Índice hora 3

Objetivos

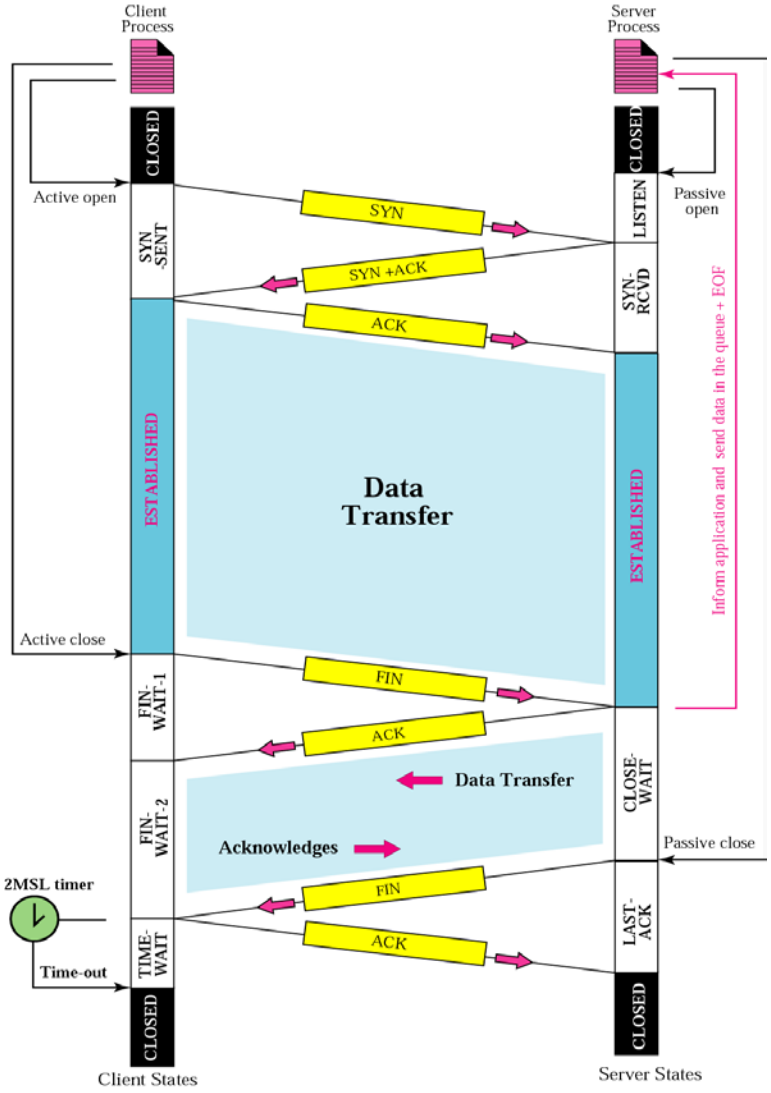
- Entender las fases y señalización en el establecimiento y cierre de conexiones TCP
- Comprender la necesidad de llevar estado y su implementación mediante la máquina de estados TCP
- Distinguir la problemática en transferencias interactivas y masivas
- Revisar los mecanismos de TCP para optimizar las comunicaciones interactivas

4.3 Conexiones TCP

Establecimiento

Transferencia

Finalización



4.3.1 Establecimiento

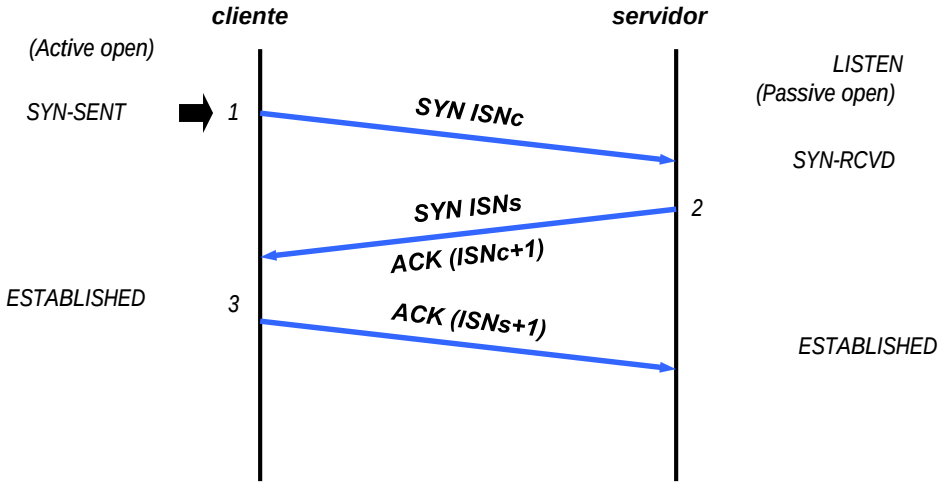
- Activo/pasivo
- Simultáneo

Establecimiento normal activo/pasivo

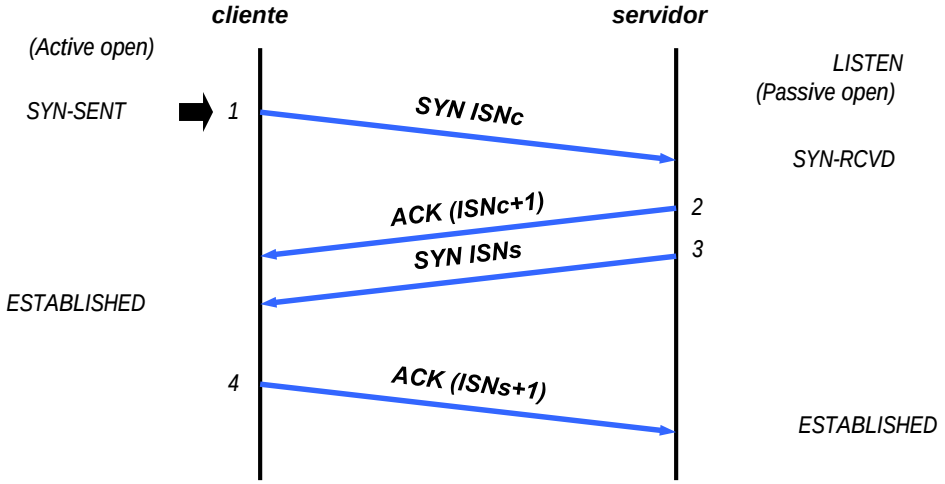
- Extremo pasivo ⇒ servidor ⇒ establecimiento pasivo.
- Extremo activo ⇒ cliente ⇒ establecimiento activo.
- Un extremo escucha peticiones de conexión (extremo pasivo) y el otro extremo mandará el primer SYN de solicitud de conexión (extremo activo) con el $ISN_{cliente}$ que le toque.
- El extremo pasivo contesta con otro SYN $ISN_{servidor}$ y a la vez ACK $ISN_{cliente}$.
- El extremo activo devuelve el ACK $ISN_{servidor}$ por el SYN recibido.

