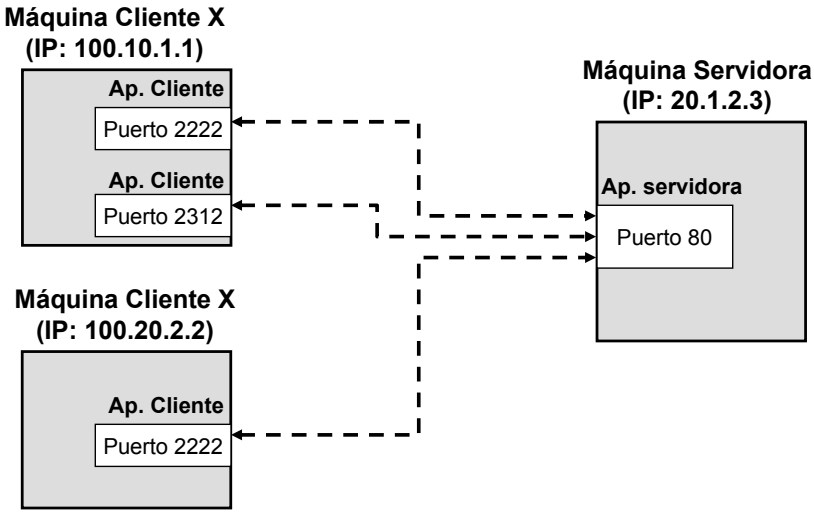
Importante: *en una misma máquina no pueden existir dos aplicaciones distintas en ejecución (ya sea cliente o servidor) con el mismo número de puerto*

Puertos TCP y UDP. Modelo cliente/servidor

Servidores concurrentes

La mayoría de aplicaciones servidoras son de tipo concurrente

- Pueden atender a varios clientes de forma simultánea
  - No es posible confundir los datos procedentes de dos clientes distintos, ya que no es posible que se repitan los cuatro parámetros de la comunicación (IP cliente – Puerto cliente – IP servidor – Puerto servidor)



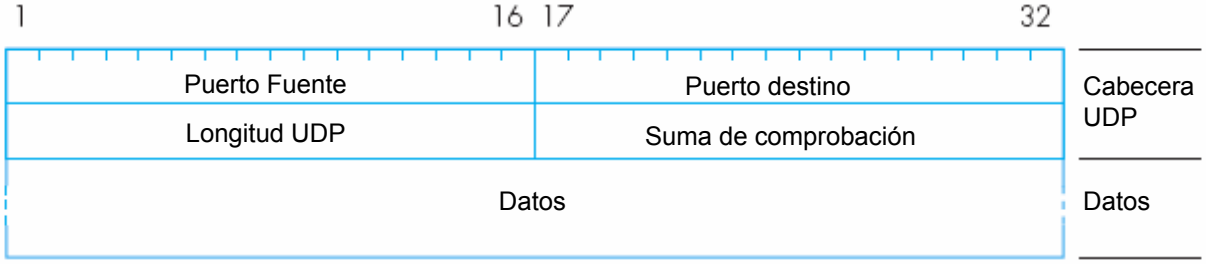
Protocolo UDP

El protocolo UDP

Conceptos generales

- El protocolo UDP es un protocolo sin conexión y no fiable
  - UDP divide los mensajes en datagramas, pero éstos NO se numeran
  - El receptor NO envía confirmación de la recepción de los mismos
- UDP no garantiza la recuperación de datagramas perdidos o erróneos, ni tampoco corrige el desordenamiento o la duplicación de los mismos

Formato del datagrama UDP



Protocolo UDP

Campos de la cabecera UDP

Puerto origen y destino:

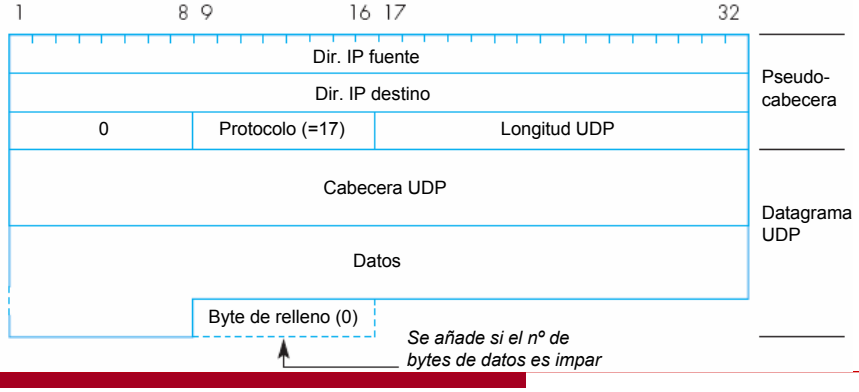
- Identifican a las aplicaciones que se comunican

Longitud UDP

- Longitud total en bytes del datagrama UDP (incluyendo datos y cabecera)

Suma de comprobación

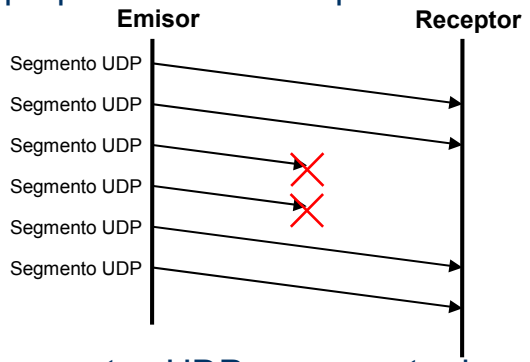
- Se calcula como el complemento a 1 de la suma en complemento a 1 de todas las palabras de 16 bits que forman parte de:
  - El datagrama UDP (datos + cabecera UDP)
  - Una **pseudo-cabecera** formada por información de la cabecera IP (dir. IP fuente, dir. IP destino, campo Protocolo y longitud UDP)



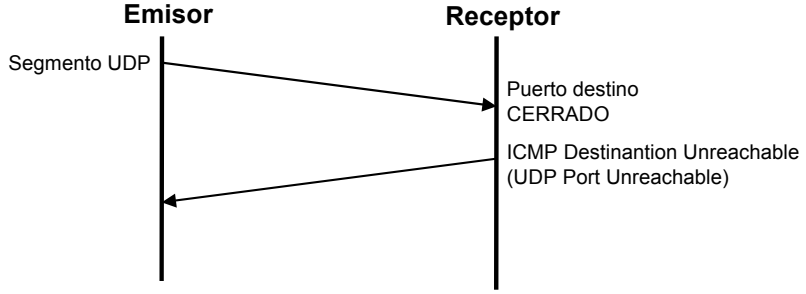
Protocolo UDP

Mecanismo de transmisión de datos en UDP

Envío de paquetes UDP a un puerto de servicio **ABIERTO**



Envío de paquetes UDP a un puerto de servicio **CERRADO**



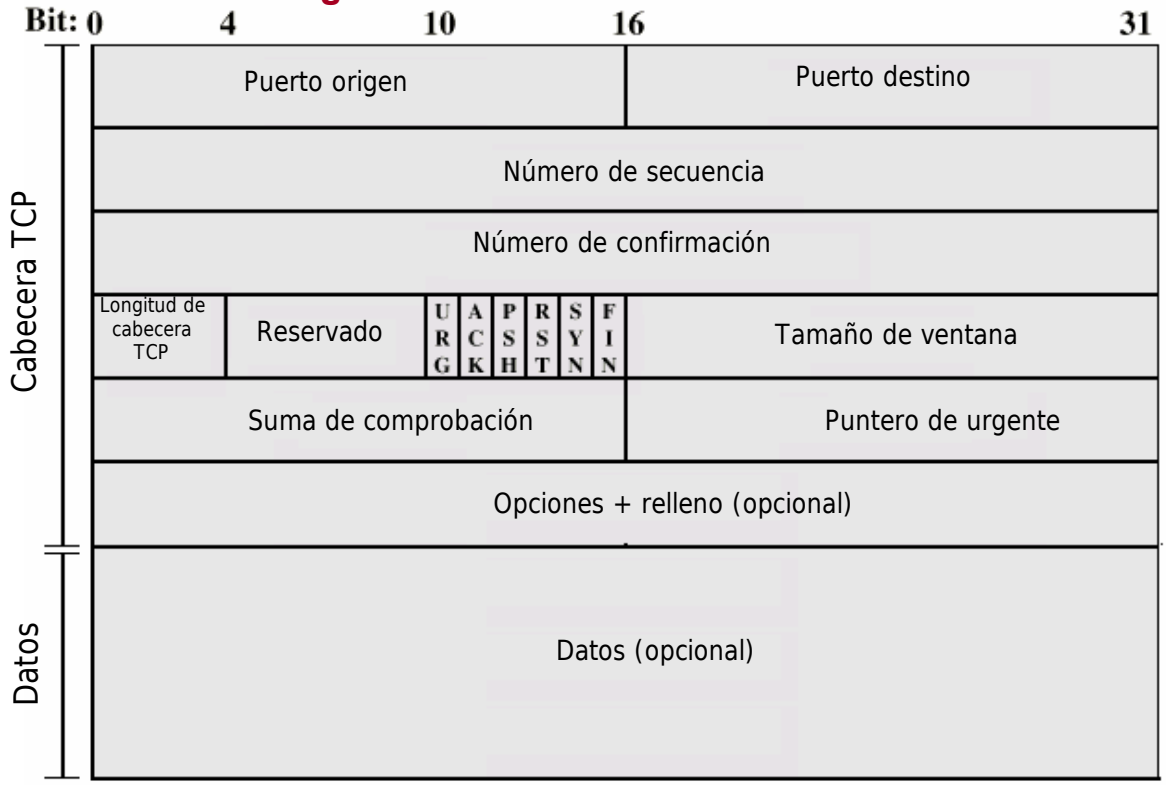
Protocolo TCP

El protocolo TCP: Conceptos generales

- Unidad de transferencia
  - Segmento TCP
- Fases en una transmisión mediante TCP
  - Establecimiento de conexión
  - Transferencia de datos
  - Cierre de conexión
- Mecanismos de control de errores en TCP
  - TCP utiliza un mecanismo de tipo ventana deslizante para el control de errores
    - Numeración de segmentos
      - Cada segmento lleva un número de secuencia de 32 bits
      - Indica la posición que ocupa el primer byte del segmento dentro del mensaje original
    - Confirmaciones superpuestas del receptor
      - Cuando el receptor recibe un segmento de datos correcto y sin errores, **envía una confirmación** al emisor
    - Retransmisión de segmentos
      - Si transcurrido un tiempo desde que se envió el segmento, el emisor no recibe confirmación, entonces **retransmite de nuevo el segmento**