Establecimiento activo/pasivo

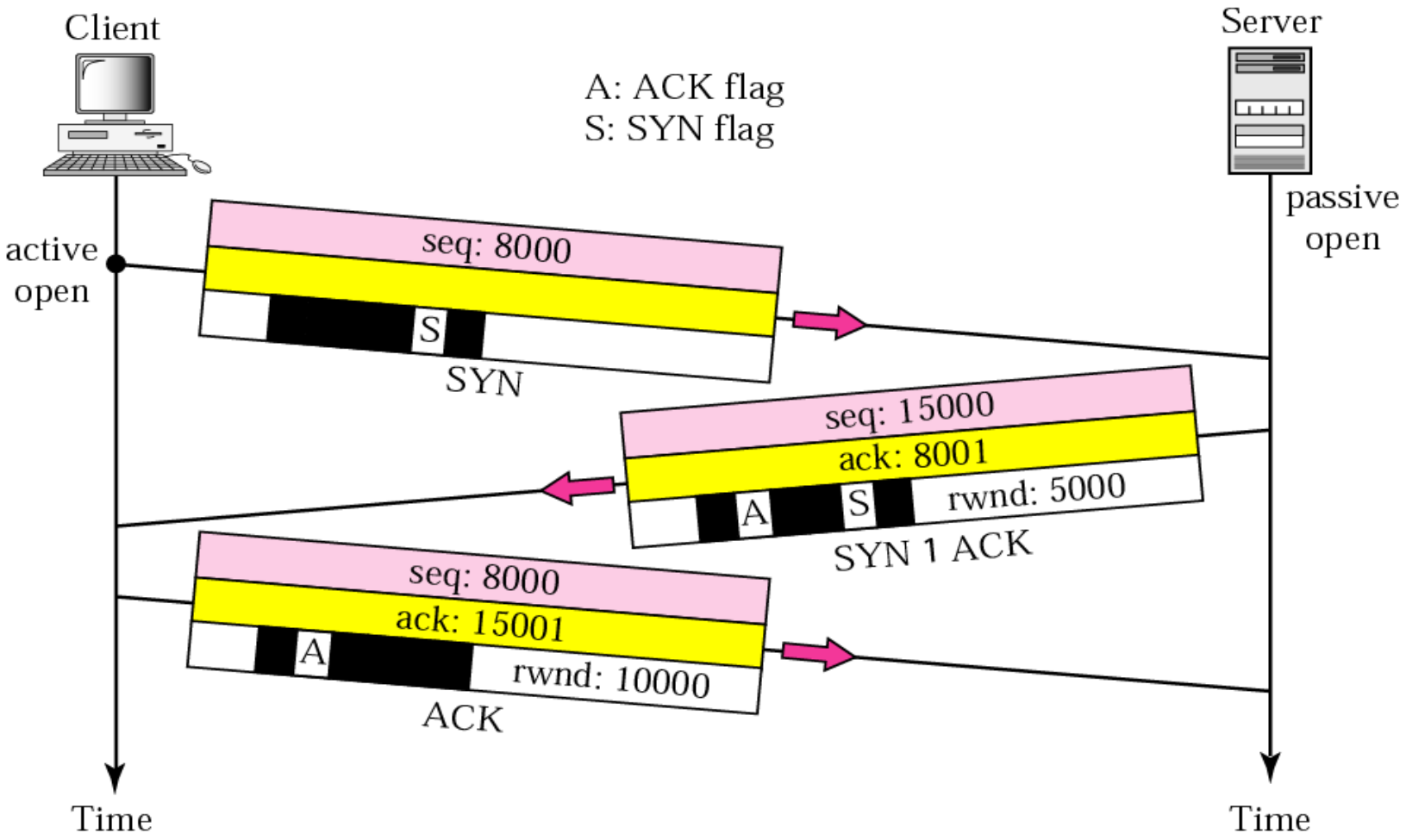
- Se denomina “3-way handshake” por los 3 pasos que implica.



- A veces aparece el ACK separado pero en todo caso antes del segundo SYN.

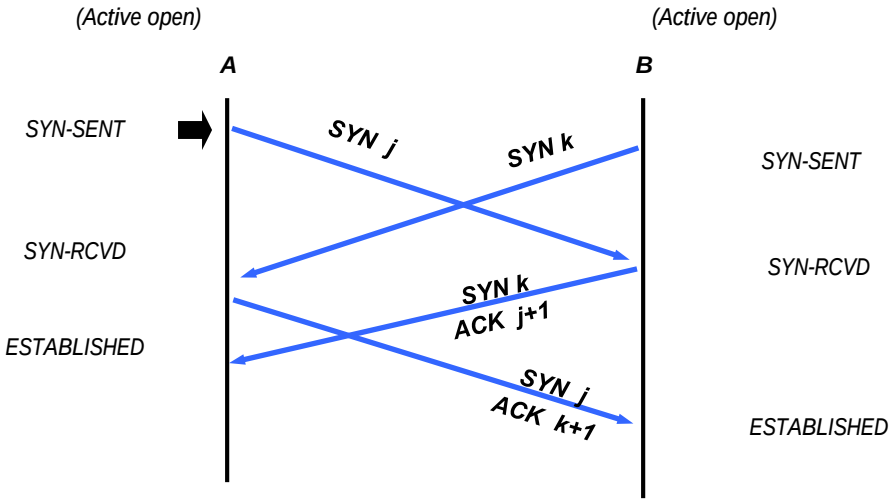


Establecimiento activo, ejemplo

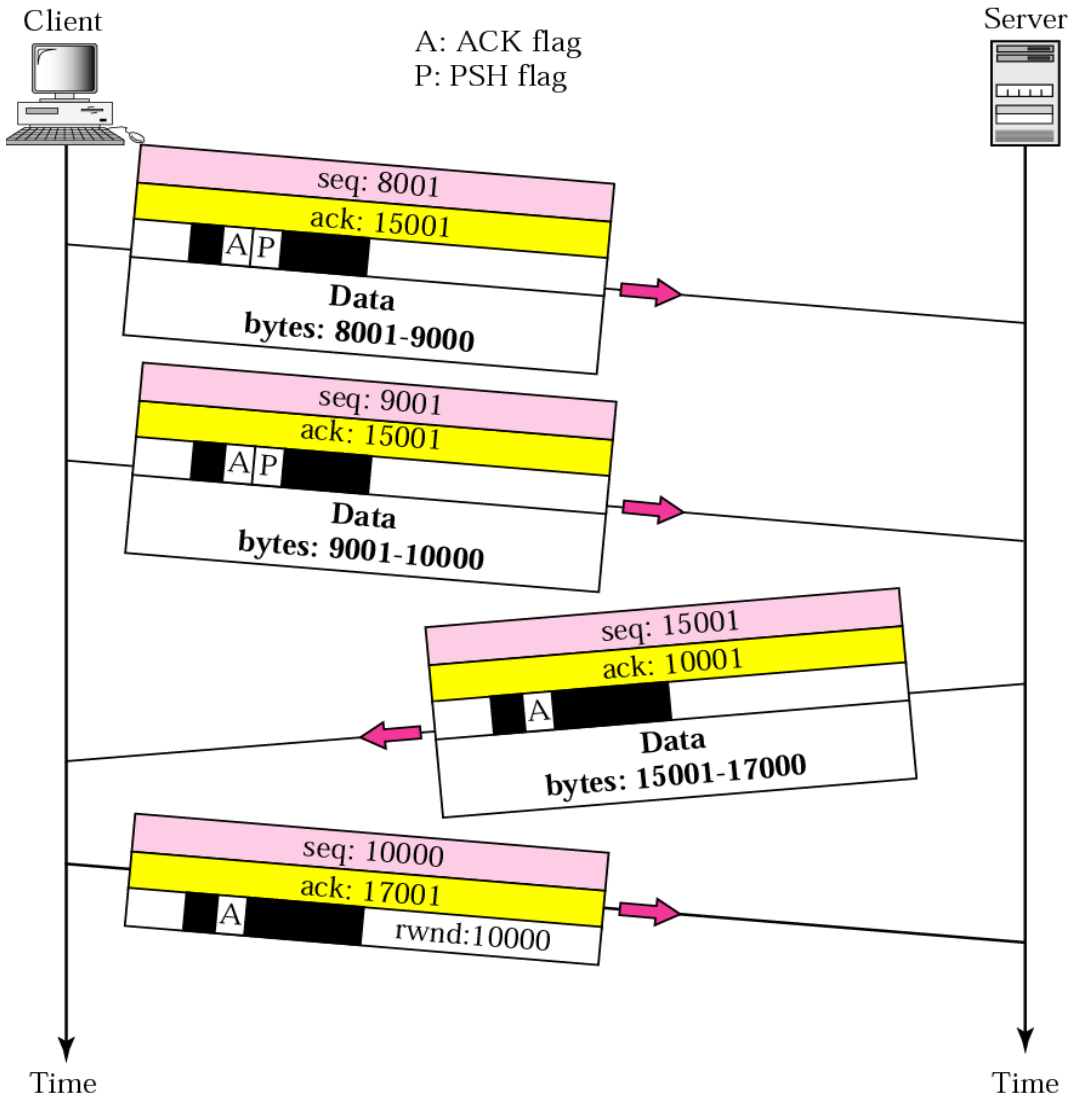


Establecimiento simultáneo

- Los dos extremos hacen un establecimiento activo, suponiendo que el puerto de cada uno es bien conocido por el otro.
- El resultado es una única conexión. Si no se repitiese el SYN j se crearían dos conexiones independientes.
  - Requiere intercambiar 4 segmentos en lugar de 3.
  - Los extremos hacen simultáneamente funciones de cliente y servidor.
  - Los dos extremos deben generar el SYN inicial aproximadamente a la vez.



4.3.2 Transferencia



Paquete sin datos ni SYN ni FIN: no significativo el nº de secuencia

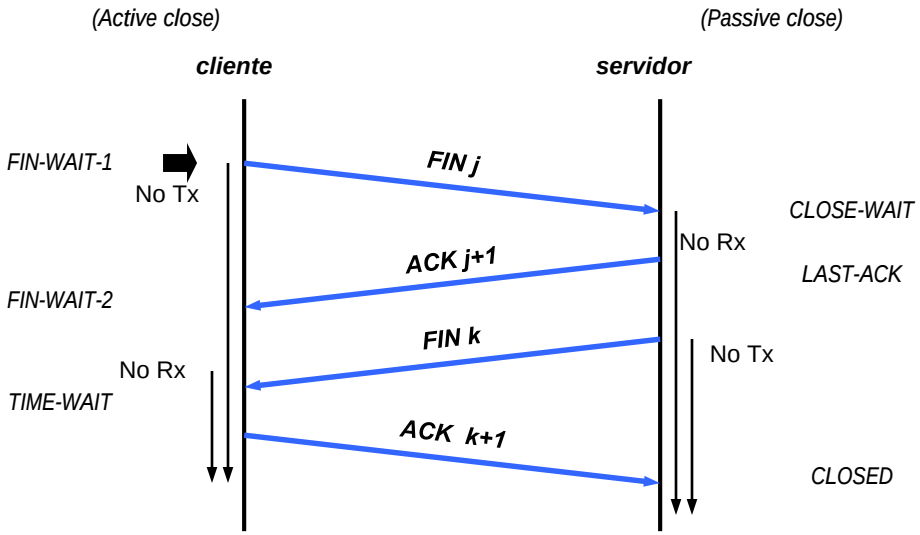
4.3.3 Finalización

- Finalización normal activo/pasivo.
- Finalización simultánea.
- Finalización a medias (half-close, half-open).
- Finalización abortiva.

Finalización normal activo/pasivo

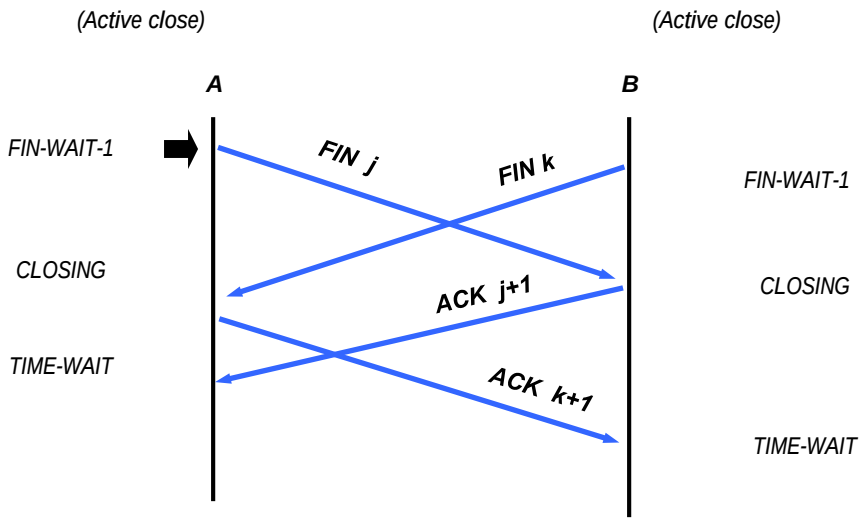
- Cierre de cada sentido independientemente.
- Se manda un FIN cuando no se tiene más datos que transmitir, pero se puede seguir recibiendo datos en ese extremo hasta que se reciba el FIN del otro sentido.
  - Extremo activo (normalmente el cliente): manda el primer FIN, cierre activo.
  - Extremo pasivo (normalmente el servidor): recibe el primer FIN, cierre pasivo.
- Cada FIN
  - Consume un nº de secuencia.
  - Necesita ser confirmado.