Protocolo TCP

Formato de la segmento TCP



Protocolo TCP

- **Campos de la cabecera del segmento TCP**
  - **Puerto origen y destino:**
    - Identifican los extremos de la conexión
  - **Nº de secuencia:**
    - Indica la posición del primer byte del segmento con respecto al mensaje original
  - **Nº de confirmación:**
    - Para enviar confirmaciones superpuestas en sentido contrario. Indica el nº de secuencia del siguiente byte que se espera recibir
  - **Longitud de la cabecera:**
    - Medida en palabras de 32 bits
  - **Tamaño de ventana:**
    - Permite anunciar el tamaño de la **ventana de recepción** durante la conexión TCP
    - El valor del campo ventana indica la cantidad de bytes (relativos al nº de byte indicado en el campo nº de confirmación) que el receptor es capaz de aceptar

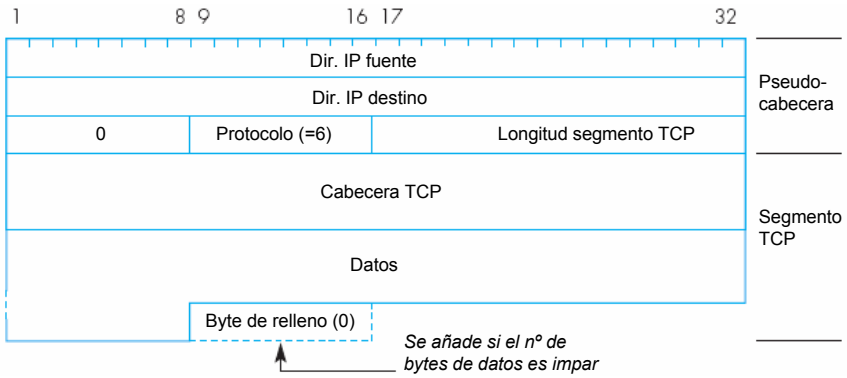
Protocolo TCP

- **Campos de la cabecera del segmento TCP**
  - **Flag URG y puntero urgente:**
    - Si URG=1, el segmento transporta datos urgentes a partir del nº de byte especificado en el campo **puntero urgente**
  - **Flag ACK:**
    - Si ACK=1, el segmento un número de confirmación válido
    - Todos los segmentos de una conexión TCP, excepto el primero, llevan ACK=1 y transportan un número de confirmación válido
  - **Flag PUSH:**
    - Si PUSH=1, indica que los datos deben ser pasados inmediatamente a la aplicación.
    - Si PUSH=0, los datos se pueden almacenar en un buffer de recepción y éstos se pasan a la aplicación cuando el buffer se llena.
  - **Flag RST:**
    - Flag utilizado para abortar una conexión
  - **Flag SYN:**
    - Flag utilizado en el establecimiento de la conexión.
    - Significa que los extremos deben sincronizar los números de secuencia iniciales de la transmisión
  - **Flag FIN:**
    - Flag utilizado en la finalización de la conexión

Campos de la cabecera del segmento TCP

Suma de comprobación

- Se calcula como el complemento a 1 de la suma en complemento a 1 de todas las palabras de 16 bits que forman parte de:
  - El segmento TCP (datos + cabecera TCP)
  - Una pseudo-cabecera formada por información de la cabecera IP (dir. IP fuente, dir. IP destino, campo Protocolo y longitud del segmento TCP)



Campos de la cabecera del segmento TCP

Opciones:

- Permite negociar parámetros adicionales de la conexión
- El parámetro más común es el **tamaño máximo del segmento (MSS)**
  - El valor por defecto del MSS es 536 Bytes
  - Si se negocia el valor del MSS, este debería ajustarse al valor de la MTU mínima de las redes que debe atravesar el segmento (MTU del camino)
  - Si se impone un MSS muy grande, puede empeorar el rendimiento
- Ejemplo de MSS demasiado grande
  - Supongamos que la MTU del camino es 576 bytes y elegimos MSS=16.384 bytes
    - Cada segmento tiene que dividirse en 30 fragmentos
    - Si se pierde uno solo de los 30 fragmentos, es necesario retransmitir el segmento completo

$$\frac{16.384}{576-20} \approx 30 \text{ fragmentos}$$

Cabecera IP repetida en cada segmento

  - Supongamos las siguientes tasas de errores en la red
    - Tasa de errores es 0,1% (se pierde un paquete de cada 1000)

      Será necesario retransmitir 1 de cada 33,3 segmentos enviados (que dan lugar a 1000 paquetes)
      → retransmisión del **3% de segmentos**
    - Tasa de errores es 1% (se pierde un paquete de cada 100)

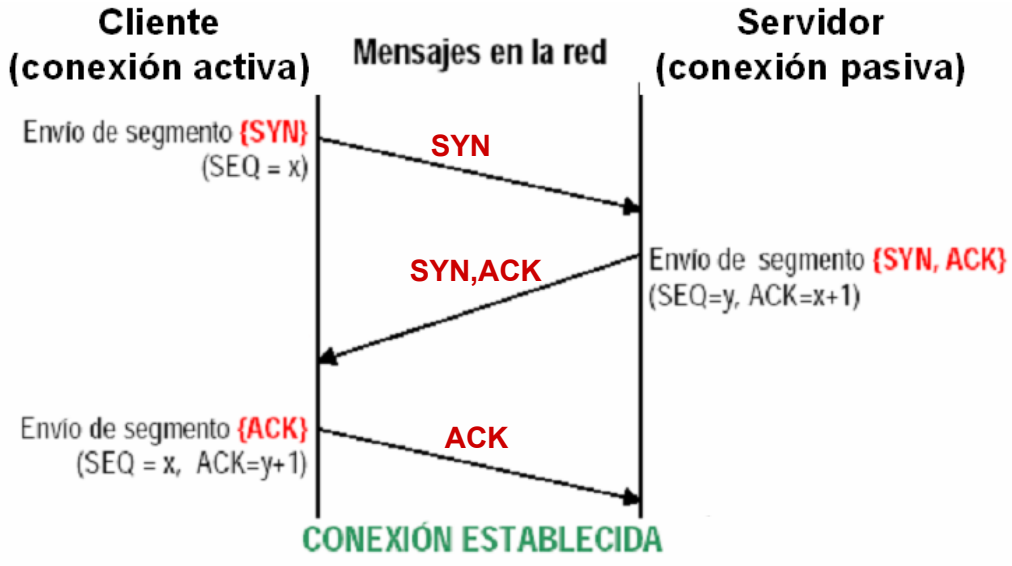
      Será necesario retransmitir 1 de cada 3,33 segmentos enviados (que dan lugar a 100 paquetes)
      → retransmisión del **30% de segmentos**
    - Tasa de errores es 5% (se pierde un paquete de cada 20)

      Será necesario retransmitir todos los segmentos (cada segmento da lugar a 30 paquetes)
      → retransmisión del **100% de segmentos**

Protocolo TCP

Mecanismo de establecimiento de conexión TCP

Protocolo de 3 vías (three-way handshake)

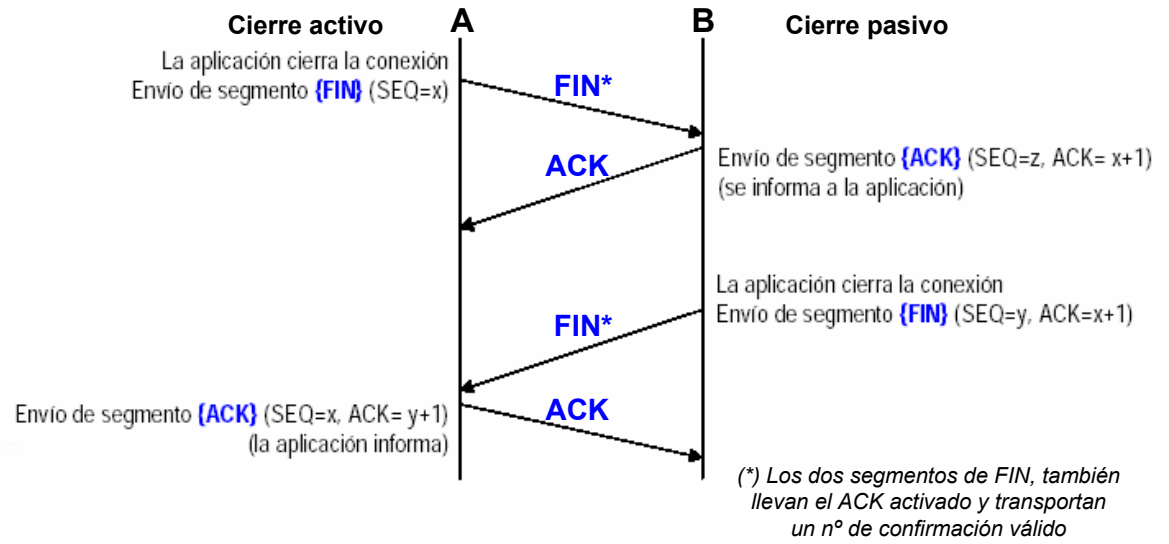


Protocolo TCP

Mecanismo de desconexión en TCP

Opción 1: protocolo de 4 vías

- Uno de los extremos (A) ha terminado de transmitir y cierra primero la conexión
  - Este extremo ya no puede enviar más datos, pero todavía puede recibir datos del extremo contrario y devolver confirmaciones
- El otro extremo (B) todavía puede tener datos pendientes de envío

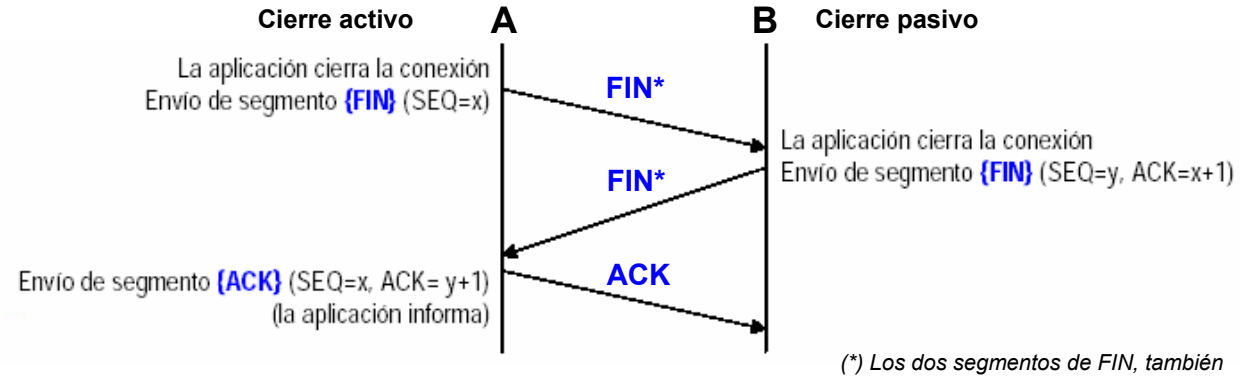


(*) Los dos segmentos de FIN, también llevan el ACK activado y transportan un nº de confirmación válido

Mecanismo de desconexión en TCP

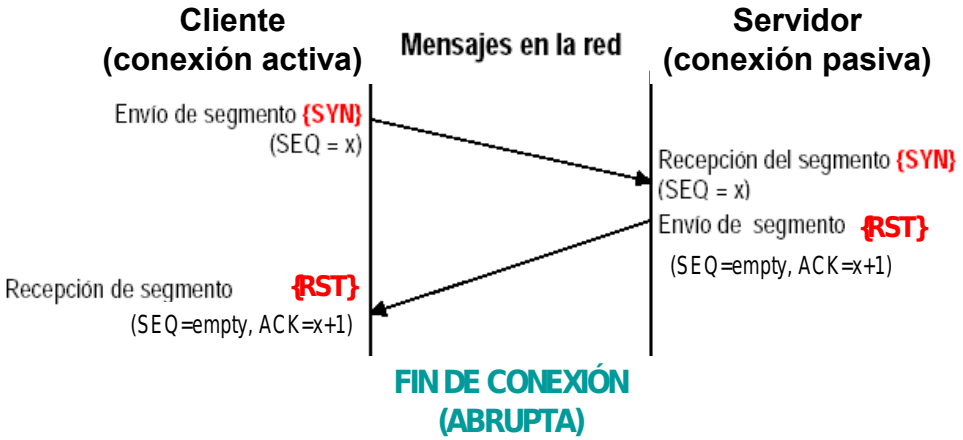
Opción 2: protocolo de 3 vías

- Ambos extremos (A y B) han terminado de transmitir y están de acuerdo en cerrar la conexión



(*) Los dos segmentos de FIN, también llevan el ACK activado y transportan un nº de confirmación válido

Mecanismo de desconexión abrupta (aborto) en TCP

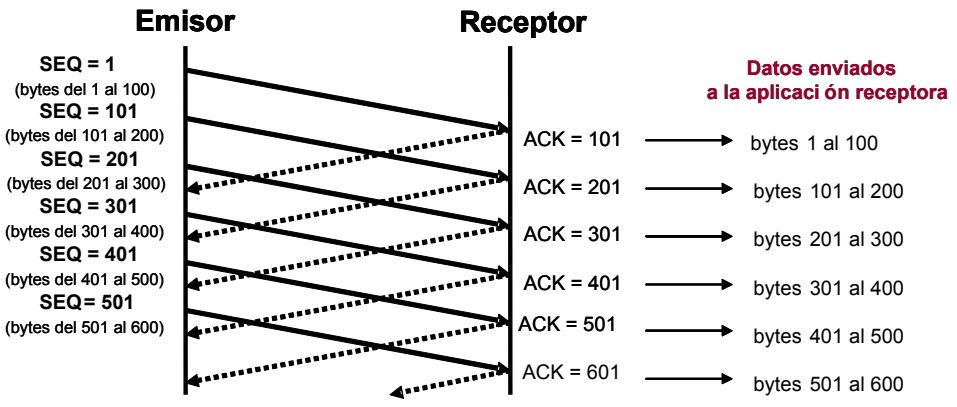


Mecanismo de transmisión de datos en TCP (1)

- Se utiliza un mecanismo similar al método de la ventana deslizante usado en la capa de enlace, aunque con algunas variantes
  - El emisor envía todas los segmentos numerados
    - El campo **nº de secuencia** indica la posición del primer byte del segmento con respecto al inicio de la conexión
  - El receptor envía una confirmación por cada segmento recibido
    - El **campo nº de confirmación** siempre contiene el identificador del siguiente byte que se espera recibir
  - En caso de que falte algún segmento
    - Por cada segmento recibido fuera de secuencia, se devuelve una confirmación que contiene el identificador del siguiente byte que se espera recibir en secuencia
    - Cuando llega el segmento que completa la secuencia, el receptor envía una confirmación de la secuencia completa

Mecanismo de transmisión de datos en TCP (2)

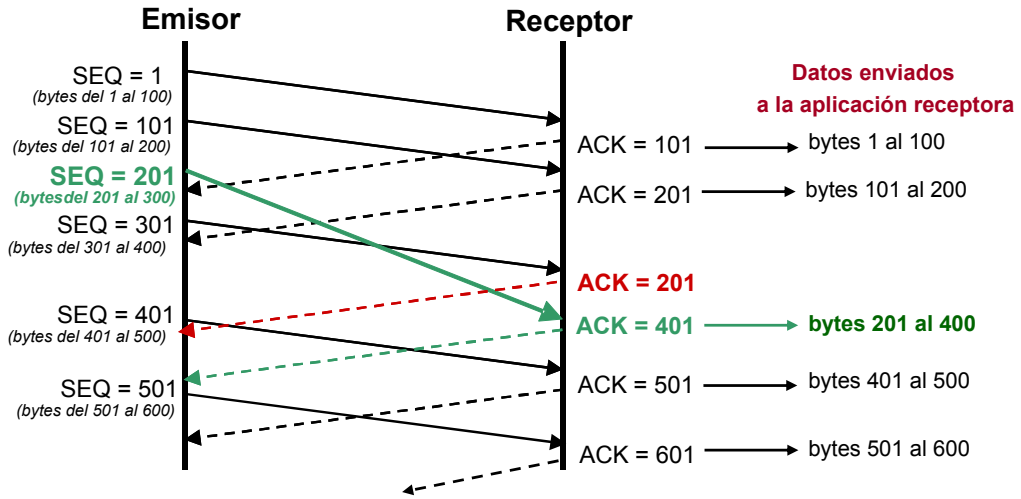
- Simplificaciones
  - Se supone que sólo se transmiten datos en un sentido (Emisor → Receptor)
    - En el caso general la transmisión es bidireccional
  - Se supone que todos los segmentos transportan 100 bytes de datos
    - En el caso general los segmentos pueden transportar diferente número de bytes, hasta un máximo fijado por el valor del MSS (Maximun Segment Size)
  - Se supone que el número de secuencia inicial del emisor es SEQ=1
    - En el caso general el valor de los números de secuencia iniciales se acuerda durante el proceso de establecimiento de conexión TCP
- Ejemplo 1: Transmisión sin errores



Mecanismo de transmisión de datos en TCP (3)

Ejemplo 2: Retardo en la transmisión de un segmento

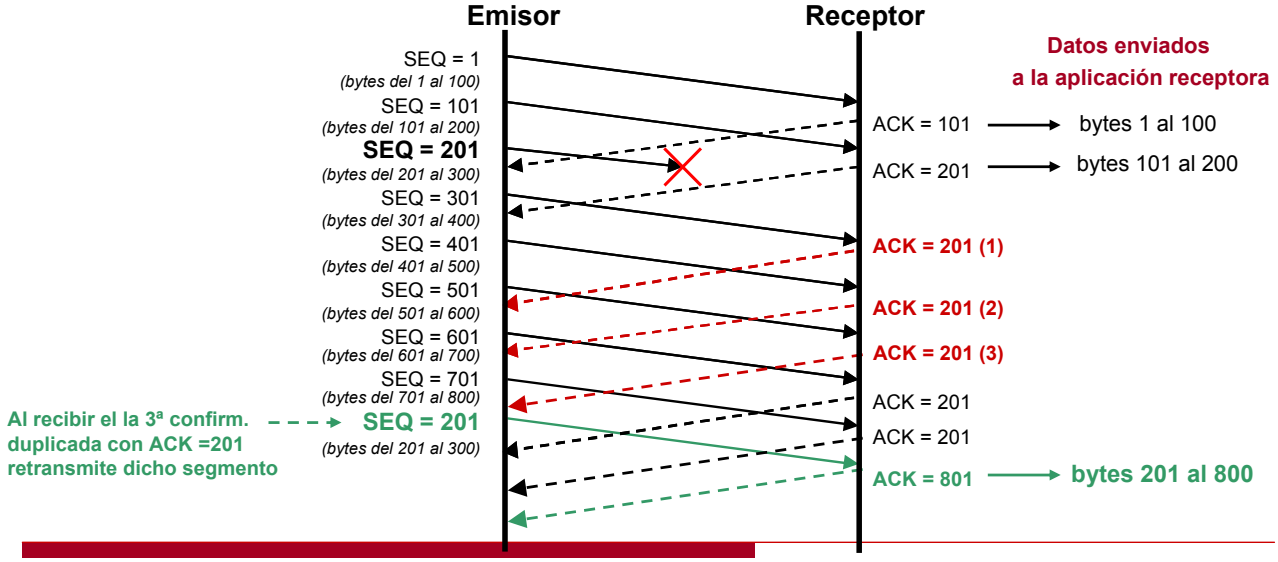
- Cuando el receptor recibe segmentos fuera de secuencia
  - Devuelve un ACK indicando cual el siguiente byte en secuencia que espera recibir
- Cuando el emisor recibe un ACK fuera de secuencia
  - No inicia la retransmisión del segmento inmediatamente después de recibir el ACK
  - Un segmento sólo se retransmite cuando se reciben tres ACKs duplicados para ese segmento
- Cuando el receptor recibe el segmento retardado que le faltaba
  - Envía una confirmación de toda la secuencia completa