Finalización normal activo/pasivo



Finalización simultánea

- Ambos extremos realizan un cierre activo aproximadamente a la vez.



Finalización a medias (half-close,half-open)

- Half-close
  - Consiste en que sólo uno de los extremos cierre la conexión y el otro no lo haga con lo que el segundo puede seguir transmitiendo datos y el primero los puede seguir recibiendo.
- Half-open
  - Un extremo se cuelga/reinicia y el otro extremo no se entera porque no ha recibido FIN. El que reinicia si recibe segmentos de la conexión anterior manda un RST.

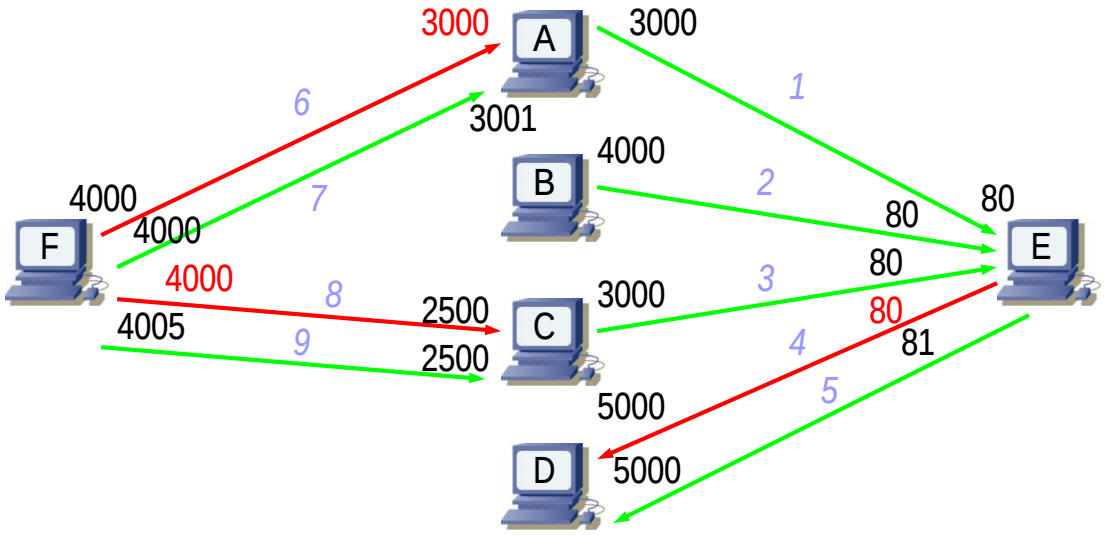
Finalización abrupta

- Alguno de los extremos manda un RST.
  - No hay confirmación.
  - Se tiran los datos que estuviesen encolados.

4.3.4 Conexiones permitidas

- Reglas para determinar si una conexión TCP es posible
  - Las conexiones son válidas si se diferencian del resto en al menos un campo de la siguiente tupla: (IPorigen, PuertoOrigen, IPdestino, PuertoDestino, ProtocoloTransporte)
  - Siempre debe existir un extremo cliente y otro servidor
  - Un puerto ocupado por una aplicaciones servidora
    - Puede recibir nuevas conexiones de clientes
    - No puede utilizarse como puerto cliente de conexión a otro servidor
  - Un puerto ocupado por una aplicación cliente
    - No puede utilizarse como puerto servidor
    - No puede utilizarse como puerto cliente de conexión a otro servidor
  - Los puertos UDP/TCP son independientes