Mecanismo de transmisión de datos en TCP (4)

Ejemplo 3: Pérdida de un segmento

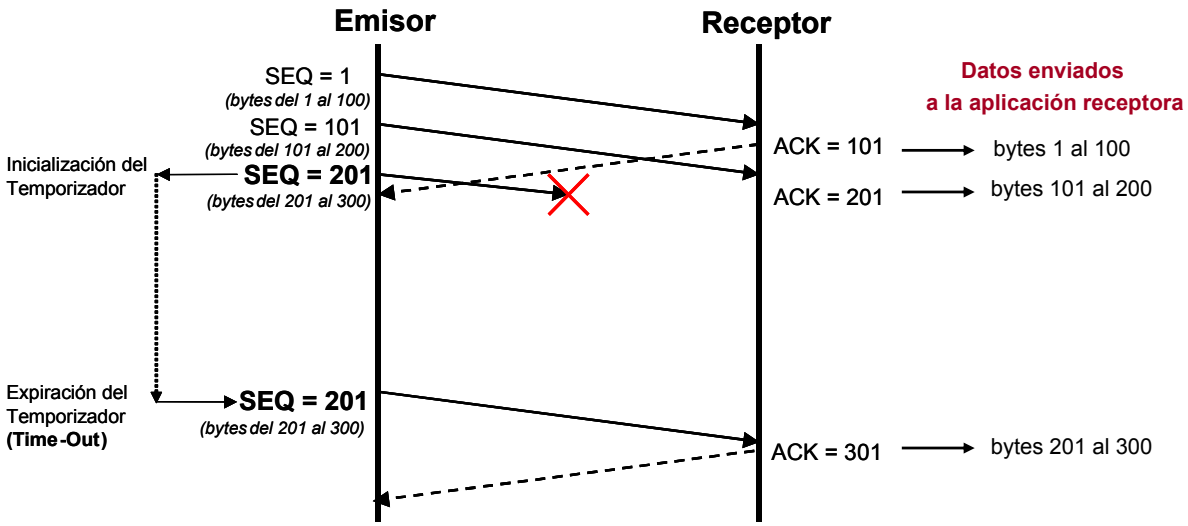
- Cuando el receptor recibe los segmentos fuera de secuencia
  - El receptor devuelve un ACK indicando cual el siguiente byte en secuencia que espera recibir
- Cuando el emisor recibe tres ACKs duplicados con el mismo identificador
  - El emisor retransmite el segmento pendiente de confirmación
- Cuando el receptor recibe el segmento que le faltaba
  - Envía una confirmación de toda la secuencia completa



Mecanismo de transmisión de datos en TCP (5)

Ejemplo 4: Temporizador de retransmisión

- Para contemplar la posibilidad de que el emisor no reciba confirmaciones
  - Por cada nuevo segmento transmitido se inicia *un temporizador de retransmisión*
  - El segmento se retransmite el emisor no recibe una confirmación antes de que expire el temporizador.



Temporizadores de retransmisión (1)

TCP utiliza un mecanismo adaptativo

- La elección del tiempo de vencimiento del temporizador de retransmisión (**timeout**) está basada en los retardos observados en la red
- Los retardos en la red pueden variar dinámicamente, por tanto los timeouts debe adaptarse a esta situación
- Las principales técnicas utilizadas para fijar los temporizadores de retransmisión son las siguientes:
  - Método de la media ponderada (algoritmo de Jacobson)
  - Método de la varianza (algoritmo de Jacobson/Karels)
  - Algoritmo de Karn

Temporizadores de retransmisión (2)

Método de la media ponderada (algoritmo de Jacobson)

- Por cada segmento enviado, se calcula el tiempo de ida y vuelta del segmento y la confirmación, denominado RTS (*Round Trip Sample*)
- A partir de este tiempo se estima un valor promedio del tiempo de ida y vuelta, RTT (*Round Trip Time*), usando un procedimiento de media ponderada:

$$RTT = \alpha \cdot RTT_{anterior} + (1-\alpha) \cdot RTS$$

  - Si α es próximo a la unidad:
    - Tiene más peso el valor histórico que los cambios inmediatos en el RTS.
    - RTT responderá lentamente a las variaciones del valor de RTS
  - Si α es próximo a la cero:
    - Tienen más peso los cambios inmediatos que el valor histórico
    - RTT responderá rápidamente a las variaciones del valor de RTS
  - Cada vez que se envía un segmento y se obtiene una nueva medida de RTS, se realiza una nueva estimación del RTT
- A partir del valor estimado de RTT se determina el tiempo de vencimiento del temporizador de la siguiente forma:

$$Timeout = \beta \cdot RTT$$

- Valores típicos recomendados para α y β :

  α = 0,875 (Las nuevas muestras afectan en un 12,5% al valor del RTT)

  β= 2 (El timeout se fija al doble del tiempo estimado de ida y vuelta)

Temporizadores de retransmisión (3)

Método de la varianza (algoritmo de Jacobson/Karels)

- La utilización de un valor de $\beta = 2$ fijo no se ajusta bien al caso de grandes variaciones del RTT
  - Jacobson propuso que cada valor nuevo del timeout debe basarse, no sólo en el valor medio del RTT, sino también en su varianza.
  - Se utiliza la desviación media, D, del RTT como una estimación de la varianza

$$RTT = \alpha \cdot RTT_{anterior} + (1-\alpha) \cdot RTS$$

$$D = \alpha' \cdot D_{anterior} + (1-\alpha') \cdot |RTT_{anterior} - RTS|$$

  - La nueva estimación del valor del timeout, viene dada por:

$$Timeout = RTT + 4 \cdot D$$

  - Valores típicos recomendados para α y α' :

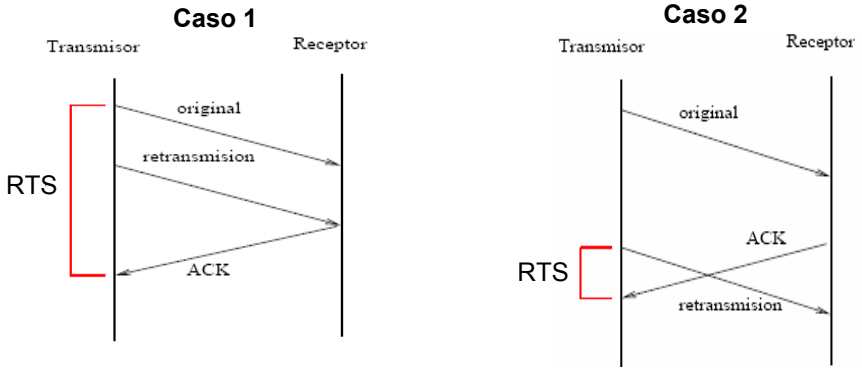
    α = 0,875

    α' = 0,75

Temporizadores de retransmisión (4)

Algoritmo de Karn

- El problema de la ambigüedad de la confirmación
  - Cuando un segmento se retransmite y el emisor recibe un ACK, éste no tiene forma de saber si este ACK se debe asociar con la primera o con la segunda transmisión, para calcular el RTS del segmento
    - Si se toma el primer segmento enviado, se puede estar utilizado un RTS demasiado grande (caso 1)
    - Si se toma el último segmento enviado, se puede estar utilizado un RTS demasiado pequeño (caso 2)



Temporizadores de retransmisión (5)

Algoritmo de Karn (cont.)

- Solución al problema
  - El algoritmo de Karn establece que para el cálculo de RTT, sólo deben tenerse en cuenta los tiempos de los segmentos enviados una sola vez
    - Con esto se evita la ambigüedad.
  - En caso de retransmisión de un segmento
    - No se realiza una nueva estimación del RTT
    - El valor del nuevo timeout se fija como el doble del último timeout, en vez de basarse en RTT estimado

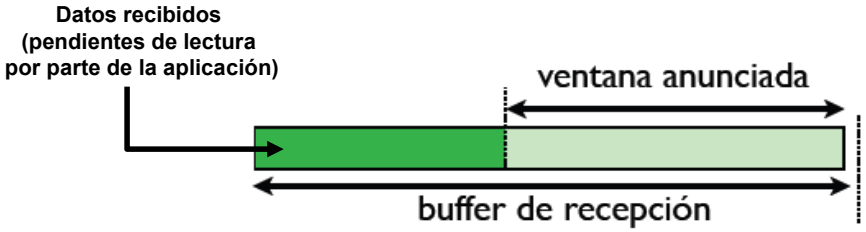
$$Timeout = 2 \cdot Timeout_{anterior}$$

    - Para que no se produzcan temporizaciones excesivamente largas, la mayoría de las implementaciones fijan un valor máximo para el timeout.

El control de flujo en TCP (1)

La ventana de recepción

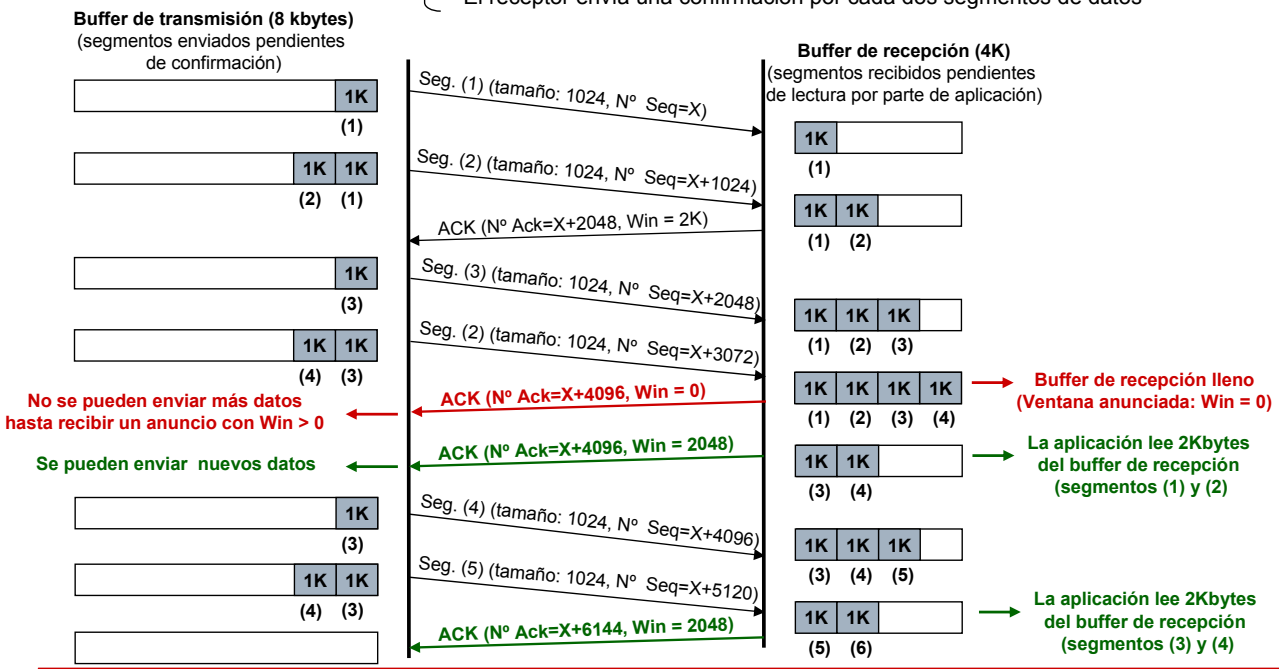
- El control de flujo en TCP se realiza mediante la ventana de recepción
  - El tamaño de esta ventana indica el número de bytes que puede aceptar el receptor en un instante dado
    - Este tamaño viene determinado por el espacio libre disponible en el buffer de recepción utilizado para esa conexión TCP
    - El buffer de recepción, almacena aquellos datos recibidos a través de la conexión TCP, hasta que estos son leídos por parte de la aplicación receptora
  - El tamaño de la ventana de recepción puede variar a lo largo de una conexión
    - El tamaño de la ventana de recepción se anuncia en cada segmento de confirmación
    - Si el receptor anuncia una ventana de tamaño 0, el emisor no puede enviar más datos hasta que se anuncie un nuevo tamaño de ventana mayor



El control de flujo en TCP (2)

Ejemplo

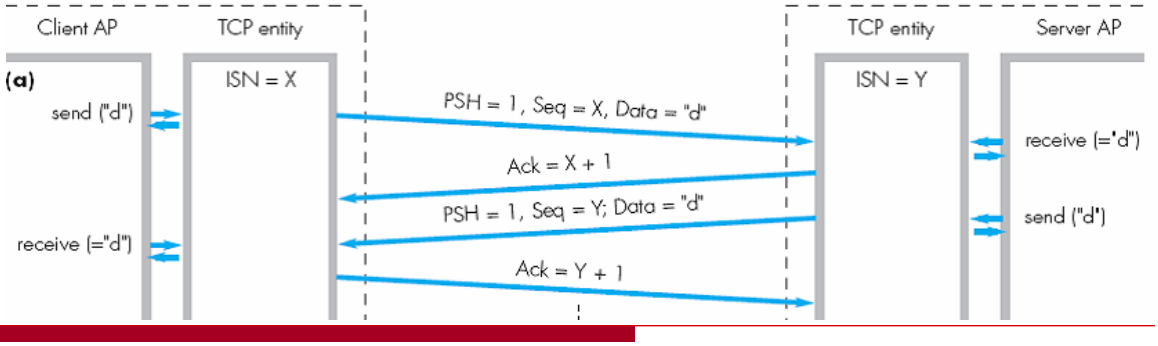
- Suposiciones
  - El tamaño del buffer de recepción es de 4Kbytes
  - El tamaño del buffer de transmisión es de 8Kbytes
  - Todos los segmentos enviados son de 1 Kbyte
  - El receptor envía una confirmación por cada dos segmentos de datos



El control de flujo en TCP (3)

El problema de los segmentos de pequeño tamaño

- **Problema**
  - Se plantea cuando el emisor envía segmentos con poca cantidad de datos
  - Por ejemplo, cuando el emisor envía 1 byte por segmento
- Esta situación se produce en algunas aplicaciones interactivas, p. ej. Telnet:
  - La aplicación Telnet, funciona de la siguiente forma:
    - Cada carácter que escribe el cliente debe ser enviado al servidor
    - El servidor lee el carácter almacenado en el buffer de recepción y devuelve un "eco" del mismo carácter al cliente
    - Ambos segmentos deben ser confirmados
  - Por tanto, por cada carácter transmitido se intercambian 4 segmentos
    - Cada segmento incluye 40 bytes de cabeceras (cabecera IP + cabecera TCP)
    - En total se intercambian más de 160 bytes para transmitir un único carácter



El control de flujo en TCP (4)

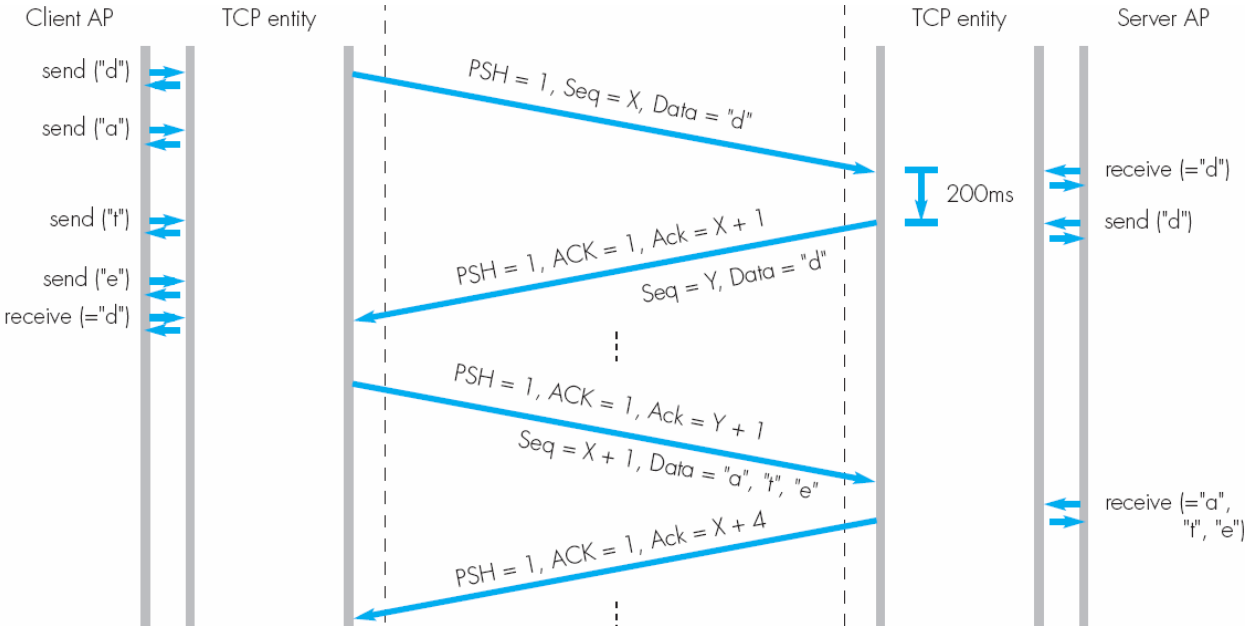
El problema de los segmentos de pequeño tamaño (cont.)

- **Solución: Método de Nagle**
  - Las confirmaciones no se envían inmediatamente
    - Se retrasan un intervalo de tiempo (máximo 200 ms.) para aprovechar el tráfico en sentido contrario y enviar confirmaciones superpuestas
  - Cada uno de los extremos de la conexión TCP sólo puede enviar datos en alguno de los dos casos siguientes:
    a) Si se ha recibido la confirmación del último segmento enviado
    (en este caso, mientras se espera la confirmación, el usuario tiene tiempo de escribir por teclado varios caracteres que se enviarán en un único segmento)
    b) Si el número de bytes almacenado en la ventana de transmisión ≥ MSS
  - El método de Nagle convierte el control de flujo de TCP en un mecanismo similar al método de parada y espera
    - Este método se puede activar o desactivar en función del tipo de aplicación
    - No es recomendable su activación en cierto tipo de aplicaciones interactivas (por ejemplo, cuando se arranca un sistema de ventanas remoto de tipo "X-windows", el método de Nagle puede dar lugar a un comportamiento errático de los movimientos de ratón)

El control de flujo en TCP (5)

El problema de los segmentos de pequeño tamaño (cont.)