Conexiones permitidas, ejemplos

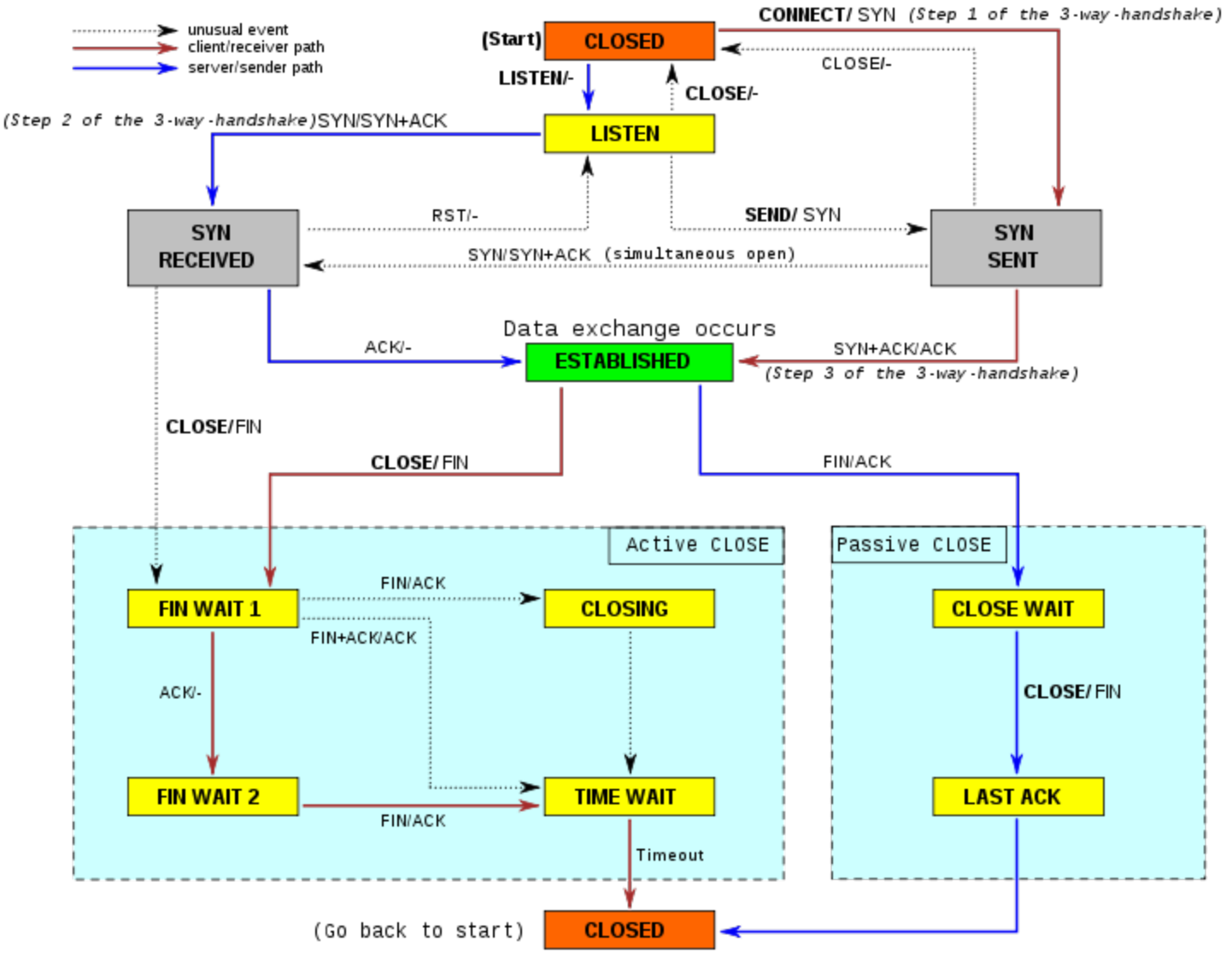


Puerto cliente *Nº orden* Puerto servidor

Conexión válida

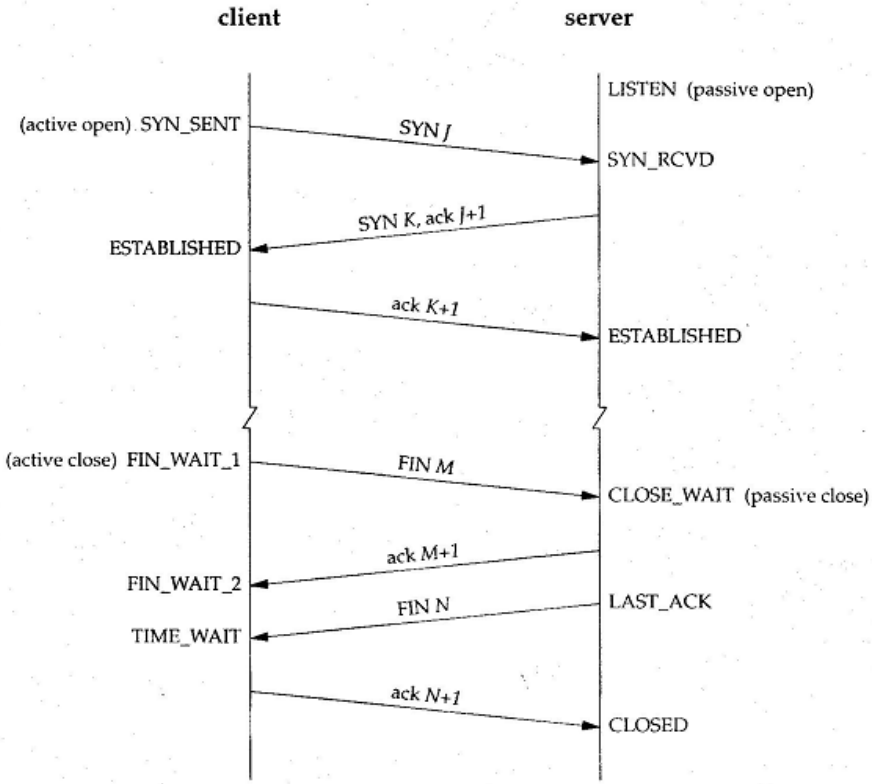
Conexión no válida

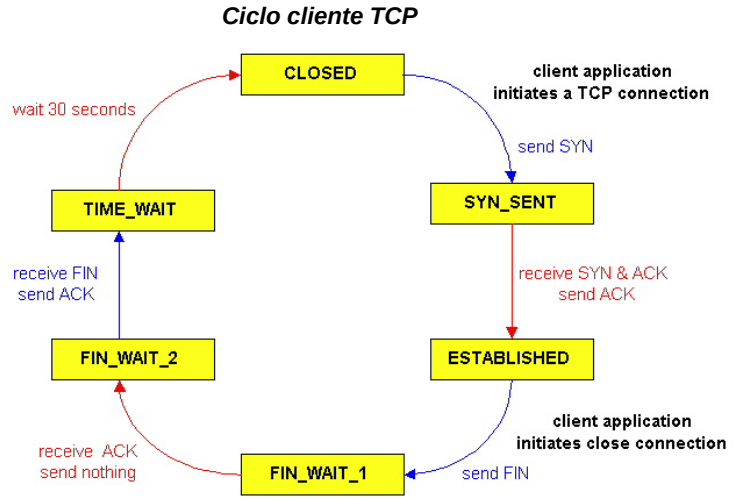
4.4 Diagrama de transición de estados de TCP