- **Método de Nagle: ejemplo**



El control de flujo en TCP (6)

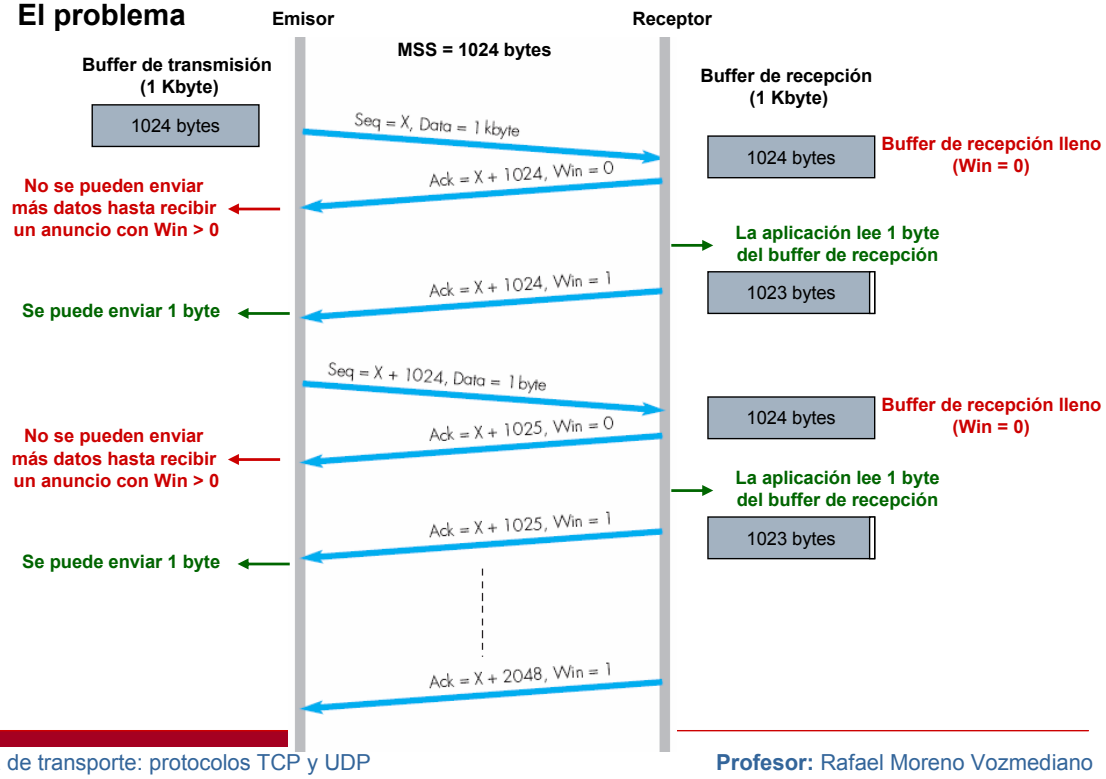
El síndrome de la ventana trivial (*silly window syndrome*)

- **El problema**
  - Se produce cuando la aplicación receptora lee los datos del buffer de recepción en bloques muy pequeños (por ejemplo, de byte en byte)
    - Estas situaciones pueden provocar que se anuncien ventanas muy pequeñas
    - Como consecuencia, el emisor envía segmentos de pequeño tamaño, lo que provoca una reducción del rendimiento, debido a las cabeceras IP y TCP
- **La solución: método de Clark**
  - El extremo receptor: no puede enviar anuncios de ventanas inferiores a un determinado tamaño mínimo. Este tamaño se fija como el menor de los dos siguientes valores:
    - MSS
    - ½ Tamaño_buffer_recepción
  - El extremo emisor: no puede enviar bloques de datos más pequeños que un cierto tamaño mínimo. Este tamaño se fija como el menor de los dos siguientes valores:
    - MSS
    - ½ Tamaño_buffer_transmisión
- NOTA: El método de Clark y el método de Nagle son soluciones complementarias a dos problemas similares y, en muchos casos, ambos pueden actuar de forma conjunta

El control de flujo en TCP (7)

El síndrome de la ventana trivial (cont.)

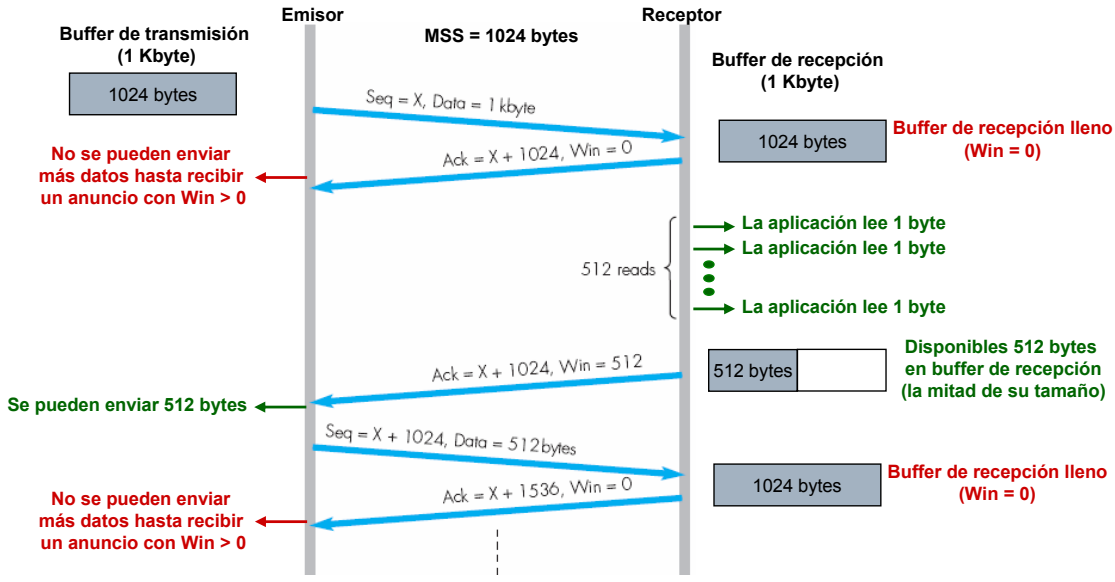
- **El problema**



El control de flujo en TCP (8)

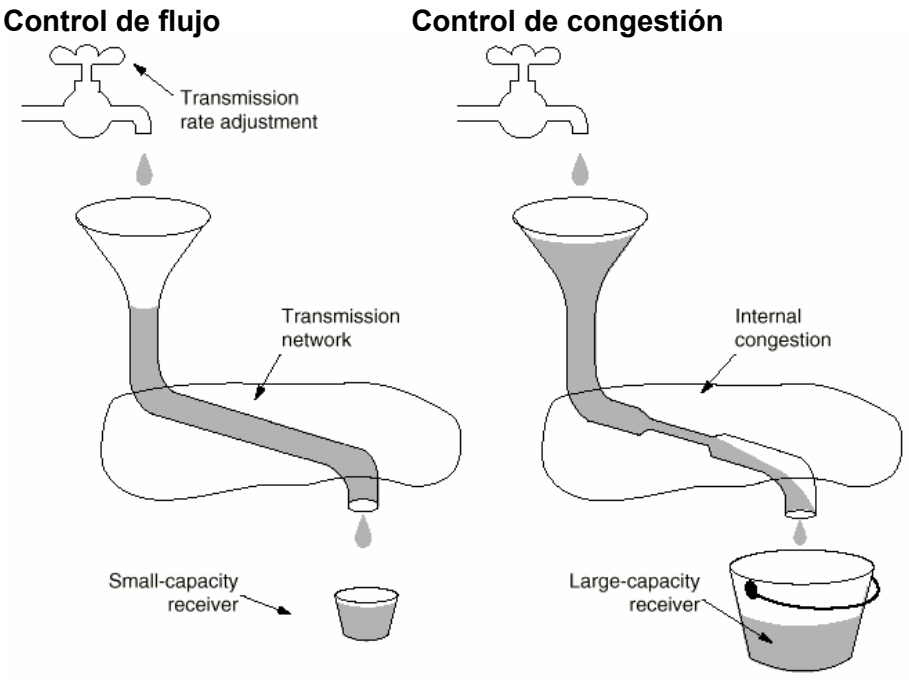
El síndrome de la ventana trivial (cont.)

- **La solución: el método de Clark**



El control de congestión en TCP (1)

Control de flujo vs. Control de congestión



El control de congestión en TCP (2)

El problema de la congestión

- Cuando se pierde paquetes en Internet, la mayoría de las veces se debe a un problema de congestión en algún punto de la red, normalmente un router:
  - El router no puede procesar y reexpedir paquetes al ritmo al que los recibe
  - Cuando el router se satura, empieza a descartar paquetes (incluidas las confirmaciones)
- TCP incorpora un mecanismo de control de congestión:
  - Por cada conexión, se utiliza el ritmo de llegada de confirmaciones para regular el ritmo de envío de segmentos de datos
  - Esto se implementa mediante la **ventana de congestión (CW)**
    - La ventana de congestión es complementaria a la **ventana de recepción (RW)** usada para el control de flujo
    - En una situación de no congestión (sin pérdida o retraso de segmentos) la ventana de congestión alcanza el mismo tamaño que la ventana de recepción (CW=RW)
    - Cuando se produce una situación de congestión el tamaño de CW se va reduciendo progresivamente
    - Cuando la situación de congestión desaparece, el tamaño de CW se va aumentando progresivamente
    - El número máximo de bytes que puede enviar el emisor (AW, Allowed Window) es el mínimo de ambos tamaños de ventana:

$$AW = \min \{ RA, CW \}$$

El control de congestión en TCP (3)

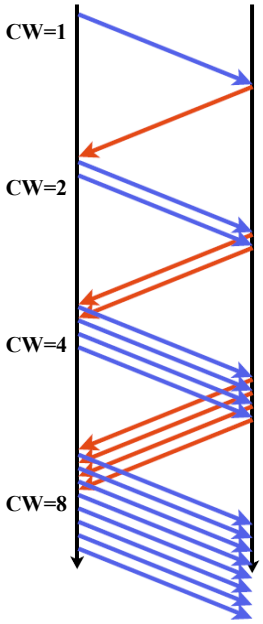
Funcionamiento sin congestión

- La red está en una situación de no congestión cuando no se produce pérdida o retraso de segmentos
- La transmisión comienza con un tamaño de ventana de congestión CW = 1
  - El emisor envía un único segmento de tamaño máximo igual a MSS
- A continuación, la CW va aumentando, pasando por tres fases distintas:
  - **Fase de arranque lento (*slow start*)**
    - La CW se incrementa en uno por cada segmento enviado y confirmado
    - Esto provoca un crecimiento exponencial (CW = 1, 2, 4, 8, 16, 32, …)
    - Esta fase termina cuando el tamaño de CW alcanza un cierto umbral, denominado umbral de arranque lento (STT, *Slow Start Threshold*)
    - Inicialmente, el valor del STT suele ser de 64 Kbytes
  - **Fase de evitación de congestión (*congestion avoidance*)**
    - A partir del STT, la CW se incrementa en 1 cada vez que se envía y se confirma una ventana completa (es decir, CW segmentos)
    - Esto provoca un crecimiento lineal
    - Esta fase termina cuando la CW alcanza el tamaño de la ventana de recepción (RW)
  - **Fase constante**
    - En esta fase, la CW se mantiene a un valor constante (CW = RW)

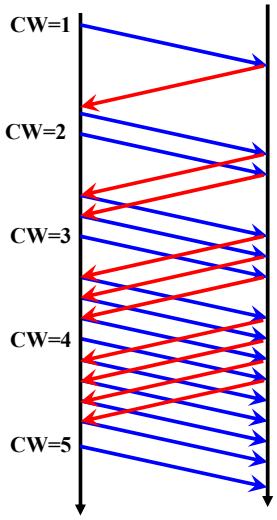
El control de congestión en TCP (4)

Funcionamiento sin congestión

Método de arranque lento (exponencial)
(La ventana se incrementa en 1 por cada segmento enviado y confirmado)



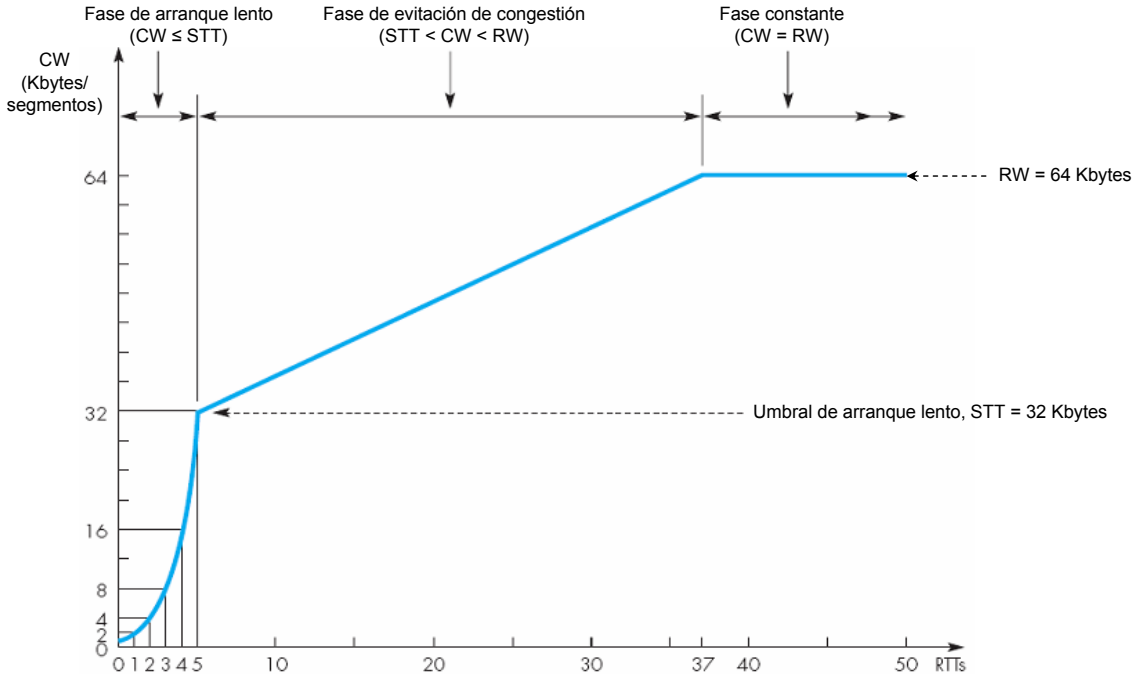
Método de evitación de colisiones (lineal)
(La ventana se incrementa en 1 por cada CW segmentos enviados y confirmados)



El control de congestión en TCP (5)

Funcionamiento sin congestión

- Ejemplo **MSS** = 1 Kbyte **STT$_{inicial}$** = 32 Kbytes **RW** = 64 Kbytes



El control de congestión en TCP (6)

Funcionamiento en caso de congestión

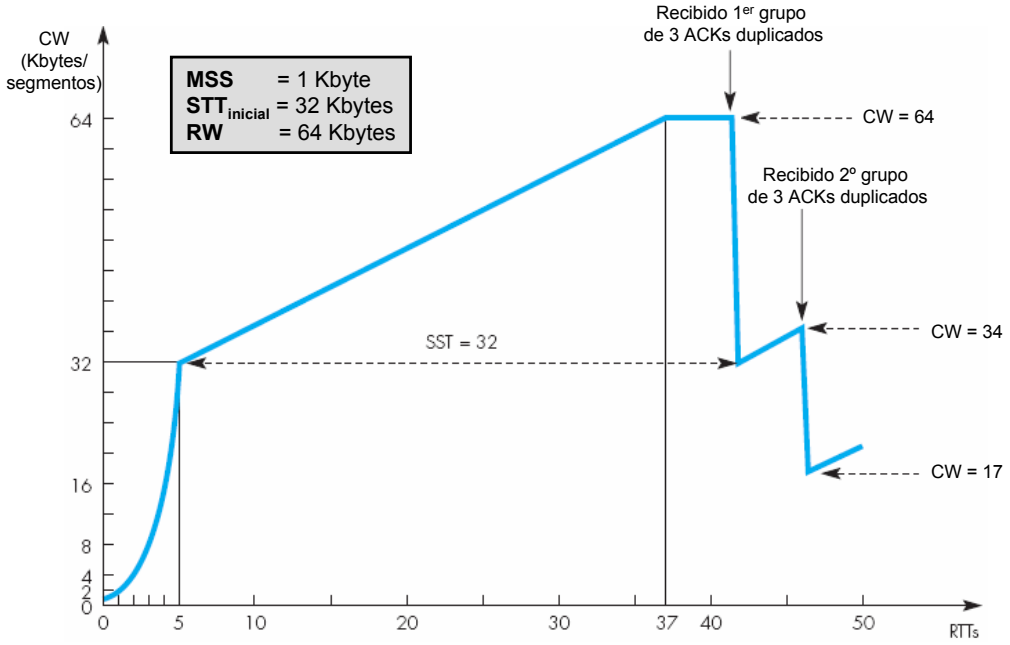
- Cuando se produce la pérdida o retraso de un segmento, se puede detectar por dos métodos:

1) Por recepción de 3 ACKs duplicados
   - En este caso se supone un **nivel de congestión leve**, puesto que sigue habiendo tráfico en la red (llegan las confirmaciones)
   - En este caso se pone en marcha el **método de recuperación rápida** (*fast recovery*), que implica las siguientes acciones
     - Dividir el valor de CW a la mitad
     - Ejecutar el método de evitación de colisiones a partir de ese valor de CW
2) Por expiración del temporizador de retransmisión (timeout)
   - En este caso se supone un **nivel de congestión elevado**, puesto que se interpreta que el tráfico en la red está interrumpido (no llegan confirmaciones)
   - En este caso se realizan las siguientes acciones
     - Inicializar el tamaño de la ventana de congestión a CW = 1
     - Reducir el umbral de arranque lento (STT), fijándolo a la mitad del valor que tenía la CW antes de producirse el timeout
     - Ejecutar el método de arranque lento a partir de CW = 1

El control de congestión en TCP (7)

Funcionamiento en caso de congestión

- **Ejemplo 1.** Detección de congestión por recepción de ACKs duplicados (método de recuperación rápida)