Máquina de estados finita

Diagrama de transición de estados





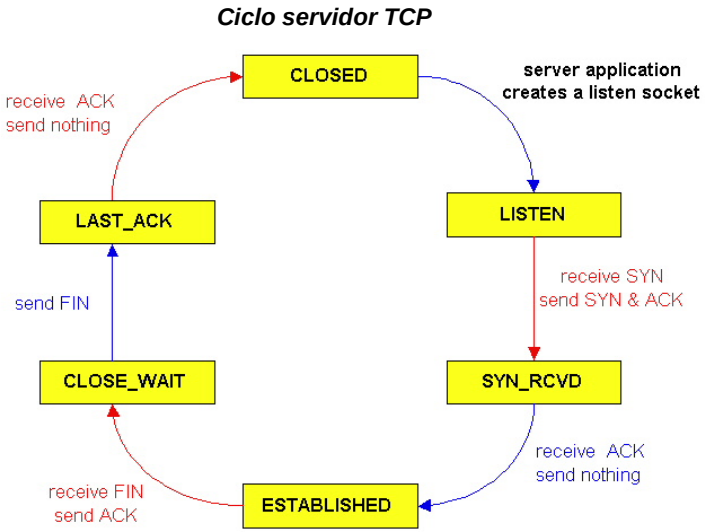


Diagrama de transición de estados

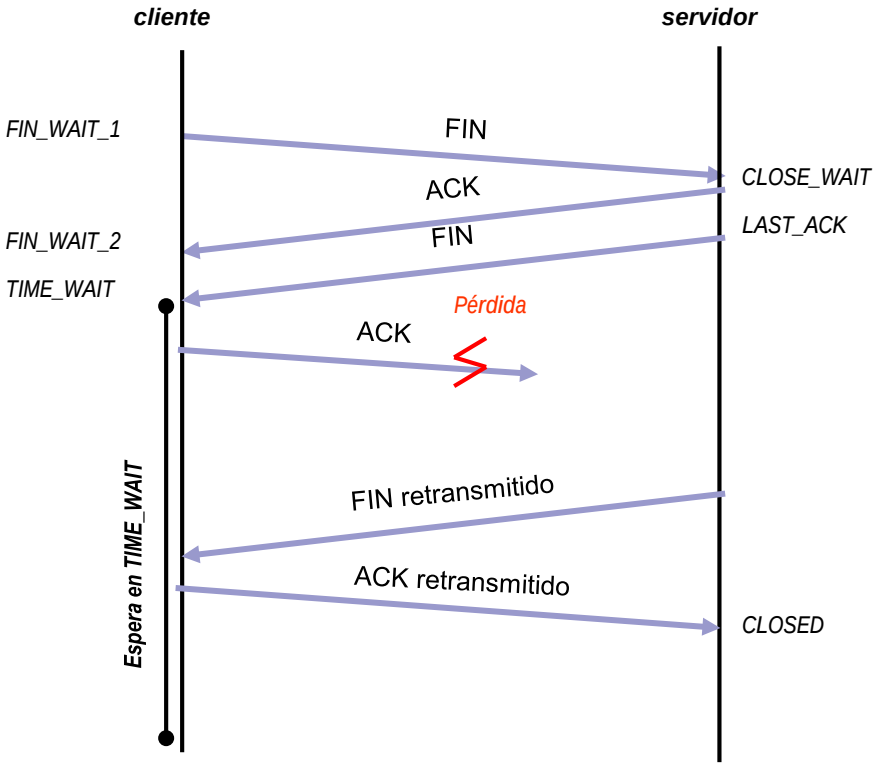
- El nombre de los estados coincide con el que podemos ver en sistemas UNIX-Windows al ejecutar el comando *netstat*.
- El estado ESTABLISHED es el único en el que cualquiera de los extremos puede transferir datos.
- La transición LISTEN → SYN_SENT es legal pero no soportada en APIs BSD.
- La transición SYN_RCVD → LISTEN es sólo válida si se entró al estado SYN_RCVD desde el estado LISTEN.
- FIN_WAIT_2: mandado un FIN y recibido el ACK, la conexión está cerrada sólo en un sentido (half-closed). Puede permanecer de forma indefinida en este estado recibiendo datos en el otro sentido de la conexión hasta recibir un FIN en ese sentido.
- Intento de conexión a un puerto sin servidor escuchando devuelve un RST.

Diagrama de transición de estados

- TIME_WAIT: permanece en ese estado cuando un extremo hace el cierre activo mandando el ACK final, para en caso de necesidad reenviar este ACK (si recibe un nuevo FIN del otro extremo).

Tiempo espera TIME_WAIT = 2MSL

- MSL (Maximum Segment Lifetime), máximo tiempo que un segmento TCP puede existir en la red sin ser descartado.
- TCP → IP → TTL pone el límite.
- RC793 fija MSL=2min, pero implementaciones usan 30s, 1min o 2 min.
- Durante este tiempo no se puede abrir otra conexión sobre ese puerto, en escucha pasiva (Ej: servidores mal finalizados) o apertura activa.
- Datos recibidos en este intervalo se tiran.



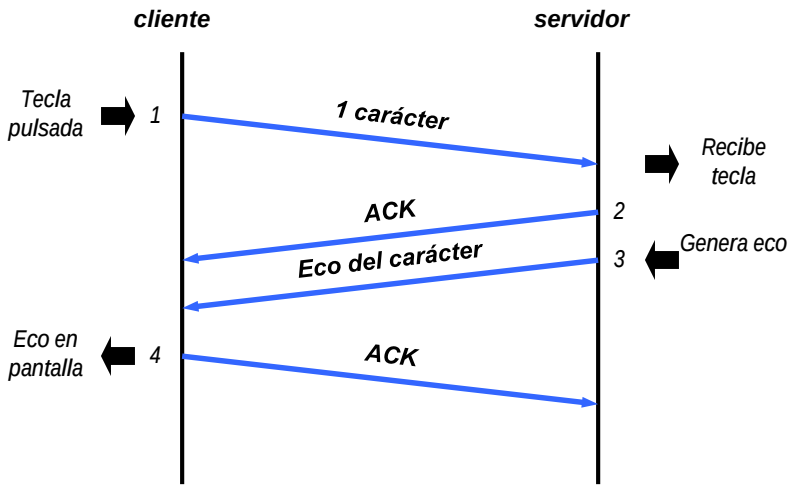
4.5 Transferencia interactiva

- Según los requisitos de las aplicaciones podemos distinguir entre dos grandes tipos de tráfico:
  - Transferencia Interactiva
    - Requisitos temporales en el envío de datos.
    - Segmentos pequeños.
    - Aproximadamente suponen menos del 10% del tráfico total de Internet.
    - Ejemplo: Telnet, rlogin, IRC, SSH, etc.
    - Problema: compromiso entre eficiencia del protocolo e interactividad (sobrecarga por cabeceras de protocolos si se envían paquetes pequeños).
  - Transferencia masiva
    - Importancia de la velocidad frente al retardo.
    - Segmentos grandes.
    - Aproximadamente suponen más del 90% del tráfico total de Internet.
    - Ejemplo: FTP, Web, Email, P2P, etc.
    - Problema: el transmisor puede mandar más rápido que lo que es capaz de soportar la red y el receptor.

Transferencia interactiva

Ejemplo de aplicación interactiva: TELNET

- Por cada tecla pulsada por el usuario en principio se envía a la máquina remota un segmento TCP con un sólo byte de datos que es el código ASCII (carácter) de la tecla pulsada.
- Al llegar el carácter al servidor se confirma.
- El eco en pantalla lo produce un paquete de vuelta con el mismo carácter del servidor al cliente.



Para optimizar el funcionamiento:

- Juntar segmentos 2 y 3 en el mismo. Se le llama *piggyback* (incluir datos y ACK en el mismo segmento).
- Evitar enviar paquetes pequeños, (no uno por cada tecla).