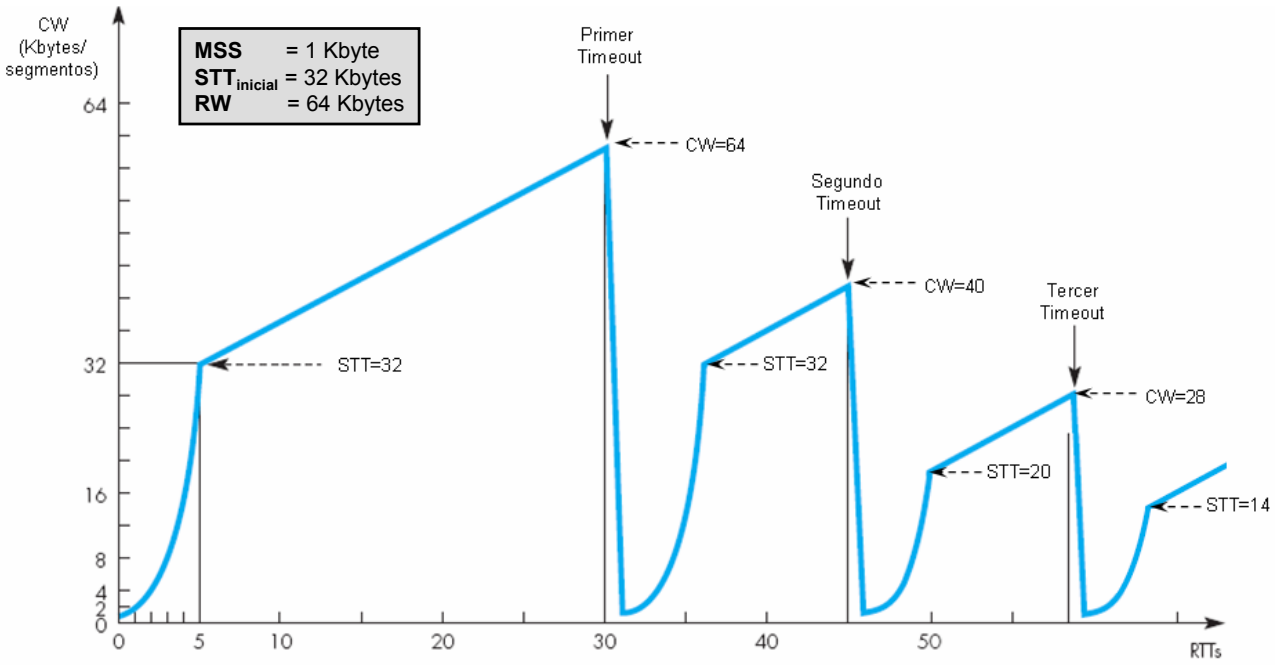


CW = 64

El control de congestión en TCP (8)

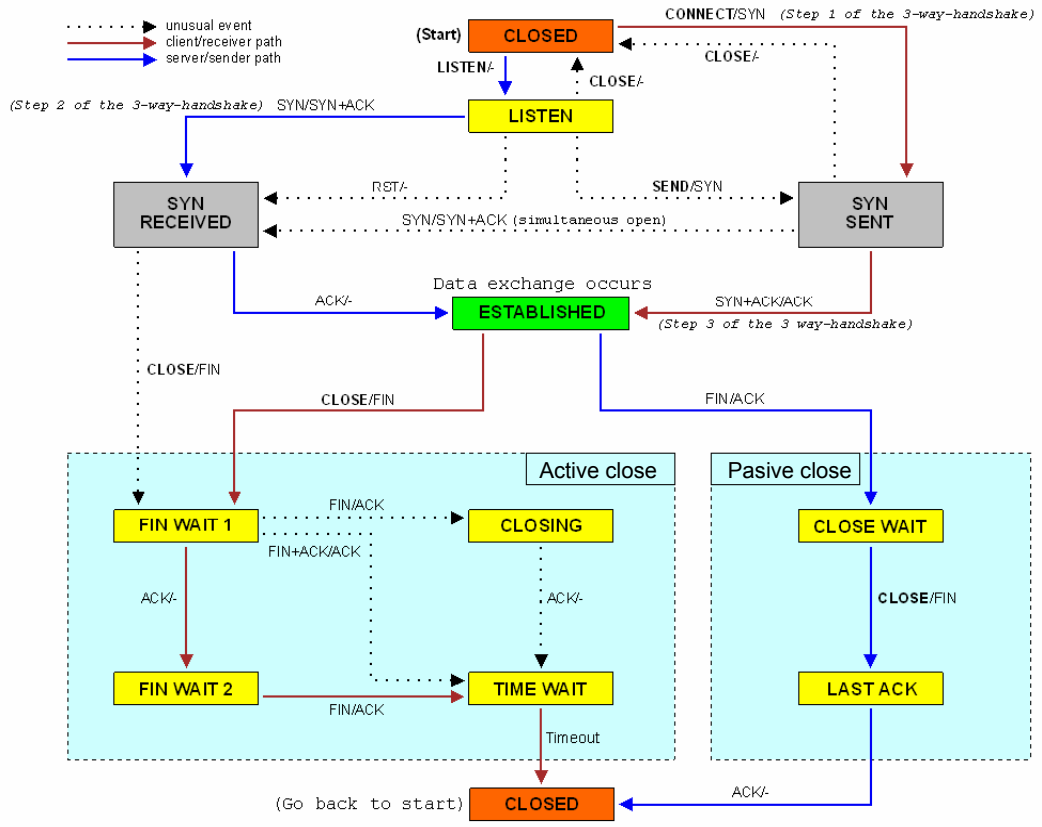
Funcionamiento en caso de congestión

- **Ejemplo 2.** Detección de congestión por timeout



Protocolo TCP

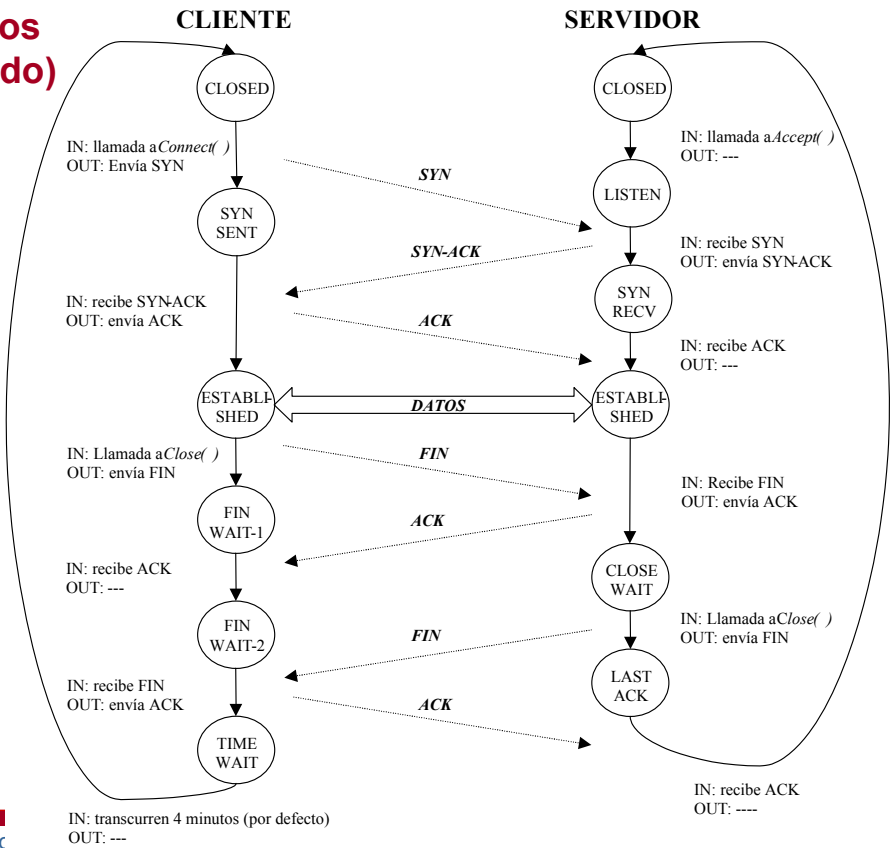
Diagrama de estados de TCP



Protocolo TCP

Diagrama de estados de TCP (simplificado)

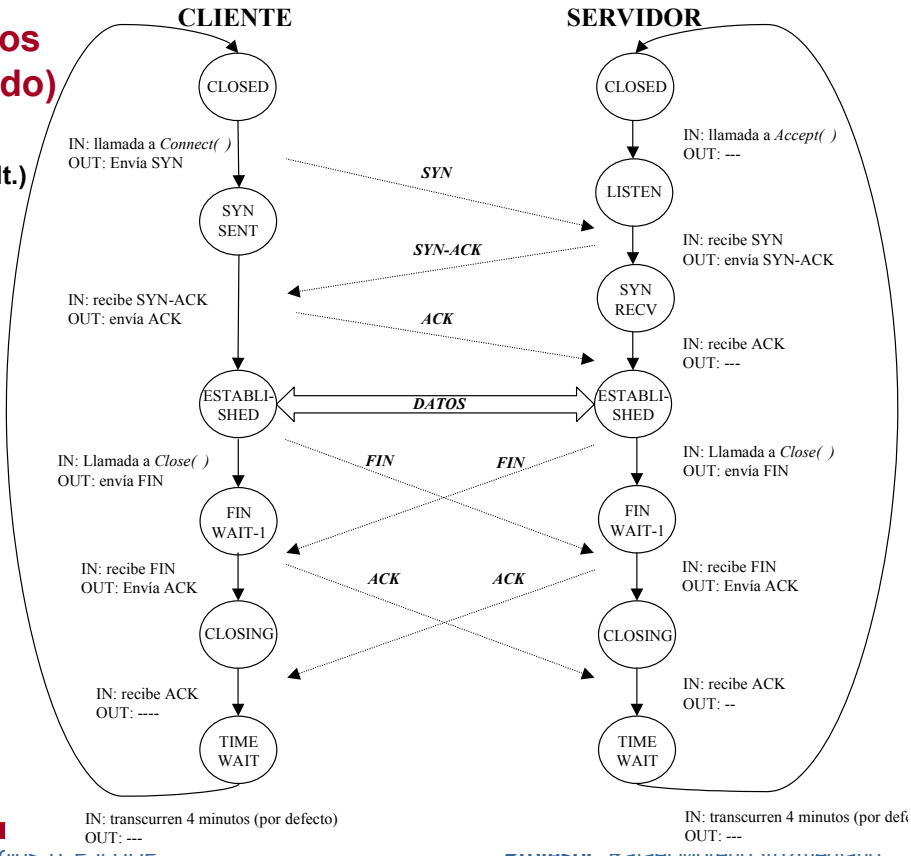
Cierre de 4 vías
(cierra 1º el cliente)



Protocolo TCP

- **Diagrama de estados de TCP (simplificado)**
  - **Cierre de 4 vías**

    (cierran ambos simult.)



Protocolo TCP

- **Diagrama de estados de TCP (simplificado)**
  - **Cierre de 3 vías**

    (cierra 1º el cliente)