1 byte datos + 20 bytes cab. TCP + 20 bytes cab. IP → 1/41: mala eficiencia

Transferencia interactiva

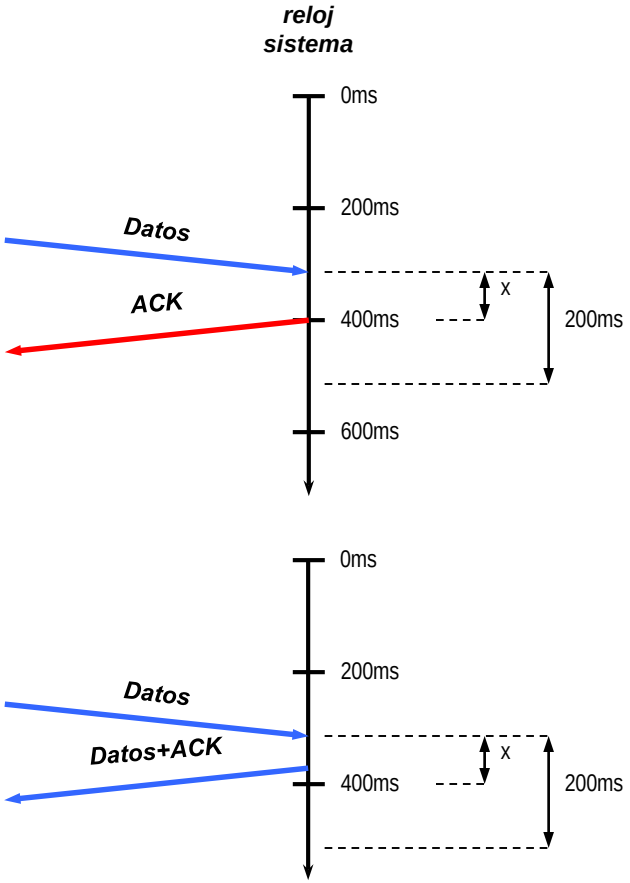
- El tráfico de datos interactivo impone:
  - Muchas confirmaciones asociadas a segmentos de datos que interesará reducir:
    - Delayed ACK.
  - Elevado número de segmentos de pequeño tamaño, que muchas veces interesará minimizar:
    - Algoritmo de Nagle.

4.5.1 Delayed ACK

- En TCP se envían típicamente ACKs cada 1 o 2 segmentos de datos recibidos en el receptor.
- Recomendación TCP: si han llegado dos segmentos de tamaño MSS se confirman inmediatamente.
- Mecanismo de Confirmación Retrasada o *Delayed ACK* trata de disminuir el número de ACKs generándolos sólo:
  - Tras un timeout ≈ 200ms (típicamente) sin tener otros datos que enviar.
  - Si hay datos que enviar de vuelta antes de agotar el timeout, se mete el ACK con los datos (*piggybacking*).
- El timeout se contabiliza no desde el instante de recepción del primer dato sino a intervalos múltiplos de un temporizador general para la máquina.

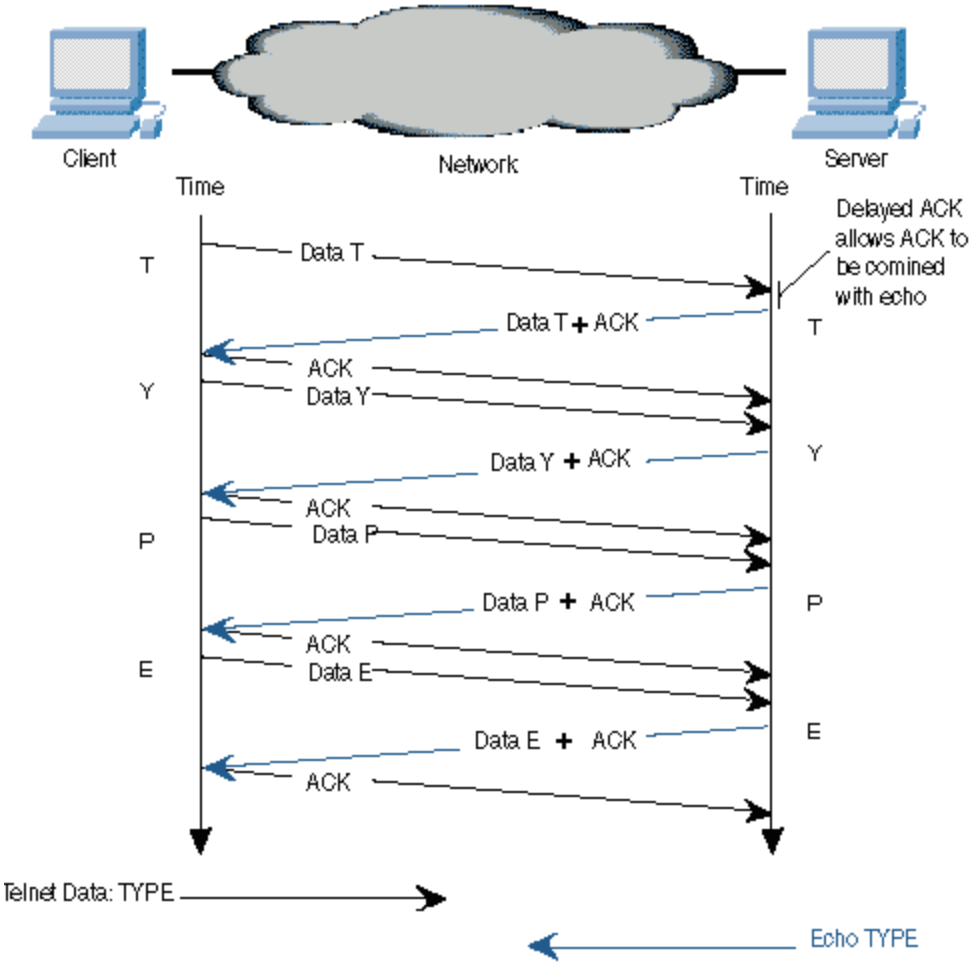
Delayed ACK

- Por tanto, cada timeout del sistema se comprueba si hay datos sin confirmar y en tal caso se confirman.



- En la figura el ACK se manda un tiempo x después, con x < 200ms.
- En realidad, el tiempo de envío del ACK será un valor entre 0 y 200ms, dependiendo como toque.
- Este valor de 200ms se elige para que no sea problemático, se interprete como una pérdida y el origen tenga que retransmitir.

Delayed ACK


Client
Network
Server
Time
Time
Delayed ACK allows ACK to be comined with echo
T
Data T
Data T + ACK
T
ACK
Y
Data Y
Data Y + ACK
Y
ACK
P
Data P
Data P + ACK
P
ACK
E
Data E
Data E + ACK
E
ACK
Telnet Data: TYPE
Echo TYPE

4.5.2 Algoritmo de Nagle

- El envío de paquetes pequeños puede ser problemático para redes WAN aunque no lo sea para LANs.
  - Gran ineficiencia por carga de cabecera de protocolos en cada paquete. Ejemplo telnet: 20(IP)+20(TCP)+1(carácter)=41 bytes
  - Muchos paquetes pequeños exigirán muchas confirmaciones (a no ser que se active el delayed ACK).
- Algoritmo de Nagle (RFC896):
  - En una conexión TCP con datos enviados que no han sido confirmados todavía no se pueden enviar segmentos pequeños hasta que los datos pendientes sean confirmados.
    1. TCP manda el primer segmento de datos incluso si es de 1 byte.
    2. TCP acumula datos de la aplicación y no envía un segmento hasta que:
       - Recibe un ACK del segmento anterior. Se vuelve al paso 1.
       - Ha acumulado datos para llenar un MSS. Se envía y se repite este paso 2.
  - Permite únicamente enviar un único segmento pequeño cada vez que se recibe confirmación de datos anteriores.

Algoritmo de Nagle

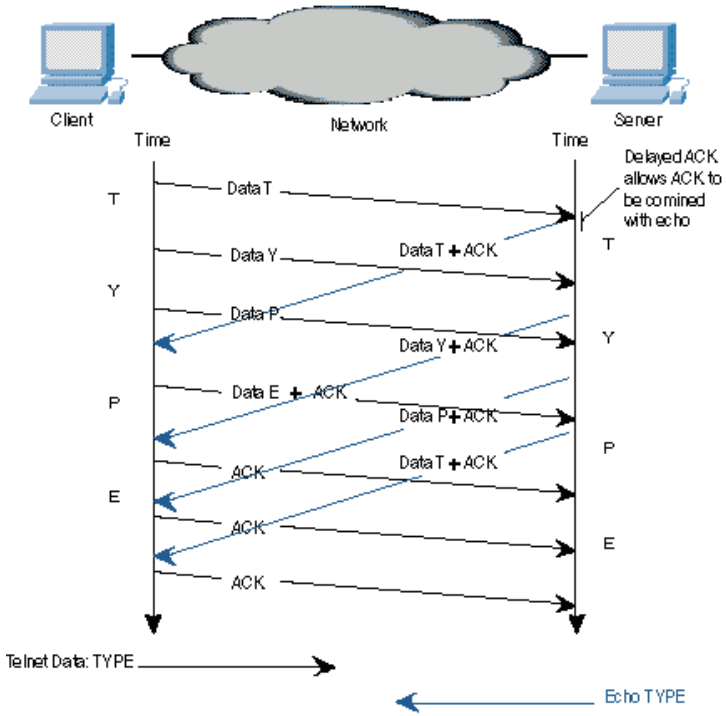
- Si el receptor acumula bytes sin confirmar, la ventana de recepción anunciada va a ir disminuyendo.
- Ventajas:
  - Simple, pero tiene en cuenta la velocidad a la que la aplicación crea datos y la velocidad a la que la red transporta los datos. Es *autoajustable*.
    - En enlaces WAN lentos → ACKs más lentos → se enviarán menos segmentos pequeños.
    - Si la aplicación crea datos más rápido que la red puede soportar, los segmentos serán mayores.
  - Los segmentos pequeños que se quieran enviar se agrupan en un sólo segmento mayor hasta cuando se reciba confirmaciones.
    - Si se reciben confirmaciones rápidamente, se podrán enviar segmentos pequeños.
    - Si no se reciben confirmaciones rápidamente, sólo se podrán enviar segmentos mayores.

Algoritmo de Nagle

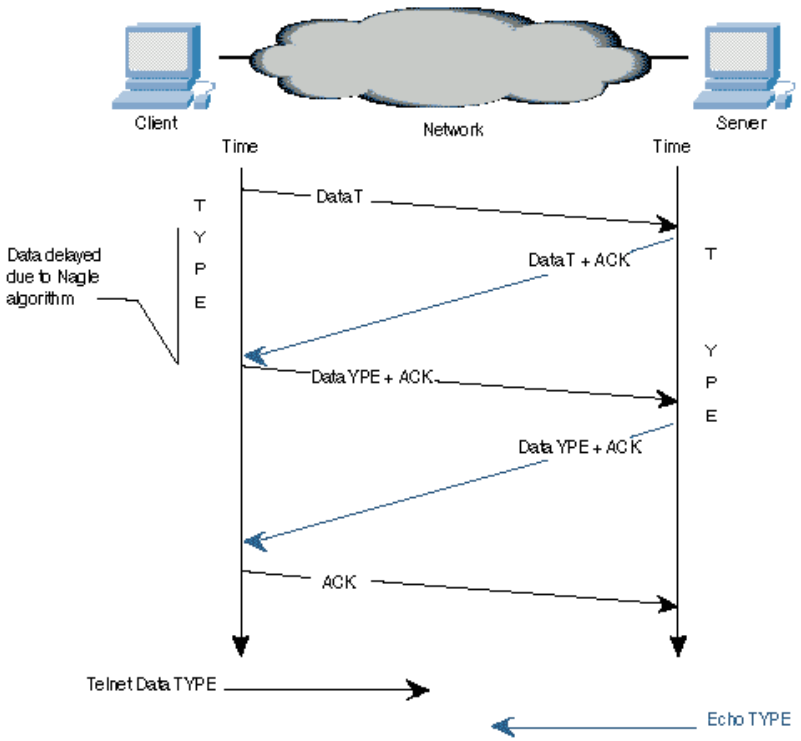
- Ejemplo:
  - RTT↓, Ethernet+Telnet: envío carácter+ACK+eco ≈ 16ms. Se necesitaría escribir a 1000ms/16ms ≈ 60 caracteres/s para que pudiese entrar en funcionamiento Nagle.
  - RTT↑: el algoritmo de Nagle entrará en funcionamiento, permitiendo reducir el número de segmentos intercambiados.
- Este algoritmo está relacionado con el *Silly Window Syndrome* que consiste en la existencia de segmentos pequeños en la red. Nagle servirá para evitarlos desde el punto de vista del emisor.
- Deshabilitar algoritmo de Nagle: podrá interesar cuando se tenga un tráfico muy interactivo que ha de mandarse lo más rápidamente posible y que genera poca información
  - Ejemplo: protocolo X-Window usa pequeños segmentos TCP para reflejar el movimiento del ratón con suavidad. En ningún caso se deben retrasar esos segmentos.

Algoritmo de Nagle

Sin Nagle



Con Nagle



Resumen

- TCP se implementa mediante una máquina de estados en cada extremo de la comunicación
- TCP al ser un protocolo orientado a conexión tendrá las fases de:
  - Establecimiento
  - Transferencia de datos
  - Finalización de las conexión
- El tráfico de datos interactivo impone:
  - Muchas confirmaciones asociadas a segmentos de datos, que se podrán reducir con el Delayed ACK.
  - Segmentos de pequeño tamaño, que muchas veces interesará minimizar para lo cual se aplica el algoritmo de Nagle.

Referencias

- [Forouzan]
  - Capítulo 15, “Transmission Control Protocol (TCP)”, secciones 15.4-15.5
- [Stevens]
  - Capítulo 18, “TCP: Connection Establishment and Termination”
  - Capítulo 19, “TCP interactive Data Flow”